9 Graphical Output

This section introduces the local `SimpleGraph` *class that is required for the next checkpoint.*

9.1 Introduction

Much of Scientific Computing involves the analysis and display of data which is best achieved graphically. In JAVA we have the ability to produce high quality graphical output directly from your application. In this course we shall use a locally written interface called `SimpleGraph` which uses the *Scientific Took Kit* (`sgt`) written and put in the public domain by the NATIONAL OCEANOGRAPHIC AND ATMOSPHERIC ADMINISTRATION (NOAA)[1].

The supplied simple interface allow access to the very powerful `sgt` graphics toolkit with the minimum of effort.

9.2 Basics of the `SimpleGraph` class

The basic use of the best explained by a simple example to plot out a *cosine* graph.

```
import gov.noaa.pmel.sgt.cplab.*;                                    // (1)

public class CosPlot{                                                // (2)
  public static void main(String args[]){                            // (3)

    SimpleGraph graph = new SimpleGraph("A Cos Graph","angle","Cos"); // (4)
    DataSet cosData = new DataSet();                                 // (5)

    for(int i = 0; i < 100; i++){                                    // (6)
      double angle = 4.0*Math.PI*(double)i/100.0;                    // (7)
      cosData.addPoint(angle,  Math.cos(angle));                     // (8)
     }

     graph.addData(cosData,"Cos");                                   // (9)
     graph.showGraph();                                              // (10)
     }
  }
```

Go through this line by line,

(1) Import the `cplab` additions to the `sgt` classes.

(2)-(3) Usual start for all main programs.

(4) Create a `SimpleGraph` object called `graph` with three `Strings` as parameters, these being the "Graph Title", "X-axis label" and "Y-axis label" respectively.

(5) Create a `DataSet` called `cosData` to hold the graphical data that will be plotted.

[1]NOAA is the US Government oceanographic and weather research institute, see details at `www.noaa.gov`.

(6)-(8) Go round a loop 100 times, scale the `angle` into the range $0 \rightarrow 4\pi$, and each time add a new point to `cosData` using the `.addPoint(double x, double y)` method.

(9) Add the completed data, held in object `cosData` to the `graph` using the `.addData(DataSet d)` method and sets it title to `"Cos"`.

(10) Display the completed graph in its own window.

Again the basic idea is just like the `Display` class, you create a new `Graph` object, you then access the properties via its *methods*.

Again it is *strongly* suggested that you type-in this short program and make sure you understand how it works.

More Advanced use of `SimpleGraph`

Additional DataSets

You can create and add additional `DataSet`s to the single `SimpleGraph` by simply creating another `DataSet` and adding it with the `.addData()`. The displayed graph will show multiple lines with the *x* and *y* axis scaled to fit the largest.

Note: You need only call `.showGraph()` once, if new `DataSet`s are added after the graph is visible it will be re-drawn automatically with the additional data.

Additional exercise: Modify the above Cosine graph to produce one graph showing both Sine and Cosine.

Multiple Graphs

You can create multiple graph windows by creating each of them with the `SimpleGraph` constructor. Each graph will be in its own window and will be totally independent and you can then send any number of `DataSet`s to them.

Lines and Points

The linetype is a property of the `DataSet`, and can be set prior to being *added* to the `Simplegraph` using the `.setStyle(style)` method. The useful style options are:

Parameter	Line
`SimpleGraph.SOLID`	Solid line (the default)
`SimpleGraph.DASHED`	Dashed line
`SimpleGraph.HEAVY`	Bold line
`SimpleGraph.STROKE`	Bold dashed line
`SimpleGraph.MARK`	Mark point only (no line)
`Simplegraph.MARK_LINE`	Mark point and draw line

In the above example if you add the line

```
cosData.setStyle(SimpleGraph.DASHED);
```

immediately before line (9), then the graph will be drawn with a dashed line.

Colo(u)r of Lines

As with linetype above, the colour of the line is a property of the `DataSet` and can be set, prior to being added to the graph, with the `.setColor(Color)` method[2].

The specification of `Color` is more complex as it used the standard JAVA colour model. To use `Color` you must first put

```
import java.awt.Color;
```

at the top of your program. You can then use the standard colours,

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

Note again the American spelling of *grey* is *gray*!

In the above example you can add

```
cosData.setColor(Color.magenta);
```

before line (9), will result in the line being drawn in magenta.

There are many other ways to specify a colour, including specifying the Red, Green and Blue components. See any good reference book, for example DEITEL page 517-519.

Adding units to graphs

There is a variant of the `Simplegraph()` constructor with 5 `String` parameters that allows you to add units to the x and y axis defined by

```
SimpleGraph(Title-String, X-axis, X-Units, Y-axis, Y-Units);
```

where each parameter is a `String`.

Additional Features

`SimpleGraph` can also be used with arrays of `double`s to specify *x* and *y* points, do elementary least square fitting, and a range of other advanced features not needed in this course. See the on-line documentation (linked from the SCIENTIFIC PROGRAMMING HOME PAGE) for details.

Examples

The following on-line source examples are available:

- For a cos and sin graph with different colours CosSinPlot

What next ?

You are yow ready to try *Checkpoint 4* which involves graphs of SMH motion.

[2]Note the American spelling of colour, this is unfortunately standard in programming!