Part IB Java Tick 0

May 12, 2009

1 Introduction

This exercise is intended to form a bridge between the Part Ia Java course and work that will be done in Part Ib. Although it is handed out at the end of the Easter Term during Part Ia, your submission is not due in until **noon on Friday 31 October**. At that time you will have had just three weeks of Michaelmas Term lectures, and so it is expected that there will not be too much conflict with supervision work or other lab-issued practical tasks.

Many people may find it convenient to work on this exercise over the vacation, and the starred parts of it are intended for those who wish to give themselves a running start with Part Ib and the advanced programming languages course in particular. Any student who changes into Part Ib of the CST having taken some Tripos other than Part Ia NST or CST and hence who has missed the Ia Java course can use this exercise as a way of proving that personal study over the long vacation has brought them sufficiently up to speed to survive Part Ib. They will be expected to submit their solutions at the normal time. Any affiliated students who take direct entry into Ib in the Michaelmas Term should consult the department as necessary, special arrangements will be made for them. The exercise concerns building a Java application that has windows and menus and that responds to mouse activity. It is acceptable to work on it either using the command-line Java tools (e.g. the javac compiler) or an environment such as Eclipse.

The starred extension to this exercise explores the Microsoft .NET family of languages and the latest XAML features of the .NET Framework version 3.0.

In many cases when programmers build serious windowed applications they will use a development tool that permits them to design visual aspects of their code graphically, and which then generates skeletal code that calls suitable parts of the Java libraries to make the indicated windows, menus and so on appear. Even though you may use such tools later in the Tripos or your career it's valuable for you to have some exposure to the classes and constructs that are involved since you'll be better prepared to modify and maintain automatically generated user-interface code if you have had hands-on experience working with at least a small example at a level where you get to see all the details.

For this exercise you are given some template code (which has in fact been created at least partly mechanically) which pops up a window that bears a superficial resemblance to the one that the "BlueJ" system provides. It has a button marked "Add Class" to the left, a large rectangular area to the right and a narrower rectangular region at the bottom. It also has a drop-down menu, and it responds to the mouse as well as menu requests (including one that makes it print itself). The template code is available from `http://www.cl.cam.ac.uk/teaching/0809/IbAsExBrfg/TickFrame.java`. If you compile and run this code, you should see something a bit like Figure 1.

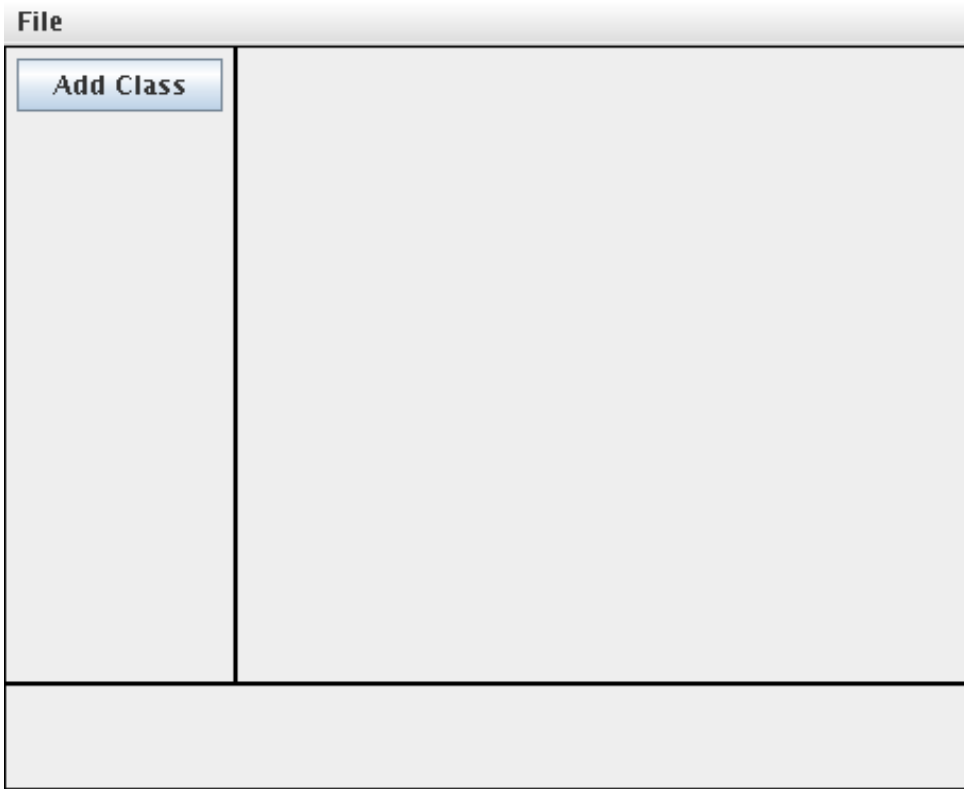

Figure 1: Example output from the template code

2 What has to be done

You are to do three things to it:

1. You will be extending the program so that you can have boxes in the main window that will each hold a class name and a list of methods that the class would provide. In fact your code is just going to end up as a diagram-drawing program that might be used to plan the class hierarchy of a program you might be developing. Your first task is to arrange to be able to represent and draw items that look a bit like UML class descriptions.

   Devise a class, `ClassBox`, to represent the important features of an item in the diagram. At least include the name of the class represented by the item, a list of the class's methods, the (x, y) co-ordinates where it appears on the screen, and the width and height of the box. Provide suitable methods to manipulate the item: to move it, resize it, and to add additional methods to its list. There is no need to provide methods that draw the item; merely storing the information needed to draw the item is sufficient.

   Devise another new class, `Document`, to represent a document. A document is permitted to consist of a variable number of items (as represented by your first class) and a variable number of arrows. It is NOT acceptable to assume that the number of `ClassBox`es to be stored will never exceed any particular constant: you must use a data structure which expands its capacity on demand.

   Produce a third class to store the fields needed to represent an arrow, including co-ordinates, thickness, connected `ClassBox`es, etc.

The code provided uses just a `JPanel` for its `main_panel`. Change the code provided by adding a method `setDocument(Document d)` to specify the document to be associated with the graphical depiction. Override the `paintComponent` method so it can draw on the window each of the items and arrows contained within the document. You ought to make sure that you paint the background of your `JPanel` as well as drawing the rectangles and text that make up the boxes. Show the class name in a larger font size than that used for method names, and in bold type (hint: look for `getFontMetrics` in the Java API documentation). If the class's name or any of the method names are too wide to fit the width of the box, replace any letters that would spill outside with an ellipsis ("...") to indicate that the name has been truncated. Ensure the ellipsis itself does not spill over the boundary of the box.

2. Arrange that when the "Add Class" button is pressed a new `ClassBox` is created and added to the panel's underlying document. For a very minimum tick you can use a simple dialogue box to allow the user to type in a name to give each new box. Now add another, narrow panel on the right hand side of the window to show various counters. You will need another class that is derived from `JPanel` and which also has a `setDocument(Document d)` method. Arrange for the `paintComponent` method to print several lines of text, wrapped to fit the width of the panel. These lines of text should indicate counts of the numbers of items in the document, the number of arrows, and the total number of methods. Ensure the counts are refreshed whenever the underlying document is updated.

   Now construct a toolbox with a button, "Add Arrow" beneath "Add Class". To add an arrow, the user should click within each of two boxes in turn and the resulting arrow should be a line from the centre of the first box to the middle of the top line of the second box. Draw an arrow head at the latter terminal. Arrows should 'remember' their source and destination boxes such that if either is repositioned the arrow is altered appropriately.

3. As the final mandatory part of the exercise you should look at the sample code and check the parts that deliver mouse events to the `JComponent`. Your task is to add code to the `JPanel` so that it detects when the mouse button is pressed within the region occupied by a `ClassBox`. It should then handle mouse drag and release events and on their basis arrange to re-position the `ClassBox` and then call `repaint()` so that the screen is brought up to date. The objective is to allow the user to move boxes around freely! Make it possible, by double-clicking within a box, to add another method to the appropriate object's list.

3 Getting a star

There are clearly an amazing number of ways that it would be possible to move on from what has been requested so far. The following three items indicated here would lead to the award of a star on the tick list...

1. Add extra menu items called `Open` and `Save` and arrange to be able to preserve and re-load the state of a diagram. You may find it useful to read up on what Java calls "serialization", which may turn out to do almost all the hard work for you, so look up the `Serializable` interface!

2. Make it possible to remove as well as add `ClassBox`es, and to edit the information that they display. You will need to design and manage extra dialog boxes and perhaps menu items to do this, but there is no need to go over-board in the levels of complication.

3. Finally, look up "reflection" in the Java API documentation and use Java's reflection mechanism to offer a new means to add a box to the diagram: the user should be able to enter the filename of a `.class` file and your program should use reflection to determine the class's name and list of methods. If the class contains overloaded methods, show the method name just once in the list but in italicised characters. If the class derives from an another class

(other than `java.lang.Object`), also add the base class to the diagram as another box and add an arrow pointing from the box corresponding to the base class to that of the derived class. Test your code by getting your program to draw its own `.class` files. Move the boxes around yourself so the arrows are easier to follow and include a screenshot in your submission for the starred exercise.

A separate task which, although it will not earn a "star", will prove to be extremely useful preparation for the second year of the Computer Science Tripos, is to re-implement the basic and starred exercises using the Microsoft .NET framework (use version 2 or later). Try using the XAML GUI description language, which is new to .NET version 3 and integrates with the Windows Presentation Foundation. You can use 3D graphics, lighting effects, animations and pixel shaders with simple XML descriptions of what you want. If you do not already know about the MSDN Academic Alliance, look this up on the Computer Laboratory website — you will be able to obtain the Microsoft development tools free of charge. Use the C# language and, as you work, compare the syntax and semantics of the C# keywords with those of the Java language. List as many ways as you can in which C# has been changed from standard C/C++ to be more like Java, and in which ways the designers chose to stick with the C/C++ ways of doing things instead of mirroring features of the Java language. Try to work out why each decision was taken the way it was.

Anybody who has nothing better to do with their vacation is welcome to see how far they can push this project, perhaps aiming at autogenerating skeleton classes and/or test harnesses. In any case, ensure that what is submitted to the tickers satisfies all of the core requirements, and if you end up with something spectacular, then send it to `jt0+smh22@cl.cam.ac.uk` and request that it gets put on a website or that you get a prize or something.

4 The initial skeleton code

This code and all the rest of the information provided in this briefing is available for download via the web-site `http://www.cl.cam.ac.uk/teaching/0809/IbAsExBrfg/`

Please check this URL from time to time, since any updates or errata will be published there.

5 Other resources

Full documentation of all the Java classes can be downloaded from Sun's website for installation on your own computer (`http://java.sun.com`). You have probably been already looking at that information on-line during the course of the year and so know what's there and how to navigate the documentation.

Much of the stuff in the basic tick has been covered previously in the 1a "Programming in Java" course — if you are transferring into CST from elsewhere, you may find the following useful: `http://www.cl.cam.ac.uk/teaching/0809/ProgJava/` (and in particular workbooks 7 and 8). For general Java programming advice, the book "Thinking in Java" by Bruce Eckel (`http://mindview.net/Books`) is available on the web as a free download. Either that electronic version or a copy from a bookshop provides fairly competent coverage of lots of the grubby details of Java.

Good luck!

Steve Hand.

(with thanks to Dr Tim Harris, Dr John Fawcett and Dr Arthur Norman).