F29OC Assignment 2

2020-21 Semester 2

1 Introduction

This is an individual project to produce a job dispatcer system using Extrinsic Monitors.

It is worth 18% of a student's F29OC mark.

2 Plagiarism

Code will be run through GitLab code plagiarism checks. Code shared between two or more students will be regarded as plagiarism. Code that is the same as code in the public domain (except for that contained in F29OC lab examples) will be regarded as plagiarism. Students suspected of plagiarism will be subject to standard university plagiarism reporting and assessment procedures. See https://www.hw.ac.uk/students/studies/examinations/plagiarism.htm.

3 Project Development & GitLab Records

While you are working on CW2 we require you to perform multiple commits and pushes (with meaningful commit messages) to your *remote* repository. These should be performed on days that you update the project (both for the `JobDispatcher` class itself, and the associated tests).

The main purpose of this is to provide evidence that the code has been developed incrementally and is the student's own work. Thus, commit messages are required to be meaningful to an external reader, such that they can follow the incremental development of your classes and tests. In addition, changes to the code itself must also show evidence of incremental development and backup the commit messages.

Students may therefore be called to interview if there is not sufficient evidence of incremental development in the GitLab commits of the *remote* repository for their project. Students are required to:

a) Have a working knowledge of *all* code developed or used in the project,
b) Be able to explain the overall development of the project, and
c) Be able to present a detailed evidence and explanation of the incremental development of their code (this must use the commit history of the project).

Students who clearly cannot demonstrate the above will have their mark for Assignment 2 set to zero.

4 Programming Task and Marking (18 marks in total)

You are required to write a `JobDispatcher` class and test it using *your own test code*.

4.1 The GitLab Files

The GitLab stub for CW2 provides you with 3 files:

`JobDispatcher.java`: This is where you will develop your solution. It must implement the interface below and provide the functionality described in this specification.

`Dispatcher.java`: This provides the *interface* that your class must implement. It contains three methods, the functions of which are specified in this file.

`Tests.java:` This contains a very simple example JUnit test. This is where you will develop your own tests to assure yourself that your class performs as per this specification. Failure to provide evidence that you have incrementally developed your tests here means that your project will not be marked.

4.2 Functional Requirements

4.2.1 Overview

- Your `JobDispatcher` must manage a set of *Worker* threads and release them for a set of specified *jobs.*
- It should be implemented using an Extrinsic Monitor.
- There are two types of *Worker* threads: *Compute* threads and *Storage* threads.
- Compute and Storage threads will notify your `JobDispatcher` class that they are available for a job by calling your JobDispatcher's `.queueComputeThread()` and `queueStorageThread()` methods respectively.
- Your will block *Worker* threads until there are enough of them to perform a specified *job*. Once the right combination of threads is available for a job, the `JobDipatcher` will stop blocking the threads (needed for the job) and allow them to proceed.
- Jobs will be specified to your class via your `.specifyJob(nComputeThreads, nStorageThreads)` method.

4.2.2 UR1 - One Job requiring four Compute threads (4 marks)

A instance of your `JobDispatcher` class must be able to accept single job specified by `.specifyJob(4, 0)`, and **then** be able to deal with Compute and Storage threads making themselves available for this job, by calling your `.queueComputeThread()` and `queueStorageThread()` methods respectively.

4.2.3 UR2 - Multiple Jobs (4 marks)

A instance of your `JobDispatcher` class must be able to accept multiple jobs specified by repeated calls to `.specifyJob`, and **then** be able to deal with Compute and Storage threads making themselves available for these jobs, by calling your `.queueComputeThread()` and `queueStorageThread()` methods respectively.

4.2.4 UR3 - Multiple Jobs (4 marks)

As UR2 except that calls to methods `.specifyJob`, `.queueComputeThread()` and `queueStorageThread()`, may occur in any order.

4.2.5 UR4 – FILO Order (4 marks)

As UR3 except that Worker threads must be selected in FILO (First In Last Out) order.

For instance, if, after initialisation:

i. Compute thread `C1` calls, and is blocked by `.queueComputeThread()`, then
ii. Compute thread `C2` calls, and is blocked by `.queueComputeThread()`, and then
iii. Compute thread `C3` calls, and is blocked by `.queueComputeThread()`.

And the above is *followed* by a call to `.specifyJob(2, 0),` then threads `C2` and `C3` will be released to proceed with their execution but thread `C1` will remain blocked.

4.3 The marking Software will be run Four Times

4.3.1 Requirement Marking

Your code will be compiled and marked by automatic marking software. This software will be applied to your project *four* times and you will receive the average of the four runs for each requirement. (Note that in the past, tests like these have not run consistently because a students' code was not *thread safe*).

4.3.2 Consistency of Execution (2 additional marks)

Additionally, a mark of 0.5 will be awarded for each of UR1-4 where a UR's tests run consistently and achieve a score of 2.0 or more out of the 4 marks available for that UR.

4.4 Constraint Requirements

Note: failure to meet any of the constraint requirements specified below will result in your project receiving a mark of zero.

4.4.1 Extrinsic Monitor Classes

You must use the following classes:

```
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.ReentrantLock;
```

No other 'thread safe' classes should be used in `JobDispatcher.java.` In particular, the keyword `synchronization`, and no classes from the package `java.util.concurrent` should be used.

4.4.2 Java SE1.8

Your code must compile with Java SE1.8.

4.4.3 Testing

Note that the marking software will not be available to students. It is your responsibility to design and implement unit tests to assure that your `JobDispatcher` class works as specified here. (This is normal practice in industry.)

Sharing any test code will be considered as plagiarism.

The source code for your tests should be supplied in `Tests.java.`

You may use any classes in Java SE 1.8 for your tests.

4.4.4 Project name, Filenames and Interfaces

- You must **fork** your project from project **F29OC-2019-20-CW2-Coursework** at https://gitlab-student.macs.hw.ac.uk/f29oc-2020-21-students/assignments into your own private and

personal GitLab space (https://gitlab-student.macs.hw.ac.uk/<*your-user-name*>/f29oc-2020-21-cw2-Coursework). This is your ***"remote"* repository.**

- If we cannot uniquely identify and download a single project with the name **f29oc-2020-21-cw2-Coursework** from your private namespace, your project will not be marked.
- You must not change the names of the GitLab project or any of its files.
- You must not add any further files (`JobDispatcher.java` should be self-contained and can include additional private classes written by the student).
- Your `JobDispatcher` class must *implement* the `Dispatcher` interface and have no other public methods.
- You must not change the interface specified in `Dispatcher.java`.
- Your remote repository must show evidence of incremental development through its GitLab commit history (see section 3).

4.4.5 Freezing GitLab

You must not alter your remote GItLab repository after the deadline. We will take a copy of it and this must be the same as the Vision upload.

4.4.6 Vision Upload

In *addition* to the project in your remote GitLab repository described above, you also have to upload a copy of your project to Vision.

Thus, you should download a *zip* file of your project from your remote repository and then upload this to Vision > F29OC > Assignments > CW2, within the deadline. Filename details are in the upload instructions.

It is recommended that a safe copy be uploaded to Vision the day before the deadline.

5 Late submissions

The university's normal late submission policy will apply (30% reduction after 5 working days). Submissions after this late submission deadline will not be marked.

6 Mitigating Circumstances

Students with mitigating circumstances should submit their case via the normal mitigating circumstances procedure and these will be considered by the Mitigating Circumstances board in due course.

7 Feedback

We will aim to give feedback within 3 weeks of the last submission.

8 Project Stub

See GitLab > CW2.