Imagine that you are creating a program for drawing shapes such as circles, lines and rectangles. You intend to arrange things so that each shape has a corresponding position (usually an APPoint representing a point of the shape) that determines the shape's location. In addition, you will certainly need a method for drawing the shape and another for moving the shape to a different location. So you might begin by defining a Shape class as follows:

public class Shape   
{   
  private APPoint myPosition;   
  
  public Shape( APPoint position )   
  {   
    myPosition = position;   
  }   
  
  public void draw()   
  {   
  }   
  
  public void moveTo( APPoint p )   
  {   
    myPosition = p;   
    draw();   
  }   
}

Your plan is to subclass the Shape class, with each subclass (Circle, Rectangle, and so on) overriding the draw method. This would produce a hierarchy something like this:

The fact that the Shape class defines the moveTo method means that you will probably not have to override this method in the subclasses, and this will save you some time. Furthermore, by including an empty draw method you give yourself the possibility of using the polymorphism properties of the hierarchy. That is, you make it possible to write code like this:

ArrayList<Shape> list = new ArrayList<Shape>();   
  
list.add( new Circle( new APPoint( 102, 50 ), 5.2 ) );   
list.add( new Rectangle( new APPoint( 102, 50 ), 10.0, 12.5 ) );   
// < more shapes added to list >   
  
// draw shapes in list   
for ( Shape s : list )   
  s.draw();

in which, by implicitly casting all the elements in the ArrayList list to Shape, we can with a single statement — namely, s.draw() — cause each Shape in list to be drawn using its own version of draw, customized to the particular type of Shape that it is.

It should be clear, however, that the Shape class itself is not intended to be instantiated directly. That is, your program would probably never contain a statement like this:

Shape s = new Shape( new APPoint( 0, 0 ) );

Rather, the Shape class exists

* to provide a method (moveTo) and a field (myPosition) that are shared by all its subclasses, and
* to provide a method, draw, that will be overridden by each of its subclasses.

In Java, there is a type of class, called an *abstract class*, that is designed never to be instantiated. To designate a class as an abstract class, use the keyword abstract as shown here:

public abstract class Shape   
{   
  private APPoint myPosition;   
  
  public Shape( APPoint position )   
  {   
    myPosition = position;   
  }   
  
  public void draw()   
  {   
  }   
  
  public void moveTo( APPoint p )   
  {   
    myPosition = p;   
    draw();   
  }   
}

Java will not permit an abstract class to be instantiated. However, any subclass of an abstract class that is not itself marked as abstract may be instantiated.

**Exercise 152**

Using the code area below,

1. attempt to instantiate the Shape class in the main method. What is the resulting error message?
2. create a Line class derived from Shape in such a way that the new class can be instantiated, and verify that this is indeed the case. [For the purposes of this exercise, do not override the draw method. Furthermore, think of all Line objects as being parallel to the x-axis. The Line class constructor need therefore only take two arguments, an APPoint and a length]

public abstract class Shape   
{   
  private APPoint myPosition;   
  
  public Shape( APPoint position )   
  {   
    myPosition = position;   
  }   
  
  public void draw()   
  {   
  }   
  
  public void moveTo( APPoint p )   
  {   
    myPosition = p;   
    draw();   
  }   
}

public class Line extends Shape

 {

   private double myLength;

   public Line( APPoint position, double length )

   {

     super( position );

     myLength = length;

   }

 }

public class MainClass

{

  public static void main( String[] args )

  {

    Line d = new Line( new APPoint( 0, 0 ), 5.7 );

    System.out.println( d );

  }

}

1. The following is a sample attempt to instantiate Shape:

Shape s = new Shape( new APPoint( 0, 0 ) );

Executing this causes the following error message to be generated:

Shape is abstract; cannot be instantiated

1. A suitable definition for the Line class is as follows:

public class Line extends Shape  
{  
  private double myLength;  
  
  public Line( APPoint position, double length )  
  {  
    super( position );  
    myLength = length;  
  }  
}

A suitable test that this class can be implemented is provided by these two statements:

Line d = new Line( new APPoint( 0, 0 ), 5.7 );  
System.out.println( d );

When the resulting program is executed, it runs without error and outputs something similar to the following:

Line@765291

Here again is the Shape class:

public abstract class Shape   
{   
  private APPoint myPosition;   
  
  public Shape( APPoint position )   
  {   
    myPosition = position;   
  }   
  
  public void draw()   
  {   
  }   
  
  public void moveTo( APPoint p )   
  {   
    myPosition = p;   
    draw();   
  }   
}

As we remarked on the previous page, we only include the draw method in this definition so that it may be overridden in subclasses. There is no reason to provide any code for the body of this method since we fully intend to override it in every subclass.

Methods such as draw are common in abstract classes, so Java provides a shorthand for declaring them. We add the keyword abstract to the declaration of the method and replace the empty body by a statement-ending semicolon, like this:

public abstract class Shape   
{   
  private APPoint myPosition;   
  
  public Shape( APPoint position )   
  {   
    myPosition = position;   
  }   
  
  public abstract void draw();   
  
  public void moveTo( APPoint p )   
  {   
    myPosition = p;   
    draw();   
  }   
}

A method that is declared in this way is called an *abstract method*. The only classes that may contain an abstract method are abstract classes; if a class contains an abstract method, it must itself be abstract.

Classes that are *not* abstract — that is, classes that can be instantiated — are said to be *concrete*. In order for a subclass, ClassS, of an abstract class, ClassA, to be concrete,

* ClassS must have no abstract methods, and
* for each abstract superclass, ClassB, of ClassS — and ClassA is one such superclass — each of its abstract methods must be implemented either in ClassS or in some class that is at the same time a subclass of ClassB and a superclass of ClassS.

Put simply, for class ClassS to be a concrete class it must implement all of the abstract methods of all of its superclasses.

*Implementing* an abstract method consists in overriding it by means of a definition that does *not* use the keyword abstract. The definition must have the same return data type as the abstract method and a signature that matches that of the abstract method. (In the case of a method with a void return data type, it is allowed to have an empty body.)

If any abstract method from anywhere higher up the hierarchy has still not been overridden non-abstractly in a subclass or any of its superclasses, then that subclass must itself be declared as an abstract class.

Run the following code, then report and explain the error message that results.

public abstract class Shape   
{   
  private APPoint myPosition;   
  
  public Shape( APPoint position )   
  {   
    myPosition = position;   
  }   
  
  public abstract void draw();   
  
  public void moveTo( APPoint p )   
  {   
    myPosition = p;   
    draw();   
  }   
}

public class Line extends Shape

{

  private double myLength;

  public Line( APPoint p,  double length )

  {

    super( p );

    myLength = length;

  }

  public void draw()

  {

  }

}

public class MainClass

{

  public static void main( String[] args )

  {

    Shape s = new Line( new APPoint( 100, 25 ), 10.6 );

  }

}

### Exercise 153

1. The error message states:

Line is not abstract and does not override abstract method draw() in Shape.

This means that the Line class is not declared as abstract and yet, although it is a direct subclass of the Shape class, it includes no non-abstract definition of Shape's abstract draw method. Java does not tolerate such a state of affairs, so it interrupts execution and issues the above error.

1. To fix this, in the definition of Line we must override draw with a definition that does not involve the keyword abstract. Even a definition with an empty body will do the trick, like this:

public class Line extends Shape  
{  
  private double myLength;  
  
  public Line( APPoint p, double length )  
  {  
    super( p );  
    myLength = length;  
  }  
  
  public void draw()  
  {  
  }  
}