The UBC Java graphics tutorial—a still picture

We shall begin by looking at a very simple applet which displays a square in the middle of a canvas.

First a few words on some very practical matters and the basic mechanism of Java.

Java is the name of a computer language, and also the name of a **virtual machine** which interprets this language. A copy of the virtual machine sits inside the browser on which someone will see your program run. Your program, in a form readable by the Java virtual machine, is loaded over the Internet. The steps in making this program are these:

- create the program in one or more **Java source files**, which must have the extension `.java`;
- compile these files into **class files** which contain instructions in the machine language of the Java virtual machine, and in particular to build a main class file which I shall call the **applet file**;
- make an `.html` file which your browser reads, and which runs your applet file.

We shall examine these steps here, working with a very simple graphics applet called `psSimpleApplet`. The last step is by all means the simplest, and the first we look at. Here is an `.html` file which will run it inside a browser:

```
<html>
</h2>A simple graphics applet</h2>

<applet codebase = ".." code="psapplets/psSimpleApplet.class" width=500 height=300>
</applet>

</html>
```

The key lines here are the ones in the middle.

- Every applet must be contained in between two tags `<applet ... width = ... height = ... ></applet>`. In our context, the **codebase** must also be specified.
- The applet file you are running is `psSimpleApplet.class`, and the Java file it was compiled from is `psSimpleApplet.java`. It is a good idea to include the term 'Applet' in the names of files that construct applets, although certainly not necessary. Applets are rather special Java programs, and files which hold applets ought to be immediately recognizable by their names.
- The term `codebase` tells the browser to interpret all file names relative to a particular directory I'll call the **base directory**. The applet we are running is in the file `psSimpleApplet.class`, which is contained in the directory `psapplets`, which is a subdirectory of the base directory. The `.html` file which runs it is also in `psapplets`. This applet uses programs from the packages `psplot` and `real`, which are in subdirectories of the base directory with the same names as the packages. There are several ways to run an applet you have made, and because this applet uses home-made packages the one here is more complicated than some.
- In general the `.html` file which runs an applet is allowed to have several parameters assigned in it. This is somewhat analogous to command line arguments in an ordinary program. The two arguments `width` and `height` *must* be assigned. The unit of length is a pixel.

Once you have written your `.java` file, say `x.java`, it is compiled with the command

```
javac x.java
```

One of the technical things about Java that often causes trouble at the beginning of a Java programming career is that the operating system has to know where to look for files specific to Java. In particular, the Java compiler `javac` has to be in your command path. You may have to locate the command `javac` on your machine and then add it to that path. How to do this varies from one machine to the other. Another technical item is that the compiler will have to find the `psplot` and `real` packages—it needs to be told which directory they are in. You do this by setting the environment variable `CLASSPATH`, so it includes the directory in which the subdirectories `psplot`, `real`, and your applet directory are located.

Another technical matter: when you are developing your program, errors are likely if not inevitable. Browsers have one feature that makes the development task inside them impossible—for security reasons, they only load an applet once in a session! Therefore, you should only use a browser to run your applet when it is finally working. During development, you can use a tool called `appletviewer` that comes with the Java Developer's Kit. You run `appletviewer` on the `.html` file the browser would read. For example

```
appletviewer ps.html
```

This program ignores text in an `.html` file, and displays all applets it finds there, each in its own window. `Appletviewer` requires yet the environment variable `JAVA_HOME` to be set. This is the directory in which the basic Java files are located.

On the undergraduate UNIX system in the UBC Mathematics Department, you can deal with these matters by putting the lines

```
setenv CLASSPATH $CLASSPATH\:<base directory>
setenv JAVA_HOME /math/local/java/jdk1.0.2
setenv PATH $PATH\:/math/local/java/jdk1.0.2/bin
```

at the end of the `.cshrc` file in your home directory, where `<base directory>` is the full name of the base directory. You can find out what that full name is by typing `pwd` while you are in it.

So now we come to the hard part, which is actually writing the program file. As I have already mentioned, I expect you to make `.java` programs by copying the examples in `psapplets`. When the examples don't tell you enough, you will probably have to learn more about general features of Java, by reading a text or browsing through the API documentation. Of course, too, I hope this tutorial will answer more and more of your questions as time goes on.

Here is the complete file containing one the minimal applet `psSimpleApplet.java`:

```
package psapplets;

import java.applet.*;
import java.awt.*;
import psplot.*;
import real.*;

class psSimpleCanvas extends PlotCanvas {
    public psSimpleCanvas(Applet app, int w, int h) {
        super(app, w, h);
        double ht = 3;
        // preserve aspect ratio:
        double wd = w*(ht/h);
        setCorners(-wd, -ht, wd, ht);
    }

    public void draw() {
        PlotPath p;
        p = new PlotPath(this);
        p.moveto(-0.5, -0.5);
        p.lineto( 0.5, -0.5);
        p.lineto( 0.5,  0.5);
        p.lineto(-0.5, 0.5);
        p.close();
        p.fill(Color.red);
        p.stroke(Color.black);
    }
```

```
}

public class psSimpleApplet extends Applet {
    psSimpleCanvas p;

    public void init() {
        p = new psSimpleCanvas(this, bounds().width, bounds().height);
        add(p);
        show();
    }
}
```

We shall now go through it piece by piece.

```
package psapplets;
```

This means that the `.java` file is in a directory `psapplets` which is a subdirectory of a directory in your class path.

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

We are using several collections of programs already constructed, and we are working in another. There are official ones with `java` in their names, and the special ones for use in this tutorial. These collections are called **packages** in Java. The package `psplot` contains basic graphics stuff, while the package `real` contains some simple utilities for handling real numbers (doubles) apparently missing from the official packages. The command `import` makes stuff in these packages immediately accessible to our program. They do not load code, but just make definitions accessible. These lines are somewhat analogous to the inclusion of header files in `C` with the preprocessor command `#include`. But in Java the source files also serve as header files. Probably all of your programs should begin with at least these `import` statements.

It is a good idea to put all Java programs in some package, although certainly not necessary. This will help you avoid name clashes and further confusion later on when you have lots of Java programs.

```
class psSimpleCanvas extends PlotCanvas {
    ...
}
```

Except for `package` and `import` lines, all of a Java program must be contained in a **class**. I'm not going to say much about classes, a standard feature of object-oriented programming. The main point is that each class sets up its own environment. Inside a class, Java programs look a lot like `C`. The classes you use yourself will probably extend another class already built. The `psplot` package contains several different classes, but the most important one is `PlotCanvas`. All of your drawing canvases will extend it in some way. The class `PlotCanvas` has lots of complicated code in it, which you might want to examine later. In particular, lots of things in your program are going on behind the scenes.

Most of the source code in the class `psSimpleClass` is conatined in the code defining the class `PlotCanvas` which you are extending. Right now we won't even contemplate what is going on there, because a simple drawing applet won't need to know it.

```
public psSimpleCanvas(Applet app, int w, int h) {
        super(app, w, h);
        double ht = 3;
        // preserve aspect ratio:
        double wd = w*(ht/h);
        setCorners(-wd, -ht, wd, ht);
```

```
    }
```

Variables in Java are either of primitive type or of the type of some class. The primitive types you will see most here are integers, real numbers, and booleans (`int`, `double`, and `boolean`). A class variable is essentially a pointer to a structure which is an **instance** of that class. Just declaring the variable in your program does not allocate any space for the structure, but just for the pointer. The space itself is created by a **constructor** procedure. We shall see an example in a moment. A constructor procedure in a class has the same name as the class. A class can have several different constructors, distinguished by the type of arguments they have.

This constructor is called by the main applet class in the file. The integers w and h are the width and height of the applet in pixels. You will usually use these to set up a coordinate system on your canvas. The constructor here calls the constructor for the basic class `PlotCanvas`, with the same arguments. This establishes a default coordinate system with $(0, 0)$ at the lower left, and the basic unit still one pixel. We want to adjust the coordinate system to more reasonable units, and we want the geometry of our canvas to match that of the applet, so that squares are still square. We scale things here so the y-coordinate at the bottom is -3, the top is 3, and the x-values at left and right are set to preserve the aspect ratio. That is to say, the variable ht is set to 3, wd is calculated to preserve aspect, and the coordinate system in our canvas is chosen so the lower left is $(-\mathrm{wd}, \mathrm{ht})$ and the upper right is $(\mathrm{wd}, \mathrm{ht})$. In the new coordinate system the unit lengths in both x and y directions are the same size in pixels. Lots of drawing canvases you create will have almost identical constructors.

```
public void draw() {
       PlotPath p;
       p = new PlotPath(this);
       p.moveto(-0.5, -0.5);
       p.lineto( 0.5, -0.5);
       p.lineto( 0.5,  0.5);
       p.lineto(-0.5, 0.5);
       p.close();
       p.fill(Color.red);
       p.stroke(Color.black);
    }
```

The individuality of the canvas comes out in the only other routine visible here. It does the actual drawing. Things are set up in the class `PlotCanvas` so the drawing is done automatically every time your browser opens your applet.

Drawing is reminiscent of PostScript. A path is created with a constructor. The path is built with `moveto` and `lineto` commands. Then it can be stroked or filled in various colours. You can see which colours are available by looking at the API. In principle there are zillions of them, but in practice different browsers don't seem to be able to render very subtle shades. I recommend using only *red*, *green*, *blue*, *magenta*, *cyan*, *yellow*, *black*, *white*, or *gray*, just like that very small crayon box you used in kindergarten. Recall that in Java all variables are attached to classes; the colors are attached to the class `Color`. The colors `Color.red` etc. are the nearest thing Java has to global variables. Other global variables of this kind are `Math.PI` and `Math.E`.

```
public class psSimpleApplet extends Applet {
    psSimpleCanvas p;

    public void init() {
        p = new psSimpleCanvas(this, bounds().width, bounds().height);
        add(p);
        show();
    }
}
```

Every applet program has to have something like this. The name of the single public class must agree with the name of the file (so this program is contained in a file `psSimpleApplet.java`), and the class must have the words `extends Applet`. We won't examine the notion of **class** in any detail, but I remind you that classes are much like

structures in Pascal or C, and that a class variable which is a class is essentially a pointer to a structure. However, the pitfalls of pointers in other languages are avoided; the possible uses of a class variable are tightly controlled in Java in ways that pointers are not controlled in most languages.

I repeat also that all variables are local to some class, or more precisely to some **instance** of a class.

Classes that you yourself create will usually be extensions of classes that already exist. Every applet must be an extension of the `Applet` class.

The applets we construct using the `psplot` package will all have a variable in the applet class which is a canvas of a specific kind, one which must be created by you. This canvas must extend the class `PlotCanvas` which is one of the classes in the `psplot` package. Each different applet will use a different type of extension. Here it is named `psSimpleCanvas`. The convention implied in this name is conventional but not required.

Every one of your applets will have a routine called `init()`, which is called as soon as your applet is created by your browser. It will usually create your canvas, add it to the applet, and then `show()` your applet. These three lines are just about the minimum your applet class has to have.

The applet class here contains a variable `p` whose type is `psSimpleCanvas`. As mentioned above, space for what it refers to created with the constructor call

```
p = new psSimpleCanvas(this, bounds().width, bounds().height);
```

Arguments to all procedures in Java are severely typed. The procedure `bounds` is one in the basic applet class that returns a structure with two components, width and height. The values it returns are are those set in the calling `.html` file. They are both integers. Suc information is contained in the API documentation and in the Nutshell Java book.

As an exercise, you might try to draw a triangle or a regular pentagon. There are no global functions in Java, and you will use the functions `Math.cos` and `Math.sin`. They take a `double` as argument, which is assumed to be in radians. You should probably look over the whole `Math` package in the Java API to see what's there.