OOJ Lecture 2
Inheritance

- Reading: Savitch, Chapter 7
- Reference: Big Java, Horstman, Chapter 11

Objectives

- To further explore inheritance and the overriding of superclass methods
- To learn to invoke superclass constructors -- super

Method overriding & Inheritance

- Override method:
  - Supply a different implementation of a method that exists in the superclass
- Inherit method:
  - Don't supply a new implementation of a method that exists in the superclass
- Add method:
  - Supply a new method that doesn't exist in the superclass

Inheritance and Fields

- Inherit field:
  - All non-private fields from the superclass are automatically inherited
- Add field:
  - Supply a new field that doesn't exist in the superclass
- Override (shadowing) fields
  - The subclass field shadows the superclass field
  - superclass's field is present, but the subclass does not have direct access to the field

Overriding Versus Overloading

- Overriding
  - Same method name
  - Same signature
  - One method in ancestor, one in descendant
- Overloading
  - Same method name
  - Different signature
  - Both methods can be in same class

The `final` Modifier

- "final" indicates that a method definition cannot be overridden with a new definition in a derived class
- Example:

```
public final void specialMethod()
{
...
}
```

- Used in specification of some methods in standard libraries
- Allows the compiler to generate more efficient code
- Can also declare an entire class to be final, which means it cannot be used as a base class to derive another class

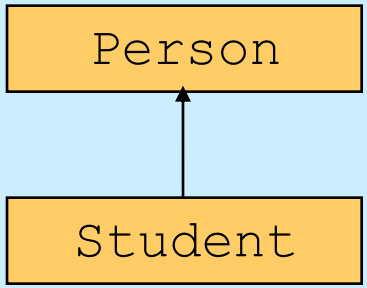
Adding Constructor in a Derived Class

```
//Reference: Savitch, Chapter 7.1
public class Person
{    private String name;
     public Person()
     {  name = "No name yet.";
     }
     public Person(String initialName)
     {  name = initialName;
     }
     public void setName(String newName)
     {
        name = newName;
     }
     public String getName()
     {
        return name;
     }
   ...
}
```

Adding Constructor in a Derived Class

```
public class Student extends Person
{ private int studentNumber;

  public Student()
  {
      super();
      studentNumber = 0; //Set to 0 temporarily
  }

  public Student(String name, int num)
  {   super(name);
      studentNumber = num;
  }

  public void setDetails(String newName, int newNum)
  { … }

  public int getStudentNumber()
  { … }

  public void setStudentNumber(int newNum)
  { … }
}
```

Adding Constructor in a Derived Class



`Person` is the parent class of `Student` in this example.

Calling a Superclass Constructor

- Invoking a constructor of the superclass use the *super()* method
- Syntax

```
subClassName(parameters)
{
    super(parameters);
    . . .
}
```

- Must be the *first* statement in subclass constructor

Super() in Subclass Constructors

- Constructors may be overloaded.
- If a subclass does not have a constructor, Java uses a default constructor. This looks like:

```
public class A extends B
{
    public A()
    {
        super();
    }
}
```

- Also for the superclass, if there is no constructor at all, Java provides one for you. (Remember, constructors are like methods and if desired, can be overloaded many times)
- If there is a class constructor, Java will not create a default no-arg constructor.

Example: Constructor in Base Class

```
public class Rabbit
{  private int Age;
   private char Sex;
   private int  Speed;
   public Rabbit(int Age, char Sex,int Speed)//constructor
   {  this.Age = Age;
      this.Sex = Sex;
      this.Speed = Speed;
   }
    void  run(int duration, boolean zigzag)
    { char  C = (zigzag? 'Y':'N');
      System.out.println("I am " +Age+ "years old.");
      System.out.println("I hate to run, but will run
              for " + duration + "minutes. Zigzag fashion?"+C);
    }
    void  sleep(int duration)
    { System.out.println("I am a " + Sex + " rabbit");
      System.out.println("I now sleep for"+ duration+"minutes");
    }
}
```

Example: Constructor in Derived class

```
public class JadeRabbit extends Rabbit {
{
   private String HomeAddress;          // String type later
   public JadeRabbit()
   {
        super(2000,'F',1000); //call to superclass constructor
        HomeAddress = "The Moon";
   }
 }
```

- Here, the subclass constructor calls the superclass constructor first.
- Initializes the fields that were inherited from the superclass (Rabbit), then it initializes the HomeAddress field defined by itself.

Using *this* reference

- *'this'* reference is required if you use the same name of variables or method
  - an object often needs to know its own name. "*this*" is the current object`s own name.
- Example

```
// This method is a constructor
public Rabbit(int Age, char Sex,int Speed)
{
   this.Age = Age;
   this.Sex = Sex;
   this.Speed = Speed;
}
```

If we do not use *this* here, then we have:

Age = Age! Parameter Age is assigned to itself!

Constructors in a Derived Class (More)

Constructors can call other constructors

- Use `super` to invoke a constructor in parent class
  - as shown on slide 16 (next slide)
- Use `this` to invoke a constructor within the class
  - shown on slide 17
- Whichever is used must be the first action taken by the constructor
- Only one of them can be first, so if you want to invoke both:
  - Use a call with `this` to call a constructor with `super`

Example of a constructor using `super`

- `Student` class has a constructor with two parameters:
  - `String` for the `name` attribute (which is in the Person class)
  - `int` for the `studentNumber` attribute

```
public Student(String name, int num)
{    super(name);
     studentNumber = num;
}
```

Example of a constructor using `this`

- You may define another constructor for **`Student`**.

  Example:

```
public Student(String initialName)
{
    this(initialName, 0);
}
```

Call the constructor of self class defined in previous slide

- This example takes a **`String`** argument. It then initializes the **`studentNumber`** attribute to a value of 0.
- It calls another constructor of **`Student`** with two arguments: initialName (**`String`**) and 0 (**`int`**)
- We are overloading within the same class!

Calling a Superclass Method

Syntax: super.*methodName*(*parameters*)

Purpose:

To call a method of the superclass instead of the overridden method/variable of the current class

- The "super" refers to the superclass' overridden method or a hidden variable.

Using an overridden method in a subclass

- Using the previous example:

```
class Rabbit extends Animals
{   void wish()
    { super.wish(); //call wish from Animal class
      System.out.println("and Disneyland.");
    }
}
```

- The following code:

```
Rabbit bugs = new Rabbit();
bugs.speech();
```

- Outputs:

```
Thank you, I want to go home and
Disneyland.
```

Examples: using super to refer to the hidden superclass's variables

```
class Look
{ int i = j = 1; // would this work if declared private?
}
class Taitai extends Look
{ int j = 2;   // j is hidden since i= j=1 in Look.
  void PrintIt() {
     System.out.println(i + " "+ j +" "+ super.j);
  }
}
```

- The following code:

```
Taitai look = new Taitai();
look.PrintIt();
```

- Outputs: 1 2 1

In this class, super.j refers to the original j value in Look class

Summary

- A subclass inherits the instance variables & methods of the superclass automatically.
- A subclass can create additional instance variables and methods.
- The first thing a constructor in a subclass normally does is call a constructor in the superclass.
- If a subclass redefines a method defined in the superclass, the version in the subclass *overrides* that of the superclass.

Class Exercise

1. Implement a `RectangleBase` class using the follow design :
2. Extend `Rectangle3D` as a subclass of `RectangleBase`, as the name suggests, the class models a three dimensional rectangle.

- Use inheritance as much as possible!! You will have no marks if you do not use inheritance in the exam!