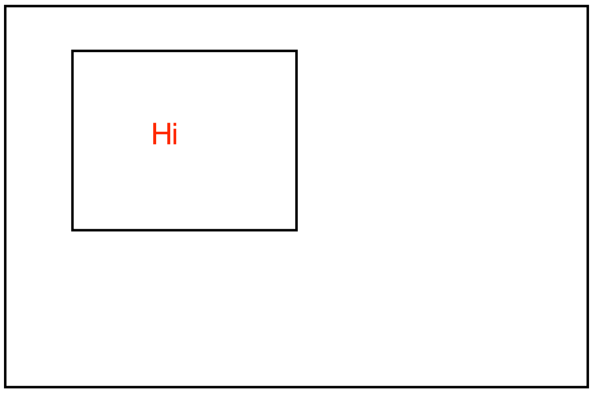
Chapter 1 A First Java Program

1.1 A first Java program

The following program illustrates some basic components of a Java program. The program draws the black outline of a rectangle and the word 'Hi' written in red.



```
import java.awt.*;

public class BoxWithHi extends java.applet.Applet
   {

   public void paint(Graphics g)
      {
      g.setColor(Color.black);
      g.drawRect(30, 20, 100, 80);
      g.setColor(Color.red);
      g.drawString("Hi", 75, 65);
      }

   }
```

Three type styles are used in the example to indicate the following:

- Items in **bold** are reserved words whose meaning is fixed by the Java language.
- Items in `typewriter` are also known to Java. Use exactly as written.
- Items in sans serif are chosen by the programmer (includes all numbers). While the letter g in the above program need not be a g, it must be the same in all five places that it appears.

Each of the words and symbols serves a purpose, though, in the beginning, we will be just be copying many of them.

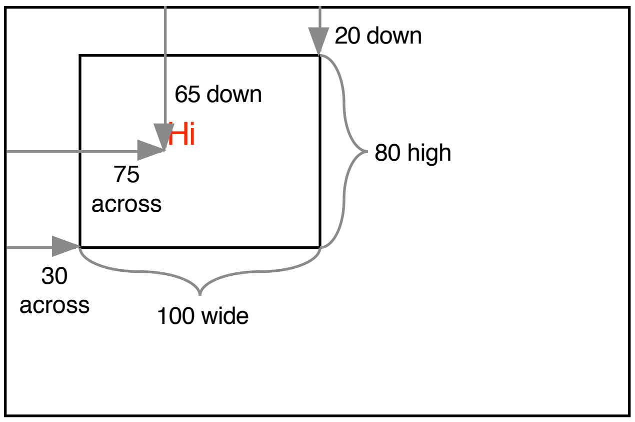
Parts of the program are repeated below with explanations. Explanations may be included in a program. The part of each line which begins with '//' is a comment (explanation). Comments are not interpreted by the computer — they are for human readers of the program only.

```
import java.awt.*;          // declares our intent to use a standard set of graphics commands.

public class BoxWithHi extends java.applet.Applet
                            // names our program " BoxWithHi" and declares that our program is
                            // a modification of a pre-existing program "java.applet.Applet".

   {                        // braces are used to enclose each named portion of a program.
   }
```

```
public void paint(Graphics g)         // Permits "public" access to this part of the program.
                                      // "void" is the return type for this part of the program.
                                      //     Until chapter 3, all return types are "void".
                                      // Names this part of the program "paint".
                                      // Declares that "paint" requires a "Graphics" object "g".

g.setColor(Color.black);              // These statements provide graphics instructions
g.drawRect(30, 20, 100, 80);          // At this point, you should consider modifying only
g.setColor(Color.red);                //     graphics instructions.
g.drawString("Hi", 75, 65);
```

Specifics of the graphics commands above

Colors are set first to black (for the rectangle) and then later to red (for the word "Hi").
Each number counts pixels (dots) on the screen measured from left to right or top to bottom.



The rectangle has its left side 30 pixels from the left of the screen; its top is 20 pixels from the top of the screen. The rectangle is 100 pixels wide and 80 pixels high.

The word "Hi" starts 75 pixels from the left of the screen and 65 pixels from the top of the screen. The positions of words are measured from the bottom of the letters because this is the traditional way typographers do it. The measurement is to the bottom of common letters — letters such as 'y' extend lower. Typographers call the position of the bottom of the common letters the *baseline*.

Important note:

Computers are extremely fussy about how things are spelled and punctuated. The program above has 10 periods, 5 semi-colons and 5 commas. The quotation marks around the word "Hi" are the double quotes generated by holding down the shift key and pressing the key to the right of the semi-colon key. Get one symbol wrong and the program will not work. Furthermore, the Java language is "case sensitive". This means that the proper use of upper case and lower case letters is critical.

Graphics commands

a. Commands

```
g.setColor(Color.red);                          // other colors are listed as constants below
g.setColor(new Color(0, 120, 0));               // red, green, blue (0…255 each)

g.setFont(new Font("SansSerif", Font.BOLD, 24));// font, style (constants below), size in points.

g.drawString(“Any Text”, 50, 100);              // place text in window at over 50, down 100

g.drawLine(30, 20, 120, 100);                   // xstart, ystart, xend, yend

g.drawRect(30, 20, 100, 80);                    // left, top, width, height   – outlined
g.fillRect(30, 20, 100, 80);                    // left, top, width, height   – solid

g.drawRoundRect(30, 20, 100, 80, 10, 10);       // left, top, width, height ,
                                                // arcWidth, arcHeight*  – outlined
g.fillRoundRect(30, 20, 100, 80, 10, 10);       // left, top, width, height ,
                                                // arcWidth, arcHeight*  – solid

g.drawOval(30, 20, 100, 80);                    // left, top, width, height   – outlined
g.fillOval(30, 20, 100, 80);                    // left, top, width, height   – solid

g.drawArc(30, 20, 50, 80, 30, 60);              // left, top, width, height ,
                                                // startAngle, extentAngle
                                                //   – outlines part of the oval described by
                                                //   – – left, top, width, height
                                                //   – startAngle is measured in degrees
                                                //   – – counter clockwise from the right (east)
                                                //   – extentAngle is measured in degrees
                                                //   – – counter clockwise from the startAngle
g.fillArc(30, 20, 50, 80, 30, 60);              // left, top, width, height ,
                                                // startAngle, extentAngle
                                                //   – fills part of the oval (a ‘pie piece’)

Polygon p;                                      // A polygon, any number of sides
p = new Polygon();
p.addPoint(20, 20);                             // To draw a second polygon
p.addPoint(20, 120);                            //   – repeat all lines except
p.addPoint(120, 120);                           //   – “Polygon p;”
g.drawPolygon(p); or g.fillPolygon(p);
```

* *arcWidth* and *arcHeight* are the horizontal and vertical diameter of an oval; the corners of the round rect are quarters of the oval.

b. Constants

```
Color.red, Color.green, Color.blue, Color.yellow, Color.magenta, Color.cyan,
Color.white, Color.black, Color.lightGray, Color.gray, Color.darkGray, Color.pink,
Color.orange
```

```
Font.PLAIN, Font.BOLD, Font.ITALIC, Font.BOLD+Font.ITALIC
"SansSerif" (SansSerif), "Serif"(Serif), "MonoSpaced"(MonoSpaced)
```

Any font on your computer can be used, however the standard fonts on the line above should be available on all computers.

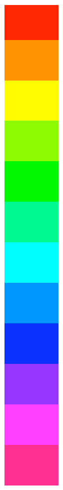
Mixing Colors

Colors are mixed by combining light in each of the primary colors *red*, *green* and *blue*. For each primary color, the value 0 represents none and the value 255 is the maximum available. Using equal amounts of all three primaries will make *black* (0 0 0), or *white* (255 255 255), or *gray*.

The rainbow color strip at the right shows 12 colors and how to mix them. From one color to the next, just one primary is changed. To make a color between one color and the next, set the primary that is changed to an in-between value.

To make dark colors, use about half of the given values. If one-half is too dark, try two-thirds; not dark enough, try one-third. Many of the colors shown are quite bright. A more normal green is 0 170 0, a more normal violet is 130 0 130.

To make pastel colors, change 0s to 205 and 128s to 230 (which is half-way between 205 and 255). To lighten the pastels, use a larger number than 205 for 0 and use the value half-way between your larger number and 255 for 128.



	red	green	blue
← *red*	255	0	0
← orange	255	128	0
← *yellow*	255	255	0
← yellow-green	128	255	0
← *green*	0	255	0
← green blue-green	0	255	128
← *blue-green**	0	255	255
← blue blue-green	0	128	255
← *blue*	0	0	255
← blue-violet	128	0	255
← *violet***	255	0	255
← red-violet	255	0	128

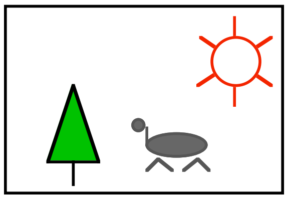
* *bright blue-green* is also called *cyan*.
** *bright violet* is also called *magenta*.

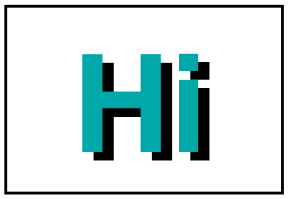
The colors in the color strip whose names are shown in italics are standard colors listed on page 3. Java uses *cyan* and *magenta* rather than *blue-green* and *violet*. The *orange* supplied by Java has more yellow than the orange listed above (quite a bit too much yellow in my opinion).
To make *brown*, use 130 *red*, 65 *green* and 0 *blue*. Use a bit more red for a redder brown.

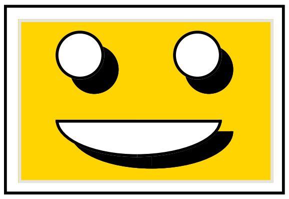
Exercises — 1.1

1. Run the example above on a computer. Your instructor will give you specifics as to how to do this.

2. Change the example to draw something else. Experiment with the additional graphics commands given above. The figures given here are intended to stimulate your imagination.







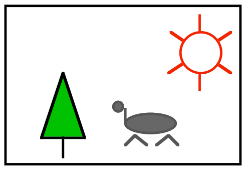
. The frame around ‘Sally & Joe’ in the second example is made by putting a filled white rectangle on top of a filled dark brown rectangle. The word ‘Hi’ is drawn twice, first in black and then in color, a little higher and farther left. The smile is a filled arc starting at 180° and then drawn for 180°.

3. Make a smiley face.

4. Experiment with colors using the command: `g.setColor(new Color(red, green, blue))`. Try various values for red, green and blue. Use the section on mixing colors above for hints as to what values to use.

5. Experiment with the graphics commands and constants to become familiar with them.

1.2 Organizing a Program with Private Methods



The programs below each produce the same result — a picture similar to the illustration. In the right hand version the commands that create the three objects in the picture have been grouped together and named. Named groups of commands are called *methods*.

```
import java.awt.*;

public class Landscape extends
                   java.applet.Applet
   {
   public void paint(Graphics g)
      {
      g.setColor(Color.red);
      g.drawOval(200, 30, 60, 60);
      g.drawLine(190, 40, 203, 50);
      g.drawLine(270, 40, 257, 50);
      g.drawLine(190, 80, 203, 70);
      g.drawLine(270, 80, 257, 70);
      g.drawLine(230, 15, 230, 30);
      g.drawLine(230, 105, 230, 90);

      g.setColor(Color.darkGray);
      g.fillOval(150, 150, 10, 10);
      g.fillOval(160, 160, 70, 20);
      g.drawLine(160, 160, 160, 170);
      g.drawLine(170, 177, 160, 188);
      g.drawLine(170, 177, 180, 188);
      g.drawLine(220, 177, 210, 188);
      g.drawLine(220, 177, 230, 188);

      Polygon triangle;
      triangle = new Polygon();
      triangle.addPoint(70, 90);
      triangle.addPoint(50, 170);
      triangle.addPoint(90, 170);
      g.setColor(Color.green);
      g.fillPolygon(triangle);
      g.setColor(Color.black);
      g.drawPolygon(triangle);
      g.drawLine(70, 170, 70, 190);
      }
   }
```

```
import java.awt.*;

public class Landscape extends
                   java.applet.Applet
   {
   public void paint(Graphics g)
      {
      paintSun(g);
      paintBeast(g);
      paintTree(g);
      }

   private void paintSun(Graphics g)
      {
      g.setColor(Color.red);
      g.drawOval(200, 30, 60, 60);
      g.drawLine(190, 40, 203, 50);
      g.drawLine(270, 40, 257, 50);
      g.drawLine(190, 80, 203, 70);
      g.drawLine(270, 80, 257, 70);
      g.drawLine(230, 15, 230, 30);
      g.drawLine(230, 105, 230, 90);
      }

   private void paintBeast(Graphics g)
      {
      g.setColor(Color.darkGray);
      g.fillOval(150, 150, 10, 10);
      g.fillOval(160, 160, 70, 20);
      g.drawLine(160, 160, 160, 170);
      g.drawLine(170, 177, 160, 188);
      g.drawLine(170, 177, 180, 188);
      g.drawLine(220, 177, 210, 188);
      g.drawLine(220, 177, 230, 188);
      }

   private void paintTree(Graphics g)
      {
      Polygon triangle;
      triangle = new Polygon();
      triangle.addPoint(70, 90);
      triangle.addPoint(50, 170);
      triangle.addPoint(90, 170);
      g.setColor(Color.green);
      g.fillPolygon(triangle);
      g.setColor(Color.black);
      g.drawPolygon(triangle);
      g.drawLine(70, 170, 70, 190);
      }
   }
```

The program on the right is somewhat longer but it has the advantage of better organization. On the left, the *paint* method has 28 lines; on the right, the *paint* method has 3 lines with helpful names. The parts of the program and their purposes stand out clearly in the second program. Also, shorter methods tend to make editing a program easier.

Method names

Methods are customarily given names which start with a lower case letter. In the interest of program readability, you should follow this custom. The design of the Java language makes it impossible for the names of methods to contain spaces. Multiword method names are made more readable by capitalizing the first letter of each word after the first. While these capitalization rules are not enforced by the computer, you should follow them as though they were.

The names *paintSun*, *paintBeast* and *paintTree* differ from the name *paint* in that *paint* was a name already known to Java whereas the others were created for this particular program. Pre-existing Java instructions use the method *paint* to create the picture on the screen. No pre-existing Java instructions would use the methods *paintSun*, *paintBeast* and *paintTree*. To make absolutely sure that no pre-existing instructions would unexpectedly use the newly named methods, we have declared them to be *private*. Private ensures a method will only be available to methods within the class containing the private method.

In the *paint* method, the new methods are used as commands.

```
public void paint(Graphics g)
    {
    paintSun(g);
    paintBeast(g);
    paintTree(g);
    }
```

The statement *paintSun(g);* commands the method *paintSun*, to paint the sun. The parentheses contain the names of any required objects a method will use. *paintSun* requires a graphics object, and the graphics object *g* is placed in the parentheses.

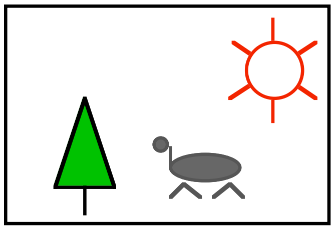
Exercises — 1.2

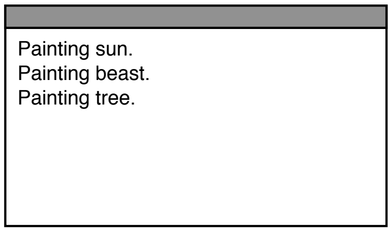
Write several programs that lend themselves to the use of *private* methods for breaking up the *paint* method. Your programs should use 3 or 4 private methods. Rewrite the programs from exercises 1.1 using *private* methods.

1.3 The Console Window

Often there is a need to report information during the execution of a program. The console window provides a simple method to accomplish this. For example, if we wanted the program which draws the sun, beast, and tree to report when it was starting to draw each of the items, we might put 3 additional lines into the program:

```
import java.awt.*;

public class Landscape extends java.applet.Applet
   {
   public void paint(Graphics g)
      {
      paintSun(g);
      paintBeast(g);
      paintTree(g);
      }

   private void paintSun(Graphics g)
      {
      System.out.println("Painting sun.");        // added line
      g.setColor(Color.red);
      ...                               // in this case, ellipses (…) mean "and so on as before."
      }

   private void paintBeast(Graphics g)
      {
      System.out.println("Painting beast.");      // added line
      g.setColor(Color.darkGray);
      ...
      }

   private void paintTree(Graphics g)
      {
      System.out.println("Painting tree.");       // added line
      Polygon triangle;
      ...
      }
   }
```





```
Painting sun.
Painting beast.
Painting tree.
```

The picture is unchanged by these additional lines. The text-only, console window should now contain the lines:

```
Painting sun.
Painting beast.
Painting tree.
```

Each line is created immediately before the corresponding graphic is drawn. However, the speed of the computer eliminates any chance of seeing this.

When programs get longer, lines printed in the console window are an effective way of finding out which commands are being performed and in which order. As you might imagine, this is most commonly done when a program isn't working correctly.

Console window experiment

As an experiment, drag some other window on top of part or all of your image. Now uncover your complete image. On some computer systems, the lines above will appear an additional time in the console window. When using these systems, each time a previously covered portion of the picture is uncovered, the instructions in the *paint* method of the program are used to draw the uncovered portion of the image.

Exercise — 1.3

Start with a previous program and add commands that place text in the console window. It is not necessary to do this extensively, just enough that you understand the effect of placing such commands in various places within the methods of a program.