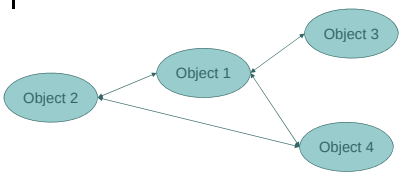
Object Oriented Programming

For : COP 3330.
Object oriented Programming (Using C++)
http://www.compgeom.com/~piyush/teach/3330

Piyush Kumar

OOP



Objects: State (fields), Behavior (member functions), Identity
Class : Blue print of an object.

Data and behavior are strongly linked in OOP.
Objects are responsible for their behavior.
Example: Complex numbers, Rational numbers, Floating point numbers , all understand addition.

OOP components

- Data Abstraction
  - Information Hiding, ADTs
- Encapsulation
- Type Extensibility
  - Operator Overloading
- Inheritance
  - Code Reuse
- Polymorphism

Recap: ADTs

- Specify the meaning of the operations *independent* of any implementation/definition.
  - Least common denominator of all possible implementations.
  - Information Hiding: Do not expose unnecessary information.

Inheritance

- Two example classes
- Class Employee

```
class Employee {
    public:
        Employee(string theName, float PayRate);
        string Name() const;
        float   PayRate() const;
        float   compute_pay(float hoursWorked) const;
    protected:
        string name;
        float payrate;
};
```

Inheritance

- Two example classes
- Class Manager

```
class Manager {
    public:
        Manager(string theName, float PayRate);
        void set_manages(int n);
        string Name() const;
        float   PayRate() const;
        float   compute_pay(float hoursWorked) const;
    protected:
        string name;
        float payrate;
        int manages_n_employees;
};
```

Reuse

- We have done unnecessary work to create Manager, which is similar to (and really is a "**is a**") Employee.
- We can fix this using the OO concept of *inheritance*.
- We let a manager inherit from an employee.
  - A manager gets all the data and functionality of an employee after inheritance.
  - We can then add any new data and methods needed for a manager and *redefine* any methods that differ for a manager.

Manager

```
class Manager : public Employee {  // is a relationship
    public:
        Manager(string theName, float PayRate, int n);
        void set_manages(int n);
    protected:
        int manages_n_employees;
};
```

Methods of Manager have access to name, payrate because they were declared in Employee as "protected" .

More on Inheritance : Access privileges.

- In a public inheritance:
  - Public members are accessible to derived class.
  - Protected members are accessible to derived class. These members are not accessible to the users of the base class.
  - Private members are not accessible to derived class.

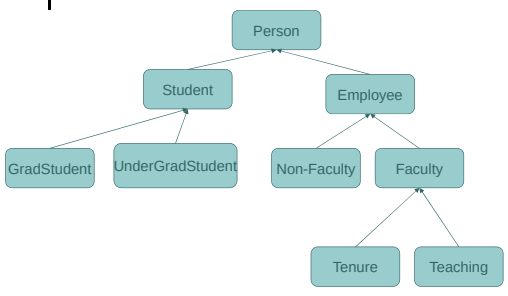
Inheritance

- Derive a new class (subclass) from an existing class (base class).
  - Syntax:
    - class classname : access-label base-class { … }
    - Access-labels = { public, private, protected }
- Inheritance creates a hierarchy of related classes (types) which share code and interface.

More Examples

Base Class	Derived Classes
Student	GradStudent UnderGradStudent
Shape	Circle Triangle Rectangle Tetrahedron
Loan	CarLoan HomeImprovementLoan MortgageLoan

More Examples



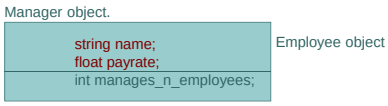
"Is a" relationships.

Inheritance: Subclass

- Code reuse
  - derive *GradStudent* from *Student*
    (also adds fields/methods)
- Specialization: Customization
  - derive *bounded-stack* from *stack*
    (by overriding/redefining *push*)
- Generalization: Factoring Commonality
  - Avoid code-duplications (why?)

Inheritance

- Derived classes contain their base classes as subobjects.



- Functions in the derived may use members from the base.

There is no requirement that the compiler lay out the base and derived parts of an object contiguously.

Inheritance

- A class must be *defined* to be used as a base class.
- A derived class can be used as a base-class.
- Forward declarations are same for base-classes as well as derived classes.
  - class Manager;
  - class Employee;
  - class Manager: public employee; // Error

Open-Closed principle in OOP

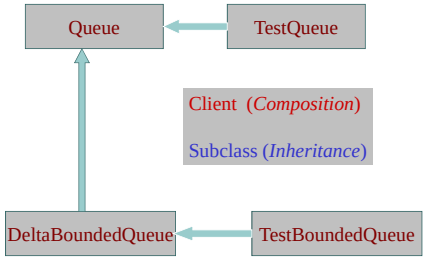
- The **open/closed principle** states that a class must be both open and closed.
  - Open: means it has the ability to be extended
  - Closed: means it cannot be modified other than by extension.

An interesting paper:
http://www.craiglarman.com/articles/The%20Importance%20of%20Being%20Closed%20-%20Larman%20-%20IEEE%20Software.pdf

Open-Closed principle in OOP

- Once a class has been approved for use after having gone through code reviews, unit tests, and other qualifying procedures, you don't want to change the class very much, just extend it.
- Changing base class can complicate all derived classes.

Example : Open-Closed Pr.



More on Inheritance

- A pointer to a derived class can always be used as a pointer to a base class when public inheritance is used. (But not vice-versa)
  - Private base classes are different
- STL Containers which need to contain both base/derived classes should be made of pointers to base classes.
  - Otherwise : Slicing problem.

Virtual Methods

- A base class must indicate which of its member functions it intends its derived classes to redefine.
- These member functions are defined as “virtual” in the base class.

Example

```
Base *bp;
Derived d;
bp = &d;
bp->print();  // invokes
```

```
class Base {
public:
  int i;
  virtual void print()
  {
            cout << "i value is " << i
                 << " inside object of type Base\n\n";
  }
};

class Derived: public Base {
public:
  virtual void print()
  {
            cout << "i value is " << i
                 << " inside object of type Derived\n\n";
  }
};
```

Dynamic Binding

- Allows invocation of general methods using a base class pointer.
- The fact that a reference or pointer might refer to either a base or derived-class object is the key to dynamic binding.
- Allows easy extensibility.

Dynamic Vs Static Binding

- Static Binding: The compiler uses the type of the pointer to find out which method to call.
- Dynamic Binding: The decision is made at runtime. (uses ‘virtual’ keyword)

Dynamic Vs Static Binding

- Static Binding
  - Less time and space overhead.
  - Inlining possible
- Dynamic Binding
  - Extensibility
  - Better code-reuse.

Dynamic Vs Static Binding

- Efficiency Vs Flexibility
- Static Binding
  - More efficient
    - Less time and space overhead, can use inlining.
- Dynamic Binding
  - Flexible: Enables extension of behavior of a system easily.

Virtual Functions

- Have a fixed interface.
- Derived implementations can change.
- Dispatched using object's "dynamic type" to select the appropriate method.
- "Once Virtual, always virtual" rule.
  - Once a base-class defines a function as virtual, it remains virtual through out the inheritance hierarchy.

An example base class

```
// Item sold at an undiscounted price
// derived classes will define various discount strategies
class Item_base {
        friend std::istream& operator>>(std::istream&, Item_base&);
        friend std::ostream& operator<<(std::ostream&, const Item_base&);
        public:
           virtual Item_base* clone() const { return new Item_base(*this); }
        public:
           Item_base(const std::string &book = "", double sales_price = 0.0): isbn(book), price(sales_price) { }

           std::string book() const { return isbn; }

           // returns total sales price for a specified number of items
           // derived classes will override and apply different discount algorithms
           virtual double net_price(std::size_t n) const { return n * price; }

           // no work, but virtual destructor needed
           // if base pointer that points to a derived object is ever deleted
           virtual ~Item_base() { }   // Always virtual. Why? (Hint: Static Vs Dynamic Binding)
        private:
           std::string isbn;    // identifier for the item
        protected:
           double price;       // normal, undiscounted price
};
```

Scoping rules

- In a public base class, public and protected members of the base class remain public and protected members of the derived class.
  - Example:

```
class circle: public point {};
circle c;
//can call  point::move(int,int)
c.move(1,2);
```

Scoping Rules.

Out of scope of this class. Do not use.

- Private derivation:
  - public base class members are private in derived class.
  - Example:

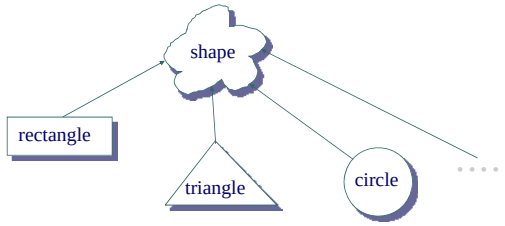
```
class stack: private linkedList {};
stack s;
s.insert(1,2);
// cannot call linkedList::insert(int,int)
   where insert is public
```

- Protected derivation:
  - public base class members are protected in derived class.

"Is a" Vs "Has a"

- Inheritance
  - Considered an "Is a" class relationship
  - e.g.: An HourlyEmployee "is a" Employee
  - A Convertible "is a" Automobile
- A class contains objects of another class as it's member data
  - Considered a "Has a" class relationship
  - e.g.: One class "has a" object of another class as it's data

OOP Shape Example



Public inheritance: “Is A” relationships

Abstract Base class: Shape.

```
class Shape {
    public:
        Shape ( Point2d& position, Color& c) : center_(position) , color_ (c) {};
        virtual void rotate( double radians ) = 0;
        virtual bool draw(Screen &) = 0;
        virtual ~Shape(void) = 0;
        void  move(Point2d& p) { _center = p; };
    private:
        Point2d center_;
        Color color_;
};
```

C++ Shape example

```
class Triangle: public Shape {
    public:
        Triangle( Point2d& p[3] );
        virtual void rotate ( double radians ){…}
        virtual bool draw(Screen &s) {…};
        virtual ~Triangle(void) {…};

    private:
        Point2d vertices[3];
        Color color_;
};
```