Inheritance

Chapter 7
Reminders

- Project 4 was due last night
- Project 5 released: due Oct 20 @ 10:30 pm
- Project 2 regrades due by midnight tonight
- Project 3 grades posted on WebCT – regrade requests due by Friday Oct 14 midnight (send to wsumner@cs.purdue.edu)
Next Week

- October Break on Monday/Tuesday
- No Lecture on Monday (There is lecture on Wednesday)
- No Labs at all next week
- Recitations will meet Friday as usual
Discussion Groups

• Due to popular demand there will be two discussion groups each week
  • Monday 7-9pm in CS G066
  • Wednesday 7-9pm in Phys 11

• For more information, visit http://www.cs.purdue.edu/homes/bwittman/discussion.html or e-mail me at bwittman@gmail.com.
Introduction to Inheritance

• *Inheritance* allows us to define a general class and then define more specialized classes simply by adding new details to the more general class definition.

• A more specialized class *inherits* the properties of the more general class, so that only new features need to be programmed.
Introduction to Inheritance, cont.

• example

  – General class `Vehicle` might have instance variables for weight and maximum occupancy.
  
  – More specialized class `Automobile` might add instance variables for wheels, engine size, and license plate number.

  – General class `Vehicle` might also be used to define more specialized classes `Boat` and `Airplane`.
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;
    }
    public void writeOutput() {
        System.out.println("Name: " + name);
    }
    public boolean sameName(Person otherPerson) {
        return (this.name.equalsIgnoreCase(otherPerson.name));
    }
}
Derived Classes, cont.

Display 7.2
A Class Hierarchy
Derived Classes, cont.

- `public class Student extends Person`

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

    public Student() {
        super();  // Indicating no number yet
        studentNumber = 0;
    }

    public Student(String initialName, int initialStudentNumber) {
        super(initialName);
        studentNumber = initialStudentNumber;
    }

    public void setStudentNumber(int newStudentNumber) {
        studentNumber = newStudentNumber;
    }

    public void writeOutput() {
        System.out.println("Name: " + getName());
        System.out.println("Student Number: " + studentNumber);
    }

    public boolean equals(Student otherStudent) {
        return (this.lastName.equals(otherStudent.lastName) &&
                this.firstName.equals(otherStudent.firstName));
    }

    public int getStudentNumber() {
        return studentNumber;
    }

    public void reset(String newName, int newStudentNumber) {
        setName(newName);
        studentNumber = newStudentNumber;
    }
}
```

Display 7.3
A Derived Class
Derived Classes, cont.

• When you define a derived class, you declare only the added instance variables and you define only the added and overridden methods.

• The variables and methods of the parent class which are not declared private are inherited automatically.
Derived Classes, cont.

- class InheritanceDemo

```java
public class InheritanceDemo {
    public static void main(String[] args) {
        Student s = new Student();
        s.setName("Warren Peace");
        s.setStudentNumber(1234);
        s.writeOutput();
    }
}
```

Display 7.4
Demonstrating Inheritance

setName is inherited from the class Person.
Constructors in Derived Classes

• A base class has its own constructors.
  – Their purpose typically is to initialize the instance variables declared in the base class.

• A derived class has its own constructors.
  – Their purpose typically is to call a constructor in the base class, and then to initialize the instance variables declared in the derived class.
Constructors in Derived Classes, cont.

• To call a constructor in the base class, use

  super(Values_for_Instance_Variables
          _Declared_in_the_Base_Class);

• example

  super(initialName);

  not

  Person(initialName);  //ILLEGAL
Using `super`

- The call to the constructor in the base class (using `super`) must be the first action taken in the constructor of a derived class.
- When no call to the constructor in the base class is included, Java automatically includes a call to the default constructor in the base class.
Using `super`, cont.

- equivalent definitions:

  ```java
  public Student()
  {
      super();
      studentNumber= 0;
  }
  ```

  and

  ```java
  public Student()
  {
      studentNumber= 0;
  }
  ```
The **this** Method

- Within the definition of one constructor, it can be appropriate to call another constructor in the same class.
- The keyword **this** is used to call another constructor in the same class.
- **example**
  ```
  this(initialName, 0)
  ```
The *this* Method, cont.

- Any use of *this* must be the first action in the constructor definition.
  - Thus, a constructor definition cannot contain a call using `super` and a call using `this`.

- To use both `super` and `this`, include a call using `this` in one constructor and a call using `super` in the constructor called using `this`. 
Calling an Overridden Method

• `super` can be used to call a method in the base class that has been overridden in the derived class.

• example

  `super.writeOutput();`

• However, you cannot repeat the use of `super` to invoke a method in some ancestor class other than the immediate base (parent) class.
Dynamic Binding

• Different objects can invoke different method definitions using the same method name.

• For example, if $b$ references a `Box` and $t$ references a `Triangle`, $b$ and $t$ invoke different definitions of method `drawAt` even if $b$ and $t$ are declared to be objects of type `Figure`. 
Dynamic Binding, cont.

- Handling the invocation of a method that may be overridden later is called *dynamic binding* or *late binding*.

- The type of object being referenced at the time of the method call, not the type of reference that was declared, determines which method is invoked.
Dynamic Binding, cont.

• Consider

```java
Figure f;
Box b = new Box(1, 4, 4);
f = b;
f.drawAt(2);
Triangle t = new Triangle(1,2);
f = t;
f.drawAt(2);
```
Dynamic Binding, cont.

- Method `drawAt` is inherited from class `Figure` and is not overridden.
- But, method `drawHere` is invoked within the definition of method `drawAt`, and method `drawHere` is overridden.
- The type of object referred to by `f` determines which method `drawHere` is invoked.
Remember:

Display 7.2
A Class Hierarchy
Example

```java
Person[] x = new Person[3];
x[0] = new Student();
x[1] = new Graduate();
x[2] = new Staff();
```

Are these legal statements?
An Object Can Have More than One Type

- If class Undergraduate is derived from class Student and class Student is derived from class Person, then every object of class Undergraduate is also an object of class Student and an object of class Person.

- A reference to an object of class Undergraduate can be substituted for a reference to an object of class Student or a reference to an object of class Person.
Remember: “is a” Relationship

- $x[0] = \text{new Student}();$
  - a Student is a Person
- $x[1] = \text{new Graduate}();$
  - a Graduate is a Person
- $x[2] = \text{new Staff}();$
  - a Staff is a Person
More Examples

Graduate[] x = new Graduate[3];

x[0] = new Student();
x[1] = new Employee();
x[2] = new MastersDegree();

Which of these are legal?
More Examples

- Graduate[] x = new Graduate[3]

- x[0] = new Student();
  - **Illegal**: a Student is not a Graduate (see hierarchy)

- x[1] = new Employee();
  - **Illegal**: an Employee is not a Graduate

- x[2] = new MastersDegree();
  - **Legal**: a MastersDegree is a Graduate
Dynamic Binding with `toString`

- Recall method `toString` typically is used to prepare and return a string, describing an object, for output to the screen.

- The name of this method can be omitted, thanks to dynamic binding, because one definition of method `println` expects a single argument of type `Object` which it uses to invoke the method `toString` associated with the object.
Polymorphism

- *Polymorphism* comes from Greek meaning “many forms.”
- In Java, polymorphism refers to the dynamic binding mechanism that determines which method definition will be used when a method name has been overridden.
- Thus, polymorphism refers to dynamic binding.
Subtle Difference

• Dynamic binding refers to the process carried out by the computer.
• Polymorphism can be thought of as something objects do.
• Polymorphism, encapsulation, and inheritance, and considered to be the main features of object-oriented programming.
The Class *JApplet*

- An applet is a derived class from the class *JApplet*

```
public class LabelDemo extends JApplet
```

- **Class** *JApplet* has methods named *init* and *paint*.

- When you define methods *init* or *paint* in an applet, you are overriding inherited methods.
The Class `JApplet`, cont.

- Methods `init` and/or `paint` can be invoked without having to (re)define them.
- Thanks to polymorphism, methods `init` and/or `paint` defined in an applet will be invoked by library class methods and other methods when you run the applet with the applet (of type `JApplet`) as the parameter to the method.
The Class \textit{JApplet}, cont.

- The class \textit{JApplet} is used as a base class to derive applets to run from a webpage.
The Class **JFrame**

- A *Graphical User Interface* or *GUI* is simply a windowing interface for some kind of program.
- The class `JFrame`, rather than the class `JApplet`, is used to produce GUIs that can run as regular Java applications.
The Class `JFrame`, cont.

- class ButtonDemo

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

/**
 * Simple demonstration of putting buttons in a JFrame.
 *
 * To compile this class, you should have the class WindowDestroyer in the same directory (folder) as this class. The class WindowDestroyer is discussed a bit later in this Graphics Supplement.
 */
public class ButtonDemo extends JFrame implements ActionListener
{
    public static final int WIDTH = 400;
    public static final int HEIGHT = 300;

    public ButtonDemo()
    {
        setSize(WIDTH, HEIGHT);
        WindowDestroyer listener = new WindowDestroyer();
        addWindowListener(listener);

        Container contentPane = getContentPane();
        contentPane.setBackground(Color.WHITE);
        contentPane.setLayout(new FlowLayout());

        JButton sunnyButton = new JButton("Sunny");
        sunnyButton.addActionListener(this);
        contentPane.add(sunnyButton);
        JButton cloudyButton = new JButton("Cloudy");
        cloudyButton.addActionListener(this);
        contentPane.add(cloudyButton);

    }

    public void actionPerformed(ActionEvent e)
    {
        String actionCommand = e.getActionCommand();
        Container contentPane = getContentPane();

        if (actionCommand.equals("Sunny")){
            contentPane.setBackground(Color.BLUE);
        } else if (actionCommand.equals("Cloudy")){
            contentPane.setBackground(Color.GRAY);
        } else{
            System.out.println("Error in button interface.");
        }
    }
}
```

Display 7.18
A Window Interface Derived from JFrame
The Class `JFrame`, cont

class ShowButtonDemo

    public class ShowButtonDemo
    {
        public static void main(String[] args)
        {
            ButtonDemo gui = new ButtonDemo();
            gui.setVisible(true);
        }
    }

Display 7.19

Running a JFrame Class from an Application
The Class `JFrame`, cont.

Resulting GUI

Display 7.19
Running a JFrame Class from an Application

This GUI changes color in reaction to button clicks in the same way as the applet in Display 5.26.
The Class \texttt{JFrame}, cont.

• The class \texttt{JFrame}, and every class derived from it, has a method named \texttt{setVisible}.
  – When its argument has the value \texttt{true}, the GUI is visible.

• Labels, buttons, \texttt{JFrames}, and other components inherit method \texttt{setVisible} from a common ancestor.
The Class JFrame, cont.

• A class derived from JFrame has no init method, but it does use a constructor.

• Code that would be in the init method of an applet is placed in the constructor of the class derived from JFrame.

• A GUI derived from a JFrame sets an initial size

  setSize(WIDTH, HEIGHT);
Window Events and Window Listeners

• With a JFrame, a GUI’s close-window button needs to be programmed.
• A close-window button generates a window event which is handled by a window listener.
• Class WindowAdapter is a window listener, so every class derived from class WindowAdapter is a window listener.
Window Events and Window Listeners, cont.

- A window listener is registered with a JFrame GUI using method `addWindowListener`.
Window Events and Window Listeners, cont.

- class WindowDestroyer

```java
import java.awt.*;
import java.awt.event.*;

/**
   If you register an object of this class as a listener to any
   object of the class JFrame, then if the user clicks the
   close-window button in the JFrame, the object of this class
   will end the program and close the JFrame.
*/
public class WindowDestroyer extends WindowAdapter
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
}
```

Display 7.20
A Listener Class for Window Events from JFrame GUIs
Window Events and Window Listeners, cont.

- A WindowDestroyer object is created and registered as a window listener for our JFrame GUI using
  
  ```java
  WindowDestroyer listener =
  new WindowDestroyer();
  addWindowListener(listener);
  ```

- When the window-close button is clicked, the JFrame GUI ends.
The ActionListener Interface

- The ActionListener interface has only one method heading that must be implemented.
  
  ```java
  public void actionPerformed(ActionEvent e)
  ```

- A listener that responds to button clicks in an applet or in a JFrame must implement the ActionListener interface.
Programming Example: Smiley Face as a JFrame

• Class JFrame and every class derived from class JFrame has a paint method.
• The paint method can be redefined to draw a figure.
Programming Example: Smiley Face as a JFrame, cont.

• class HappyFace

```java
import javax.swing.*;
import java.awt.*;

public class HappyFace extends JFrame {
    public static final int WIDTH = 400;
    public static final int HEIGHT = 300;

    public HappyFace() {
        setSize(WIDTH, HEIGHT);
        WindowDestroyer listener = new WindowDestroyer();
        addWindowListener(listener);
    }

    public void paint(Graphics canvas) {
        canvas.drawOval(100, 70, 200, 200);
        canvas.fillOval(155, 120, 10, 20);
        canvas.fillOval(230, 120, 10, 20);
        canvas.drawArc(150, 195, 100, 50, 180, 180);
    }
}
```

Display 7.21
Drawing in a JFrame
Programming Example:

Smiley Face as a `JFrame`, cont.

- `class ShowHappyFace`

```java
public class ShowHappyFace {
    public static void main(String[] args) {
        HappyFace gui = new HappyFace();
        gui.setVisible(true);
    }
}
```

Display 7.22
Running the JFrame with a Drawing
Programming Example: Smiley Face as a JFrame, cont.

Resulting GUI

Display 7.22
Running the JFrame with a Drawing
Summary

• You have become acquainted with inheritance.
• You have learned how to define and use derived classes.
• You have learned about dynamic binding and polymorphism.
• (optional) You have learned about the class JFrame used to produce windowing interfaces.