Lecture 2.1

Inheritance

References:

- J. Hershberger, “Java Programming: CS491”, 2004
- Angela Chang, http://www.cs.auckland.ac.nz, 2007

Inheritance

- Inheritance is another representation of an OO relationship between two classes. In an inheritance relationship, an object (called a subclass) obtains data and behavior from another object (called the base class or super class)
- Inheritance is implemented in java using the *extends* keyword
- *super* keyword is used to refer base class

Derived class in inheritance

- Derived class (subclass) in an inheritance relationship:
  - Inherits all the public variables and methods of a base class
  - Adds additional variables and methods
  - Can change the meaning of inherited methods (override methods)

Use of inheritance

- Allows for **code reusability**. This is accomplished by allowing the subclasses (or derived classes) to directly use methods that have been implemented by their base class.
- Java does not support multiple inheritance
- In UML diagram inheritance is shown as a clear triangle.

Inheritance in Java

If class A inherits from class B, it can be depicted with java program as:

class A extends B{

}

Example

```
public abstract class Animal // class is abstract {
    private String name;
    public Animal(String nm) { name=nm; }
    public String getName() { // regular method
        return (name); }
}

public class Dog extends Animal {
    public Dog(String nm) { // builds ala parent
        super(nm);}
    public void work() // this method specific to Dog {
        System.out.println("I can herd sheep and cows"); }
}
```

Multiple Inheritance

- Some programming languages like C++ allow multiple inheritance where a derived class inherits from more then one base class in the same level of inheritance hierarchy.

Access level

- The access level determines to what extent an object will allow another object to access its members (sometimes this is called object visibility).
- Access levels are used to:
  - enforce encapsulation of an object's data by hiding the attributes
  - restrict access to only certain methods
  - restrict access depending upon what packages the object resides in

Types of Access level

- private ('-' notation in UML)
  - Only the class itself has access to its private members
  - Think: "only you know the secret"
- package ('~' notation in UML)
  - Only the class itself, and any other classes in the same package have access to the class's members. This is the default access level if no access level is specified
  - Think: "everyone in the same group knows the secret"
- protected ('#' notation in UML)
  - Only the class itself, any other classes in the same package , and any subclasses have access to the class's members
  - Think: "you know, everyone in the same group knows, and your children know the secret"
  - Note: Protected access level is only concerned with subclasses.
- public ('+' notation in UML)
  - All other classes have access to the class's members
  - Think: "everyone knows the secret"

Method Overloading

- Method overloading is the process of using the same method name for multiple methods
- The signature of each overloaded method must be unique
  - The signature includes the number, type, and order of the parameters
  - The return type of the method is NOT part of the signature
  - The compiler must be able to determine which version of the method is being invoked by analysing the parameters

Example

```
class Circle {

//declaring the instance variable
protected double radius;
                public Circle(double radius) {
                this.radius = radius;
}

// This method can be overidden
public double getArea() {
                return Math.PI*radius*radius;
}//this method returns the area of the circle

} //end of Circle class
```

```
class Cylinder extends Circle {

        //declaring the instance variable
        protected double length;

        public Cylinder(double radius, double length) {
                super(radius);
                this.length = length;
        }

        // This method has been overidden
        public double getArea() { // method overriden here
                return 2*super.getArea()+2*Math.PI*radius*length;
        }//this method returns the cylinder surface area

}// end of class Cylinder
```

Method Overriding

- A derived class can use the methods of its base class(es), or it can override them
- The method in the derived class must have exactly the **same signature** as the base class method to override.
- The signature is number and type of arguments and the constantness (const, non- const) of the method.
- The **return type** must match the base class method to override.

Using Overriding

- If an inherited property or method needs to behave differently in the derived class it can be overridden; that is, you can define a new implementation of the method in the derived class.
- You can change the meaning (override) of the method declared in the superclass
  - Completely, or
  - Add more functionality to the method
    - The new method can call the original method in the parent class by specifying Super before the method name.
- Rules:
  - A Subclass cannot override final methods declared in the base class.
  - The Overridden method must have the same arguments as the inherited method from the base class.

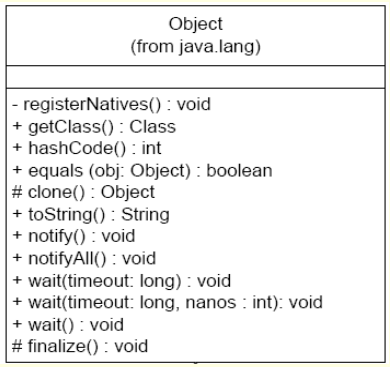
Example

```
public void aMethod( String y){
    //…
}

public void aMethod( int x){ //overloading
    //…
}

public void aMethod( int x, String y ){ //overloading
    //…
}

public int aMethod(float y){ //Not overloading
    //…
}
```

Inheriting from Object class

- In Java, all classes (except primitive data types) inherit from a base class. Each class can have exactly one base class.
- If a class does not explicitly inherit from another class, that class will implicitly inherit from the default base class (which is the *Object* class).
- Class *Object* is the root (i.e. top) of the Java class hierarchy. Every class has *Object* as a superclass (if no other superclass has been given).

UML diagram of Object class

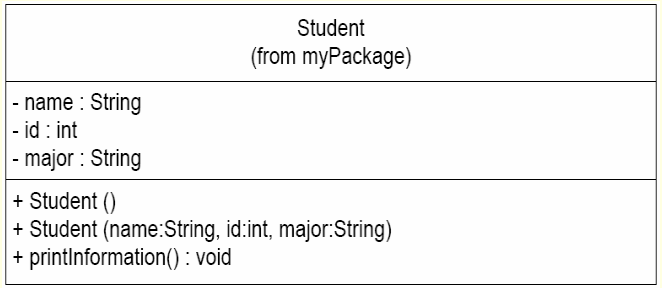


Inheritance with Object class

- The Object class members are not explicitly listed when listing all the members of a Java class. However, if a Java class includes a method which has the same signature of the Object class, you must realize that you are overriding that Object class's member.

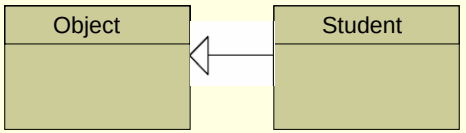
Example

- Student class is declared with some members.



Example cont..

- *Student* class has not explicitly defined any base class but *Object* class will automatically become base class for student.



Example cont..

- if you have the following lines of code:

```
Student S1 = new Student("s1",1,"NOMAJOR");
System.out.println(S1.toString());
// System.out.println(S1);
```

toString() method is a member of Object class which returns a string representation of the object. It can be overloaded to print information about Student.

```
Public String toString(){
    return name + String.valueOf(id) + major;
}
```

Usage of super

- Constructor : super() or super(…)
  - Automatically called in derived constructor if not explicitly called
  - Cannot call super.super()
- super.member
  - Members can be either method or instance variables
  - Refers to the members of the superclass of the subclass in which it is used
  - Used from anywhere within a method of the subclass

Usage of this

- Can be used inside any method to refer to the current object
- Constructor: this(), this(…): refer to its constructor
- this.member
  - Members can be either method or instance variables
  - this.instance_variable:
  - To resolve name-space collisions that might occur between instance variables and local variables