Introduction to Physics 120 Java Applets

Programming and Java Basics

Throughout the Physics 120, 121, 124 sequence, there will be homework problems that require numerical analysis and computer programming. The official programming language for the course will be java. Java is an object-oriented language similar to `C++`. It has two useful advantages: First, java has relatively simple built-in functions supporting a graphical user interface. Second, compiled java code will run under any operating system.

For this course, I will not ask that you learn any advanced aspects of java. Instead, we will use Java in the following way: I will write an 'applet', a mini-application that may include graphics and other user-interface elements. This applet will be complete except for one crucial piece, the function that actually does the computation. I will ask you to provide this function according to certain guidelines that I will present. When you write the function correctly and compile it together with the other codes for the applet, you will get an interesting program that you can then experiment with and learn from.

An applet is a mini-application associated with a web page. To run the applet, access the web page using your browser. Alternatively, on UNIX or Windows systems, you can view the applet `A.html` by typing the command `appletviewer A.html &`. The same statement applies to Macs running OS X, which is a version of UNIX, if the 'Developer Disk' that comes with OS X is installed. To print the result of an applet, use the 'Print' menu item on the appletviewer or the web browser (or, if all else fails, take a screen shot).

The instructions for compiling Java programs are different on different operating systems. The instructions are simplest under UNIX (or OS X). A Java program will typically consist of a file `A.java` containing the main program, plus other files with various subroutines. To compile the main program `A.java`, create a directory, and put all necessary files in it. Then go into the directory and type `javac A.java`. If you have syntax errors, error messages will appear; otherwise, you are done. Under Windows, you will need to download a java compiler. This can be obtained free by following the links from the Web page

`http://java.sun.com/docs/books/tutorial/getStarted/cupojava/index.html`

Java on older Mac systems is not exactly transparent and may also be out of date. As an alternative for either Mac or Windows, Stanford students can download the Metroworks 'Code Warrier' development environment free of charge by Stanford students from the web site:

`http://www.stanford.edu/class/cs/compilers/`

This environment will give you C, `C++`, and java compilers.

The syntax for numerical functions in java is very similar to that of C. The language is case sensitive. All variables must be of a declared type. Floating-point variables are `float` or `double`. Variables declared `int` are integers; be aware that

```
int X =  999/1000;
```

produces `X` = 0. All assignments end with a semicolon (;). Functions must be declared by their return type, or `void` if no value is returned. Typically, I will provide an example for the syntax of the function you must define. If I ask you to use any special functions of java beyond this level, they will be simple and spelled out explicitly. If you find that you need a java reference, there are a huge number of books available. I recommend particularly *Core Java*, by Horstmann and Cornell. The first few chapters of that tome will give you much more information than you will need for this course.

There is one annoying feature of numerical analysis in java. The standard mathematical functions such as square root, power, sine, and absolute value must be writtten `Math.sqrt`, `Math.pow`, `Math.sin`, and `Math.abs`. The constant π is available as `Math.PI`.

First Applet

To try out your ability to download and compile java programs, download the files `FirstApplet.html` and `FirstApplet.java` from the directory

`http://www.slac.stanford.edu/~mpeskin/Physics120/java/`

(There is a link to this directory on the Physics 120 web page.) Typing the command `javac FirstApplet.java` will create a compiled java program called `FirstApplet.class`. The entire content of the file `FirstApplet.html` is a line:

```
<applet code="FirstApplet.class" width=200 height=100></applet>
```

This opens an applet of size 200 × 100 pixels and runs the program. Hopefully, the code in `FirstApplet.java` is almost self-explanatory.

Plotter Applet

In the first two problem sets, and in some later problem sets in this course, we will make use of a java applet that plots points and curves. To obtain the applet, go to the directory `http://www.slac.stanford.edu/~mpeskin/Physics120/java/` and download the following files:

`Plotter.java`, `Plotter.jar`, `plotStream.java`, `myPlotter.java`, `myPlotter.html`.

To compile the applet, type `javac myPlotter.java`. This will create a number of `class` files. To see the result, access `myPlotter.html` with your web browser, or use the appletviewer. You should see a graph of the function $y = x^2$, with some points plotted on it.

The structure of the java code in this example is one that we will see repeatedly through the course. The files `Plotter.java` and `plotStream.java` are long and somewhat technical. You do not want to modify these files, and you may not even want to look inside them. The

file myPlotter.java is brief and (hopefully) clear. Its function is to lay out the problem and solve it mathematically. It then calls Plotter.java to do the technical work of writing to the screen. In the rest of this section, I will step you through the files myPlotter.java and myPlotter.html so that you can see how they work and how to modify them to solve interesting problems.

The purpose of the file myPlotter.html is only to call the applet. The whole file contains only one command:

```
<applet code="myPlotter.class" archive="Plotter.jar"
                                    width="600" height="500"></applet>
```

This command instricts the viewer to run the compiled program myPlotter.class, using the additional class files that are contained in the jar (java archive) file Plotter.jar, and any other class files that might be present in the same directory. In this example, I have used Plotter.jar to collect the various class files that appear when Plotter is compiled. If you wish to change the names, please note that a java file that begins public class AA ... must be named AA.java. This file will compile to a file AA.class, and it is this file that should be referenced in the html file.

Now look at the code in myPlotter.java, shown in Figure 1. The basic unit of a java program is the 'class', a programming structure that includes both data and intrinsic functions, 'methods'. The most important item to notice is the structure of myPlotter is that it 'extends Plotter', that is, it uses plotting methods defined in the file Plotter.java. The data is contained in two classes A, B of type plotStream, a structure defined in plotStream.java. In both cases, it is not necessary to know what is inside this classes but only what methods are available and how to call them.

The function setup contains calls to setPlotSize, setLimits and setTicks. The call setPlotSize(x,y) gives a plot of size x by y pixels. The call setLimits(x1, x2,y1,y2) gives a plot whose x axis runs from x1 to x2 and whose y axis runs from y1 to y2. The call setTicks(a,b) gives a plot with tick marks along the axes at intervals a for the x axis and b for the y axis.

The data to be plotted is transmitted to the plotter through the class plotStream. This class has the method add(x,y). If you define a plotStream PS and then call

```
PS.add(x,y)
```

the point (x, y) will be added to PS. The myPlotter applet has two plotStreams, one for each data stream. In the setup function, storage is assigned for these variables, and the variables are initialized. Alternatively, initialization could be done in a 'constructor' function called when the class myPlotter is initialized. The constructor is a public function with the same name as the class.

I hope that you can decipher the syntax of the functions fillA and fillB that fill the plotStreams. The function fillA simply puts points (x, y) into A. The function fillB uses

```
import java.awt.*;
import java.awt.event.*;

public class myPlotter extends Plotter{

  plotStream A, B;

  public void setup(){
    setPlotSize(500,400);
    setLimits(-5.0,5.0, 0.0, 20.0);
    setTicks(1.0,1.0);
    A = new plotStream();
    B = new plotStream();
  }

  public void plot(Graphics g){
    verticalAxis(g);
    fillA();
    g.setColor(Color.red);
    plotPoints(g,A);
    fillB();
    g.setColor(Color.blue);
    plotCurve(g,B);
  }

  void fillA(){
    A.add(2.0, 4.0);
    A.add(3.0, 9.0);
    A.add(-1.0, 1.0);
    A.add(-4.0, 16.0);
  }

  void fillB(){
    double x;
    for (x = -5.0; x <= 5.0; x+= 0.2){
      B.add(x, x*x);
    }
  }
}
```

Figure 1: The source file `myPlotter.java`.

a loop construction to add to `B` points on the curve $y = x^2$. The `for` loop steps the variable `x` from -5.0 to 5.0 in increments of 0.2. Note the way that the range of each loop and function is delimited by braces. The syntax is just the same as in `C` and `C++`.

The actual plotting is done by the function `plot`. This function depends on a `Graphics` object, an internal java class whose methods include various ways of drawing on the screen. You do not have to know exactly what a `Graphics` object is, but you should follow the way that the object `g` is used in the sample code. The Plotter has methods

```
plotPoints(g,P), plotCurve(g,P), plotLines(g,P), plotHistogram(g,P)
```

which, draw the `plotStream`, respectively, as a set of points, a smooth curve, a broken line running from point to point, and a histogram. Additional methods are

```
horizontalAxis(g), horizontalAxis(g,y), verticalAxis(g), verticalAxis(g,x)
```

which draw horizontal and vertical axes at 0 or at the given location, and

```
drawLine(g,x1,y1,x2,y2),  drawString(g,"Information",x,y)
```

which, respectively, draws a line from `(x1,y1)` to `(x2,y2)` and writes a string of characters at `(x,y)`.

You might also wish to draw different lines in different colors. Java includes the command

```
g.setColor(Color.blue)
```

which sets the color with which subsequent lines are drawn. Java understands the colors `white, black, blue, green, red, orange, cyan, yellow, magneta, gray`. If you would just like your curves to be in different colors and you do not care what color they are, you can use the `Pplotter` command

```
switchColor(g)
```

which changes the drawing color each time it is called.

Debugging

Only by a miracle will a computer program work correctly the first time. Computers are infuriatingly literal. If you omit a capitalization or a punctuation mark, or put `=` where you should put `==`, they will refuse to do what they are asked. Typically, much of the work of writing a computer program consists of patiently removing such errors of execution.

Computer programmers use a variety of tools for identifying the location of errors. If there is an error in the computer syntax, javac will report to you where it failed, and you can modify the file at that point. Errors in which the computer program runs but does not do the calculation correctly are more difficult to find. In such a case, I find the following simple trick useful. If the program involves some variable `x`, the Java statement

```
System.out.println(x+ "  ");
```

prints out the value of `x`. More generally, `println` takes as its argument a string of characters, `+` is string concatenation, and the concatenation of a number with a string turns the whole expression into a string that can be printed. You can use this print command to verify that, at each point in the program, the variables are being assigned the proper values. Wherever the test fails, an error correction is needed.