Key Points

- **What is a class?** A class is a group of attributes (data) and methods. If class C represents a program, then must be declared as *public class C* in a file *C.java*, and must have a *main* method, which is where execution starts when we type *java C* on command line.
- **What is a method?** A method is a named piece of executable code. When method is executed, statements inside it are executed in order (from top to bottom of the method text).
- **What is a variable?** A variable is a named piece of computer memory. Once declared, variables can be updated and read. We can store integers in an *int* variable, rational numbers in a *double* variable, and strings – sequences of characters – in a *String* variable.

Part 4: GRAPHICS

Java 2 has rich set of graphics classes, providing drawing and painting operations, using variety of geometrical shapes and a wide range of colours and patterns. Text and existing images such as photographs can be incorporated.

- Using graphics
- Writing applets
- Rectangles and ellipses
- Introducing colour.
- Painting shapes.
- Writing text.
- Reading input.

Using Graphics

Two ways to run Java programs: as applications, and as applets. An applet is executed by Web browser such as Netscape or Internet Explorer when it finds a link to program in an HTML page it is displaying.

An applet can display graphics – web browser will set aside display area which applets use to display any kind of information. Then browser will start executing applet.

So to display graphics we do:

1. Write an applet that displays graphics, and compile it.
2. Write a Web page that contains a link to the compiled applet.
3. Use *appletviewer* browser to view page.

Using Graphics

We will write an applet that draws or paints ('renders') graphics in its display area, and write Web page that consists of link to compiled applet.

A Web page consists of a text file that uses HTML notation. The pages we use will all be of form:

```
<APPLET CODE="Prog.class" WIDTH=300 HEIGHT=300>
</APPLET>
```

This contains link to file *Prog.class*. Contains compiled Java applet. Will change this to whatever file contains compiled applet we want to run.

Writing an Applet

The applet itself consists of a Java class. It differs from classes we have seen previously in that it contains at least one method provided for window manager to use. The window manager is software that maintains windows on screen.

One of these methods is *paint*. Called first time display area appears on screen and when it is refreshed.

Task of *paint* method is to render contents of display area (i.e. draw lines, or paint areas, or write text).

Writing an Applet

Write a paint method that will render a simple shape, a triangle. To do this, make use of a *Graphics2D* object. This is like *System.out*, except it is connected to applet's high resolution display area, not crude DOS window, and is much more versatile.

Graphics2D has method called *drawString* to display text, another method *draw* that will draw a rectangle or circle. Also, *Graphics2D* object has methods to tell it to start using particular colour (or pattern) for painting shapes or text it is given, etc.

Writing an Applet

If $g2$ is a *Graphics2D* object connected to applet's display area, and $line1$ is a line, then statement

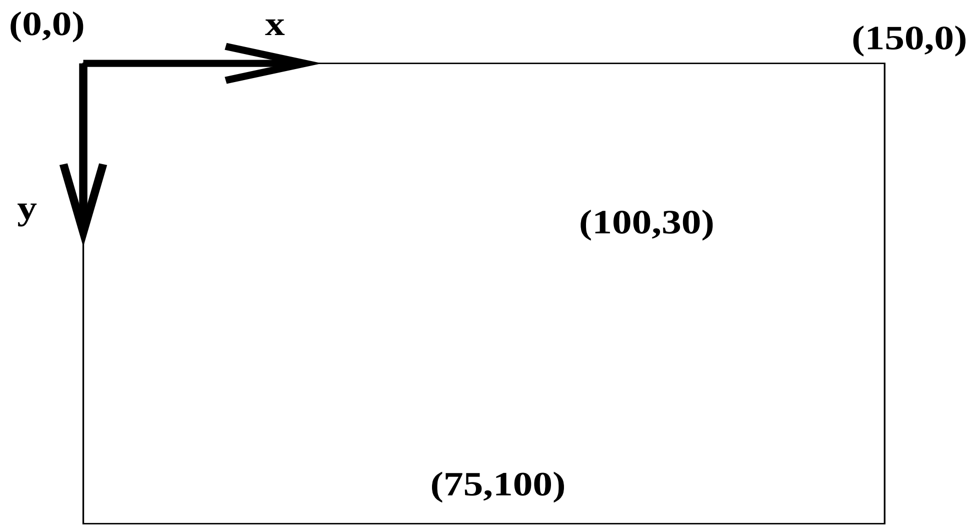
```
g2.draw(line1);
```

will display $line1$ in display area.

Various geometrical shapes can be drawn and painted using a *Graphics2D* object: lines, rectangles and ellipses (including circles). Each is a Java object. For example, line is an object of type *Line2D.Double*.

To create a line, specify its coordinates. Positions in the display area are given x and y coordinates.

X and Y Coordinates



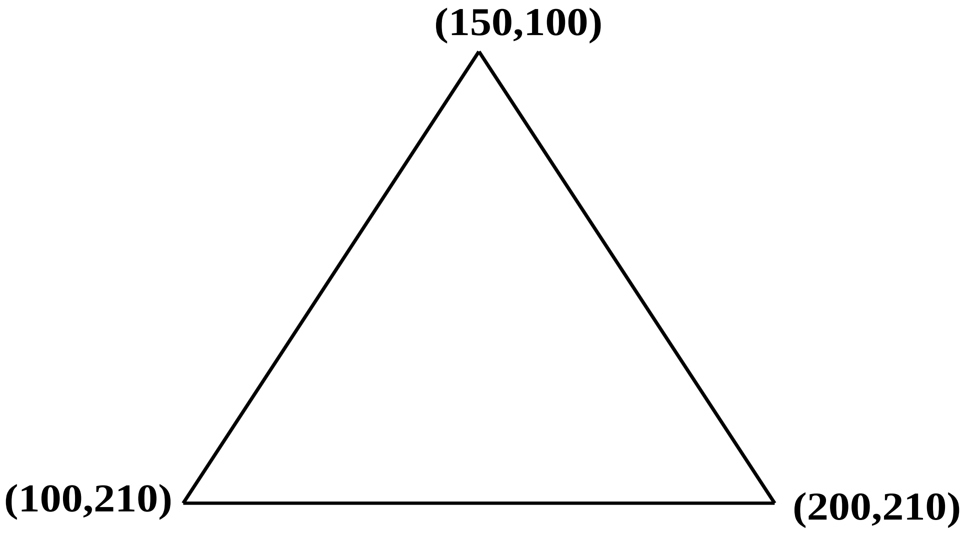
Writing an Applet

To create line object, use *Line2D.Double* constructor that has coordinates of the line's endpoints as parameters:

```
Line2D.Double(x0,y0,x1,y1)
```

Creates an object that represents the line from $(x0, y0)$ to $(x1, y1)$.

Triangle Example



Creating a Triangle

Can use following statements.

```
Line2D.Double line1 =
   new Line2D.Double(150,100,200,210);
g2.draw(line1);

Line2D.Double line2 =
   new Line2D.Double(200,210,100,210);
g2.draw(line2);

Line2D.Double line3 =
   new Line2D.Double(100,210,150,100);
g2.draw(line3);
```

To write an applet that just draws this triangle on screen, write *paint* method that contains these six statements. The $Graphics2D$ object $g2$ is passed to *paint* as a parameter, by window manager that calls *paint*. Heading of *paint* will look like this:


```
public void paint(Graphics g)
```

Need to cast $g2$ to $Graphics2D$:

```
Graphics2D g2 = (Graphics2D) g;
```

This says: create $Graphics2D$ variable, $g2$, and assign it the object g which *is* a $Graphics2D$ object. The cast `(Graphics2D)` is our promise to compiler that the parameter is really a $Graphics2D$ object.

An exception would occur if g isn't of this type at runtime.

EXAMPLE 4.1

```
import java.applet.*;
import java.awt.*;
import java.awt.geom.*;

/* Applet that displays a triangle. */

public class Triangle extends Applet
{  public void paint(Graphics g)
   {  Graphics2D g2 = (Graphics2D) g;

      Line2D.Double line1 =
         new Line2D.Double(150,100,200,210);
      g2.draw(line1);

      Line2D.Double line2 =
         new Line2D.Double(200,210,100,210);
```



```
      g2.draw(line2);

      Line2D.Double line3 =
         new Line2D.Double(100,210,150,100);
      g2.draw(line3);
   }
}
```

The `import` statements make library classes such as *Applet* and *Line2D.Double* available for use in the program.

Triangle Applet

Instead of writing

```
Line2D.Double line1 =
   new Line2D.Double(150,100,200,210);
g2.draw(line1);
```

we could just write

```
g2.draw(new Line2D.Double(150,100,200,210));
```

Both versions are equally good.

To run this program, the following text file (that uses HTML notation) can be used. It contains link to file containing compiled *Triangle* class:

```
<APPLET CODE="Triangle.class"
        WIDTH=300
        HEIGHT=300>
</APPLET>
```


Assume file is called *Triangle.html*. Then invoke appletviewer on it by the command:

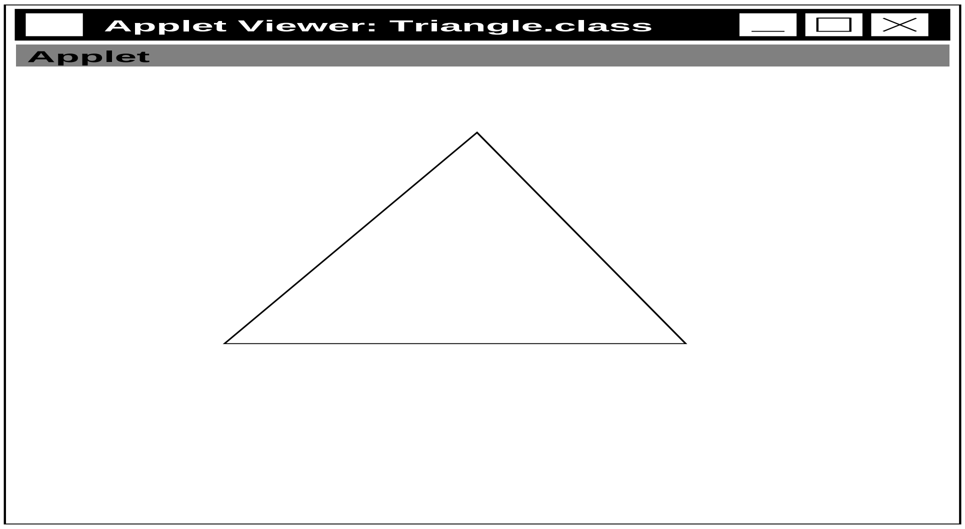
```
appletviewer Triangle.html
```

Java library consists of a number of packages. The first line of example 1

```
import java.applet.*;
```

tells compiler that package called `java.applet` is being used. The * tells compiler to expect any class in package to be used. In fact, only `Applet` is used. If had not imported `java.applet`, would have to use full name `java.applet.Applet`. `java.awt` package contains *Graphics*, *Graphics2D* classes. `java.awt.geom` provides shapes such as `Line2D.Double`.

Screen Dump of Triangle Applet



Rectangles and Ellipses

Java 2 includes various useful geometrical shapes. One of these is rectangles, in class *Rectangle2D.Double*. A rectangle can be created using following constructor.

```
Rectangle2D.Double(x,y,w,h)
```

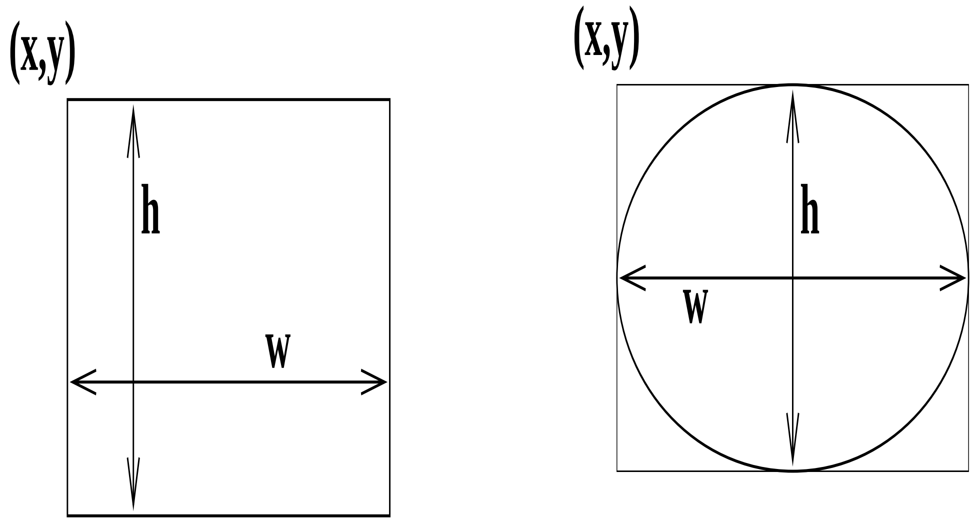
Creates object that represents rectangle with horizontal and vertical sides. Top-left corner has coordinates (x, y). Width is w, height is h.

A related shape is ellipse, given by class *Ellipse2D.Double*. Created by constructor

```
Ellipse2D.Double(x,y,w,h)
```

Creates ellipse with enclosing rectangle described by x, y, w and h. Top-left corner has coordinates (x, y). Width is w, height is h.

Rectangles and Ellipses

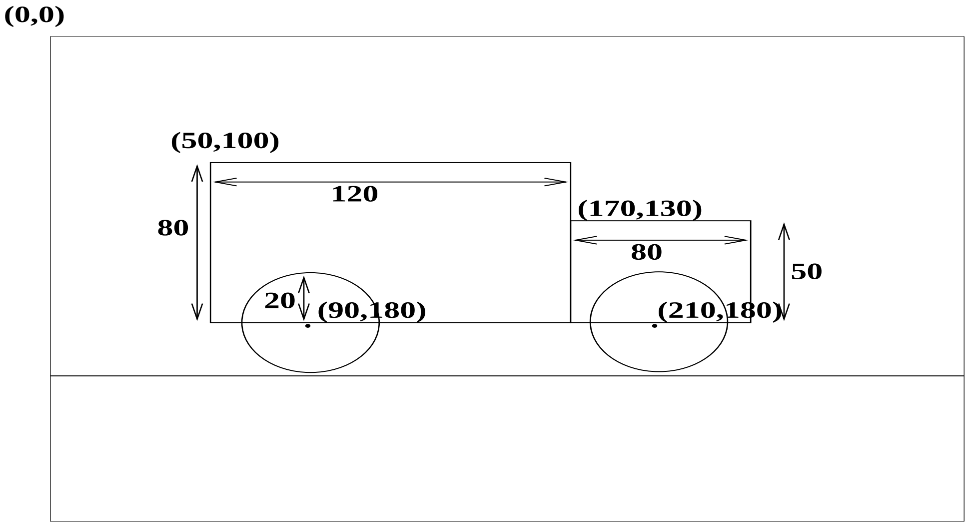


Circles

Can make a circle by creating ellipse whose width is equal to its height. If centre of circle is at (xc, yc) and radius is r, then it is ellipse with containing rectangle corner at $(xc - r, yc - r)$, width $2 * r$, height $2 * r$.

In next example use lines, rectangles and ellipses to construct simple drawing of a van. Must carefully work out positions of different components.

Components of Van Drawing

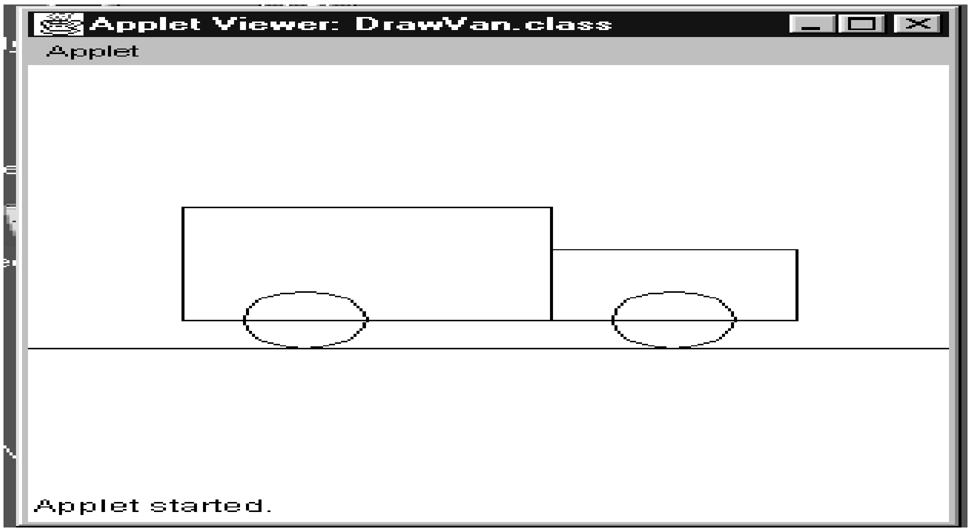


EXAMPLE 4.2

```
import java.applet.*;
import java.awt.*;
import java.awt.geom.*;

public class DrawVan extends Applet
{  public void paint(Graphics g)
   {  Graphics2D g2 = (Graphics2D) g;
      Rectangle2D.Double back =
         new Rectangle2D.Double(50,100,120,80);
      g2.draw(back);

      Rectangle2D.Double front =
         new Rectangle2D.Double(170,130,80,50);
      g2.draw(front);

      Ellipse2D.Double backWheel =
```

```
        new Ellipse2D.Double(70,160,40,40);
      g2.draw(backWheel);

      Ellipse2D.Double frontWheel =
        new Ellipse2D.Double(190,160,40,40);
      g2.draw(frontWheel);

      Line2D.Double baseLine =
        new Line2D.Double(0,200,300,200);
      g2.draw(baseLine);
   }
}
```


Screen shot of image



Introducing Colour

A *Graphics2D* object will normally draw in black. It also has a *setColor*(*Color c*) method to change drawing colour to *c*.

To set *Graphics2D* object *g2* to start drawing in red, insert statement:

```
g2.setColor(Color.red);
```

After this, until *setColor* is invoked on *g2* again, everything *g2* displays will be coloured red.

Can define own colours by specifying amounts of three basic colours: red, green and blue: `Color col = new Color(r, g, b);`

Values r, g and b are floating point numbers in range 0 to 1, of type float.

```
Color c = new Color(1.0f, 1.0f, 0.0f);  /* yellow */
```


Painting Shapes

Can use a *Graphics2D* object to colour inside a shape such as a rectangle or an ellipse that encloses an area.

The method to use is called *fill*:

```
g2.fill(s)
```

Use *Graphics2D* object *g2* to display shape *s* filled with current colour assigned to *g2*.

For example, following statements create small circle and display it, painted yellow:

```
Ellipse2D.Double sun =
   new Ellipse2D.Double(140,50,20,20);
g2.setColor(Color.yellow);
g2.fill(sun);
```

Draw van, painted red. The wheels are white, and have dark grey tyres. The base line has been replaced by a pale blue background.

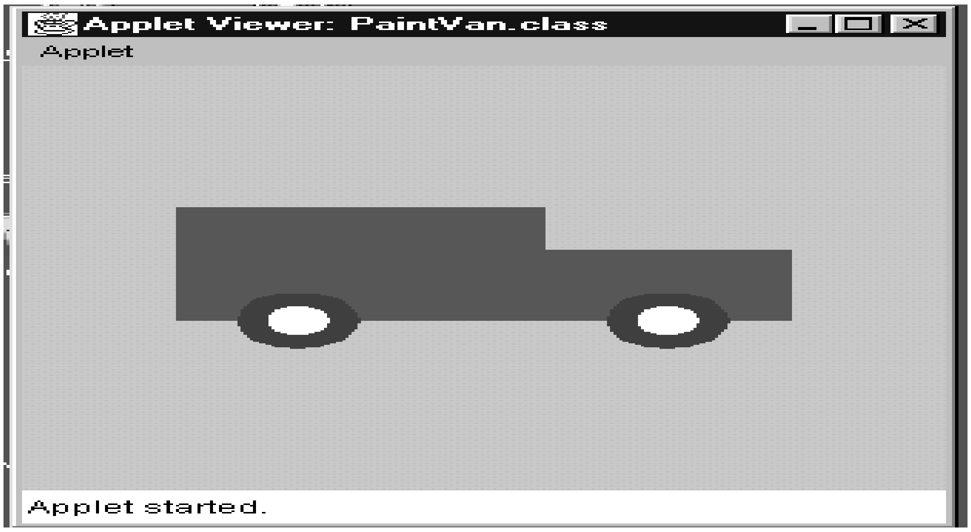
EXAMPLE 4.3

```
import java.applet.*;
import java.awt.*;
import java.awt.geom.*;

public class PaintVan extends Applet
{  public void paint(Graphics g)
   {  Graphics2D g2 = (Graphics2D) g;

      /* Paint the background. */
      Color paleBlue =
         new Color(0.75f, 0.75f, 1.0f);
      g2.setColor(paleBlue);
      g2.fill(new Rectangle2D.Double(0,0,300,300));
```



```
      /* Paint the body of the van. */
      g2.setColor(Color.red);
      g2.fill(new Rectangle2D.Double(50,100,120,80));
      g2.fill(new Rectangle2D.Double(170,130,80,50));

      /* Paint the back wheel. */
      g2.setColor(Color.darkGray);
      g2.fill(new Ellipse2D.Double(70,160,40,40));
      g2.setColor(Color.white);
      g2.fill(new Ellipse2D.Double(80,170,20,20));

      /* Paint the front wheel. */
      g2.setColor(Color.darkGray);
      g2.fill(new Ellipse2D.Double(190,160,40,40));
      g2.setColor(Color.white);
      g2.fill(new Ellipse2D.Double(200,170,20,20));
   }
}
```

Painted Van



Writing Text

A *Graphics2D* object can be used to display text on screen. Can set text *font*, created by:

```
Font(family, style, size)
```

family is a string which is name of a set of related fonts. Eg. `"Times Roman"` and `"Courier"` which is used for the text of programs.

Following logical names can be used for font families.

```
Serif
SansSerif
Monospaced
Dialog
DialogueInput
```

style has one of following values, indicating if italics or boldface should be used:

```
Font.PLAIN
Font.ITALIC
Font.BOLD
Font.ITALIC + Font.BOLD.
```

size is point size of font. This is height of characters in units of 1/72 inches.

Once you have a *Font* object f, can set font of a *Graphics2D* to be f:

```
g2.setFont(f)
```


Writing Text

To output string to particular point in display area use:

```
g2.drawString(text, x, y)
```

Following example illustrates use of logical font families listed above. The *Graphics2D* object is assigned each one in turn, and name of family is output. In each case style is FONT.PLAIN, and point size is 30.

EXAMPLE 4.4

```
import java.applet.*;
import java.awt.*;

public class Fonts extends Applet
{ private final int SIZE = 30;

  public void paint(Graphics g)
  { Graphics2D g2 = (Graphics2D) g;
    Font font1 = new Font("Serif", Font.PLAIN, SIZE);
    g2.setFont(font1);
    g2.drawString("Serif", 50, 50);

    Font font2 = new Font("SansSerif", Font.PLAIN, SIZE);
    g2.setFont(font2);
    g2.drawString("SansSerif", 50, 100);

    Font font3 =
```



```
      new Font("Monospaced", Font.PLAIN, SIZE);
    g2.setFont(font3);
    g2.drawString("Monospaced", 50, 150);

    Font font4 = new Font("Dialog", Font.PLAIN, SIZE);
    g2.setFont(font4);
    g2.drawString("Dialog", 50, 200);

    Font font5 =
      new Font("DialogInput", Font.PLAIN, SIZE);
    g2.setFont(font5);
    g2.drawString("DialogInput", 50, 250);
  }
}
```

Creating a Van Object

Sometimes makes good sense to turn a subject for drawing or painting, such as van in example 2 and 3, into object which has a method that can be used to render it. (Increases flexibility of use).

Following example contains definition of *Van* class and an applet that creates and draws a number of vans.

A *Van* object contains location of a van, and its size. Create vans with constructor:

```
Van(x, y, w)
```

Creates *Van* object whose top-left corner is (x, y), and whose width is w.

A Van object has one method.

```
v.paintVan(g2)
```

Paint picture of van v using $Graphics2D$ object $g2$.

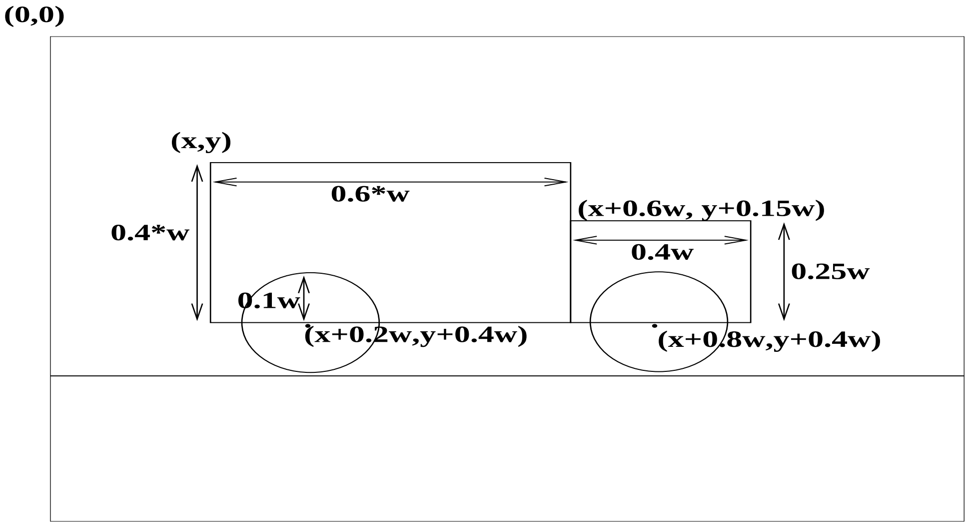
Change statements used to render van so they work correctly for any top-left starting point (x, y) and any width w.

Example 6 contains complete *Van* class. In addition *paintVan* method paints text, in white, on side of van:

```
Java
Delivers
```

The second class is applet class *ManyVans*. This creates series of vans and paints each one in its display area.

Generalised Van Layout



EXAMPLE 4.6

```
import java.awt.*;
import java.awt.geom.*;

public class Van
{ private double x, y;  // Top-left corner of van.
  private double w;     // Width of van.

  final Color COLOR1 = Color.red;        // Colour for body.
  final Color COLOR2 = Color.darkGray;   // Colour of tyres.
  final Color COLOR3 = Color.lightGray;  // Colour of wheels.

  public Van(double x0, double y0, double w0)
  { x = x0; y = y0; w = w0; }

  public void paintVan(Graphics2D g2)
  { /* Paint body of van. */
```

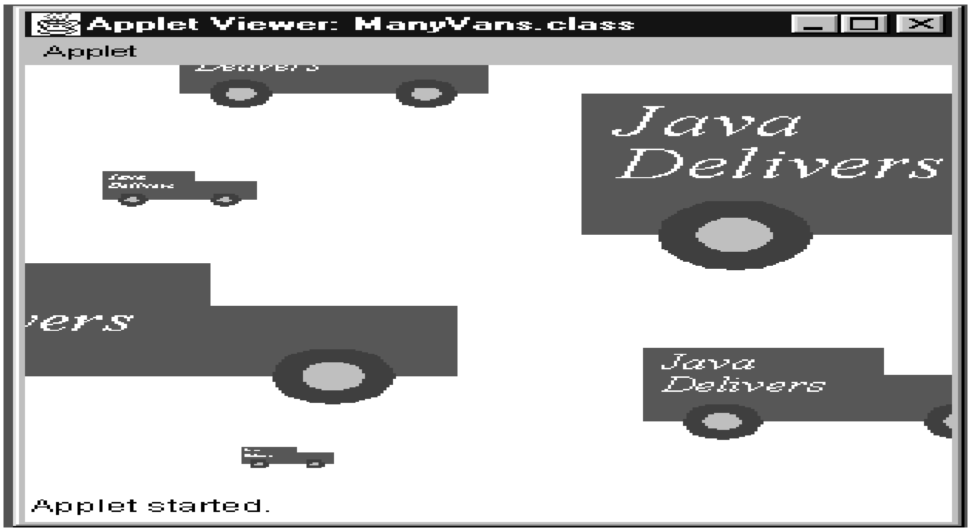
```
    g2.setColor(COLOR1);
    Rectangle2D.Double back =
      new Rectangle2D.Double(x, y, w*0.6, w*0.4);
    g2.fill(back);
    Rectangle2D.Double front =
      new Rectangle2D.Double(x+w*0.6, y+w*0.15, w*0.4, w*0.25);
    g2.fill(front);
    /* Paint tyres. */
    g2.setColor(COLOR2);
    Ellipse2D.Double tyre1 =
      new Ellipse2D.Double(x+w*0.1, y+w*0.3, w*0.2, w*0.2);
    g2.fill(tyre1);
    Ellipse2D.Double tyre2 =
      new Ellipse2D.Double(x+w*0.7, y+w*0.3, w*0.2, w*0.2);
    g2.fill(tyre2);

    /* Paint wheels. */
    g2.setColor(COLOR3);
```



```
    Ellipse2D.Double wheel1 =
      new Ellipse2D.Double(x+w*0.15, y+w*0.35, w*0.1, w*0.1);
    g2.fill(wheel1);
    Ellipse2D.Double wheel2 =
      new Ellipse2D.Double(x+w*0.75, y+w*0.35, w*0.1, w*0.1);
    g2.fill(wheel2);

    /* Paint logo. */
    Font f = new Font("Serif", Font.ITALIC, (int)(w/8));
    g2.setFont(f);
    g2.setColor(Color.white);
    g2.drawString("Java", (int)(x+w*0.05), (int)(y+w*0.12));
    g2.drawString("Delivers", (int)(x+w*0.05), (int)(y+w*0.24));
  }
}
```

```
import java.applet.*;
import java.awt.*;

/* Display vans. */
public class ManyVans extends Applet
{  public void paint(Graphics g)
   {  Graphics2D g2 = (Graphics2D) g;

      new Van(200,200,130).paintVan(g2);
      new Van(25,75,50).paintVan(g2);
      new Van(-60,140,200).paintVan(g2);
      new Van(180,20,250).paintVan(g2);
      new Van(50,-20,100).paintVan(g2);
      new Van(70,270,30).paintVan(g2);
   }
}
```


Output of ManyVans



Reading Input

In order to enter data for an applet, can use a *dialog.* This is a secondary window which appears temporarily to allow program and user to exchange messages.

Simple form of dialog used here is created by *JOptionPane* class (in the `swing` package).

It contains message from program to user, and has a slot where user can type reply.

For example, following statement causes dialog to appear, displaying message 'Enter your name'. It waits for user to type string of characters and click OK (or type Return). Finally string will be assigned to new variable *reply* and dialog will disappear.

```
String reply =
   JOptionPane.showInputDialog("Enter your name: ");
```


Basic JOptionPane dialog

Dialogs

Same technique can be used to read numbers: user's input is read as a string, st, then converted to a number using $Integer.parseInt(st)$ or $Double.parseDouble(st)$.

In example 4.7 the program paints a picture of a van. It presents dialog to user asking how big van should be.

We want dialog to appear just once at start – the

```
public void init()
```

method of applets is suitable, since window manager calls it just once, at beginning of applet run.

EXAMPLE 4.7

```
import java.applet.*;
import java.awt.*;
import javax.swing.*;
   // Needed for the JOptionPane class.

/* Read size and display van. */

public class SetSize extends Applet
{ private double vanSize;

  public void init()
  { String input =
      JOptionPane.showInputDialog("Enter width of van: ");
    vanSize = Double.parseDouble(input);
  }
```

```
  public void paint(Graphics g)
  { Graphics2D g2 = (Graphics2D) g;

    double x = (300 - vanSize)/2;
    double y = (300 - vanSize/2)/2;
    new Van(x,y,vanSize).paintVan(g2);
  }
}
```


Display sizes

Applets can find out current size of display area – *getWidth*() will return current width of display area, *getHeight*() will return current height. If replace statements 2 to 4 of *paint* by

```
double x = (getWidth() - vanSize)/2;
double y = (getHeight() - vanSize/2)/2;
new Van(x,y,vanSize).paintVan(g2);
```

Window manager will call paint if window resized, and this repaints van centered in the new display area.