CSCU9Y4 Programming Language Paradigms Practical 1

User names and passwords are unchanged from last year. The first few steps of this practical go in detail over some points you should know; this is to refresh your memory and to ensure you really understand what happens when objects are created in a Java program. Ideally, you should do the practical work here *before* the session, so you can make best use of discussion time and demonstrator assistance.

1. Navigate to your home folder. We suggest that you create a new folder called **Y4Pracs** within your home folder to hold the work of your practical classes.

2. Java programs are available for your use in the **Groups on Wide: CSCU9Y4** folder. Open the folder **Practicals** and copy the folder **Practical1** into your **Y4Pracs** folder.

3. Open the project **debugDemo** (from your own **Y4Pracs** folder). **BlueJ** is fine; feel free to use another IDE if you prefer, as long as it has a debugger. Without the debugger you can use the BlueJ IDE to

- Run individual methods.
- Inspect the contents of objects.

The debugger additionally allows you to

- Set breakpoints to control code execution.
- Step through the code one line at a time to observe how values in store change.
- Inspect the values of variables as code is executed.
- Observe the difference between creation (and inspection) of objects and primitive object types.
- Notice the sequence of method calls (although they're not very interesting here).

If you haven't used the BlueJ debugger then see the BlueJ tutorial (also in the **Practicals** folder on **Groups on Wide**) and follow the instructions in section 7 on using the debugger to examine aspects of the debugDemo program. The aim of this practical is to reinforce some of the ideas about execution of programs you have. You'll do this by using the debugger on some examples you may have seen before. Use any debugger you like.

4. Now open CoffeeShop1 (This is an example drawn from CSCU9A3, so should be familiar). This is a simple queuing system. Repeat the process above, running the tests in ShopTestTask1 and using the debugger to make sure you understand the point at which objects and variables are created, and their contents. Be sure to create suitable breakpoints to help you understand the code. Are the tests in ShopTestTask1 sufficient to ensure the code is correct? What does correctness mean here anyway? Add more if necessary.

5. Consider the sequence of calls in running testAddCustomer, the memory allocation for the items customer1 and customer2 and also the items in the queue. Note your answers for steps 4 and 5 here:

In class, discuss these answers with your neighbours (small groups of up to 4).

6. Now open CoffeeShop2. This is an extended version with multiple tills. Run the tests in ShopTestTask2 and check your results in the debugger. The code is deliberately poor: can you make it fail? When using the debugger can you see any patterns emerging in how local variables are created? What about objects? When do local variables and objects cease to exist? Write your answers here:

As above, discuss these answers with your neighbours (small groups of up to 4).

7. Modify CoffeeShop2 so that instead of adding a customer to a specified queue, you add the customer to a randomly chosen queue. (Don't know how to create a random number? Google it!) What changes do you have to make to the tests to accommodate this change?

Checkpoint

Now show a demonstrator that you have completed all this week's tasks. You should show us the modified CoffeeShop2 with tests, and your written notes on how method calls and object creation work.

If you didn't get to this point, finish the work off during the coming week and get the checkpoint marked next time. Checkpoints are dealt with as Canvas quizzes.

You are advised to consider these extra steps to broaden your knowledge:
CoffeeShop2 uses LinkedList to implement the queue. ArrayList provides a different implementation of lists. If you changed your use of LinkedList to ArrayList would there be any other changes necessary in the program?

You know about recursion. What do you think the debugger will show when you call a recursive method? Find an example of such a program and have a look.