

CSC-20004 Assignment

For this assignment you are provided with an unfinished Java project that is implemented using the NetBeans IDE. Please download the compressed archive of this project called `assignment.zip` as well as the two text files called `example_output_1.txt` and `example_output_2.txt`. Each of the text files contains an example output of the expected completed project.

The project implements a typical workplace scenario in which a number of jobs need to be executed. Each job requires a number of resources for it to be completed. Think of these resources as reusable tools. The resources are all of the same type and there is a limited number of them. A number of workers is also created that can execute the jobs.


A worker can handle a single job at a time but multiple workers can work **concurrently**.

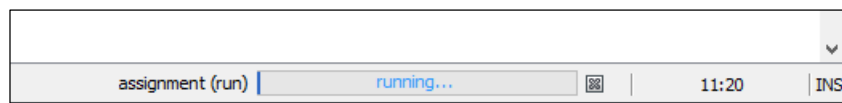
The code supplied to you compiles and you can run the only `main` method in the `Workplace` class.

In the existing code the following are created (see the constructor method of the `Workplace` class):

- (i) a number of jobs that are placed on a job stack
- (ii) a number of resources that are placed on a resource stack and
- (iii) a number of workers.

After the above are created the status of each worker is monitored continuously (in a `while` loop). When all jobs are removed from the job stack and completed the workers will have finished and the `while` loop should exit. The application should then finish by displaying out a report showing the job record of each worker (see example outputs).

As things stand the workers have not started executing any jobs and therefore the `while` loop mentioned above never exits. If you run the program you will therefore need to forcibly terminate it. You can do this by clicking the  to the right of the running process indicator on the bottom right corner of the NetBeans application window:



Part A – Programming Task (50%)

For this part of the assignment you are required to complete the project given to you by implementing the necessary code that will cause the workers to take jobs from the job stack and execute them using the necessary number of resources required for each job (`resourceRequirement`).

Workers must be made to work concurrently but each worker must execute one job at a time.

Execution of a job should be simulated by pausing for the number of milliseconds that the job requires (`timeToComplete`). Execution of a job should not happen by a worker unless they have acquired the necessary number of resources from the job stack.

Once a job has been executed the worker that held it should append their job record with the job's ID and the IDs of all the resources that were used to execute it. Finally, they should return/release the resources used and then they should attempt to acquire the next job from the job stack.

A worker will have finished when the job stack becomes empty.

Once all workers are finished and there are no more jobs left on the job stack a report should be displayed showing the job record of each worker.

Part B – Extended Programming Task (25%)

For the second part of the assignment you are required to adapt your previous program to make calls to a web-based reporting API each time a worker thread completes a job. Complete the `report` method in the `Worker` class. The remote API expects a **get** request that supplies a job and worker parameter, representing the completed job ID, and the worker ID of that worker. The get request should be made to `http://www.scm.keele.ac.uk/staff/stan/app_notify.php`. Use the example below as a guide:

`http://www.scm.keele.ac.uk/staff/stan/app_notify.php?job=42&worker=23`

It will be necessary to research how to use the `URLConnection` and `URL` classes to achieve this part of the objective.

Your `Worker` thread should collect the response sent from the web server (a JSON object) and print out the text of this response to `System.err` (instead of `System.out`). The responses from the web server can be checked by navigating to the above URL using your web browser. In general, they look like this:

```
{"response":"OK", "job":17, "worker":18, "checksum":149}
```

Important

To complete Parts A and B of the project you must only work in the `Worker` and `Workforce` classes and only add code below the lines labelled **UNDER CONSTRUCTION**. Also you are expected to make use of the code that is given to you and you should not remove/change any existing code. You can (if you wish) import any additional packages but you should not need to do that in order to complete either part A or part B of the assignment.

Make sure you fully understand the code that is given to you before you start this assignment.

Part C – Questions (25%)

Once you have completed the programming tasks for this assignment, answer the following questions that relate to your program. The questions can be attempted even if you do not complete Parts A and B but you may find them more challenging to answer.

1. Notice that the `push` and `pop` methods of the `ResourceStack` class are made to acquire (`pop`) and release (`push`) all the resources of a job in one go (i.e. with a single call to each function respectively). Specifically, if the `pop` method was created in such a way so as to remove and return only one resource per call (i.e. if a worker acquires one resource at a time until they have accumulated enough resources for a job) then a **deadlock** could have occurred. Explain why.
2. Explain the purpose of the `resourceStackChangeLock` and `sufficientResourcesCondition` objects used in the `ResourceStack` class. Your answer should relate to this assignment. In other words, you should not give the general definition of `Lock` and `Condition` objects in Java.
3. Notice that the number of resources required for a specific job is the same every time you run your program. You can verify this fact by looking at the two example output files. The reason for this is that a *seed* is used to create the random number generator in the `JobStack` class. For the same reason the time-to-complete for a certain job is also the same in each run of the code. Explain why, each time you run your application a worker is likely to be allocated with different jobs (than those they were allocated in a different/previous run) and (for the same reason) different resources (i.e. with different IDs) are used to complete the certain job. For example, worker 12 in example run 1 executed jobs 32,52,60,89 but the same worker in example run 2 executed jobs 32,43,55,66,87. Similarly, for job 3 (that always needs 3

School of Computing and Mathematics
CSC-20004 Advanced Programming Practices
Assignment

resources to complete), resources 29, 1, 21 were used in run 1 and 27, 5, 12 were used in run 2.

4. The web *server* that responds to the connections for the final part of the assignment must be able to handle multiple requests simultaneously, both from multiple workers in the same program as well as from multiple programs run by different users. Outline how you would construct a simple Java server application that would be able to handle these concurrent requests. Include details of the Java API functionality you would use in order to manage the HTTP connections and ensure concurrent requests were serviced in a timely fashion. Include a flowchart or pseudocode that details how your server application would function, making reference to the API where appropriate.

Deliverables

For Parts A and B, only submit the `Worker` and `Workforce` classes of your project – not your entire project. These are the two files in the `src` directory of your NetBeans project folder called `Worker.java` and `Workforce.java`

For Part C submit a PDF document with your answers to the questions in that part.

The submission deadline for the above is Friday 5 April 2018 at 23:59.

Late submissions by a maximum of one week after the above deadline will be capped to 40% and anything submitted beyond that will have a mark of 0% returned.

Incorrectly submitted work (anything other than 2 .java files and 1 .pdf file) will earn a mark of zero (0%).

If you will require an extension to the above deadline for good reasons, then please submit an exceptional circumstances form to the School.

Finally, please be aware that the university takes plagiarism very seriously. A random selection of submissions will be verified by oral examination (i.e. you might be asked to come in and explain your code and answers verbally). Make sure you know what you are talking about.

If you are uncertain about what constitutes plagiarism, please refer to the relevant section at <https://www.keele.ac.uk/studentacademicconduct/>