8.4 Interfaces and Abstract Classes

- Java Interfaces
- Implementing an Interface
- An Interface as a Type
- Extending an Interface

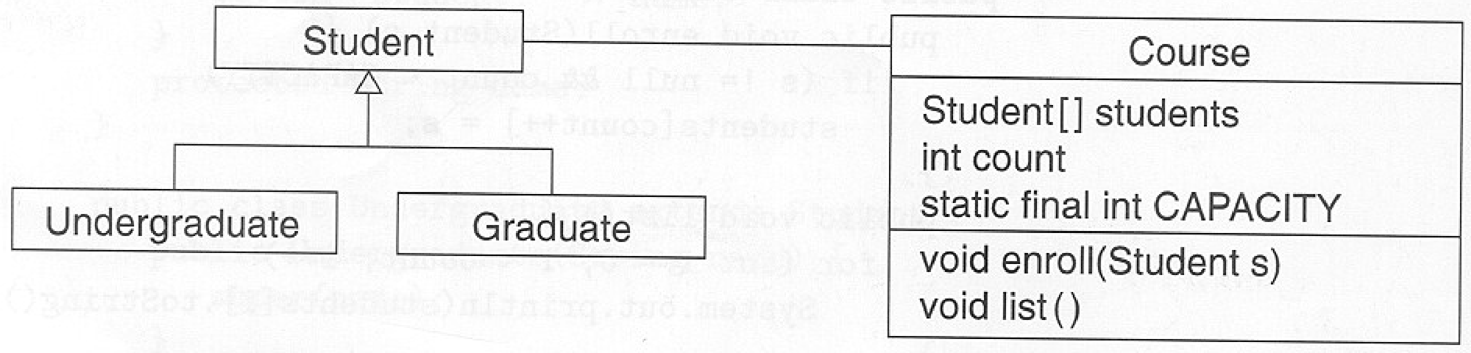


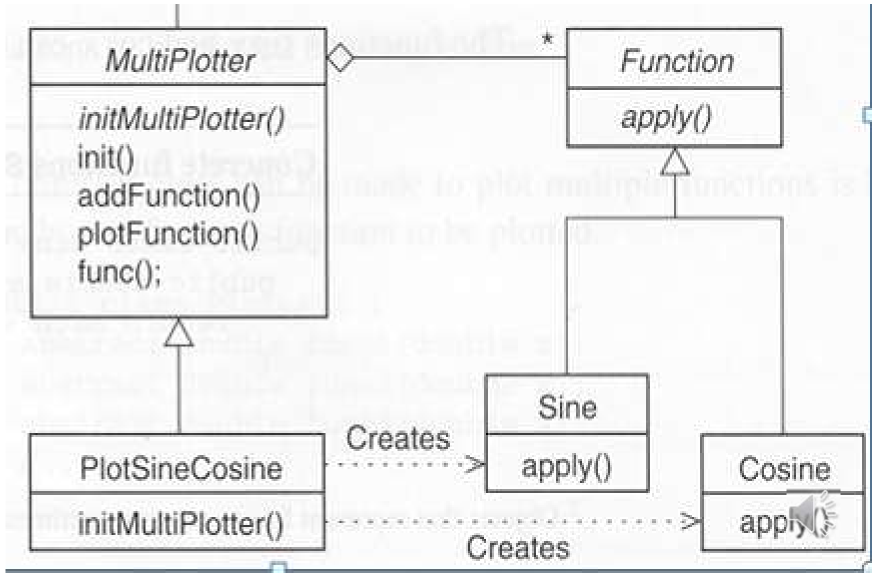
Figure 5.2

Students and courses.

Java Interfaces

- A program component that contains headings for a number of public methods
  - » Will include comments that describe the methods

- Interface can also define public named constants
- View example interface, listing 8.1
  interface Measurable



// 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 ();
}
```

Java Interfaces

- Interface name begins with uppercase letter
- Stored in a file with suffix `.java`
- Interface does not include
  - » Declarations of constructors
  - » Instance variables
  - » Method bodies

Implementing an Interface

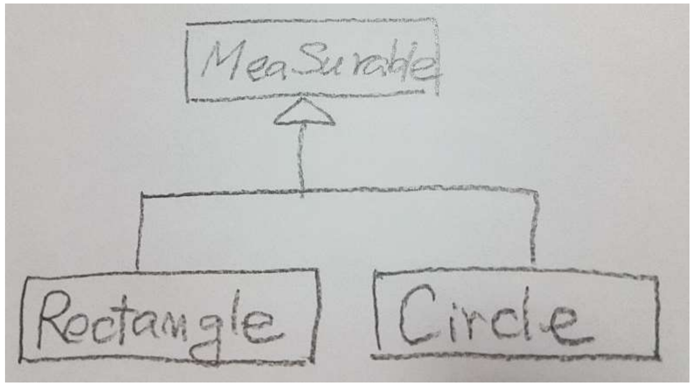
- To implement a method, a class must
  - » Include the phrase

    ```
    implements Interface_name
    ```

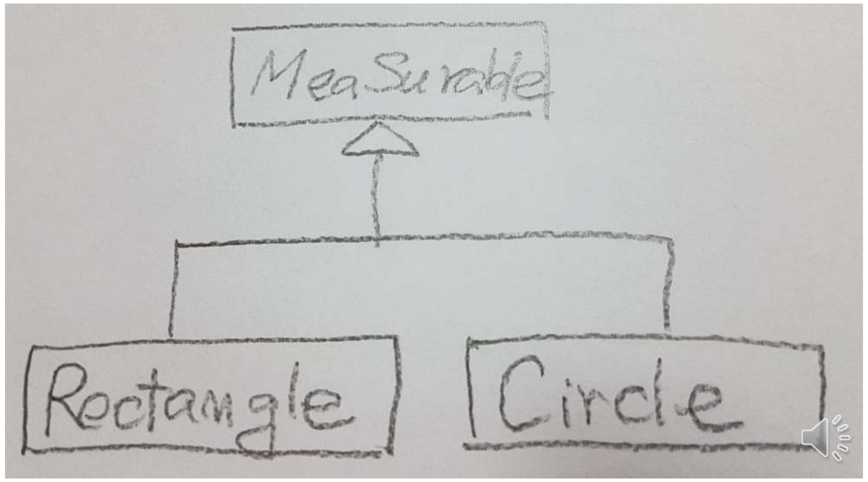
  - » Define each specified method
- View sample class, listing 8.8

  ```
  class Rectangle implements Measurable
  ```

- View another class, listing 8.3 which also implements Measurable class Circle

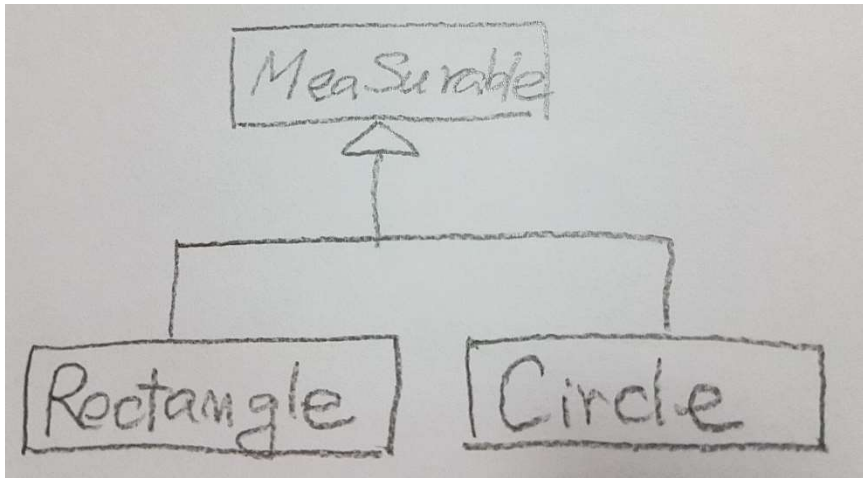


```
// Listing 8.8 An implementation of the Interface Measurable
/**A class of rectangles.*/
public class Rectangle implements Measurable
{
    private double myWidth;
    private double myHeight;

    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

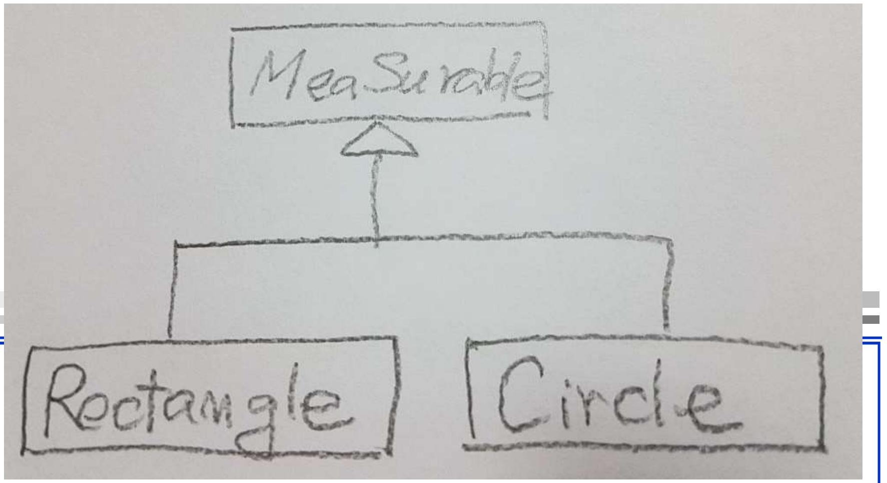
```
/**
A class of circles.
*/
public class Circle implements Measurable
{
    private double myRadius;
    public Circle (double radius)
    {
        myRadius = radius;      }
    public double getPerimeter ()
    {
        return 2 * Math.PI * myRadius;      }
 public double getCircumference ()
 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



An Interface as a Type

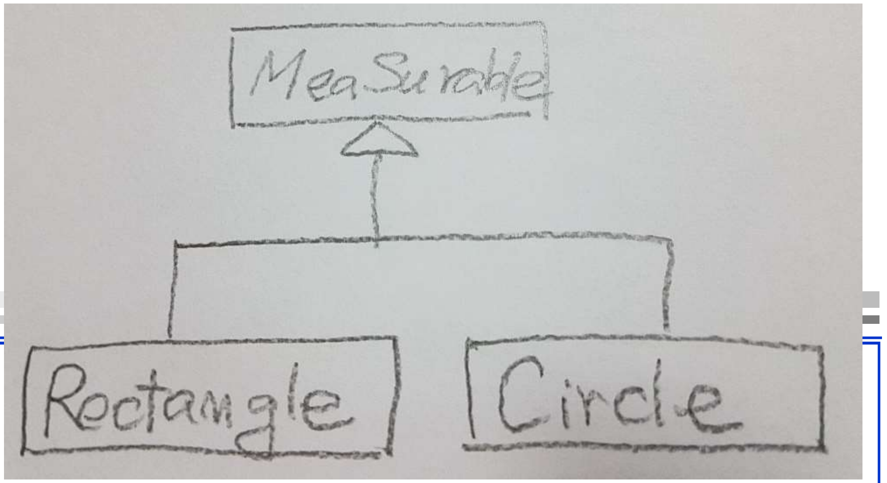
- Possible to write a method that has a parameter as an interface type
  - » An interface is a reference type
- Program invokes the method passing it an object of any class which implements that interface

- The method can substitute one object for another
  - » Called *polymorphism*
- This is made possible by mechanism
  - » *Dynamic binding*
  - » Also known as *late binding*



```
Public static void display (Measurable figure)
{
    double perimeter = figure.getParimeter():
    double area = figure.getArea();
    System.out.println("Perimeter = " + perimeter + ": area= " + area);
}
```

```
Measurable  box = new Rectangle(5.0, 5.0) ;
Measurable disc = new Circle(5.0);
```



```
Measurable m;
Rectangle box = new Rectangle(5.0, 5.0);
m = box;
Display(m) ;
Circle disc = new Circle(5.0);
m = disc
Display(m);
```

```
Measurable  m = new Circle(5.0);
System.out.printlm(m.getCircumference());  //
```

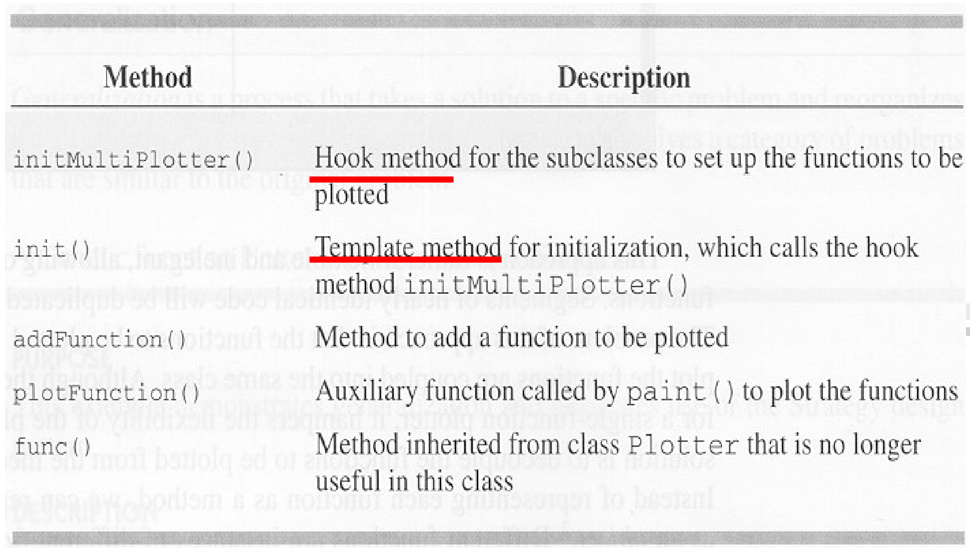
➔ can invoke only a method that is in the

```
Circle c= (Circle) m;
System.out.printlm(c.getCircumference());
```

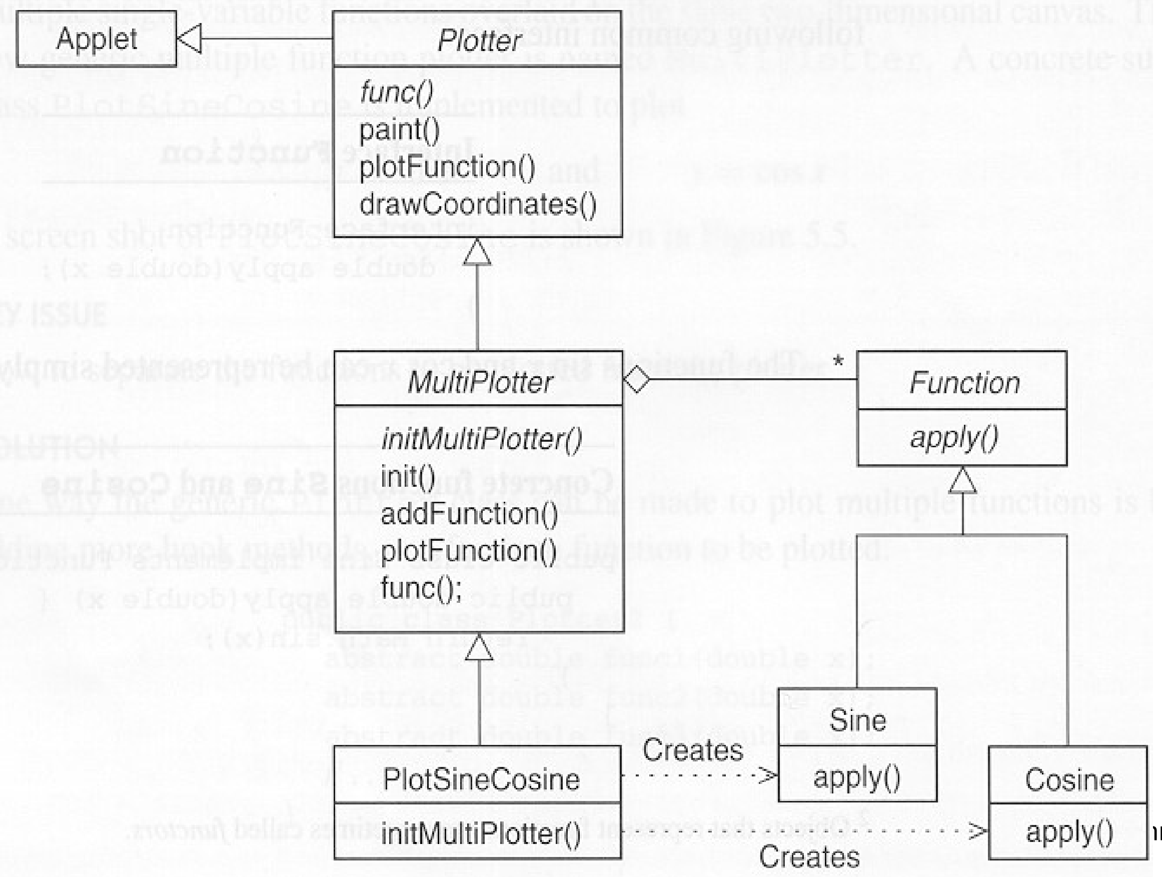
(**Generalizing)

```
//p. 272 : Interface Function
Interface Function {
            double apply (double x);
}
```

```
//p. 272 : Concrete  Functions Sine and Cosine
Public class Sine implements Function{
            public double apply(double x) {
                        return Math.sin(x);
            }
}
Public class Cosine implements Function{
            public double apply(double x) {
                        return Math.cos(x);
            }
}
```



Method	Description
`initMultiPlotter()`	Hook method for the subclasses to set up the functions to be plotted
`init()`	Template method for initialization, which calls the hook method `initMultiPlotter()`
`addFunction()`	Method to add a function to be plotted
`plotFunction()`	Auxiliary function called by `paint()` to plot the functions
`func()`	Method inherited from class `Plotter` that is no longer useful in this class



```java
// Generic multiple function plotter : MultiPlotter.java

import java.awt.*;

public abstract class MultiPlotter extends Plotter {

  abstract public void initMultiPlotter();

  public void init() {   // template method
    super.init();
    initMultiPlotter();  // hook method
  }

  final public void addFunction(Function f, Color c) {
    if (numOfFunctions < MAX_FUNCTIONS && f  !=  null) {
      functions[numOfFunctions] = f;
      colors[numOfFunctions++] = c;
    }
  }
```

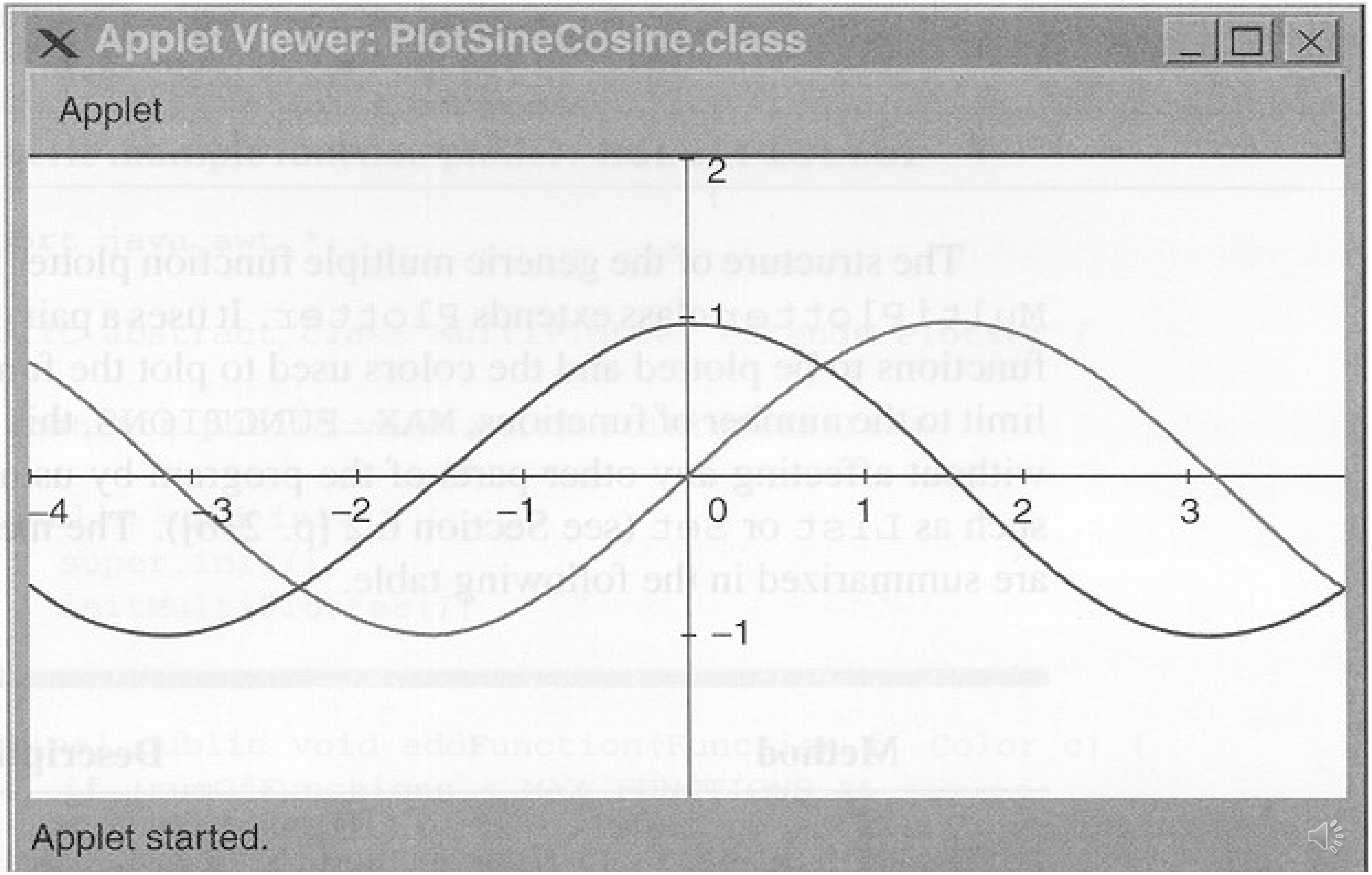
```
// auxiliary function called by paint() to plot the functions
  protected void plotFunction(Graphics g) {
    for (int i = 0; i < numOfFunctions; i++) {
      if (functions[i] != null) {
        Color c = colors[i];
        if (c != null)
          g.setColor(c);
        else
          g.setColor(Color.black);
          for (int px = 0; px < d.width; px++) {
            try {
              double x = (double) (px - xorigin) / (double) xratio;
              double y = functions[i].apply(x);
              int py = yorigin - (int) (y * yratio);
              g.fillOval(px - 1, py - 1, 3, 3);
            } catch (Exception  e) {}
          }
      }
    }
  }
```

```
// method inherited from class Plotter that is no longer useful in this class
 public double func(double x) {
   return 0.0;
 }

 // MAX_FUNCTIONS : a limit to the number of functions
 protected static int MAX_FUNCTIONS = 5;
 protected int numOfFunctions = 0;

 // uses a pair of parallel arrays to store the function to be plotted
 //     and the colors used to plot the functions

 protected Function functions[] = new Function[MAX_FUNCTIONS]; //
 protected Color colors[] = new Color[MAX_FUNCTIONS];


}
```

(**Generalizing)

- Concrete subclass of Multiplotter

```
//Concrete multiple function plotter : MultiPlotter.java

import java.awt.Color;

public class PlotSineCosine extends MultiPlotter {
  public void initMultiPlotter() {
    addFunction(new Sine(), Color.green);
    addFunction(new Cosine(), Color.blue);
  }
}
```


Applet Viewer: PlotSineCosine.class
Applet
2
1
−1
−4
−3
−2
−1
0
1
2
3
Applet started.

Extending an Interface

- to extend the existing interface
  - » to define a new interface which builds on an existing interface
  - » A class that implements the new interface must implement all the methods of both interfaces

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 drawing.
    */
    public void setOffset(int newOffset);

    /**
     Returns the offset for the drawing.
    */
    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();
}
```

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 Basic Class – ShapeBasics

```
/**
 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;
   }

   /**
    Draws the shape at lineNumber lines down
    from the current line.
   */
   public void drawAt(int lineNumber)
   {
       for (int count = 0; count < lineNumber; count++)
           System.out.println( );
       drawHere( );
   }

   /**
    Draws the shape at the current line.
   */
   public void drawHere()
   {
       for (int count = 0; count < offset; count++)
           System.out.print(' ');
       System.out.println('*');
   }
}
```

Listing 8.13 The Class – Rectangle

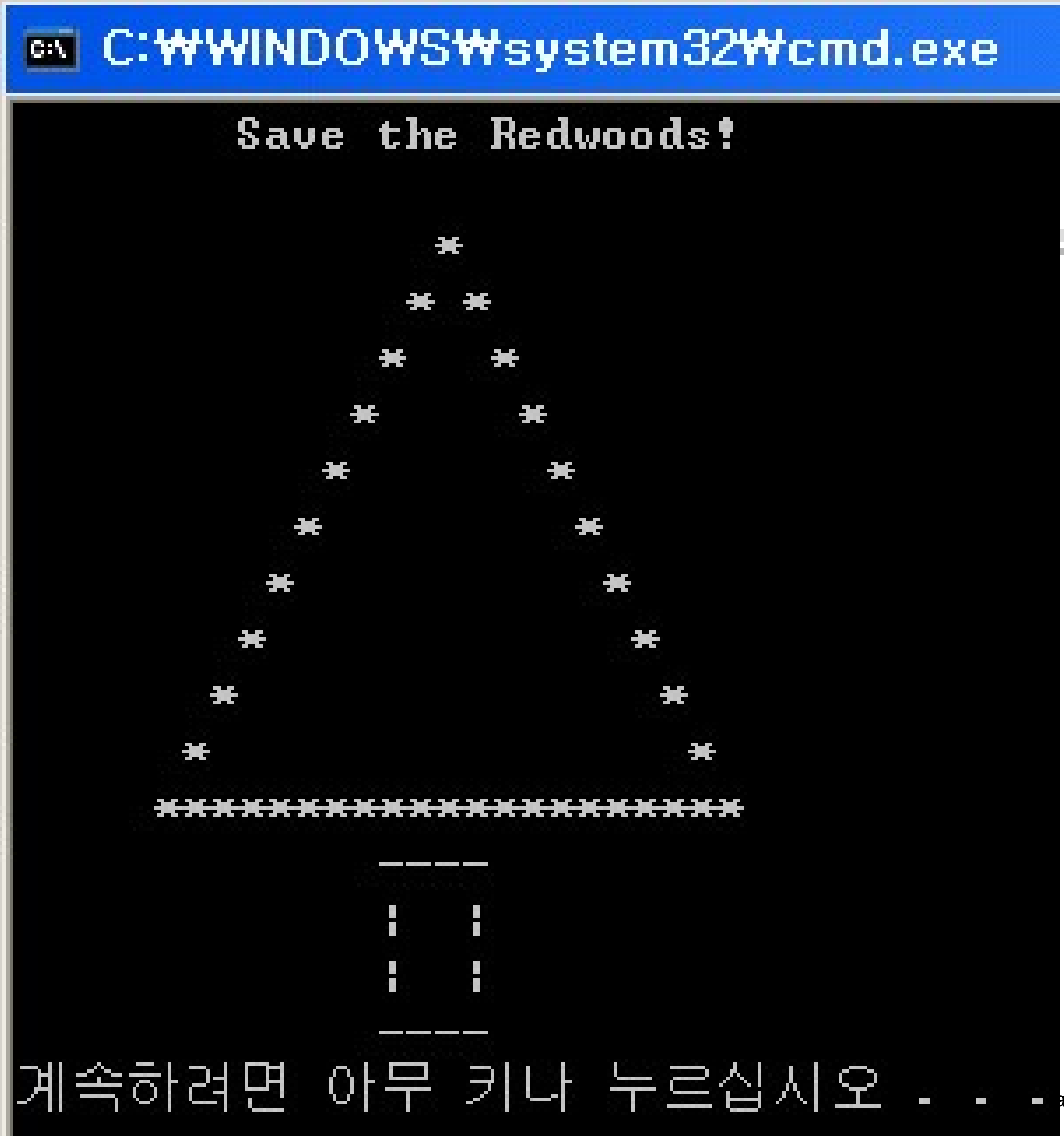
```
/**
 Class for drawing rectangles on the screen using keyboard
 characters. Each character is higher than it is wide, so
 these rectangles will look taller than you might expect.
 Inherits getOffset, setOffset, and drawAt from the class
 ShapeBasics.
*/
public class Rectangle extends ShapeBasics 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;
    }
/**
  Draws the shape at the current line.
 */
 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(' ');
 }
}
```

Listing 8.14 The Class – Triangle

```
**
 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;
    }
/**
  Draws the shape at current line.
 */
 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

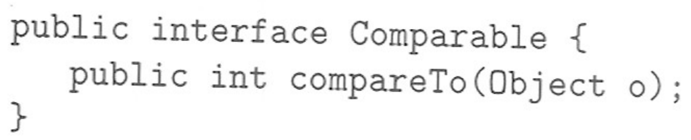
```
/**
 A program that draws a fir tree composed of a triangle and
 a rectangle, both drawn using keyboard characters.
*/
public class TreeDemo
{
   public static final int INDENT = 5;
   public static final int TREE_TOP_WIDTH = 21;// must be odd
   public static final int TREE_BOTTOM_WIDTH = 4;
  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



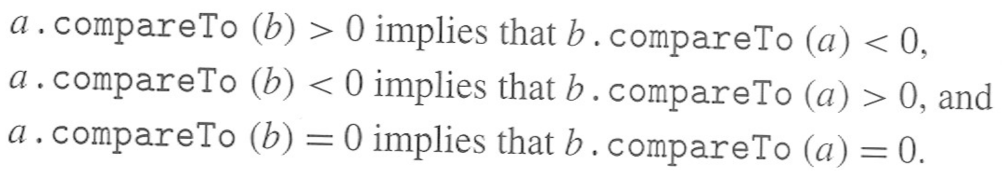
```
        Save the Redwoods!

               *
              * *
             *   *
            *     *
           *       *
          *         *
         *           *
        *             *
       *               *
      *                 *
     *********************
             ----
             |  |
             |  |
             ----
계속하려면 아무 키나 누르십시오 . . .
```

Case Study : Comparable Interface

- Two way to define orders on objects
  - » 1) a natural order : by implementing Comparable interface
  - » 2) Arbitrary order : by comparators, or classes that implement Comparator interface



```
public interface Comparable {
   public int compareTo(Object o);
}
```



a.compareTo $(b) > 0$ implies that b.compareTo $(a) < 0$,

a.compareTo $(b) < 0$ implies that b.compareTo $(a) > 0$, and

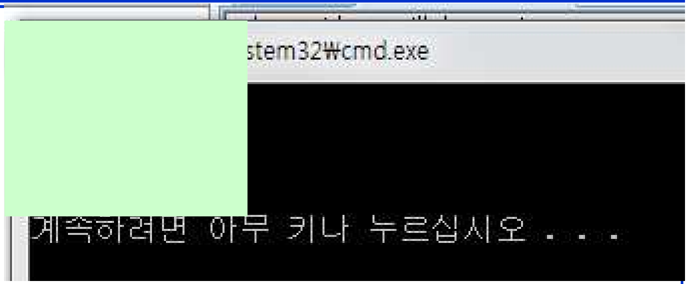
a.compareTo $(b) = 0$ implies that b.compareTo $(a) = 0$.

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());
                }
        }
}
```

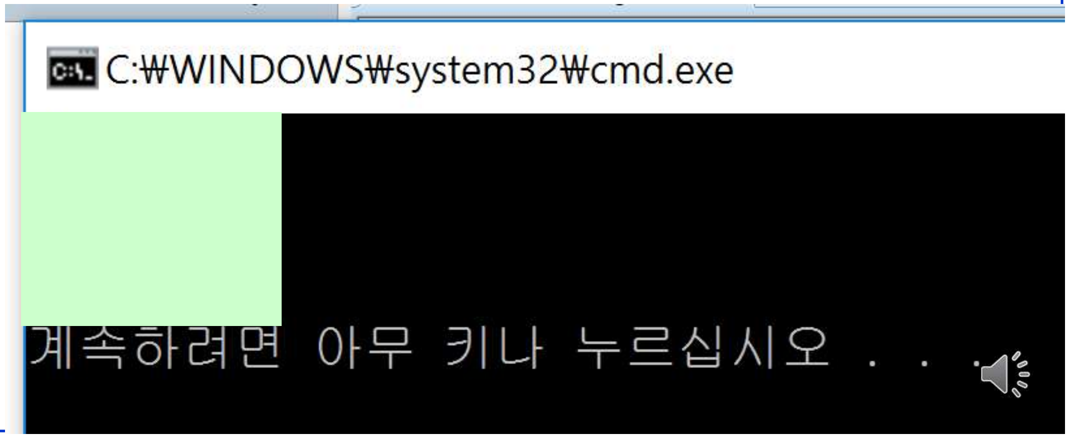


```
계속하려면 아무 키나 누르십시오 . . .
```

Listing 8.18 a Fruit Class implementing Comparable

```
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));
  /*** Alternate definition of comparison using fruit length ***/
  /*
                          if (fruitName.length() > otherFruit.fruitName.length())
                                      return 1;
                          else if (fruitName.length() < otherFruit.fruitName.length())
                                      return -1;
                          else
                                      return 0;
 */
              }
              return -1;              // Default if other object is not a Fruit
  }

}
```



```
C:₩WINDOWS₩system32₩cmd.exe
계속하려면 아무 키나 누르십시오 . . .
```

Abstract Classes

- Cannot [redacted] of an ***abstract class***
  - » Example: `Figure` class in character graphics program
  - » An abstract class is used [redacted] instead of being used to create objects.
- Abstract classes simplify program design by not requiring you to supply methods [redacted]
  - » Example: <u>`drawHere` method</u> is overridden in all classes derived from `Figure`.
- Specify that a method is abstract if you don't want to implement it:

```
public abstract void drawHere();
```

- Any class that has an abstract method must be declared as an abstract class:

```
public abstract class Figure
```

Listing 8.19 The Figure class Redone as an Abstract Class

```
// Listing 8.15
public abstract class ShapeBase implements ShapeInterface
{
    private int offset;
    public abstract void drawHere ();
    /*   The rest of the class is identical to ShapeBasics in Listing 8.11,
    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 heir headings.
    We repeat one such method here:    */
    public void drawAt (int lineNumber)
    {
        for (int count = 0 ; count < lineNumber ; count++)
            System.out.println ();
        drawHere ();
    }
```

끝