Scarborough College
University of Toronto
Fall 1997

CSC A06F: Introduction to Computer Programming
Lecture notes 7:
Applets and Graphics

Reading:

- Lewis and Loftus, chapter 7.

A simple example

```
import java.awt.*;

public class GraphicsTest {

   public static void main(String args[]) {
      TestFrame f = new TestFrame();  // create a frame
      f.init();       // initialize it
      f.show();       // show it -- this calls f.paint()
   }
}

class TestFrame extends Frame {

   public void init() {
      setSize(400,400);  // you MUST set the size
   }

   // draw some ovals
   //   NOTE: paint() is always called automatically
   //         by other methods
   public void paint (Graphics page) {
      page.drawOval(50,50, 300,300);
      page.fillOval(125,125,50,50);
      page.fillOval(225,125,50,50);
   }
}
```

Creating graphics

Where are you going to draw your graphics? The normal 'text' window can only do text.

There are two ways to get a drawing surface:

- Use a **`Frame`** (called from your own application)
- Use an **`Applet`** (called from a web browser)

Frames and applets are objects. A frame is a window that is displayed on your screen; an applet is displayed on a web page.

Frames and most graphics commands are defined in the package `java.awt`: *Abstract Windowing Toolkit* .

The `Applet` class is in the package `java.applet`

An Applet

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

public class GraphicsTest_Applet extends Applet {

   public void start() {
      setSize(400,400);  // you have to set the size
   }

   // draw some ovals
   public void paint (Graphics page) {
      page.drawOval(50,50, 300,300);
      page.fillOval(125,125,50,50);
      page.fillOval(225,125,50,50);
   }
}
```

Frames vs. applets

An applet is a *mini-application* . It can't run on its own, but it can be plugged into a web browser that is already running.

An applet has a **standard interface** (i.e., a set of service methods) that the web browser knows how to use.

So, you can send someone an applet over the web, and their web browser can run it.

You use a frame when you want to create your own complete application.

You decide on the interface. You decide how the application should interract with the frame.

Both frames and applets are very complex objects.

To create your own, *you must* ***extend*** *already written code.*

Aside: Inheritance

When you define a new class, you can **extend** the definition of an already defined one.

The statement:

```
class TestFrame extends Frame { ... }
```

says that a `TestFrame` object is everything that a `Frame` object is, *plus* whatever is added here.

I.e., the `TestFrame` **inherits** the properties of the `Frame`.

- You *add* new instance variables and methods.
- You can also *override* , or replace, methods.

The drawing surface

Every frame and applet automatically has a **drawing surface** .

A drawing surface is an object of the class `Graphics`.

The `Graphics` class contains all the methods for drawing things on the surface.

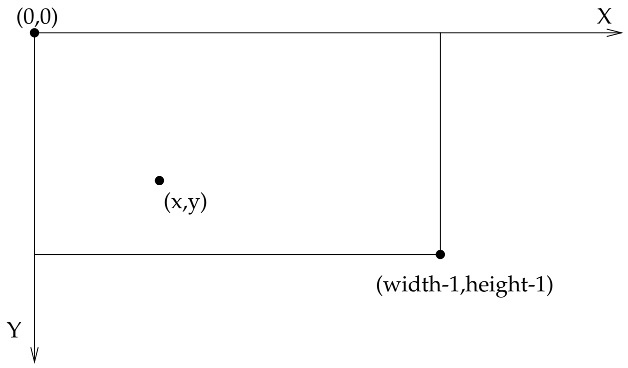
The `getGraphics` method of the frame or applet returns the drawing surface object.

But normally you don't need to use this method because the drawing surface is passed to the `paint` method.

- For applets, `paint` is called automatically by the browser when it needs to display the applet.
- For frames, `paint` is called when you call the frame's `show` method.

The coordinate system

Every drawing surface has its own coordinate system.



An (x,y) coordinate represents a single pixel on the surface.

x and y must be integers.

A drawing surface has a particular size (height and width).

Anything you draw outside this area won't be visible.

Drawing in colour

The **foreground colour** specifies which colour shapes and text are drawn in. (Default is black.)

You can change the foreground color using the `setColor` method of the `Graphics` class.

```
page.setColor (Color.red);
page.drawRect (10, 10, 50, 50);
```

Colours are objects. They are defined by the `Color` class.

Several predefined `static` colour objects are declared in the `color` class: (see page 256 of L&L).

```
Color.black, Color.red, Color.green,
Color.gray, Color.magenta, ...
```


Creating your own colours

You can create your own colours by instantiating the `Color` class. E.g.,

```
Color mycolor = new Color (34, 183, 91);
page.setColor (mycolor);
```

This defines a new colour object `mycolor`.

The numbers 34, 183, 91 refer to the amount of red, green, and blue in this colour.

They can range from 0 (none) to 255 (all). So the **RGB** values of black are (0,0,0), and of white are (255,255,255).

How many colours are there?

(Most computers can't display them all at once.)

Drawing shapes

You can draw:

- Lines
- Rectangles (including squares)
- Ovals (including circles)
- Arcs
- Polygons
- PolyLines

Most shapes can be drawn as an outline, or filled in.

Examples of shapes

- Rectangles: specify top left corner, and size.

```
page.drawRect (int x, int y, int width, int heigth)
page.fillRect (int x, int y, int width, int heigth)
```

- Polygons: specify an array of x coordinates and y coordinates.

```
page.drawPolygon (int xpoints[], int ypoints, int numpoints)
page.fillPolygon (int xpoints[], int ypoints, int numpoints)
```

Experiment with these and the others in section 7.3 of L&L.

Text and fonts

You can also draw text at specific coordinates:

```
page.drawString (String text, int x, int y)
```

You can specify the font of the text.

The **font** is the style of lettering used. Fonts are objects, so to get one you have to instantiate the `Font` class.

- You specify the name of the font, its style, and its size. E.g.,

```
Font myfont = new Font ("Helvetica", Font.BOLD, 24);
page.setFont (myfont);
```

- Font names are different for every computer (see page 275 of L&L). Some are `"TimesRoman"`, `"Courier"`, `"Palatino"`.
- Style is `PLAIN, BOLD,` or ITALIC.
- Size refers to point size. A larger size produces a larger font.