String Manipulation in Java

Lecture 5
CGS 3416 Spring 2020

January 21, 2020

Strings in Java

- In Java, a string is an object. It is not a primitive type.
- The String class is used to create and store immutable strings.
  - Immutable objects are objects that don't change once created.
  - Kinda like "final" primitive types.
- Class StringBuilder creates objects that store flexible and changeable strings.
  - We'll learn this later on in the course.

The String class

- Part of java.lang package
- 13 constructors and close to 50 methods
- String class API from java.oracle.com – full listing of String class features
- Once you build a String object, it is fixed – it cannot be changed.
  - This is easier than it sounds. The only methods that can alter or set the instance variables are the constructors. All other methods that seem to change a string do so by returning a brand new String object
  - You can assign a String reference variable to a new string, discarding the old one

Two ways to construct a String

One constructor allows the use of a string literal as the parameter.
Example string constructions:

```
String greeting = new String("Hello, World!");
String name = new String("Marvin Dipwart");
String subject = new String("Math");

// Enclose the String literal in quotes, a shorthand
notation for building strings

String sentence = "The quick brown fox sat around for
                a while";

// this is not quite equivalent to using the
//constructor above, but you still get a string
//variable
```

String Pool

Java String literal is created by using double quotes. Before creating a String literal first looks for String with the same value in the String pool, if found it returns the reference else it creates a new String in the pool and returns the reference:

```
String str1 = "Hello";
String str2 = = "Hello";
```

```
String object created using \new" keyword it always
create a new object in heap memory.
```

```
String str3 = new String("Hello");
String str4 = new String("Hello");
```

Don't try to compare strings by using $==$, $<$, $>$, etc. These would only compare the String reference variables, not the String objects themselves.

The equals() method

equals() – for comparing two strings (i.e. their contents), returns true or false

```
if (str1.equals(str2))
     System.out.print("The strings are the same");
```

equalsIgnoreCase() - just like equals(), except that the case of the letters doesn't matter in making a match. For instance, "Apple" would be equal to "apple" with this method.

The compareTo() method

compareTo() – also for comparing two strings, good for sorting.

```
if (str1.compareTo(str2) <0)
     System.out.print("str1 comes before str2 in
             lexicographic ordering");
else if (str1.compareTo(str2) == 0)
     System.out.print("str1 is the same as str2");
else if (str1.compareTo(str2) >0)
     System.out.print("str2 comes before str1 in
             lexicographic ordering");
```

Empty Strings

The constructor with no parameters allows the building of an empty string:

```
String s = new String();
// s refers to an empty string object
```

Note that if you only declare a String variable, but you do not assign it to anything, it is not yet attached to any string:

```
String s1; // s1 does not refer to any string yet
```

Concatenation

- `concat()` – String concatenation. Returns a concatenation of two strings.

```
String s1 = "Dog";
String s2 = "food";
String s3 = s1.concat(s2);
                //s3 now stores "Dogfood"
                //note:  s1 and s2 are NOT changed
```

- The + symbol also performs String concatenation (as we've already used in print statements).

```
String s1 = "Cat";
String s2 = "nap";
String s3 = s1 + s2;
      //s3 now stores "Catnap" (s1, s2 unchanged)
```

Substrings

- `substring()` – extracts part of a string and returns it.
- Takes in two parameters (begin index and end index) or 1 parameter (begin index).
- First character in a String has index 0. Substring returned is the index range [begin,end).

Substrings

```
String s1 = "Hello, World";
String s2 = s1.substring(0,5);// s2 is now "Hello".
                 // picks up indices 0 - 4

String s3 = s1.substring(0,7) + "Dolly";
System.out.print(s3);// prints "Hello, Dolly"
System.out.print(s3.substring(4));//prints "o, Dolly"

// can even use substring on string literals
String s4= "What's up doc?".substring(10,13);
                 // s4="doc"
```

String length

- `length()` – returns a string's length (number of characters).

```
String s1 = "Hello";
String s2 = "Goodbye world";

System.out.print(s1.length()); // output:  5
System.out.print(s2.length()); // output:  13
```

charAt() method

- `charAt()` – returns a specific character, given an index.

```
String s1 = "Rumplestiltskin";

System.out.print(s1.charAt(0)); // output:  R
System.out.print(s1.charAt(5)); // output:  e
System.out.print(s1.charAt(12)); // output:  k
```

Some Conversion methods

- `toLowerCase()` – returns all lower case version of string
- `toUpperCase()` – returns all upper case version of string
- `trim()` – returns a string that eliminates leading and trailing blank characters from original
- `replace()` – returns a string with an old character replaced with a new one. old character and new character passed as parameters

Examples

```
String s1 = "Zebra

String s2 = s1.toLowerCase(); // s2 is "zebra"
String s3 = s1.toUpperCase(); // s3 is "ZEBRA"


String s4 = " Apple ";

String s5 = s4.trim(); // s5 is "Apple"
String s6 = s5.replace(‘e’, ‘y’); // s6 is "Apply"
```

valueOf() method

- `valueOf()` – there are several of these methods.
- They are **static** methods, and are used for converting other values to String objects

```
int x = 12345;

String s7 = String.valueOf(4.56); // s7 is "4.56"
String s8 = String.valueOf(16); // s8 is "16"
String s9 = String.valueOf(x); // s9 is "12345"
```

Parsing Strings into other types

```
String x = "1";
String y = "2";

System.out.println(x+y);

int i = Integer.parseInt(x);
int j = Integer.parseInt(y);

System.out.println(i+j);

String gpa = "3.98;
double gpa2 = Double.parseDouble(gpa);

System.out.println(gpa2 - 1.0);
```

The StringBuilder Class

- The `StringBuilder` class is a part of the `java.lang` package.
- A `StringBuilder` object is mutable (i.e. it can be changed).
- Three of the four constructors shown here. Here are sample creations:
  - creates an empty string builder with initial capacity of 16 characters
    `StringBuilder buf1 = new StringBuilder();`
  - creates empty string builder with initial capacity given in parameter
    `StringBuilder buf2 = new StringBuilder(50);`
  - creates string builder filled with argument – initial capacity is length of given string plus 16
    `StringBuilder buf3 = new StringBuilder("Hello");`

The append() method

- `append()` – adds data to string in the builder object, at the end. Several versions for different parameter types (see API for full set)

```
StringBuilder buf1 = new StringBuilder();

buf1.append("Hello");
buf1.append(',');
buf1.append(" world!");
     // buf1 is now "Hello, world!"

buf1.append(' ');
buf1.append(123.45);
     // buf1 is now "Hello, world!  123.45"
```

The insert() method

- `insert()` – insert data at a certain starting index. Like append, multiple versions for different types of data (see API for full set)

```
StringBuilder buf2 = new StringBuilder();

buf2.append("Welcome home");
     // buf2 now "Welcome home"

buf2.insert(8,"to my humble ");
     // buf2 = "Welcome to my humble home"
```

More StringBuilder methods

- `delete()` – delete data from a string builder object

```
StringBuilder buf3 = new
         StringBuilder("abcdefghijklm");

buf3.delete(4,9);
// deletes indices 4-8.  buf3 is now "abcdjklm"
```

- `deleteCharAt()` – delete a character at specified index

```
StringBuilder buf4 = new
         StringBuilder("abcdefg");

buf4.deleteCharAt(3); // buf4 is now "abcefg"
buf4.deleteCharAt(1); // buf4 is now "acefg"
```

More StringBuilder methods

- `reverse()` – reverses the contents of the string builder
- `setCharAt()` – sets a character at specified index (similar to deleteCharAt()
- `capacity()` – returns current capacity of builder
- `length()` – returns length of current string in builder (less than or equal to capacity)
- `setLength()` – sets the exact length of the string in the builder to new value (parameter).
  - This is the actual string, not the capacity.
  - If the new length is smaller than previous length, characters are truncated from the string.
  - If new length bigger, null characters are appended.
- charAt() – returns character at a specified index (parameter)