Intro to Java: after-course work

22 April 2010

Visit the Sun Java Tutorial page, `http://java.sun.com/docs/books/tutorial/`.

1 Consolidation of basic idea

1. Follow the Getting Started tutorial trail.
2. Using an appropriate IDE, check that you can write, compile and run a Hello World application in Java. Here's the one from the Sun tutorial:

```
/**
 * The HelloWorldApp class implements an application that
 * simply prints "Hello World!" to standard output.
 */
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the string.
    }
}
```

3. Make sure you can run your application both from the IDE and from the command line.
4. Make sure you can start the debugger in your IDE and step through the program.
5. Now *without having the code above in front of you*, write a Hello World application from scratch using a text editor instead of the IDE. (If you did this without even one minor syntactic error, congratulations!)

2 Bread and butter

1. Look at the Learning the Java Language tutorial: `http://java.sun.com/docs/books/tutorial/java/TOC.html` Go through it in as much detail as you can find time for, paying particular attention to the parts that were not mentioned in the course.

2. Do your own selection of the Questions and Exercises in the tutorial and check your answers there.

3 Putting it together

NB to complete this, you will need to look up odd things not covered in the session. Try to approach it in an incremental way.

Your task is to write an applet integrating the following skills:

- building an applet
- general control structures
- basic GUI building using Swing (we'll talk more about this: for now you will probably want JPanel, JButton and JTextField at a minimum)
- basic event handling (ditto: for now all you need is for your main class to implement ActionListener by defining a method `public void actionPerformed(ActionEvent e)`; then you add this same class as the actionListener to appropriate buttons. See this page: `http://java.sun.com/docs/books/tutorial/uiswing/events/intro.html`.)

Provided you practise those it doesn't much matter what you do, so if you have a project in mind that you'd actually like to get done, do that. As an example, here's a description of a simple game I made when my preschooler, Robin, was obsessed with arithmetic. (This uses sound, which can be handled in various ways: at the time I used `javazoom.jl.player.Player` available from `http://www.javazoom.net`.)

The applet requires a small collection of sound files: I used animal sounds and instrument snatches. It presents a collection of buttons each labelled with the name of a sound. Robin clicks one of the buttons, but the sound doesn't play yet. The applet randomly chooses an addition sum whose answer is between 0 and 9 inclusive, and presents it, e.g. $1 + 2 =$. It also presents a number line implemented as a collection of buttons, each button labelled with an integer. Robin's task is to press the button of the number which is the correct answer. If he does that, the displayed sum becomes the correct $1 + 2 = 3$, and his chosen sound plays. If he presses the wrong button, nothing happens.

Enhancement: let the applet cover a wider range of sums. On initialisation it asks for parameters of the problems it should set, allowing Robin to choose the range of answers and the operation(s) to be tested. It lays out the screen appropriately, using a number board so that e.g. 10 comes directly below 0.

4 A design-principle focused question

Consider the following Java code, which comes (with one comment deleted) from the open source UML modelling tool project ArgoUML.

```
public void addNode(Object node) {
    if (!canAddNode(node)) {
        return;
    }
    getNodes().add(node);

    if (Model.getFacade().isAModelElement(node)
            && Model.getFacade().getNamespace(node) == null) {
        Model.getCoreHelper().addOwnedElement(getHomeModel(), node);
    }

    fireNodeAdded(node);
}
```

1. Check that you understand the flow of control: draw an informal diagram. (Later, we'll cover UML sequence diagrams which are appropriate for this.) Which methods do you think are static?

2. This code violates the Law of Demeter. Explain how.

3. Guessing what you can about the context of this code from the names it contains: to what extent, if at all, do you consider this likely to be a design flaw? Using this code as a starting example, discuss situations in which Java programmers often violate the Law of Demeter, and what alternatives they have. Do you think Java programmers should always abide by the Law of Demeter or some variant of it? Justify your answer in no more than 500 words.