APS101 lab 10 – week 12

This document contains the instructions for the week 12 APS101 lab.

1 Objectives

1. Working with a new IDE called Eclipse.
2. Working with arrays and sorting algorithms.
3. Practice testing with arrays.

2 Eclipse

So far, we have been using DrJava as an Integrated Development Environment (IDE) to develop and compile our code. While DrJava is a very simple and good IDE for beginners (for example, it has a nice Interactions Pane for trying out Java expressions), there are much more "serious", versatile, and powerful IDEs out there. Eclipse is one such IDE that can be used for many different programming languages, including Java, C, C++, etc. It is installed on ECF machines and you might be using it in your other courses. In this section, you will spend around 20 minutes exploring some of its basic functionalities.

To run Eclipse: Click on "Applications→ECF→ Eclipse". The first time you run it, it will ask you about your "Workspace" directory. This is the directory where you keep your files for various projects. It's OK to choose the default value for now.

When Eclipse starts, you will see the "Welcome" window with a bunch of circle-shaped icons (e.g. what's new, Eclipse tutorial, etc). Close or minimize this window (you can open it later from the "Help" menu by choosing "welcome").

Now, you will see the main window with a lot of panes. This view is called "java perspective".

You can start a new "project". A project is a set of files necessary to create an application program. For example, in the TicTacToe game, all those Java files we wrote in lecture are needed to run the TicTacToe application, so you may call it project "TicTacToe".

Let's create a project called "lab10": Select File→New→Project. Then, select "Java Project". Choose the name "lab10" for your project. Press "finish".

Now you are ready to add classes to your project. Select File→New→Class. In the "Name" box, type your class name, e.g. "Hello". There are many options in this window (have a quick look but don't change them for now). Press finish.

You will see in the left pane, under lab10 project, that you have a class called Hello.java. In the center pane (i.e. the editor) you will see the empty body for class Hello. You can write your code there.

Let's add an instance variable to the code:

```
private int[] A;
```

As you can see, while you type, Eclipse checks the syntax for you. And when you finish typing, you will see in the right pane (called "Outline") the "A" instance variable is added!

Now, let's add a method:

```
public int square(int i){
  return i*i;
}
```

You can see "square(int)" is added to the Outline pane as well. The outline pane is a nice window to keep track of your variables and methods in each class. For example, if you click on a variable/method name, the cursor goes right to where its definition is. As for indentation, you can select all the code with the mouse and then from Source menu: Source→correct indentation (or simply press CTRL+I).

You should now save your code. File→Save or press CTRL+S.

To compile your code, first de-select "build automatically" from the Project menu. Now press CTRL+B or (Menu→build all). You will see any errors or warnings in the bottom pane under the "Problems" tab.

Let's add a main method. Try this: just type main

```
main
```

now press "CTRL+Spacebar" and you will see that it guesses what you want to type! Press enter to accept its suggestion and you have the main method skeleton added to your code! CTRL+Space can be used in a lot of places to "autocomplete" what you want to do.

Now, add the following line to the main body:

```
System.out.println("Hello World!");
```

Save and compile (i.e. build) the project.

Check the outline pane: you see your instance variables and your methods; they are color coded: can you guess why A is red while "square(int)" is green? (Yes, red means private, green means public!)

Also, notice that on top of the green bullet for "main" there is a letter "s", which means that it's a static method (there are tons of features like these in Eclipse).

Try running your code (it has a main method, so it should be OK). From menus: Run→Run. you will see a window open — in the left list select "Java Application", and under it the "Hello" class. Double-click on "Hello" and then press the Run button.

You will see in the bottom pane, a new tab called "Console" is added and the output of the program is shown there (which is "Hello World!").

Eclipse is very powerful. You can ask it to automatically generate and insert JavaDoc tags, constructors, getter and setter methods, JUnit test cases, loops, skeletons, etc. You can even customize it and add macros. There are a lot of tutorials out there to get the best out of this IDE. You can use it for many different languages, like Java, C, C++, etc. It has a nice debugger functionality so you can debug your code while it's running! You can also customize the look of Eclipse. The appearance and configuration of available windows in Eclipse is called "perspective". There are default perspectives like "java", "debug", etc. But you can download others. For example, by default Eclipse does not have the "Interactions Pane" that DrJava has. But you can download a plug-in to get a look and feel of DrJava! For this you will need to download the DrJava plug-in from `http://www.drjava.org/eclipse.shtml` and add it to the Eclipse plug-in directory. Then, you will have the "DrJava" perspective available, which will give you the Interactions Pane. **Note: you cannot do this on ECF lab machines as you don't have admin permissions to write in the plug-ins directory**. You are encouraged to install Eclipse on your personal computer or laptop. It's free and open source and available for various platforms (see `http://www.eclipse.org/downloads/`).

For the rest of the lab you will use DrJava.

3 Printing an array

Create a class called `Sorting`. Add a `static` method called `print` to class `Sorting` that has an `int[]` as a parameter and prints the array so that it looks like a real array, including the square brackets. Here is an example of the output for an array with three integers: `[1, 2, 3]`. Test it from the Interactions Pane.

4 Printing a String array

Switch navigator and driver.

Add a `static` method called `print` to class `Sorting` that has a `String[]` as a parameter and prints the array so that it looks like a real array, including the square brackets. Here is an example of the output for an array with three `String`s: `["Cows", "eat", "grass"]`. Test it from the Interactions Pane.

5 Comparing two String arrays

Switch navigator and driver.

Add a `static` method called `equal` to class `Sorting` that has two `String[]` parameters and returns true if they have the same `String`s in the same order, and false otherwise. Use `String`'s `equals` method to compare, of course. You can assume that no element in the arrays is `null`.

6 Backwards selection sort

Switch navigator and driver.

In class you saw a selection sort that behaved like this:

```
for each index i in the range 0 .. list.length - 1
  select the smallest item in list[i .. list.length - 1] and swap it with list[i]
```

Add a `static` method called `backwardsSelectionSort` to class `Sorting` that has an `int[]` as a parameter and sorts starting at the other end of the array and looking backwards through the array for the largest element.

Note: backwards does not mean descending order. Reread the explanation above and look at the pseudocode below.

```
for each index i in the range list.length - 1 .. 0   <-- Start at the right end
  select the largest item in list[0 .. i] and swap it with list[i]
```

7 Selection sort with Strings

Switch navigator and driver.

Add a `static` method called `backwardsSelectionSort` to class `Sorting` that has a `String[]` as a parameter and sorts in alphabetical order starting at the other end of the array and looking backwards through the array for the largest element. Your algorithm should be identical to the previous section, except it should work using `String`'s `compareTo` method. You can assume that no element of the array is `null`.

8 Backwards insertion sort

Switch navigator and driver.

In class you saw an insertion sort that behaved like this:

```
for each index i in the range 0 .. list.length - 1
  insert list[i] where it belongs in list[0 .. i]
```

Add a `static` method called `backwardsInsertionSort` that has an `String[]` as a parameter and sorts in alphabetical order starting at the other end of the array.

Note: backwards does not mean descending order. Reread the explanation above and look at the pseudocode below.

```
for each index i in the range list.length - 1 .. 0   <-- Start at the right end
  insert list[i] where it belongs in list[i .. list.length - 1]
```

9 Testing

Switch navigator and driver.

Write a JUnit test class that tests your `String[]` sorting methods, using your `equal` method from part 5. Test only one array per method. As always, interesting test cases are 0, 1, and many, but for this you should test even and odd length arrays as well.