

# JAVA™

AN INTRODUCTION TO  
PROBLEM SOLVING  
AND PROGRAMMING

7TH EDITION



WALTER SAVITCH

**Inheritance,  
Polymorphism,  
and Interfaces**

**8**

## LISTING 8.1 The Class Person

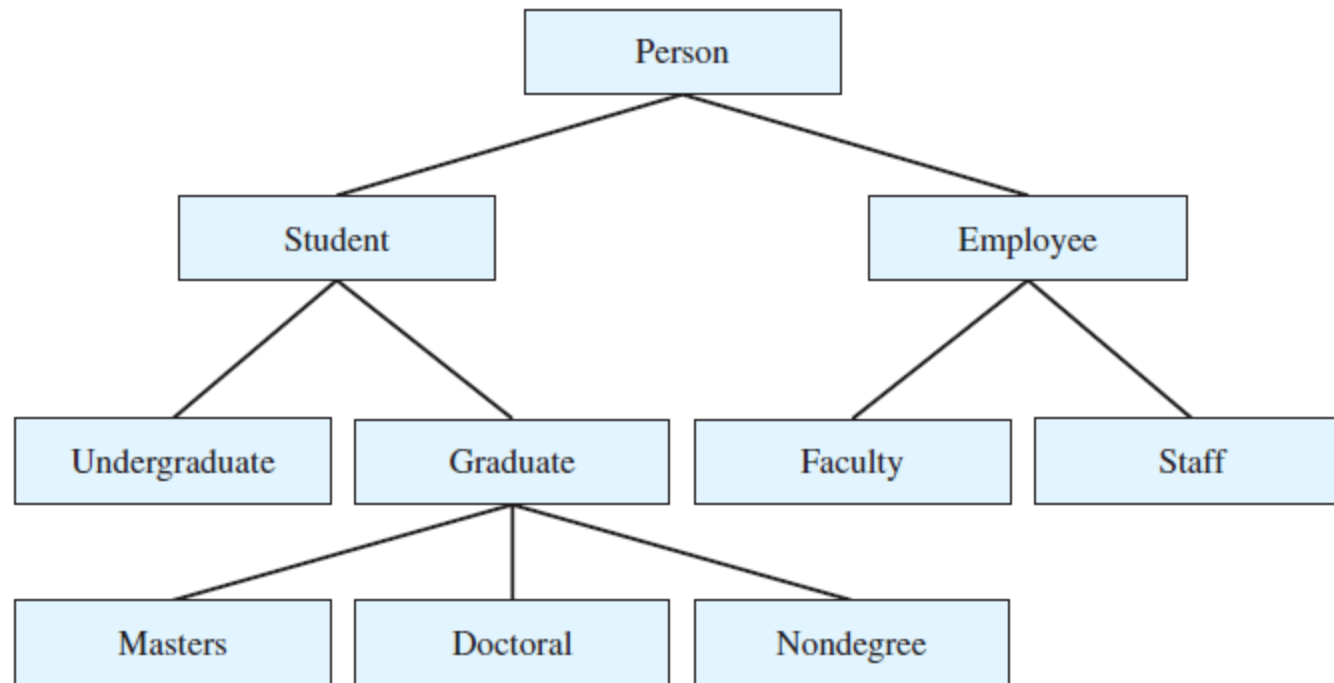
---

```
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 hasSameName(Person otherPerson)
    {
        return this.name.equalsIgnoreCase(otherPerson.name);
    }
}
```

---

**FIGURE 8.1 A Class Hierarchy**

---



## LISTING 8.2 A Derived Class (part 1 of 2)

---

```
public class Student extends Person
{
    private int studentNumber;
    public Student()
    {
        super();
        studentNumber = 0; // Indicating no number yet
    }
    public Student(String initialName, int initialStudentNumber)
    {
        super(initialName);
        studentNumber = initialNumber;
    }
    public void reset(String newName, int newStudentNumber)
    {
        setName(newName);
        studentNumber = newStudentNumber;
    }
    public int getStudentNumber()
    {
        return studentNumber;
    }
    public void setStudentNumber(int newStudentNumber)
    {
        studentNumber = newStudentNumber;
    }
    public void writeOutput()
    {
        System.out.println("Name: " + getName());
        System.out.println("Student Number: " + studentNumber);
    }
}
```

*super is explained in a later section. Do not worry about it until you reach the discussion of it in the text.*


```
public boolean equals(Student otherStudent)
{
    return this.hasSameName(otherStudent) &&
           (this.studentNumber == otherStudent.studentNumber);
}
}
```

### LISTING 8.3 A Demonstration of Inheritance Using Student

---

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

*setName is inherited from  
the class Person.*

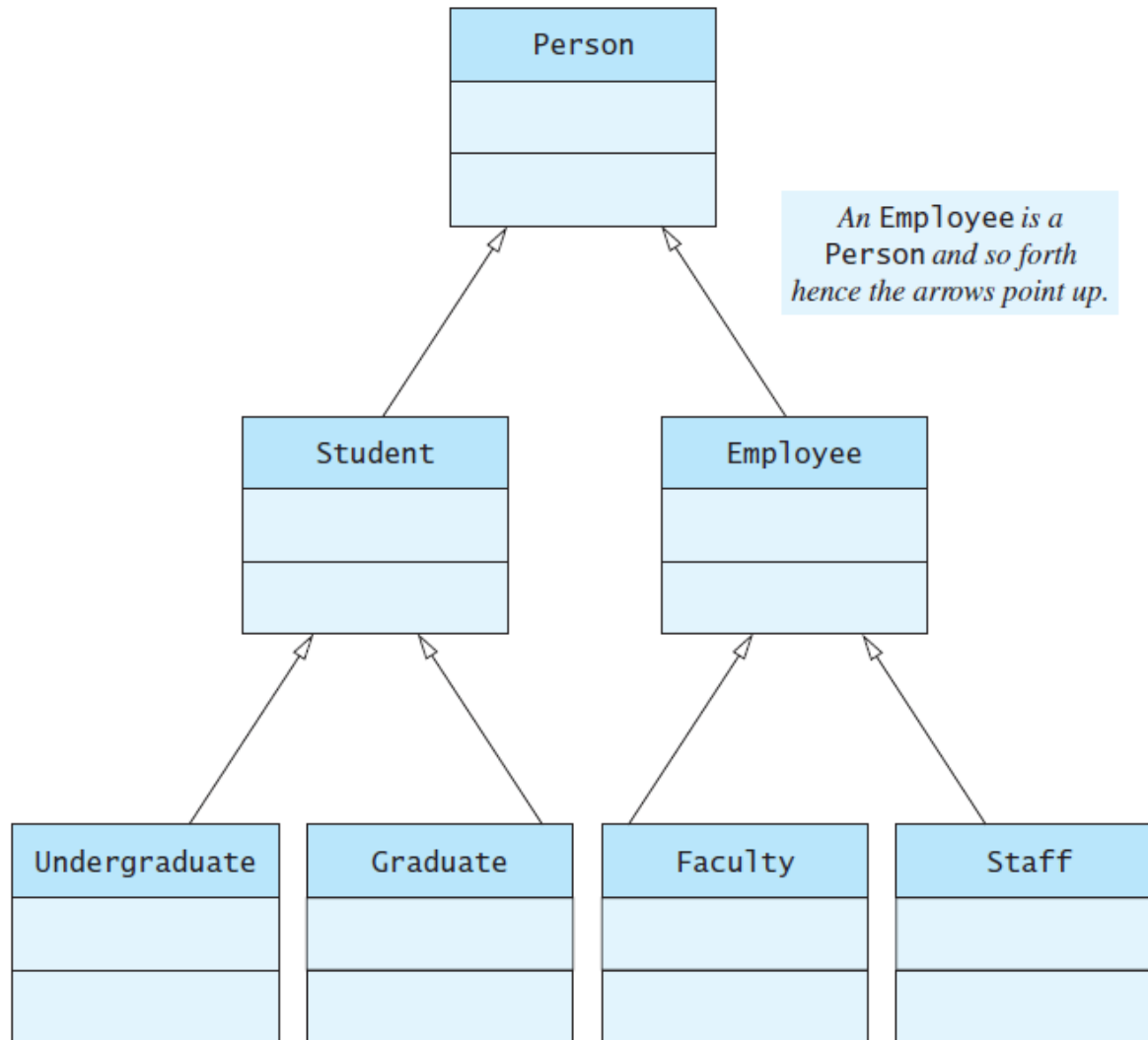


---

#### Screen Output

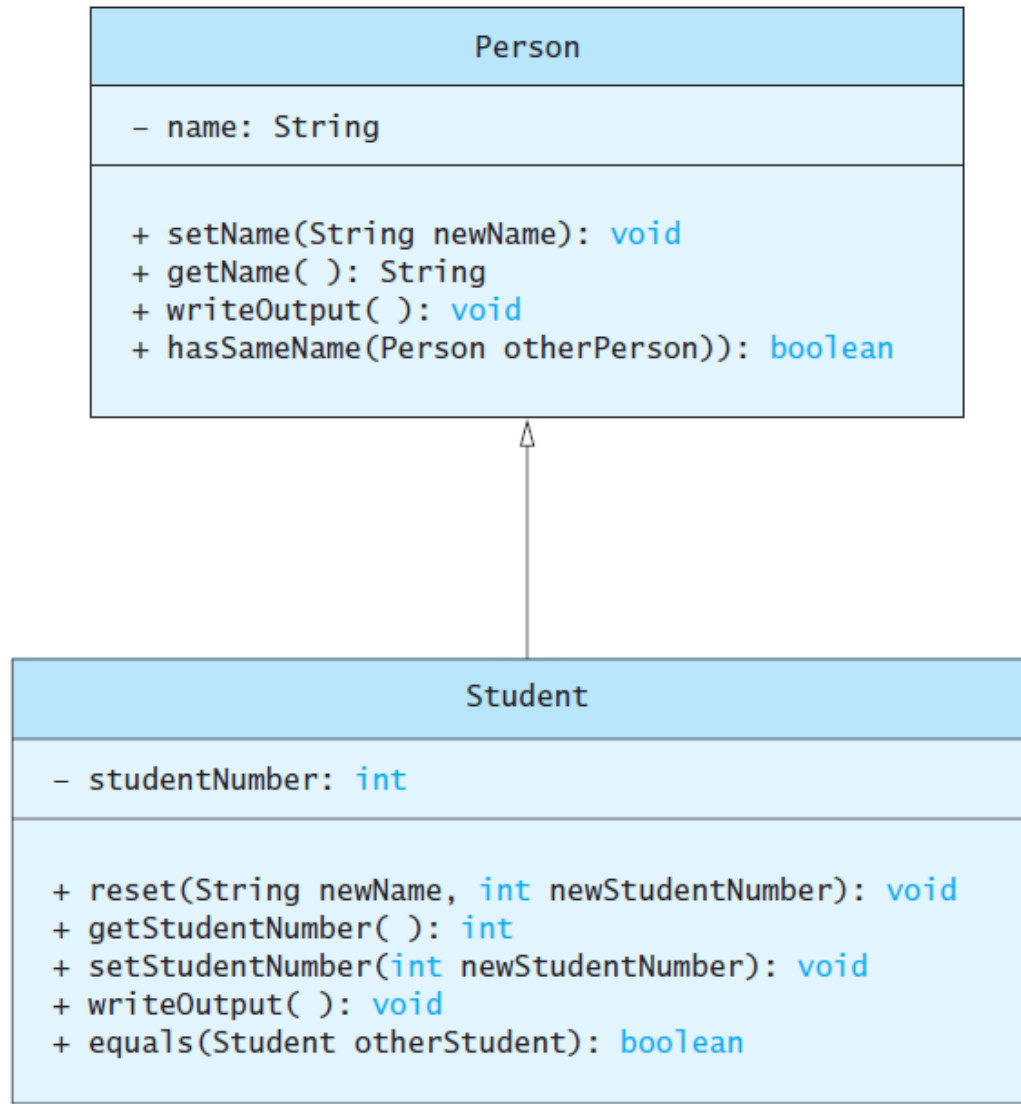
```
Name: Warren Peace
Student Number: 1234
```

**FIGURE 8.2** A Class Hierarchy in UML Notation



**FIGURE 8.3** Some Details of the UML Class Hierarchy Shown in Figure 8.2

---



## LISTING 8.4 A Derived Class of a Derived Class

---

```
public class Undergraduate extends Student
{
    private int level; //1 for freshman, 2 for sophomore
                        //3 for junior, or 4 for senior.
    public Undergraduate()
    {
        super();
        level = 1
    }
    public Undergraduate(String initialName,
                        int initialStudentNumber, int initialLevel)
    {
        super(initialName, initialStudentNumber);
        setLevel(initialLevel); //checks 1 <= initialLevel <= 4
    }
    public void reset(String newName, int newStudentNumber,
                    int newLevel)
    {
        reset(newName, newStudentNumber); //Student's reset
        setLevel(newLevel); //Checks 1 <= newLevel <= 4
    }
}
```

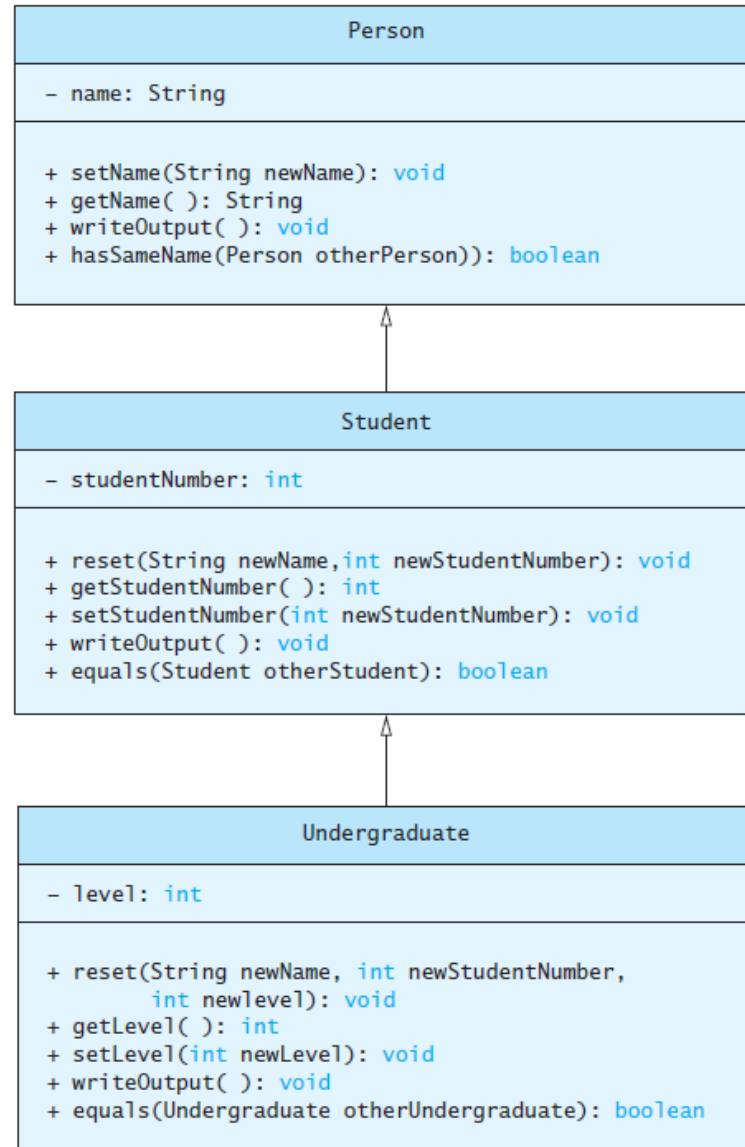


```

public int getLevel()
{
    return level;
}
public void setLevel(int newLevel)
{
    if ((1 <= newLevel) && (newLevel <= 4))
        level = newLevel;
    else
    {
        System.out.println("Illegal level!");
        System.exit(0);
    }
}
public void writeOutput()
{
    super.writeOutput();
    System.out.println("StudentLevel: " + level);
}
public boolean equals(Undergraduate otherUndergraduate)
{
    return equals(Student)otherUndergraduate) &&
        (this.level == otherUndergraduate.level);
}
}

```

**FIGURE 8.4** More Details of the UML Class Hierarchy Shown in Figure 8.2



### LISTING 8.5 A Better equals Method for the Class Student

---

```
public boolean equals(Object otherObject)
{
    boolean isEqual = false;
    if ((otherObject != null) &&
        (otherObject instanceof Student))
    {
        Student otherStudent = (Student)otherObject;
        isEqual = this.sameName(otherStudent) &&
                  (this.studentNumber ==
                   otherStudent.studentNumber);
    }
    return isEqual;
}
```

---

## LISTING 8.6 A Demo of Polymorphism (part 1 of 2)

---

```
public class PolymorphismDemo
{
    public static void main(String[] args)
    {
        Person[] people = new Person[4];
        people[0] = new Undergraduate("Cotty, Manny", 4910, 1);
        people[1] = new Undergraduate("Kick, Anita", 9931, 2);
        people[2] = new Student("DeBanque, Robin", 8812);
        people[3] = new Undergraduate("Bugg, June", 9901, 4);
        for (Person p : people)
        {
            p.writeOutput(); ←
            System.out.println();
        }
    }
}
```

*Even though `p` is of type `Person`, the `writeOutput` method associated with `Undergraduate` or `Student` is invoked depending upon which class was used to create the object.*

---

### Screen Output

```
Name: Cotty, Manny
Student Number: 4910
Student Level: 1

Name: Kick, Anita
Student Number: 9931
Student Level: 2
```

## **LISTING 8.6** A Demo of Polymorphism *(part 2 of 2)*

---

Name: DeBanque, Robin

Student Number: 8812

Name: Bugg, June

Student Number: 9901

Student Level: 4

---

## LISTING 8.7 A Java Interface

---

```
/**  
  An interface for methods that return  
    the perimeter and area of an object.  
*/  
public interface Measurable  
{  
    /** Returns the perimeter. */  
    public double getPerimeter();  
    /** Returns the area. */  
    public double getArea();  
}
```

*Do not forget the semicolons at  
the end of the method headings.*

## LISTING 8.8 An Implementation of the Interface `Measurable`

---

```
/**
 * A class of rectangles.
 */
public class Rectangle implements Measurable
{
    private double myWidth;
    private double myHeight;

    public Rectangle(double width, double height)
    {
        myWidth = width;
        myHeight = height;
    }

    public double getPerimeter()
    {
        return 2 * (myWidth + myHeight);
    }

    public double getArea()
    {
        return myWidth * myHeight;
    }
}
```

---

## LISTING 8.9 Another Implementation of the Interface Measurable

---

```
/**
 * A class of circles.
 */
public class Circle implements Measurable
{
    private double myRadius;
    public Circle(double radius)
    {
        myRadius = radius;
    }
    public double getPerimeter()
    {
        return 2 * Math.PI * myRadius;
    }
    public double getCircumference()
    {
        return getPerimeter();
    }
    public double getArea()
    {
        return Math.PI * myRadius * myRadius;
    }
}
```

*This method is not declared  
in the interface.*

*Calls another method instead  
of repeating its body*



## LISTING 8.10 An Interface for Drawing Shapes Using Keyboard Characters

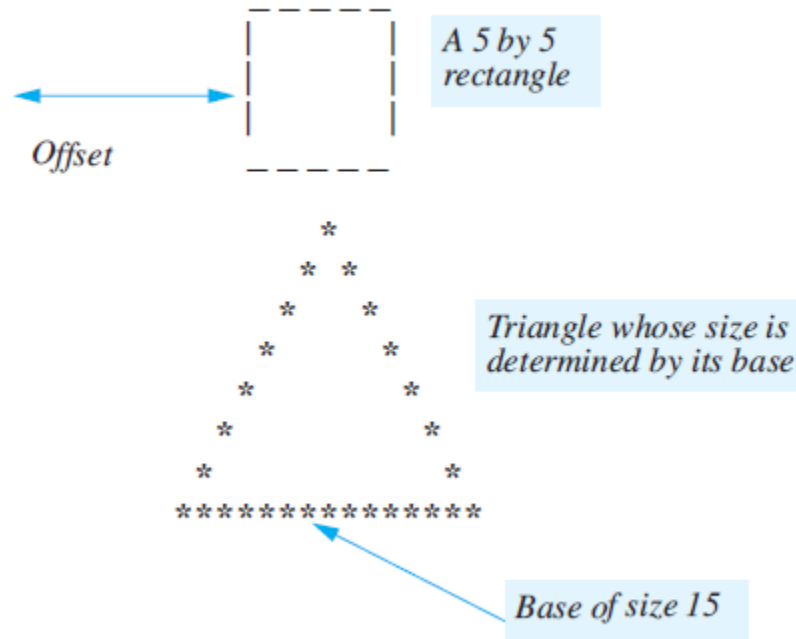
---

```
/**
Interface for simple shapes drawn on
the screen using keyboard characters.
*/
public interface ShapeInterface
{
    /**
    Sets the offset for the shape.
    */
    public void setOffset(int newOffset);
    /**
    Returns the offset for the shape.
    */
    public int getOffset();
    /**
    Draws the shape at lineNumber lines down
    from the current line.
    */
    public void drawAt(int lineNumber);
    /**
    Draws the shape at the current line.
    */
    public void drawHere();
}
```

---

**FIGURE 8.5** A Sample Rectangle and Triangle

---



## LISTING 8.11 Interfaces for Drawing Rectangles and Triangles

---

```
/**
Interface for a rectangle to be drawn on the screen.
*/
public interface RectangleInterface extends ShapeInterface
{
    /**
    Sets the rectangle's dimensions.
    */
    public void set(int newHeight, int newWidth);
}

/**
Interface for a triangle to be drawn on the screen.
*/
public interface TriangleInterface extends ShapeInterface
{
    /**
    Sets the triangle's base.
    */
    public void set(int newBase);
}
```

---

## LISTING 8.12 The Base Class ShapeBasics (part 1 of 2)

---

```
/**
Class for drawing simple shapes on the screen using keyboard
characters. This class will draw an asterisk on the screen as a
test. It is not intended to create a "real" shape, but rather
to be used as a base class for other classes of shapes.
*/
public class ShapeBasics implements ShapeInterface
{
    private int offset;
    public ShapeBasics()
    {
        offset = 0;
    }
    public ShapeBasics(int theOffset)
    {
        offset = theOffset;
    }
    public void setOffset(int newOffset)
    {
        offset = newOffset;
    }
}
```

```

public int getOffset()
{
    return offset;
}
public void drawAt(int lineNumber)
{
    for (int count = 0; count < lineNumber; count++)
        System.out.println();
    drawHere();
}
public void drawHere()
{
    for (int count = 0; count < offset; count++)
        System.out.print(' ');
    System.out.println('*');
}
}

```

### LISTING 8.13 The Class Rectangle (part 1 of 2)

---

```
/**  
Class for drawing rectangles on the screen using keyboard  
characters. Each character is higher than it is wide, so  
these rectangles will look taller than you might expect.  
Inherits getOffset, setOffset, and drawAt from the class  
ShapeBasics.  
*/  
public class Rectangle extends ShapeBasics  
                        implements RectangleInterface  
{  
    private int height;  
    private int width;  
  
    public Rectangle()  
    {  
        super();  
        height = 0;  
        width = 0;  
    }  
    public Rectangle(int theOffset, int theHeight,  
                    int theWidth)  
    {  
        super(theOffset);  
        height = theHeight;  
        width = theWidth;  
    }  
}
```

```

public void set(int newHeight, int newWidth)
{
    height = newHeight;
    width = newWidth;
}
public void drawHere();
{
    drawHorizontalLine();
    drawSides();
    drawHorizontalLine();
}
private void drawHorizontalLine()
{
    skipSpaces(getOffset());
    for (int count = 0; count < width; count++)
        System.out.print('-');
    System.out.println();
}
private void drawSides()
{
    for (int count = 0; count < (height - 2); count++)
        drawOneLineOfSides();
}
private void drawOneLineOfSides()
{
    skipSpaces(getOffset());
    System.out.print('|');
    skipSpaces(width - 2);
    System.out.println('|');
}
//Writes the indicated number of spaces.
private static void skipSpaces(int number)
{
    for (int count = 0; count < number; count++)
        System.out.print(' ');
}
}

```

*For clarity, the method skipSpaces was made static because it does not depend on an object.*

## LISTING 8.14 The Class Triangle (part 1 of 2)

---

```
/**  
 Class for drawing triangles on the screen using keyboard  
 characters. A triangle points up. Its size is determined  
 by the length of its base, which must be an odd integer.  
 Inherits getOffset, setOffset, and drawAt from the class  
 ShapeBasics.  
 */  
public class Triangle extends ShapeBasics  
    implements TriangleInterface  
{  
    private int base;  
  
    public Triangle()  
    {  
        super();  
        base = 0;  
    }  
    public Triangle(int theOffset, int theBase)  
    {  
        super(theOffset);  
        base = theBase;  
    }  
}
```



```

/** Precondition: newBase is odd. */
public void set(int newBase)
{
    base = newBase;
}
public void drawHere()
{
    drawTop();
    drawBase();
}
private void drawBase()
{
    skipSpaces(getOffset());
    for (int count = 0; count < base; count++)
        System.out.print('*');
    System.out.println();
}
private void drawTop()
{
    //startOfLine == number of spaces to the
    //first '*' on a line. Initially set to the
    //number of spaces before the topmost '*'.
    int startOfLine = getOffset() + base / 2;
    skipSpaces(startOfLine);
    System.out.println('*'); //top '*'
}

```

```

    int lineCount = base / 2 - 1; //height above base

    //insideWidth == number of spaces between the
    //two '*'s on a line.
    int insideWidth = 1;
    for (int count = 0; count < lineCount; count++)
    {
        //Down one line, so the first '*' is one more
        //space to the left.
        startOfLine --;
        skipSpaces(startOfLine);
        System.out.print('*');
        skipSpaces(insideWidth);
        System.out.println('*');

        //Down one line, so the inside is 2 spaces wider.
        insideWidth = insideWidth + 2;
    }
}

private static void skipSpaces(int number)
{
    for (int count = 0; count < number; count++)
        System.out.print(' ');
}
}

```

## LISTING 8.15 A Demonstration of Triangle and Rectangle (part 1 of 2)

---

```
/**  
A program that draws a fir tree composed of a triangle and  
a rectangle, both drawn using keyboard characters.  
*/  
public class TreeDemo  
{  
    public static final int INDENT = 5;  
    public static final int TREE_TOP_WIDTH = 21; // must be odd  
    public static final int TREE_BOTTOM_WIDTH = 4;  
    public static final int TREE_BOTTOM_HEIGHT = 4;  
  
    public static void main(String[] args)  
    {  
        drawTree(TREE_TOP_WIDTH, TREE_BOTTOM_WIDTH,  
                 TREE_BOTTOM_HEIGHT);  
    }  
}
```

```

public static void drawTree(int topWidth, int bottomWidth,
                           int bottomHeight)
{
    System.out.println("    Save the Redwoods!");
    TriangleInterface treeTop = new Triangle(INDENT, topWidth)
    drawTop(treeTop);
    RectangleInterface treeTrunk = new Rectangle(INDENT +
                                                (topWidth / 2) - (bottomWidth / 2),
                                                bottomHeight, bottomWidth);
    drawTrunk(treeTrunk);
}
private static void drawTop(TriangleInterface treeTop)
{
    treeTop.drawAt(1);
}
private static void drawTrunk(RectangleInterface treeTrunk)
{
    treeTrunk.drawHere(); // or treeTrunk.drawAt(0);
}
}

```

### Screen Output

## Save the Redwoods!

1001

## LISTING 8.16 First Attempt to Define a Fruit Class

---

```
public class Fruit
{
    private String fruitName;
    public Fruit()
    {
        fruitName = "";
    }
    public Fruit(String name)
    {
        fruitName = name;
    }
    public void setName(String name)
    {
        fruitName = name;
    }
    public String getName()
    {
        return fruitName;
    }
}
```

---

## LISTING 8.17 Program to Sort an Array of Fruit Objects

---

```
import java.util.Arrays;
public class FruitDemo
{
    public static void main(String[] args)
    {
        Fruit[] fruits = new Fruit[4];
        fruits[0] = new Fruit("Orange");
        fruits[1] = new Fruit("Apple");
        fruits[2] = new Fruit("Kiwi");
        fruits[3] = new Fruit("Durian");
        Arrays.sort(fruits);
        // Output the sorted array of fruits
        for (Fruit f : fruits)
        {
            System.out.println(f.getName());
        }
    }
}
```

---

### **LISTING 8.18** A Fruit Class implementing Comparable *(part 1 of 2)*

---

```
public class Fruit implements Comparable
{
    private String fruitName;
    public Fruit()
    {
        fruitName = "";
    }
    public Fruit(String name)
    {
        fruitName = name;
    }
}
```



```

public void setName(String name)
{
    fruitName = name;
}
public String getName()
{
    return fruitName;
}
public int compareTo(Object o)
{
    if ((o != null) &&
        (o instanceof Fruit))
    {
        Fruit otherFruit = (Fruit) o;
        return (fruitName.compareTo(otherFruit.fruitName));
    }
    return -1;    // Default if other object is not a Fruit
}
}

```

## LISTING 8.19 The Abstract Class ShapeBase

---

```
/**  
Abstract base class for drawing simple shapes on the screen  
using characters.  
*/  
public abstract class ShapeBase implements ShapeInterface  
{  
    private int offset;  
    public abstract void drawHere();
```

*The rest of the class is identical to ShapeBasics in Listing 8.12, except for the names of the constructors. Only the method drawHere is abstract. Methods other than drawHere have bodies and do not have the keyword abstract in their headings. We repeat one such method here:*

```
    public void drawAt(int lineNumber)  
    {  
        for (int count = 0; count < lineNumber; count++)  
            System.out.println();  
        drawHere();  
    }  
}
```

---

## LISTING 8.20 A Window Interface Derived from JFrame (part 1 of 2)

---

```
import javax.swing.JButton;  
import javax.swing.JFrame;  
import java.awt.Container;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;
```

*To complete 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.*

```
/**
```

```
Simple demonstration of putting buttons in a JFrame window.
```

```
*/
```

```
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.");
    }
}

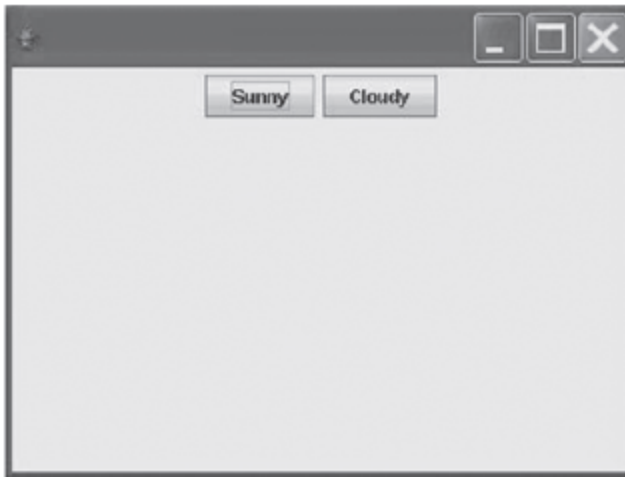
```

## LISTING 8.21 Running a JFrame Class from an Application

---

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

### Screen Output



*This GUI changes color in reaction to button clicks in the same way as the applet in Listing 6.22.*

## LISTING 8.22 A Listener Class for Window Events from JFrame GUIs

---

*The class WindowDestroyer is not defined for you in Java. This is a class that you, the programmer, must define.*

```
import java.awt.event.WindowAdapter;  
import java.awt.event.WindowEvent;
```

```
/**
```

*If you register an object of this class as a listener to any object of the class JFrame, the object will end the program and close the JFrame window if the user clicks the window's close-window button.*

```
*/
```

```
public class WindowDestroyer extends WindowAdapter  
{  
    public void windowClosing(WindowEvent e)  
    {  
        System.exit(0);  
    }  
}
```

---