# Advanced Programming

**Lab Sheet 5 – Exceptions, Debugging & Profiling**

## Introduction

This lab gives you some extra practice at handling errors and exceptions, and using the various tools provided by your IDE to help you understand what's happening when your programmes aren't performing as you imagined they would. It relies on the information in this week's lectures, and will help you with both future weeks' labs, and the eventual assignments on the unit.

Before starting this lab, make sure you have completed the previous exercises from the previous lab sheets. You'll be asked to go back and work on some of them later in the session.

## Task 1: Using Exceptions When Errors Occur

Create a new eclipse project, containing a class called Restaurant. It isn't going to be a very good restaurant, it will only serve cheeseburgers and is loosely based on a fairly famous comedy sketch. The Restaurant class should have a name property, as well as a getter and setter method for the name. The class should also have a stockLevel property, which can be set either by a parameter passed to the constructor, or by calling a special stockUp(int amount) method.

Your Restaurant class will need a method called serveCheeseBurger() method, which reduces the stockLevel by one, and also prints "Cheeseburger!" to the console. The serveCheeseBurger() method should be defined as throwing an OutOfStockException, which it will throw if someone attempts to call serveCheeseBurger() when there's no stock left.

You will need to add the OutOfStockException class to the project yourself. Don't forget that this class needs to extend Exception in order to work!

Add another class to the project containing a main() method. Write the main() method such that it stocks the Restaurant with 5 cheeseburgers, but then tries to serve 6. You'll be forced to use a try and catch block in order to handle the error. Output a suitable message to stderr (System.err.println) if the exception is thrown.

## Task 2: Using the Debugger

Modify the main() method from the previous task so that the Restaurant's stock level is set to a random number between 1 and 10 when you run the program. Add a breakpoint to the Restaurant class's serveCheeseBurger() method, just before it checks the stock level by double clicking in the "gutter", just to the left of the line number in eclipse.

Instead of running your project, click the bug button in eclipse's toolbar. Your program should pause execution when it reaches the line on which you set the breakpoint. Eclipse might ask you to switch to the debugging perspective when this happens, say yes. If you need to get back to the regular Java perspective, click the button at the top right of the eclipse window with a J, as depicted below.

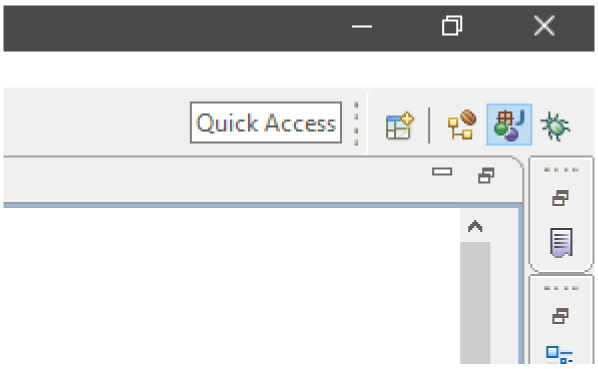


Figure 1: Eclipse Perspective Switcher

Use the debugger to find out the answers to the following questions:

1. What line of code, in which class, did execution pause?

2. What line of code, in which other class, called the serveCheeseBurger() method?

3. How many cheeseburgers were placed in stock randomly?

Once you have answered these questions, go back over one of the programs you wrote in recent weeks, and use the debugger to check on some of the values of variables at various points in the program. Add multiple breakpoints, using the "resume" (play icon) button to advance the program from one breakpoint to the next.

## Task 3: Using a Profiler to Identify Performance Issues

For many years, Java has included a little-known tool named JVisualVM, which is capable of monitoring a running Java program and identifying how much memory is being used, how much CPU time is being used, and pinpointing this usage to the classes and methods responsible. You can use JVisualVM to help tune the performance of programmes that you write, and in some past projects I myself have used the tool to identify areas that need re-writing. I once managed a speed increase of over 30x, although that says more about how bad a job I did of writing the algorithm in the first place than my skill in writing high-performance Java code second time round!

Although JvisualVM doesn't come as standard with the JDK (Java Development Kit) any more, it is still available (without the "J" in the name) at https://visualvm.github.io/. You should simply be able to unzip and run it. If everything is working, you should see the interface as depicted in Fig. 1, below.

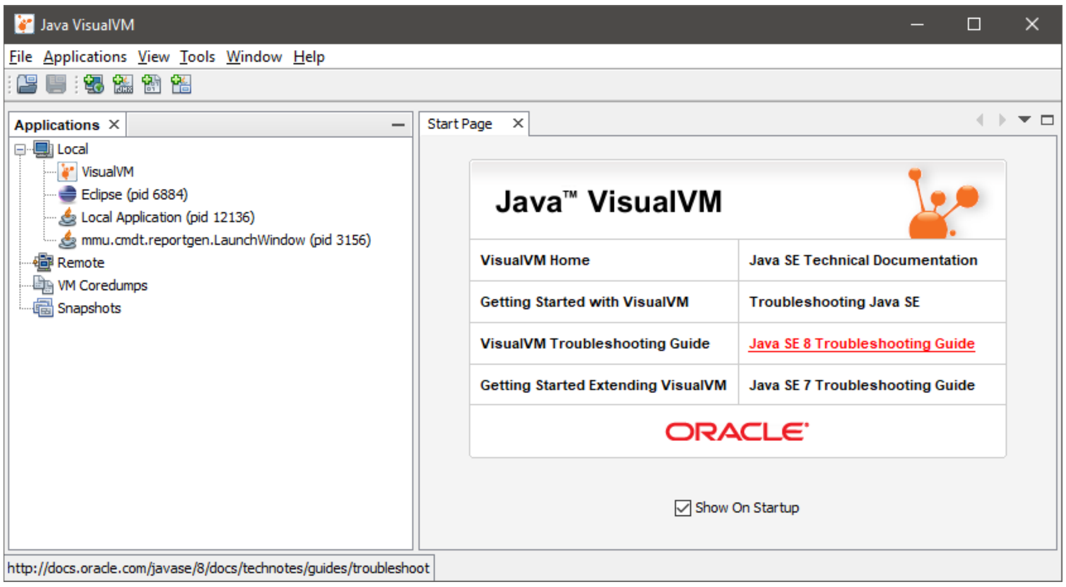

Java VisualVM
File Applications View Tools Window Help
Applications
Local
VisualVM
Eclipse (pid 6884)
Local Application (pid 12136)
mmu.cmdt.reportgen.LaunchWindow (pid 3156)
Remote
VM Coredumps
Snapshots
Start Page
Java™ VisualVM
VisualVM Home
Java SE Technical Documentation
Getting Started with VisualVM
Troubleshooting Java SE
VisualVM Troubleshooting Guide
Java SE 8 Troubleshooting Guide
Getting Started Extending VisualVM
Java SE 7 Troubleshooting Guide
ORACLE
Show On Startup
http://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot

Figure 2: JVisualVM Interface

If you're having difficulty running VisualVM, it might be that your system is configured to run Java programs using a Java Runtime Environment (JRE) rather than a Java Development Kit (JDK). You'll ned to find the location of a JDK on your machine's hard drive, normally in a subfolder of c:\Program Files\Java\, and pass the folder as a "--jdkhome" parameter to the VisualVM program (e.g. by using a shortcut), as depicted below:

```
visualvm.exe --jdkhome "C:\Program Files\Java\jdk1.6.0"
```

If you're able to get VisualVM running, but are having difficulty attaching it to programs running inside of eclipse, it might be that eclipse isn't using the installed JDK either, instead preferring is own copy of the JRE, which it installs at setup. To fix this, you'll need to select and possibly add the JDK in the Window→Preferences dialog, under the *Java* and *Installed JRE* headings.

The list on the left of JVisualVM's interface shows Java programmes currently running. When one of your programmes is running inside eclipse, you should see an entry for it on this list. Unfortunately, you haven't yet written many pieces of software that run for more than a few seconds each. To help with this, I have made an eclipse project available on Moodle that needs to run for much longer.

The StringRosterer project is an example of a Hill Climber, a type of program that is used for generating solutions to problems for which there's no way to produce a perfect answer. Examples of these sorts of problems are the generation of rotas and timetables. Hill climbers work by making random changes to a candidate solution, measuring whether the changes made it better or worse, accepting positive and rejecting negative changes, and trying again many thousands of times. Eventually, they converge on what we hope is a near-optimal solution.

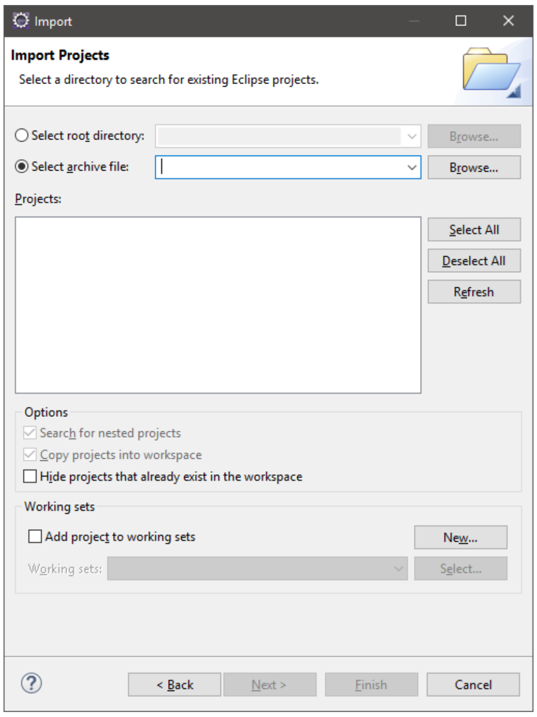
The StringRosterer works by assigning a shift letter ([**E**]arly, [**M**]orning, [**D**]ay, or [**L**]ate) to each employee in the organisation each day. A roster's quality is measured objectively by awarding 10 points of "badness" each time an employee is asked to work a Late followed by an Early, and 5 points for every sequence of 3 Late shifts. It swaps shifts between employees many thousands of times, in the hope of finding a schedule with no points of badness. The program outputs just how close it came to this at the end of its execution, a lower number is better.

This is a very simplified version of a problem that the University has helped businesses in the region to solve. You can easily imagine that if you were to add in minimum staff numbers for particular shifts, staff holidays, etc. that you'd rapidly end up with a wickedly complex problem to solve. You should also be able to imagine just how tough it is to create your University time table each year!

You are going to run the program, and do some profiling in JVisualVM to see what part of the code is consuming the greatest amount of CPU time. Once you have identified the method responsible, try to understand how it works, and why it takes so long.

The code is available on Moodle as an eclipse project, exported as a zip file. You can import it into your eclipse workspace by selecting File → Import, and then selecting "Existing Projects Into Workspace". You will see a dialog similar to that depicted below.

On this dialog, you should make sure "Select archive file" is selected, then browse to the zip you downloaded from Moodle. Make sure that the "StringRosterer" project is ticked, before clicking the "Finish" button at the bottom of the dialog. You will see the project in the left-hand panel in eclipse, along with your previous projects.


Import Projects

Select a directory to search for existing Eclipse projects.

○ Select root directory:
● Select archive file:

Options
☑ Search for nested projects
☑ Copy projects into workspace
☐ Hide projects that already exist in the workspace

Working sets
☐ Add project to working sets

Figure 3: Eclipse Import Projects Dialog

Run the program, and use JVisualVM to profile it as it runs. You can double click the application on the left-hand pane of JVisualVM once it is running, and CPU profiling is relatively self-explanatory. In the results, self-time refers to how much time is spent inside a method, and total time refers to the time spent inside this method and everything else it calls. Thus, all single-threaded programs will show a 100% total time value for their main() method.

## Too Easy? Try this Extension!

If you've identified what it is that makes the StringRosterer slow, what can you do to fix it? The program already measures how much time it takes using the System.currentTimeMillis() method, so you should be able to quickly and easily tell if a change you make had a meaningful impact on execution speed.

## Too Hard?

You are not expected to be an expert on hill climbers, and have an in-depth understanding of what's going on with the rostering program. It serves only as an example program for you to profile. Don't worry if you're finding this side of things difficult.

If you're finding the earlier tasks difficult, then it might be worth doing some additional reading on Java exceptions, the throw and throws keywords, and on the try and catch block in Java.