AUTOMATED REFACTORING TOOL

By

Alain Coutu, CD1
B. Sc., Collège Militaire Royal de St-Jean, 1985

PROJECT REPORT

Submitted in partial fulfillment of the requirements
For the Master's degree in Science (Software Engineering)
In the Computer Science Department of the School of Arts and Science
Florida State University, 2003

Panama City, Florida

Major Professor: Dr. Sara Stoecklin
Committee Members: Dr. Chris Lacher
Dr. Dan Schwartz

TABLE OF CONTENT

TABLE OF CONTENT ..... **I**

LIST OF FIGURES ..... **III**

ABSTRACT ..... **IV**

1.0 INTRODUCTION **1**

2.0 REFACTORING BACKGROUND **2**

2.1 Definition of Refactoring 2
2.2 Purpose of Refactoring 2
2.3 Advantages and Disadvantages of Refactoring 3

3.0 PROJECT OVERVIEW **4**

3.1 Project Description 4
3.2 Project Goals 4
3.3 Project Background References 4
3.4 Project Development Methodology 5
3.5 Project Development Standards 5

4.0 REQUIREMENTS **6**

4.1 High Level Requirements 6
4.2 Detailed Requirements 6

5.0 DESIGN **8**

5.1 Overall Design 8
5.2 Patterns Used and Their Benefits 15
5.2.1 AbstractFactory pattern 15
5.2.2 Iterator pattern 16
5.2.3 Model-View Controller pattern 17
5.2.4 Strategy pattern 18

6.0 SOFTWARE DESCRIPTION **21**

6.1 Graphical User Interface 21
6.2 Parsing 21
6.3 Refactoring 22
6.4 Compiling 23

7.0 SOFTWARE VERIFICATION AND VALIDATION **24**

7.1 Testing Philosophy 24
7.2 Tests Performed 24

8.0 FUTURE RESEARCH **25**

8.1 Adaptability of New Refactoring Transformations 25
8.2 Pattern Recognition 25
8.3 Reflective Architecture 25

APPENDIX A - SOFTWARE REQUIREMENT SPECIFICATIONS

APPENDIX B - - SOFTWARE DESIGN SPECIFICATIONS

APPENDIX C - SOFTWARE TEST PLAN

APPENDIX D SOFTWARE TEST PROCEDURES

APPENDIX E - SOFTWARE TEST CASES

APPENDIX F - CODE LISTING

APPENDIX G - CODING CONVENTIONS

APPENDIX H - LIST OF REFERENCES

LIST OF FIGURES

Figure 5.1.1 - ART GUI 9
Figure 5.1.2 - ART GUI analysis level class diagram 10
Figure 5.1.3 - Final ART GUI class diagram 11
Figure 5.1.4 - Analysis level Deparser Component class diagram 12
Figure 5.1.5 - Design level Deparser class diagram 13
Figure 5.1.6 - Final Deparser class diagram 14
Figure 5.1.7 - Refactor class diagram 15
Figure 5.2.1.1 - **AbstractFactory pattern** implementation class diagram 16
Figure 5.2.2.1 - **Iterator pattern** implementation class diagram 17
Figure 5.2.3.1 - **MVC pattern** implementation class diagram 18
Figure 5.2.4.1 - **Strategy pattern** implementation class diagram for reformatting tokens 19
Figure 5.2.4.2 - **Strategy pattern** implementation class diagram for refactoring transformations 20

ABSTRACT

Performing refactoring transformations manually is a long and tedious process depending on the size of the code. Software engineers must preserve the behavior of the code when refactoring. Refactored code becomes flexible, reusable, and more maintainable. Tools that support refactoring transformations automatically are becoming available but many of them still require human involvement. Research on automated refactoring tools presents the use of graph representation, program invariants and formal specifications for the development of automated refactoring tools. This project investigates the implementation of a framework using design patterns for the incremental and adaptive development of an automated refactoring tool. It allows the investigation and implementation of each refactoring transformation independently of others. Refactoring transformations are implemented by allowing the framework design to change permitting better reusability and flexibility of the code developed. The tool is expanded to include refactoring transformations used for proving the framework as well as implementing other object-oriented concepts such as reflective architecture and pattern recognition.

1.0 INTRODUCTION

Refactoring transformations is a software engineering concept that permits the modification of code while preserving its behavior. The code modifications are such that the newly modified code is more maintainable, more reusable, and easier to read. To perform these code modifications, software engineers must perform refactoring transformations either manually or using an automated tool. However, automated tools surfaced only recently (RefactorIt, JRefactory, jFactor, Refactoring Browser for Smalltalk to name a few) as a limited amount of research is performed to develop them and make them available for any object-oriented language.

To perform a refactoring transformation, a software engineer must traverse his/her code and apply modification rules. It is a tedious process and may lead to errors. Currently, some of the refactoring tools available require the selection of code expressions on which refactoring transformations are applied. It is called automated because the tool takes care of generating the transformed code expressions while preserving the behavior but does not select the code expression to be refactored.

This project consists of developing a framework using design patterns for the incremental and adaptive development of refactoring transformations, a graphical user interface (GUI) and a refactoring transformer. A parser, also a component of the tool, was generated using JavaCC and integrated with the tool. For proving the framework developed, the tool implements two refactoring transformations: Replace Magic Number with Symbolic Constant and Encapsulate Fields.

The projects applied known software methodology standards for the development of software: Software Requirement Specification, Software Design Specification, Software Test Plan, Software Test Procedures, and Software Test Cases. In addition, the developed framework implements various design patterns. In addition, during the design and writing of the code, refactoring transformations were applied. For ease of reading, through out this document, classes name are italicized, pattern names are in bold and refactoring transformation names are underlined.

Section 2 provides background information on refactoring. Section 3 provides an overview of the project. Section 4 defines the high-level and detailed requirements for the tool. Section 5 discusses the overall design of the tool and provides a discussion on the design patterns implemented. Section 6 details the inner workings of the tool. Section 7 provides the testing philosophy used and a brief discussion on the various tests performed. Section 8 provides a discussion on future research resulting from this project.

2.0 REFACTORING BACKGROUND

2.1 Definition of Refactoring

Members of the software engineering object-oriented community differ on the definition of refactoring. For this reason, it is not practical to search for one definition that is global, simple and accepted by all software engineers. However, Ralph Johnson, Kent Beck, Alistair Cockburn and Martin Fowler provide some common definitions in [Wiki1] and [Fowler00]. Ralph Johnson, [Wiki1], defines refactoring as being a change in a program that affects only the organization of the program but not its behavior. He further states that it is referred to as "*restructuring or behavior preserving transformation*". Kent Beck, [Wiki1], defines refactoring as "*not just aimlessly moving code between the infinite permutations of a program that behave identically. It is there to add qualities that were previously non-existent*". Alistair Cockburn, [Wiki1], compares it to performing algebraic factorization using code instead of mathematical formulas. Martin Fowler defines the term refactoring in his book, [Fowler00], as "*the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure*".

Opdyke, [Opdyke96], defines refactoring as "*a means for achieving evolution and reuse in the midst of legacy constraints*". Roberts, [Roberts99], defines refactoring as "*a program transformation that has peculiar preconditions that must be satisfied before the transformation can be applied, and a set of post conditions that will hold afterwards*". Roberts defines refactoring in this manner because it is not always feasible to prove the behavior of a program is preserved after applying refactoring transformations.

Based on all these definitions, refactoring is defined in this research as the process used by software engineers to transform existing code into a more reusable, portable, flexible format while keeping the design integrity. In this context, design integrity means the design of the initial requirements is kept intact.

2.2 Purpose of Refactoring

Refactoring is not a new idea. As discussed in [Roberts] and [Roberts97] by Roberts et al., Smalltalk programmers have been performing refactoring manually since the inception of the language. Opdyke, [Opdyke99], mentions contrary to popular belief, that Smalltalk programmers started refactoring. Structured programming development methodologies (Waterfall, Spiral, etc.) were adapted over the years to handle changes. However, because object-oriented programming is new, beginning of the 80's, current development methodologies are improving constantly to meet the needs of the object-oriented community. Opdyke also maintains that refactoring was introduced as a technique to support changes of the object-oriented development methodologies.

If refactoring is a technique to support change, the potential exists to extend it to support change over the life-cycle development and maintenance of a program. Opdyke, [Opdyke96], makes an important point regarding the development of a reusable solution from a directive to make a change to an existing piece of software. Some issues regarding "legacy constraints" must

be addressed before implementing the refactoring approach. These constraints deal with short and long term benefits of the new solution, the understanding of the refactoring approach applied to the problem, the possible introduction of subsequent changes that are currently unknown and the misunderstanding that refactoring is an overhead activity.

Opdyke, [Opdyke99], goes further in defining refactoring as a middle ground between performing a complete re-write of a program and the cut and paste approach to extend a program capabilities. In addition, Opdyke writes that the refactoring approach helps leverage past investment in software development, reduce any code duplication and improve the efficiency of a program.

In summary, the purpose of performing refactoring on a program, as discussed above, is to perform transformations on a program to increase its maintainability, flexibility and understandability for software engineers.

2.3 Advantages and Disadvantages of Refactoring

Tokuda and Batory, [Tokuda01], discussed some advantages and disadvantages of performing refactoring following experiments with the C++ language. The main advantages of refactoring discussed are that it:

a) enables the automation of changes in design,
b) reduces the amount of effort for testing,
c) generates designs that are simpler,
d) assists in the verification of conflicts between the code and the desired design, and
e) allows designers to experiment with new design ideas.

However, they also mention that refactoring applied to large-scale applications has some disadvantages. They are:

a) refactoring transformations do not deal with preprocessor directives if using C++;
b) conservative enabling conditions predicate refactoring transformations; and
c) it is sometime difficult to verify automatically conservative enabling conditions.

According to Tokuda and Batory, the Java language has the same advantages as C++. However, they contend that Java has less disadvantages than C++ because it does not have any preprocessor, the source files are put together easily, it does not have free floating procedures and the methods are co-located with their associated classes.

3.0 PROJECT OVERVIEW

3.1 Project Description

The purpose of this project is to develop an automated tool that applies specifics refactoring transformations to code that may or may not require refactoring transformations. The tool is implemented using the Java language. The intent of the tool is to guide developers in performing refactoring. As mentioned in [Roberts97] by Roberts et al., in [Tokuda] and [Tokuda01] by Tokuda and Batory, and [Riggs] by Riggs and Stoecklin, even with the automation of refactoring transformations, there exist situations where a developer provides information (variable names, method names, etc) or accepts the suggested refactoring transformations.

3.2 Project Goals

The project consists of implementing the following four (4) major components:

a) develop a framework using design patterns for the incremental and adaptive development of refactoring transformations
b) a parser,
c) a refactoring transformer, and,
d) a graphical user interface (GUI).

The framework is responsible of providing the basic classes and methods that are required to permit applying automatically refactoring transformations. The parser is responsible for reading Java modules and applying parsing rules to generate tokens representing the code read. The refactoring transformer is responsible for modifying the set of tokens provided by the parser. To validate the framework developed, the following refactoring transformations as defined in [Fowler00] were implemented:

a) Encapsulate Fields, and
b) Replace Magic Number with Symbolic Constant.

The GUI is responsible for displaying to the user the initial code, the newly transformed code, and the refactoring transformation(s) applied.

3.3 Project Background References

For some time, research has been conducted for the development of automated refactoring tools. In [Kataoka], Kataoka et al. discuss performing refactoring transformations using program invariants. In [Riggs], Riggs and Stoecklin discuss the ability to formally specify, establish the rules of automation, and define mechanisms for automation for the performance of automated refactoring transformations. In [Mens], Mens et al. present the concept of using graph representation for preserving refactoring transformations. In [Roberts97], Roberts et al. discuss the refactoring tool developed for SmallTalk. The tool makes use of a parser and a browser and performs dynamic code analysis to perform the refactoring transformations. RefactorIt is an

automated tool, developed by Agris software, that analyzes code to determine where expression s are used and then apply a refactoring specified by the user. The user must supply information such as variable name or method names. The Refactoring browser for SmallTalk, developed by Roberts et al, parses the SmallTalk code, and then using the browser, the user selects the code he wants to refactor with the specific refactoring transformations. JRefactory is another automated refactoring tool, developed under the SourceForge open source project, where the user selects expressions or code portions and then specifies which refactoring to execute. The user also supplies variable names and method names if necessary. jFactor, developed by Instantiations Inc., is also an automated refactoring tool. It performs the refactoring transformations in the same manner as JRefactory.

The tool developed in this project performs refactoring transformations by parsing the code and then applying specific refactoring transformations, as chosen by the user. It does not require the user to supply variable or method names as the application generates them.

3.4 Project Development Methodology

The development methodology used for developing software requirement and design specifications is the Waterfall method. For the remaining part of the development, code and test, it is done incrementally with refactoring transformations applied to the design. The rationale for this selection is as follows:

a) the requirements for the tool are clearly defined and stable;
b) the design of the application is clearly determined by the requirements; and,
c) the transformation of the design to code is better suited incrementally as each refactoring transformation can be implemented individually.

The development consists of developing System Requirements Specifications (SRS), Software Design Specifications (SDS), writing the code and performing various tests to validate the refactoring transformations.

3.5 Project Development Standards

When producing the tool source code and applying refactoring transformations, a modified version of the Sun Microsystems Java Coding Conventions [Sun] was used. Appendix G contains the coding conventions.

The Software Requirement Specifications (SRS) was developed in accordance with the IEEE standard 1233, 1998 Edition, [IEEE1233]. Appendix A contains the document developed.

The Software Design Specifications (SDS) was developed in accordance with modified IEEE standards 1016.1-1993, [IEEE1016.1] and 1016-1998, [IEEE1016]. Appendix B contains the document developed.

The Software Test Plan (STP), Software Test Procedures and the Software Test Cases (STC) documents were developed in accordance with modified IEEE standards 1012-1998, [IEEE1012]. Appendices C, D and E contain the documents developed.

4.0 REQUIREMENTS

4.1 High Level Requirements

As described in paragraph 2.1 of Appendix A of this document, the high level requirements of the ART system are:

- ART will contain a graphical user interface (GUI)
- ART will contain a parser
- ART will contain a refactoring transformer.

4.2 Detailed Requirements

As described in paragraphs 2.2 to 2.5, 2.7, 2.8, and 4.1 to 4.4 of Appendix A of this document, the detailed requirements of the ART system are:

- ART can be in the following states:
  - Read Source
  - Parse
  - Display Token
  - Refactor
  - Redisplay Token
- ART shall contain a graphical user interface (GUI) subsystem adaptable to other languages.
- ART shall contain a parser subsystem.
- ART shall contain a refactor subsystem.
- ART shall produce tokens for the source code read using the syntax grammar.
- The Parse Data option shall automatically parse the source file opened.
- The Compile option shall compile the refactored code automatically.
- ART shall perform the Encapsulate Field refactoring transformation.
- ART shall perform the Replace Magic Number with Symbolic Constant refactoring transformation.
- The refactor subsystem shall create tokens resulting from performing refactoring transformations.
- ART shall not parse a file before the language syntax is generated.
- The Parse Data option shall ensure that an opened source code file.
- ART shall store the tokens using the *Vector* class.
- ART shall implement the **Iterator pattern** for accessing the tokens.
- ART shall reformat the tokens using the **Strategy pattern**
- ART shall implement the refactoring transformations using the **Strategy pattern**
- ART shall create new tokens using the **AbstractFactory pattern.**
- The GUI shall implement the **Model-View Controller (MVC) pattern**.
- The Open option shall accept a user specified or selected file for opening.
- The Close option shall close the file(s) opened.

- The Save option shall permit the user to specify a file name before saving a file.
- The Exit option shall terminate the application.
- ART shall be written using the Java Coding Convention described in Appendix G.
- ART shall be written using the JBuilder 5.0 IDE.
- ART shall be written using the Java SDK version 1.3.
- The source code used for performing refactorings shall be error free.
- ART shall consists of the following operational scenarios:
  - CompileCode.
  - OpenFile.
  - SaveFile.
  - CloseFile.
  - RefactorCode.
  - ParseCode.
  - SelectParser.
  - EncapsulateFields.
  - ReplaceMagicNumber.
  - DisplayFile.
- ART shall allow the user to select the file for parsing.
- ART shall allow the user to select the language parser using menu selection.
- The GUI shall contain a menubar.
- The GUI shall contain two text display areas.
- The GUI shall display the source code read in one text area.
- The GUI shall display the refactored source code in one text area.
- The GUI menubar shall contain a File menu option.
- The File menu option shall contain the following sub-menu options:
  - Open
  - Close
  - Save
  - Exit
- The GUI menubar shall contain a Parse menu option.
- The Parse menu option shall contain the following sub-menu options:
  - Select Language Parser
  - Parse Data
- The GUI menubar shall contain a Compile menu option.
- The GUI menubar shall contain a Refactoring menu option.
- The Refactoring menu option shall contain the following sub-menu option:
  - Encapsulate Field
  - Replace Magic Number with Symbolic Constant
- The GUI menubar shall allow access to the options using the Alt + key combination.
- The GUI menubar options and sub-options shall be accessible using a left mouse click.
- The GUI window shall be a standard Window environment.
- The GUI window shall permit the use of buttons at the right hand corner of the window to minimize, maximize and close the window.

5.0 DESIGN

5.1 Overall Design

As mentioned in section 3.4, ART consists of three major components: a GUI, a parser and a refactor. In addition to these main components, a deparser component was created to permit the display of the source code used for parsing and the resulting refactored code. One of the benefits of ART is that each component is independent of the other. Therefore, the design of each component was performed incrementally. The parser was generated using a tool available on the world-wide web. A more in-depth discussion of the parser is given in section 6.2.

The GUI consists of a standard window menubar and two display areas. Figure 5.1.1 shows a screen capture of the GUI components. The initial class diagram of the GUI for ART is shown at Figure 5.1.2. It depicts the GUI implementation using the **Command pattern.** Following a careful analysis of the design, it was found that the **Command pattern** was not the pattern to use in this case. All the classes defined for each of the menu command were identical. Furthermore, if the GUI is partitioned, we find that we have display areas where information is shown, menu items that identify actions to be taken on the information displayed and source code files for refactoring. In other words, all the elements of the **Model-View Controller (MVC) pattern** were present. The views being the display areas and the menu bar, the model, the source code file being refactored and the controller which performs actions on the model and uses the view to present the information. The revised GUI design with the implementation of the **MVC pattern** is shown at Figure 5.1.3.

The *RefactoringTool* class acts as the controller. It contains all the action listeners for the menu items of the menu bar. The *SourceCodeView* and the *WindowMenuBarView* classes act as the views and the *ProjectFile* class acts as the model. In addition, the *WindowMenu* and *WindowMenuItems* classes are used to generate the components of the menu bar. The reason for having a class for the top level menu and the menu items components is that every time a top level menu or menu items are created, the same actions are necessary. Thus, by creating classes for the necessary actions, it becomes more flexible. By designing the GUI using the **MVC pattern**, the number of classes used was reduced from twenty-nine to eleven which makes it is easier to maintain.

The parser component of ART was created using the JavaCC tool available on the internet. JavaCC permits the generation of language parser based on grammars for an object-oriented language such as Java and C++. The code generated by the tool was grouped in a package called CIS5915JavaParser. By having created a package, the use of the import command provides access to all the classes of the package. Analysis of the code generated showed that the parser was not returning any information following its execution. Therefore, the code was changed to return a vector after the completion of the code parsing. Finally, it was discovered that the token class used by the parser was not sufficient to the requirements of the ART program. A new class, *AbstractToken* was created and used to generate tokens of the parsed code.

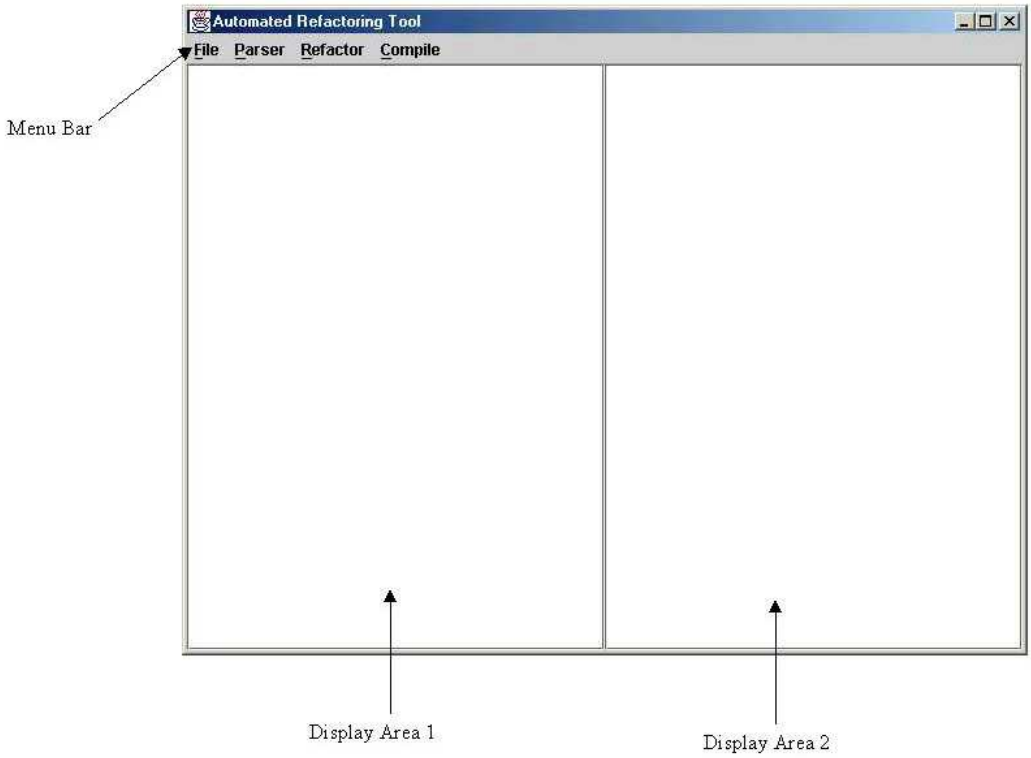


Figure 5.1.1 - ART GUI

A deparser was designed to permit the display of either the source code parsed or the refactored code. Again, the design was performed incrementally. The initial design, shown at Figure 5.1.4, implemented the **Iterator pattern.** As the parser returns a vector of tokens representing the source code parsed, a means for traversing the vector was needed. The classes, created for the deparser component, are: *JavaDeparser*, which performs the deparsing of the vector, the *ObjectCollection*, *ObjectIterator*, *TokenCollection* and *TokenIterator* classes, which implement the **Iterator pattern.** The *Indent* class was created to permit the creation of indentation objects required when reformatting the tokens. The second iteration of the design, Figure 5.1.5, resulted in the implementation of the **AbstractFactory pattern** to perform the reformatting of the tokens. It was discovered that based on the type of the token being reformatted, there were some specific reformatting rules that were applied. The classes *AbstractFactory*, *TokenFactory*, *AbstractToken*, *TokenDefault*, *TokenLBrace*, *TokenMultiLineComment*, *TokenRBrace*, *TokenSemiColon*, and *TokenSingleLineComment* were created for the implementation of the pattern. The final iteration of the design, shown at figure 5.1.6, implemented the **Strategy pattern** for performing the token reformatting. Implementing this pattern was more logical for the performance of token reformatting. When a token is reformatted, the *Deparser* class does not need to know how to reformat it and the **Strategy pattern** permits this concept. The classes *TokenContext*, *TokenStrategy*,

TokenParenBracketDotBang and *TokenReformat* were created for the implementation of the **Strategy pattern.**

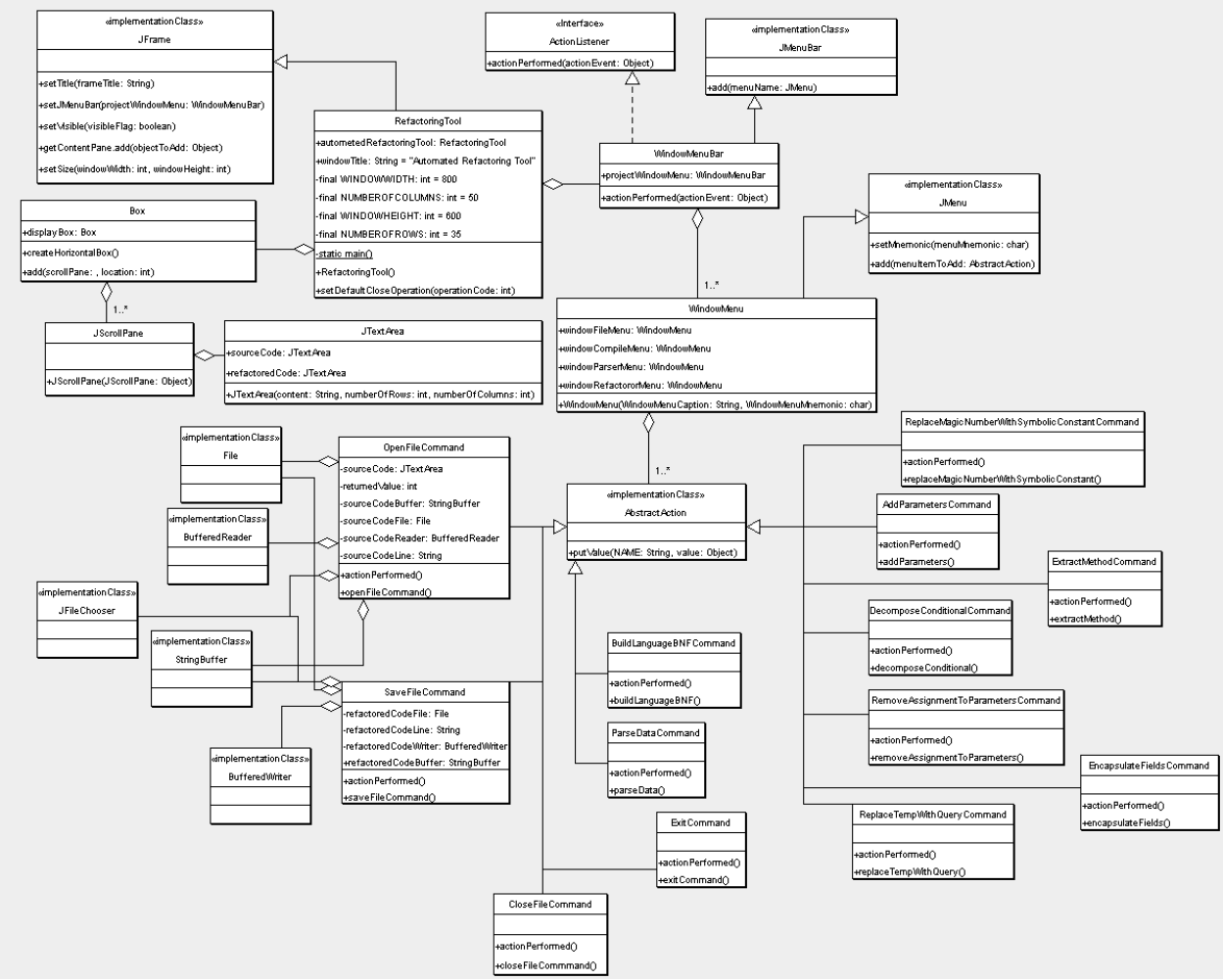


Figure 5.1.2 - ART GUI analysis level class diagram

The refactor component was designed last as it was felt it was the most difficult component of the ART tool. Careful analysis pointed to implement the refactoring transformations using the **Strategy pattern**. Each refactoring transformation is considered a different strategy and thus can be implemented independently of the others. Figure 5.1.7 shows the class diagram for the refactor design. The *RefactoringContext*, *RefactoringStrategy*, *EncapsulateFields*, and *ReplaceMagicNumber* classes were created for the implementation of the **Strategy pattern.** The algorithms used for performing the refactoring transformations were adapted from the description of the refactoring found in [Fowler00]. The *TokenFactory* and *AbstractFactory* classes were modified from their initial intent. Their new purpose is to create new tokens added to the code vector instead of for performing token reformatting. In addition, the *AbstractToken* class from the CIS5915JavaParser package is used instead of being used for token refactoring.

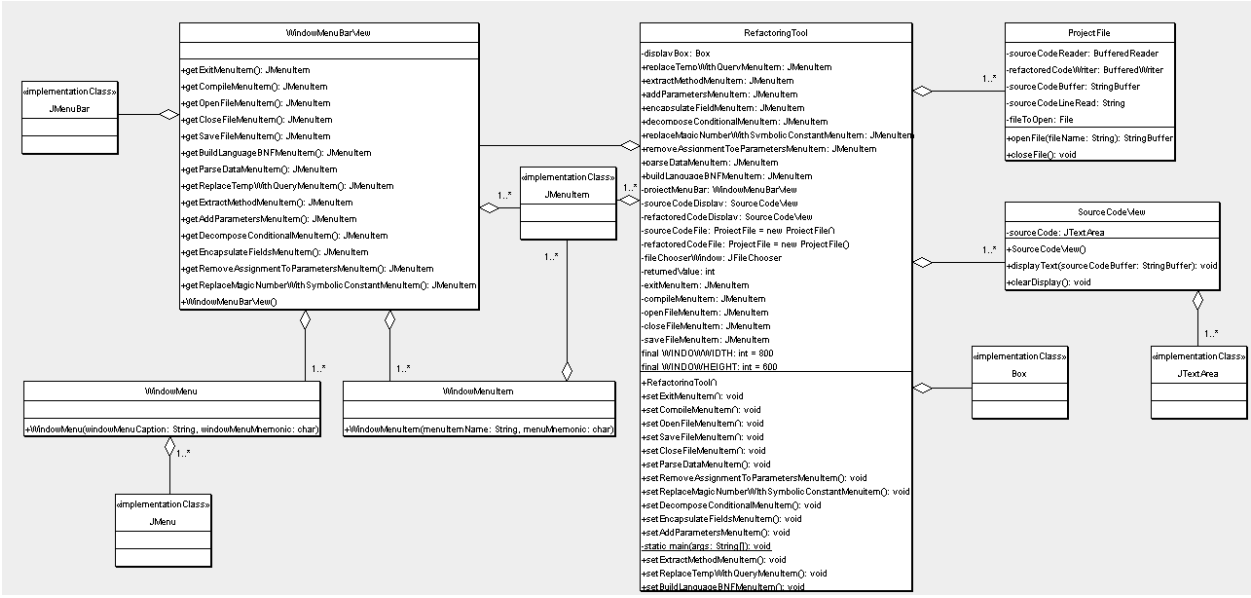


Figure 5.1.3 - Final ART GUI class diagram

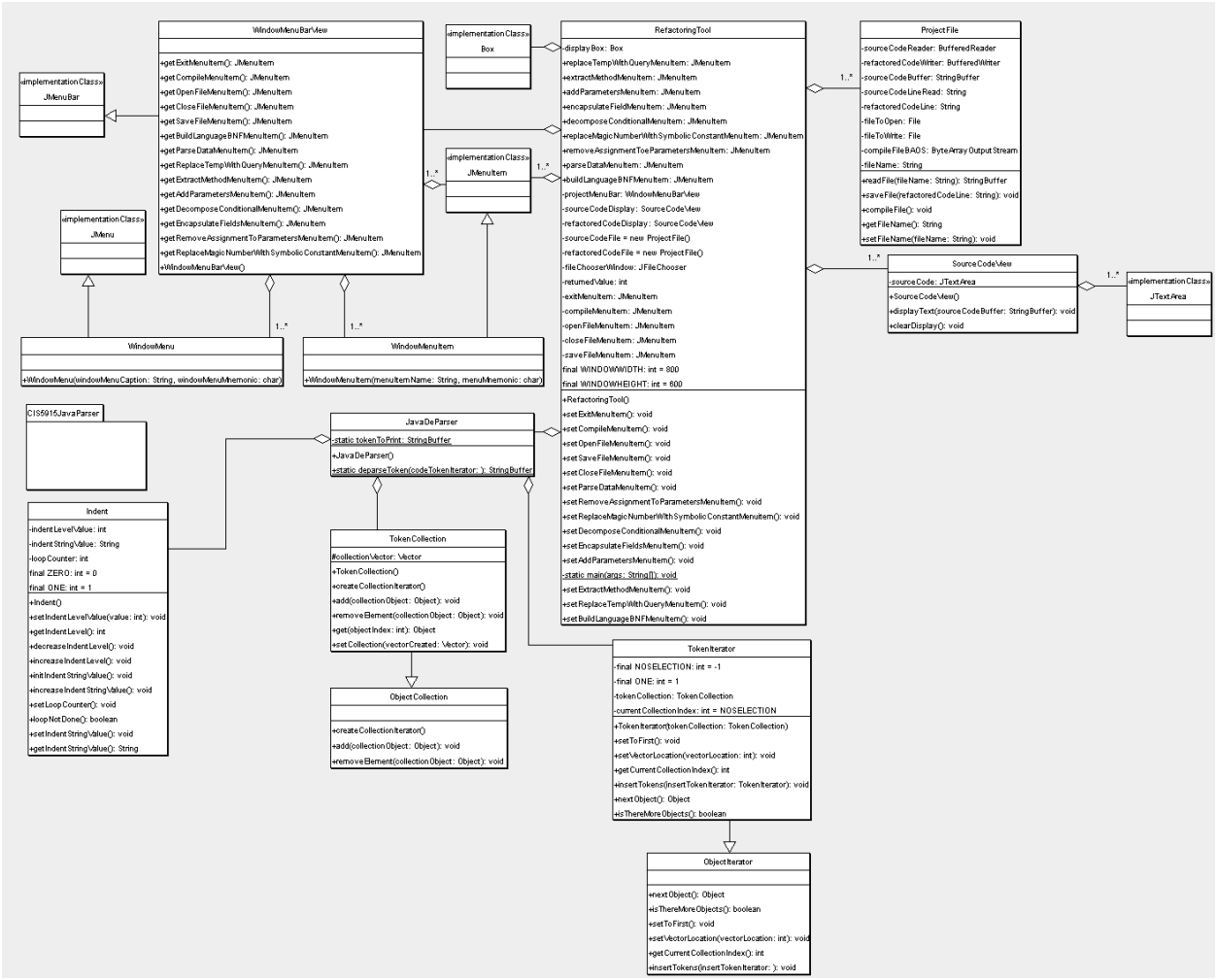


Figure 5.1.4 - Analysis level Deparser Component class diagram

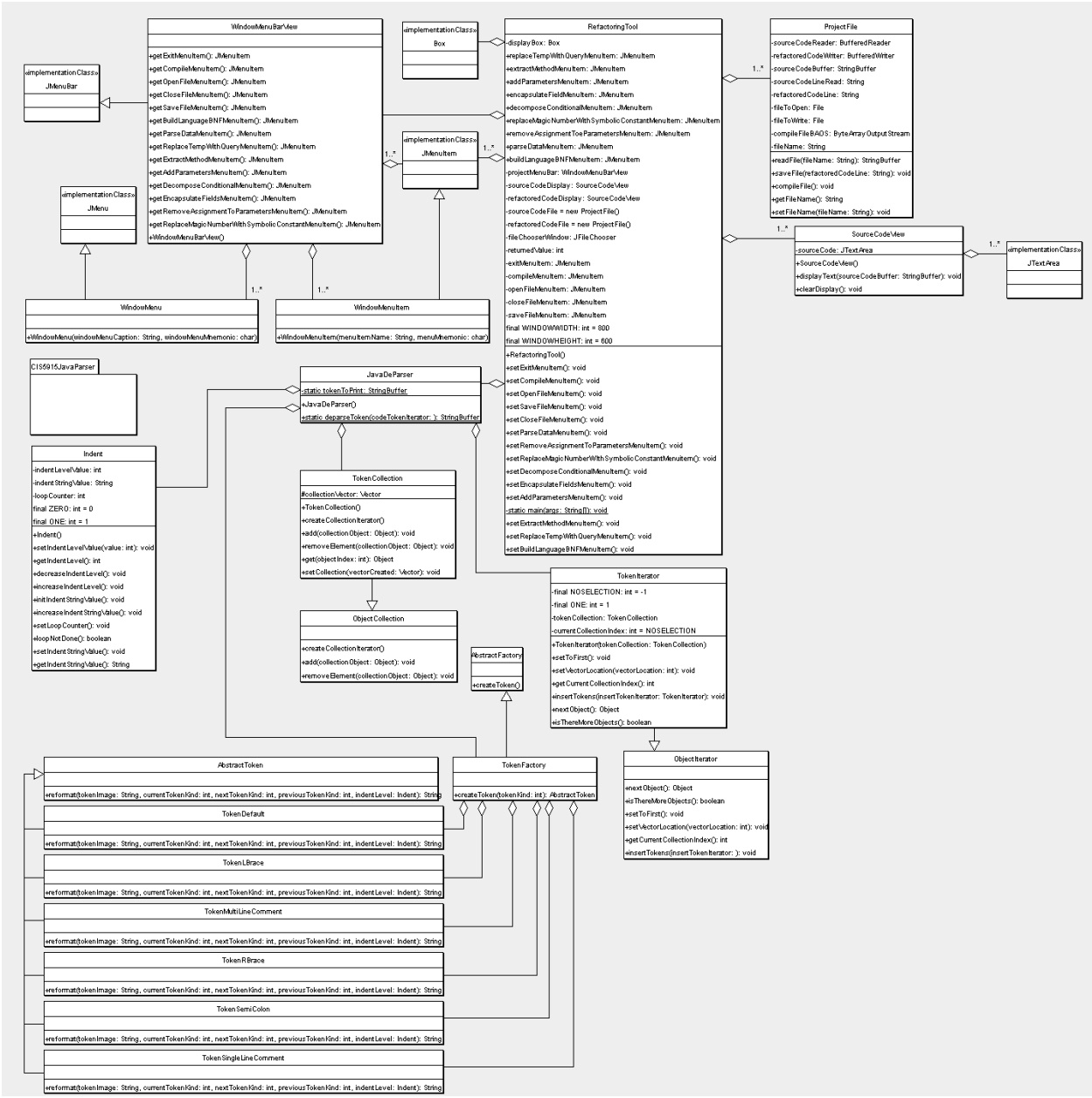


Figure 5.1.5 - Design level Deparser class diagram

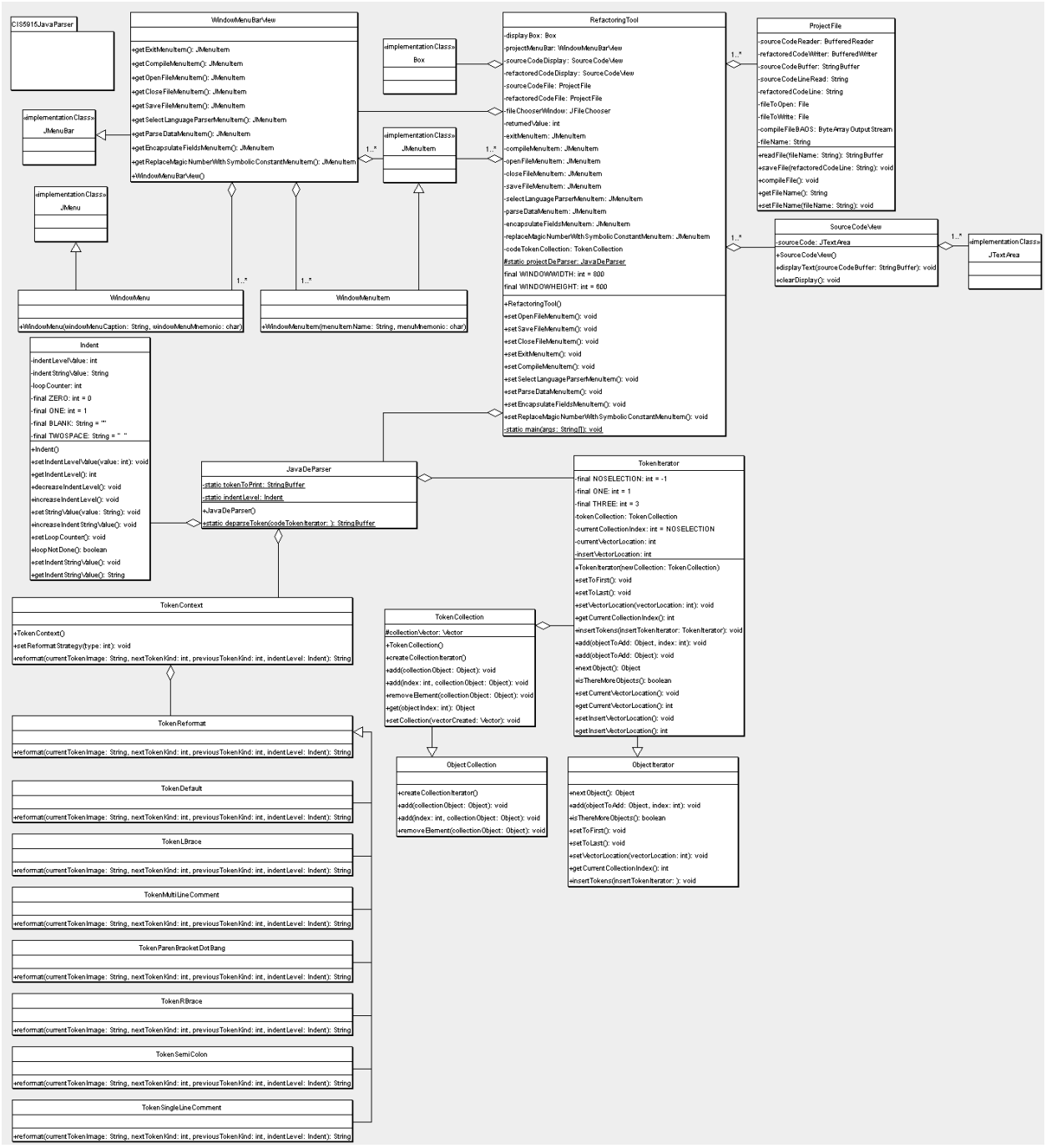


Figure 5.1.6 - Final Deparser class diagram

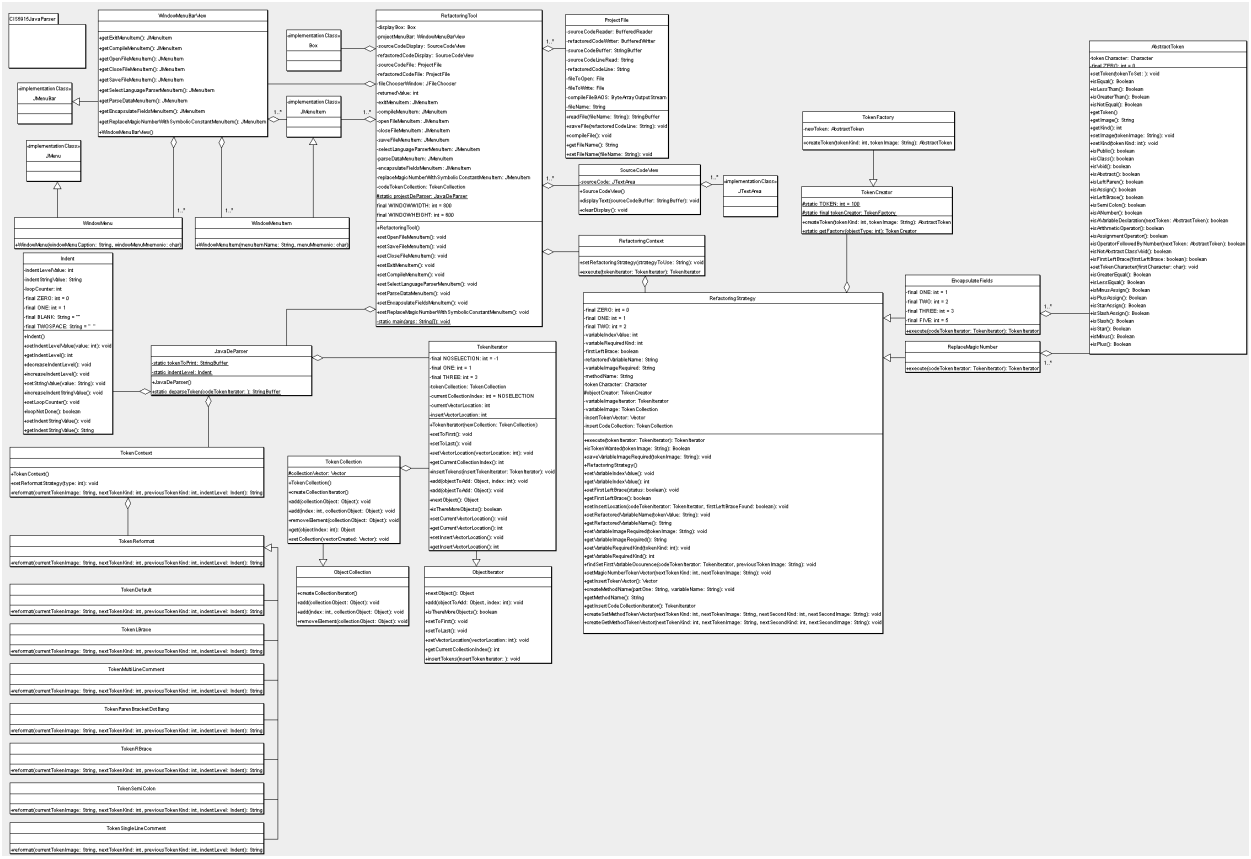


Figure 5.1.7 - Refactor class diagram

5.2 Patterns Used and Their Benefits

5.2.1 AbstractFactory pattern

[Grand] defines the forces of implementing the **AbstractFactory pattern** as *"system that works with multiple products should function independently of the specific product that it is working with, should be possible to configure a system to work with one or multiple members of a family of products, class instances intended for interfacing with a product should be used together and only with that product, remaining of the system work with a product without being aware of specific classes used for interfacing with the product and system should be extensible to work with additional products by adding additional sets of classes"*. In ART, the source code is parsed into tokens and when being refactored, new tokens are created for the various refactoring transformation strategies. In addition, these tokens are all of the same class, *AbstractToken*, but may be of different types. By using the **AbstractFactory pattern**, the tokens are created by the refactoring strategies without having the knowledge of the type of tokens being created. The **AbstractFactory pattern** is very similar to the **Strategy pattern** that was also implemented in ART and discussed in section 5.2.4. Figure 5.2.1.1 shows the **AbstractFactory pattern** implementation class diagram.

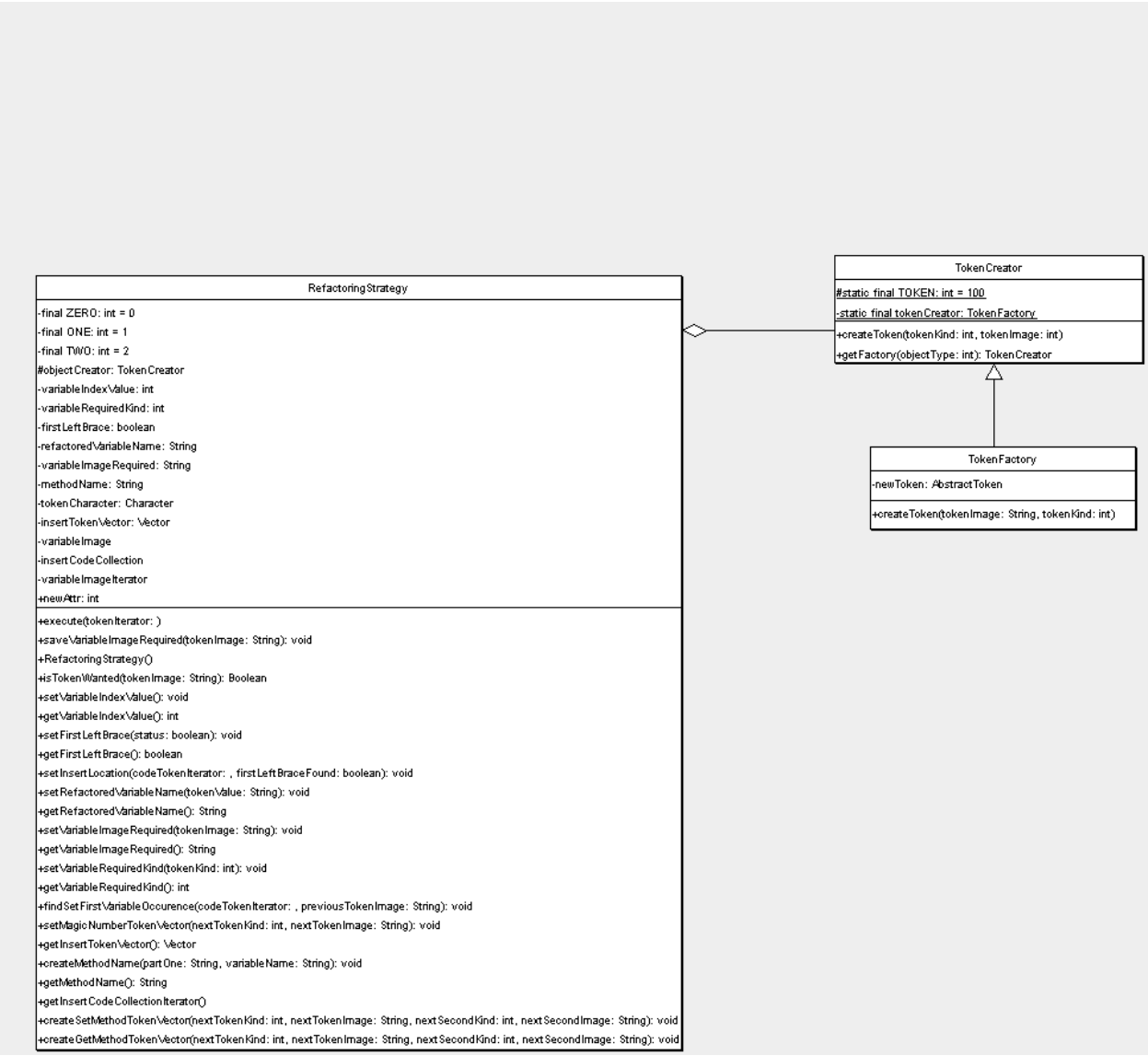


Figure 5.2.1.1 - **AbstractFactory pattern** implementation class diagram

5.2.2 Iterator pattern

[Grand] defines the forces of implementing the **Iterator pattern** as *"a class needs access to the content of a collection without dependency of the class used to implement the collection"* and *"class needs a uniform way of accessing multiple collections"*. As the *Vector* object is used to store the tokens generated after parsing and refactoring a file, a means that keep independent the *Vector* object from the classes that access the *Vector* object is necessary thus the implementation of the **Iterator pattern.** In Java, the *Vector* object is accessible using the *Enumerator* object resulting in each class using the vector to know the underlying collection of object. In addition, the *Enumerator* object is not flexible in permitting access at specific location. In ART, the *Deparser*, *EncapsulateFields* and *ReplaceMagicNumber* classes need to

access and modify the content of the *Vector* object. Figure 5.2.2.1 shows the **Iterator pattern** implementation class diagram.

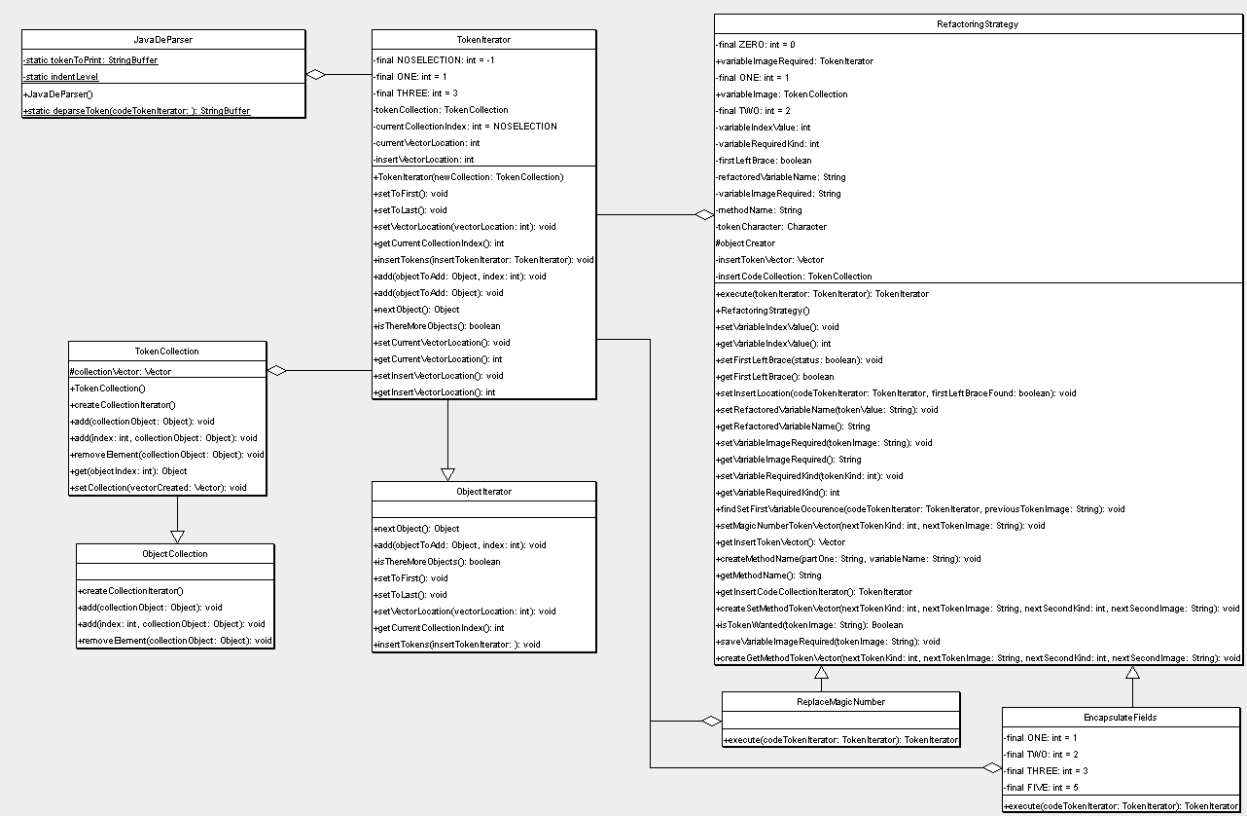


Figure 5.2.2.1 - **Iterator pattern** implementation class diagram

5.2.3 Model-View Controller pattern

The **Model-View Controller (MVC) pattern** was implemented because we can separate the components of ART into the three categories which the pattern is build on: a model, a view and a controller. The model in ART are the files containing the code to be refactored, the view(s) are the menu bar and the display areas and the controller is the class which performs actions using the views and/or the model. The classes, implemented for the **MVC pattern**, are: *RefactoringTool* that acts as the controller, *ProjectFile* that acts as the model and *SourceCodeView* and *WindowMenubarView* that acts as the views. One of the benefits of the **MVC pattern** is that it decouples the GUI from the code implementing the various algorithms implemented in an application. In addition, the **MVC pattern** permits to apply independently changes to the GUI, the views or the model. Figure 5.2.3.1 shows the ART **MVC Pattern** implementation class diagram.

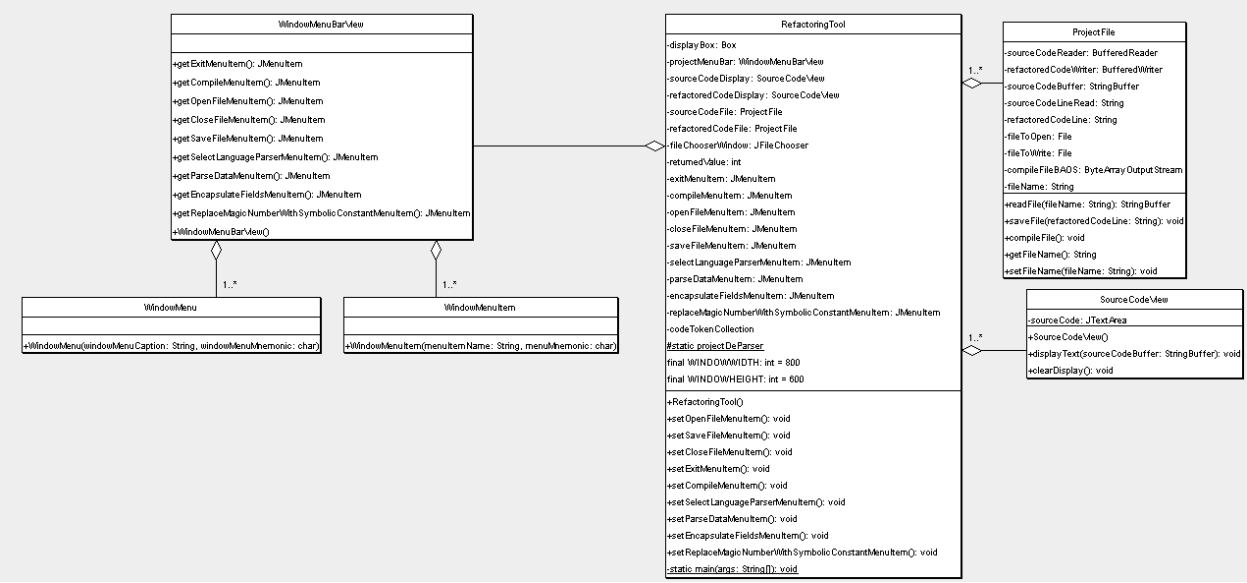


Figure 5.2.3.1 - **MVC pattern** implementation class diagram

5.2.4 Strategy pattern

[Grand] defines the forces of implementing the **Strategy pattern** as *"a program has to provide multiple variations of an algorithm or behavior, behavior variations can be encapsulated in various classes providing a consistent access methodology to these behaviors, and classes using these behaviors do not require knowledge of the implementation of the behaviors"*. In ART, there are two situations lending themselves to the **Strategy pattern**. When reformatting tokens, the behavior varies based on the type of token being reformatted. When reformatting tokens, the deparser only needs to know that a token is being reformatted. The other situation lending itself to implementing the **Strategy pattern** is the refactoring transformations. Each of the transformation consists of a set of rules that must be followed. The **Strategy pattern** is easily implemented, as each refactoring is a different strategy. In addition, some common behavior to all refactorings is factored in the common strategy superclass so that the only behavior included in each of the strategies is the rules necessary. Figure 5.2.4.1 shows the ART **Strategy pattern** implementation class diagram for the token reformatting and Figure 5.2.4.2 shows the ART **Strategy pattern** implementation class diagram for the refactoring transformations.

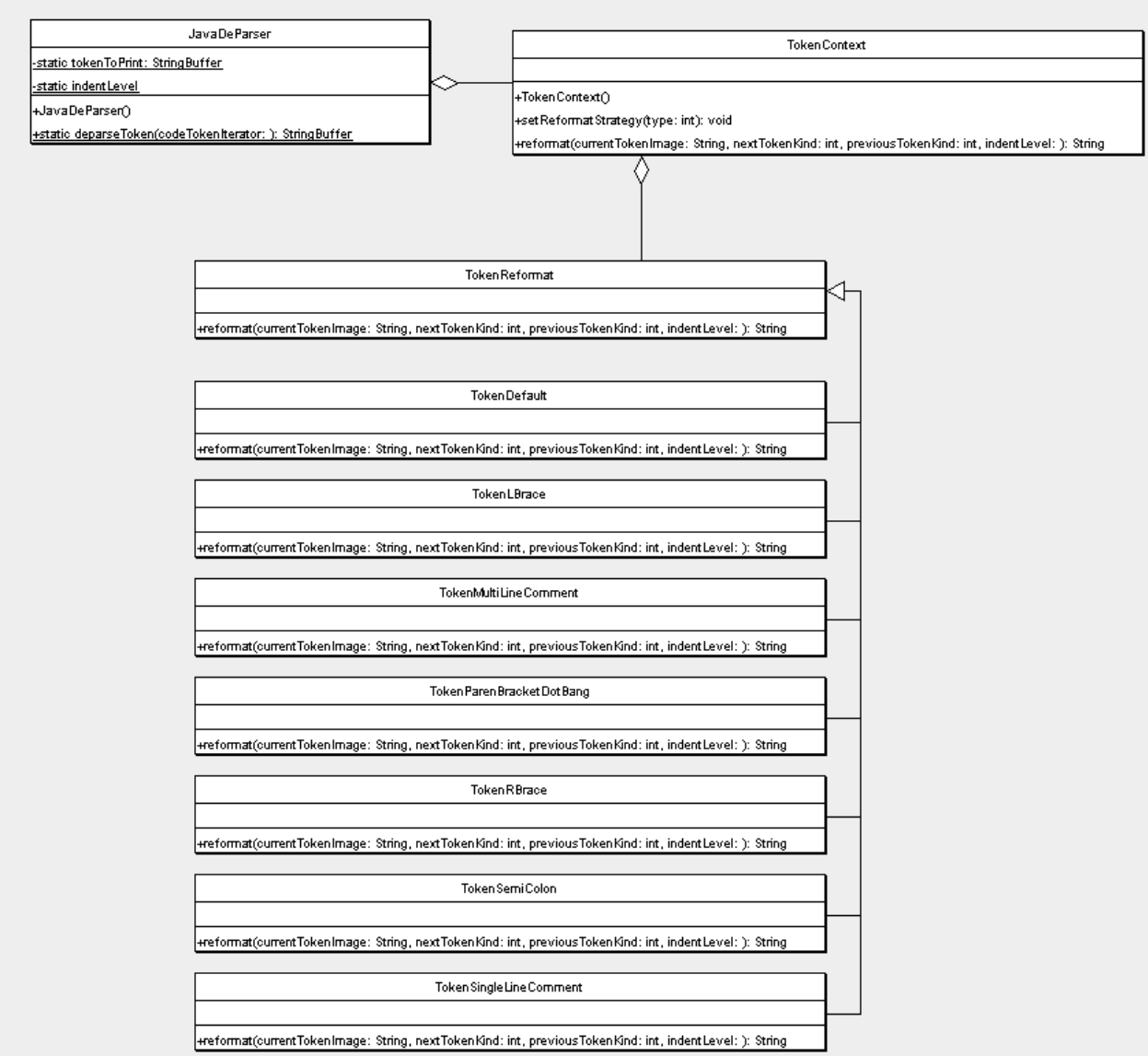


Figure 5.2.4.1 - **Strategy pattern** implementation class diagram for reformatting tokens

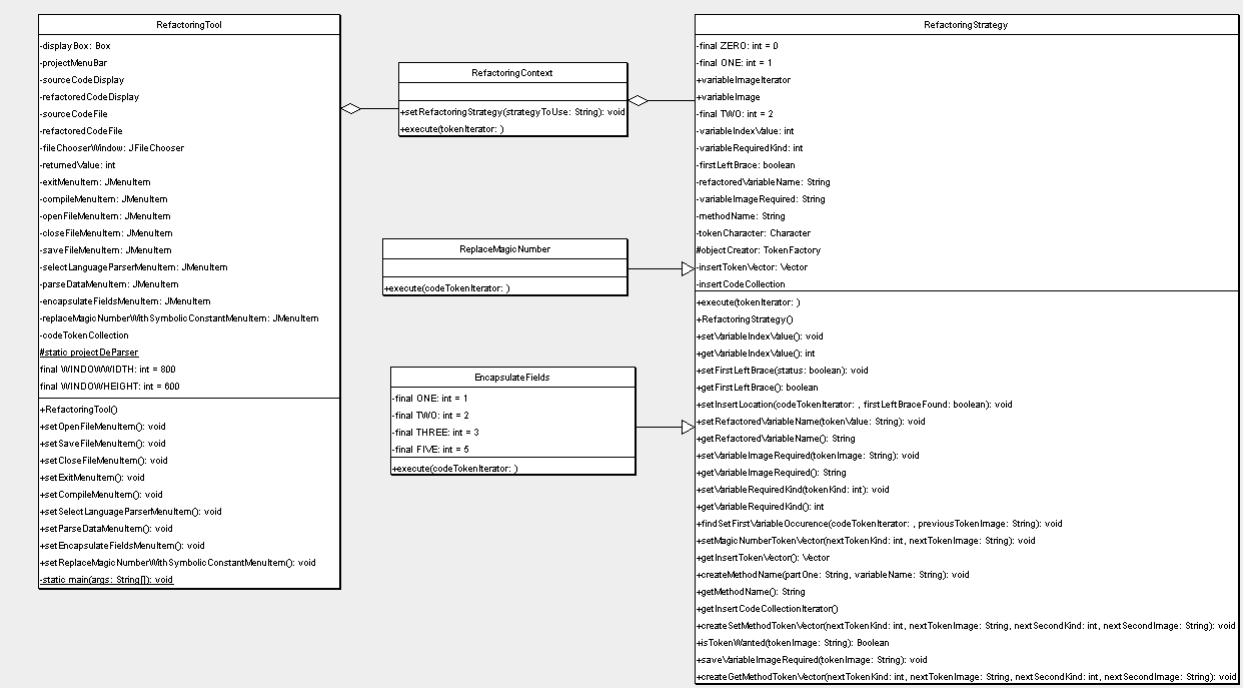


Figure 5.2.4.2 - **Strategy pattern** implementation class diagram for refactoring transformations

6.0 SOFTWARE DESCRIPTION

6.1 Graphical User Interface

When ART executes, it creates the class *RefactoringTool* that is responsible to create the entire GUI and the action listeners required. The *RefactoringTool* class instantiates classes to create the menu bar, the two display areas, for reading and writing source code, for reformatting the code and for performing the refactoring transformations. In addition, it defines action listener methods for each of the menu items contained in the menu bar.

To create the two views as part of the **MVC pattern**, the class *WindowMenuBarView* is instantiated. The classes' constructor creates each of the top-level menus by instantiating the class *WindowMenu* and creates each of the menu items by instantiating the class *WindowMenuItem*. Class *WindowMenuBar* also creates the get methods for each of the menu items. When, the display areas are created by class *RefactoringTool* it first creates a *Box* container object that will hold the two *JScrollPane* objects used to display the code in. The *JScrollPane* objects contain a *SourceCodeView* object representing either the source code or the refactored code. To create the model, the *ProjectFile* class is instantiated to permit access to the source and refactored code.

Because the *RefactoringTool* class acts as the controller in the **MVC pattern** implementation, it declares and executes the set methods of all the menu items that are part of the menu bar. Each of the set methods consist of retrieving the appropriate menu item using the get methods defined in the *WindowMenuBarView* object and adding an action listener that permits executing commands based on the user selection.

The effect of implementing the **MVC pattern** for the ART GUI makes adding top-level menus, menu items or views simple. For adding top-level menus and/or menu items, the class *WindowMenuBarView* would need modifications to include the new menus and/or menu items. This is accomplished by instantiating the classes *WindowMenu* and *WindowMenuItem* and adding the associated get methods. Then, class *RefactoringTool* would need modifications to include the set method(s) associated with the new menu items created. To add new views to the GUI, the classes representing the new views would require being instantiated. Finally, the action listener(s) required by the new menu items would need to be created.

6.2 Parsing

The parser, used in ART, was generated using JavaCC version 2.1, freely available at http://www.suntest.com/JavaCC. It creates object-oriented language parsers based on user specified requirements. When the user has specified the language, it generates Java code representing the complete parser. The parser takes for input a text file consisting of the code to be parsed. It reads the code, line by line, and using the appropriate language grammar, the parser converts each of the line of code into tokens. However, the parser code generated required modifications because ART uses a vector of tokens representing the source or refactored code. The parser code was changed such that it returned a vector containing the parsed tokens. In addition, when the class *AbstractToken* was implemented into the ART design, it was added to

the parser package. Thus, when the parser returns a vector of tokens, they are already of the *AbstractToken* class.

To view the parsed code, a deparser was created for ART. The deparser is instantiated by the *RefactoringTool* class and used either to display the source code parsed or refactored. The deparser executes when the parser returns the token vector within the action listeners for the Parse Data, the Encapsulate Fields and Replace Magic Number menu selections. The deparser traverses the vector of tokens and based on the type of token, it reformats the token using the strategy defined for that token. Because the reformatting of tokens is implemented using the **Strategy pattern**, the deparser sets the reformatting strategy using the type of token, which in turn executes the reformat method for the selected strategy.

To add different language parsers to the application, JavaCC would require being executed specifying the new language. In addition, the parser code generated would require the modification, mentioned above, to return the vector that ART needs. In addition, reformatting strategies would be required for that language if the strategies do not exist. Creating new reformatting strategies would involve only creating new classes to cover the missing rules and adding the new strategy choices available.

6.3 Refactoring

ART implements refactoring transformations using the **Strategy pattern.** When a user selects a refactoring transformation, the strategy is set appropriately based on the selection and the refactoring transformation's rules execute. The set of rules, used to implement the refactorings, was based on [Fowler00] and transposed into Java code.

For the Replace Magic Number with Symbolic Constant refactoring, the vector of token is searched for any occurrence where a numeric value is assigned to a variable or used in an arithmetic operation. When such an occurrence is found, the numeric value is changed to a constant variable that is created. The variable declaration is consistent with Appendix G as well as being the type of the numeric value. The tokens created for the constant newly created is added to the vector of tokens at the beginning of the variable declaration section.

For the Encapsulate Fields refactoring, the vector of tokens is searched for public declaration of variables. When such an occurrence is found, two methods are created, a set and a get, with the appropriate parameters. Then the vector is searched for occurrences of the variable being used and being assigned a value. In either case, the set or get methods, with the proper parameters, replace the variable name usage. The set and get methods created are added to the end of the code.

The *RefactoringStrategy* superclass defines methods that any refactoring transformations may require. As this project provides a basic framework from which to build upon, some methods may be missing or were created without knowing if other refactorings may require them. In any case, adding methods to the *RefactoringStrategy* class is simple.

Adding new refactoring transformations to ART is simple because of the **Strategy pattern** implementation. To do so would only require the addition of the new menu items, see

section 6.1, the addition of the new strategy choice and the new class that would contain the refactoring transformation set of rules.

6.4 Compiling

After the code is refactored, it can be compiled directly by ART without having to exit the application. The method compile, available under the package sun.tools.javac.Main, is used as part of the compile option action listener. The latter sets some arguments required for the compile method then it instantiates a new *ByteArrayOutputStream* object. Finally, the action listener creates an object for compiling using the *sun.tools.javac.Main* class using as arguments the *ByteArrayOutputStream* object and execute the compile method using the compile arguments set previously. It returns either true or false based on the results of the compilation.

7.0 SOFTWARE VERIFICATION AND VALIDATION

7.1 Testing Philosophy

Because the development of ART was done incrementally, the various tests designed and performed were also incremental. Each of the test cases contained in Appendix E, verify specific requirements. Nevertheless, the nature of the application results in having regression testing executed while testing new requirements. For example, when testing the parser subsystem in test case TCS-002, some of the functionalities of the GUI are tested again even though the functionality was proven during the execution of test case TCS-001. Therefore, when performing test case TCS-004, the functionality of the entire application was verified and validated even after the successful completion of the previous test cases. The documents generated during the development of ART were validated and verified as they were produced. If a previous document was used to generate a new one, it was used to ensure consistency and completeness prior to accepting the newly generated document.

7.2 Tests Performed

Appendices C, D, and E define the test plan, test procedures and test cases designed and executed for verifying and validating ART. All the test cases defined in Appendix E were executed and completed successfully. When the GUI for ART was finalized, less the various functionality for each of the menu options, test case TCS-001 was executed. The purpose of TCS-001 was to prove the implementation of the MVC pattern as well as the functionality of the GUI. Following the completion of the parser component of ART, test case TCS-002 was performed. The purpose of TCS-002 was to prove that an open source code file could be read, parsed and redisplayed properly and that the tokens generated by the parser were valid. TCS-002 also proved the final implementation of the open, close menu items as well as the proper display and reformatting of the tokens generated. Next was performing test case TCS-003 following the completion of the refactor component. The purpose of test case TCS-003 was to prove that after the implementation of the two (2) refactoring transformation, the code generated was correct and error free. Another purpose for TCS-003 was to prove the framework developed. As each of the refactoring transformation is independent of the other, their implementation can be in any order. Finally, test case TCS-004 was performed. The purpose of TCS-004 was to prove any requirements not covered by the previous test cases as well as ensuring the entire functionality of ART.

Paragraph 5.3 of the SDS, Appendix B, shows the relationship between the test cases and most of the requirements contained in the SRS, Appendix A. The reason for that is that the missing requirements, 2.7.1, 2.7.2 and 2.7.3, are intangible and not verifiable using a specific test case. For example, requirement 2.7.1 states that the application shall be written according to coding conventions. The only method permitting to verify and validate this type of requirement is to perform a code review to ensure proper conformance to the coding conventions.

8.0 FUTURE RESEARCH

8.1 Adaptability of New Refactoring Transformations

The implementation of the **Strategy pattern** for the refactoring transformation makes adding new refactorings simpler. First, the strategy superclass developed lends easily itself to expansion without affecting its superclasses. Second, each of the new refactorings being implemented is treated individually. Furthermore, some of the existing refactorings implemented can be accessed for the new refactorings implementation. However, some massaging may be required as new refactoring are introduced in ART. In [Fowler00], when discussing refactoring transformations, some refactorings require the execution of others. By having the refactoring transformations implemented as subclasses, a call to the execute method of the appropriate strategy subclass would be necessary.

8.2 Pattern Recognition

An area where ART can benefit a lot is pattern recognition. As discussed in [Arcelli] and [Grimmond], pattern recognition is the method by which Java code is analyzed using pre-defined rules and design patterns are identified. When reading code, ART could be designed in such a manner that based on predefined rules, some design patterns could be recognized. After the recognition of a pattern, the code is modified to implement the particular pattern using pre-defined templates. It would add a new dimension to ART that would improve its flexibility.

8.3 Reflective Architecture

[Kleinoeder] and [Schubert] define reflective architecture as the capability of objects of knowing their run-time environment and to change their properties. In ART, although the GUI was implemented using the **MVC pattern**, an improvement to this design would be the implementation of reflection for the various menu items. When looking at the code generated for all the action listeners required for a menu system, it is quickly realized that because of the wanted action, the code varies. If a reflective architecture could be implemented, we could have all the execution information for the various actions kept at the meta level. Then when executing the application, when a certain action is accessed, the object representing the menu items would execute the actions using the meta-level information. This greatly enhances the flexibility of the framework, as adding new menu items requires only changing the meta-level information.

AUTOMATED REFACTORING TOOL (ART)

APPENDIX A

SOFTWARE REQUIREMENT SPECIFICATIONS (SRS)

APPENDIX A

TABLE OF CONTENTS

1. INTRODUCTION 1

1.1 System Purpose 1
1.2 System Scope 1
1.3 Definitions, Acronyms and Abbreviations 1
1.4 References 1
1.5 System Overview 2

2.0 GENERAL SYSTEM DESCRIPTION 2

2.1 System Context 2
2.1.1 Application GUI subsystem 2
2.1.2 Parser subsystem 2
2.1.3 Refactor subsystem 5
2.1.4 ART Analysis Class Diagram 7
2.1.5 ART Entity-Relationship Diagram 7
2.1.5.1 ART ERD DeMarco Composition 8
2.2 System modes and states 8
2.3 Major system capabilities 9
2.4 Major system conditions 10
2.5 Major system constraints 11
2.6 User characteristics 11
2.7 Assumptions and dependencies 12
2.8 Operational scenarios 12
2.8.1 Use Case DeMarco Composition 14

3.0 SYSTEM CAPABILITIES, CONDITIONS, AND CONSTRAINTS 14

3.1 Physical 14
3.1.1 Construction 14
3.1.2 Durability 14
3.1.3 Adaptability 14
3.1.4 Environmental conditions 15
3.2 System performance characteristics 15
3.3 System security 15
3.4 Information management 15
3.5 System operations 15
3.5.1 System human factors 15
3.5.2 System maintainability 15
3.5.3 System reliability 15
3.6 Policy and regulation 15
3.7 System life cycle sustainment 15

4.0 SYSTEM INTERFACES 15

4.1 System interfaces requirements 15
4.2 Parser Package 16
4.3 System Interfaces Forms 17
4.3.1 ART Main Window (ARTMainWindow) 17
4.3.1.1 ART Main Window DeMarco Composition 17
4.3.2 File Menu Options (WindowFileMenu) 18
4.3.2.1 File Menu Options DeMarco Composition 18
4.3.3 Parser Menu Options (WindowParseMenu) 19
4.3.3.1 Parser Menu Options DeMarco Composition 19
4.3.4 Refactor Menu Options (WindowRefactorMenu) 20
4.3.4.1 Refactor Menu Options DeMarco Composition 20

APPENDIX A

4.3.5 Compile Menu Option 21
4.3.5.1 Compile Menu Options DeMarco Composition 21
4.3.6 Display View form 22
4.3.6.1 Display View DeMarco Composition 22
4.3.7 Open File Dialog Window 23
4.3.7.1 Open File Dialog Window DeMarco Composition 23
4.3.8 Save File Dialog Window 24
4.3.8.1 Save File Dialog Window DeMarco Composition 24
4.4 ART Data Dictionary 25

APPENDIX A

LIST OF FIGURES

Figure 2.1 - ART decomposition diagram 3
Figure 2.1.1 - Application GUI subsystem decomposition diagram 3
Figure 2.1.1.1 - Menu options decomposition diagram 4
Figure 2.1.1.2 - File Menu option decomposition diagram 4
Figure 2.1.2 - Parser subsystem decomposition diagram 5
Figure 2.1.2.1 - Parser subsystem functional diagram 5
Figure 2.1.3 - Refactor subsystem decomposition diagram 6
Figure 2.1.3.1 - Refactor subsystem functional diagram 6
Figure 2.1.4.1 - ART Analysis Class Diagram 7
Figure 2.1.5.1 - ART ERD 8
Figure 2.2.1 - ART State Diagram 9
Figure 2.8 - ART Use Cases Diagram 12
Figure 4.3.1.1 - ART Main Window 17
Figure 4.3.2.1 - File Menu Options 18
Figure 4.3.3.1- Parser Menu Options 19
Figure 4.3.4.1 - Refactor Menu Options 20
Figure 4.3.5.1 - Compile Menu Option 21
Figure 4.3.6.1 - Display View form 22
Figure 4.3.7.1- Open File Dialog Window 23
Figure 4.3.8.1 - Save File Dialog Window 24

1. Introduction

1.1 System Purpose

When software engineers use object-oriented language to develop source code, they perform manual refactoring transformations on the code written. The refactoring transformations applied to source code make the code more flexible, portable and maintainable. There exist many refactoring transformations as described by Fowler, [Fowler00]. However, the application of the refactoring transformations can be difficult and time consuming. The purpose of the Automated Refactoring Tool (ART) is to permit software engineers the capability to perform refactoring transformation on object-oriented source code. The main objective of ART is the partial automation of the refactoring transformations. In addition, ART permits the compilation of refactored Java code without terminating the tool execution.

1.2 System Scope

ART consists of three main components:

a) a parser, responsible for building a vector of tokens representing the source code being refactored,
b) a refactoring transformer, responsible to apply refactorings to the source code, and,
c) a graphical user interface (GUI), responsible for displaying the source code and its refactored counterpart as well as providing user access to the various options of ART.

The parser is capable of using the syntax grammar for Java, C++ and other object-oriented languages. Because of limitations in the Java language, source code written in object-oriented languages other than Java cannot be compiled within the tool. Finally, two (2) refactoring transformations are implemented. They are:

a) Encapsulate Field,
b) Replace Magic Number with Symbolic Constant.

1.3 Definitions, Acronyms and Abbreviations

ART	Automated Refactoring Tool
ERD	Entity-relationship Diagram
GUI	Graphical User Interface
IEEE	Institute of Electrical and Electronics Engineers
MVC	Model-View Controller
SDK	Software Development Kit

1.4 References

NUMBER	TITLE	DATE
IEEE Std 1233	IEEE Guide for Developing Software Requirements Specifications	1998

1.5 System Overview

ART consists of three (3) components: a parser, a refactoring transformer and a GUI. The parser consists of reading a source code file and generating a vector of tokens representing the source code to be refactored. The refactoring transformer uses the generated vector of tokens to perform specific refactoring transformations. The GUI permits the developer to select the refactoring transformation, to perform file operations, to display the source code and its refactored version and to compile the refactored code (Java language only).

2.0 General System Description

2.1 System Context

As mentioned before, ART will consists of three components/subsystems: the application GUI, the parser and the refactoring transformer. When the ART application is started (decomposition diagram, Figure 2.1), the application GUI subsystem will be invoked. It will provide the user with menu access for the Parser and Refactor subsystems, file operations and displaying the initial and refactored source code. The Parser subsystem will provide the refactor subsystem a vector of tokens representing the source code parsed to perform the refactoring transformations. The Refactor subsystem will send to the Application GUI subsystem a vector of tokens representing the refactored code for displaying.

2.1.1 Application GUI subsystem

The application GUI subsystem will provide the user with various menu options (decomposition diagram, Figure 2.1.1). The menu options available to the user will be File, Parser, Refactor and Compile (decomposition diagram, Figure 2.1.1.1. The File menu option will provide further options to the user as define in the decomposition diagram, Figure 2.1.1.2. The Open File option will allow the user to select or specify a source code file to be opened and will be displayed in a display area. The Close File option will allow the user to close the files that are currently opened/displayed. The Save File option will allow the user to save the files that are currently opened/displayed. The Exit option will allow the user to terminate the application.

2.1.2 Parser subsystem

The Parser subsystem will provide the user with two menu options (decomposition diagram, Figure 2.1.2). The menu options available to the user will be Select Parser Language and Parse Data. The Select Parser Language option will permit the user to identify the language of the parser. The Parse Data option will parse the source code file.

APPENDIX A

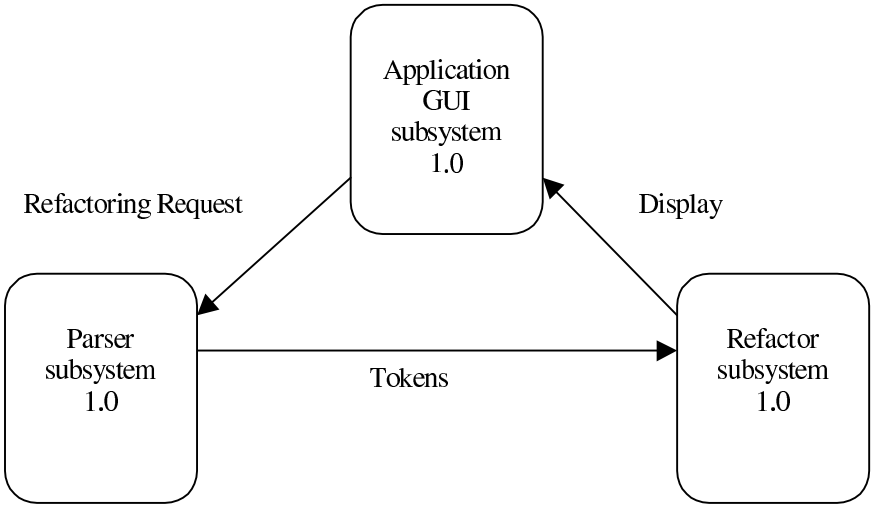


Figure 2.1 - ART decomposition diagram

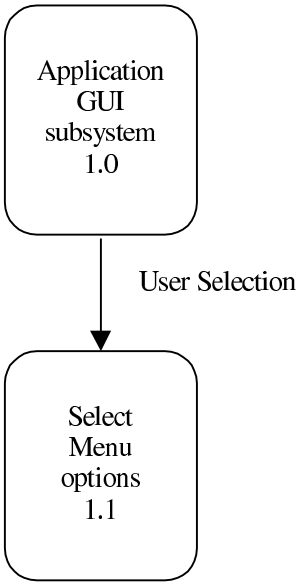


Figure 2.1.1 - Application GUI subsystem decomposition diagram

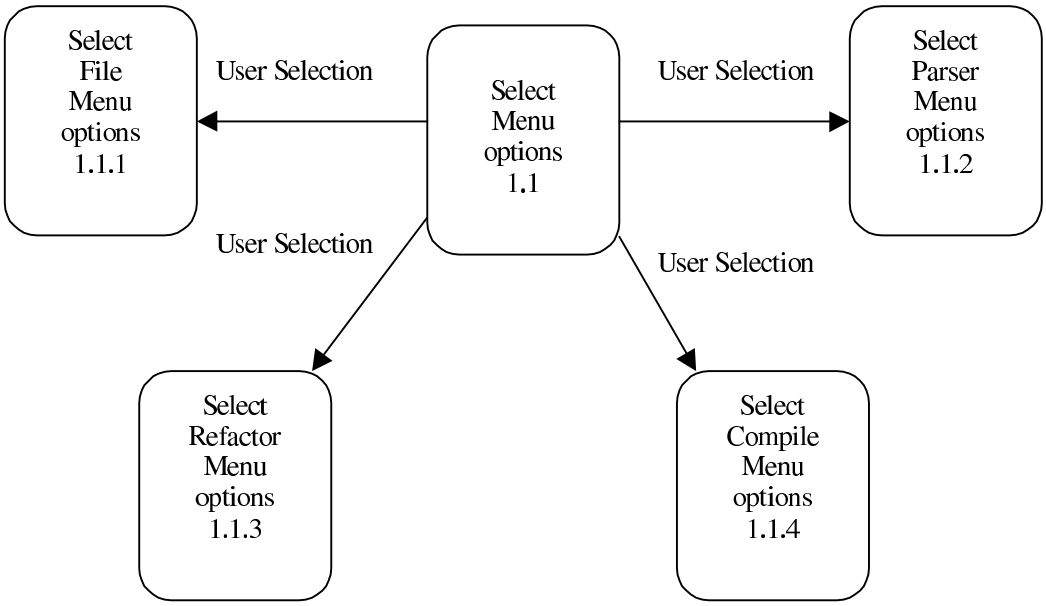


Figure 2.1.1.1 - Menu options decomposition diagram

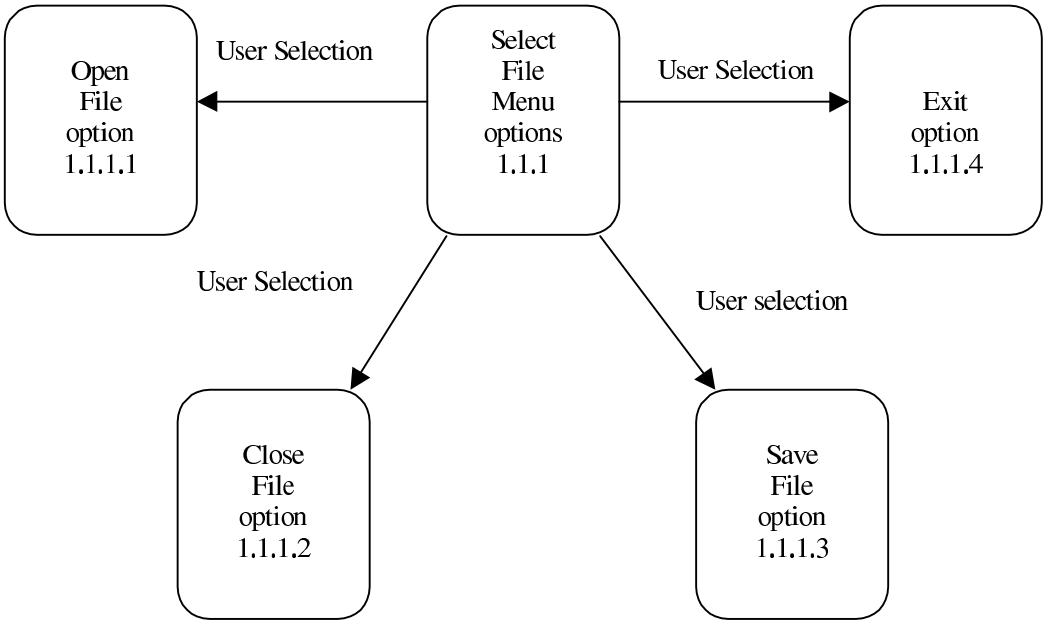


Figure 2.1.1.2 - File Menu option decomposition diagram

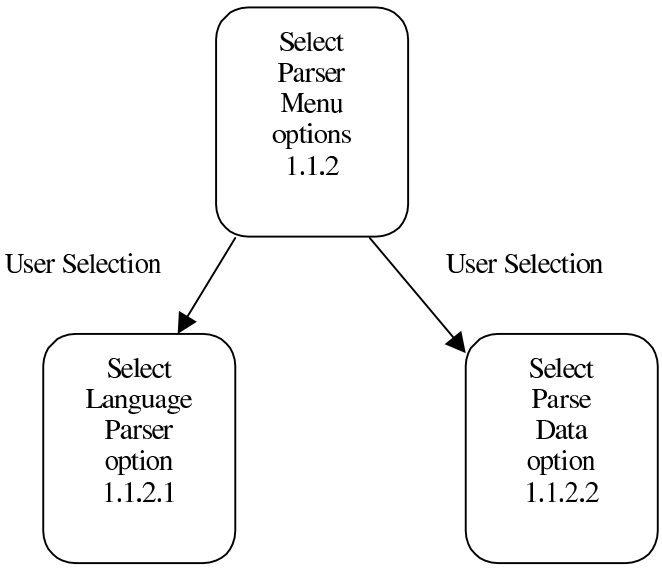


Figure 2.1.2 - Parser subsystem decomposition diagram

The functional diagram of the Parser subsystem is shown at Figure 2.1.2.1. In brief, the Parser subsystem consists of two (2) functional areas: selection of the language parser and parsing the data. Various language parsers will be accessible by the Parser subsystem through user selection. Parsing the data will use the source code of the file opened and the selected language parser to generate tokens. These tokens represent the source code to be refactored and are forwarded to the refactor subsystem.

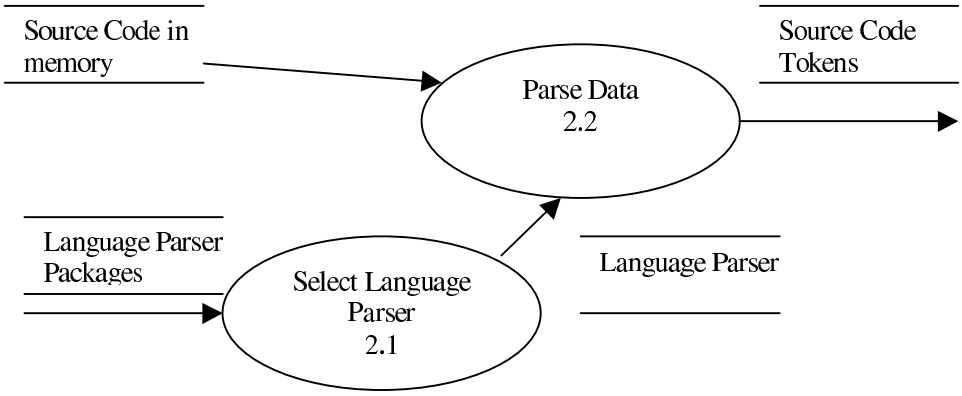


Figure 2.1.2.1 - Parser subsystem functional diagram

2.1.3 Refactor subsystem

The Refactor subsystem will provide the user with two menu options (decomposition

diagram, Figure 2.1.3). The menu options available to the user will be Encapsulate Fields option and Replace Magic Number with Symbolic Constant option. The Encapsulate Fields menu option will permit the user to apply the Encapsulate Fields refactoring on source code. The replace Magic Number with Symbolic Constant option will permit the user to apply the Replace Magic Number with Symbolic Constant refactoring to source code.

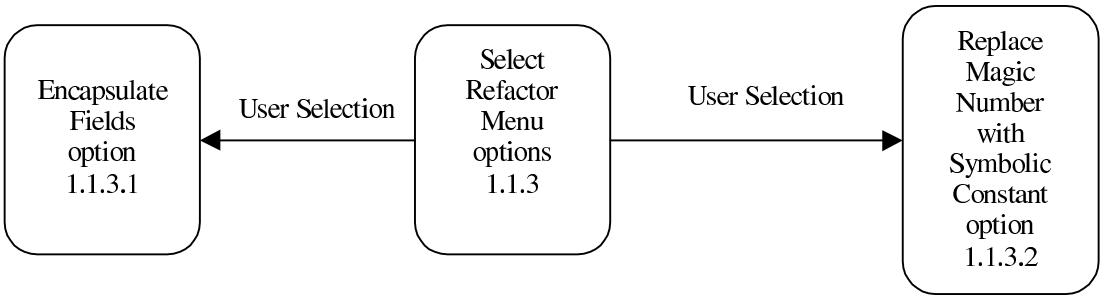


Figure 2.1.3 - Refactor subsystem decomposition diagram

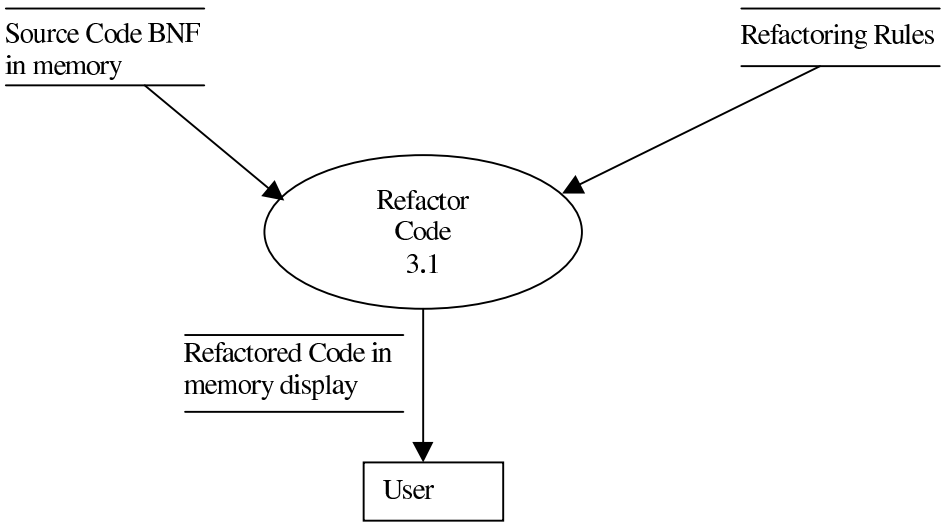


Figure 2.1.3.1 - Refactor subsystem functional diagram

The functional diagram of the refactor subsystem is shown at Figure 2.1.3.1. In brief, it consists of one (1) functional area: refactor code. This function takes the tokens produced by the Parser subsystem and applies to them the refactoring rules of the refactoring transformations

selected.

2.1.4 ART Analysis Class Diagram

Figure 2.1.4.1 defines the analysis class diagram for the ART system. They are five (5) classes: Source Code, Parser, Token, Grammar and Refactor. The class source code consists of the source code parsed or the source code refactored. The class Parser consists of the algorithm required for parsing the source code to prepare it for the Refactor class. The Token class consists of the result of the parser or the Refactor classes. The Grammar class consists of the various grammars necessary for the Parser class. The Refactor class consists of the algorithms required for performing the various refactoring transformations.

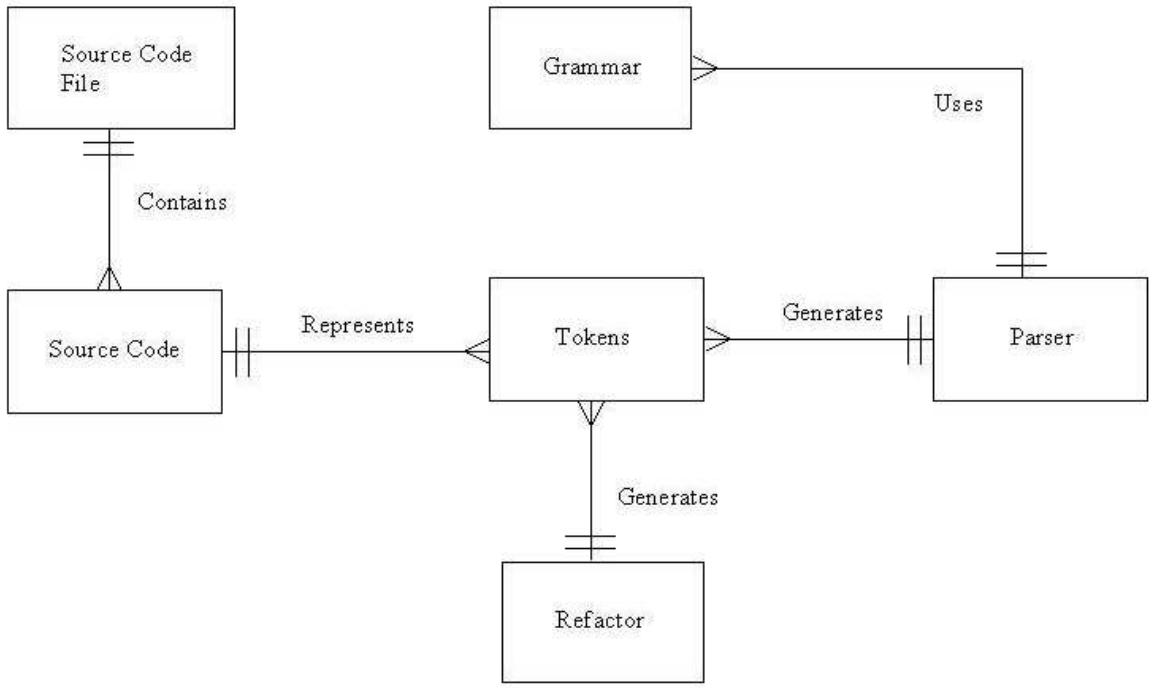


Figure 2.1.4.1 - ART Analysis Class Diagram

2.1.5 ART Entity-Relationship Diagram

Figure 2.1.5.1 defines the entity relationship diagram (ERD) for the ART system.

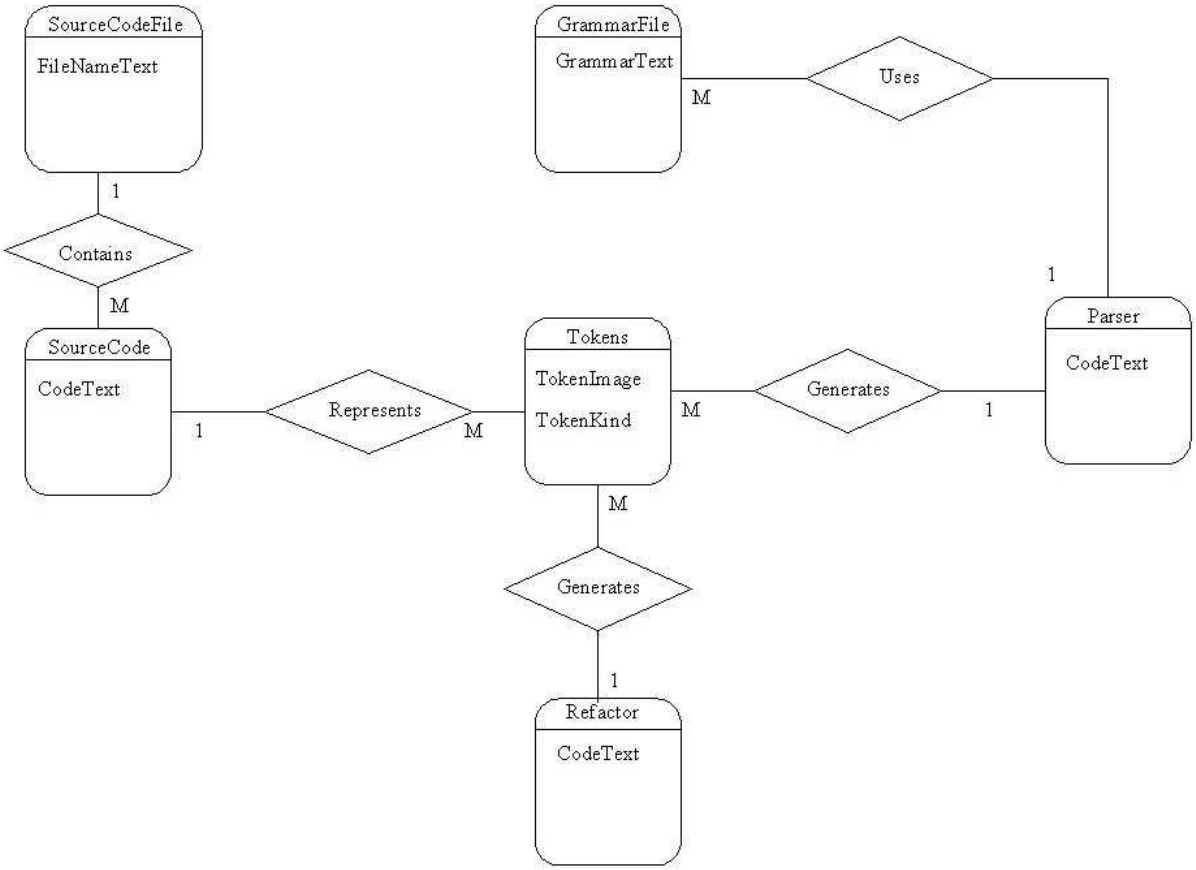


Figure 2.1.5.1 - ART ERD

2.1.5.1 ART ERD DeMarco Composition

SourceCode = CodeText

Token = TokenImage + TokenKind

SourceCodeFile = FileNameText

Grammar = GrammarText

Refactor = CodeText

Parser = CodeText

2.2 System modes and states

The ART system can be in the following states:

a) Read Source,
b) Parse,
c) Display Token,
d) Refactor, and

e) Redisplay Token.

The state diagram associated with the above mentioned states can be seen below.

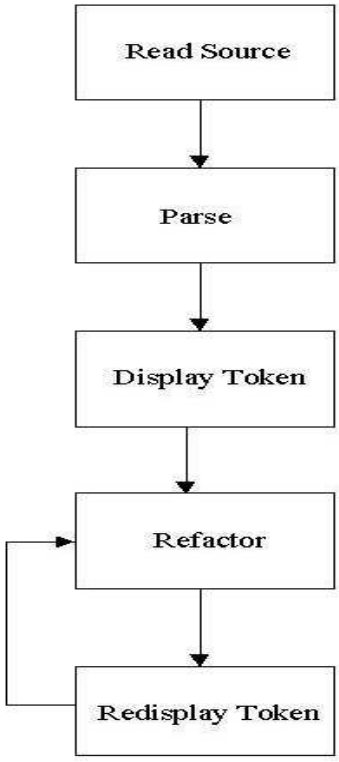


Figure 2.2.1 - ART State Diagram

2.3 Major system capabilities

Requirement Number	Description	Context Diagram Reference
2.3.1	The application shall contain a graphical user interface (GUI) subsystem adaptable to other languages.	1.0
2.3.2	The application shall contain a parser subsystem.	2.0
2.3.3	The application shall contain a refactor subsystem.	3.0

2.4 Major system conditions

Requirement Number	Description	Context Diagram Reference
2.4.1	The application shall produce tokens for the source code read using the syntax grammar.	2.3
2.4.2	The Parse Data option shall automatically parse the source file opened.	1.1.2.2
2.4.3	The Compile option shall compile the refactored code automatically.	1.1.4
2.4.4	The application shall perform the Encapsulate Field refactoring transformation.	3.1
2.4.5	The application shall perform the Replace Magic Number with Symbolic Constant refactoring transformation.	3.1
2.4.6	The refactor subsystem shall create tokens as a result of performing refactoring transformations.	3.1

2.5 Major system constraints

Requirement Number	Description	Context Diagram Reference
2.5.1	The application shall not allow parsing a file before the language syntax is generated.	2.3
2.5.2	The Parse Data option shall ensure that a source code file is opened.	2.3
2.5.3	The application shall store the tokens using the Vector class.	2.3
2.5.4	The application shall implement the Iterator pattern for accessing the tokens.	2.3
2.5.5	The application shall reformat the tokens using the Strategy pattern	3.1
2.5.6	The application shall implement the refactoring transformations using the Strategy pattern	3.1
2.5.7	The application shall create new tokens using the Abstract Factory Pattern.	3.1
2.5.8	The GUI shall be implemented using the Model-View Controller (MVC) pattern.	1.1
2.5.9	The Open option shall accept a user's specified or selected file to be opened.	1.1.1.1
2.5.10	The Close option shall close the file(s) opened.	1.1.1.2
2.5.11	The Save option shall permit the user to specify a file name before saving a file.	1.1.1.3
2.5.12	The Exit option shall terminate the application.	1.1.1.4

2.6 User characteristics

Not applicable.

2.7 Assumptions and dependencies

Requirement Number	Description	Context Diagram Reference
2.7.1	The application shall be written using the Java Coding Convention described in Annex B.	N/A
2.7.2	The application shall be written using the JBuilder 5.0 IDE.	N/A
2.7.3	The application shall be written using the Java SDK version 1.3.	N/A
2.7.4	The source code used for performing refactorings shall be error free.	2.1

2.8 Operational scenarios

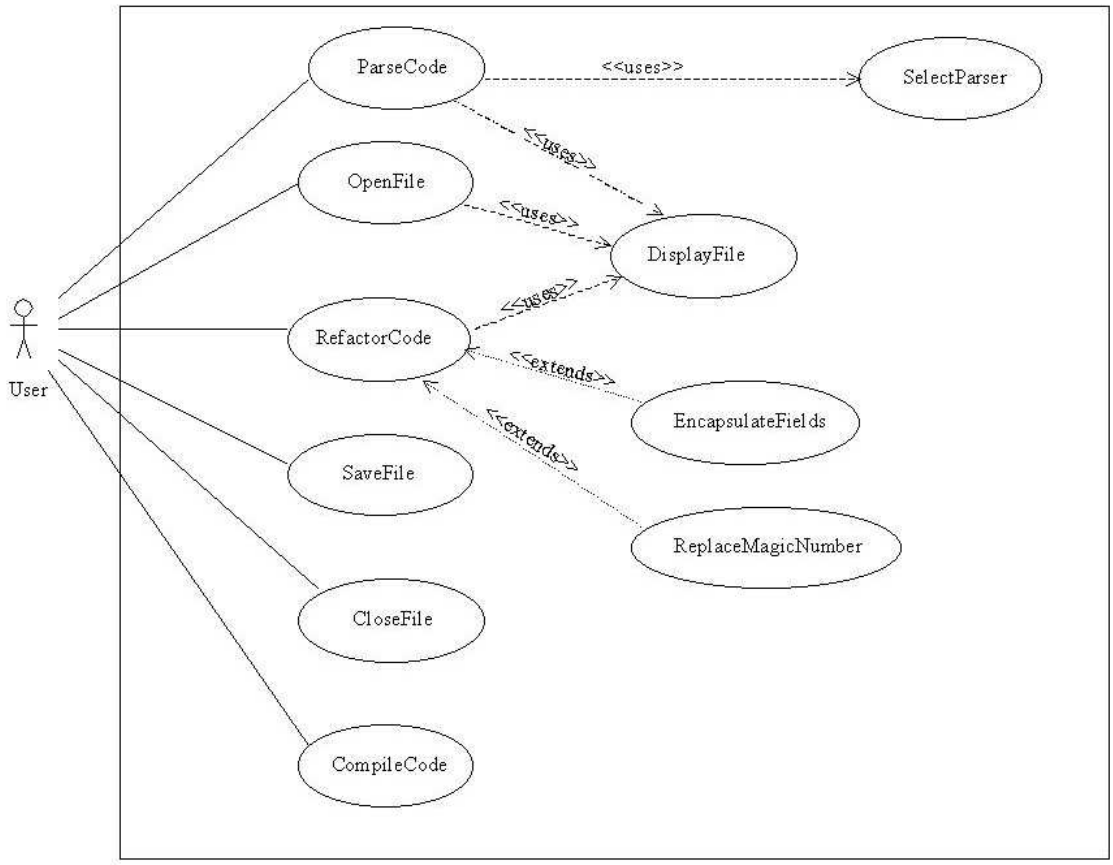


Figure 2.8 - ART Use Cases Diagram

Use Case: OpenFile

The user selects the Open File menu option from the window menu bar. A dialog box is displayed on the screen. The user selects the directory name where to find the file to be opened. The directory file list is updated. The user selects the file to be opened. The file is then opened and displayed to the user.

Use Case: ParseCode

The user selects the Parse Data menu option. A source code file needs to be opened and the language parser needs to be selected before the parser is executed. The parser takes the opened file and parses it according to the language specific rules determined by the selected language parser. The parser creates tokens representing the code read from the opened file. When the parsing is completed, the file is displayed to the user.

Use Case: RefactorCode

The user selects the Refactor menu option. The user then selects the refactoring transformation to be applied to the opened file. The refactoring transformations are not applied unless a file has been parsed previously. Following successful completion of the refactoring transformations, the refactored code is displayed to the user.

Use Case: SelectParser

The user selects the Select Language Parser menu option. A dialog box is displayed listing the available languages, which have a parser available. The user selects the language he desires and the associated parser will be used for parsing the opened file.

Use Case: DisplayFile

Following the opening, the parsing or the refactoring of a file, the resulting file content is read and displayed to the user.

Use Case: EncapsulateFields

The user selects the Encapsulate Fields menu option. A file needs to be open and parsed. The refactoring transformation Encapsulate Fields is applied to the opened and parsed file.

Use Case: ReplaceMagicNumber

The user selects the Replace Magic Number with Symbolic Constant menu option. A file needs to be open and parsed. The refactoring transformation Replace Magic Number with Symbolic Constant is applied to the opened and parsed file.

Use Case: SaveFile

The user selects the Save File menu option. . A dialog box is displayed on the screen. The user selects the directory name where to find the file to be opened. The directory file list is updated. The user selects the file to be opened. The file is then opened.

Use Case: CloseFile

The user selects the Close File menu option. All opened files are closed and the display areas are refreshed.

Use Case: CompileCode

The user selects the Compile Code menu option. The file recently refactored is compiled. Any errors are displayed to the user. Successful completion of the compiled is displayed to the user.

2.8.1 Use Case DeMarco Composition

CompileCode = CompileMenuItem + DisplayFile
OpenFile = OpenFileWindow + OpenMenuItem + DisplayFile
SaveFile = SaveFileWindow + SaveMenuItem + DisplayFile
CloseFile = CloseMenuItem + DisplayFile
RefactorCode = WindowRefactorMenu + EncapsulateFields + ReplaceMagicNumber + DisplayFile
ParseCode = ParseDataMenuItem + SelectParser + OpenFile + DisplayFile
SelectParser = SelectLanguageParserMenuItem + ParserFileName
EncapsulateFields = EncapsulateFieldsMenuItem + DisplayFile
ReplaceMagicNumber =ReplaceMagicNumberMenuItem + DisplayFile
DisplayFile = DisplayView

3.0 System capabilities, conditions, and constraints

3.1 Physical

3.1.1 Construction

Not applicable.

3.1.2 Durability

Not applicable.

3.1.3 Adaptability

Not applicable.

3.1.4 Environmental conditions

Not applicable.

3.2 System performance characteristics

Not applicable.

3.3 System security

Not applicable.

3.4 Information management

Not applicable.

3.5 System operations

3.5.1 System human factors

Not applicable.

3.5.2 System maintainability

Not applicable.

3.5.3 System reliability

Not applicable.

3.6 Policy and regulation

Not applicable.

3.7 System life cycle sustainment

Not applicable.

4.0 System Interfaces

4.1 System interfaces requirements

Requirement Number	Description	Context Diagram Reference
4.1.1	The application shall allow the user to select the file to be parsed.	1.1.1.1
4.1.2	The application shall allow the user to select the language parser using menu selection.	1.1.2.1
4.1.3	The GUI shall contain a menu bar.	1.1
4.1.4	The GUI shall contain two text display areas.	1.2 1.3

4.1.5	The GUI shall display the source code read in one text area.	1.2 1.1.1.5
4.1.6	The GUI shall display the refactored source code in one text area.	1.3
4.1.7	The GUI menu bar shall contain a File menu option.	1.1.1
4.1.8	The File menu option shall contain the following sub-menu options: Open Close Save Exit	 1.1.1.1 1.1.1.2 1.1.1.3 1.1.1.4
4.1.9	The GUI menu bar shall contain a Parse menu option.	1.1.2
4.1.10	The Parse menu option shall contain the following sub-menu options: Select Language Parser Parse Data	 1.1.2.1 1.1.2.2
4.1.11	The GUI menu bar shall contain a Compile menu option.	1.1.4
4.1.12	The GUI menu bar shall contain a Refactoring menu option.	1.1.3
4.1.13	The Refactoring menu option shall contain the following sub-menu option: Encapsulate Field Replace Magic Number with Symbolic Constant	 1.1.3.1 1.1.3.2
4.1.14	The GUI menu bar options shall be accessed using the Alt + key combination.	1.1
4.1.15	The GUI menu bar options and sub-options shall be accessible using a left mouse click.	1.1
4.1.16	The application GUI window shall be a standard Window environment.	1.0
4.1.17	The GUI window shall permit to be minimized, maximized and closed using buttons at the right hand corner.	1.0

4.2 Parser Package

To implement the parsing capability, the ART interfaces with a parser package. JavaCC2.1 was used to generate the parser package. JavaCC is available free of charge on the web, http://www.webgain.com. JavaCC is a tool that generates parsers based on a grammar supplied. It creates seven (7) Java source code files using the grammar for implementation. For ART, JavaCC was used to generate the Java source code files required for parsing Java programs using the grammar of version 1.2 of the Java SDK. An attempt was made to generate the Java source code files for C++ but some problems were encountered.

4.3 System Interfaces Forms

4.3.1 ART Main Window (ARTMainWindow)

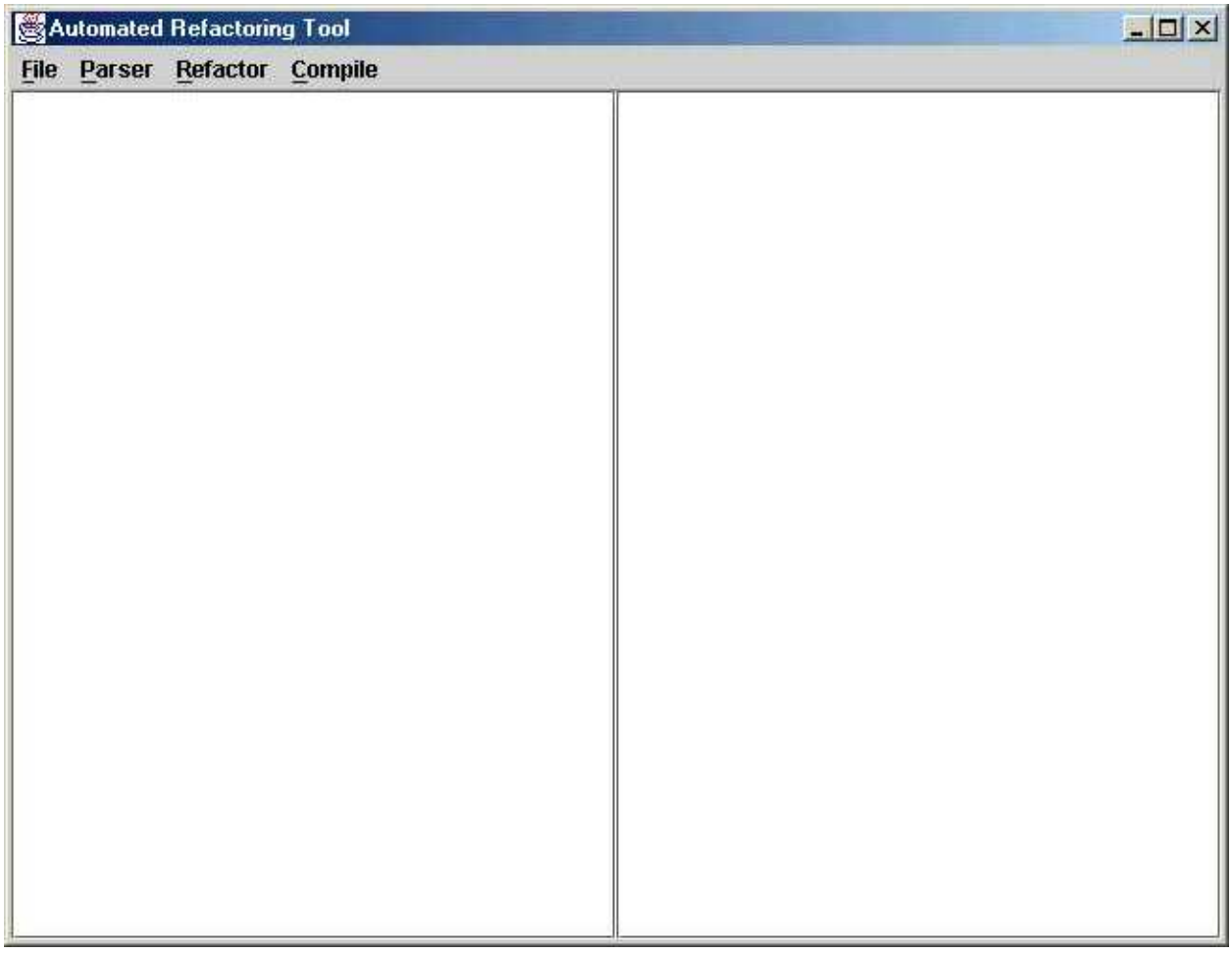


Figure 4.3.1.1 - ART Main Window

4.3.1.1 ART Main Window DeMarco Composition

ARTMainWindow = JFrame + WindowMenuBar + DisplayView
JFrame = Icon + JFrameTitle + JFrameButtons
JFrameButtons = MinimizeButton + MaximizeButton + CloseButton
WindowMenuBar = WindowMenu + WindowMenuCaption + WindowMenuMnemonic
WindowMenu = WindowFileMenu + WindowParseMenu + WindowRefactorMenu +
WindowCompileMenu
MinimizeButton = Button + MinimizeImage
MaximizeButton = Button + MaximizeImage
CloseButton = Button + CloseImage

4.3.2 File Menu Options (WindowFileMenu)

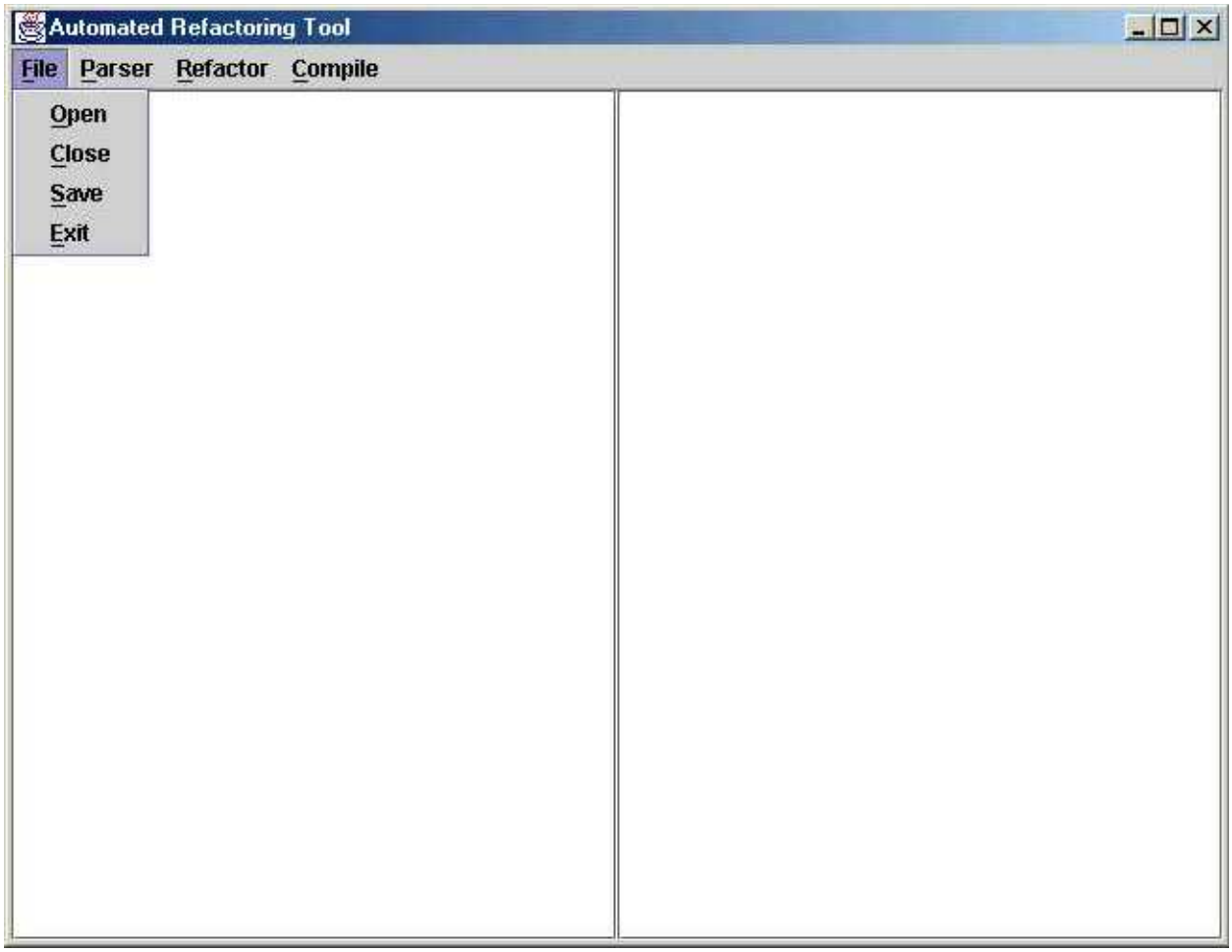


Figure 4.3.2.1 - File Menu Options

4.3.2.1 File Menu Options DeMarco Composition

WindowFileMenu = ARTMainWindow + OpenMenuItem + CloseMenuItem + SaveMenuItem + ExitMenuItem
OpenMenuItem = WindowMenuItem
CloseMenuItem = WindowMenuItem
SaveMenuItem = WindowMenuItem
ExitMenuItem = WindowMenuItem
WindowMenuItem = WindowMenuItemCaption + WindowMenuItemMnemonic

4.3.3 Parser Menu Options (WindowParseMenu)

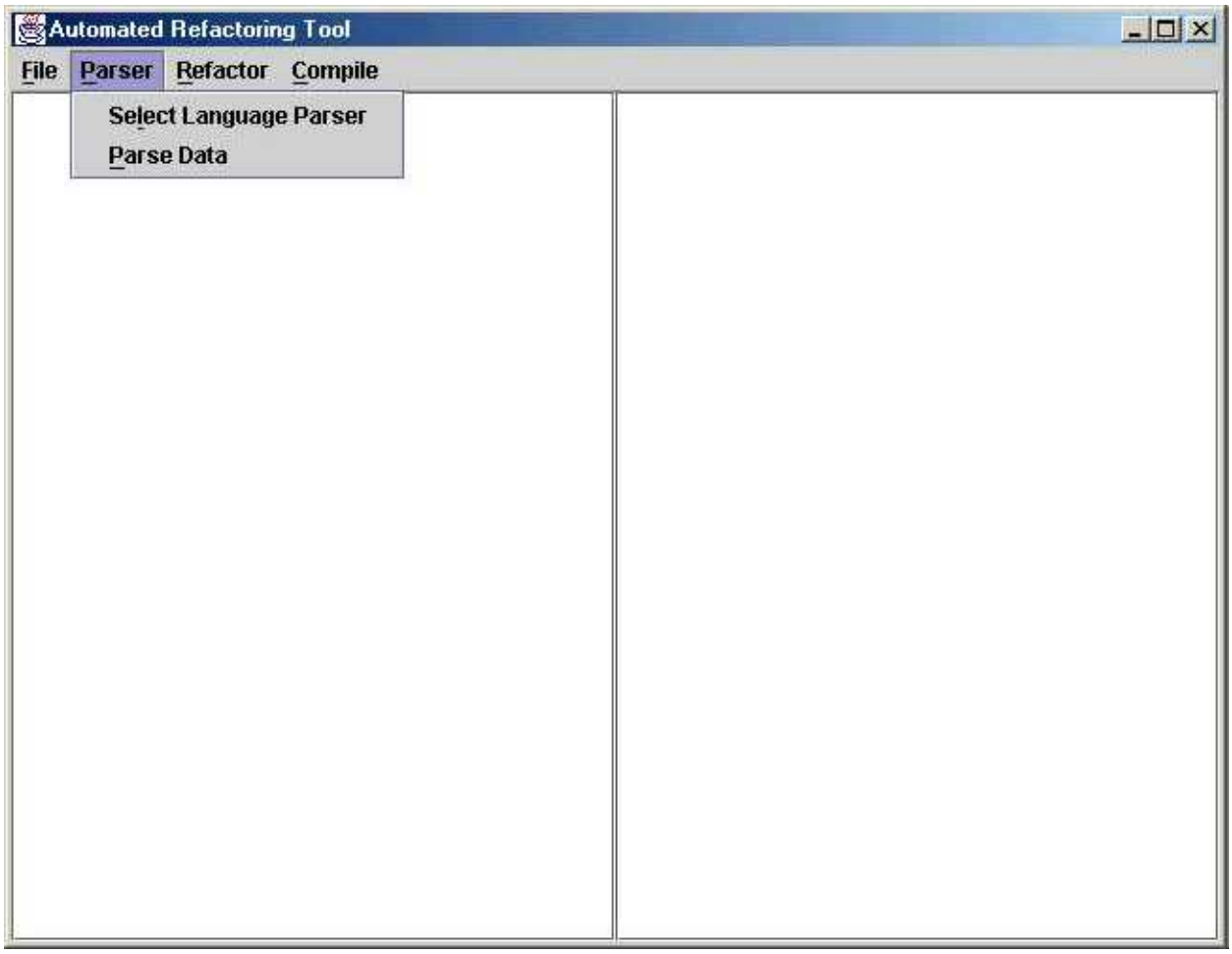


Figure 4.3.3.1- Parser Menu Options

4.3.3.1 Parser Menu Options DeMarco Composition

WindowParseMenu = ARTMainWindow + SelectLanguageParserMenuItem +
ParseDataMenuItem
SelectLanguageParserMenuItem = WindowMenuItem
ParseDataMenuItem = WindowMenuItem
WindowMenuItem = WindowMenuItemCaption + WindowMenuItemMnemonic

4.3.4 Refactor Menu Options (WindowRefactorMenu)

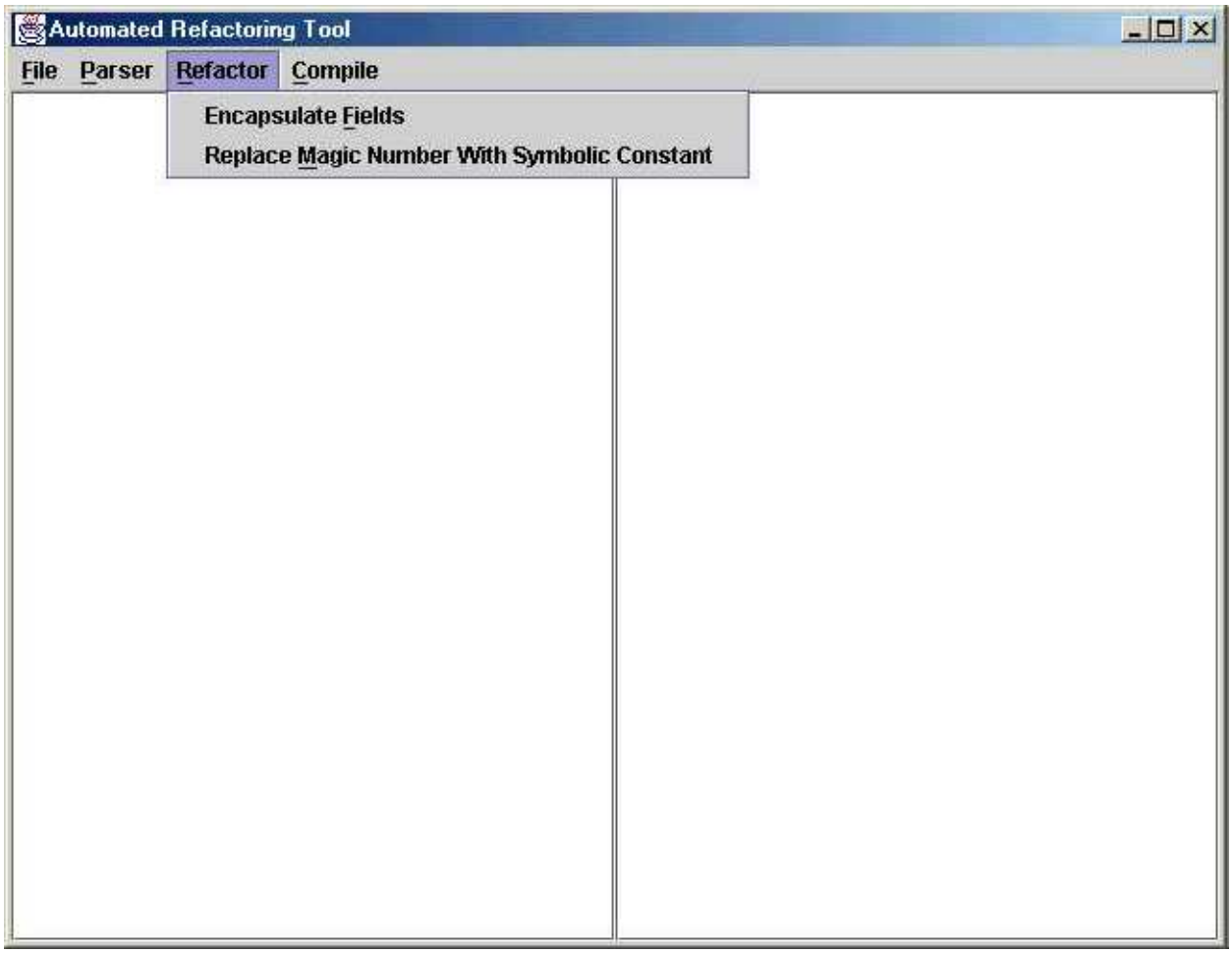


Figure 4.3.4.1 - Refactor Menu Options

4.3.4.1 Refactor Menu Options DeMarco Composition

WindowRefactorMenu = ARTMainWindow + ReplaceMagicNumberMenuItem +
EncapsulateFieldsMenuItem
ReplaceMagicNumberMenuItem = WindowMenuItem
EncapsulateFieldsMenuItem = WindowMenuItem
WindowMenuItem = WindowMenuItemCaption + WindowMenuItemMnemonic

4.3.5 Compile Menu Option

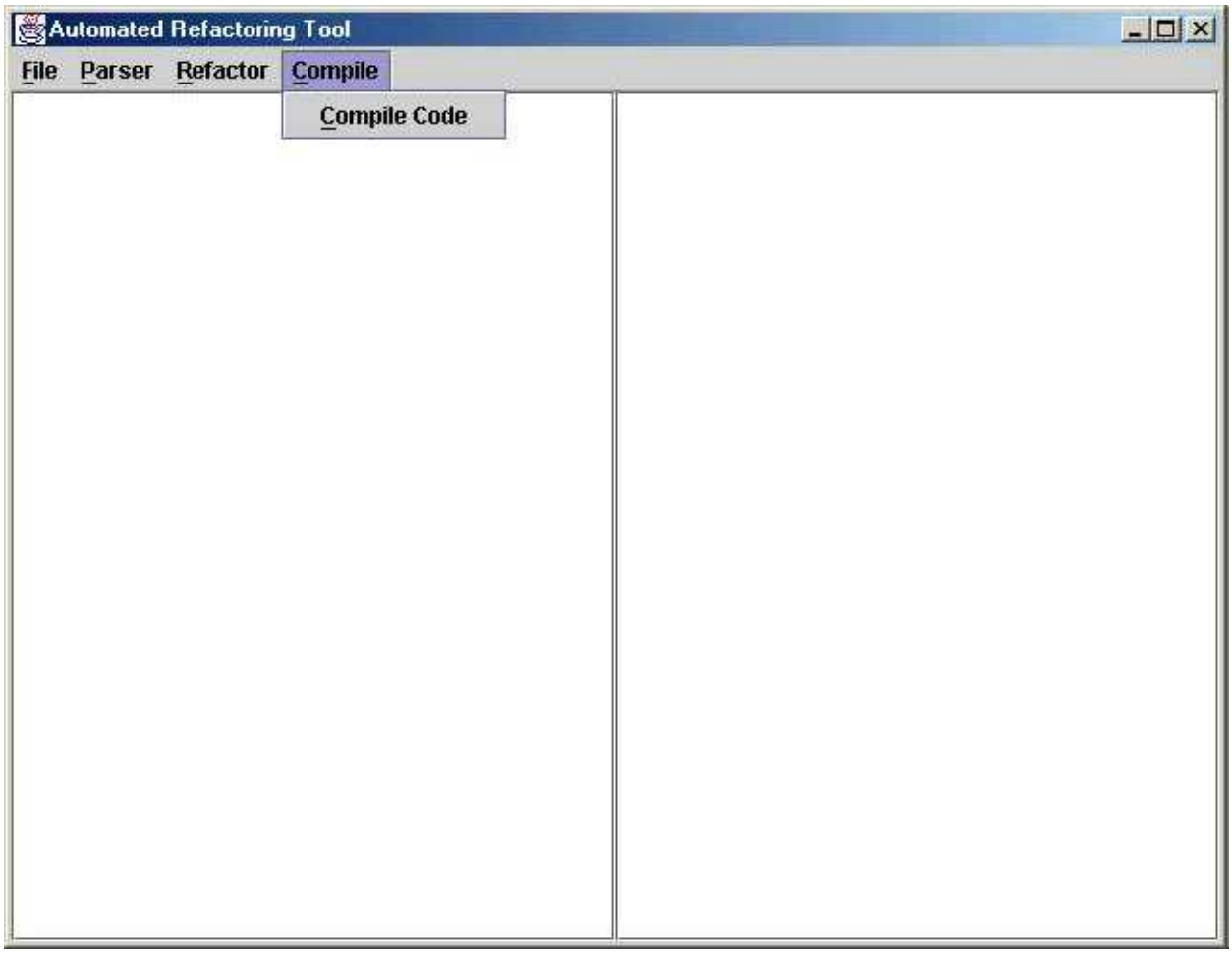


Figure 4.3.5.1 - Compile Menu Option

4.3.5.1 Compile Menu Options DeMarco Composition

WindowCompileMenu = ARTMainWindow + CompileMenuItem
CompileMenuItem = WindowMenuItem
WindowMenuItem = WindowMenuItemCaption + WindowMenuItemMnemonic

4.3.6 Display View form

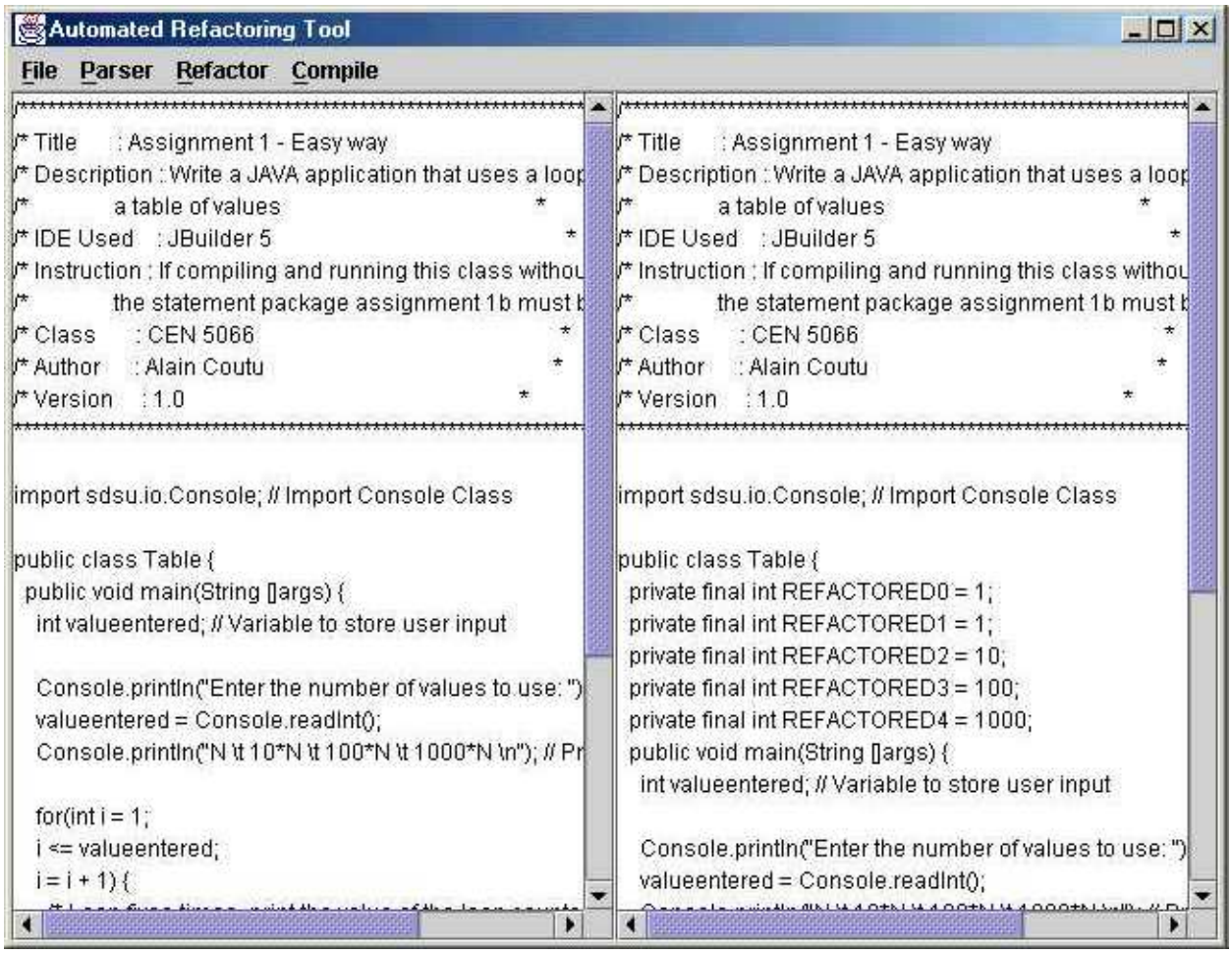


Figure 4.3.6.1 - Display View form

4.3.6.1 Display View DeMarco Composition

DisplayView = ARTMainWindow + SourceCodeView + RefactoredCodeView

4.3.7 Open File Dialog Window

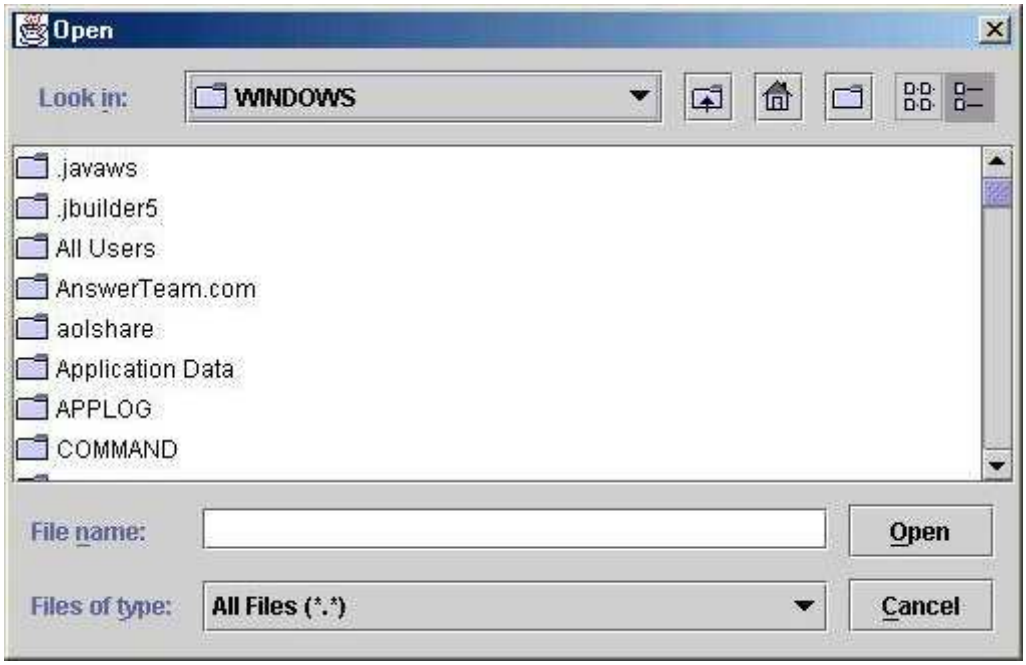


Figure 4.3.7.1- Open File Dialog Window

4.3.7.1 Open File Dialog Window DeMarco Composition

OpenFileWindow = JFrame + DirectoryLabel + DirectoryList + FolderUpButton +
HomeButton+ FolderlistButton + CreateFolderButton + ListButton
DetailsButton + OpenButton + CancelButton + ContentList +
FileNameLabel + FileNameText + FileTypeLabel + FileTypeList
JFrame = Icon + JFrameTitle + JFrameButtons
JFrameButton = CloseButton
DirectoryList = ScroolList + ItemsName
FileTypeList = ScroolList + ItemsName
FolderUpButton = Button + UpImage
HomeButton = Button + HomeImage
CreateFolderButton= Button + CreateImage
ContentList = ScroolList + ItemsName
ListButton = Button + ListImage
DetailsButton= Button + DetailsImage
OpenButton = Button + OpenImage
CancelButton = Button = CancelImage
CloseButton = Button + CancelImage

4.3.8 Save File Dialog Window

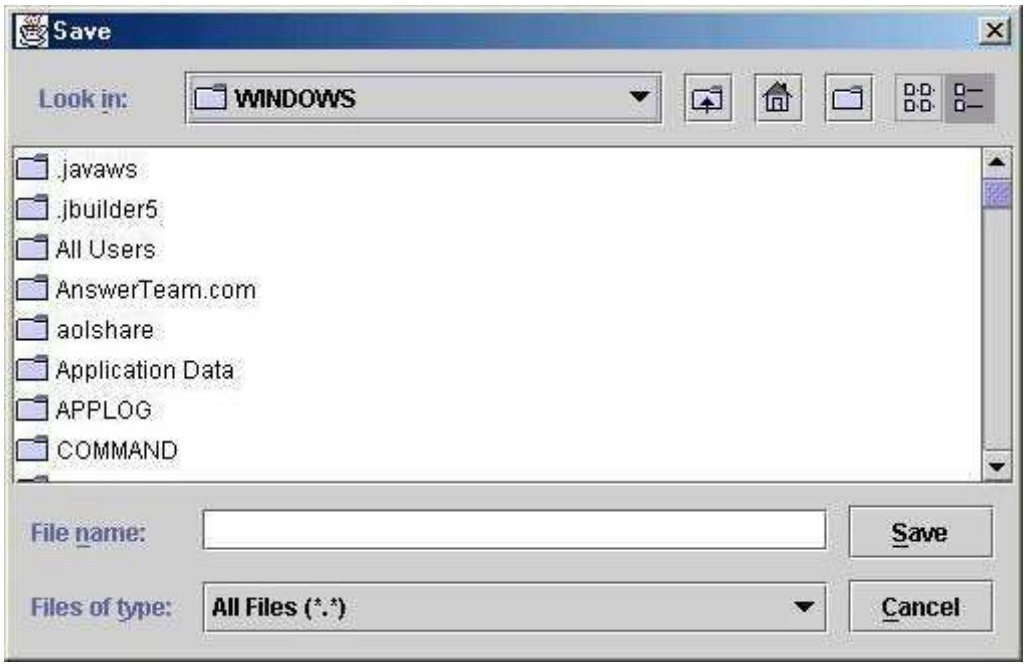


Figure 4.3.8.1 - Save File Dialog Window

4.3.8.1 Save File Dialog Window DeMarco Composition

SaveFileWindow = JFrame + DirectoryLabel + DirectoryList + FolderUpButton +
HomeButton+ FolderlistButton + CreateFolderButton + ListButton
DetailsButton + OpenButton + CancelButton + ContentList +
FileNameLabel + FileNameText + FileTypeLabel + FileTypeList
JFrame = Icon + JFrameTitle + JFrameButtons
JFrameButton = CloseButton
DirectoryList = ScroolList + ItemsName
FileTypeList = ScroolList + ItemsName
FolderUpButton = Button + UpImage
HomeButton = Button + HomeImage
CreateFolderButton= Button + CreateImage
ContentList = ScroolList + ItemsName
ListButton = Button + ListImage
DetailsButton= Button + DetailsImage
OpenButton = Button + OpenImage
CancelButton = Button = CancelImage
CloseButton = Button + CancelImage

4.4 ART Data Dictionary

Item Name	Type	Implementation Type	Description	Constraints	Environment	Security	Composition
ARTMainWindow	Screen	JFrame	Main application screen	None	Windows	None	JFrame + WindowMenuBar + DisplayView
Button	Button	JButton	Button	None	Windows	None	Button graphic
CancelButton	Composite	Button	Cancel button	None	Windows	None	Button + CancelImage
CancelImage	Image	Image	Cancel image	None	Windows	None	Graphical representation of cancel
CloseButton	Composite	JButton	Close button	None	Windows	None	Button + CloseImage
CloseFile	Use case	Process	Close file capability	None	Windows	None	CloseMenuItem + DisplayFile
CloseImage	Image	Image	Close image	None	Windows	None	Graphical representation of close
CloseMenuItem	Composite	JMenuItem	Close file menu option	None	Windows	None	WindowMenuItem
CodeText	String	Text	Code from the files	None	Windows	None	characters
CompileCode	Use Case	Process	Compile capability	None	Windows	None	CompileMenuItem + DisplayFile
CompileMenuItem	Composite	JMenuItem	Compile menu option	None	Windows	None	WindowMenuItem
ContentList	Composite	JList	List of files available	None	Windows	None	ScrollList + ItemsName
CreateFolderButton	Composite	JButton	Create folder button	None	Windows	None	Button + CreateFolderImage
CreateFolderImage	Image	Image	Create folder image	None	Windows	None	Graphical representation of create folder
DetailsButton	Composite	JButton	Display button	None	Windows	None	Button + DetailsImage
DetailsImage	Image	Image	Details image	None	Windows	None	Graphical representation of details
DirectoryLabel	String	Label	Title for directory list	None	Windows	None	"Look in:"
DirectoryList	Composite	JList	List of available directories	None	Windows	None	ScroolList + ItemsName
DisplayFile	Use Case	Process	File display capability	None	Windows	None	DisplayView
DisplayView	Composite	HorizontalBox	Display area for the code	None	Windows	None	ARTMainWindow + SourceCodeView + RefactoredView
EncapsulateFields	Use case	Process	Encapsulate Fields refactoring	None	Windows	None	EncapsulateFieldsMenuItem + DisplayFile
EncapsulateFieldsMenuItem	Composite	JMenuItem	Encapsulate fields menu option	None	Windows	None	WindowMenuItem
ExitMenuItem	Composite	JMenuItem	Exit menu option	None	Windows	None	WindowMenuItem
FileNameLabel	String	Label	Label for file name	None	Windows	None	["File name:"]
FileNameText	String	JTextField	File name	None	Windows	None	characters
FileTypeLabel	String	Label	Label for file type	None	Windows	None	[Files of type:"]
FileTypeList	Composite	JList	List of file names	None	Windows	None	ScroolList + ItemsName
FolderListButt	Composite	JButton	Folder list	None	Windows	None	Button +

on			button				FolderListImage
FolderListImage	Image	Image	Folder List Image	None	Windows	None	Graphical representation of folder list
FolderUpButton	Composite	JButton	Folder up button	None	Windows	None	Button + FolderUpImage
FolderUpImage	Image	Image	Folder up Image	None	Windows	None	Graphical representation of folder up
Grammar	Entity	File	Language grammar	None	Windows	None	GrammarText
GrammarText	String	Text	Grammar	None	Windows	None	characters
HomeButton	Composite	JButton	Home button	None	Windows	None	Button + HomeButtonImage
HomeButtonImage	Image	Image	Home image	None	Windows	None	Graphical representation of home
Icon	Image	Image	Frame icon	None	Windows	None	Image
ItemsName	String	Text	Items name	None	Windows	None	characters
JFrame	Composite	JFrame	Window container	None	Windows	None	Icon + JFrameTitle + JFrameButton
JFrameButton	Composite	Button	ART main window buttons	None	Windows	None	MinimizeButton + MaximizeButton + CloseButton
JFrameTitle	String	Text	Application title	None	Windows	None	characters
ListButton	Composite	JButton	List button	None	Windows	None	Button + ListImage
ListImage	Image	Image	Details image	None	Windows	None	Graphical representation of list
MaximizeButton	Composite	JButton	Maximize button	None	Windows	None	Button + MaximizeImage
MaximizeImage	Image	Image	Maximize image	None	Windows	None	Graphical representation of maximize
MinimizeButton	Composite	JButton	Minimize button	None	Windows	None	Button + MinimizeImage
MinimizeImage	Image	Image	Minimize image	None	Windows	None	Graphical representation of minimize
OpenButton	Composite	JButton	Open button	None	Windows	None	Button + OpenImage
OpenButtonImage	Image	Image	Open image	None	Windows	None	Graphical representation of open
OpenFile	Use Case	Process	Open file capability	None	Windows	None	OpenFileWindow + OpenMenuItem + DisplayFile
OpenFileWindow	Window	JFileChooser	Dialog box for opening file	None	Windows	None	JFrame + DirectoryLabel + DirectoryList + FolderUpButton + HomeButton+ FolderlistButton + CreateFolderButton + ListButton + DetailsButton +ContentList + FileNameLabel + FileNameText + OpenButton + FileTypeLabel + FileTypeList + CancelButton
OpenMenuItem	Composite	JMenuItem	Open file menu option	None	Windows	None	WindowMenuItem
ParseDataMenuItem	Composite	JMenuItem	Parse data menu option	None	Windows	None	WindowMenuItem

APPENDIX A

Parser	Entity	Java Files	Java Code	None	Windows	None	CodeText
ParserFileNam e	String	Text	Name of file to parse	None	Windows	None	characters
RefactorCode	Use Case	Process	Refactoring capability	None	Windows	None	WindowRefactorM enu + EncapsulateFields + ReplaceMagicNum ber + DisplayFile
RefactoredCo deView	String	JScrollPane	Source code read	None	Windows	None	characters
ReplaceMagic Number	Use Case	Process	Replace Magic Number refactoring	None	Windows	None	ReplaceMagicNum berMenuItem + DisplayFile
Refactor	Entity	Text	Refactored source code text	None	Windows	None	CodeText
ReplaceMagic NumberMenuI tem	Composite	JMenuItem	Replace magic number menu option	None	Windows	None	WindowMenuItem
SaveFile	Use Case	Process	Save File capability	None	Windows	None	SaveFileWindow + SaveMenuItem + DisplayFile
SaveFileWind ow	Window	JFileChooser	Dialog box for opening file	None	Windows	None	JFrame + DirectoryLabel + DirectoryList + FolderUpButton + HomeButton+ FolderlistButton + CreateFolderButton + ListButton + DetailsButton +ContentList + FileNameLabel + FileNameText + OpenButton + FileTypeLabel + FileTypeList + CancelButton
SaveMenuIte m	Composite	JMenuItem	Save file menu option	None	Windows	None	WindowMenuItem
Scroollist	String	JList	Scrollable list	None	Windows	None	List
SelectLanguag eMenuItem	Composite	JMenuItem	Select language parser menu option	None	Windows	None	WindowMenuItem
SelectParser	Use Case	Process	Select language parser capability	None	Windows	None	SelectLanguagePars erMenuItem + ParserFileName
SourceCode	Entity	Text	Source code text	None	Windows	None	CodeText
SourceCodeFil e	Entity	File	Source code file	None	Windows	None	FileNameText
SourceCodeVi ew	String	JScrollPane	Source code read	None	Windows	None	characters
Token	Entity	Vector	Code representation	None	Windows	None	TokenImage + TokenKind
TokenImage	String	Text	Code	None	Windows	None	characters
TokenKind	String	Integer	Type of code	None	Windows	None	integer
WindowComp ileMenu	Composite	JMenu	Compille Menu option	None	Windows	None	ARTMainWindow + CompileMenuItem
WindowFileM enu	Composite	JMenu	File Menu option	None	Windows	None	ARTMainWindow + OpenMenuItem + CloseMenuItem + SaveMenuItem + ExitMenuItem

WindowMenu	Composite	JMenu	Menu for the menu bar	None	Windows	None	WindowFileMenu + WindowParseMenu + WindowRefactorMenu + WindowCompileMenu
WindowMenuBar	Composite	JMenuBar	Menu bar for the main window	None	Windows	None	WindowMenu + windowMenuCaption + WindowMenuMnemonic
WindowMenuCaption	String	String	Name of the menu option	None	Windows	None	characters
WindowMenuItem	Composite	JMenuItem	Menu item	None	Windows	None	WindowMenuItemCaption + WindowMenuItemMnemonic
WindowMenuItemCaption	String	String	Name of the menu item	None	Windows	None	characters
WindowMenuItemMnemonic	String	Character	Menu item shortcut menu	None	Windows	None	[a..z / A..Z]
WindowMenuMnemonic	String	Character	Menu shortcut letter	None	Windows	None	[a..z / A..Z]
WindowParseMenu	Composite	JMenu	Parse Menu option	None	Windows	None	ARTMainWindow + SelectLanguageParserMenuItem + ParseDataMenuItem
WindowRefactorMenu	Composite	JMenu	RefactorMenu option	None	Windows	None	ARTMainWindow + ReplacemagicNumberMenuItem + EncapsulateFieldsMenuItem

APPENDIX A

AUTOMATED
REFACTORING
TOOL (ART)

APPENDIX B

SOFTWARE DESIGN SPECIFICATIONS (SDS)

APPENDIX B

TABLE OF CONTENTS

1.0 INTRODUCTION 1

1.1 Purpose 1
1.2 Scope 1
1.3 Definitions and Acronyms 1
1.4 References 1

2.0 DECOMPOSITION DESCRIPTION 2

2.1 GUI Components - layouts, navigation 2
2.1.1 ART Main Window 2
2.1.2 File Menu 3
2.1.3 Parser Menu 4
2.1.4 Refactor Menu 5
2.1.5 Compile Menu 6
2.1.6 Open Window 7
2.1.7 Save Window 8
2.1.8 Display View 9
2.2 Data Dictionary for GUI Components 9

3.0 DECOMPOSITION DESCRIPTION 13

3.1 Preliminary Design Description 13
..... 13
3.2 Sequence Diagrams 14
3.2.1 ParseCode 14
3.2.2 RefactorCode 14
3.2.3 DisplayFile 15
3.2.4 EncapsulateFields, ReplaceMagicNumber, other refactoring transformations* 16
3.3 Decomposition Data Dictionary 16

4.0 COMPONENT DETAIL DESIGN DESCRIPTION 28

4.1 AbstractToken Class 28
4.1.1 Structure 28
4.1.2 Function 29
4.1.3 Interfaces 29
4.1.4 Application Data 29
4.1.5 Detailed Design Description 30
4.1.5.1 setToken Method 30
4.1.5.2 getToken Method 30
4.1.5.3 getImage Method 30
4.1.5.4 getKind Method 30
4.1.5.5 setImage Method 30
4.1.5.6 setKind Method 30
4.1.5.7 isPublic Method 30
4.1.5.8 isClass Method 31
4.1.5.9 isVoid Method 31
4.1.5.10 isAbstract Method 31
4.1.5.11 isLeftParen Method 31
4.1.5.12 isAssign Method 31
4.1.5.13 isGreaterThan Method 31
4.1.5.14 isLessThan Method 31
4.1.5.15 isEqual Method 31
4.1.5.16 isNotEqual Method 32
4.1.5.17 isGreaterEqual Method 32

4.1.5.18 isLessEqual Method 32
4.1.5.19 isIncrement Method 32
4.1.5.20 isDecrement Method 32
4.1.5.21 isPlus Method 32
4.1.5.22 isMinus Method 32
4.1.5.23 isStar Method 32
4.1.5.24 isSlash Method 33
4.1.5.25 isPlusAssign Method 33
4.1.5.26 isMinusAssign Method 33
4.1.5.27 isStarAssign Method 33
4.1.5.28 isSlashAssign Method 33
4.1.5.29 isLeftBrace Method 33
4.1.5.30 isSemiColon Method 33
4.1.5.31 isANumber Method 34
4.1.5.32 isAVariableDeclaration Method 34
4.1.5.33 isArithmeticOperator Method 34
4.1.5.34 isAssignmentOperator Method 34
4.1.5.35 isOperatorFollowedByNumber Method 34
4.1.5.36 isNotAbstractClassVoid Method 34
4.1.5.37 isFirstLeftBrace Method 34
4.1.5.38 setTokenCharacter Method 34
4.2 EncapsulateFields Class 35
4.2.1 Structure 35
4.2.2 Function 35
4.2.3 Interfaces 35
4.2.4 Application Data 35
4.2.5 Detailed Design Description 35
4.2.5.1 execute Method 35
4.2 Indent Class 36
4.3.1 Structure 36
4.3.2 Function 36
4.3.3 Interfaces 36
4.3.4 Application Data 36
4.3.5 Detailed Design Description 36
4.3.5.1 Indent Constructor 36
4.3.5.2 setIndentLevelValue Method 36
4.3.5.3 getIndentLevelValue Method 37
4.3.5.4 decreaseIndentLevel Method 37
4.3.5.5 increaseIndentLevelValue Method 37
4.3.5.6 setStringValue Method 37
4.3.5.7 increaseIndentStringValue Method 37
4.3.5.8 setLoopCounter Method 37
4.3.5.9 loopNotDone Method 37
4.3.5.10 setIndentStringValue Method 38
4.3.5.11 getIndentStringValue method 38
4.3 JavaDeParser Class 38
4.4 Structure 38
4.5 Function 38
4.6 Interfaces 38
4.7 Application Data 38
4.8 Detailed Design Description 38
4.9 JavaDeParser Constructor 38
4.9.5.2 deparseToken Method 39
4.10 ObjectCollection Class 39
4.5.1 Structure 39
4.5.2 Function 39

4.5.3 Interfaces 39
4.5.4 Application Data 39
4.5.5 Detailed Design Description 39
4.5.5.1 createCollectionIterator Method 39
4.5.5.2 add Method 39
4.5.5.3 add Method 40
4.5.5.4 removeElement method 40
4.6 ObjectIterator Class 40
4.6.1 Structure 40
4.6.2 Function 40
4.6.3 Interfaces 40
4.6.4 Application Data 40
4.6.5 Detailed Design Description 41
4.6.5.1 nextObject Method 41
4.6.5.2 add Method 41
4.6.5.3 isThereMoreObjects Method 41
4.6.5.4 setToFirst Method 41
4.6.5.5 setToLast Method 41
4.6.5.6 setVectorLocation Method 41
4.6.5.7 getCurrentCollectionIndex Method 41
4.6.5.8 insertTokens Method 41
4.7 ProjectFile Class 42
4.7.1 Structure 42
4.7.2 Function 42
4.7.3 Interfaces 42
4.7.4 Application Data 42
4.7.5 Detailed Design Description 42
4.7.5.1 readFile Method 42
4.7.5.2 saveFile Method 43
4.7.5.3 compileFileMethod 43
4.7.5.4 getFileName Method 43
4.7.5.5 setFileName Method 43
4.8 RefactoringContext Class 43
4.8.1 Structure 43
4.8.2 Function 43
4.8.3 Interfaces 44
4.8.4 Application Data 44
4.8.5 Detailed Design Description 44
4.8.5.1 setRefactoringStrategy Method 44
4.8.5.2 execute Method 44
4.9 RefactoringStrategy Class 44
4.9.1 Structure 44
4.9.2 Function 44
4.9.3 Interfaces 45
4.9.4 Application Data 45
4.9.5 Detailed Design Description 45
4.9.5.1 RefactoringStrategy Constructor 45
4.9.5.2 execute Method 46
4.9.5.3 setVariableIndexValue Method 46
4.9.5.4 getVariableIndexValue Method 46
4.9.5.5 setFirstLeftBrace Method 46
4.9.5.6 getFirstLeftBrace Method 46
4.9.5.7 setInsertLocation Method 46
4.9.5.8 setRefactoredVariableName Method 46
4.9.5.9 getRefactoredVariableName Method 47
4.9.5.10 setVariableImageRequired Method 47

4.9.5.11 getVariableImageRequired Method 47
4.9.5.12 setVariableRequiredKind Method 47
4.9.5.13 saveVariableImageRequired Method 47
4.9.5.14 getVariableRequiredKind Method 47
4.9.5.15 isTokenWanted Method 47
4.9.5.16 findSetFirstVariableOccurence Method 47
4.9.5.17 setMagicNumberTokenVector Method 48
4.9.5.18 getInsertTokenVector Method 48
4.9.5.19 createMethodName Method 48
4.9.5.20 getMethodName Method 48
4.9.5.21 getInsertCodeCollectionIterator Method 48
4.9.5.22 createSetMethodTokenVector Method 48
4.9.5.23 createGetMethodTokenVector Method 49
4.10 RefactoringTool Class 49
4.10.1 Structure 49
4.10.2 Function 49
4.10.3 Interfaces 49
4.10.4 Application Data 50
4.10.5 Detailed Design Description 50
4.10.5.1 RefactoringTool Constructor 50
4.10.5.2 setOpenFileMenuItem Method 50
4.10.5.3 setSaveFileMenuItem Method 51
4.10.5.4 setCloseFileMenuItem Method 51
4.10.5.5 setExitMenuItem Method 51
4.10.5.6 setCompileMenuItem Method 51
4.10.5.7 setSelectLanguageParserMenuItem Method 51
4.10.5.8 setParseDataMenuItem Method 52
4.10.5.9 setEncapsulateFieldsMenuItem Method 52
4.10.5.10 setReplaceMagicNumberWithSymbolicConstantMenuItem Method 52
4.10.5.11 main Method 52
4.11 ReplaceMagicNumber Class 53
4.11.1 Structure 53
4.11.2 Function 53
4.11.3 Interfaces 53
4.11.4 Application Data 53
4.11.5 Detailed Design Description 53
4.11.5.1 execute Method 53
4.12 SourceCodeView Class 53
4.12.1 Structure 53
4.12.2 Function 54
4.12.3 Interfaces 54
4.12.4 Application Data 54
4.12.5 Detailed Design Description 54
4.12.5.1 SourceCodeView Constructor 54
4.12.5.2 displayText Method 54
4.12.5.3 clearDisplayMethod 54
4.13 TokenCollection Class 54
4.13.1 Structure 54
4.13.2 Function 55
4.13.3 Interfaces 55
4.13.4 Application Data 55
4.13.5 Detailed Design Description 55
4.13.5.1 TokenCollection Constructor 55
4.13.5.2 createCollectionIterator Method 55
4.13.5.3 add Method 55
4.13.5.4 add Method 55

4.13.5.5 removeElement Method 55
4.13.5.6 get Method 56
4.13.5.7 setCollection Method Method 56
4.14 TokenContext Class 56
4.14.1 Structure 56
4.14.2 Function 56
4.14.3 Interfaces 56
4.14.4 Application Data 56
4.14.5 Detailed Design Description 56
4.14.5.1 TokenContext Constructor 56
4.14.5.2 setReformatStrategy Method 56
4.14.5.3 reformat Method 57
4.15 TokenCreator Class 57
4.15.1 Structure 57
4.15.2 Function 57
4.15.3 Interfaces 57
4.15.4 Application Data 57
4.15.5 Detailed Design Description 57
4.15.5.1 createToken Method 57
4.15.5.2 getFactory Method 58
4.16 TokenDefault Class 58
4.16.1 Structure 58
4.16.2 Function 58
4.16.3 Interfaces 58
4.16.4 Application Data 58
4.16.5 Detailed Design Description 58
4.16.5.1 reformat Method 58
4.17 TokenFactory Class 59
4.17.1 Structure 59
4.17.2 Function 59
4.17.3 Interfaces 59
4.17.4 Application Data 59
4.17.5 Detailed DesignDescription 59
4.17.5.1 createToken Method 59
4.18 TokenIterator Class 59
4.18.1 Structure 59
4.18.2 Function 59
4.18.3 Interfaces 60
4.18.4 Application Data 60
4.18.5 Detailed Design Description 60
4.18.5.1 TokenIterator Constructor 60
4.18.5.2 setToFirst Method 60
4.18.5.3 setToLast Method 60
4.18.5.4 setVectorLocation Method 60
4.18.5.5 getCurrentCollectionIndex Method 61
4.18.5.6 insertTokens Method 61
4.18.5.7 add Method 61
4.18.5.8 add Method 61
4.18.5.9 nextObject Method 61
4.18.5.10 isThereMoreObjects Method 61
4.18.5.11 setCurrentVectorLocation Method 61
4.18.5.12 getCurrentVectorLocation Method 62
4.18.5.13 setInsertVectorLocation Method 62
4.18.5.14 getInsertVectorLocation Method 62
4.19 TokenLBrace Class 62
4.19.1 Structure 62

4.19.2 Function 62
4.19.3 Interfaces 62
4.19.4 Application Data 62
4.19.5 Detailed Design Description 62
4.19.5.1 reformat Method 62
4.20 TokenMultiLineComment Class 63
4.20.1 Structure 63
4.20.2 Function 63
4.20.3 Interfaces 63
4.20.4 Application Data 63
4.20.5 Detailed Design Description 63
4.20.5.1 reformat Method 63
4.21 TokenParenBracketDotBang Class 63
4.21.1 Structure 63
4.21.2 Function 64
4.21.3 Interfaces 64
4.21.4 Application Data 64
4.21.5 Detailed Design Description 64
4.21.5.1 reformat Method 64
4.22 TokenRBrace Class 64
4.22.1 Structure 64
4.22.2 Function 64
4.22.3 Interfaces 64
4.22.4 Application Data 64
4.22.5 Detailed Design Description 65
4.22.5.1 reformat Method 65
4.23 TokenReformat Class 65
4.23.1 Structure 65
4.23.2 Function 65
4.23.3 Interfaces 65
4.23.4 Application Data 65
4.23.5 Detailed Design Description 65
4.23.5.1 reformat Method 65
4.24 TokenSemiColon Class 66
4.24.1 Structure 66
4.24.2 Function 66
4.24.3 Interfaces 66
4.24.4 Application Data 66
4.24.5 Detailed Design Description 66
4.24.5.1 reformat Method 66
4.25 TokenSingleLineComment Class 66
4.25.1 Structure 66
4.25.2 Function 66
4.25.3 Interfaces 67
4.25.4 Application Data 67
4.25.5 Detailed Design Description 67
4.25.5.1 reformat Method 67
4.26 WindowMenu Class 67
4.26.1 Structure 67
4.26.2 Function 67
4.26.3 Interfaces 67
4.26.4 Application Data 67
4.26.5 Detailed Design Description 67
4.26.5.1 WindowMenu Constructor 67
4.27 WindowMenuBarView Class 68
4.27.1 Structure 68

4.27.2 Function 68
4.27.3 Interfaces 68
4.27.4 Application Data 68
4.27.5 Detailed Design Description 68
4.27.5.1 WindowMenuBarView Constructor 68
4.27.5.2 getExitMenuItem Method 68
4.27.5.3 getCompileMenuItem Method 69
4.27.5.4 getOpenFileMenuItem Method 69
4.27.5.5 getCloseFileMenuItem Method 69
4.27.5.6 getSaveFileMenuItem Method 69
4.27.5.7 getSelectLanguageParserMenuItem Method 69
4.27.5.8 getParseDataMenuItem Method 69
4.27.5.9 getEncapsulateFieldsMenuItem Method 69
4.27.5.10 getReplaceMagicNumberWithSymbolicConstantMenuItem Method 70
4.28 WindowMenuItem Class 70
4.28.1 Structure 70
4.28.2 Function 70
4.28.3 Interfaces 70
4.28.4 Application Data 70
4.28.5 Detailed Design Description 70
4.28.5.1 WindowMenuItem Constructor 70

5.0 QUALITY ASSURANCE 70

5.1 Software Test Plan 70
5.2 Software Test Procedures 70
5.3 Test Cases 71
5.4 Test Cross reference Matrix 71

6.0 TRACEABILITY MATRIX LINKING SHALL LIST TO METHODS, CLASSES, USE CASES 71

APPENDIX B

LIST OF FIGURES

Figure 2.1.2.1 - File Menu window 3
Figure 2.1.2.2 - File Menu window navigation table 3
Figure 2.1.3.1 - Parser Menu window 4
Figure 2.1.3.2 - Parser Menu window navigation table 4
Figure 2.1.4.1 Refactor Menu window 5
Figure 2.1.4.2 - Refactor Menu window navigation table 6
Figure 2.1.5.1 - Compile Menu window 6
Figure 2.1.5.2 - Compile Menu window navigation table 6
Figure 2.1.6.1 - Open window 7
Figure 2.1.6.2 - Open window navigation table 7
Figure 2.1.7.1 - Save window 8
Figure 2.1.7.2 - Save window navigation table 8
Figure 2.1.8.1 - Display View window 9
Figure 3.2.2.1 - Refactor Code sequence diagram 14
Figure 3.2.3.1 - Display File sequence diagram 15
Figure 3.2.4.1 - EncapsulateFields, ReplaceMagicNumber, other refactoring transformations sequence diagram 16

1.0 Introduction

1.1 Purpose

The purpose of this document is to define the design of the Automated Refactoring Tool (ART). The goal of ART is to permit software engineers the capability to perform refactoring transformation on object-oriented source code. The main objective of ART is the partial automation of the refactoring transformations. In addition, ART permits the compilation of refactored Java code without terminating the tool execution. The objectives of ART are to reduce development time, increase the code reusability and increase the code flexibility.

1.2 Scope

The scope of ART includes a graphical user interface (GUI), a parser, a refactor and a framework for the incremental development of refactoring transformations. ART will be developed using the coding conventions defined in Appendix B, Coding Conventions, (SRS Requirement # 2.7.1). The code of ART will be written using the JBuilder 5.0 IDE and the Java SDK version 1.3, (SRS Requirements # 2.7.2 and 2.7.3). ART will parse a source code file based on a language grammar and perform refactoring transformations on the parsed code. ART also will permit the compilation of the refactored code without terminating the application and will be error free, (SRS Requirement # 2.7.4).

1.3 Definitions and Acronyms

ART	Automated Refactoring Tool
GUI	Graphical User Interface
IEEE	Institute of Electrical and Electronics Engineers
SDK	Software Development Kit

1.4 References

NUMBER	TITLE	DATE
SRS	Software Requirement Specifications (SRS) - Appendix A to ART Project Report	2003
IEEE Std 1016.1	IEEE Guide to Software Design Descriptions	1993
IEEE Std 1016	IEEE Recommended Practice for Software Design Descriptions	1998

2.0 Decomposition Description

2.1 GUI Components - layouts, navigation

2.1.1 ART Main Window

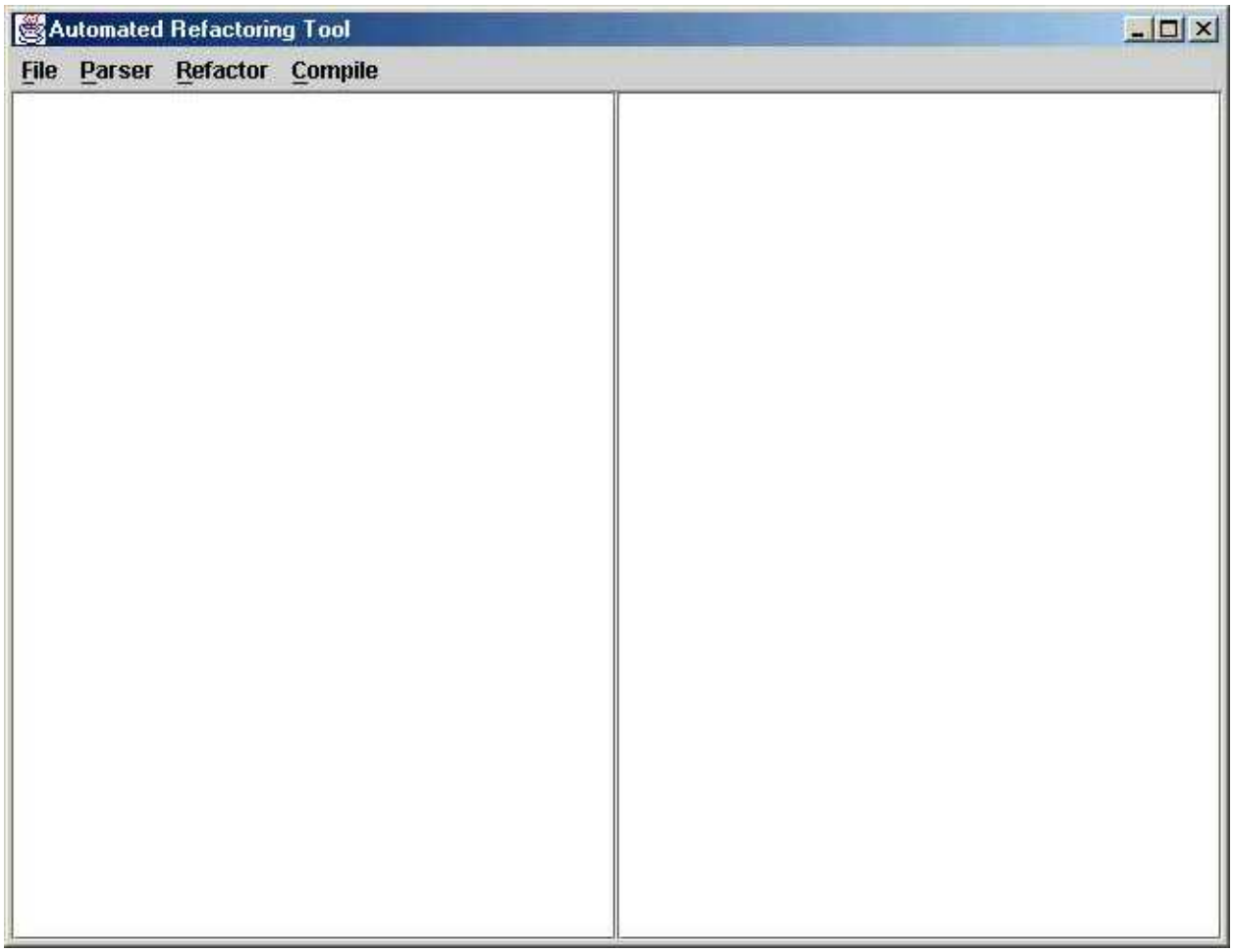


Figure 2.1.1.1 - ART main window

NAVIGATION TABLE		
2.2.1	**State**	**Resulting State**
	Static Display	
File		File Menu
Parse		Parse Menu
Refactor		Refactor Menu
Compile		Compile Menu

Figure 2.1.1.2 - ART main window navigation table

2.2.2 2.1.2 File Menu

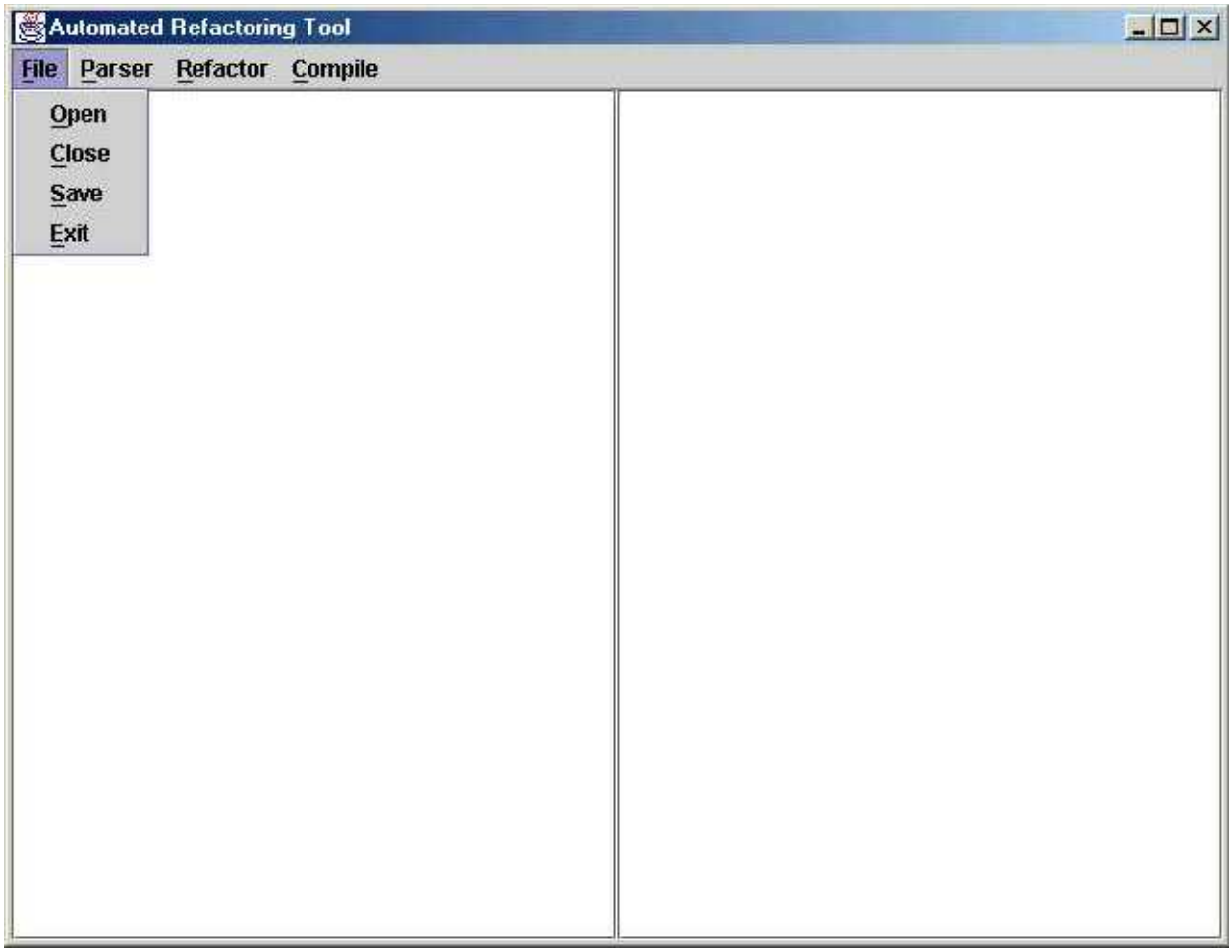


Figure 2.1.2.1 - File Menu window

NAVIGATION TABLE		
2.2.3	**State**	**Resulting State**
	File Menu	
Open		Open Window
Close		File closed and display cleared
Save		Save Window
Exit		Application Terminated

Figure 2.1.2.2 - File Menu window navigation table

2.2.4 2.1.3 Parser Menu

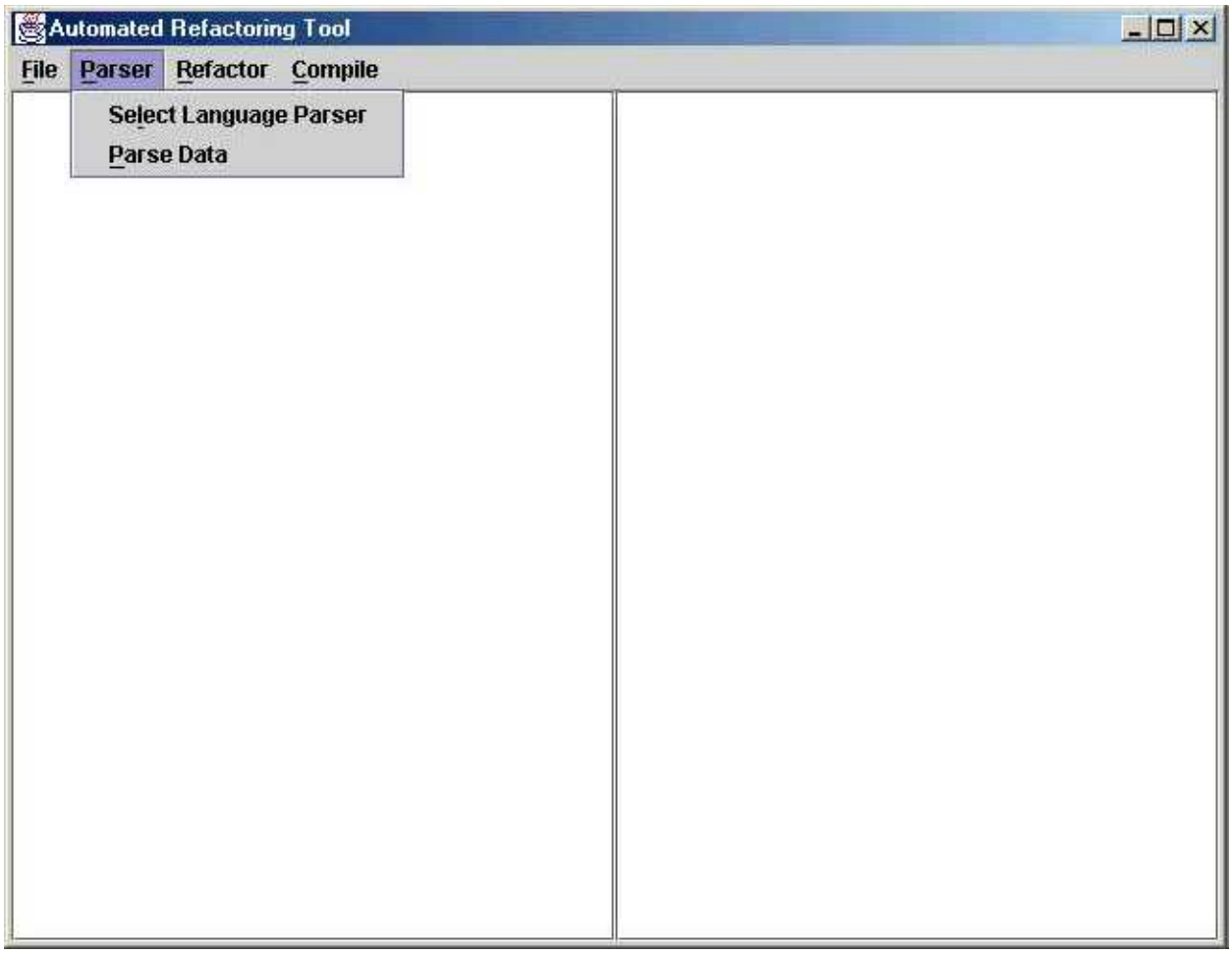


Figure 2.1.3.1 - Parser Menu window

NAVIGATION TABLE		
2.2.5	**State**	**Resulting State**
	Parser Menu	
Select Language Parser		Language Parser Selected
Parse Data		Code Parsed

Figure 2.1.3.2 - Parser Menu window navigation table

2.2.6 2.1.4 Refactor Menu

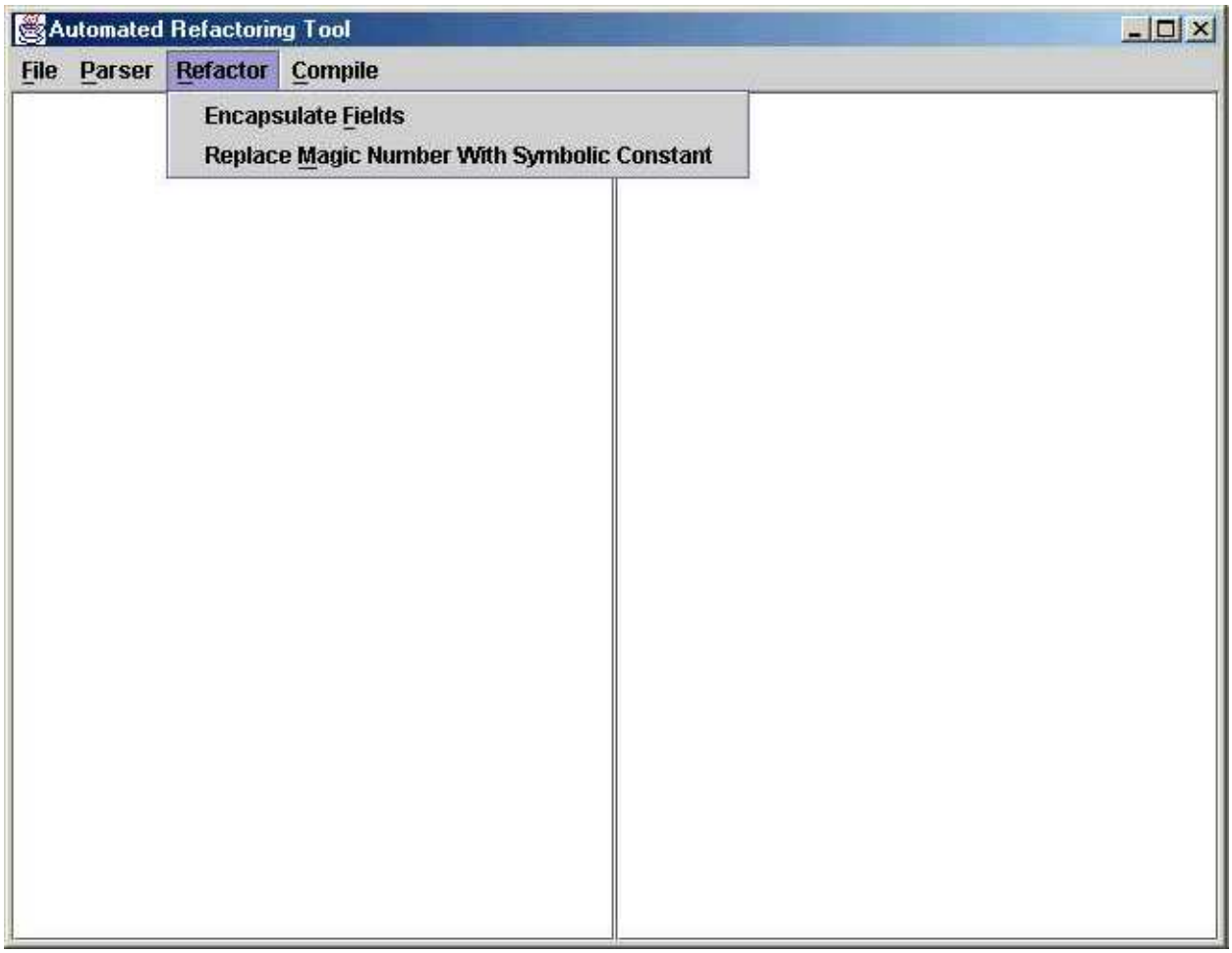


Figure 2.1.4.1 Refactor Menu window

NAVIGATION TABLE		
2.2.7	**State**	**Resulting State**
	Refactor Menu	
Replace Magic Number with Symbolic Constant		Replace Magic Number refactoring transformation applied and code redisplayed
Encapsulate Fields		Encapsulate fields refactoring transformation applied and code redisplayed

Figure 2.1.4.2 - Refactor Menu window navigation table

2.2.8 2.1.5 Compile Menu

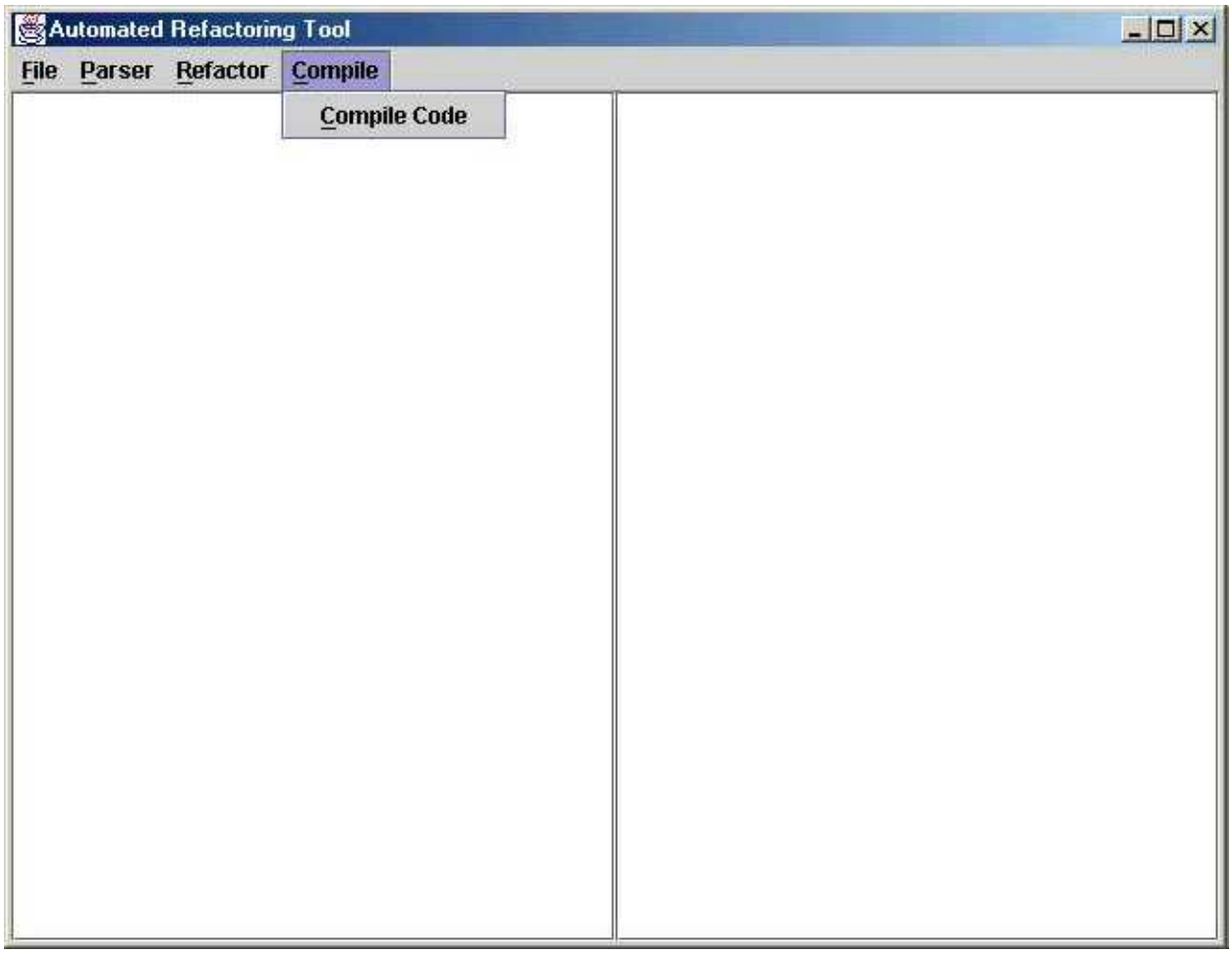


Figure 2.1.5.1 - Compile Menu window

NAVIGATION TABLE		
2.2.9	**State**	**Resulting State**
	Compile Menu	
Compile Code		Code Compiled

Figure 2.1.5.2 - Compile Menu window navigation table

2.2.10 2.1.6 Open Window

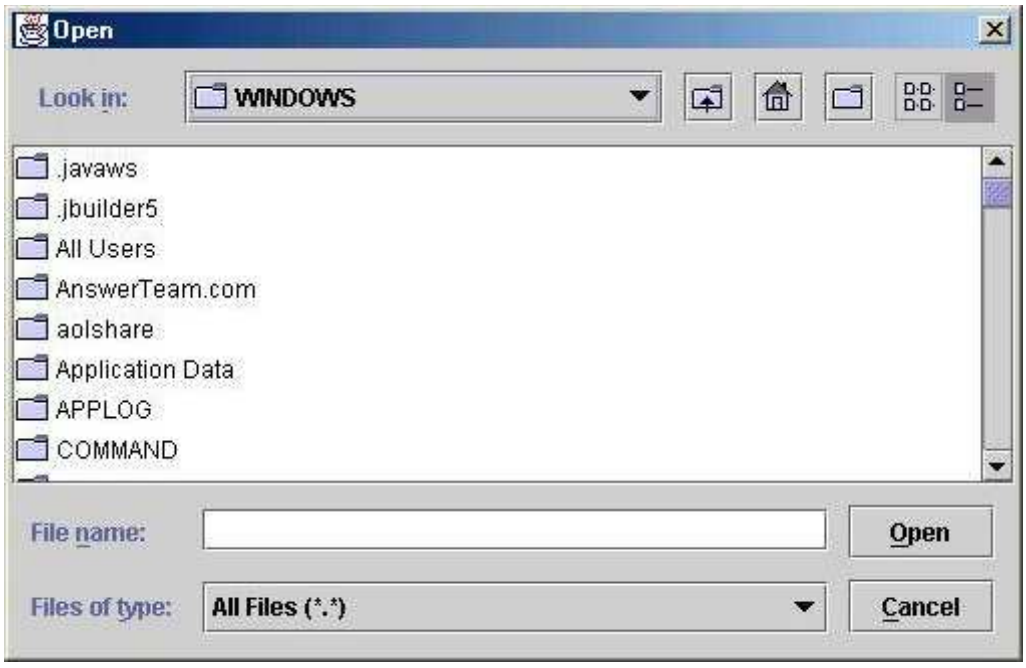


Figure 2.1.6.1 - Open window

NAVIGATION TABLE		
2.2.11	**State**	**Resulting State**
	Open Window	
Open		File selected opened and displayed
Cancel		Window Closed

Figure 2.1.6.2 - Open window navigation table

2.1.7 Save Window

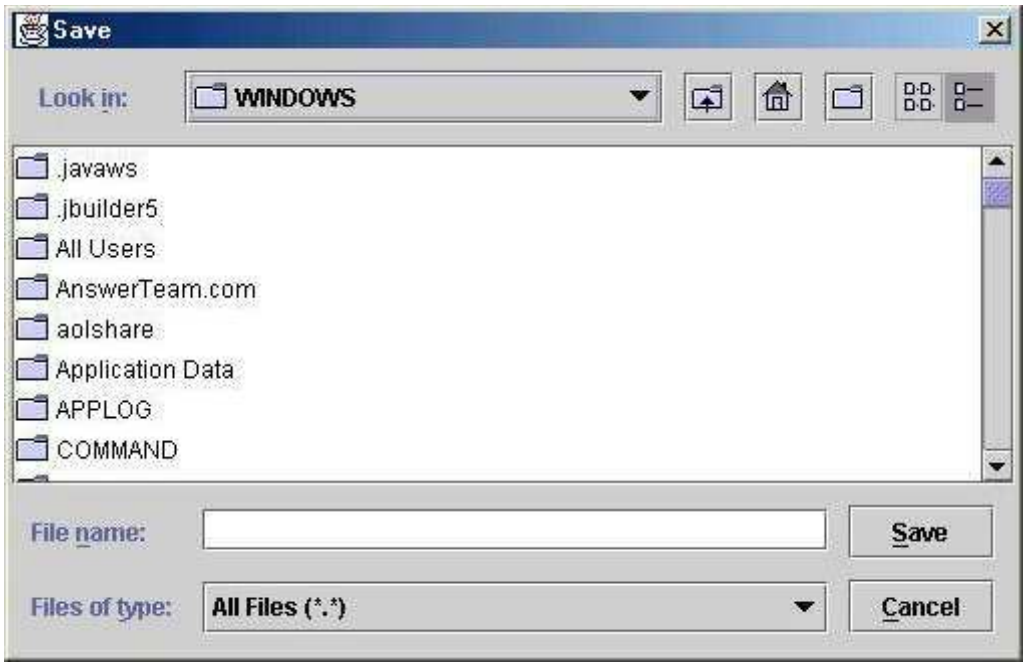


Figure 2.1.7.1 - Save window

NAVIGATION TABLE		
2.2.12	**State**	**Resulting State**
	Open Window	
Open		File selected saved
Cancel		Window Closed

Figure 2.1.7.2 - Save window navigation table

2.2.13 2.1.8 Display View

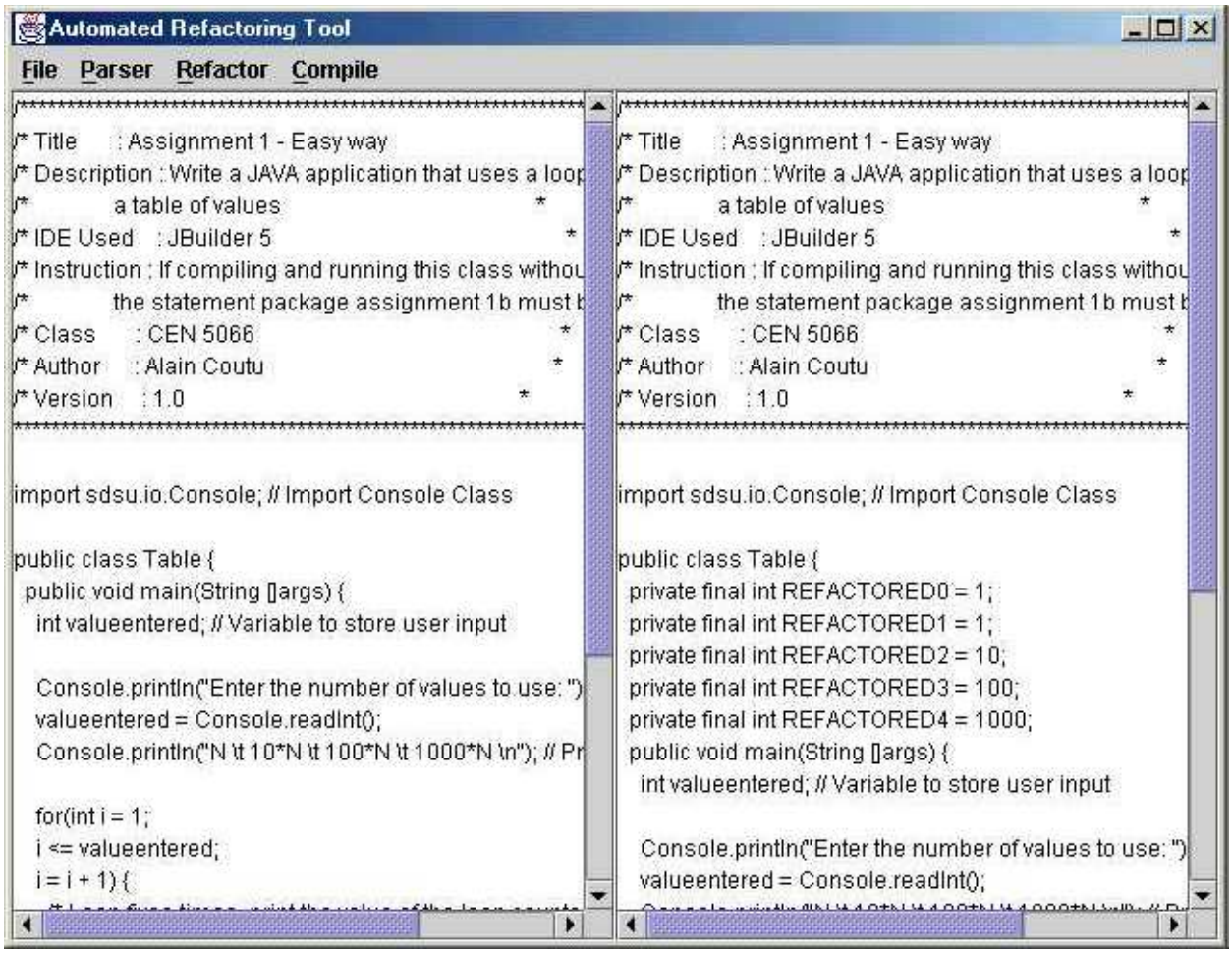


Figure 2.1.8.1 - Display View window

2.2 Data Dictionary for GUI Components

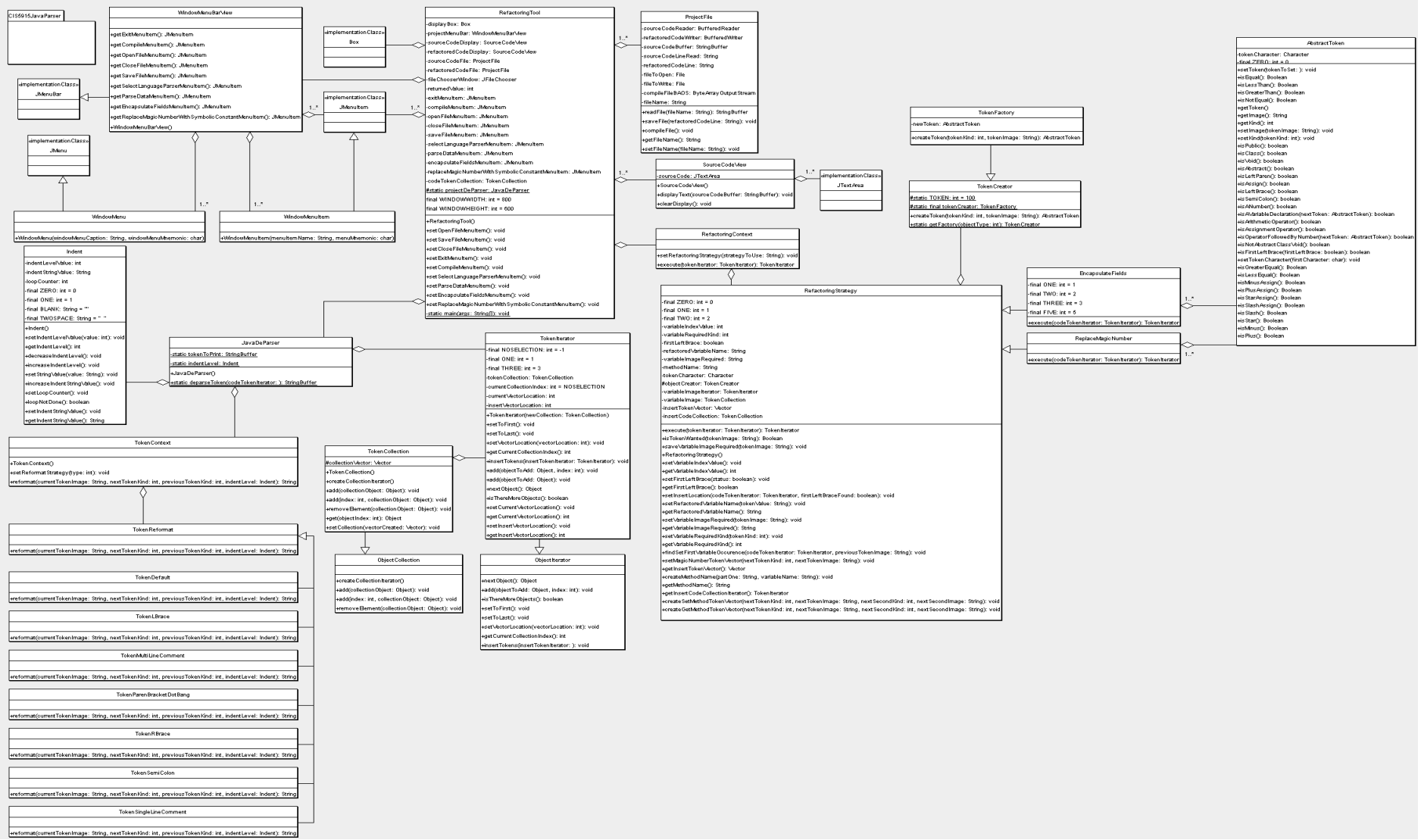
Item Name	Type	Description	Constraints	Environment	Security	Composition
ARTMainWindow	JFrame	Main application screen	None	Windows	None	JFrame + WindowMenuBar + DisplayView
Button	JButton	Button	None	Windows	None	Button graphic
CancelButton	Button	Cancel button	None	Windows	None	Button + CancelImage
CancelImage	Image	Cancel image	None	Windows	None	Graphical representation of cancel
CloseButton	JButton	Close button	None	Windows	None	Button + CloseImage
CloseImage	Image	Close image	None	Windows	None	Graphical representation of close
CloseMenuItem	JMenuItem	Close file menu option	None	Windows	None	WindowMenuItem
CompileMenuItem	JMenuItem	Compile menu option	None	Windows	None	WindowMenuItem

ContentList	JList	List of files available	None	Windows	None	ScrollList + ItemsName
CreateFolderButton	JButton	Create folder button	None	Windows	None	Button + CreateFolderImage
CreateFolderImage	Image	Create folder image	None	Windows	None	Graphical representation of create folder
DetailsButton	JButton	Display button	None	Windows	None	Button + DetailsImage
DetailsImage	Image	Details image	None	Windows	None	Graphical representation of details
DirectoryLabel	Label	Title for directory list	None	Windows	None	"Look in:"
DirectoryList	JList	List of available directories	None	Windows	None	ScroolList + ItemsName
DisplayView	HorizontalBox	Display area for the code	None	Windows	None	ARTMainWindow + SourceCodeView + RefactoredView
EncapsulateFieldsMenuItem	JMenuItem	Encapsulate fields menu option	None	Windows	None	WindowMenuItem
ExitMenuItem	JMenuItem	Exit menu option	None	Windows	None	WindowMenuItem
FileNameLabel	Label	Label for file name	None	Windows	None	["File name:"]
FileNameText	JTextField	File name	None	Windows	None	characters
FileTypeLabel	Label	Label for file type	None	Windows	None	[Files of type:"]
FileTypeList	JList	List of file names	None	Windows	None	ScroolList + ItemsName
FolderListButton	JButton	Folder list button	None	Windows	None	Button + FolderListImage
FolderListImage	Image	Folder List Image	None	Windows	None	Graphical representation of folder list
FolderUpButton	JButton	Folder up button	None	Windows	None	Button + FolderUpImage
FolderUpImage	Image	Folder up Image	None	Windows	None	Graphical representation of folder up
HomeButton	JButton	Home button	None	Windows	None	Button + HomeButtonImage
HomeButtonImage	Image	Home image	None	Windows	None	Graphical representation of home
Icon	Image	Frame icon	None	Windows	None	Image
ItemsName	Text	Items name	None	Windows	None	characters
JFrame	JFrame	Window container	None	Windows	None	Icon + JFrameTitle + JFrameButton
JFrameButton	Button	ART main window buttons	None	Windows	None	MinimizeButton + MaximizeButton + CloseButton
JFrameTitle	Text	Application title	None	Windows	None	characters
ListButton	JButton	List button	None	Windows	None	Button + ListImage
ListImage	Image	Details image	None	Windows	None	Graphical representation of list
MaximizeButton	JButton	Maximize button	None	Windows	None	Button + MaximizeImage
MaximizeImage	Image	Maximize image	None	Windows	None	Graphical representation of maximize
MinimizeButton	JButton	Minimize button	None	Windows	None	Button + MinimizeImage
MinimizeImage	Image	Minimize image	None	Windows	None	Graphical representation of minimize
OpenButton	JButton	Open button	None	Windows	None	Button + OpenImage
OpenButtonImage	Image	Open image	None	Windows	None	Graphical representation of open
OpenFile	Process	Open file capability	None	Windows	None	OpenFileWindow + OpenMenuItem + DisplayFile
OpenFileWindow	JFileChooser	Dialog box for opening file	None	Windows	None	JFrame + DirectoryLabel + DirectoryList + FolderUpButton + HomeButton+ FolderListButton

						+ CreateFolderButton + ListButton + DetailsButton +ContentList + FileNameLabel + FileNameText + OpenButton + FileTypeLabel + FileTypeList + CancelButton
OpenMenuItem	JMenuItem	Open file menu option	None	Windows	None	WindowMenuItem
ParseDataMenuItem	JMenuItem	Parse data menu option	None	Windows	None	WindowMenuItem
RefactoredCodeView	JScrollPane	Source code read	None	Windows	None	characters
ReplaceMagicNumberMenuItem	JMenuItem	Replace magic number menu option	None	Windows	None	WindowMenuItem
SaveFileWindow	JFileChooser	Dialog box for opening file	None	Windows	None	JFrame + DirectoryLabel + DirectoryList + FolderUpButton + HomeButton+ FolderListButton + CreateFolderButton + ListButton + DetailsButton +ContentList + FileNameLabel + FileNameText + OpenButton + FileTypeLabel + FileTypeList + CancelButton
SaveMenuItem	JMenuItem	Save file menu option	None	Windows	None	WindowMenuItem
ScroolList	JList	Scrollable list	None	Windows	None	List
SelectLanguageMenuItem	JMenuItem	Select language parser menu option	None	Windows	None	WindowMenuItem
SourceCodeView	JScrollPane	Source code read	None	Windows	None	characters
WindowCompileMenu	JMenu	Compile Menu option	None	Windows	None	ARTMainWindow + CompileMenuItem
WindowFileMenu	JMenu	File Menu option	None	Windows	None	ARTMainWindow + OpenMenuItem + CloseMenuItem + SaveMenuItem + ExitMenuItem
WindowMenu	JMenu	Menu for the menu bar	None	Windows	None	WindowFileMenu + WindowParseMenu + WindowRefactorMenu + WindowCompileMenu
WindowMenuBar	JMenuBar	Menu bar for the main window	None	Windows	None	WindowMenu + WindowMenuCaption + WindowMenuMnemonic
WindowMenuCaption	String	Name of the menu option	None	Windows	None	characters
WindowMenuItem	JMenuItem	Menu item	None	Windows	None	WindowMenuItemCaption + WindowMenuItemMnemonic
WindowMenuItemCaption	String	Name of the menu item	None	Windows	None	characters
WindowMenuItemMnemonic	Character	Menu item shortcut menu	None	Windows	None	[a..z / A..Z]
WindowMenuMnemonic	Character	Menu shortcut letter	None	Windows	None	[a..z / A..Z]
WindowParseMenu	JMenu	Parse Menu option	None	Windows	None	ARTMainWindow + SelectLanguageParserMenuItem + ParseDataMenuItem
WindowRefactorMenu	JMenu	RefactorMenu option	None	Windows	None	ARTMainWindow + ReplacemagicNumberMenuItem + EncapsulateFieldsMenuItem

3.0 Decomposition Description

3.1 Preliminary Design Description



3.2 Sequence Diagrams

2.2.14 3.2.1 ParseCode

The sequence diagrams of the ParseCode Use Case is not provided for the following reasons:

a) The code is part of the CIS5915JavaParser package, and
b) The package code was generated using the JavaCC parser generator code.

3.2.2 RefactorCode

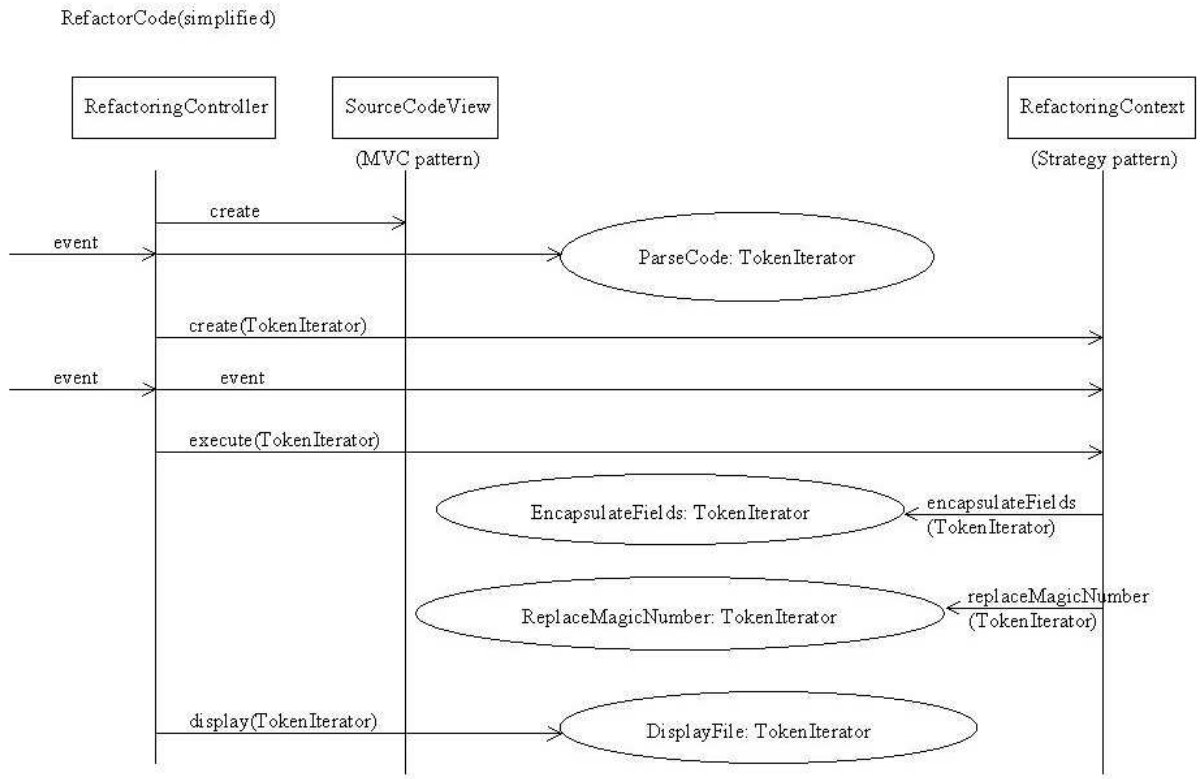


Figure 3.2.2.1 - Refactor Code sequence diagram

3.2.3 DisplayFile

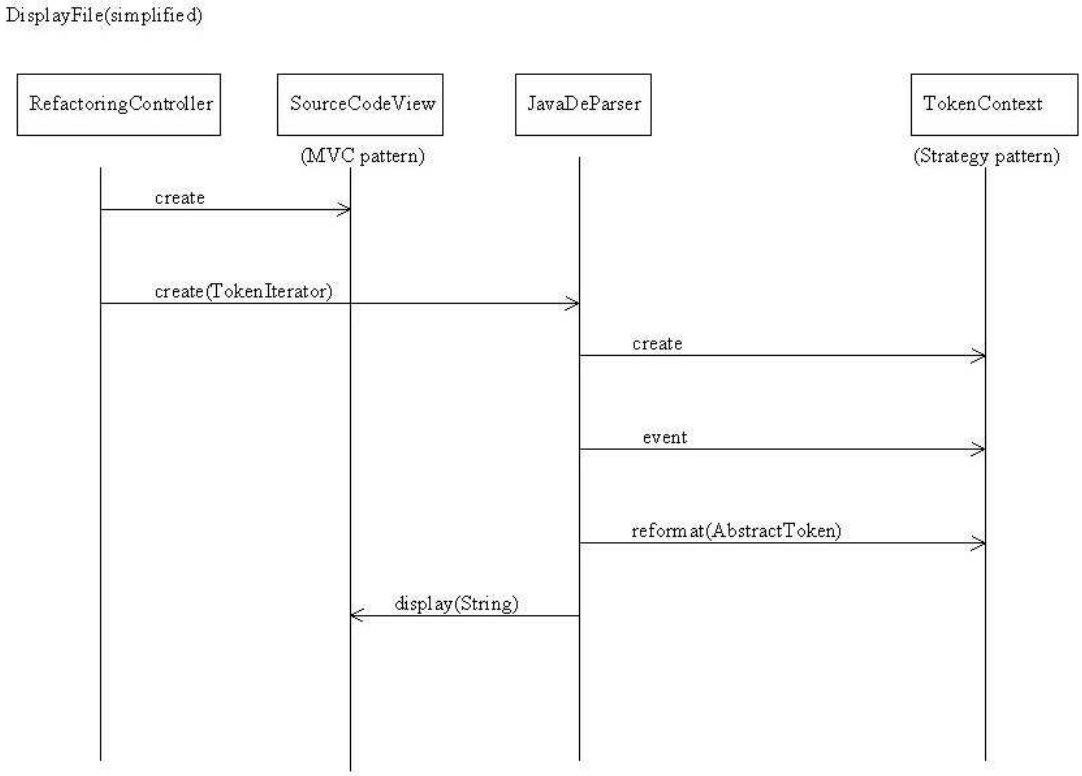


Figure 3.2.3.1 - Display File sequence diagram

2.2.15 3.2.4 EncapsulateFields, ReplaceMagicNumber, other refactoring transformations*

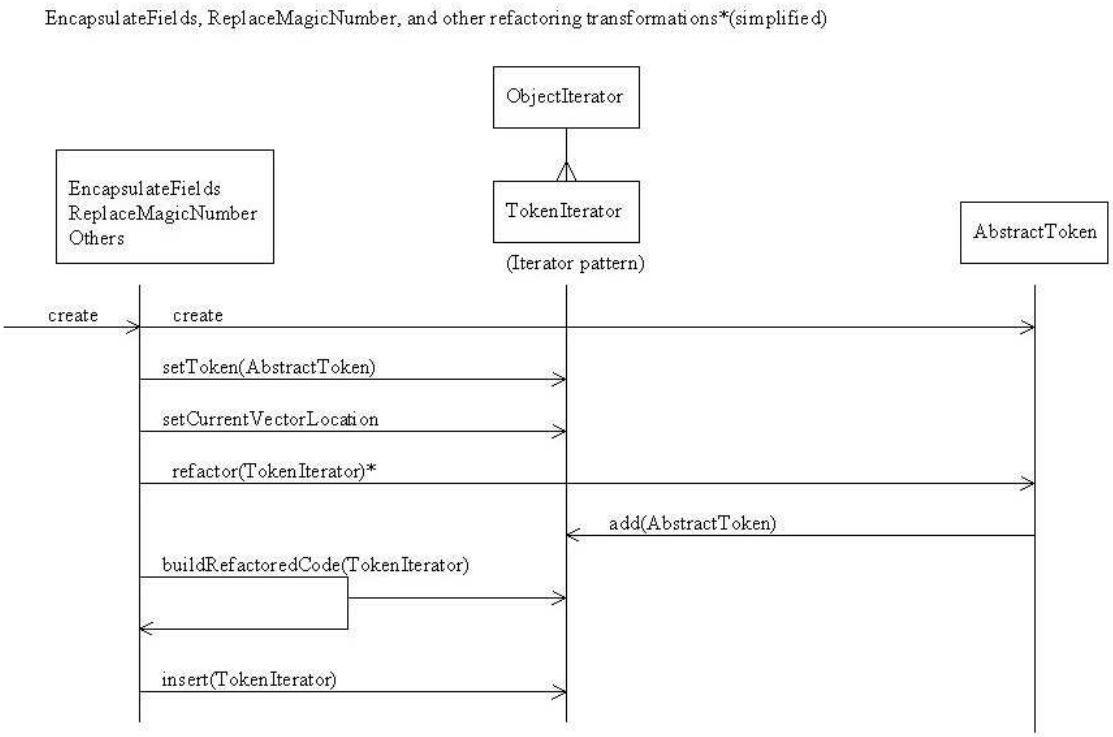


Figure 3.2.4.1 - EncapsulateFields, ReplaceMagicNumber, other refactoring transformations sequence diagram

3.3 Decomposition Data Dictionary

Item Name	Type	Description	Constraints	Environment	Security	Composition
AbstractToken	Class	Token Class for capturing code sequence	Public class. Extends Token class.	Windows	None	vectorToken + tokenCharacter + ZERO + setToken + getToken + getKind + setKind + getImage + setImage + isPublic + isClass + isVoid + isAbstract + isLeftParen + isAssign + isGreaterThan + isLessThan + isEqual + isNotEqual + isGreaterEqual +

						isLessEqual + isIncrement + isDecrement + isPlus + isMinus + isStar + isSlash + isPlusAssign+ isMinusAssign + isStarAssign + isSlashAssign + isLeftBrace + isSemiColon + isANumber + isAVariableDeclaration + isArithmeticOperator + isAssignmentOperator + isOperatorFollowedByNumber + isNotAbstractClassVoid + isFirstLeftBrace + setTokenCharacter
add	Method	Method to add object to a collection or an iterator	Public	Windows	None	collectionObject
add	Method	Method to add object to a collection or an iterator at a specific index	Public (As well abstract for ObjectCollection and ObjectIterator classes)	Windows	None	collectionObject + index
BLANK	String	Constant value representing one blank space	Private and final	Windows	None	character
clearDisplay	Method	Method that clear the GUI display	Public	Windows	None	N/A
closeFileMenuItem	JMenuItem	Menu item for closing a file from the GUI	Private	Windows	None	JMenuItem
closeMenuItem	WindowMenuItem	Value representing the close file menu item	Private	Windows	None	WindowMenuItem
codeTokenCollection	TokenCollection	Collection of tokens	Private	Windows	None	TokenCollection
codeTokenIterator	TokenIterator	Iterator object containing the tokens read	Public	Windows	None	TokenIterator
collectionObject	Object	Object representing a collection	None	Windows	None	N/A
collectionVector	Vector	Object representing a vector	Protected	Windows	None	N/A
compileFile	Method	Method that compiles a file	Public	Windows	None	N/A
compileFileBAOS	ByteArrayOutputStream	Stream object used for compiling code	Private	Windows	None	ByteArrayOutputStream
compileMenuItem	JMenuItem	Menu item for compiling code from the GUI	Private	Windows	None	JMenuItem
compileMenuItem	WindowMenuItem	Value representing the compile file menu item	Private	Windows	None	WindowMenuItem
compiler	sun.tools.javac.Main	Object used for compiling code	Private	Windows	None	sun.tools.javac.Main
createCollectionIterator	Method	Method returning a new TokenIterator	Public (As well abstract for ObjectCollection class)	Windows	None	N/A
createGetMethodTokenVector	Method	Method that creates the vector of tokens for a get method	Public	Windows	None	nextTokenImage + nextTokenKind + nextSecondKind + nextSecondImage
createMethodName	Method	Method that creates a method name	Public	Windows	None	partOne + variableName
createSetMethodTokenVector	Method	Method that creates the vector of tokens for a set method	Public	Windows	None	nextTokenImage + nextTokenKind + nextSecondKind + nextSecondImage
createToken	Method	Method for creating abstract tokens	Public. (As well abstract for TokenCreator class)	Windows	None	tokenKind + tokenImage
currentCollectionIndex	int	Current index in the collection	Private. Initialize to	Windows	None	{1..9}

APPENDIX B

			NOSELECTION			
current	AbstractToken	Current token read	Private.	Windows	None	AbstractToken
currentToken	AbstractToken	Current token read	Private. (As well static in JavaDeParser class)	Windows	None	AbstractToken
currentTokenImage	String	Value representing the image of the current token	Public	Windows	None	character
currentVectorLocation	int	Current index in the iterator	Private.	Windows	None	{1..9}
decreaseIndentLevel	Method	Method that decreases the indentation level value	Public	Windows	None	N/A
deparseToken	Method	Method used to reformat tokens	Public and Static	Windows	None	codeTokenIterator
displayBox	Box	Object to contain the JTextArea	Private	Windows	None	Box
displayText	Method	Method that display text to the GUI	Public	Windows	None	sourceCodeBuffer
EncapsulateFields	Class	Class to implement the encapsulate fields refactoring using strategy pattern	Public. Need JOptionPane and AbstractToken classes. Extends RefactoringStrategy class.	Windows	None	currentToken + nextToken + previousToken + nextSecondToken + nextThirdToken + ONE + TWO + THREE + FIVE + execute
encapsulateFields MenuItem	JMenuItem	Menu item for performing the refactoring encapsulate fields from the GUI	Private	Windows	None	JMenuItem
encapsulateFields MenuItem	Window MenuItem	Value representing the encapsulate fields menu item	Private	Windows	None	WindowMenuItem
execute	Method	Method to perform the refactoring strategy	Public	Windows	None	codeTokenIterator
exitMenuItem	JMenuItem	Menu item for exiting the GUI	Private	Windows	None	JMenuItem
exitMenuItem	Window MenuItem	Value representing the exit menu item	Private	Windows	None	WindowMenuItem
fileChooserWindow	JFileChooser	Window object for file section	Private	Windows	None	JFileChooser
fileName	String	Object used to hold the name of a file	Private	Windows	None	{character}
fileToRead	File	Object used to hold a file for reading	Private.	Windows	None	File
fileToWrite	File	Object used to hold a file for writing	Private.	Windows	None	File
findSetFirstVariableOccurrence	Method	Method that finds the first occurrence of a variable and get the previoud item token	Public	Windows	None	codeTokenIterator + previousTokenImage
firstCharacter	char	Character of a token	Private	Windows	None	{character}
firstLeftBrace	boolean	Value representing if the token if the first left brace	Private	Windows	None	True | False
firstLeftBraceFound	boolean	Value representing if the first left brace was found	Private	Windows	None	True | False
FIVE	int	Constant for value 5	Private and final	Windows	None	5
get	Method	Method to retrieve a collection object at a given index	Public	Windows	None	objectIndex
getCloseMenuItem	Method	Method that returns the close menu item	Public	Windows	None	N/A
getCompileMenuItem	Method	Method that returns the compile menu item	Public	Windows	None	N/A
getCurrentCollectionIndex	Method	Method that retrieve the current index in the collection	Public	Windows	None	N/A
getCurrentVectorLocation	Method	Method that retrieve the current index in the iterator	Public	Windows	None	N/A
getEncapsulateFieldsMenuItem	Method	Method that returns the encapsulate fields menu	Public	Windows	None	N/A

		item				
getExitMenuItem	Method	Method that returns the exit menu item	Public	Windows	None	N/A
getFactory	Method	Method returning the factory for creating tokens	Static and final	Windows	None	objectType
getFileName	Method	Method that retrieves the name of a file	Public	Windows	None	N/A
getFirstLeftBrace	Method	Method that retrieves the firstLeftBrace value	Public	Windows	None	N/A
getIndentLevel	Method	Method that retrieves the indentation level value	Public	Windows	None	N/A
getIndentStringValue	Method	Method that returns the indentation value	Public	Windows	None	N/A
getInsertCodeCollectionIterator	Method	Method that returns the iterator of the insertion vector	Public	Windows	None	N/A
getInsertTokenVector	Method	Method that retrieves the vector of tokens to be inserted	Public	Windows	None	N/A
getInsertVectorLocation	Method	Method that retrieve the current index in the iterator for insertion	Public	Windows	None	N/A
getImage	Method	Method to retrieve the image of abstract token	Public	Windows	None	N/A
getKind	Method	Method to retrieve the kind of abstract token	Public	Windows	None	N/A
getOpenMenuItem	Method	Method that returns the open menu item	Public	Windows	None	N/A
getMethodName	Method	Method that returns the name of a method	Public	Windows	None	N/A
getParseDataMenuItem	Method	Method that returns the parse data menu item	Public	Windows	None	N/A
getSaveMenuItem	Method	Method that returns the save menu item	Public	Windows	None	N/A
getSelectLanguageParserMenuItem	Method	Method that returns the select language parser menu item	Public	Windows	None	N/A
getRefactoredVariableName	Method	Method that retrieves the refactored variable name	Public	Windows	None	N/A
getReplaceMagicNumberWithSymbolicConstantMenuItem	Method	Method that returns the replace magic number with symbolic constant menu item	Public	Windows	None	N/A
getToken	Method	Method to retrieve the abstract token	Public	Windows	None	N/A
getVariableImagerequired	Method	Method that retrieves the image of the variable	Public	Windows	None	N/A
getVariableIndexValue	Method	Method that retrieves the index value in the vector of a variable	Public	Windows	None	N/A
getVariableRequiredKind	Method	Method that retrieves the kind of the variable	Public	Windows	None	N/A
increaseIndentLevel	Method	Method that increases the indentation level value	Public	Windows	None	N/A
increaseIndentStringValue	Method	Method that increase the indentation value	Public	Windows	None	N/A
Indent	Class	Class used to represent the indentation required for deparsing tokens	Public	Windows	None	indentLevelValue + indentStringValue + loopCounter + ZERO + ONE + BLANK + TWOSPACE + setIndentLevelValue + getIndentLevel + decreaseIndentLevel + increaseIndentLevel + setStringValue + increaseIndentStringValue + setLoopCounter +

APPENDIX B

						loopNotDone + setIndentStringValue + getIndentStringValue
indentLevel	Indent	Value representing the indentation required	Private and static	Windows	None	Indent
indentLevelValue	int	Value representing the indentation level	Private	Windows	None	{1..9}
indentStringValue	String	Value representing the indentation	Private	Windows	None	character
index	int	Value representing the location in a vector or an iterator	None	Windows	None	{1..9}
insertCodeCollection	TokenCollection	Value representing a collection of tokens	Private	Windows	None	TokenCollection
insertTokens	Method	Method that inserts tokens in the iterator object	Public	Windows	None	insertTokenIterator
insertTokenIterator	TokenIterator	Iterator object containing tokens to be inserted	Public	Windows	None	TokenIterator
insertTokenVector	Vector	Value representing a vector of token for insertion	Private	Windows	None	Vector
insertVectorLocation	int	Current index in the iterator for insertion	Private.	Windows	None	{1..9}
isAbstract	Method	Method to determine if the token is the reserved word Abstract	Public	Windows	None	N/A
isANumber	Method	Method to detwermine if the token is a number	Public	Windows	None	N/A
isArithmeticOperator	Method	Method to determine if the token is an arithmetic operator	Public	Windows	None	N/A
isAssign	Method	Method to determine if the token is the reserved word Assign	Public	Windows	None	N/A
isAssignmentOperator	Method	Method to determine if the token is an assignment operator	Public	Windows	None	N/A
isAVariableDeclaration	Method	Method to determine if the token is variable declaration	Public	Windows	None	N/A
isClass	Method	Method to determine if the token is the reserved word Class	Public	Windows	None	N/A
isDecrement	Method	Method to determine if the token is the decrement sign	Public	Windows	None	N/A
isEqual	Method	Method to determine if the token is the equal sign	Public	Windows	None	N/A
isFirstLeftBrace	Method	Method to determine if the token is the first left brace	Public	Windows	None	N/A
isGreaterEqual	Method	Method to determine if the token is the greater than equal sign	Public	Windows	None	N/A
isGreaterThan	Method	Method to determine if the token is the greater than sign	Public	Windows	None	N/A
isIncrement	Method	Method to determine if the token is the increment sign	Public	Windows	None	N/A
isLeftBrace	Mehtod	Method to determine if the token is a left brace	Public	Windows	None	N/A
isLeftParen	Method	Method to determine if the toke is a left parenthesis	Public	Windows	None	N/A
isLessEqual	Method	Method to determine if the token is the less than equal sign	Public	Windows	None	N/A
isLessThan	Method	Method to determine if the token is the less than sign	Public	Windows	None	N/A
isMinus	Mehtod	Method to determine if the token is the minus sign	Public	Windows	None	N/A
isMinusAssign	Mehtod	Method to determine if the token is the minus assign sign	Public	Windows	None	N/A

isNotAbstractClassVoid	Method	Method to determine if the token is not the reserved words abstract, class or void	Public	Windows	None	N/A
isNotEqual	Method	Method to determine if the token is the not equal sign	Public	Windows	None	N/A
isOperatorFollowedByNumber	Method	Method to determine if the token is an operator followed by a number	Public	Windows	None	N/A
isPlus	Mehtod	Method to determine if the token is the plus sign	Public	Windows	None	N/A
isPlusAssign	Mehtod	Method to determine if the token is the plus assign sign	Public	Windows	None	N/A
isPublic	Method	Method to determine if the token is the reserved word Public	Public	Windows	None	N/A
isSemiColon	Method	Method to determine if the token is a semi-colon	Public	Windows	None	N/A
isSlash	Mehtod	Method to determine if the token is the slash sign	Public	Windows	None	N/A
isSlashAssign	Mehtod	Method to determine if the token is the slash assign sign	Public	Windows	None	N/A
isStar	Mehtod	Method to determine if the token is the star sign	Public	Windows	None	N/A
isStarAssign	Mehtod	Method to determine if the token is the star assign sign	Public	Windows	None	N/A
isThereMoreObjects	Method	Method that returns if the iterator is empty	Public	Windows	None	N/A
isTokenWanted	Method	Method to determine isf the token image is in the vector of token images	Public	Windows	None	tokenImage
isVoid	Method	Method to determine if the token is the reserved word Void	Public	Windows	None	N/A
JavaDeParser	Class	Class used to unparse the parsed code	Public. Need JavaParserConstants and AbstractToken classes. Implements JavaParserConstants	Windows	None	tokenToPrint + currentToken + nextToken + previousToken + indentLevel + tokenContext + deparseToken
loopCounter	int	Value representing a loop counter	Private	Windows	None	{1..9}
loopNotDone	Method	Method that returns if a loop is completed	Public	Windows	None	N/A
main	Method	Method that control the application	Private and static	Windows	None	N/A
menuItemName	String	Value representing the name of the menu item	Public	Windows	None	character
menuMnemonic	char	Value representing the menu item mnemonic	Public	Windows	None	{a..z | A..Z}
methodName	String	Value representing the name of a method	Private	Windows	None	character
nextObject	Method	Method that retrieve the next object in the iterator	Public	Windows	None	Object
nextSecondImage	String	Value representing the image of a token	Public	Windows	None	Character
nextSecondKind	int	Value representing the kind of token	Public	Windows	None	{1..9}
nextSecondToken	AbstractToken	Previous token read	Private	Windows	None	AbstractToken
nextThirdToken	AbstractToken	Previous token read	Private	Windows	None	AbstractToken
nextToken	AbstractToken	Previous token read	Private. (As well static in JavaDeParser class)	Windows	None	AbstractToken
nextTokenKind	int	Value representing the kind	Public	Windows	None	{1..9}

		for the next token				
nextTokenImage	String	Value representing the image of the next token	Public	Windows	None	character
newFileName	String	Name of a file	Public	Windows	None	{character}
newToken	AbstractToken	Value representing a new token	Private	Windows	None	AbstractToken
NOSELECTION	int	Constant for value 0	Private and final	Windows	None	0
NUMBEROFCOLUMNS	int	Constant value for the number of columns	Private and final	Windows	None	50
NUMBEROFROWS	int	Constant value for the number of rows	Private and final	Windows	None	30
ObjectCollection	Class	Class that implements the iterator pattern	Public and Abstract.	Windows	None	createCollectionIterator + add + add + removeElement
objectCreator	TokenCreator	Object used to implement the factory pattern	Protected	Windows	None	TokenCreator
objectIndex	int	Index of an object in a collection	None	Windows	None	{1..9}
ObjectIterator	Class	Class that implements the iterator pattern	Public and Abstract.	Windows	None	nextObject + add + isThereMoreObjects + setToFirst + setToLast + setVectorLocation + getCurrentVectorLocation + insertTokens
objectType	int	Type of factory to select	None	Windows	None	{1..9}
ONE	int	Constant for value 1	Private and final	Windows	None	1
openFileMenuItem	JMenuItem	Menu item for opening a file from the GUI	Private	Windows	None	JMenuItem
openMenuItem	WindowMenuItem	Value representing the open file menu item	Private	Windows	None	WindowMenuItem
parseDataMenuItem	JMenuItem	Menu item for parsing data from the GUI	Private	Windows	None	JMenuItem
parseDataMenuItem	WindowMenuItem	Value representing the parse data menu item	Private	Windows	None	WindowMenuItem
partOne	String	Value representing the first part for a method name	Public	Windows	None	character
previous	AbstractToken	Previous token read	Private	Windows	None	AbstractToken
previousToken	AbstractToken	Previous token read	Private. (As well static in JavaDeParser class)	Windows	None	AbstractToken
previousTokenImage	String	Value representing the token image	Public	Windows	None	character
previousTokenKind	int	Value representing the kind of the previous token	Public	Windows	None	{1..9}
projectDeParser	JavaDeParser	Value representing the java deparser	Protected and static	Windows	None	JavaDeParser
ProjectFile	Class	Class used to manipulate source code	Public. Needs io, JOptionPane and sun.tools classes	Windows	None	sourceCodeReader + refactoredCodeWriter + sourceCodeBuffer + sourceCodeLineRead + refactoredCodeLine + fileToRead + fileToWrite + compileFileBAOS + compiler + fileName + readFile + saveFile + compileFile + getFileName + setFileName
projectMenuBar	WindowMenuBarView	Window menu bar view for the GUI	Private	Windows	None	WindowMenuBarView
readFile	Method	Method that reads a text file line by line	Public	Windows	None	fileName
refactoredCodeDisplay	SourceCodeView	Display of the refactored code	Private	Windows	None	SourceCodeView
refactoredCodeFile	ProjectFile	File that contains the refactored code	Private	Windows	None	ProjectFile

refactoredCodeLine	String	Value used to store text to write to a file	Private	Windows	None	{character}
refactoredCodeWriter	Buffered Writer	Writer object to write text one line at a time	Private	Windows	None	BufferedWriter
refactoringContext	RefactoringContext	Value representing the refactoring context for the strategy pattern	Private	Windows	None	RefactoringContext
RefactoringContext	Class	Class used to implement the context of the strategy pattern for performing refactoring	Public	Windows	None	refactoringStrategy + execute + setRefactoringStrategy
refactoringStrategy	RefactoringStrategy	Object that identify the strategy to use for refactoring code	Private	Windows	None	RefactoringStrategy
RefactoringStrategy	Class	Class used to implement the strategy pattern for performing refactoring	Public and Abstract. Need JavaParserConstants, AbstractToken and Vector classes. Implements JavaParserConstants	Windows	None	ZERO + ONE + TWO + variableIndexValue + variableRequiredkind + firstLeftBrace + refactoredVariableName + variableImageRequired + methodName + previous + current + tokenCharacter + objectCreator + insertTokenVector + insertCodeCollection + variableImage + variableImageIterator + execute + setVariableIndexValue + getVariableIndexValue + setFirstLeftBrace + getFirstLeftBrace + setInsertLocation + setRefactoredVariableName + getRefactoredVariableName + setVariableImageRequired + getVariableImageRequired + setVariableRequiredKind + saveVariableImageRequired + getVariableRequiredKind + isTokenWanted + findSetFirstVariableOccurence + setMagicNumberTokenVector + getInsertTokenVector + createMethodName + getMethodName + getInsertCodeCollectionIterator + createSetMethodTokenVector + createGetMethodTokenVector
RefactoringTool	Class	Class used as the controller in the implementation of the MVC pattern	Public. Need the swing, awt, event, Vector and CIS5915JavaParser classes. Extends JFrame	Windows	None	displayBox + projectMenubar + sourceCodeDisplay + refactoredCodeDisplay + sourceCodeFile + refactoredCodeFile + fileChooserWindow + returnedValue + exitMenuItem + compileMenuItem +

						openFileMenuItem + closeFileMenuItem + saveFileMenuItem + selectLanguageParserMenuItem + parseDataMenuItem + encapsulateFieldsMenuItem + replaceMagicNumberWithSymbolicConstantMenuItem + codeTokenCollection + projectDeParser + refactoringContext + WINDOWWIDTH + WINDOWHEIGHT + setOpenFileMenuItem + setSaveFileMenuItem + setCloseFileMenuItem + setExitMenuItem + setCompileMenuItem + setSelectLanguageParserMenuItem + setParseDataMenuItem + setEncapsulateFieldsMenuItem + setReplaceMagicNumberWithSymbolicConstantMenuItem + main
refactoredVariableName	String	Value representing a refactored variable name	Private	Windows	None	Character
reformat	Method	Method that reformats tokens	Public	Windows	None	currentTokenImage + nextTokenKind + previousTokenKind + indentLevel
reformatStrategy	TokenReformat	Value representing the strategy for reformatting tokens	Private	Windows	None	TokenReformat
removeElement	Method	Method to remove collection objects	Public	Windows	None	collectionObject
ReplaceMagicNumber	Class	Class to implement the encapsulate fields refactoring using strategy pattern	Public. Need JOptionPane and AbstractToken classes. Extends RefactoringStrategy class.	Windows	None	currentToken + nextToken + previousToken + execute
replacemagicNumberWithSymbolicConstantMenuItem	JMenuItem	Menu item for performing the refactoring replace magic number with symbolic constant from the GUI	Private	Windows	None	JMenuItem
replaceMagicNumberWithSymbolicConstant	WindowMenuItem	Value representing the replace magic number with symbolic constant menu item	Private	Windows	None	WindowMenuItem
returnedValue	int	Value representing the choice returned from the fileChooserWindow	Private	Windows	None	{1..9}
saveFile	Method	Method that writes a text file line by line	Public	Windows	None	refactoredCodeLine
saveFileMenuItem	JMenuItem	Menu item for saving a file from the GUI	Private	Windows	None	JMenuItem
saveMenuItem	WindowMenuItem	Value representing the save file menu item	Private	Windows	None	WindowMenuItem
saveVariableImageRequired	Method	Method to save the variable image required	Public	Windows	None	tokenImage
selectLanguageParserMenuItem	JMenuItem	Menu item for selecting a language parser from the GUI	Private	Windows	None	JMenuItem
selectLanguagePar	Window	Value representing the select	Private	Windows	None	WindowMenuItem

serMenuItem	MenuItem	language parser menu item				
setCloseFileMenuItem	Method	Method to control the close menu item event action	Public	Windows	None	N/A
setCollection	Method	Method to set a collection using a vector object	Public	Windows	None	vectorCreated
setCompileMenuItem	Method	Method to control the compile menu item event action	Public	Windows	None	N/A
setCurrentVectorLocation	Method	Method that sets the current index in the iterator	Public	Windows	None	N/A
setEncapsulateFieldsMenuItem	Method	Method to control the encapsulate fields menu item event action	Public	Windows	None	N/A
setExitMenuItem	Method	Method to control the exit menu item event action	Public	Windows	None	N/A
setFileName	Method	Method that sets the name of a file	Public	Windows	None	newFileName
setFirstLeftBrace	Method	Method that sets the firstLeftBrace variable	Public	Windows	None	status
setIndentLevelValue	Method	Method that sets the indentation level value	Public	Windows	None	value
setIndentStringValue	Method	Method that sets the indentation value	Public	Windows	None	N/A
setInsertLocation	Method	Method that sets the insertion location in the vector for new tokens	Public	Windows	None	codeTokenIterator + firstLeftBraceFound
setInsertVectorLocation	Method	Method that sets the current index in the iterator for insertion	Public	Windows	None	N/A
setImage	Method	Method to set the token image	Public	Windows	None	tokenImage
setKind	Method	Method to set the token kind	Public	Windows	None	tokenKind
setLoopCounter	Method	Method that initialize the loop counter value	Public	Windows	None	N/A
setMagicNumberTokenVector	Method	Method that creates the vector of tokens required to perform refactoring Replace Magic Number	Public	Windows	None	nextTokenKind + nextTokenImage
setOpenFileMenuItem	Method	Method to control the open menu item event action	Public	Windows	None	N/A
setParseDataMenuItem	Method	Method to control the parse data menu item event action	Public	Windows	None	N/A
setRefactoredVariableName	Method	Method that sets the name of the refactored variable	Public	Windows	None	tokenValue
setRefactoringStrategy	Method	Method that sets the strategy to use for refactoring code	Public	Windows	None	strategyToUse
setReformatStrategy	Method	Method that sets the strategy for reformatting tokens	Public	Windows	None	type
setReplaceMagicNumberWithSymbolicConstantMenuItem	Method	Method to control the replace magic number with symbolic constant menu item event action	Public	Windows	None	N/A
setSaveFileMenuItem	Method	Method to control the save menu item event action	Public	Windows	None	N/A
setSelectLanguageParserMenuItem	Method	Method to control the select language parser menu item event action	Public	Windows	None	N/A
setStringValue	Method	Method that sets the indentation value	Public	Windows	None	value
setToFirst	Method	Method to set the index of the iterator to the first object	Public	Windows	None	N/A
setToken	Method	Method to set the token object	Public	Windows	None	AbstractToken
SetTokenCharacter	Method	Method to set the token character	Public	Windows	None	firstCharacter
setToLast	Method	Method to set the index of the iterator to the last object	Public	Windows	None	N/A
setVariableImage	Method	Method that sets the image	Public	Windows	None	tokenImage

Required		of the variable				
setVariableIndexValue	Method	Method that sets the index value in the vector of a variable	Public	Windows	None	N/A
setVariableRequiredKind	Method	Method that sets the kind of the variable	Public	Windows	None	tokenKind
setVectorLocation	Method	Method that sets the location in the iterator	Public	Windows	None	vectorLocation
sourceCode	JTextArea	Value representing the area where text is displayed	Private	Windows	None	JTextArea
sourceCodeBuffer	StringBuffer	Buffer used to store text read	Private	Windows	None	StringBuffer
sourceCodeDisplay	SourceCodeView	Display for the source code	Private	Windows	None	SourceCodeView
sourceCodeFile	ProjectFile	File containing the source code	Private	Windows	None	ProjectFile
sourceCodeLineRead	String	Object to hold line of text read	Private	Windows	None	{character}
sourceCodeReader	BufferedReader	Reader object to read text file one line at a time	Private	Windows	None	BufferedReader
SourceCodeView	Class	Class used to display text from a file	Public. Extends JTextArea	Windows	None	sourceCode + NUMBEROFROWS + NUMBEROFCOLUMNS + displayText + clearDisplay
strategyToUse	String	Object that holds the refactoring strategy	Public	Windows	None	{character}
status	boolean	Value representing the firstLeftBrace variable value	Public	Windows	None	True | False
THREE	int	Constant for value 3	Private and final	Windows	None	3
tokenCharacter	Character	Value representing a character from the token image	Private	Windows	None	{a..z | A..Z}
tokenCollection	TokenCollection	Collection object representing the vector of tokens	Private	Windows	None	TokenCollection
TokenCollection	Class	Class for implementing the iterator pattern	Public. Extends ObjectCollection class. Need Vector class	Windows	None	collectionVector + createCollectionIterator + add + add + removeElement + get + setCollection
tokenContext	TokenContext	TokenContext object representing the strategy for deparsing tokens	Private and static	Windows	None	TokenContext
TokenContext	Class	Class to implement the context of the strategy for reformatting tokens	Public. Needs JavaParserConstants class. Implements JavaParserConstants	Windows	None	reformatStrategy + setReformatStrategy + reformat
tokenCreator	TokenFactory	Object define the token factory	Private	Windows	None	TokenFactory
TokenCreator	Class	Class to create token using abstract factory pattern	Public and Abstract. Need JavaParserConstants and AbstractToken classes. Implements JavaParserConstants class.	Windows	None	tokenCreator + getFactory + createToken
TokenDefault	Class	Class representing the default token reformatting strategy	Public. Extends TokenReformat	Windows	None	reformat
TokenFactory	Class	Class used to implement the abstract factory pattern	Public. Need the Token and AbstractToken classes. Extends AbstractFactory	Windows	None	newToken + createToken

TokenIterator	Class	Class for implementing the iterator pattern	Public. Extends ObjectIterator class	Windows	None	NOSELECTION + ONE + FOUR + tokenCollection + currentCollectionIndex + currentVectorLocation + insertVectorLocation + setToFirst + setToLast + setVectorLocation + getCurrentCollectionIndex + insertTokens + add + add + nextObject + isThereMoreObjects + setCurrentVectorLocation + getCurrentVectorLocation + setInsertVectorLocation + getInsertVectorLocation
tokenKind	int	Kind of abstract token created	None	Windows	None	{1..9}
tokenImage	String	String representation of the abstract token	None	Windows	None	character
TokenLBrace	Class	Class representing the left brace token reformatting strategy	Public. Extends TokenReformat	Windows	None	reformat
TokenMultiLineComment	Class	Class representing the multi line comment token reformatting strategy	Public. Extends TokenReformat	Windows	None	reformat
TokenParenBracketDotBang	Class	Class representing the parentheses, brackets, dot and bang tokens reformatting strategy	Public. Extends TokenReformat	Windows	None	reformat
TokenRBrace	Class	Class representing the right brace token reformatting strategy	Public. Extends TokenReformat	Windows	None	reformat
TokenReformat	Class	Class used to implement the strategy pattern for reformatting tokens	Public and Abstract. Needs JavaParserConstants class. Implements JavaParserConstants	Windows	None	reformat
TokenSemiColon	Class	Class representing the semi colon token reformatting strategy	Public. Extends TokenReformat	Windows	None	reformat
TokenSingleLineComment	Class	Class representing the single line comment token reformatting strategy	Public. Extends TokenReformat	Windows	None	reformat
tokenToPrint	StringBuffer	Buffer of strings representing the unparsed code	Private and Static	Windows	None	N/A
tokenValue	String	String representation of the abstract token	None	Windows	None	character
type	int	Value representing the reformatting strategy	Public	Windows	None	{1..9}
TWO	int	Constant for value 2	Private and final	Windows	None	2
TWOSPACE	String	Value representing 2 blank spaces	Private and final	Windows	None	character
value	int	Value representing the indentation level	None	Windows	None	{1..9}
value	String	Value representing the indentation value	None	Windows	None	character
variableImage	TokenCollection	Collection object representing the vector of token images	Private	Windows	None	TokenCollection
variableImageIterator	TokenIterator	Iterator object containing token images	Public	Windows	None	TokenIterator
variableImageRequired	String	Value representing the token image	Private	Windows	None	character
variableIndexValue	int	Value representing the index of a variable in the iterator	Private	Windows	None	{1..9}

variableName	String	Value representing the last part of a method name	Public	Windows	None	character
variableRequiredKind	int	Value representing the kind of token	Private	Windows	None	{1..9}
vectorCreated	Vector	Vector object to set a collection	None	Windows	None	Vector
vectorLocation	int	Index in the vector to set	None	Windows	None	{1..9}
vectorToken	Token	Token object representing code	Private	Windows	None	Token
WINDOWHEIGHT	int	Constant for value for the window heightb	Private and Final	Windows	None	600
WindowMenu	Class	Class used to create menus for the GUI	Public. Needs JMenu class. Extends JMenu	Windows	None	windowMenuCaption +windowMenuMnemonic
WindowMenuBarView	Class	Class used to create the menu bar view of the GUI	Public. Needs swing class. Extends JMenuBar	Windows	None	openMenuItem + closeMenuItem + exitMenuItem + saveMenuItem + selectLanguageParserMenuItem + parseDataMenuItem + compileMenuItem + encapsulateFieldsMenuItem + replaceMagicNumberWithSymbolicConstantMenuItem + getOpenMenuItem + getCloseMenuItem + getExitMenuItem + getSaveMenuItem + getSelectLanguageParserMenuItem + getParseDataMenuItem + getCompileMenuItem + getEncapsulateFieldsMenuItem + getReplaceMagicNumberWithSymbolicConstantMenuItem
windowMenuCaption	String	Value representing the menu caption	Public	Windows	None	character
WindowMenuItem	Class	Class used to create menu items for a menu	Public. Needs JMenuItem class. Extends JMenuItem	Windows	None	menuItemName + menuMnemonic
windowMenuMnemonic	char	Value representing the mnemonic of the menu	Public	Windows	None	{a..z | A..Z}
WINDOWWIDTH	int	Constant for value for the window width	Private and Final	Windows	None	800
ZERO	int	Constant for value 0	Private and final	Windows	None	0

4.0 Component Detail Design Description

4.1 AbstractToken Class

2.2.16 4.1.1 Structure

SuperClass: None
Type: Public
SubClass: None
Implementation: JavaParserConstants
Constructor: None

2.2.17 4.1.2 Function

public void setToken(Token tokenToSet)
public Token getToken()
public String getImage()
public int getKind()
public void setImage(String tokenImage)
public void setKind(int tokenKind)
public boolean isPublic()
public boolean isClass()
public boolean isVoid()
public boolean isAbstract()
public boolean isLeftParen()
public boolean isAssign()
public boolean isGreaterThan()
public boolean isLessThan()
public boolean isEqual()
public boolean isNotEqual()
public boolean isGreaterEqual()
public boolean isLessEqual()
public boolean isIncrement()
public boolean isDecrement()
public boolean isPlus()
public boolean isMinus()
public boolean isStar()
public boolean isSlash()
public boolean isPlusAssign()
public boolean isMinusAssign()
public boolean isStarAssign()
public boolean isSlashAssign()
public boolean isLeftBrace()
public boolean isSemiColon()
public boolean isANumber()
public boolean isAVariableDeclaration(AbstractToken nextToken)
public boolean isArithmeticOperator ()
public boolean isAssignmentOperator()
public boolean isOperatorFollowedByNumber(AbstractToken nextToken)
public boolean isNotAbstractClassVoid()
public boolean isFirstLeftBrace(boolean firstLeftBrace)
public void setTokenCharacter(char firstCharacter)

2.2.18 4.1.3 Interfaces

import CIS5915JavaParser.JavaParserConstants

2.2.19 4.1.4 Application Data

private Token vectorToken

private Character tokenCharacter
private final int ZERO = 0

2.2.20 4.1.5 Detailed Design Description

4.1.5.1 setToken Method

Type: Public
Input Parameters: Token tokenToSet
Output Parameters: None
Description: set the value of vectorToken to tokenToSet

4.1.5.2 getToken Method

Type: Public
Input Parameters: None
Output Parameters: Token
Description: returns vectorToken

4.1.5.3 getImage Method

Type: Public
Input Parameters: None
Output Parameters: String
Description: returns the image value of vectorToken

4.1.5.4 getKind Method

Type: Public
Input Parameters: None
Output Parameters: int
Description: returns kind value of vectorToken

4.1.5.5 setImage Method

Type: Public
Input Parameters: String tokenImage
Output Parameters: None
Description: sets the image value of vectorToken

4.1.5.6 setKind Method

Type: Public
Input Parameters: int tokenKind
Output Parameters: None
Description: sets the kind value of vectorToken

4.1.5.7 isPublic Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is the Public reserved word

4.1.5.8 isClass Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is the Class reserved word

4.1.5.9 isVoid Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is the Void reserved word

4.1.5.10 isAbstract Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is the Abstract reserved word

4.1.5.11 isLeftParen Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is a left paren

4.1.5.12 isAssign Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is the assign operator

4.1.5.13 isGreaterThan Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is the greater than operator

4.1.5.14 isLessThan Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is the less than operator

4.1.5.15 isEqual Method

Type: Public
Input Parameters: None
Output Parameters: boolean

Description: returns true if the token is the equal operator

4.1.5.16 isNotEqual Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is the not equal operator

4.1.5.17 isGreaterEqual Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is the greater than equal operator

4.1.5.18 isLessEqual Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is the less than equal operator

4.1.5.19 isIncrement Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is the increment operator

4.1.5.20 isDecrement Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is the decrement operator

4.1.5.21 isPlus Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is the plus operator

4.1.5.22 isMinus Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is the minus operator

4.1.5.23 isStar Method

Type: Public

Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is the star operator

4.1.5.24 isSlash Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is the slash operator

4.1.5.25 isPlusAssign Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is the plus assign operator

4.1.5.26 isMinusAssign Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is the minus assign operator

4.1.5.27 isStarAssign Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is the star assign operator

4.1.5.28 isSlashAssign Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is the slash assign operator

4.1.5.29 isLeftBrace Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is a left brace

4.1.5.30 isSemiColon Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is a semi-colon

4.1.5.31 isANumber Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is a number

4.1.5.32 isAVariableDeclaration Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the current and the next tokens forms a variable declaration

4.1.5.33 isArithmeticOperator Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is an arithmetic operator

4.1.5.34 isAssignmentOperator Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is an assignment operator

4.1.5.35 isOperatorFollowedByNumber Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is an arithmetic operator and is followed by a nymber

4.1.5.36 isNotAbstractClassVoid Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the next token is not the reserved word Class, Abstract or Void

4.1.5.37 isFirstLeftBrace Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns true if the token is the first left brace encountered

4.1.5.38 setTokenCharacter Method

Type: Public

Input Parameters: char firstCharacter
Output Parameters: None
Description: sets the value of the first character of the token image

4.2 EncapsulateFields Class

2.2.21 4.2.1 Structure

SuperClass: RefactoringStrategy
Type: Public
SubClass: None
Implementation: None
Constructor: None

2.2.22 4.2.2 Function

public TokenIterator execute(TokenIterator codeTokenIterator)

2.2.23 4.2.3 Interfaces

import javax.swing.JOptionPane
import CIS5915JavaParser.AbstractTokenh

2.2.24 4.2.4 Application Data

private AbstractToken currentToken, nextToken, previousToken
private AbstractToken nextSecondToken, nextThirdToken
private final int ONE = 1
private final int TWO = 2
private final int THREE = 3
private final int FIVE = 5

2.2.25 4.2.5 Detailed Design Description

4.2.5.1 execute Method

Type: Public
Input Parameters: TokenIterator codeTokenIterator
Output Parameters: TokenIterator
Description: performs the encapsulate fields refactoring transformation. It traverses the codeTokenIterator object and analyzes each entry. It searches for entries representing attributes defined as public. Then, using the attribute name, it creates a set and a get method names with the associated parameters. It creates AbstractToken objects, adds them in a TokenIterator object for later use, and changes the attribute type to *private*. After completion, it traverses again the codeTokenIterator object and searches for assignment and usage of the attributes that had set and get methods created. If it is an assignment, it replaces it with a set method and if it is a usage, it replaces it with a get method and their associated parameters. When completed, it inserts the TokenIterator containing the set, gets methods at the end of the codeTokenIterator object, and returns it.

4.2 Indent Class

2.2.26 4.3.1 Structure

SuperClass: None
Type: Public
SubClass: None
Implementation: None
Constructor: Indent

2.2.27 4.3.2 Function

public Indent()
public void setIndentLevelValue(int value)
public int getIndentLevel()
public void decreaseIndentLevel()
public void increaseIndentLevel()
public void setStringValue(String value)
public void increaseIndentStringValue()
public void setLoopCounter()
public boolean loopNotDone()
public void setIndentStringValue()
public String getIndentStringValue()

2.2.28 4.3.3 Interfaces

None

2.2.29 4.3.4 Application Data

private int indentLevelValue
private String indentStringValue
private int loopCounter
private final int ZERO = 0
private final int ONE = 1
private final String BLANK = ""
private final String TWOSPACE = " "

2.2.30 4.3.5 Detailed Design Description

4.3.5.1 Indent Constructor

Type: Public
Input Parameters: None
Output Parameters: None
Description: calls setIndentLevelValue and setStringValue to initialize the indent level and its string values to 0 and 2 spaces.

4.3.5.2 setIndentLevelValue Method

Type: Public

Input Parameters: int value
Output Parameters: None
Description: assigns to indentLevelValue the parameter value.

4.3.5.3 getIndentLevelValue Method

Type: Public
Input Parameters: None
Output Parameters: None
Description: returns indentLevelValue.

4.3.5.4 decreaseIndentLevel Method

Type: Public
Input Parameters: None
Output Parameters: None
Description: decreases by 1 indentLevelValue by calling setIndentLevelValue. It uses as parameter to setIndentLevelValue the expression getIndentLevel - ONE.

4.3.5.5 increaseIndentLevelValue Method

Type: Public
Input Parameters: None
Output Parameters: None
Description: increases by 1 indentLevelValue by calling setIndentLevelValue. It uses as parameter to setIndentLevelValue the expression getIndentLevel + ONE.

4.3.5.6 setStringValue Method

Type: Public
Input Parameters: String value
Output Parameters: None
Description: assigns to indentStringValue the parameter value.

4.3.5.7 increaseIndentStringValue Method

Type: Public
Input Parameters: None
Output Parameters: None
Description: increases by 2 spaces indentStringValue by calling setStringValue. It uses as parameter to setStringValue the expression getIndentStringValue + TWOSPACES.

4.3.5.8 setLoopCounter Method

Type: Public
Input Parameters: None
Output Parameters: None
Description: assigns to loopCounter the value of ZERO

4.3.5.9 loopNotDone Method

Type: Public
Input Parameters: None

Output Parameters: boolean
Description: returns the evaluation result of the expression loopCounter ++< getIndentLevel.

4.3.5.10 setIndentStringValue Method

Type: Public
Input Parameters: None
Output Parameters: None
Description: calls setLoopCounter to initialize the loop counter. Calls setStringValue with as parameters BLANK. It then calls increaseIndentStringValue for the indentation level required.

4.3.5.11 getIndentStringValue method

Type: Public
Input Parameters: None
Output Parameters: None
Description: returns indentStringValue

4.3 JavaDeParser Class

4.4 Structure

SuperClass: None
Type: Public
SubClass: None
Implementation: JavaParserConstants
Constructor: JavaDeParser

4.5 Function

public JavaDeParser()
public static StringBuffer deparseToken(TokenIterator codeTokenIterator)

4.6 Interfaces

import CIS5915JavaParser.JavaParserConstants
import CIS5915JavaParser.AbstractToken

4.7 Application Data

private static StringBuffer tokenToPrint
private static AbstractToken currentToken, previousToken, nextToken
private static Indent indentLevel
private static TokenContext tokenContext

4.8 Detailed Design Description

4.9 JavaDeParser Constructor

Type: Public
Input Parameters: None
Output Parameters: None
Description: creates a new Indent object, a new TokenContext object and new AbstractToken objects.

4.9.5.2 deparseToken Method

Type: Public
Input Parameters: TokenIterator codeTokenIterator
Output Parameters: StringBuffer
Description: it traverses the codeTokenIterator object and using the current AbstractToken kind, it sets the reformat strategy. It then reformats the object using the previous and next object kind and appends the resulting string to the tokenToPrint object. When completed, it returns tokenToPrint.

4.10 ObjectCollection Class

4.5.1 Structure

SuperClass: None
Type: Public and Abstract
SubClass: TokenCollection
Implementation: None
Constructor: None

4.5.2 Function

public abstract TokenIterator createCollectionIterator
public abstract void add(Object collectionObject)
public abstract void add(int index, Object collectionObject)
public abstract void removeElement(Object collectionObject)

4.5.3 Interfaces

None

4.5.4 Application Data

None

2.2.31 4.5.5 Detailed Design Description

4.5.5.1 createCollectionIterator Method

Type: Public and Abstract
Input Parameters: None
Output Parameters: TokenIterator
Description: returns a new a new collection iterator object

4.5.5.2 add Method

Type: Public and Abstract
Input Parameters: Object collectionObject
Output Parameters: None
Description: adds a collectionObject to the end of the collection

4.5.5.3 add Method

Type: Public and Abstract
Input Parameters: Object collectionObject and int index
Output Parameters: None
Description: adds a collectionObject to the collection using the index provided for its location

4.5.5.4 removeElement method

Type: Public and Abstract
Input Parameters: Object collectionObject
Output Parameters: None
Description: removes a collectionObject from the collection

4.6 ObjectIterator Class

2.2.32 4.6.1 Structure

SuperClass: None
Type Public and Abstract
SubClass: TokenIterator
Implementation: None
Constructor: None

2.2.33 4.6.2 Function

public abstract Object nextObject()
public abstract void add(Object objectToAdd, int index)
public abstract boolean isThereMoreObjects()
public abstract void setToFirst()
public abstract void setToLast()
public abstract void setVectorLocation(int vectorLocation)
public abstract int getCurrentCollectionIndex()
public abstract void insertTokens(TokenIterator insertTokenIterator)

2.2.34 4.6.3 Interfaces

None

2.2.35 4.6.4 Application Data

None

2.2.36 4.6.5 Detailed Design Description

4.6.5.1 nextObject Method

Type: Public and Abstract
Input Parameters: None
Output Parameters: Object
Description: returns the next Object in the iterator

4.6.5.2 add Method

Type: Public and Abstract
Input Parameters: Object objectToAdd and int index
Output Parameters: None
Description: adds an objectToAdd to the iterator at the index provided

4.6.5.3 isThereMoreObjects Method

Type: Public and Abstract
Input Parameters: None
Output Parameters: boolean
Description: returns if there are more objects in the iterator

4.6.5.4 setToFirst Method

Type: Public and Abstract
Input Parameters: None
Output Parameters: None
Description: points the iterator object to the first object

4.6.5.5 setToLast Method

Type: Public and Abstract
Input Parameters: None
Output Parameters: None
Description: points the iterator object to the last object

4.6.5.6 setVectorLocation Method

Type: Public and Abstract
Input Parameters: int vectorLocation
Output Parameters: None
Description: points the iterator to the location provided by vectorLocation

4.6.5.7 getCurrentCollectionIndex Method

Type: Public and Abstract
Input Parameters: None
Output Parameters: int
Description: returns the current index in the iterator

4.6.5.8 insertTokens Method

Type: Public and Abstract
Input Parameters: TokenIterator insertTokenIterator
Output Parameters: None
Description: adds to the end of the iterator the content of insertTokenIterator

4.7 ProjectFile Class

2.2.37 4.7.1 Structure

SuperClass: None
Type: Public
SubClass: None
Implementation: None
Constructor: None

2.2.38 4.7.2 Function

public StringBuffer readFile(String fileName)
public void saveFile(String refactoredCodeLine)
public void compileFile()
public String getFileName()
public void setFileName(String newFileName)

2.2.39 4.7.3 Interfaces

import java.io.*
import javax.swing.JOptionPane
import sun.tools.*

2.2.40 4.7.4 Application Data

private BufferedReader sourceCodeReader
private BufferedWriter refactoredCodeWriter
private StringBuffer sourceCodeBuffer
private String sourceCodeLineRead
private String refactoredCodeLine
private File fileToOpen, fileToWrite
private ByteArrayOutputStream compileFileBAOS
private sun.tools.javac.Main compiler
private String fileName

2.2.41 4.7.5 Detailed Design Description

4.7.5.1 readFile Method

Type: Public
Input Parameters: String fileName
Output Parameters: StringBuffer
Description: saves the file name using setFileName. Creates a BufferedReader, a sourceCodeReader, a StringBuffer, and a sourceCodeBuffer objects. It uses

sourceCodeReader to retrieve lines of text from the file fileName. It appends each line of text retrieved to sourceCodeBuffer. On completion, it closes sourceCodeReader and returns sourceCodeBuffer. If there is an error opening the file, it displays an error message.

4.7.5.2 saveFile Method

Type: Public
Input Parameters: String refactoredCodeLine
Output Parameters: None
Description: creates the file to write using the file name returned by the method getFileName. Creates a BufferedWriter, refactoredCodeWriter object. It uses refactoredCodeWriter to write line of text to the file. On completion, it closes refactoredCodeWriter. If there is an error writing to the file, it displays an error message.

4.7.5.3 compileFileMethod

Type: Public
Input Parameters: None
Output Parameters: None
Description: initializes compilation arguments. Creates a ByteArrayOutputStream, a compileFileBAOS, a sun.tools.javac.Main, and a compiler, objects. Calls the compile method of compiler. Displays a message upon successful or unsuccessful compilation.

4.7.5.4 getFileName Method

Type: Public
Input Parameters: None
Output Parameters: None
Description: returns fileName

4.7.5.5 setFileName Method

Type: Public
Input Parameters: String newFileName
Output Parameters: None
Description: assigns to fileName the value of newFileName

4.8 RefactoringContext Class

4.8.1 Structure

SuperClass: None
Type: Public
SubClass: None
Implementation: None
Constructor: None

4.8.2 Function

public void setRefactoringStrategy(String strategyToUse)
public TokenIterator execute(TokenIterator tokenIterator)

2.2.42 4.8.3 Interfaces

None

2.2.43 4.8.4 Application Data

private RefactoringStrategy refactoringStrategy

2.2.44 4.8.5 Detailed Design Description

4.8.5.1 setRefactoringStrategy Method

Type: Public
Input Parameters: String strategyToUse
Output Parameters: None
Description: creates the strategy to use for performing refactoring transformations. Use strategyToUse to determine the strategy.

4.8.5.2 execute Method

Type: Public
Input Parameters: TokenIterator tokenIterator
Output Parameters: TokenIterator
Description: executes the refactoring strategy and returns a TokenIterator created by the strategy.

4.9 RefactoringStrategy Class

2.2.45 4.9.1 Structure

SuperClass: None
Type: Public and Abstract
SubClass: EncapsulateFields and ReplaceMagicNumber
Implementation: JavaParserConstants
Constructor: None

2.2.46 4.9.2 Function

public abstract TokenIterator execute(TokenIterator tokenIterator)
public void setVariableIndexValue()
public int getVariableIndexValue()
public void setFirstLeftBrace(boolean status)
public boolean getFirstLeftBrace()
public void setInsertLocation(TokenIterator codeTokenIterator, boolean firstLeftBraceFound)
public void setRefactoredVariableName(String tokenValue)
public String getRefactoredVariableName()
public void setVariableImageRequired(String tokenImage)
public String getVariableImageRequired()
public void setVariableRequiredKind(int tokenKind)
public void saveVariableImageRequired(String tokenImage)

public int getVariableRequiredKind()
public boolean isTokenWanted(String tokenImage)
public void findSetFirstVariableOccurence(TokenIterator codeTokenIterator, String previousTokenImage)
public void setMagicNumberTokenVector(int nextTokenKind, String nextTokenImage)
public Vector getInsertTokenVector()
public void createMethodName(String partOne, String variableName)
public String getMethodName()
public TokenIterator getInsertCodeCollectionIterator()
public void createSetMethodTokenVector(int nextTokenKind, String nextTokenImage, int nextSecondKind, String nextSecondImage)
public void createGetMethodTokenVector(int nextTokenKind, String nextTokenImage, int nextSecondKind, String nextSecondImage)

2.2.47 4.9.3 Interfaces

import CIS5915JavaParser.JavaParserConstants
import CIS5915JavaParser.AbstractToken
import java.util.Vector

2.2.48 4.9.4 Application Data

private final int ZERO = 0
private final int ONE = 1
private final int TWO = 2
private int variableIndexValue
private int variableRequiredKind
private boolean firstLeftBrace
private String refactoredVariableName, variableImageRequired, methodName
private AbstractToken previous, current
private Character tokenCharacter
protected TokenFactory tokenFactory
private Vector insertTokenVector
private TokenCollection insertCodeCollection
private TokenCollection variableImage
private TokenIterator variableImageIterator

2.2.49 4.9.5 Detailed Design Description

4.9.5.1 RefactoringStrategy Constructor

Type: Public
Input Parameters: None
Output Parameters: None
Description: creates a new Vector object to hold the AbstractToken objects to be inserted. Creates a new TokenFactory object for creating the AbstractToken objects. Creates a new TokenCollection object for use with the Vector object. Creates two AbstractToken objects.

4.9.5.2 execute Method

Type: Public and Abstract
Input Parameters: TokenIterator tokenIterator
Output Parameters: TokenIterator
Description: performs the refactoring strategy selected using tokenIterator and returns an updated TokenIterator object to be reformatted.

4.9.5.3 setVariableIndexValue Method

Type: Public
Input Parameters: None
Output Parameters: None
Description: assigns the constant ZERO to variableIndexValue.

4.9.5.4 getVariableIndexValue Method

Type: Public
Input Parameters: None
Output Parameters: int
Description: returns variableIndexValue and increments it by 1.

4.9.5.5 setFirstLeftBrace Method

Type: Public
Input Parameters: boolean status
Output Parameters: None
Description: assigns status to firstLeftBrace.

4.9.5.6 getFirstLeftBrace Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns firstLeftBrace.

4.9.5.7 setInsertLocation Method

Type: Public
Input Parameters: TokenIterator codeTokenIterator and boolean firstLeftBraceFound
Output Parameters: None
Description: verifies if firstLeftBraceFound is true or not. If it is true, it sets the location of the insertion point by calling the setInsertVectorLocation method of codeTokenIterator and calls setFirstLeftBrace with as parameter the value false. Otherwise, it does nothing.

4.9.5.8 setRefactoredVariableName Method

Type: Public
Input Parameters: String tokenValue
Output Parameters: None
Description: assigns to refactoredVariableName the expression "Refactored" and the value returned by getVariableIndexValue.

4.9.5.9 getRefactoredVariableName Method

Type: Public
Input Parameters: None
Output Parameters: String
Description: returns refactoredVariableName

4.9.5.10 setVariableImageRequired Method

Type: Public
Input Parameters: String tokenImage
Output Parameters: None
Description: assigns tokenImage to variableImageRequired.

4.9.5.11 getVariableImageRequired Method

Type: Public
Input Parameters: None
Output Parameters: String
Description: returns variableImageRequired

4.9.5.12 setVariableRequiredKind Method

Type: Public
Input Parameters: int tokenKind
Output Parameters: None
Description: assigns tokenKind to variableRequiredKind.

4.9.5.13 saveVariableImageRequired Method

Type: Public
Input Parameters: String tokenImage
Output Parameters: None
Description: saves in a vector the token image for later usage

4.9.5.14 getVariableRequiredKind Method

Type: Public
Input Parameters: None
Output Parameters: int
Description: returns variableRequiredKind

4.9.5.15 isTokenWanted Method

Type: Public
Input Parameters: String tokenImage
Output Parameters: boolean
Description: traverses the vector of token images and determines if he input parameter is contained in the vector. It returns true if it is.

4.9.5.16 findSetFirstVariableOccurence Method

Type: Public
Input Parameters: TokenIterator codeTokenIterator and String previousImage

Output Parameters: None
Description: searches codeTokenIterator for the first occurrence of previousImage. When it is found, it calls setVariableImageRequired with as parameter the expression previous.getImage and calls setVariableRequiredKind with as parameter the expression previous.getKind.

4.9.5.17 setMagicNumberTokenVector Method

Type: Public
Input Parameters: int nextTokenKind and String nextTokenImage
Output Parameters: None
Description: creates and adds AbstractToken objects required for the replace magic number refactoring to insertTokenVector. The objects created are those required for the definition of a constant, which are: PRIVATE, FINAL, the type of the constant, the constant name, ASSIGN, the object value, and SEMICOLON.

4.9.5.18 getInsertTokenVector Method

Type: Public
Input Parameters: None
Output Parameters: Vector
Description: returns insertTokenVector

4.9.5.19 createMethodName Method

Type: Public
Input Parameters: String partOne and String variableName
Output Parameters: None
Description: creates a method name for the encapsulate fields refactoring. The method name consists of partOne, "Refactored", and variableName with its first character in upper case.

4.9.5.20 getMethodName Method

Type: Public
Input Parameters: None
Output Parameters: String
Description: returns methodName

4.9.5.21 getInsertCodeCollectionIterator Method

Type: Public
Input Parameters: None
Output Parameters: TokenIterator
Description: returns the TokenIterator object associated with insertCodeCollection using insertTokenVector

4.9.5.22 createSetMethodTokenVector Method

Type: Public
Input Parameters: int nextTokenKind, String nextTokenImage, int nextSecondKind and String nextSecondImage

Output Parameters: None
Description: creates and adds the set method AbstractToken objects required for the encapsulate fields refactoring to insertTokenVector. The objects created are: SINGLE_LINE_COMMENT, PUBLIC, VOID, methodName, LPAREN, the type of variable, the parameter name, RPAREN, LBRACE, the variable name, ASSIGN, the parameter name, SEMICOLON, RBRACE, SINGLE_LINE_COMMENT.

4.9.5.23 createGetMethodTokenVector Method

Type: Public
Input Parameters: int nextTokenKind, String nextTokenImage, int nextSecondKind and String nextSecondImage
Output Parameters: None
Description: creates and adds the get method AbstractToken objects required for the encapsulate fields refactoring to insertTokenVector. The objects created are: SINGLE_LINE_COMMENT, PUBLIC, the type of variable, methodName, LPAREN, RPAREN, LBRACE, RETURN, the variable name, SEMICOLON, RBRACE, SINGLE_LINE_COMMENT.

4.10 RefactoringTool Class

2.2.50 4.10.1 Structure

SuperClass: JFrame
Type: Public
SubClass: None
Implementation: None
Constructor: RefactoringTool()

2.2.51 4.10.2 Function

public void setOpenFileMenuItem()
public void setSaveFileMenuItem()
public void setCloseFileMenuItem()
public void setExitMenuItem()
public void setCompileMenuItem()
public void setSelectLanguageParserMenuItem()
public void setParseDataMenuItem()
public void setEncapsulateFieldsMenuItem()
public void setReplaceMagicNumberWithSymbolicConstantMenuItem()
private static void main (String args[])

2.2.52 4.10.3 Interfaces

import javax.swing.*
import java.awt.*
import java.awt.event.*
import java.util.Vector
import CIS5915JavaParser.*

2.2.53 4.10.4 Application Data

private Box displayBox
private WindowMenuBarView projectMenuBar
private SourceCodeView sourceCodeDisplay
private SourceCodeView refactoredCodeDisplay
private ProjectFile sourceCodeFile
private ProjectFile refactoredCodeFile
private JFileChooser fileChooserWindow
private int returnedValue
private JMenuItem exitMenuItem
private JMenuItem compileMenuItem
private JMenuItem openFileMenuItem
private JMenuItem closeFileMenuItem
private JMenuItem saveFileMenuItem
private JMenuItem selectLanguageParserMenuItem
private JMenuItem parseDataMenuItem
private JMenuItem encapsulateFieldsMenuItem
private JMenuItem replaceMagicNumberWithSymbolicConstantMenuItem
private TokenCollection codeTokenCollection
protected static JavaDeParser projectDeParser
private RefactoringContext refactoringContext
private final int WINDOWWIDTH = 800
private final int WINDOWHEIGHT = 600

2.2.54 4.10.5 Detailed Design Description

4.10.5.1 RefactoringTool Constructor

Type: Public
Input Parameters: None
Output Parameters: None
Description: creates and adds to the GUI projectMenuBar, displayBox, sourceCodeDisplay, and refactoredCodeDisplay. The latter two are SourceCodeView objects included in JScrollPane objects. Creates the input and Output file objects sourceCodeFile and refactoredCodeFile. Creates TokenCollection, JavaDeParser and RefactoringContext objects. Calls each menu items of the GUI to enable their action listener. Sets the GUI window size and make the GUI visible.

4.10.5.2 setOpenFileMenuItem Method

Type: Public
Input Parameters: None
Output Parameters: None
Description: displays the menu item for opening a file. Defines the code for implementing the action listener. When the user selects this menu item, a file chooser window will be displayed from which the user selects the file to open. The content of the file selected is displayed using the method displayText of sourceCodeDisplay and the expression sourceCodeFile.readFile(fileChooserWindow.getSelectedFile.getPath) and

calls the setFileName method of refactoredCodeFile using as parameter the expression fileChooserWindow.getSelectedFile.getPath

4.10.5.3 setSaveFileMenuItem Method

Type: Public
Input Parameters: None
Output Parameters: None
Description: displays the menu item for saving file. Defines the code for implementing the action listener. When the user selects this menu item, a file chooser window will be displayed from which the user selects the file to save. The content of the file selected is saved using the method saveFile of refactoredCodeFile using as parameter the expression refactoredCodeDisplay.getText)

4.10.5.4 setCloseFileMenuItem Method

Type: Public
Input Parameters: None
Output Parameters: None
Description: displays the menu item for closing a file. Defines the code for implementing the action listener. When the user selects this menu item, the clearDisplay method of sourceCodeDisplay and refactoredCodeDisplay is executed.

4.10.5.5 setExitMenuItem Method

Type: Public
Input Parameters: None
Output Parameters: None
Description: displays the menu item for exiting. Defines the code for implementing the action listener. When the user selects this menu item, the exit method of the System object is executed with as parameter the value 0.

4.10.5.6 setCompileMenuItem Method

Type: Public
Input Parameters: None
Output Parameters: None
Description: displays the menu item for compiling a file. Defines the code for implementing the action listener. When the user selects this menu item, the compileFile method of refactoredCodeFile is executed

4.10.5.7 setSelectLanguageParserMenuItem Method

Type: Public
Input Parameters: None
Output Parameters: None
Description: displays the menu item for saving file. Defines the code for implementing the action listener. When the user selects this menu item, a window indicating the menu selected is displayed. As language parser becomes available for other language than for Java, this option can be modified easily.

4.10.5.8 setParseDataMenuItem Method

Type: Public
Input Parameters: None
Output Parameters: None
Description: displays the menu item for saving file. Defines the code for implementing the action listener. When the user selects this menu item, the parseData method of the JavaDeParser object is executed with as parameter the opened file. The returned vector is then used as the parameter for the setCollection method of the codeTokenCollection object. The deparseToken method of the projectDeParser is executed with as parameter the expression codeTokenCollection.createCollectionIterator. The resulting TokenIterator object is used as parameter for the displayText method of the refactoredCodeDisplay object.

4.10.5.9 setEncapsulateFieldsMenuItem Method

Type: Public
Input Parameters: None
Output Parameters: None
Description: displays the menu item for saving file. Defines the code for implementing the action listener. When the user selects this menu item, the clearDisplay method of the refactoredCodeDisplay object is executed. Then, it executes the setRefactoringStrategy method of the refactoringContext object to set the refactoring strategy. Finally, the execute method of the refactoringContext is executed with as parameter the expression codeTokenCollection.createCollectionIterator. The resulting TokenIterator is then used as parameter to the deparseToken method of the projectDeParser object. The ensuing TokenIterator is used as parameter for the displayText method of the refactoredCodeDisplay object.

4.10.5.10 setReplaceMagicNumberWithSymbolicConstantMenuItem Method

Type: Public
Input Parameters: None
Output Parameters: None
Description: displays the menu item for saving file. Defines the code for implementing the action listener. When the user selects this menu item, the clearDisplay method of the refactoredCodeDisplay object is executed. Then, it executes the setRefactoringStrategy method of the refactoringContext object to set the refactoring strategy. Finally, the execute method of the refactoringContext is executed with as parameter the expression codeTokenCollection.createCollectionIterator. The resulting TokenIterator is then used as parameter to the deparseToken method of the projectDeParser object. The ensuing TokenIterator is used as parameter for the displayText method of the refactoredCodeDisplay object.

4.10.5.11 main Method

Type: Public and Static
Input Parameters: String args[]
Output Parameters: None

Description: creates a new application object by instantiating a new RefactoringTool object. Sets the default close operation by calling the setDefaultCloseOperation with as parameter the EXIT_ON_CLOSE constant of the JFrame object.

4.11 ReplaceMagicNumber Class

2.2.55 4.11.1 Structure

SuperClass: RefactoringStrategy
Type: Public
SubClass: None
Implementation: None
Constructor: None

2.2.56 4.11.2 Function

public TokenIterator execute(TokenIterator codeTokenIterator)

2.2.57 4.11.3 Interfaces

import javax.swing.JOptionPane
import CIS5915JavaParser.AbstractToken

2.2.58 4.11.4 Application Data

private AbstractToken currentToken, nextToken, previousToken

2.2.59 4.11.5 Detailed Design Description

4.11.5.1 execute Method

Type: Public
Input Parameters:
Output Parameters:
Description: performs the replace magic number with symbolic constant refactoring transformation. It traverses the codeTokenIterator object and analyzes each entry. It searches for entries representing an arithmetic operator followed by a number. When one is found, it creates a new variable name and set the location for insertion after the first left brace. It then fins the type of variable it is and creates the TokenIterator containing the AbstractToken objects representing the new variable. Finally, it saves the new image for the AbstractToken and inserts the TokenIterator contents in the codeTokenIterator following the first left brace.

4.12 SourceCodeView Class

2.2.60 4.12.1 Structure

SuperClass: JTextArea
Type: Public
SubClass: None

Implementation: None
Constructor: SourceCodeView()

2.2.61 4.12.2 Function

public void displayText(StringBuffer sourceCodeBuffer)
public void clearDisplay()

2.2.62 4.12.3 Interfaces

import javax.swing.*
import java.io.*

2.2.63 4.12.4 Application Data

private JTextArea sourceCode
private final int NUMBEROFROWS = 35
private final int NUMBEROFCOLUMNS = 50

2.2.64 4.12.5 Detailed Design Description

4.12.5.1 SourceCodeView Constructor

Type: Public
Input Parameters: None
Output Parameters: None
Description: creates a new JTextArea object of NUMBEROFROWS and NUMBEROFCOLUMNS.

4.12.5.2 displayText Method

Type: Public
Input Parameters: StringBuffer sourceCodeBuffer
Output Parameters: None
Description: executes the append method using as parameter the expression sourceCodeBuffer.toString.

4.12.5.3 clearDisplayMethod

Type: Public
Input Parameters: None
Output Parameters: None
Description: executes the setText method with the expression "" as parameter

4.13 TokenCollection Class

2.2.65 4.13.1 Structure

SuperClass: ObjectCollection
Type: Public
SubClass: None
Implementation: None

Constructor: TokenCollection()

2.2.66 4.13.2 Function

public TokenIterator createCollectionIterator()
public void add(Object collectionObject)
public void add(int index, Object collectionObject)
public void removeElement(Object collectionObject)
public Object get(int objectIndex)
public void setCollection(Vector vectorCreated)

2.2.67 4.13.3 Interfaces

import java.util.Vector

2.2.68 4.13.4 Application Data

protected Vector collectionVector

2.2.69 4.13.5 Detailed Design Description

4.13.5.1 TokenCollection Constructor

Type: Public
Input Parameters: None
Output Parameters: None
Description: creates a new Vector object

4.13.5.2 createCollectionIterator Method

Type: Public
Input Parameters: None
Output Parameters: TokenIterator
Description: returns a new TokenIteraotr object with as parameter the reserved word this.

4.13.5.3 add Method

Type: Public
Input Parameters: Object collectionObject
Output Parameters: None
Description: adds collectionObject to the Vector object by executing the add method

4.13.5.4 add Method

Type: Public
Input Parameters: int index and Object collectionObject
Output Parameters: None
Description: adds collectionObject to the Vector at the location provided by index.

4.13.5.5 removeElement Method

Type: Public
Input Parameters: Object collectionObject

Output Parameters: None
Description: executes the removeElement method with as parameter collectionObject.

4.13.5.6 get Method

Type: Public
Input Parameters: int objectIndex
Output Parameters: Object
Description: returns the object at location objectIndex from the Vector object.

4.13.5.7 setCollection Method Method

Type: Public
Input Parameters: vector vectorCreated
Output Parameters: None
Description: sets vectorCreated as a collection object.

4.14 TokenContext Class

2.2.70 4.14.1 Structure

SuperClass: None
Type: Public
SubClass: None
Implementation: JavaParserConstants
Constructor: None

2.2.71 4.14.2 Function

public String reformat(String currentTokenImage, int nextTokenKind, int previousTokenKind, Indent indentLevel)
public void setReformatStrategy(int type)

2.2.72 4.14.3 Interfaces

import CIS5915JavaParser.JavaParserConstants

2.2.73 4.14.4 Application Data

private TokenReformat reformatStrategy

2.2.74 4.14.5 Detailed Design Description

4.14.5.1 TokenContext Constructor

Type: Public
Input Parameter: None
Output Parameter: None
Description: sets the reformat strategy to the default by executing the setreformatStrategy with as parameter DEFAULT.

4.14.5.2 setReformatStrategy Method

Type: Public
Input Parameter: int type
Output Parameter: None
Description: sets the reformat strategy using the value provided by type. Possible values are: L:BRACE, RBRACE, SEMICOLON, SINGLE_LINE_COMMENT, MULTI_LINE_COMMENT, FORMAL_COMMENT, RPAREN, DOT, LPAREN, RBRACKET, LBRACKET, BANG and default.

4.14.5.3 reformat Method

Type: Public
Input Parameter: String currentTokenImage, int nextTokenKind, int previousTokenKind and Indent indentLevel
Output Parameter: String
Description: returns the result of executing the reformat method of the refactoringStrategy object using as parameters currentTokenImage, nextTokenKind, previousTokenKind and indentLevel.

4.15 TokenCreator Class

2.2.75 4.15.1 Structure

SuperClass: None
Type: Public and Abstract
SubClass: TokenFactory
Implementation: CIS5915JavaParser.JavaParserConstants
Constructor: None

4.15.2 Function

createToken(int tokenKind, String tokenImage)
getFactory(int objectType

2.2.76 4.15.3 Interfaces

import CIS5915JavaParser.JavaParserConstants
import CIS5915JavaParser.AbstractToken

2.2.77 4.15.4 Application Data

private static final TokenFactory tokenCreator
protected static final int TOKEN = 100

2.2.78 4.15.5 Detailed Design Description

4.15.5.1 createToken Method

Type: Public and Abstract
Input Parameters: int tokenKind and String tokenImage
Output Parameters: AbstractToken

Description: creates AbstractToken objects using the tokenKind and tokenImage parameters.

4.15.5.2 getFactory Method

Type: Static and Final
Input Parameters: int objectType
OutputParameter: TokenCreator
Decription: returns the proper factory object based on objectType.

4.16 TokenDefault Class

2.2.79 4.16.1 Structure

SuperClass: TokenReformat
Type: Public
SubClass: None
Implementation: None
Constructor: None

2.2.80 4.16.2 Function

public String reformat(String currentTokenImage, int nextTokenKind, int previousTokenKind, Indent indentLevel)

2.2.81 4.16.3 Interfaces

None

2.2.82 4.16.4 Application Data

None

2.2.83 4.16.5 Detailed Design Description

4.16.5.1 reformat Method

Type: Public
Input Parameters: String currentTokenImage, int nextTokenKind, int previousTokenKind and Indent indentLevel
Output Parameters: String
Description: it verifies that nextTokenKind is not a dot and not a semi-colon and not a right parenthesis and not a left parenthesis and previousTokenKind is not a dot. If nextTokenKind is not a dot and not a semi-colon and not a right parenthesis and not a left parenthesis and previousTokenKind is not a dot, it returns currentTokenImage and a space. Otherwise, it returns currentTokenImage.

4.17 TokenFactory Class

2.2.84 4.17.1 Structure

SuperClass: AbstractFactory
Type: Public
SubClass: None
Implementation: None
Constructor: None

2.2.85 4.17.2 Function

public AbstractToken createToken(int tokenKind, String tokenImage)

2.2.86 4.17.3 Interfaces

import CIS5915JavaParser.Token
import CIS5915JavaParser.AbstractToken

2.2.87 4.17.4 Application Data

private AbstractToken newToken

2.2.88 4.17.5 Detailed DesignDescription

4.17.5.1 createToken Method

Type: Public
Input Parameters: int tokenKind and String TokenImage
Output Parameters: AbstractToken
Description: returns a new AbstractToken object with its attributes Image and Kind sets to tokenImage and tokenKind.

4.18 TokenIterator Class

2.2.89 4.18.1 Structure

SuperClass: ObjectIterator
Type: Public
SubClass: None
Implementation: None
Constructor: TokenIterator(TokenCollection newCollection)

2.2.90 4.18.2 Function

public void setToFirst()
public void setToLast()
public void setVectorLocation(int vectorLocation)
public int getCurrentCollectionIndex()
public void insertTokens(TokenIterator insertTokenIterator)
public void add(Object objectToAdd, int index)

public void add(Object objectToAdd)
public Object nextObject()
public boolean isThereMoreObjects()
public void setCurrentVectorLocation()
public int getCurrentVectorLocation()
public void setInsertVectorLocation()
public int getInsertVectorLocation()

2.2.91 4.18.3 Interfaces

import CIS5915JavaParser.Token

2.2.92 4.18.4 Application Data

private final int NOSELECTION = -1
private final int ONE = 1
private final int FOUR = 4
private TokenCollection tokenCollection
private int currentCollectionIndex = NOSELECTION
private int currentVectorLocation, insertVectorLocation

2.2.93 4.18.5 Detailed Design Description

4.18.5.1 TokenIterator Constructor

Type: Public
Input Parameters: TokenCollection newCollection
Output Parameters: None
Description: assigns newCollection to tokenCollection.

4.18.5.2 setToFirst Method

Type: Public
Input Parameters: None
Output Parameters: None
Description: assigns NOSELECTION to currentCollectionIndex.

4.18.5.3 setToLast Method

Type: Public
Input Parameters: None
Output Parameters: None
Description: assigns the expression tokenCollection.collectionVector.size - FOUR to currentCollectionIndex.

4.18.5.4 setVectorLocation Method

Type: Public
Input Parameters: int vectorLocation
Output Parameters: None
Description: assigns vectorLocation to currentCollectionIndex.

4.18.5.5 getCurrentCollectionIndex Method

Type: Public
Input Parameters: None
Output Parameters: int
Description: returns currentCollectionIndex.

4.18.5.6 insertTokens Method

Type: Public
Input Parameters: TokenIterator insertTokenIterator
Output Parameters: None
Description: loops through insertTokenIterator and adds each object to TokenIterator and increases currentCollectionIndex.

4.18.5.7 add Method

Type: Public
Input Parameters: Object objectToAdd and int index
Output Parameters: None
Description: executes the add method of tokenCollection using as parameters objectToAdd and index.

4.18.5.8 add Method

Type: Public
Input Parameters: Object objectToAdd
Output Parameters: None
Description: executes the add method of tokenCollection using as poartameter objectToAdd.

4.18.5.9 nextObject Method

Type: Public
Input Parameters: None
Output Parameters: Object
Description: returns the next object of the collection by executing the expression tokenCollection.collectionVector.elementAt(++currentCollectionindex).

4.18.5.10 isThereMoreObjects Method

Type: Public
Input Parameters: None
Output Parameters: boolean
Description: returns the results of the expression currentCollectionIndex >= tokenCollection.collectionVector.size-ONE.

4.18.5.11 setCurrentVectorLocation Method

Type: Public
Input Parameters: None
Output Parameters: None

Description: assigns the result of executing the method getCurrentCollectionIndex to currentVectorLocation.

4.18.5.12 getCurrentVectorLocation Method

Type: Public
Input Parameters: None
Output Parameters: int
Description: returns currentVectorLocation.

4.18.5.13 setInsertVectorLocation Method

Type: Public
Input Parameters: None
Output Parameters: None
Description: assigns the result of executing the method getCurrentCollectionIndex to insertVectorLocation

4.18.5.14 getInsertVectorLocation Method

Type: Public
Input Parameters: None
Output Parameters: int
Description: returns insertVectorLocation.

4.19 TokenLBrace Class

2.2.94 4.19.1 Structure

SuperClass: TokenReformat
Type: Public
SubClass: None
Implementation: None
Constructor: None

2.2.95 4.19.2 Function

public String reformat(String currentTokenImage, int nextTokenKind, int previousTokenKind, Indent indentLevel)

2.2.96 4.19.3 Interfaces

None

4.19.4 Application Data

None

2.2.97 4.19.5 Detailed Design Description

4.19.5.1 reformat Method

Type: Public

Input Parameters: String currentTokenImage, int nextTokenKind, int previousTokenKind and Indent indentLevel
Output Parameters: String
Description: increases the indentation level and resets the indentation value. It verifies that nextTokenKind is not a single line comment. If it is not a single line comment, it verifies if nextTokenKind is not a right brace. If it is not a right brace, it returns currentTokenImage, a new line character and the indentation value. If it is a right brace, it returns currentTokenImage and a new line character. Otherwise, it returns returns currentTokenImage and a space.

4.20 TokenMultiLineComment Class

2.2.98 4.20.1 Structure

SuperClass: TokenReformat
Type: Public
SubClass: None
Implementation: None
Constructor: None

2.2.99 4.20.2 Function

public String reformat(String currentTokenImage, int nextTokenKind, int previousTokenKind, Indent indentLevel)

2.2.1004.20.3 Interfaces

None

4.20.4 Application Data

None

2.2.1014.20.5 Detailed Design Description

4.20.5.1 reformat Method

Type: Public
Input Parameters: String currentTokenImage, int nextTokenKind, int previousTokenKind and Indent indentLevel
Output Parameters: String
Description: it verifies that nextTokenKind is not a right brace. If it is not a right brace, it returns currentTokenImage, two new line characters and the indentation value. Otherwise, it returns currentTokenImage and two new line characters.

4.21 TokenParenBracketDotBang Class

2.2.1024.21.1 Structure

SuperClass: TokenReformat

Type: Public
SubClass: None
Implementation: None
Constructor: None

2.2.1034.21.2 Function

public String reformat(String currentTokenImage, int nextTokenKind, int previousTokenKind, Indent indentLevel)

2.2.1044.21.3 Interfaces

None

2.2.1054.21.4 Application Data

None

4.21.5 Detailed Design Description

4.21.5.1 reformat Method

Type: Public
Input Parameters: String currentTokenImage, int nextTokenKind, int previousTokenKind and Indent indentLevel
Output Parameters: String
Description: it verifies that nextTokenKind is not a left brace. If it is not a left brace, it returns currentTokenImage. Otherwise, it returns currentTokenImage and a space.

4.22 TokenRBrace Class

2.2.1064.22.1 Structure

SuperClass: TokenReformat
Type: Public
SubClass: None
Implementation: None
Constructor: None

2.2.1074.22.2 Function

public String reformat(String currentTokenImage, int nextTokenKind, int previousTokenKind, Indent indentLevel)

2.2.1084.22.3 Interfaces

None

2.2.1094.22.4 Application Data

None

4.22.5 Detailed Design Description

4.22.5.1 reformat Method

Type: Public
Input Parameters: String currentTokenImage, int nextTokenKind, int previousTokenKind and Indent indentLevel
Output Parameters: String
Description: decreases the indentation level and resets the indentation value. It verifies that nextTokenKind is not a single line comment. If it is not a single line comment, it returns the indentation value, currentTokenImage and a new line character. Otherwise, it returns returns the indentation value, currentTokenImage and a space.

4.23 TokenReformat Class

2.2.1104.23.1 Structure

SuperClass: TokenReformat
Type: Public and Abstract
SubClass: None
Implementation: None
Constructor: None

2.2.1114.23.2 Function

public String reformat(String currentTokenImage, int nextTokenKind, int previousTokenKind, Indent indentLevel)

2.2.1124.23.3 Interfaces

import CIS5915JavaParser.JavaParserConstants

2.2.1134.23.4 Application Data

None

4.23.5 Detailed Design Description

4.23.5.1 reformat Method

Type: Public
Input Parameters: String currentTokenImage, int nextTokenKind, int previousTokenKind and Indent indentLevel
Output Parameters: String
Description: returns a string consisting of currentTokenImage and/or a new line character and/or spaces and/or indentation value based on nextTokenKind and previousTokenKind.

4.24 TokenSemiColon Class

2.2.1144.24.1 Structure

SuperClass: TokenReformat
Type: Public
SubClass: None
Implementation: None
Constructor: None

2.2.1154.24.2 Function

public String reformat(String currentTokenImage, int nextTokenKind, int previousTokenKind, Indent indentLevel)

2.2.1164.24.3 Interfaces

None

4.24.4 Application Data

None

2.2.1174.24.5 Detailed Design Description

4.24.5.1 reformat Method

Type: Public
Input Parameters: String currentTokenImage, int nextTokenKind, int previousTokenKind and Indent indentLevel
Output Parameters: String
Description: verifies that nextTokenKind is not a single line comment. If it is not, it verifies that nextTokenKind is not a right brace. If it is not a right brace, it returns currentTokenImage, a new line character and the current indentation. If it is a right brace, it returns currentTokenImage and a new line character. Otherwise it returns only currentTokenImage and a space.

4.25 TokenSingleLineComment Class

2.2.1184.25.1 Structure

SuperClass: TokenReformat
Type: Public
SubClass: None
Implementation: None
Constructor: None

2.2.1194.25.2 Function

public String reformat(String currentTokenImage, int nextTokenKind, int previousTokenKind, Indent indentLevel)

2.2.1204.25.3 Interfaces

None

2.2.1214.25.4 Application Data

None

2.2.1224.25.5 Detailed Design Description

4.25.5.1 reformat Method

Type: Public
Input Parameters: String currentTokenImage, int nextTokenKind, int previousTokenKind and Indent indentLevel
Output Parameters: String
Description: verifies that nextTokenKind is a left brace or not. If it is not a left brace, it returns currentTokenImage, a new line character and the current indentation. Otherwise it returns only currentTokenImage and a new line character.

4.26 WindowMenu Class

2.2.1234.26.1 Structure

SuperClass: JMenu
Type Public
SubClass: None
Implementation: None
Constructor: WindowMenu

2.2.1244.26.2 Function

None

2.2.1254.26.3 Interfaces

import javax.swing.JMenu

2.2.1264.26.4 Application Data

None

2.2.1274.26.5 Detailed Design Description

4.26.5.1 WindowMenu Constructor

Type : Public
Input Parameters: String windowMenuCaption, char windowMenuMnemonic
Output Parameters: None
Description: sets the menu name by calling the method super with windowMenuCaption as parameter. Sets the mnemonic by calling the setmnemonic method with windowMenuMnemonic as parameter

4.27 WindowMenuBarView Class

2.2.1284.27.1 Structure

SuperClass: JMenuBar
Type Public
SubClass: None
Implementation: None
Constructor: WindowMenuBarView

2.2.1294.27.2 Function

public JMenuItem getExitMenuItem()
public JMenuItem getCompileMenuItem()
public JMenuItem getOpenFileMenuItem()
public JMenuItem getCloseFileMenuItem()
public JMenuItem getSaveFileMenuItem()
public JMenuItem getSelectLanguageParserMenuItem()
public JMenuItem getParseDataMenuItem()
public JMenuItem getEncapsulateFieldsMenuItem()
public JMenuItem getReplaceMagicNumberWithSymbolicConstantMenuItem()

2.2.1304.27.3 Interfaces

import javax.swing.*

2.2.1314.27.4 Application Data

private WindowMenuItem openMenuItem
private WindowMenuItem closeMenuItem
private WindowMenuItem saveMenuItem
private WindowMenuItem exitMenuItem
private WindowMenuItem selectLanguageParserMenuItem
private WindowMenuItem parseDataMenuItem
private WindowMenuItem encapsulateFieldsMenuItem
private WindowMenuItem replaceMagicNumberWithSymbolicConstantMenuItem
private WindowMenuItem compileMenuItem

2.2.1324.27.5 Detailed Design Description

4.27.5.1 WindowMenuBarView Constructor

Type: Public:
Input Parameters:
Output Parameters: None
Description: creates the various menus by creating new WindowMenu objects and their related menu items by ccreating new WindowmenuItem objects.

4.27.5.2 getExitMenuItem Method

Type: Public:

Input Parameters: None
Output Parameters: JMenuItem
Description: returns exitMenuItem

4.27.5.3 getCompileMenuItem Method

Type: Public:
Input Parameters: None
Output Parameters: JMenuItem
Description: returns compileMenuItem

4.27.5.4 getOpenFileMenuItem Method

Type: Public:
Input Parameters: None
Output Parameters: JMenuItem
Description: returns openFileMenuItem

4.27.5.5 getCloseFileMenuItem Method

Type: Public:
Input Parameters: None
Output Parameters: JMenuItem
Description: returns closeFileMenuItem

4.27.5.6 getSaveFileMenuItem Method

Type: Public:
Input Parameters: None
Output Parameters: JMenuItem
Description: returns saveFileMenuItem

4.27.5.7 getSelectLanguageParserMenuItem Method

Type: Public:
Input Parameters: None
Output Parameters: JMenuItem
Description: returns selectLanguageParserMenuItem

4.27.5.8 getParseDataMenuItem Method

Type: Public:
Input Parameters: None
Output Parameters: JMenuItem
Description: returns parseDataMenuItem

4.27.5.9 getEncapsulateFieldsMenuItem Method

Type: Public:
Input Parameters: None
Output Parameters: JMenuItem
Description: returns encapsulateFieldsMenuItem

4.27.5.10 getReplaceMagicNumberWithSymbolicConstantMenuItem Method

Type: Public:
Input Parameters: None
Output Parameters: JMenuItem
Description: returns replaceMagicNumberWithSymbolicConstantMenuItem

4.28 WindowMenuItem Class

2.2.1334.28.1 Structure

SuperClass: JMenuItem
Type: Public
SubClass: None
Implementation: None
Constructor: WindowMenuItem

2.2.1344.28.2 Function

None

2.2.1354.28.3 Interfaces

import javax.swing.JMenuItem

2.2.1364.28.4 Application Data

None

2.2.1374.28.5 Detailed Design Description

4.28.5.1 WindowMenuItem Constructor

Type: Public
Input Parameters: String menuItemName, char menuMnemonic
Output Parameters: None
Description: sets the menu item name by calling the method super with menuItemName as parameter. Sets the mnemonic by calling the setmnemonic method with menuMnemonic as parameter.

5.0 Quality Assurance

5.1 Software Test Plan

The software test plan used to verify and validate ART are contained as Appendix E.

5.2 Software Test Procedures

The software test proceduresused to verify and validate ART are contained as Appendix F.

5.3 Test Cases

The various test cases used to verify ART are contained in Appendix G.

5.4 Test Cross reference Matrix

Test Case	SRS Ref #
TCS-001	2.3.1, 2.5.8, 4.1.1, 4.1.2, 4.1.3, 4.1.4, 4.1.7, 4.1.8, 4.1.9, 4.1.10, 4.1.11, 4.1.12, 4.1.13, 4.1.14, 4.1.15, 4.1.16, 4.1.17
TCS-002	2.3.2, 2.4.1, 2.4.2, 2.5.1, 2.5.2, 2.5.3, 2.5.4, 2.5.5, 2.5.9, 2.5.10, 2.5.12, 4.1.1, 4.1.2, 4.1.5,
TCS-003	2.3.3, 2.4.3, 2.4.4, 2.4.5, 2.4.6, 2.5.3, 2.5.4, 2.5.5, 2.5.6, 2.5.7, 2.5.9, 2.5.10, 2.5.12, 2.7.4, 4.1.1, 4.1.2, 4.1.5, 4.1.6,
TCS-004	2.3.1, 2.3.2, 2.3.3, 2.4.3, 2.4.4, 2.4.5, 2.4.6, 2.5.1, 2.5.2, 2.5.3, 2.5.4, 2.5.5, 2.5.6, 2.5.7, 2.5.8, 2.5.9, 2.5.10, 2.5.11, 2.5.12, 2.7.4, 4.1.1, 4.1.2, 4.1.3, 4.1.4, 4.1.5, 4.1.6, 4.1.7, 4.1.8, 4.1.9, 4.1.10, 4.1.11, 4.1.12, 4.1.13,

6.0 Traceability matrix linking shall list to methods, classes, use cases

Item Name	Type	SRS Ref #
AbstractFactory	Class	2.4.6, 2.5.7
AbstractToken	Class	2.4.6, 2.5.7
add	Method	2.5.4
add	Method	2.5.4
CIS5915JavaParser	Package	2.3.2, 2.4.1, 2.4.2, 2.5.1, 2.5.3
clearDisplay	Method	2.5.10
CloseFile	Use Case	2.3.1, 2.5.10
CompileCode	Use Case	2.4.3, 2.7.4
createCollectionIterator	Method	2.5.4
createGetMethodTokenVector	Method	2.5.3, 2.5.6
createMethodName	Method	2.5.6
createSetMethodTokenVector	Method	2.5.3, 2.5.6
createToken	Method	2.4.6, 2.5.7
decreaseIndentLevel	Method	2.5.5
deparseToken	Method	2.5.5
DisplayFile	Use Case	2.3.1, 2.5.8, 4.1.5, 4.1.6
displayText	Method	4.1.5, 4.1.6
EncapsulateFields	Class	2.4.4, 2.5.6
EncapsulateFields	Use Case	2.3.3, 2.4.4, 2.5.6
execute	Method	3.5.6
findFirstVariableOccurrence	Method	2.5.6
get	Method	2.5.4

getCloseMenuItem	Method	2.3.1, 2.5.8, 4.1.8
getCompileMenuItem	Method	2.3.1, 2.5.8, 4.1.11
getCurrentCollectionIndex	Method	2.5.4
getCurrentVectorLocation	Method	2.5.3, 2.5.4
getEncapsulateFieldsMenuItem	Method	2.3.1, 2.5.8, 4.1.13
getExitMenuItem	Method	2.3.1, 2.5.8, 4.1.8
getFileName	Method	2.5.9, 2.5.11, 4.1.1
getFirstLeftBrace	Method	2.5.6
getIndentLevel	Method	2.5.5
getIndentStringValue	Method	2.5.5
getInsertCodeCollectionIterator	Method	2.5.4
getInsertTokenVector	Method	2.5.3, 2.5.4
getInsertVectorLocation	Method	2.5.3, 2.5.4
getImage	Method	2.4.6
getKind	Method	2.4.6
getOpenMenuItem	Method	2.3.1, 2.5.8, 4.1.8
getMethodName	Method	2.5.6
getParseDataMenuItem	Method	2.3.1, 2.5.8, 4.1.10
getSaveMenuItem	Method	2.3.1, 2.5.8, 4.1.8
getSelectLanguageParserMenuItem	Method	2.3.1, 2.5.8, 4.1.2, 4.1.10
getRefactoredVariableName	Method	2.5.6
getReplaceMagicNumberWithSymbolicConstantMenuItem	Method	2.3.1, 2.5.8, 4.1.13
getToken	Method	2.4.6
getVariableImageRequired	Method	2.5.6
getVariableIndexValue	Method	2.5.6
getVariableRequiredKind	Method	2.5.6
increaseIndentLevel	Method	2.5.5
increaseIndentStringValue	Method	2.5.5
Indent	Class	2.5.5
insertTokens	Method	2.5.4, 2.5.6
isArithmeticOperator	Method	2.4.6
isAssignmentOperator	Method	2.4.6
isLeftParen	Method	2.4.6
isNotAbstractClassVoid	Method	2.4.6
isPublic	Method	2.4.6
isThereMoreObjects	Method	2.5.4
isSemiColon	Method	2.4.6
JavaDeParser	Class	2.5.5
loopNotDone	Method	2.5.6
main	Method	2.3.1, 2.5.8
nextObject	Method	2.5.4
ObjectCollection	Class	2.5.4
ObjectIterator	Class	2.5.4
OpenFile	Use Case	2.3.1, 2.5.8

ParseCode	Use Case	2.3.2, 2.4.2, 2.5.2
ProjectFile	Class	2.3.1, 2.5.9
readFile	Method	2.5.9
RefactorCode	Use Case	2.3.3, 2.5.6
RefactoringContext	Class	2.5.6
RefactoringStrategy	Class	2.5.6
RefactoringTool	Class	2.3.1
reformat	Method	2.5.5
removeElement	Method	2.5.4
ReplaceMagicNumber	Class	2.4.5, 2.5.6
ReplaceMagicNumber	Use Case	2.3.3, 2.4.5, 2.5.6
saveFile	Method	2.5.11
SaveFile	Use Case	2.3.1, 2.5.8, 2.5.11
SelectParser	Use Case	2.3.2, 4.1.2
setCloseFileMenuItem	Method	2.3.1, 2.5.8, 4.1.8
setCollection	Method	2.5.4
setCompileMenuItem	Method	2.3.1, 2.5.8, 4.1.11
setCurrentVectorLocation	Method	2.5.3
setEncapsulateFieldsMenuItem	Method	2.3.1, 2.5.8, 4.1.13
setExitMenuItem	Method	2.3.1, 2.5.8, 2.5.12, 4.1.8
setFileName	Method	2.5.9, 2.5.11, 4.1.1
setFirstLeftBrace	Method	2.5.6
setIndentLevelValue	Method	2.5.5
setIndentStringValue	Method	2.5.5
setInsertLocation	Method	2.5.3, 2.5.4
setInsertVectorLocation	Method	2.5.3, 2.5.4
setImage	Method	2.4.6
setKind	Method	2.4.6
setLoopCounter	Method	2.5.6
setMagicNumberTokenVector	Method	2.5.3
setOpenFileMenuItem	Method	2.3.1, 2.5.8, 4.1.8
setParseDataMenuItem	Method	2.3.1, 2.5.8, 4.1.10
setRefactoredVariableName	Method	2.5.6
setRefactoringStrategy	Method	2.5.6
setReformatStrategy	Method	2.5.5
setReplaceMagicNumberWithSymbolicConstantMenuItem	Method	2.3.1, 2.5.8, 4.1.13
setSaveFileMenuItem	Method	2.3.1, 2.5.8, 4.1.8
setSelectLanguageParserMenuItem	Method	2.3.1, 2.5.8, 4.1.2, 4.1.10
setStringValue	Method	2.5.5
setToFirst	Method	2.5.4
setToken	Method	2.4.6
setToLast	Method	2.5.4
setVariableImageRequired	Method	2.5.6

setVariableIndexValue	Method	2.5.6
setVariableRequiredKind	Method	2.5.6
setVectorLocation	Method	2.5.3, 2.5.4
SourceCodeView	Class	2.3.1, 2.5.8, 4.1.4
TokenCollection	Class	2.5.4
TokenContext	Class	2.5.5
TokenDefault	Class	2.5.5
TokenFactory	Class	2.5.7
TokenIterator	Class	2.5.4
TokenLBrace	Class	2.5.5
TokenMultiLineComment	Class	2.5.5
TokenParenBracketDotBang	Class	2.5.5
TokenRBrace	Class	2.5.5
TokenReformat	Class	2.5.5
TokenSingleLineComment	Class	2.5.5
TokenSemiColon	Class	2.5.5
WindowMenu	Class	2.3.1, 2.5.8, 4.1.7, 4.1.9, 4.1.11, 4.1.12, 4.1.13, 4.1.14, 4.1.15
WindowMenuBarView	Class	2.3.1, 2.5.8, 4.1.3, 4.1.16, 4.1.17
WindowMenuItem	Class	2.3.1, 2.5.8, 4.1.8, 4.1.10, 4.1.13, 4.1.15

AUTOMATED
REFACTORING
TOOL (ART)

APPENDIX C

SOFTWARE TEST PLAN

APPENDIX C

TABLE OF CONTENTS

1.0 INTRODUCTION 1

1.1 Purpose 1
1.2 References 1
1.3 ART Components 1

2.0 TESTING 2

2.1 Test Items 2
2.2 Test Objectives 2
2.3 General Approach 2
2.4 Pass/Fail Criteria 2

3.0 TESTING REQUIREMENTS 3

3.1 Environmental Requirements 3
3.2. Staffing and Training Requirements 3
3.3. Schedule 3

1.0 Introduction

1.1 Purpose

The purpose of this document is to describe the strategy and all testing activities for the verification and validation of the Automatic Refactoring Tool (ART). It focuses on the validation testing performed to ensure compliance of ART to the requirements defined in the Software Requirement Specifications (SRS)

1.2 References

NUMBER	TITLE	DATE
SRS	Software Requirement Specifications (SRS) - Appendix A to ART Project Report	2003
SDS	Software Design Specifications (SDS) - Appendix B to ART Project Report	2003
IEEE Std 1012	IEEE Standard for Software Verification and Validation	1998

1.3 ART Components

ART consists of the following four (4) major components:

e) a framework using design patterns for the incremental development of refactoring transformations
f) a parser,
g) a refactoring transformer, and,
h) a graphical user interface (GUI).

The framework is responsible of providing the basic classes and methods that are required to permit applying automatically refactoring transformations. The parser is responsible for reading Java modules and applying parsing rules to generate tokens representing the code read. The refactoring transformer is responsible for modifying the set of tokens provided by the parser. To prove the framework developed, the following refactoring transformations as defined in [Fowler00] are implemented:

c) Encapsulate Fields, and
d) Replace Magic Number with Symbolic Constant.

The GUI is responsible for displaying to the user the initial code, the newly transformed code, and the refactoring transformation(s) applied.

2.0 TESTING

2.1 Test Items

As mentioned in paragraph 1.3, there exist four (4) components requiring verification and validation: the refactoring framework, the parser, the refactoring transformations implemented and the GUI. The following table defines the various test cases developed for properly validating ART against the SRS content.

Test Case	Description
Test Case 1	GUI validation
Test Case 2	Parser validation
Test Case 3	Refactoring transformations validation
Test Case 4	Refactoring framework validation

2.2 Test Objectives

Software verification and validation, applicable to ART, consists of four (4) primary objectives. The first objective involves finding inconsistencies between the various documents generated for ART. The second objective involves finding errors inadvertently introduced during code development. The third objective centers on verifying that the design and code developed for ART are based solely on the requirements identified in the SRS. The final objective is to validate the executable code ensuring it complies with the original requirements.

A secondary objective of software verification and validation is to provide assurance that if, after initial testing, errors were discovered, the software corrections did not have an adverse effect of the software. This objective is accomplished through performing the appropriate test cases on the corrected code.

2.3 General Approach

Throughout the life cycle development of ART, all documents produced go through a verification and validation. These activities are accomplished using, as a baseline, the previous version of the document for verification and any generated document referenced for validation. As the code is designed incrementally, after the completion of a component, the appropriate test case(s) will be conducted to ensure all requirements are met. In addition, all code generated will be reviewed to ensure adherence to the coding standards (Annex B) and being error free. When all components of ART are completed, each of the test cases is executed again to ensure proper integration of the various components.

2.4 Pass/Fail Criteria

Successful completion of the software verification and validation is based on satisfying that the document are consistent between themselves and that all test cases were performed successfully. At any time, if errors are discovered, they will be corrected and validated and verified prior to moving forward.

3.0 TESTING REQUIREMENTS

3.1 Environmental Requirements

To perform the test cases, the latest version of ART is always used with error free source code files necessary to perform the refactoring transformations implemented.

3.2. Staffing and Training Requirements

Not applicable.

3.3. Schedule

The validation and verification activities for each document generated for ART, (SRS and SDS), will be performed at the completion of each document. Only upon successful verification and validation that work can proceed. Therefore, the SDS cannot be started until the SRS is verified and validated successfully and code development cannot be attempted until the SDS is successfully validated and verified. During the code development, appropriate test cases will be conducted after each of the code development increment. When all code development increments are validated and verified successfully, all test cases are executed again.

AUTOMATED
REFACTORING
TOOL (ART)

APPENDIX D

SOFTWARE TEST PROCEDURES

APPENDIX D

TABLE OF CONTENTS

1.0 INTRODUCTION 1

1.1 Purpose 1
1.2 References 1

2.0 TESTING 1

2.1 Testing Scope 1
2.2 Testing Refinement 1

3.0 TEST CASES 1

3.1 Test Cases Descriptions 1

4.0 PASS/FAIL CRITERIA 2

4.1 Test cases pass/fail criteria 2

1.0 Introduction

1.1 Purpose

The purpose of this document is to describe the various test procedures required for the verification and validation of the Automated Refactoring Tool (ART). This document defines the scope of the testing as well as identifies the test cases used for performing the verification and validation. However, validation and verification of the documents produced for ART are not covered in this document.

1.2 References

NUMBER	TITLE	DATE
SRS	Software Requirement Specifications (SRS) - Appendix A to ART Project Report	2003
SDS	Software Design Specifications (SDS) - Appendix B to ART Project Report	2003
STP	Software Test Plan (STP) - Appendix C to ART Project Report	2003

2.0 Testing

2.1 Testing Scope

The testing scope defined in this document is ART. As ART consists of three (3) distinct components, a graphical user interface (GUI), a parser and a refactor, each of the components requires validation either individually or in conjunction with one another.

2.2 Testing Refinement

As ART was developed incrementally, testing ART is also performed incrementally. The test cases designed and discussed below demonstrate this approach. To properly validate and verify ART, the test cases must cover the following:

a) Verify the functionality of the GUI,
b) Verify the functionality of the parser,
c) Verify the functionality of the refactor, and
d) Verify the functionality of the application as a whole.

3.0 Test Cases

3.1 Test Cases Descriptions

TCS-001: This test case verifies the functionality of each menu item in the GUI.

TCS-002: This test case verifies that the parser component of the application performs properly.

TCS-003: This test case verifies that the refactor component of the application performs properly.

TCS-004: This test case verifies that all components of the application perform properly when used in conjunction with one another.

4.0 Pass/fail criteria.

4.1 Test cases pass/fail criteria

Each test case design is expected to perform certain function(s). The purpose of each test case is to validate that the result(s) of the function(s) are as expected. When the feature performs as expected it would pass. When the feature does not perform as expected it would fail. There may be special cases where acceptable tolerances are applied. These tolerances are identified in the expected output within the test case.

AUTOMATED REFACTORING TOOL (ART)

APPENDIX E

SOFTWARE TEST CASES

APPENDIX E

TABLE OF CONTENTS

1.0 INTRODUCTION 1

1.1 Purpose 1
1.2 References 1

2.0 TEST CASES 1

2.1 TCS-001 1
2.1.1 Purpose 1
2.1.3 Requirements verified 1
2.1.3 Steps 1
2.2 TCS-002 3
2.2.1 Purpose 3
2.2.2 Requirements verified 3
2.2.3 Steps 3
2.3 TCS-003 4
2.3.1 Purpose 4
2.3.2 Requirements verified 4
2.3.2 Steps 4
2.4 TCS-004 7
2.4.1 Purpose 7
2.4.2 Requirements verified 7
2.4.3 Steps 7

1.0 Introduction

1.1 Purpose

This document describes the various test cases developed for the validation and verification of the Automated Refactoring Tool (ART). Each of the text case is designed to verify specific functionality of ART.

1.2 References

NUMBER	TITLE	DATE
SRS	Software Requirement Specifications (SRS) - Appendix A to ART Project Report	2003
SDS	Software Design Specifications (SDS) - Appendix B to ART Project Report	2003
STP	Software Test Plan (STP) - Appendix C to ART Project Report	2003
STPr	Software Test Procedures (STPr) - Annex D to ART Project Report	2003

2.0 Test Cases

2.1 TCS-001

2.2.1382.1.1 Purpose

The purpose of test case TCS-001 is to verify and validate the functionality of the menu items in contained in the graphical user interface (GUI) of ART. This test is performed following the completion of the GUI. Because of the iterative development, the complete functionality of the menu options is tested during the performance of TCS-002, TCS-003 and TCS-004. When performing test case TCS-001, messages are displayed advising the menu option selected.

2.1.3 Requirements verified

The requirements verified and validated by this test are:

2.1.3 Steps

Step	Description/Keystroke	Expected Result
1	Start the application	The ART main window is displayed
2	Click on the File menu option	The File sub-menu items: **Open, Close, Save** and **Exit** , are displayed
3	Click on the Open menu option	The message **Open File** is displayed in a message window

4	Click on the Close menu option	The message **Close File** is displayed in a message window
5	Click on the Save menu option	The message **Save File** is displayed in a message window
6	Click on the Exit menu option	The message **Exit** is displayed in a message window
7	Click on the Parser menu option	The Parser sub-menu items: **Select Language Parser** and **Parse Data** sub-menu items , are displayed
8	Click on Select Language Parser menu option	The message **Select Language Parser** is displayed in a message window
9	Click on Parse Data menu option	The message **Parsing Data** is displayed in a message window
10	Click on the Refactor menu option	The Refactor sub-menu items: **Encapsulate Fields** and **Replace Magic Number With Symbolic Constant** sub-menu items , are displayed
11	Click on Encapsulate Fields menu option	The message **Performing Encapsulate Fields Refactoring** is displayed in a message window
12	Click on Replace Magic Number with Symbolic Constant menu option	The message **Performing Replace Magic Number with Symbolic Constant Refactoring** is displayed in a message window
13	Click on the Compile menu option	The Compile sub-menu items: **Compile Code** sub-menu item is displayed
14	Click on Compile Code menu option	The message **Compiling Code** is displayed in a message window
15	Terminate the application by clicking the X at the top right of the window	The application is terminated
16	Press the Alt-F key combinations	The File sub-menu items: **Open, Close, Save** and **Exit** , are displayed
17	Press the O key	The message **Open File** is displayed in a message window
18	Press the C key	The message **Close File** is displayed in a message window
19	Press the S key	The message **Save File** is displayed in a message window
20	Press the E key	The message **Exit** is displayed in a message window
21	Press the Alt-P key combinations	The Parser sub-menu items: **Select Language Parser** and **Parse Data** sub-menu items , are displayed

22	Press the l key	The message **Select Language Parser** is displayed in a message window
23	Press the P key	The message **Parsing Data** is displayed in a message window
24	Press the Alt-R key combinations	The Refactor sub-menu items: **Encapsulate Fields** and **Replace Magic Number With Symbolic Constant** sub-menu items , are displayed
25	Press the F key	The message **Performing Encapsulate Fields Refactoring** is displayed in a message window
26	Press the M key	The message **Performing Replace Magic Number with Symbolic Constant Refactoring** is displayed in a message window
27	Press the Alt-C key combinations	The Compile sub-menu items: **Compile Code** sub-menu item is displayed
28	Press the C key	The message **Compiling Code** is displayed in a message window

2.2 TCS-002

2.2.1392.2.1 Purpose

The purpose of test case TCS-002 is to verify and validate the functionality of the parser component of ART. This test is performed following the successful verification and validation of the GUI and the completion of the parser package modifications. This test case also verifies and validates the functionality of the Open, Close and Exit menu options.

2.2.1402.2.2 Requirements verified

The requirements verified and validated by this test are:

2.2.1412.2.3 Steps

Step	Description/Keystroke	Expected Result
1	Start the application	The ART main window is displayed
2	Click on the File menu option	The File sub-menu items **Open, Close, Save** and **Exit** are displayed
3	Click on the Open menu option	The Open File window is displayed
4	Change the directory to where the java source code used for testing is located (C:\Masters\CIS5915\Code\Testing)	The content of the directory is displayed
5	Select any .java file by double clicking on it	The content of the file is displayed

		on the left-hand side of the display area of the GUI.
6	Click on the Parser menu option	The Parser sub-menu items **Select Language Parser** and **Parse Data** sub-menu items are displayed
7	Click on Parse Data menu option	A message window will appear stating the execution time of the parsing
8	Click the Ok button	The message window disappears and the file content is displayed on the right-hand side of the display area of the GUI and is identical to the left-hand side
9	Peruse the JBuilder message window and verify the tokens displayed matches the file content	The file content is identical to the tokens displayed
10	Click on the File menu option	The File sub-menu items **Open, Close, Save** and **Exit** are displayed
11	Click on the Close menu option	The left and right hand side display areas are cleared
12	Click on the File menu option	The File sub-menu items **Open, Close, Save** and **Exit** are displayed
13	Click on the Exit menu option	The ART main window is closed

2.3 TCS-003

2.3.1 Purpose

The purpose of test case TCS-003 is to verify and validate the functionality of the refactor component of ART. This test is performed following the successful verification and validation of the parser component and the completion of the refactor component. This test also verifies and validates the Open, Exit, Parse Data and Compile Code menu options

2.2.1422.3.2 Requirements verified

The requirements verified and validated by this test are:

2.3.2 Steps

Step	Description/Keystroke	Expected Result
1	Start the application	The ART main window is displayed
2	Click on the File menu option	The File sub-menu items: **Open, Close, Save** and **Exit** , are displayed
3	Click on the Open menu option	The Open File window is displayed
4	Change the directory to where the java source code used for testing is located (C:\Masters\CIS5915\Code\Testing)	The content of the directory is displayed
5	Double click the file Table.java	The content of the file is displayed

		on the left-hand side of the display area of the GUI.
6	Click on the Parser menu option	The Parser sub-menu items: **Select Language Parser** and **Parse Data** sub-menu items , are displayed
7	Click on Parse Data menu option	A message window will appear stating the execution time of the parsing
8	Click the Ok button	The message window disappears and the file content is displayed on the right-hand side of the display area of the GUI and is identical to the left-hand side
9	Click on the Refactor menu option	The Refactor sub-menu items: **Encapsulate Fields** and **Replace Magic Number With Symbolic Constant** sub-menu items , are displayed
10	Click on Replace Magic Number with Symbolic Constant menu option	The message **Performing Replace Magic Number with Symbolic Constant Refactoring** is displayed in a message window
11	Click the Ok button	The message window disappears and the refactored code is displayed
12	Verify that the code was refactored properly by ensuring that new variables REFACTORED0, REFACTORED1, REFACTORED2, REFACTORED3, REFACTORED4 are declared as private final int and equal to 1, 1, 10, 100 and 1000 respectively and that where the values of 1, 1, 10, 100 and 1000 were used initially, they have been replaced with the new variables	The variables REFACTORED0, REFACTORED1, REFACTORED2, REFACTORED3, REFACTORED4 are declared as private final int and equal to 1, 1, 10, 100 and 1000 respectively and the values of 1, 1, 10, 100, and 1000 used are replaced with the new variables
13	Click on the Compile menu option	The Compile sub-menu items: **Compile Code** sub-menu item is displayed
14	Click on Compile Code menu option	The message **Compiling** ***filename*** is displayed in a message window
15	Click on the Ok button	The message window disappears and another window will appear stating the successful compilation of the file.
16	Click the Ok button	The message window disappears
17	Click on the File menu option	The File sub-menu items: **Open, Close, Save** and **Exit** , are displayed
18	Click on the Exit menu option	The ART Main window is closed

19	Repeat steps 1 through 4	
20	Select the file indent.java	The content of the file is displayed on the left-hand side of the display area of the GUI.
21	Click on the Parser menu option	The Parser sub-menu items: **Select Language Parser** and **Parse Data** sub-menu items , are displayed
22	Click on Parse Data menu option	A message window will appear stating the execution time of the parsing
23	Click the Ok button	The message window disappears and the file content is displayed on the right-hand side of the display area of the GUI and is identical to the left-hand side
24	Click on Replace Magic Number with Symbolic Constant menu option	The message **Performing Replace Magic Number with Symbolic Constant Refactoring** is displayed in a message window
25	Click on Encapsulate Fields menu option	The message **Performing Encapsulate Fields Refactoring** is displayed in a message window
26	Click the Ok button	The message window disappears and the refactored code is displayed
27	Verify that the variable indentLevelValue is now declared as private int, that two new methods, setRefactoredIndentLevelValue and getRefactoredIndentLevelValue, were generated. Also verify that any assignment to the variable uses the set method and that any usage of the variable uses the get method	The variable indentLevelValue is declared as private int, the methods setRefactoredIndentLevelValue and getRefactoredIndentLevelValue were created properly, any usage of the variable makes use of the get method and any assignment to the variable uses the set method.
28	Click on the Compile menu option	The Compile sub-menu items: **Compile Code** sub-menu item is displayed
29	Click on Compile Code menu option	The message **Compiling *filename*** is displayed in a message window
30	Click on the Ok button	The message window disappears and another window will appear stating the successful compilation of the file.
31	Click the Ok button	The message window disappears
32	Click on the File menu option	The File sub-menu items: **Open, Close, Save** and **Exit** , are displayed
33	Click on the Exit menu option	The ART Main window is closed

2.4 TCS-004

2.4.1 Purpose

The purpose of test case TCS-004 is to verify and validate the complete functionality of ART. This test is performed following the successful verification and validation of test cases TCS-001, TCS-002 and TCS-003.

2.2.1432.4.2 Requirements verified

The requirements verified and validated by this test are:

2.2.1442.4.3 Steps

Step	Description/Keystroke	Expected Result
1	Perform steps 1 through 17 of TCS-003	
2	Click on the Save menu option	The Save File window is displayed
3	Change the directory to where the java source code used for testing is located (C:\Masters\CIS5915\Code\Testing)	The content of the directory is displayed
4	Double click the file table.java	The Save File window disappears.
5	Click on the File menu option	The File sub-menu items: **Open, Close, Save** and **Exit** , are displayed
6	Click on the close menu option	The display areas are cleared
7	Click on the File menu option	The File sub-menu items: **Open, Close, Save** and **Exit** , are displayed
8	Click on the Open menu option	The Open file window is displayed
9	Change the directory to where the java source code used for testing is located (C:\Masters\CIS5915\Code\Testing)	The content of the directory is displayed
10	Double click the file table.java	The file is displayed in the left-hand side of the display area.
11	Verify that the file was saved properly with the new refactored code as per step 27 of TCS-003	The file was saved properly
12	Click on the File menu option	The File sub-menu items: **Open, Close, Save** and **Exit** , are displayed
13	Click on the Exit menu option	The ART Main window is closed

AUTOMATED REFACTORING TOOL (ART)

APPENDIX F

CODE LISTING

APPENDIX F

TABLE OF CONTENTS

1.0 AbstractToken.Java 1
2.0 EncapsulateFields.java 6
3.0 Indent.java 10
4.0 JavaDeParser.java 12
5.0 ObjectCollection.java 14
5.0 ObjectIterator.java 15
7.0 ProjectFile.java 16
8.0 RefactoringContext 18
9.0 RefactoringStrategy.java 19
10.0 RefactoringTool.java 24
11.0 ReplaceMagicNumber.java 29
12.0 SourceCodeView.java 31
13.0 TokenCollection.java 32
14.0 TokenContext.java 33
15.0 TokenCreator.java 34
16.0 TokenDefault.java 35
17.0 TokenFactory.java 36
18.0 TokenIterator.java 37
19.0 TokenLBrace.java 39
20.0 TokenMultiLineComment.java 40
21.0 TokenParenBracketDotBang.java 41
22.0 TokenRBrace.java 42
23.0 TokenReformat.java 43
24.0 TokenSemiColon.java 44
25.0 TokenSingleLineComment.java 45
26.0 WindowMenu.java 46
27.0 WindowMenuBarView.java 47
28.0 WindowMenuItem.java 49

1.0 AbstractToken.Java

```
/*
 Title:       Automated Refactoring Tool
 Name:        AbstractToken Class
 Description: The Abstract Token class is an abstract class to implement
              the abstract factory pattern
 Author:      Alain Coutu
 Date:        09/01/02
 Modified:    09/09/02  Builder pattern implemented
              09/10/02  Move method refactoring
              09/15/02  Performed Encapsulate Fields refactoring
              09/16/02  Added methods for the implementation of the encapsulate
                        field refactoring capability
*/

package CIS5915JavaParser;

// Define the AbstractToken class
public class AbstractToken implements JavaParserConstants{

  private Token vectorToken; // Create a Token object;
  private Character tokenCharacter; // Define a Character object
  private final int ZERO = 0; // Define the constants 0

  // Method setToken sets the token object
  public void setToken(Token tokenToSet) {
    vectorToken = tokenToSet;
  } // end setToken

  // Method getToken returns the token object
  public Token getToken() {
    return vectorToken;
  } // end getToken

  // Method getImage returns the token image
  public String getImage() {
    return vectorToken.image;
  } // end getImage

  // Method getKind returns the token kind
  public int getKind() {
    return vectorToken.kind;
  } // end getKind

  // Method setImage sets the token image
  public void setImage(String tokenImage) {
    vectorToken.image = tokenImage;
  } // end setImage

  // Method setKind sets the token kind
  public void setKind(int tokenKind) {
    vectorToken.kind = tokenKind;
  } // end setKind

  // Method isPublic returns true if the token image is public
```

```
public boolean isPublic() {
  return (getKind() == PUBLIC);
} // end isPublic

// Method isClass returns true if the token image is class
public boolean isClass() {
  return (getKind() == CLASS);
} // end isClass

// Method isVoid returns true if the token image is void
public boolean isVoid() {
  return (getKind() == VOID);
} // end isVoid

// Method isAbstract returns true if the token image is abstract
public boolean isAbstract() {
  return (getKind() == ABSTRACT);
} // end isAbstract

// Method isLeftParen returns true if the token is image left paren
public boolean isLeftParen() {
  return (getKind() == LPAREN);
}// end isLeftParen

// Method isAssign returns true if the token is an equal sign
public boolean isAssign() {
  return (getKind() == ASSIGN);
} // end isAssign

// Method isGreaterThan returns true if the token is a greater than operator
public boolean isGreaterThan() {
  return (getKind() == GT);
} // end isGreaterThan

// Method isLessThan returns true if the token is a less than operator
public boolean isLessThan() {
  return (getKind() == LT);
} // end isLessThan

// Method isEqual returns true if the token is an equal operator
public boolean isEqual() {
  return (getKind() == EQ);
} // end isEqual

// Method isNotEqual returns true if the token is a not equal operator
public boolean isNotEqual() {
  return (getKind() == NE);
} // end isNotEqual

// Method isGreaterEqual returns true if the token is a greater/equal operator
public boolean isGreaterEqual() {
  return (getKind() == GE);
} // end isGreaterEqual

// Method isLessEqual returns true if the token is a less/equal operator
public boolean isLessEqual() {
  return (getKind() == LE);
```

```
} // end isLessEqual

// Method isIncrement returns true if the token is an increment operator
public boolean isIncrement() {
  return (getKind() == INCR);
} // end isIncrement

// Method isDecrement returns true if the token is a decrement operator
public boolean isDecrement() {
  return (getKind() == DECR);
} // end isDecrement

// Method isPlus returns true if the token is a plus operator
public boolean isPlus() {
  return (getKind() == PLUS);
} // end isPlus

// Method isMinus returns true if the token is a minus operator
public boolean isMinus() {
  return (getKind() == MINUS);
} // end isMinus

// Method isStar returns true if the token is a star operator
public boolean isStar() {
  return (getKind() == STAR);
} // end isStar

// Method isSlash returns true if the token is a slash operator
public boolean isSlash() {
  return (getKind() == SLASH);
} // end isSlash

// Method isPlusAssign returns true if the token is a plus/assign operator
public boolean isPlusAssign() {
  return (getKind() == PLUSASSIGN);
} // end isPlusAssign

// Method isMinusAssign returns true if the token is a minus/assign operator
public boolean isMinusAssign() {
  return (getKind() == MINUSASSIGN);
} // end isMinusAssign

// Method isStarAssign returns true if the token is a star/assign operator
public boolean isStarAssign() {
  return (getKind() == STARASSIGN);
} // end isStarAssign

// Method isSlashAssign returns true if the token is a slash/assign operator
public boolean isSlashAssign() {
  return (getKind() == SLASHASSIGN);
} // end isSlashAssign

// Method isLeftBrace returns true if the token is a left brace
public boolean isLeftBrace() {
  return (getKind() == LBRACE);
} // end isLeftBrace
```

```
// Method isSemiColon returns true if the token is a semi colon
public boolean isSemiColon() {
  return (getKind() == SEMICOLON);
} // end isSemiColon

// Method isANumber returns true if the token is a number
public boolean isANumber() {
  setTokenCharacter(getImage().charAt(ZERO));
  return (tokenCharacter.isDigit(tokenCharacter.charValue()));
} // end isANumber

/*
  Method isAVariableDeclaration returns if the set of token is part of a
  variable declaration
*/
public boolean isAVariableDeclaration(AbstractToken nextToken) {
  return (isPublic() && !(nextToken.isAbstract() || nextToken.isClass() ||
          nextToken.isVoid()));
} // end isAVariableDeclaration

// Method isArithmeticOperator returns true if the token is an operator
public boolean isArithmeticOperator () {
  return ((isAssign()) || (isGreaterThan()) || (isLessThan()) || (isEqual())||
          (isNotEqual()) || (isLessEqual()) || (isGreaterEqual()) ||
          isIncrement()) || (isDecrement()) || (isPlus()) || (isMinus()) ||
          (isStar()) || (isSlash()) || (isPlusAssign()) || (isMinusAssign()) ||
          (isSlashAssign()) || (isStarAssign()));
} // end isArithmetic operator

// Method isAssignmentOperator returns true if the token is an operator
public boolean isAssignmentOperator() {
  return ((isAssign()) || (isIncrement()) || (isDecrement()) || (isPlus()) ||
          (isMinus()) || (isStar()) || (isSlash()) || (isPlusAssign()) ||
          (isMinusAssign()) || (isStarAssign()) || (isSlashAssign()));
} // end isAssignmentOperator

/*
  Method isOperatorFollowedByNumber returns true if the current token is an
  arithmetic operator and the next token is a number
*/
public boolean isOperatorFollowedByNumber(AbstractToken nextToken) {
  return (isArithmeticOperator() && nextToken.isANumber());
} // end isTokenOperatorFollowedByNumber

/*
  Method isNotAbstractClassVoid returns true if the next token is not
  Abstract, Class and Void
*/
public boolean isNotAbstractClassVoid() {
  return (!isAbstract() && !isClass() && !isVoid());
} // end isNotAbstractClassVoid

// Method tokenIsLeftBrace returns true if the token is a left brace
public boolean isFirstLeftBrace(boolean firstLeftBrace) {
  return (isLeftBrace() && firstLeftBrace);
} // end tokenIsLeftBrace
```

```
  // Method setTokenCharacter sets the value of the 1st character of the string
  public void setTokenCharacter(char firstCharacter) {
    tokenCharacter = new Character(firstCharacter);
  } // end setTokenCharacter
} // end AbstractToken
```

2.0 EncapsulateFields.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        EncapsulateFields Class
 Description: The EncapsulateFields class is a class to implement
              the Strategy pattern for performing the Encapsulate Fields
              refactoring transformation
 Author:      Alain Coutu
 Date:        09/24/02
 Modified:
*/

import javax.swing.JOptionPane; // Import the JOptionPane class
import CIS5915JavaParser.AbstractToken; // Import the AbstractToken class

// Definition of the EncapsulateFields class
public class EncapsulateFields extends RefactoringStrategy{

  // Declare AbstractToken objects required for refactoring
  private AbstractToken currentToken, nextToken, previousToken;
  private AbstractToken nextSecondToken, nextThirdToken;

  // Define constants required for the token insertion
  private final int ONE = 1;
  private final int TWO = 2;
  private final int THREE = 3;
  private final int FIVE = 5;

  /*
    Method execute performs the Encapsulate Fields refactoring
    and returns the updated vector of tokens
  */
  public TokenIterator execute(TokenIterator codeTokenIterator) {

    // Create the AbstractToken objects
    currentToken = new AbstractToken();
    nextToken = new AbstractToken();
    previousToken = new AbstractToken();
    nextSecondToken = new AbstractToken();
    nextThirdToken = new AbstractToken();

    // Display a dialog box
    JOptionPane.showMessageDialog(null, "Performing Encapsulate Fields",
                                  "Refactoring", JOptionPane.PLAIN_MESSAGE);

    // Save the token object read
    currentToken.setToken(((AbstractToken)codeTokenIterator.nextObject())
                          .getToken());
    // Loop trhough the token vector
    while (!codeTokenIterator.isThereMoreObjects()) {
      // Save the next two tokens of the vector
      nextToken.setToken(((AbstractToken)codeTokenIterator.nextObject())
                         .getToken());
      codeTokenIterator.setCurrentVectorLocation();
      // If the token is Public
```

```
  if (currentToken.isPublic()) {
    // If the next token is not Abstract, Void or Class
    if (nextToken.isNotAbstractClassVoid()) {
      // Get the next token
      nextSecondToken.setToken(((AbstractToken)codeTokenIterator.
                               nextObject()).getToken());
      // If the token is not a left parenthese
      if (!nextSecondToken.isLeftParen()) {
        // Read the next token
        nextThirdToken.setToken(((AbstractToken)codeTokenIterator.
                                 nextObject()).getToken());
        // If the next token is a semi-colon
        if (nextThirdToken.isSemiColon()) {
          // Set and save the variable name
          setVariableImageRequired(nextSecondToken.getImage());
          saveVariableImageRequired(nextSecondToken.getImage());
          // Create a set method name
          createMethodName("set", getVariableImageRequired());
          // Create the set method tokens
          createSetMethodTokenVector(nextToken.getKind(),nextToken.getImage(),
                                     nextSecondToken.getKind(),
                                     nextSecondToken.getImage());
          // Create a get method name
          createMethodName("get", nextSecondToken.getImage());
          // Create the get method tokens
          createGetMethodTokenVector(nextToken.getKind(),nextToken.getImage(),
                                     nextSecondToken.getKind(),
                                     nextSecondToken.getImage());
          // Set the current token image and kind to Private
          currentToken.setImage("private");
          currentToken.setKind(PRIVATE);
        } // end if
      } // end if
    } // end if
  } // end if

  currentToken.setToken(nextToken.getToken()); // Save the next token
  // Reset the location in the vector
  codeTokenIterator.setVectorLocation(codeTokenIterator.
                                      getCurrentVectorLocation());
} // end while

codeTokenIterator.setToFirst(); // Set the vector location to the start
// Save the token object read
currentToken.setToken(((AbstractToken)codeTokenIterator.nextObject()).
                      getToken());

// Loop through the token vector
while (!codeTokenIterator.isThereMoreObjects()) {
  // Save the next tokens of the vector
  nextToken.setToken(((AbstractToken)codeTokenIterator.nextObject()).
                     getToken());
  // If the token is an arithmetic operator
  if (currentToken.isArithmeticOperator()) {
    // If the next token is the one we look for
    if (isTokenWanted(nextToken.getImage())) {
      codeTokenIterator.setCurrentVectorLocation();
```

```
    codeTokenIterator.setInsertVectorLocation();
    setVariableImageRequired(nextToken.getImage());
    // Create a get method name
    createMethodName("get", getVariableImageRequired());
    // Set the image of the next token
    nextToken.setImage(getMethodName());
    // Add parentheses tokens to the vector
    codeTokenIterator.add(objectCreator.createToken(LPAREN, "("),
                          codeTokenIterator.getInsertVectorLocation() + ONE);
    codeTokenIterator.add(objectCreator.createToken(RPAREN, ")"),
                          codeTokenIterator.getInsertVectorLocation()+ TWO);
    codeTokenIterator.setVectorLocation(codeTokenIterator.
                                        getCurrentVectorLocation());
  } // end if
} // end if

if (isTokenWanted(currentToken.getImage())) {
  if (nextToken.isAssignmentOperator()) {
    setVariableImageRequired(currentToken.getImage());
    nextSecondToken.setToken(((AbstractToken)codeTokenIterator.
                              nextObject()).getToken());
    codeTokenIterator.setCurrentVectorLocation();
    codeTokenIterator.setInsertVectorLocation();
    // Create a get method name
    createMethodName("set", getVariableImageRequired());
    currentToken.setImage(getMethodName());
    if (!nextToken.isAssign()) {
      // Create a get method name
      createMethodName("get", getVariableImageRequired());
      codeTokenIterator.add(objectCreator.createToken(IDENTIFIER,
                            getMethodName()),
                            codeTokenIterator.getInsertVectorLocation());
      codeTokenIterator.add(objectCreator.createToken(LPAREN, "("),
                        codeTokenIterator.getInsertVectorLocation() + ONE);
      codeTokenIterator.add(objectCreator.createToken(RPAREN, ")"),
                        codeTokenIterator.getInsertVectorLocation() + TWO);
      if (nextToken.isPlusAssign())
        codeTokenIterator.add(objectCreator.createToken(PLUS, "+"),
                      codeTokenIterator.getInsertVectorLocation() + THREE);
      if (nextToken.isMinusAssign())
        codeTokenIterator.add(objectCreator.createToken(MINUS, "-"),
                      codeTokenIterator.getInsertVectorLocation() + THREE);
      if (nextToken.isStarAssign())
        codeTokenIterator.add(objectCreator.createToken(STAR, "*"),
                      codeTokenIterator.getInsertVectorLocation() + THREE);
      if (nextToken.isSlashAssign())
       codeTokenIterator.add(objectCreator.createToken(SLASH, "/"),
                      codeTokenIterator.getInsertVectorLocation() + THREE);
      if (nextToken.isIncrement()) {
        codeTokenIterator.add(objectCreator.createToken(PLUS, "+"),
                      codeTokenIterator.getInsertVectorLocation() + THREE);
        nextSecondToken.setImage("ONE");
        nextSecondToken.setKind(IDENTIFIER);
      } // end if
      if (nextToken.isDecrement()) {
        codeTokenIterator.add(objectCreator.createToken(MINUS, "-"),
                      codeTokenIterator.getInsertVectorLocation() + THREE);
```

```
              nextSecondToken.setImage("ONE");
              nextSecondToken.setKind(IDENTIFIER);
            }// end if
            codeTokenIterator.add(objectCreator.createToken(RPAREN, ")"),
                                  codeTokenIterator.getInsertVectorLocation() + FIVE);
          } // end if
          if (nextToken.isAssign())
            codeTokenIterator.add(objectCreator.createToken(RPAREN, ")"),
                                   codeTokenIterator.getInsertVectorLocation() + ONE);
          nextToken.setImage("(");
          nextToken.setKind(LPAREN);
        } // end if
      } // end if

      // Save the next token
      currentToken.setToken(nextToken.getToken());
    } // end while

    // Set the vector to the last entry before the end of the vector
    codeTokenIterator.setToLast();
    // Insert the tokens in the collection
    codeTokenIterator.insertTokens(getInsertCodeCollectionIterator());
    return codeTokenIterator; // Return the updated vector
  } // end execute
} // end EncapsulateFields
```

3.0 Indent.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        Indent Class
 Description: The Indent class creates a string of blank spaces used when
              the tokens are printed after being parsed
 Author:      Alain Coutu
 Date:        08/23/02
 Modified:    08/27/02  Performed Encapsulate Fields refactoring.
              09/15/02  Performed Encapsulate Fields refactoring.
*/

public class Indent { // Define the class Indent

  private int indentLevelValue; // Declare the indent level value
  private String indentStringValue; // Declare the indent string value
  private int loopCounter; // Declare a loop counter
  private final int ZERO = 0; // Declare the constant 0
  private final int ONE = 1; // Declare the constant 1
  private final String BLANK = ""; // Declare a blank string
  private final String TWOSPACE = "  "; // Declare a two spaces string

  public Indent() { // Define the constructor
    // Initialize the indent level and its string value
    setIndentLevelValue(ZERO);
    setStringValue(BLANK);
  } // end constructor

  // Method setIndentLevel sets the indentation level required
  public void setIndentLevelValue(int value) {
    indentLevelValue = value;
  } // end setIndentLevel

  // Method getIndentLevel returns the indentation level
  public int getIndentLevel() {
    return indentLevelValue;
  } // end getIndentLevel

  // Method decreaseIndentLevel decrease by 1 the indentation level
  public void decreaseIndentLevel() {
    setIndentLevelValue(getIndentLevel() - ONE);
  } // end decreaseIndentLevel

  // Method increaseIndentLevel increase by 1 the indentation level
  public void increaseIndentLevel() {
    setIndentLevelValue(getIndentLevel() + ONE);
  } // end increaseIndentLevel

  // Method setIndentStringValue sets the indentation string to blank
  public void setStringValue(String value) {
    indentStringValue = value;
  } // end setIndentStringValue

  // Method increaseIndentStringValue increase the indentation value by 2 spaces
  public void increaseIndentStringValue() {
```

```
    setStringValue(getIndentStringValue() + TWOSPACE);
  } // end increaseIndentStringValue

  // Method setLoopCounter initialize the loop couinter top 0
  public void setLoopCounter() {
    loopCounter = ZERO;
  } // end initLoopCounter

  // Method loopNotDone returns true if the loop is not completed
  public boolean loopNotDone(){
    return (loopCounter++ < getIndentLevel());
  }// end loopNotDone

  /*
    Method setIndentStringValue set the indentation string value based on the
    indentation level
  */
  public void setIndentStringValue() {
    setLoopCounter();
    setStringValue(BLANK);
    while (loopNotDone())
      increaseIndentStringValue();
  } // end setIndentStringValue

  // Method getIndentStingValue returns the indentation string value
  public String getIndentStringValue() {
    return indentStringValue;
  } // end getIndentStringValue
} // end Indent
```

4.0 JavaDeParser.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        JavaDeParser Class
 Description: The JavaDeParser class performs operations required for taking
              the parsed token and rebuild the source code file
 Author:      Alain Coutu
 Date:        08/21/02
 Modified:    09/01/02  Refactored to implement Abstract factory pattern
              09/09/02  Builder pattern implementation
              09/10/02  Move method refactoring
              09/15/02  Performed Encapsulate Fields refactoring
              09/21/02  Implemented Strategy pattern for reformatting tokens
*/

// Import the CIS5915JavaParser.JavaParserConstants class
import CIS5915JavaParser.JavaParserConstants;
import CIS5915JavaParser.AbstractToken; // Import the AbstractToken class

// Define the JavaDeParser class
public class JavaDeParser implements JavaParserConstants{

  // Define variables for the class
  private static StringBuffer tokenToPrint;
  private static AbstractToken currentToken, previousToken, nextToken;
  private static Indent indentLevel;
  private static TokenContext tokenContext;

  public JavaDeParser() { // Define the constructor
    indentLevel = new Indent(); // Create new indent object
    // Create new AbstractToken objects
    currentToken = new AbstractToken();
    previousToken = new AbstractToken();
    nextToken = new AbstractToken();
    tokenContext = new TokenContext(); // Create a context object
  } // end constructor

  /*
    Method deparseToken takes each token from the vector and puts them
    together in a proper format
  */
  public static StringBuffer deparseToken(TokenIterator codeTokenIterator) {
    tokenToPrint = new StringBuffer(); // Create a new buffer for strings
    codeTokenIterator.setToFirst(); // Set the vector at the top
    // Get a vector object
    currentToken.setToken(((AbstractToken)codeTokenIterator.nextObject())
                          .getToken());
    previousToken.setToken(currentToken.getToken());
    // Loop through the vector
    while (!codeTokenIterator.isThereMoreObjects()) {
      // Get the next vector object
      nextToken.setToken(((AbstractToken)codeTokenIterator.nextObject())
                         .getToken());
      // Set the token reformat strategy
      tokenContext.setReformatStrategy(currentToken.getKind());
```

```
      // Reformat and print the token image
      tokenToPrint.append(tokenContext.reformat(currentToken.getImage(),
                                                nextToken.getKind(),
                                                previousToken.getKind(),
                                                indentLevel));
      previousToken.setToken(currentToken.getToken());
      currentToken.setToken(nextToken.getToken());
    } // end while
    return tokenToPrint;
  } // end deparseToken
} // end JavaDeparser
```

5.0 ObjectCollection.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        ObjectCollection Class
 Description: The ObjectCollection class is a class that defines methods
              to implement the Iterator pattern.  The methods add, removeElement
              and createCollectionIterator.  The add and removeElement methods
              are the Vector class add and removeElement methods.
 Author:      Alain Coutu
 Date:        08/20/02
 Modified:
*/

// Define the ObjectCollection class
public abstract class ObjectCollection {

  /*
    Abstract method createCollectionIterator returns a new collection
    iterator object
  */
  public abstract TokenIterator createCollectionIterator();

  // Abstract method add adds an object to the collection
  public abstract void add(Object collectionObject);

  // Abstract method add adds an object to the collection at the index provided
  public abstract void add(int index, Object collectionObject);

  // Abstract method removeElement removes an object from the collection
  public abstract void removeElement(Object collectionObject);
} // end ObjectCollection
```

5.0 ObjectIterator.java

```
/*
 Title:        Automated Refactoring Tool
 Name:        ObjectIterator Class
 Description: The ObjectIterator class is a class that defines methods to
              implement the Iterator pattern.  The methods are nextObject,
              isThereMoreObjects, add, setToFirst, setToLast, setVectorLocation,
              getCurrentCollectionindex, and insertTokens.
 Author:      Alain Coutu
 Date:        08/20/02
 Modified:
*/

// Define the ObjectIterator class
public abstract class ObjectIterator {

  // Abstract method nextObject returns the next object in the collection
  public abstract Object nextObject();

  // Abstract method add adds a token to the current vector
  public abstract void add(Object objectToAdd, int index);

  // Abstract method isThereMoreObjects returns if more objects exist
  public abstract boolean isThereMoreObjects();

  // Abstract method setToFirst sets the vector to the first item in the vector
  public abstract void setToFirst();

  // Abstract method setToLast sets the vector to the second to last item
  public abstract void setToLast();

  // Abstract method setVectorLocation sets the vector to specidfied index
  public abstract void setVectorLocation(int vectorLocation);

  // Abstract method getCurrentCollectionIndex returns the current index
  public abstract int getCurrentCollectionIndex();

  //Abstract method insertTokens inserts a vector into the current vector
  public abstract void insertTokens(TokenIterator insertTokenIterator);
} // end ObjectIterator
```

7.0 ProjectFile.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        ProjectFile Class
 Description: The ProjectFile class creates file objects used by the
              application.  It enables the open, close and save operations to be
              performed.
 Author:      Alain Coutu
 Date:        08/12/02
 Modified:    08/18/02  Modified to implement compiler capability
*/

import java.io.*; // Import the io class
import javax.swing.JOptionPane; // Import the JOptionPane class
import sun.tools.*; // Import the sun.tools package

public class ProjectFile { // Define the ProjectFile class

  // Declare files operations variables
  private BufferedReader sourceCodeReader;
  private BufferedWriter refactoredCodeWriter;
  private StringBuffer sourceCodeBuffer;
  private String sourceCodeLineRead;
  private String refactoredCodeLine;
  private File fileToOpen, fileToWrite;
  private ByteArrayOutputStream compileFileBAOS;
  private sun.tools.javac.Main compiler;
  private String fileName;

  // Method readFile opens a file and read its content line by line
  public StringBuffer readFile(String fileName) {

    fileToOpen = new File(fileName); // Create a file object
    setFileName(fileName);
    try { // Read and display the content of the file opened line by line
      sourceCodeReader = new BufferedReader(new FileReader(fileToOpen));
      sourceCodeBuffer = new StringBuffer();
      while ((sourceCodeLineRead = sourceCodeReader.readLine()) != null)
        sourceCodeBuffer.append(sourceCodeLineRead + "\n");
      sourceCodeReader.close(); // Close the file
    } // end try
    // Print a message box if there is an error opening the file
    catch (IOException ioexception) {
      JOptionPane.showMessageDialog(null, "File Open Error", "File Open",
                                    JOptionPane.ERROR_MESSAGE);
    } // end catch
    return sourceCodeBuffer; // Return the line of text read from the file
  } // end readFile

  // Method saveFile writes to a file line by line
  public void saveFile(String refactoredCodeLine) {
    fileToWrite = new File(getFileName()); // Create the file object
    setFileName(fileName); // Save the file name saved
    try { // Write the content of the file
      refactoredCodeWriter = new BufferedWriter(new FileWriter(fileToWrite));
```

```
      refactoredCodeWriter.write(refactoredCodeLine);
      refactoredCodeWriter.flush(); // Close the file
      refactoredCodeWriter.close();
    }
    // Print a message box if there is an error writing the file
    catch (IOException ioexception) {
      JOptionPane.showMessageDialog(null, "File Write Error", "File Write",
                                    JOptionPane.ERROR_MESSAGE);
    } // end catch
  } // end saveFile

  // Method compileFile compiles a java file
  public void compileFile() {
    String compileArguments[] = {"-classpath", System.getProperty(
                                 "java.class.path"), "-nowarn", getFileName()};
    JOptionPane.showMessageDialog(null, "Compiling" + getFileName(), "Compile",
                                  JOptionPane.INFORMATION_MESSAGE);
    compileFileBAOS = new ByteArrayOutputStream();
    compiler = new sun.tools.javac.Main(compileFileBAOS, "javac");
    if (!compiler.compile(compileArguments)) // If compilation failed
      JOptionPane.showMessageDialog(null, "Compilation of file " + getFileName()
                                    + " failed!", "Compile",
                                    JOptionPane.INFORMATION_MESSAGE);
    else // Successful compilation
      JOptionPane.showMessageDialog(null, "Compilation of file " + getFileName()
                                    + " completed!", "Compile",
                                    JOptionPane.INFORMATION_MESSAGE);
  } // end compileFile

  // Method getFileName returns the name of the file
  public String getFileName() {
    return fileName;
  } // end getFileName

  // Method setFileName set the name of the file
  public void setFileName(String newFileName) {
    fileName = newFileName;
  } // end setFileName
} // end ProjectFile
```

8.0 RefactoringContext

```
/*
 Title:       Automated Refactoring Tool
 Name:        RefactoringContext Class
 Description: The RefactoringContext class is the context class used to
              implement the Strategy pattern for the refactoring transformations
 Author:      Alain Coutu
 Date:        09/24/02
 Modified:
*/

// Define the RefactoringContext class
public class RefactoringContext {

  // Define a RefactoringStrategy object
  private RefactoringStrategy refactoringStrategy;

  // Method setRefactoringStrategy determines the strategy to use
  public void setRefactoringStrategy(String strategyToUse) {
    if (strategyToUse.equals("Replace Magic Number"))
      refactoringStrategy = new ReplaceMagicNumber();
    if (strategyToUse.equals("Encapsulate Fields"))
      refactoringStrategy = new EncapsulateFields();
  } // end setRefactorignStrategy

  // Method execute performs the refactoring strategy selected on the tokens
  public TokenIterator execute(TokenIterator codeTokenIterator) {
    return(refactoringStrategy.execute(codeTokenIterator));
  } // end execute
} // end RefactoringContext
```

9.0 RefactoringStrategy.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        RefactoringStrategy Class
 Description: The RefactoringStrategy class is an abstract class to implement
              the Strategy pattern for performing refactoirng transformations
 Author:      Alain Coutu
 Date:        09/24/02
 Modified:
*/

// Import the JavaParserConstants class
import CIS5915JavaParser.JavaParserConstants;
import CIS5915JavaParser.AbstractToken; // Import the AbstractToken class
import java.util.Vector; // Import the Vector class

// Define the RefactoringStrategy class
public abstract class RefactoringStrategy implements JavaParserConstants{

   // Declare vairables required by all stategies
  private final int ZERO = 0;
  private final int ONE = 1;
  private final int TWO = 2;
  private int variableIndexValue;
  private int variableRequiredKind;
  private boolean firstLeftBrace;
  private String refactoredVariableName, variableImageRequired, methodName;
  private AbstractToken previous, current;
  private Character tokenCharacter;
  protected TokenCreator objectCreator;
  private Vector insertTokenVector;
  private TokenCollection insertCodeCollection;
  private TokenCollection variableImage;
  private TokenIterator variableImageIterator;

  // Abstract method execute performs the refactoringstrategy
  public abstract TokenIterator execute(TokenIterator codeTokenIterator);

  public RefactoringStrategy() {// Define the constructor
    insertTokenVector = new Vector();
    insertCodeCollection = new TokenCollection();
    variableImage = new TokenCollection();
    variableImageIterator = variableImage.createCollectionIterator();
    current = new AbstractToken();
    previous = new AbstractToken();
  } // end constructor

  // Method setVariableIndexValue sets the refactored variable index to 0
  public void setVariableIndexValue() {
    variableIndexValue = ZERO;
  } // end setVariableIndexValue

  /*
    Method getVariableIndexValue returns the index value of the refactored
    variable
```

```
*/
public int getVariableIndexValue() {
  return variableIndexValue++;
} // end getVariableIndexValue

// Method setFirstLeftBrace sets if the first left brace was encountered
public void setFirstLeftBrace(boolean status) {
  firstLeftBrace = status;
} // end setFirstLeftBrace

// Method getFirstLeftBrace returns the status of variable firstLeftBrace
public boolean getFirstLeftBrace() {
  return firstLeftBrace;
} // end getFirstLeftBrace

/*
  Method setInsertLocation sets the location in the vector where new
  tokens will be inserted
*/
public void setInsertLocation(TokenIterator codeTokenIterator,
                              boolean firstLeftBraceFound) {
  if (firstLeftBraceFound) {
    codeTokenIterator.setInsertVectorLocation();
    setFirstLeftBrace(false);
  } // end if
} // end setInsertLocation

// Method setRefactoredVariableName generates a string for a new variable name
public void setRefactoredVariableName(String tokenValue) {
  refactoredVariableName = "REFACTORED" + this.getVariableIndexValue();
}  // end setRefactoredVariableName

// Method getRefactoredVariableName returns the new variable name
public String getRefactoredVariableName() {
  return refactoredVariableName;
} // end getRefactoredVariableName

// Method setVariableImageRequired sets the image of the type of variable
public void setVariableImageRequired(String tokenImage) {
  variableImageRequired =  tokenImage;
} // end setVariableImageRequired

// Method getVariableImageRequired sets the image of the type of variable
public String getVariableImageRequired() {
  return variableImageRequired;
} // end getVariableImageRequired

// Method setVariableRequiredKind sets the kind of the type of variable
public void setVariableRequiredKind(int tokenKind) {
  variableRequiredKind =  tokenKind;
} // end setVariableRequiredKind

// Method saveVariableImageRequired save the variable image required
public void saveVariableImageRequired(String tokenImage) {
  variableImage.add(tokenImage);
} // end setVariableImageRequired
```

```
// Method getVariableRequiredKind returns the kind of the type of variable
public int getVariableRequiredKind() {
  return variableRequiredKind;
} // end getVariableRequiredKind

/*
 Method isTokenWanted returns true if the token image is in the vector of
 images of token used for refactoring
*/
public boolean isTokenWanted(String tokenImage) {
  boolean found = false;
  variableImageIterator.setToFirst();
  while (!variableImageIterator.isThereMoreObjects()) {
    if (tokenImage.equals((String)variableImageIterator.nextObject())){
      found = true;
      break;
    } // end if
  } //   end while
  return(found);
} // end isTokenWanted

/*
  Method findSetFirstVariableOccurrence finds in the vector the first occurence
  of the variable required and get the image of the previous token
*/
public void findSetFirstVariableOccurence(TokenIterator codeTokenIterator,
                                          String previousTokenImage) {
  // Loop until one is found
  while (true) {
    // Get a vector object
    current.setToken(((AbstractToken)codeTokenIterator.nextObject()).getToken());
    // If it is the object wanted, saves the values
    if (current.getImage().equals(previousTokenImage)){
      setVariableImageRequired(previous.getImage());
      setVariableRequiredKind(previous.getKind());
      break;
    }// end if
    previous.setToken(current.getToken());
  }// end while
} // end findSetFirstVariableOccurrence

/*
  Method setMagicNumberTokenVector creates a vector of tokens to be inserted
  in the code vector
*/
public void setMagicNumberTokenVector(int nextTokenKind,
                                      String nextTokenImage) {

  objectCreator = TokenCreator.getFactory(TokenCreator.TOKEN);
  insertTokenVector.add(objectCreator.createToken(PRIVATE, "private"));
  insertTokenVector.add(objectCreator.createToken(FINAL, "final"));
  insertTokenVector.add(objectCreator.createToken(getVariableRequiredKind(),
                        getVariableImageRequired()));
  insertTokenVector.add(objectCreator.createToken(IDENTIFIER,
                        getRefactoredVariableName()));
  insertTokenVector.add(objectCreator.createToken( ASSIGN, "="));
  insertTokenVector.add(objectCreator.createToken(nextTokenKind,
```

```
                          nextTokenImage));
  insertTokenVector.add(objectCreator.createToken(SEMICOLON, ";"));
} // end setMagicNumberTokenVector

// Method getInsertTokenVector returns the vector of tokens to be inserted
public Vector getInsertTokenVector() {
  return insertTokenVector;
} // end getInsertTokenVector

// Method createMethodName creates a method name
public void createMethodName(String partOne, String variableName){
  tokenCharacter = new Character(variableName.charAt(ZERO));
  methodName = partOne + "Refactored" + tokenCharacter.toUpperCase
               (tokenCharacter.charValue()) + variableName.substring(ONE);
} // end createMethodName

// Method getMethodName returns the method name created
public String getMethodName() {
  return methodName;
} // end getMethodName

// Method getInsertCodeCollectionIterator returns the iterator
public TokenIterator getInsertCodeCollectionIterator() {
  insertCodeCollection.setCollection(getInsertTokenVector());
  return insertCodeCollection.createCollectionIterator();
} // end getInsertCodeCollectionIterator

// Method createSetMethodTokenVector creates the tokens for a set method
public void createSetMethodTokenVector(int nextTokenKind,
                                       String nextTokenImage,
                                       int nextSecondKind,
                                       String nextSecondImage) {
  objectCreator = TokenCreator.getFactory(TokenCreator.TOKEN);
  insertTokenVector.add(objectCreator.createToken(SINGLE_LINE_COMMENT,
                          "// Refactored set Method"));
  insertTokenVector.add(objectCreator.createToken(PUBLIC, "public"));
  insertTokenVector.add(objectCreator.createToken(VOID, "void"));
  insertTokenVector.add(objectCreator.createToken(IDENTIFIER, getMethodName()));
  insertTokenVector.add(objectCreator.createToken(LPAREN, "("));
  insertTokenVector.add(objectCreator.createToken(nextTokenKind,
                          nextTokenImage));
  insertTokenVector.add(objectCreator.createToken(IDENTIFIER, "argument"));
  insertTokenVector.add(objectCreator.createToken(RPAREN,")"));
  insertTokenVector.add(objectCreator.createToken(LBRACE, "{"));
  insertTokenVector.add(objectCreator.createToken(nextSecondKind,
                          nextSecondImage));
  insertTokenVector.add(objectCreator.createToken(ASSIGN, "="));
  insertTokenVector.add(objectCreator.createToken(IDENTIFIER, "argument"));
  insertTokenVector.add(objectCreator.createToken(SEMICOLON, ";"));
  insertTokenVector.add(objectCreator.createToken(RBRACE, "}"));
  insertTokenVector.add(objectCreator.createToken(SINGLE_LINE_COMMENT,
                          "// end Refactored set Method"));
}  // end createSetMethodTokenVector

// Method createGetMethodTokenVector creates the tokens for a get method
public void createGetMethodTokenVector(int nextTokenKind,
                                       String nextTokenImage,
```

```
                                        int nextSecondKind,
                                        String nextSecondImage) {
    objectCreator = TokenCreator.getFactory(TokenCreator.TOKEN);
    insertTokenVector.add(objectCreator.createToken(SINGLE_LINE_COMMENT,
                          "// Refactored get Method"));
    insertTokenVector.add(objectCreator.createToken(PUBLIC, "public"));
    insertTokenVector.add(objectCreator.createToken(nextTokenKind,
                          nextTokenImage));
    insertTokenVector.add(objectCreator.createToken(IDENTIFIER, getMethodName()));
    insertTokenVector.add(objectCreator.createToken(LPAREN,"("));
    insertTokenVector.add(objectCreator.createToken(RPAREN,")"));
    insertTokenVector.add(objectCreator.createToken(LBRACE, "{"));
    insertTokenVector.add(objectCreator.createToken(RETURN, "return"));
    insertTokenVector.add(objectCreator.createToken(nextSecondKind,
                          nextSecondImage));
    insertTokenVector.add(objectCreator.createToken(SEMICOLON, ";"));
    insertTokenVector.add(objectCreator.createToken(RBRACE, "}"));
    insertTokenVector.add(objectCreator.createToken(SINGLE_LINE_COMMENT,
                          "// end Refactored get Method"));
  }  // end createGetMethodVector
} // end RefactoringStrategy
```

10.0 RefactoringTool.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        RefactoringTool Class
 Description: The RefactoringTool class creates a menu bar for the GUI
              window.  It creates all the menu items and their sub-menus, when
              applicable as well as the action listener required for selecting
              the various events generated by selecting a menu item.
 Author:      Alain Coutu
 Date:        07/24/02
 Modified:    08/18/02  Modified to add parser and compiler capabilities
              08/28/02  Added Replace Magic Number refactoring capability
              09/09/02  Builder pattern implementation
              09/20/02  Updated CIS5915JavaParser package -
                        Added AbstractToken class
              09/24/02  Implemented strategy pattern for refactoring
*/

import javax.swing.*; // Import the swing class
import java.awt.*; // Import the awt class
import java.awt.event.*; // Import the awt.event class
import java.util.Vector; // Import the vector class
import CIS5915JavaParser.*; // Import the JavaParser class

// Definition the RefactoringTool class
public class RefactoringTool extends JFrame {

  private Box displayBox; // Initialize the box to contain the code views

  // Declare the menu bar, source code and refactrored code views
  private WindowMenuBarView projectMenuBar;
  private SourceCodeView sourceCodeDisplay;
  private SourceCodeView refactoredCodeDisplay;

  // Declare the input and out file objects
  private ProjectFile sourceCodeFile;
  private ProjectFile refactoredCodeFile;

  // Declare the window object to select files
  private JFileChooser fileChooserWindow;
  private int returnedValue;  // Declare the variable used by the file chooser

  // Declare the menu items controlled
  private JMenuItem exitMenuItem;
  private JMenuItem compileMenuItem;
  private JMenuItem openFileMenuItem;
  private JMenuItem closeFileMenuItem;
  private JMenuItem saveFileMenuItem;
  private JMenuItem selectLanguageParserMenuItem;
  private JMenuItem parseDataMenuItem;
  private JMenuItem encapsulateFieldsMenuItem;
  private JMenuItem replaceMagicNumberWithSymbolicConstantMenuItem;

  // Declare variables used by the controller
  private TokenCollection codeTokenCollection;
```

```
protected static JavaDeParser projectDeParser;
private RefactoringContext refactoringContext;

final int WINDOWWIDTH = 800; // Define the application window width
final int WINDOWHEIGHT = 600; // Define the application window height

// Define the constructor
public RefactoringTool() {
  super("Automated Refactoring Tool"); // Set up the title of the window

  // Create and add the menu bar
  projectMenuBar = new WindowMenuBarView();
  setJMenuBar(projectMenuBar);

  displayBox = Box.createHorizontalBox(); // Create the box container

  // Create and add two display areas for the source and refactored code
  sourceCodeDisplay = new SourceCodeView();
  displayBox.add(new JScrollPane(sourceCodeDisplay),BorderLayout.WEST);
  refactoredCodeDisplay = new SourceCodeView();
  displayBox.add(new JScrollPane(refactoredCodeDisplay),BorderLayout.EAST);
  getContentPane().add(displayBox);

  // Create the input and output file objects
  sourceCodeFile = new ProjectFile();
  refactoredCodeFile = new ProjectFile();

  codeTokenCollection = new TokenCollection(); // Create a token collection
  projectDeParser = new JavaDeParser(); // Create a new deparser object
  // Create a new refactoring context object
  refactoringContext = new RefactoringContext();

  // Set the controller for the various menu options
  setOpenFileMenuItem();
  setCloseFileMenuItem();
  setSaveFileMenuItem();
  setExitMenuItem();
  setCompileMenuItem();
  setSelectLanguageParserMenuItem();
  setParseDataMenuItem();
  setEncapsulateFieldsMenuItem();
  setReplaceMagicNumberWithSymbolicConstantMenuItem();

  setSize(WINDOWWIDTH, WINDOWHEIGHT); // Set the size of the window
  setVisible(true); // Display it
} // end constructor

/*
  Method setOpenFileMenuItem displays a JFileChooserWindow and permits the
  user to select a file to be opened
*/
public void setOpenFileMenuItem() {
  openFileMenuItem = projectMenuBar.getOpenFileMenuItem();
  openFileMenuItem.addActionListener (new ActionListener()
    {
      public void actionPerformed (ActionEvent e) {
        fileChooserWindow = new JFileChooser(); // Create the file chooser
```

```
        // Display the file chooser window and save the user selection
        returnedValue = fileChooserWindow.showOpenDialog(null);

        // If the user selected to open a file, get the file name and display
        // its content
        if (returnedValue == fileChooserWindow.APPROVE_OPTION) {
          sourceCodeDisplay.displayText(sourceCodeFile.readFile
                                        (fileChooserWindow.getSelectedFile().
                                        getPath()));
          refactoredCodeFile.setFileName(fileChooserWindow.getSelectedFile().
                                         getPath());
        }
      } // end actionPerformed
    } // end new
  );// end addActionListener parameter
} // end setOpenFileMenuItem

// Method setSaveFileMenuItem saves the output file
public void setSaveFileMenuItem() {
  saveFileMenuItem = projectMenuBar.getSaveFileMenuItem();
  saveFileMenuItem.addActionListener (new ActionListener()
    {
      public void actionPerformed (ActionEvent e) {
      fileChooserWindow = new JFileChooser(); // Create the file chooser

        // Display the file chooser window and save the user selection
        returnedValue = fileChooserWindow.showSaveDialog(null);

        // If the user selected to open a file, get the file name and display
        // its content
        if (returnedValue == fileChooserWindow.APPROVE_OPTION)
          refactoredCodeFile.saveFile(refactoredCodeDisplay.getText());
      } // end actionPerformed
    } // end new
  );// end addActionListener parameter
} // end setSaveFileMenuItem

// Method setCloseFileMenuItem closes any opened files and clear the displays
public void setCloseFileMenuItem() {
  closeFileMenuItem = projectMenuBar.getCloseFileMenuItem();
  closeFileMenuItem.addActionListener  (new ActionListener()
    {
      public void actionPerformed (ActionEvent e) {
        sourceCodeDisplay.clearDisplay();
        refactoredCodeDisplay.clearDisplay();
      } // end actionPerformed
    } // end new
  );// end addActionListener parameter
} // end setCloseFileMenuItem

// Method setExitMenuItem exits the program when the exit menu item is pressed
public void setExitMenuItem() {
  exitMenuItem = projectMenuBar.getExitMenuItem();
  exitMenuItem.addActionListener (new ActionListener()
    {
      public void actionPerformed (ActionEvent e) {
```

```
        System.exit(0);
      } // end actionPerformed
    } // end new
  );// end addActionListener parameter
} // end setExitMenuItem

// Method setCompileMenuItem compiles the refactored source code
public void setCompileMenuItem() {
  compileMenuItem = projectMenuBar.getCompileMenuItem();
  compileMenuItem.addActionListener (new ActionListener()
    {
      public void actionPerformed (ActionEvent e) {
        refactoredCodeFile.compileFile();
      } // end actionPerformed
    } // end new
  );// end addActionListener parameter
} // end setCompileMenuItem

// Method setSelectLanguageParserMenuItem selects the language BNF to use
public void setSelectLanguageParserMenuItem() {
  selectLanguageParserMenuItem = projectMenuBar.
                                 getSelectLanguageParserMenuItem();
  selectLanguageParserMenuItem.addActionListener (new ActionListener()
    {
      public void actionPerformed (ActionEvent e) {
        JOptionPane.showMessageDialog(null, "Select Language Parser",
                                      "Select Language Parser",
                                      JOptionPane.INFORMATION_MESSAGE);

      } // end actionPerformed
    } // end new
  );// end addActionListener parameter
} // end setSelectLanguageParserMenuItem

// Method setParseDataMenuItem parse the source code using the language BNF
public void setParseDataMenuItem() {
  parseDataMenuItem = projectMenuBar.getParseDataMenuItem();
  parseDataMenuItem.addActionListener (new ActionListener()
    {
      public void actionPerformed (ActionEvent e) {
        codeTokenCollection.setCollection(
                                          JavaParser.parseData
                                          (sourceCodeFile.getFileName()));
        refactoredCodeDisplay.displayText(projectDeParser.deparseToken
                                          (codeTokenCollection.
                                          createCollectionIterator()));
        refactoredCodeFile.saveFile(refactoredCodeDisplay.getText());
      } // end actionPerformed
    } // end new
  );// end addActionListener parameter
} // end setParseDataMenuItem

/*
  Method setEncapsulateFieldsMenuItem performs the refactoring transformation
  Encapsulate Fields
*/
public void setEncapsulateFieldsMenuItem() {
```

```
    encapsulateFieldsMenuItem = projectMenuBar.getEncapsulateFieldsMenuItem();
    encapsulateFieldsMenuItem.addActionListener (new ActionListener()
      {
        public void actionPerformed (ActionEvent e) {
          refactoredCodeDisplay.clearDisplay();
          refactoringContext.setRefactoringStrategy("Encapsulate Fields");
          refactoredCodeDisplay.displayText(projectDeParser.deparseToken(
                                            refactoringContext.execute(
                                            codeTokenCollection.
                                            createCollectionIterator())));
          refactoredCodeFile.saveFile(refactoredCodeDisplay.getText());
        } // end actionPerformed
      } // end new
    );// end addActionListener parameter
  } // end setEncapsulateFieldsMenuItem

  /*
    Method setReplaceMagicNumberWithSymbolicConstantMenuItem performs the
    refactoring transformation Replace Magic Number With Symbolic Constant
  */
  public void setReplaceMagicNumberWithSymbolicConstantMenuItem() {
    replaceMagicNumberWithSymbolicConstantMenuItem =
             projectMenuBar.getReplaceMagicNumberWithSymbolicConstantMenuItem();
    replaceMagicNumberWithSymbolicConstantMenuItem.
                                        addActionListener (new ActionListener()
      {
        public void actionPerformed (ActionEvent e) {
          refactoredCodeDisplay.clearDisplay();
          refactoringContext.setRefactoringStrategy("Replace Magic Number");
          refactoredCodeDisplay.displayText(projectDeParser.deparseToken(
                                            refactoringContext.execute(
                                            codeTokenCollection.
                                            createCollectionIterator())));
          refactoredCodeFile.saveFile(refactoredCodeDisplay.getText());
        } // end actionPerformed
      } // end new
    );// end addActionListener parameter
  } // end setReplaceMagicNumberWithSymbolicConstantMenuItem

  public static void main (String args[]) { // Declaration of the main method
    // Create a new desktop application
    RefactoringTool automatedRefactoringTool = new RefactoringTool();
    // Set the application default window close operation to exit
    automatedRefactoringTool.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  } // end main
} // end RefactoringTool
```

11.0 ReplaceMagicNumber.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        ReplaceMagicNumber Class
 Description: The ReplaceMagicNumber class is a class to implement
              the Strategy pattern for performing the Replace Magic Number with
              Symbolic Constant refactoring transformation
 Author:      Alain Coutu
 Date:        09/24/02
 Modified:
*/

import javax.swing.JOptionPane; // Import the JOptionPane class
import CIS5915JavaParser.AbstractToken; // Import the AbstractToken class

// Definition of the ReplaceMagicNumber class
public class ReplaceMagicNumber extends RefactoringStrategy{

  // Declare AbstractToken objects required for refactoring
  private AbstractToken currentToken, nextToken, previousToken;

  /*
    Method execute performs the Replace Magic Number with Symbolic Constant
    refactoring and returns the updated vector of tokens
  */
  public TokenIterator execute(TokenIterator codeTokenIterator) {

    // Create the AbstractToken objects
    currentToken = new AbstractToken();
    nextToken = new AbstractToken();
    previousToken = new AbstractToken();

    // Display a dialog box
    JOptionPane.showMessageDialog(null, "Performing Replace Magic Number " +
                                  "with Symbolic Constant", "Refactoring",
                                  JOptionPane.PLAIN_MESSAGE);

    // Set variables for controlling the insertion of new tokens
    setFirstLeftBrace(true);
    setVariableIndexValue();

    // Save the token objects read
    currentToken.setToken(((AbstractToken)codeTokenIterator.nextObject())
                          .getToken());
    previousToken.setToken(currentToken.getToken());

    // Loop through the token vector
    while (!codeTokenIterator.isThereMoreObjects()) {
      // Set the current location in the vector
      codeTokenIterator.setCurrentVectorLocation();
      // Get the location for inserting new tokens in the vector
      setInsertLocation(codeTokenIterator, currentToken.isFirstLeftBrace
                        (getFirstLeftBrace()));
      // Save the next token in the vector
      nextToken.setToken(((AbstractToken)codeTokenIterator.nextObject())
```

```
                        .getToken());
      /*
        If the token is an arithmetic operator and is followed by a number
        perform the refactoring
      */
      if (currentToken.isOperatorFollowedByNumber(nextToken)) {
        setRefactoredVariableName(nextToken.getImage());
        //Set the vector location after the first left brace
        codeTokenIterator.setVectorLocation(codeTokenIterator.
                                            getInsertVectorLocation());
        // Find the variable type
        findSetFirstVariableOccurence(codeTokenIterator, previousToken.
                                      getImage());
        // Reset the vector location
        codeTokenIterator.setVectorLocation(codeTokenIterator.
                                            getInsertVectorLocation());
        // Create tokens for the refactoring performed
        setMagicNumberTokenVector(nextToken.getKind(), nextToken.getImage());
        // Save the new token image
        nextToken.setImage(getRefactoredVariableName());
      } // end if
      // Save the previous token
      previousToken.setToken(currentToken.getToken());
      currentToken.setToken(nextToken.getToken()); // Save the next token
    } // end while

    // Set the vector location for insertion of new tokens
    codeTokenIterator.setVectorLocation(codeTokenIterator.
                                        getInsertVectorLocation());
    // Insert the tokens in the collection
    codeTokenIterator.insertTokens(getInsertCodeCollectionIterator());
    return codeTokenIterator; // Return the updated vecto
  } // end execute
} // end ReplaceMagicNumber
```

12.0 SourceCodeView.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        SourceCodeView Class
 Description: The SourceCodeView class creates a view for displaying source code
              on the application window.
 Author:      Alain Coutu
 Date:        07/31/02
 Modified:    08/31/02  Moved rows and columns constants at class level
*/

import javax.swing.*; // Import the swing class
import java.io.*; // Import the io class

// Define the SourceCodeView class
public class SourceCodeView extends JTextArea{

  private JTextArea sourceCode; // Declare the text display area
  private final int NUMBEROFROWS = 35; // Define the number of rows
  private final int NUMBEROFCOLUMNS = 50; // Define the number of columns

  public SourceCodeView() { // Define the constructor
    // Create the display area
    sourceCode = new JTextArea(" ", NUMBEROFROWS, NUMBEROFCOLUMNS);
  } // end constructor

  // Method displayText displays the content of a string buffer in the JTextArea
  public void displayText(StringBuffer sourceCodeBuffer) {
    append(sourceCodeBuffer.toString());
  } // end displayText

  // Method clearDisplay clears the content of the JTextArea
  public void clearDisplay() {
    setText("");
  } // end clearDisplay
} // end SourceCodeView
```

13.0 TokenCollection.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        TokenCollection Class
 Description: The TokenCollection class is a class that permits to travel
              within a collection of objects without the internal structure
              of the objects.
 Author:      Alain Coutu
 Date:        08/20/02
 Modified:
*/

import java.util.Vector; // Import the vector class

// Define the TokenCollection class
public class TokenCollection extends ObjectCollection {

  protected Vector collectionVector; // Define a vector to hold the collection

  // Define the constructor
  public TokenCollection() {
    collectionVector = new Vector(); // Create the new collection vector
  } // end constructor

  // Method createCollectionIterator creates and returns a new iterator object
  public TokenIterator createCollectionIterator() {
    return new TokenIterator(this);
  } // end createCollectionIterator

  // Method add adds an object to the collection
  public void add(Object collectionObject) {
    collectionVector.add(collectionObject);
  } // end add

  // Method add adds an object to the collection at the location specified
  public void add(int index, Object collectionObject) {
    collectionVector.add(index, collectionObject);
  } // end add

  // Method removeElement removes an object from the collection
  public void removeElement(Object collectionObject){
    collectionVector.removeElement(collectionObject);
  } // end removeElement

  // Method get returns the object at the given index
  public Object get(int objectIndex) {
    return collectionVector.get(objectIndex);
  } // end get

  // Method setCollection sets a vector as a collection
  public void setCollection(Vector vectorCreated) {
    this.collectionVector = vectorCreated;
  } // end setCollection
} // end TokenCollection
```

14.0 TokenContext.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        TokenContext Class
 Description: The TokenContext class is the context class used to implement
              the Strategy pattern for reformating tokens
 Author:      Alain Coutu
 Date:        09/21/02
 Modified:
*/

// Import the JavaParserConstants
import CIS5915JavaParser.JavaParserConstants;

// Define the TokenContext Class
public class TokenContext implements JavaParserConstants{

  // Declare a stategy object
  private TokenReformat reformatStrategy;

  public TokenContext() { // Define the constructor
    setReformatStrategy(DEFAULT); // Set the default strategy
  } // end constructor

  // Method setReformatStrategy determines the strategy class to use
  public void setReformatStrategy(int type) {
    switch (type) {
      case LBRACE: reformatStrategy = new TokenLBrace();
                   break;
      case RBRACE: reformatStrategy = new TokenRBrace();
                   break;
      case SEMICOLON: reformatStrategy = new TokenSemiColon();
                      break;
      case SINGLE_LINE_COMMENT: reformatStrategy = new TokenSingleLineComment();
                                break;
      case MULTI_LINE_COMMENT:
      case FORMAL_COMMENT: reformatStrategy = new TokenMultiLineComment();
                           break;
      case RPAREN: case LPAREN: case RBRACKET: case LBRACKET:
      case DOT: case BANG: reformatStrategy = new TokenParenBracketDotBang();
                           break;
      default:
        reformatStrategy =  new TokenDefault();
    }// end switch
  } // end setReformatStrategy

  // Method reformat executes the reformat strategy
  public String reformat(String currentTokenImage, int nextTokenKind,
                         int previousTokenKind, Indent indentLevel){
    return reformatStrategy.reformat(currentTokenImage, nextTokenKind,
                                     previousTokenKind, indentLevel);
  } // end reformat
} // end TokenContext
```

15.0 TokenCreator.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        TokenCreator Class
 Description: The TokenCreator Factory class is an abstract class to implement
              the abstract factory pattern
 Author:      Alain Coutu
 Date:        09/01/02
 Modified:
*/

import CIS5915JavaParser.AbstractToken; // Import the AbstractToken class
import java.util.Vector; // Import the Vector class

public abstract class TokenCreator { // Define the ObjectCreator class

    // Define the token factory object
    private static final TokenFactory tokenCreator = new TokenFactory();

    // Define the constant used to access the required factory
    protected static final int TOKEN = 100;

    /*
      Method getFactory returns a concrete factory object that is an instance
     of the concrete factory class that is appropriate for the given object
    */
    static final TokenCreator getFactory(int objectType) {
      switch (objectType) {
        case TOKEN: return tokenCreator;
      }
      // If the code reaches here, the object type has not been defined
      String errMsg = Integer.toString(objectType);
      throw new IllegalArgumentException(errMsg);
    } // end getFactory

    // Method createToken creates tokens using parameters
    public abstract AbstractToken createToken(int tokenKind, String tokenImage);
} // end TokenCreator
```

16.0 TokenDefault.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        TokenDefault Class
 Description: The TokenDefault class performs reformatting of the default
              tokens to be deparse as an implementation of the Strategy pattern
 Author:      Alain Coutu
 Date:        09/21/02
 Modified:
*/

// Define the TokenDefault class
public class TokenDefault extends TokenReformat{

 // Method reformat reformats the default tokens
  public String reformat(String currentTokenImage, int nextTokenKind,
                         int previousTokenKind, Indent indentLevel) {
    if ((nextTokenKind != DOT) && (nextTokenKind != SEMICOLON) &&
        (nextTokenKind != RPAREN) && (nextTokenKind != LPAREN) &&
        (previousTokenKind != DOT))
      return (currentTokenImage + " ");
    else
      return (currentTokenImage);
  } // end reformat
} // end TokenDefault
```

17.0 TokenFactory.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        TokenFactory Class
 Description: The TokenFactory class creates new AbstractToken object
 Author:      Alain Coutu
 Date:        09/01/02
 Modified:
*/

import CIS5915JavaParser.Token; // Import the Token class
import CIS5915JavaParser.AbstractToken; // Import the AbstractToken class

// Define the TokenFactory class
public class TokenFactory extends TokenCreator {

  private AbstractToken newToken;

  // Method createToken creates a new token based on the token kind and image
  public AbstractToken createToken(int tokenKind, String tokenImage) {
    // Build the token
    newToken = new AbstractToken();
    newToken.setToken(new Token());
    newToken.setImage(tokenImage);
    newToken.setKind(tokenKind);

    return newToken; // Return the token object
  } // end createToken
} // end TokenFactory
```

18.0 TokenIterator.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        TokenIterator Class
 Description: The TokenIterator class is a class that permits to traverse
              a collection of object and retrieve an object, move to the
              next object and verify if the collection is empty.
 Author:      Alain Coutu
 Date:        08/20/02
 Modified:    08/31/02  Added two methods required when performing refactoring:
                        setVectorLocation
                        insertTokens
              09/29/02  Modified for Strategy pattern implementation
*/

import CIS5915JavaParser.Token; // Import the Token class

// Define the TokenIterator class
public class TokenIterator extends ObjectIterator {

  private final int NOSELECTION = -1; // Define the constant -1
  private final int ONE = 1; // Define the constant 1
  private final int FOUR = 4; // Define the constant 4

  // Initialize a TokenCollection
  private TokenCollection tokenCollection;
  // Initialize the current index in the collection
  private int currentCollectionIndex = NOSELECTION;
  // Declare the current location and the inserting location in the vector
  private int currentVectorLocation, insertVectorLocation;

  // Define the constructor
  public TokenIterator(TokenCollection newCollection) {
    tokenCollection = newCollection; // Save the colleciton of objects
  } // end constructor

  // Method setToFirst sets the collection index to -1
  public void setToFirst() {
    currentCollectionIndex = NOSELECTION;
  } // end setToFirst

  // Method setToLast sets the collection index to the last element
  public void setToLast() {
    currentCollectionIndex = tokenCollection.collectionVector.size()-FOUR;
  } // end setToLast

  // Method setVectorLocation points the vector to the index received
  public void setVectorLocation(int vectorLocation) {
    currentCollectionIndex = vectorLocation;
  } // end setVectorLocation

  // Method getCurrentCollectionIndex returns the current index in the vector
  public int getCurrentCollectionIndex() {
    return currentCollectionIndex;
  } // end getCurrentCollectionIndex
```

```
  // Method insertTokens insert new tokens into the iterator object
  public void insertTokens(TokenIterator insertTokenIterator) {
    while (!insertTokenIterator.isThereMoreObjects())
      add(insertTokenIterator.nextObject(), ++currentCollectionIndex);
  } // end insertTokens

  // Method add adds a token to the vector at the location specified
  public void add(Object objectToAdd, int index) {
    tokenCollection.add(index, objectToAdd);
  } // end add

  // Method add adds a token to the vector
  public void add(Object objectToAdd) {
    tokenCollection.add(objectToAdd);
  } // end add

  // Method nextObject returns the next object of the vector
  public Object nextObject() {
    return tokenCollection.collectionVector.elementAt(++currentCollectionIndex);
  } // end nextObject

  // Mehtod isThereMoreObjects returns if there is more object in the vector
  public boolean isThereMoreObjects() {
    return (currentCollectionIndex >= tokenCollection.collectionVector.size()
                                      -ONE);
  } // end isThereMoreObjects

  // Method setCurrentVectorLocation sets the current location in the vector
  public void setCurrentVectorLocation() {
    currentVectorLocation = getCurrentCollectionIndex();
  } // end setCurrentVectorLocation

  // Method getCurrentVectorVectorLocation returns the current vector location
  public int getCurrentVectorLocation() {
    return currentVectorLocation;
  } // end getCurrentVectorLocation

  // Method setInsertVectorLocation sets the insert location in the vector
  public void setInsertVectorLocation() {
    insertVectorLocation = getCurrentCollectionIndex();
  } // end setInsertVectorLocation

  // Method getInsertVectorVectorLocation returns the insert vector location
  public int getInsertVectorLocation() {
    return insertVectorLocation;
  } // end getInsertVectorLocation
} // end TokenIterator
```

19.0 TokenLBrace.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        TokenLBrace Class
 Description: The TokenLBrace class performs reformatting of the Left Brace
              token to be deparse as an implementation of the Strategy pattern
 Author:      Alain Coutu
 Date:        09/21/02
 Modified:
*/

// Define the TokenLBrace class
public class TokenLBrace extends TokenReformat{

  // Method reformat reformats the LBrace token
  public String reformat(String currentTokenImage, int nextTokenKind,
                         int previousTokenKind, Indent indentLevel) {
    indentLevel.increaseIndentLevel(); // Increase the indent level
    indentLevel.setIndentStringValue(); // Set the indentation value
    if (nextTokenKind != SINGLE_LINE_COMMENT) {
      if (nextTokenKind != RBRACE)
        return (currentTokenImage + "\n" + indentLevel.getIndentStringValue());
      else
        return (currentTokenImage + "\n");
    }
    else
      return (currentTokenImage + " ");
  } // end reformat
} // end TokenLBrace
```

20.0 TokenMultiLineComment.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        TokenMultiLineComment Class
 Description: The TokenMultiLineComment class performs reformatting of the
              MultiLineComment and FormalComment tokens to be deparse as an
              implementation of the Strategy pattern
 Author:      Alain Coutu
 Date:        09/21/02
 Modified:
*/

// Define the TokenMultiLineComment class
public class TokenMultiLineComment extends TokenReformat{

  // Method reformat reformats the MultiLineComment and FormalComment tokens
  public String reformat(String currentTokenImage, int nextTokenKind,
                         int previousTokenKind, Indent indentLevel) {
    if (nextTokenKind != RBRACE)
      return (currentTokenImage + "\n\n" + indentLevel.getIndentStringValue());
    else
      return (currentTokenImage + "\n\n");
  } // end reformat
} // end MultiLineComment
```

21.0 TokenParenBracketDotBang.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        TokenParenBracketDotBang Class
 Description: The TokenParenBracketDotBang class performs reformatting of the
              Paren, Bracket, Dot and Bang tokens to be deparse as an
              implementation of the Strategy pattern
 Author:      Alain Coutu
 Date:        09/21/02
 Modified:
*/

// Define the TokenParenBracketDotBang class
public class TokenParenBracketDotBang extends TokenReformat{

  // Method reformat reformats the Paren, Bracket, Dot and Bang tokens
  public String reformat(String currentTokenImage, int nextTokenKind,
                         int previousTokenKind, Indent indentLevel) {

    if (nextTokenKind != LBRACE)
      return (currentTokenImage);
    else
      return (currentTokenImage + " ");
  } // end reformat
} // end TokenParenBracketDotBang
```

22.0 TokenRBrace.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        TokenRBrace Class
 Description: The TokenRBrace class performs reformatting of the right brace
              token to be deparse as an implementation of the Strategy pattern
 Author:      Alain Coutu
 Date:        09/21/02
 Modified:
*/

// Define the TokenRBrace class
public class TokenRBrace extends TokenReformat{

  // Method reformat reformats the RBrace token
  public String reformat(String currentTokenImage, int nextTokenKind,
                         int previousTokenKind, Indent indentLevel) {
    indentLevel.decreaseIndentLevel(); // Decrease the indent level
    indentLevel.setIndentStringValue(); // Set the indentation value
    if (nextTokenKind != SINGLE_LINE_COMMENT)
      return (indentLevel.getIndentStringValue() + currentTokenImage + "\n");
    else
      return (indentLevel.getIndentStringValue() + currentTokenImage + " ");
  } // end reformat
} // end TokenRBrace
```

23.0 TokenReformat.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        TokenReformat Class
 Description: The TokenReformat class is an abstract class to implement
              the Strategy pattern for reformating tokens
 Author:      Alain Coutu
 Date:        09/21/02
 Modified:
*/

// Import the JavaParserConstants class
import CIS5915JavaParser.JavaParserConstants;

// Define the TokenReformat class
public abstract class TokenReformat implements JavaParserConstants{

  // Abstract method reformat
  public abstract String reformat(String currentTokenImage, int nextTokenKind,
                                  int previousTokenKind, Indent indentLevel);
} // end TokenReformat
```

24.0 TokenSemiColon.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        TokenSemiColon Class
 Description: The TokenSemiColon class performs reformatting of the
              SemiColon token to be deparse as an implementation of the
              Strategy pattern
 Author:      Alain Coutu
 Date:        09/21/02
 Modified:
*/

// Define the TokenSemiColon class
public class TokenSemiColon extends TokenReformat{

  // Method reformat reformats the SemiColon token
  public String reformat(String currentTokenImage, int nextTokenKind,
                         int previousTokenKind, Indent indentLevel) {
    if (nextTokenKind != SINGLE_LINE_COMMENT) {
      if (nextTokenKind != RBRACE)
        return (currentTokenImage + "\n" + indentLevel.getIndentStringValue());
      else
        return (currentTokenImage + "\n");
    }
    else
      return (currentTokenImage + " ");
  } // end reformat
} // end TokenSemiColon
```

25.0 TokenSingleLineComment.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        TokenSingleLineComment Class
 Description: The TokenSingleLineComment class performs reformating of the
              Single Line Comment token to be deparse as an implementation of
              the Strategy pattern
 Author:      Alain Coutu
 Date:        09/21/02
 Modified:
*/

// Define the TokenSingleLineComment class
public class TokenSingleLineComment extends TokenReformat{

  // Method reformat reformats the token Single Line Comment
  public String reformat(String currentTokenImage, int nextTokenKind,
                         int previousTokenKind, Indent indentLevel) {
    if (nextTokenKind != RBRACE)
      return (currentTokenImage + "\n" + indentLevel.getIndentStringValue());
    else
      return (currentTokenImage + "\n");
  } // end reformat
} // end SingleLineComment
```

26.0 WindowMenu.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        WindowMenu Class
 Description: The WindowMenu class creates the top level menu for the
              application GUI menu bar by creating the caption and the
              mnemonic.
 Author:      Alain Coutu
 Date:        07/24/02
 Modified:
*/

import javax.swing.JMenu; // Import the JMenu class

// Declare the WindowMenu class
public class WindowMenu extends JMenu {

  // Define the constructor
  public WindowMenu (String windowMenuCaption, char windowMenuMnemonic){
    super (windowMenuCaption); // Create the caption of the menu
    super.setMnemonic(windowMenuMnemonic ); // Set the menu mnemonic
  } // end constructor
} // end WindowMenu
```

27.0 WindowMenuBarView.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        WindowMenuBarView Class
 Description: The WindowMenuBarView class creates a menu bar for the
              application GUI window by creating menus and the menu items
              associated.
 Author:      Alain Coutu
 Date:        07/24/02
 Modified:    08/05/02  Implemented the Model-View Controller
*/

import javax.swing.*; // Import the JMenuBar class

// Definition of the class WindowMenuBarView
public class WindowMenuBarView extends JMenuBar{

  // Declare the menu items of the menu bar
  private WindowMenuItem openMenuItem;
  private WindowMenuItem closeMenuItem;
  private WindowMenuItem saveMenuItem;
  private WindowMenuItem exitMenuItem;
  private WindowMenuItem selectLanguageParserMenuItem;
  private WindowMenuItem parseDataMenuItem;
  private WindowMenuItem encapsulateFieldsMenuItem;
  private WindowMenuItem replaceMagicNumberWithSymbolicConstantMenuItem;
  private WindowMenuItem compileMenuItem;

  // Declare the functions returning a menu item
  public JMenuItem getExitMenuItem() { return exitMenuItem; }
  public JMenuItem getCompileMenuItem() { return compileMenuItem; }
  public JMenuItem getOpenFileMenuItem() { return openMenuItem; }
  public JMenuItem getCloseFileMenuItem() { return closeMenuItem; }
  public JMenuItem getSaveFileMenuItem() { return saveMenuItem; }
  public JMenuItem getSelectLanguageParserMenuItem()
                                        { return selectLanguageParserMenuItem; }
  public JMenuItem getParseDataMenuItem() { return parseDataMenuItem; }
  public JMenuItem getEncapsulateFieldsMenuItem()
                                           { return encapsulateFieldsMenuItem; }
  public JMenuItem getReplaceMagicNumberWithSymbolicConstantMenuItem()
                      { return replaceMagicNumberWithSymbolicConstantMenuItem; }

  // Define the constructor
  public WindowMenuBarView() {

    // Create and add the file menu to the menu bar
    WindowMenu windowFileMenu = new WindowMenu("File", 'F');
    this.add(windowFileMenu);

    // Create the Open, Save, Close and Exit menu items
    openMenuItem = new WindowMenuItem("Open", 'O');
    windowFileMenu.add(openMenuItem);
    closeMenuItem = new WindowMenuItem("Close", 'C');
    windowFileMenu.add(closeMenuItem);
    saveMenuItem = new WindowMenuItem("Save", 'S');
```

```
    windowFileMenu.add(saveMenuItem);
    exitMenuItem = new WindowMenuItem("Exit", 'E');
    windowFileMenu.add(exitMenuItem);

    // Create and add the parser menu option to the file menu
    WindowMenu windowParserMenu = new WindowMenu("Parser", 'P');
    this.add(windowParserMenu);

    // Create the Select Language Parser and Parse Data menu items
    selectLanguageParserMenuItem = new WindowMenuItem("Select Language Parser",
                                                      'L');
    windowParserMenu.add(selectLanguageParserMenuItem);
    parseDataMenuItem = new WindowMenuItem("Parse Data", 'P');
    windowParserMenu.add(parseDataMenuItem);

    // Create and add the refactoror menu option to the file menu
    WindowMenu windowRefactororMenu = new WindowMenu("Refactor", 'R');
    this.add(windowRefactororMenu);

    // Create the implemented refactoring transformations menu items
    encapsulateFieldsMenuItem = new WindowMenuItem ("Encapsulate Fields", 'F');
    windowRefactororMenu.add(encapsulateFieldsMenuItem);
    replaceMagicNumberWithSymbolicConstantMenuItem =
        new WindowMenuItem ("Replace Magic Number With Symbolic Constant", 'M');
    windowRefactororMenu.add(replaceMagicNumberWithSymbolicConstantMenuItem);

    // Create and add the compile menu and menu item option to the file menu
    WindowMenu windowCompileMenu = new WindowMenu("Compile", 'C');
    this.add(windowCompileMenu);
    compileMenuItem = new WindowMenuItem ("Compile Code", 'C');
    windowCompileMenu.add(compileMenuItem);
  } // end constructor
} // end WindowMenuBar
```

28.0 WindowMenuItem.java

```
/*
 Title:       Automated Refactoring Tool
 Name:        WindowMenuItem Class
 Description: The WindowMenuItem class creates a menu item for the application
              GUI window menu bar view.
 Author:      Alain Coutu
 Date:        07/31/02
 Modified:
*/

import javax.swing.JMenuItem; // Import the JMenuItem class

// Define the WindowMenuItem class
public class WindowMenuItem extends JMenuItem {
  // Define the constructor
  public WindowMenuItem(String menuItemName, char menuMnemonic) {
    super(menuItemName); // Set the menu item name
    super.setMnemonic(menuMnemonic); // Set the menu item mnemonic
  } // end contstructor
} // end WindowMenuItem
```

AUTOMATED REFACTORING TOOL (ART)

APPENDIX G

CODING CONVENTIONS

APPENDIX G

TABLE OF CONTENTS

1.0 FILES 1

1.1 Names 1
1.2 File Type Suffix 1
1.3 File Organization 1
1.4 Source File Organization 1
1.4.1 Beginning Comments 1
1.4.2 Import Statements 1
1.4.3 Class and Interface Declarations 1

2.0 LINE FORMATTING 2

2.1 Line Length 2
2.2 Wrapping Lines 2
2.3 Indentations 2

3.0 COMMENTS 2

3.1 Block Comments 2
3.2 Single-Line Comments 3
3.3 Trailing Comments 3

4.0 DECLARATIONS 3

4.1 Number of Declarations Per Line 3
4.2 Initialization 3
4.3 Placement 3
4.4 Class and Interface Declarations 3

5.0 STATEMENTS 4

5.1 Simple Statements 4
5.2 Compound Statements 4
5.3 *retur*n Statements 4
5.4 if, if-else, if else-if else Statements 4
5.5 *for* Statements 5
5.6 *while* Statements 5
5.7 *do-while* Statements 5
5.8 *switch* Statements 5
5.9 *try-catch* Statements 6

6.0 WHITE SPACES 6

6.1 Blank Lines 6
6.2 Blank Spaces 6

7.0 NAMING CONVENTIONS 7

8.0 PROGRAMMING PRACTICES 8

8.1 Providing Access to Instance and Class Variables 8
8.2 Referring to Class Variables and Methods 8
8.3 Constants 8
8.4 Variable Assignments 8
8.5 Parentheses 8
8.6 Expressions before ‘?’ in the Conditional Operator 8

The following coding conventions have been adapted from the Sun Microsystems Java Coding Conventions [Sun].

1.0 Files

This section deals with the nomenclature used in naming files, the suffixes for the files created and the general organization of the source files.

1.1 Names

File names will have the same name as the public class contained in the file. Refer to section 7.0 for the naming conventions of classes. This convention is a result of using Borland JBuilder5 as the Java development platform.

1.2 File Type Suffix

All Java source code files will have the ***.java*** suffix.

1.3 File Organization

Files will be organized by section separated by blank lines and a comment section. A file will contain no more than 1500 lines.

1.4 Source File Organization

A Java source file will contain one public class or interface. If private classes and/or interfaces are associated with the public class, they can be inserted in the same source file. The public class will be the first class or interface in the source file.

1.4.1 Beginning Comments

All Java source file will start with a comment section formatted as follows:

```
/*
 * Title: The title of the project
 * Class Name: The name of the class
 * Description: A description of the class
 * Author: The author of the class
 * Version: The version number of the class
 * Date: The date the class was created
 * Modified: Date modification were performed and what the modifications were
 * Copyright Notice: Copyright notice of the class
 */
```

1.4.2 Import Statements

The first line(s) following the beginning comments will be the import statements required for the use of various Java pre-defined classes and methods. The format is as follows:

import java.awt;

1.4.3 Class and Interface Declarations

The following table describes the class/interface declaration content and the order they appear in the Java source file.

Class/Interface Content	*Description*
Beginning comments	See section 1.4.1 for format and content
class/interface statement	As per Java language standard
Implementation comment	This section contains any comments not appropriate in the Beginning comment section.
Variables declarations	All class (static) variables are declared first starting with public, protected, and private variables.
Instance variables declarations	All instance variables used in the class are declared following the variable declarations starting with public, protected, and private instance variables.
Constructors	
Methods	The methods defined for the class are declared last. They will be grouped by functionality. All public methods will be defined first and then the private methods.

2.0 Line Formatting

The following section defines the conventions for formatting lines of code or comments used in the Java source file.

2.1 Line Length

Lines of code or comment will not exceed 80 characters in length.

2.2 Wrapping Lines

If a line of code does not fit in the length specified in section 2.1, it will be broken according to the following rules and aligned with the beginning of the previous line:

a) after a comma; and

b) before an operator.

2.3 Indentations

The unit of indentation will be two spaces.

3.0 Comments

In Java, there are two ways to identify a comment. It starts with // or is enclosed in /*…*/. Comments are used to clarify and provide overview of the code. For example, when declaring variables, a comment line is added to aid understanding. However, comments regarding the design decisions or location of files are not appropriate. The are several ways to provide comments in Java and the following sections discuss them.

3.1 Block Comments

Block comments are used when describing files, data structures, methods and algorithms. They are used at the beginning of files, see section 1.4.1, at the beginning of methods and within

a method itself. Block comments at the beginning of a method or within a method will be formatted as follows:

```
/*
 * Insert the text here
 * and more lines here
 */
```

3.2 Single-Line Comments

Single line comments are used to amplify the code that follows and they are started using the //, are indented with the next line and are preceded by a blank line. If more than one line is required, block comments will be used instead.

3.3 Trailing Comments

Trailing comments are used similarly to single-line comments but the only difference is that they are short and are located at the end of a line.

4.0 Declarations

The following section details the various conventions for initialization, declarations, and placements.

4.1 Number of Declarations Per Line

Variable declarations will be limited to one per line to permit the insertion of comments for each variable being declared.

4.2 Initialization

Local variables will be initialized where they are declared. There is one exception: when the variables initial value is depending on some code execution.

4.3 Placement

All variable declarations, public, protected, and private need to be located at the beginning of code blocks. A code block is defined as being enclosed by "curly braces" {}. The exception to this rule is *for* loop statements index declarations where the declarations are part of the statement. The following example shows both these rules.

```
void myMethod() { //beginning of method block
  int myInteger = 0;
  if (condition) {  // beginning of if block
    int myInteger1 = 0;
  }
  for (int I = 0; I < aValue; I++) {…}  // for loop statement
}
```

4.4 Class and Interface Declarations

The declaration of all Java classes and interfaces will follow the following formatting rules:

a) there is no space between a class method name and the parenthesis starting the parameter's list;
b) an open brace will appear at the end of the same line as the declaration statement;
c) the associated closing brace will be on a line by itself and aligned with the first character of the declaration statement; and
d) if the declaration is a null statement, both braces will be on the same line.

5.0 Statements

There are many different statements used in Java and each have their own formatting rules. The following sections define the conventions for each of the various statements.

5.1 Simple Statements

Each line will contain at most one simple statement, such as **argv++**.

5.2 Compound Statements

A compound statement is one that contains multiple simple statements enclosed within braces. It is governed by the following rules:

a) the enclosed statement will be indented one level from the initial statement;
b) the opening brace will be located at the end of the line that begins the compound statement;
c) the closing brace of the compound statement will be formatted as per section 4.4c; and
d) braces are used to enclosed those statements that are part of a control structure, such as *if-else*.

5.3 *retur*n Statements

Return statements will not use parentheses unless it will make the return value more obvious.

5.4 if, if-else, if else-if else Statements

The class of *if-else* statements will follow the following formats:

a)
```
if (condition) {
  statement;
}
```

b)
```
if (condition) {
  statement;
}
else {
  statement;
}
```

c)
```
if (condition) {
  statement;
}
else if (condition) {
```

```
    statement;
  }
  else {
    statement;
  }
```

5.5 *for* Statements

A *for* statement will have the following format:

```
for (initialization; condition; update) {
  statement;
}
```

If it is an empty *for* statement, where all the work is done in the *for* statement declaration, it will have the following format:

```
For (initialization; condition; update);
```

The comma operator can be used to initialize or update a variable in the *for* statement declaration statement. At most, three variables will be initialized and/or updated using the comma operator. If more than three variables are required, the initialization must be performed before the *for* loop or the update must be performed after the *for* loop.

5.6 *while* Statements

A *while* statement will have the following format:

```
while (condition) {
  statement;
}
```

If it is an empty *while* statement, it will have the following format:

```
while (condition);
```

5.7 *do-while* Statements

A *do-while* statement will have the following format:

```
do {
  statement;
} while (condition);
```

5.8 *switch* Statements

A *switch* statement will have the following format:

```
switch (condition) {
  case A:
    statement;
    break;
  case BCD:
    statement;
```

```
    /* falls through */
  case M:
    statement;
    break;
  default:
    statement;
    break;
}
```

5.9 *try-catch* Statements

A *try-catch* statement will have the following format:

```
try {
  statement;
}
  catch (ExceptionClass e){
    statement;
}
```

The *try-catch* statement may be followed by *finally*, which executes regardless or not the *try* block was executed successfully. It will have the following format:

```
try {
  statement;
}
  catch (ExceptionClass e){
    statement;
}
  finally {
    statement;
}
```

6.0 White Spaces

Good programming practices include the use of blank lines and/or white spaces through out the code to increase its readability. The following sections define the conventions for the use of blank lines and white spaces.

6.1 Blank Lines

Blank lines will be used to improve readability of the code. One blank line will be used:

a) to separate sections in a source file;
b) between class and interface definitions;
c) between methods;
d) between the local variables declaration and the first statement in a method;
e) before a block comment;
f) before a single line comment; and
g) between logical sections inside a method.

6.2 Blank Spaces

Blank spaces will be used in the following circumstances:

a) between a keyword and a parenthesis except for a method name;
b) after a comma in argument list;
c) between operand and binary operators except **.**, and the unary operators will never be separated from their operand;
d) between expressions in a *for* statement; and
e) between the casting operator and the variable being casted.

7.0 Naming Conventions

To enhance the readability of programs, naming conventions are used. They also provide information regarding the identifier's function. The following table defines the various naming conventions for Java identifier types:

Identifier Type	***Naming Conventions***	***Examples***
Classes	Names for classes will be nouns, with the first letter of each internal word capitalized. Words will be whole and not abbreviations.	class DrawingCanvas; class MenuCommandFactory;
Interfaces	Names for interfaces will follow the same naming conventions as for classes.	interface ButtonAbstractFactory; interface Addition;
Methods	Names for methods will be a combination of verbs and nouns where the first word of the method name will be a verb and its first letter in lowercase and the following words having their first letter capitalized.	method getName(); method isSummerOver();
Variables	Names for variables will be a combination of words with the first letter in lowercase and the first letter of all other internal words capitalized. Names will be short, meaningful and not start with an underscore or dollar sign. One letter name will not be used.	int backgroundColor; float rectangleHeight;
Constants	Names for constants will be capitalized and have underscore between each word in the name.	static final MAX_HEIGTH = 4; static final CPU_SPEED = 400;
Refactored methods, variables, constants	Names of methods, variables and constants created because of refactoring will follow the same naming convention as stated above but with the added noun "Refactored".	method getRefactoredName (); int refactoredRectangleHeight; static final refactoredLoopCounter;

8.0 Programming Practices

Programming languages have suggested programming practices compiled over the life of the language. Java is no exception. The following sections define the Java programming practices.

8.1 Providing Access to Instance and Class Variables

Instance or class variables will always be declared as private.

8.2 Referring to Class Variables and Methods

When referring or accessing a class (static) variable or a method, the use of objects will be avoided. A class name will be used instead. For example, classMethod() or Aclass.classMethod().

8.3 Constants

Numerical constants will not be coded directly. A constant variable will be used instead. For example, instead of the statement: myValue = circleRadius * 3.14, the following statement will be used: myValue = circleRadius * PI where PI will be declared as a constant of value 3.14.

8.4 Variable Assignments

The assignment of multiple variables to the same value in one statement will not be used. Each assignment will be done using one statement per variable. The assignment operator will not be used if it can easily be confused with the equality operator. Embedded assignment, to improve the performance will not be used.

8.5 Parentheses

When statements make use of multiple operators, parentheses will be used to separate the operators and make the statement more readable.

8.6 Expressions before '?' in the Conditional Operator

Expression containing a binary operator that appears before the *?* in the ternary operator *?:* will be enclosed in parentheses.

Appendix H - List of References

[Arcelli] Arcelli, Francesca, Tisato, Francesco, Exploiting Analogy for a Design Pattern Recognition and Classification, University of Milano-Bicocca

[Booch] Booch, Grady, Rumbaugh, Jim, Jacobsen, Ivar, The Unified Modeling Language User Guide, Addison-Wesley, 1999

[Fowler] Fowler, Martin, Refactoring: Doing Design After the Program Runs, Distributed Computing, September 1998.

[Fowler00] Fowler, Martin, Refactoring: Improving the Design of Existing Code, Addison-Wesley, 2000.

[Gosling] Gosling James, Joy, Bill, Steele, Guy, Bracha, Gilad, The Java Language Specification Second Edition, Sun Microsystems, 2000

[Grand] Grand,Mark, Patterns in Java, Volume 1 A Catalogue of Reusable Design Patterns Illustrated with UML, Wiley, 1998

[Grimmond] Grimmond, Ray, Whitesides, Christie, Generating Java Code for Design Patterns, Threshold Computer Systems Inc., 1998

[IEEE1012] IEEE Std 1012-1998, Standard for Software Verification and Validation.

[IEEE1016] IEEE Std 1016-1998, IEEE Recommended Practice for Software Design Descriptions.

[IEEE1016.1] IEEE Std 1016.1-1993, IEEE Guide to Software Design Descriptions.

[IEEE1233] IEEE Std 1233, 1998 Edition, IEEE Guide for Developing System Requirements Specifications.

[Kataoka] Kataoka, Yoshio, Ernst, Michael D., Griswold, William G, Notkin, David, Automated Support for Program Refactoring using Invariants, Proceedings of IEEE International Conference on Software Maintenance, 2001.

[Kleinoeder] Kleinoeder, Jurgen, Golm, Michael, Transparent and Adaptable Object Replication Using a Reflective Java, TR-I4-96-07, University of Erlangen-Nurmberg, 1996

[Mens] Mens, Tom, Demeyer, Serge, Janssens, Dirk, Formalising Behaviour Preserving Program Transformations, Proceedings of International Conference on Graph Transformation, 2002

[Opdyke] Opdyke, William F., Refactoring Object-Oriented Frameworks, Doctor of Philosophy in Computer Science Thesis, University of Illinois at Urbana-Champaign, 1992.

[Opdyke96] Opdyke, William F., Object-Oriented Refactoring, Legacy Constraints and Reuse, 8th Workshop on Institutionalizing Software and Reuse, 1996.

[Opdyke99] Opdyke, William F., Refactoring, Reuse and Reality, Lucent Technologies/Bell Labs, 1999.

[Riggs] Riggs, Roger, Stoecklin, Sara, Automated Process of Code Refactoring, SCI 2002/ ISAS 2002/ PDTSD'02, 6th World Multiconference on Systemics, Cybernetics and Informatics and The 8th international Conference on Information System Analysis and Synthesis, July 2002, (CD proceedings).

[Roberts] Roberts, Don, Brant, John, Johnson, Ralph, Why Every Smalltalker Should Use the Refactoring Browser, Smalltalk Report, 1996.

[Roberts97] Roberts, Don, Brant, John, Johnson, Ralph, A Refactoring Tool for Smalltalk, Theory and Practice of Object Systems Special Issue: Object-Oriented Software Evolution and Re-Engineering, Voloume 3, Number 4, 1997.

[Roberts99] Roberts, Don, Practical Analysis for Refactoring, Doctor of Philosophy in Computer Science Thesis, University of Illinois at Urbana-Champaign, 1999.

[Schubert] Schubert, Frank, A Reflective Architecture for an Adaptable Object-Oriented Operating System Based on C++, Chemnitz University of Technology, 1997

[Sun] Code Conventions for the Java™ Programming Language, Sun MicroSystems, 1999.

[Tokuda] Tokuda, Lance, Batory, Don, Automated Three Modes of Evolution for Object-Oriented Software Architectures, 5th USENIX Conference on Object-Oriented Technologies and Systems, 1999.

[Tokuda01] Tokuda, Lance, Batory, Don, Evolving Object-Oreinted Designs with Refactorings, Automated Software Engineering, 8, Proceedings, 2001, pages 89-120.

[Wiki] What is Refactoring, http://c2.com/cgi/wiki?/WhatIsRefactoring.

[Wiki1] ReFactor, http://c2.com/cgi/wiki?/ReFactor.