Object Oriented Programming: Inheritance

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

Objectives

- To understand object oriented techniques
- To lean how to inherit and override superclass methods
- To understand Overriding versus Overloading

Object Oriented Programming: Key Concepts

- Objects and Attributes
- Classes and Instances
- Abstraction
- Encapsulation
- Inheritance
- Association
- Communication with messages
- Polymorphism

Why OOP?

- To try to deal with the complexity of programs
- To apply principles of abstraction to simplify the tasks of writing, testing, maintaining and understanding complex programs
- To increase code reuse
  - to reuse classes developed for one application in other applications, instead of writing new programs from scratch .
- Inheritance is a major technique for realising these objectives

Inheritance

One of the fantastic things about object-orientation is the concept of *inheritance*.

- Inheritance defines relationships between classes so that classes may share the structure and the behaviour of other classes - super/sub classes
- Generally, a subclass would augment the existing structure and behaviour of its superclass.

Inheritance

- The general class is called the *base* or *parent class*
- The specialised classes *inherit* all the properties of the general class
  - specialized classes are *derived* from the base class
  - they are called *derived* or *child* classes
- After the general class is developed you only have to write the "difference" or "specialisation" code for each derived class

Inheritance

- A *class hierarchy:* classes can be derived from derived classes (child classes can be parent classes)
  - any class higher in the hierarchy is an *ancestor class –called superclass*
  - any class lower in the hierarchy is a *descendent class – called subclass*

Subclass and Inheritance

- Java allows only single inheritance. That is, a subclass can inherit from only one superclass.
- The *ancestor* class is called Object in Java.
  - Every class is a descendent of Object. If your class does not extend another class, then it is automatically a subclass of the Object class.
  - (Classes automatically inherit many methods of the Object class)
  - Java classes form a directed tree, root=Object class.

```
java.lang.Object
  |
  +-- java.lang.Math
java.lang.Object
        |
 /…\   /  \
………………
```

Subclass and Inheritance

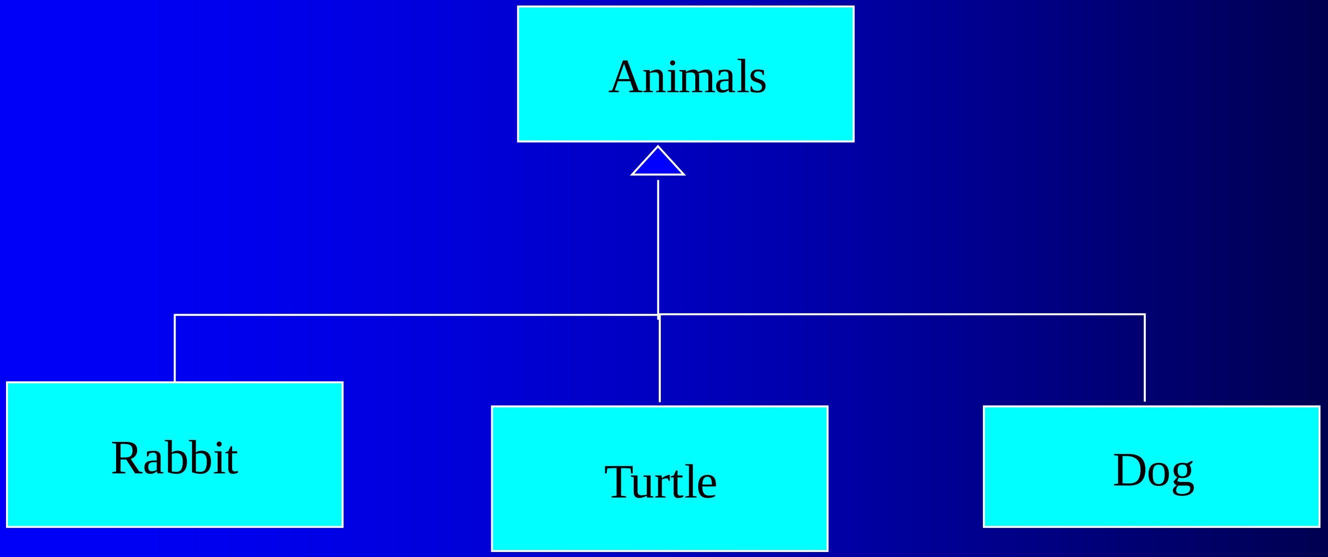
- C++ allows multiple inheritance.
- The famous diamond shape inheritance:

```
     Supplier
        / \
Customer   BusinessPartner
        \ /
    private person
```

This causes problems.

Subclass and Inheritance

- Java is a tree shape inheritance, any class has only one parent.



Subclass and Inheritance

- To define a subclass, use the `extends` keyword.

  Syntax: Inheritance:

```
class SubclassName extends superclass
       [implements interface name]
{
     [Declaration of variables]
     [Declaration of methods]
}
```

Inheritance

Example:

-
  ```
  class Rabbit extends Animals{
        new methods
        new instance fields
  }
  ```
- All methods of **`Animals`** are automatically inherited
  - `Animals` is a (parent/base) *super class*
  - `Rabbit` is a (child/derived) *subclass*

Subclass and Inheritance

```
class Animals {
   int useless = 0;
   void wish() {
       System.out.println("I want to go home”);
   }
   void speech() {
       System.out.print("Thank you.”);
       wish();     //call wish()
   }
}

class Rabbit extends Animals {
   void wish() {
       System.out.println("I want a carrot”);
       //override wish()
   }
}

class Turtle extends Animals {
   void wish() {
       System.out.println("I want a shrimp”);
       //override wish()
   }
}
```

Example - Cont'd

```
class AnimalWishes
{
   public static void main(String args[])
   {   Animals[] animals = new Animals[3];
        animals[0] = new Turtle();
        animals[1] = new Rabbit();
        animals[2] = new Turtle();
        for (int i = 0; i < animals.length;i++)
        {       animals[i].speech();               }
   }
}
```

What is the output?

```
> javac  Animals.java  AnimalWishes.java
> java  AnimalWishes
Thank  you.I  want  a shrimp
Thank  you.I  want  a carrot
Thank  you.I  want  a shrimp
```

Method Overriding & Inheritance

- *Rabbit* and *Turtle* are subclasses of Animals.
- Each subclass inherits variable *useless* and 2 methods *speech()* and *wish()*.
- Note: normally variables are private! We use non-private only for teaching points in these examples.

- The code in the previous example:

```
    animals[2].speech();   //a turtle
```

- Will generate output:

```
    Thank you. I want a shrimp
```

- Why?

Method Overriding & Inheritance

- Here the speech() method is **inherited.**

  `animals[2].speech();` //output: Thank you

- When speech() calls wish()
  - The wish() method of Animals is *overridden by* the subclasses Turtle and Rabbit.
  - The Turtle.wish() is called. The output is: I want a shrimp
- When there is a method with the same name in the subclass, the method overrides the superclass.
- Otherwise the non-private superclass methods are inherited without change. This is called inheritance.

Instance Variable Hiding

Example

```
class Look
{   int i = j = 1;
}
class HideJ extends Look
{ int j = 2;       // Hide j=1
  void PrintIt()
  { System.out.println(i + " " + j); }
}
```

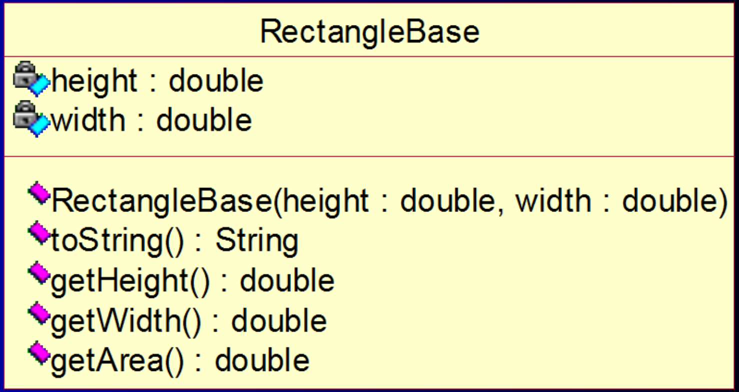
j=1 is hidden. The following code:

```
  HideJ SeeSee = new HideJ();
  SeeSee.PrintIt();
will output:  1  2
```

● A method is called overridden, a variable is called hidden.

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.



- What additional attributes and methods will the `Rectangle3D` class need?
- What methods has the `Rectangle3D` class inherited?