Nuber

Cloud and Concurrent Programming

Assignment 2

Version 1.0

Table of Contents

Overview

In this assignment you'll be developing a simulator for Nüber (pronounced Noob-er), an Über[1] clone, to evaluate the impact of new government regulations that have come into effect. New government rules are dictating how many drivers may be active in each area of the city for picking up customers. Whilst there can be an almost unlimited number of drivers ready to drive (999), there is a cap on the number of active jobs that started from the North side of the city (for example). Your task in the assignment is to develop a basic simulator that will allow Nüber to evaluate the impact of these regulations, based on different numbers for the caps – evaluating the impact of a limit of 10 vs 1,000, for example.

You have been given some skeleton code with this specification. Nothing should be added/removed in the AssignmentDriver class, this class should run properly when you have a fully working solution. In order to test your code, you can comment out certain parts of this class to try and ensure you are building the individual classes correctly.

Your program **should never crash**. Given the description of the project provided, your program should allow for multiple bookings to execute concurrently, regardless of how many bookings are added. Your program should naturally **terminate and the main() function return when all bookings are complete**, i.e. your program shouldn't just continue to run in the background.

[1] https://www.uber.com/

Class Breakdown

Whilst the assignment has a number of different classes, your task is to implement the following classes that will handle the actual concurrent processing of bookings:

- Driver.java
- Booking.java
- NuberDispatch.java
- NuberRegion.java

Classes marked with a ** below should not require any changes based on the original solution, however you may modify them if your design requires.

AssignmentDriver**

This class is your "main" class that starts the program. No logic should be present in this class, rather it creates a number of "Simulation" objects to run different scenarios. You can uncomment later parts of the main() function as you get further through the assignment. **You should not need to modify this class other than to uncomment code as you progress, or to add your own testing code.**

Person**

The person class cannot be instantiated on its own. It stores one public string, for the person's name, and a protected int as the maxDelay, both of which are set by it's constructor. The maxDelay is used as to the maximum time that person can cause a thread to sleep. Person has a static function called getRandomName() that returns a name from the list of SAMPLE_NAMES, sequentially from the first index, so the first time it's called it will return Bryan, then Olivia, etc. If it hits the end of the array, it should return to the start of the array.

Passenger**

Passenger inherits from Person (so we can store the person's name and maxSleep), and provides a function called getTravelTime(), that returns a number between 0-maxDelay, for how long that person's travel time will take.

Driver

Driver inherits from Person (so we can store the person's name and maxSleep). It provides a function called pickUpPassenger that takes in a Passenger as an argument, stores it into a private variable, and then sleeps the current thread for a delay between 0-maxDelay milliseconds. The class has another function driveToDestination(), that sleeps the current thread for a delay based on the current passenger's getTravelTime().

NuberDispatch

Dispatch handles everything to do with running an instance of Nuber: adding/providing drivers, booking passengers, and when required, shutting everything down.

It contains a map of the regions it's responsible for, and a central collection of Drivers awaiting a booking. It also has a variable, set by its constructor, for whether the reportEvent() function prints information out to the console. You can use this function to print out debugging information if required.

When a passenger is booked into dispatch using bookPassenger(), dispatch is also told which region the job is in. Dispatch then needs to get that region, and book the passenger into that specific "Nuber Region" using the region's bookPassenger() function.

Drivers are allocated on a first-in, first-out basis. Because passengers are allocated to a specific region, different regions will be accessing a central collection of available drivers from Dispatch.

NuberRegion

Dispatch controls a number of different geographical regions, e.g. one for North and one for South. Each region has a maximum number of bookings that can be running with a driver at any point. If a job for a passenger starts in the North region, it counts towards North's active job count for the entirety of the job, i.e. it doesn't start in the North region's count, and end in the South region. Wherever the job begins, that's where it remains for counting purposes.

When a new region is created, it is given a reference to the Dispatch object, the region's name, as well as the maximum number of simultaneous jobs that this region can perform. You can think of this like having a *pool* of available jobs, and those jobs can be *executed*.

When a region is asked to book a passenger by dispatch, it creates a new Booking object, and adds the booking ready for processing.

Booking

When a booking is made, it's given a reference to the Nuber Dispatch, as well as the passenger that's made the booking, these should be stored as private variables. Each Booking has a globally unique, sequential, job ID. The first job created should be number 1. When a booking is created, the current time should be recorded so we can track how long a booking takes.

When a booking is created, Nuber might not have enough resource available to start that booking immediately (e.g. a region that has no available spots in it's booking pool, or no drivers are available from Dispatch). At some point, the Nuber Region responsible for the booking can start it (has free spot), and calls the Booking.call() to carry out the booking (see the class code for more information).

Booking Result**

BookingResult stores the booking ID, the passenger, driver, and total drip duration as public variables. It has no functions and is only used to return information about a Booking when it's complete.

Simulation**

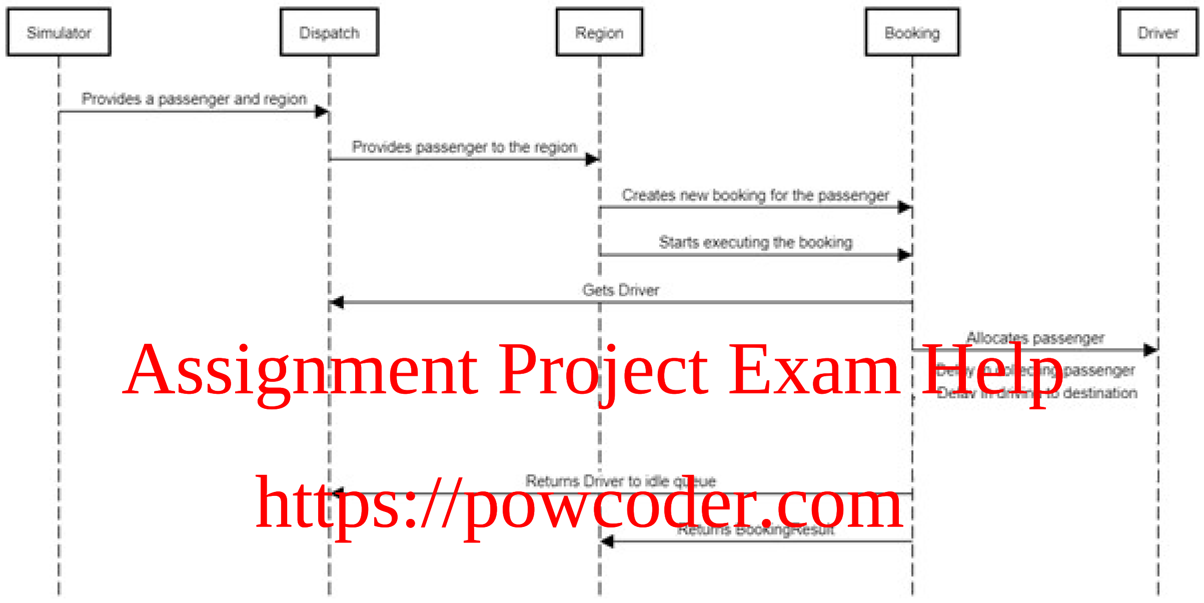
Simulation allows us to easily complete a trial run of the new government policies in a single function call, passing a number of different options for driver and passenger count, maximum timeouts, max bookings per region, etc. The simulation objects allow you to easily run multiple different configurations to both test your code, but more importantly, to evaluate the impact of the new regulations!

You can review sample runs in the Appendix – Sample Output section. Please note that because you will providing your own logging statements, your output will be different from that shown in this document.

System Design

The assignment solution uses the same function names, visibility, and arguments, as those provided to you, however you may need to change whether the function throws certain types of exceptions or is synchronised. You will need to add instance variables to some classes, as well as change the base class of certain class(es) based on your solution. As long as your solution runs **as expected** using the original Driver and Simulator classes, it is an accepted solution.

Figure 1 and Figure 2 below show a high-level system sequence diagram for a passenger's booking, as well as a high level view of how the different classes are encapsulated within each other.



Simulator
Dispatch
Region
Booking
Driver
Provides a passenger and region
Provides passenger to the region
Creates new booking for the passenger
Starts executing the booking
Gets Driver
Allocates passenger
Delay in collecting passenger
Delay in driving to destination
Returns Driver to idle queue
Returns BookingResult

Figure 1 Approximate system sequence diagram for a single booking

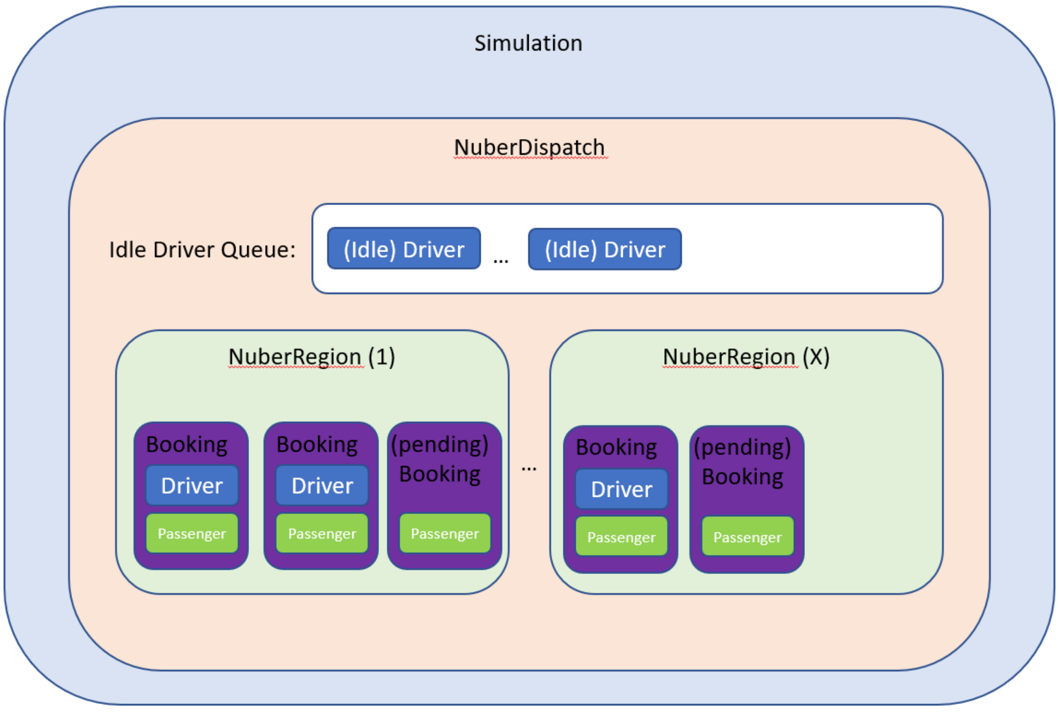


Figure 2 High level conceptual diagram showing how objects are encapsulated within others

Deliverables

The assignment only has a single deliverable of your source code. Please ensure it is submitted correctly as per the Submission and Due Date section.

Marking Scheme

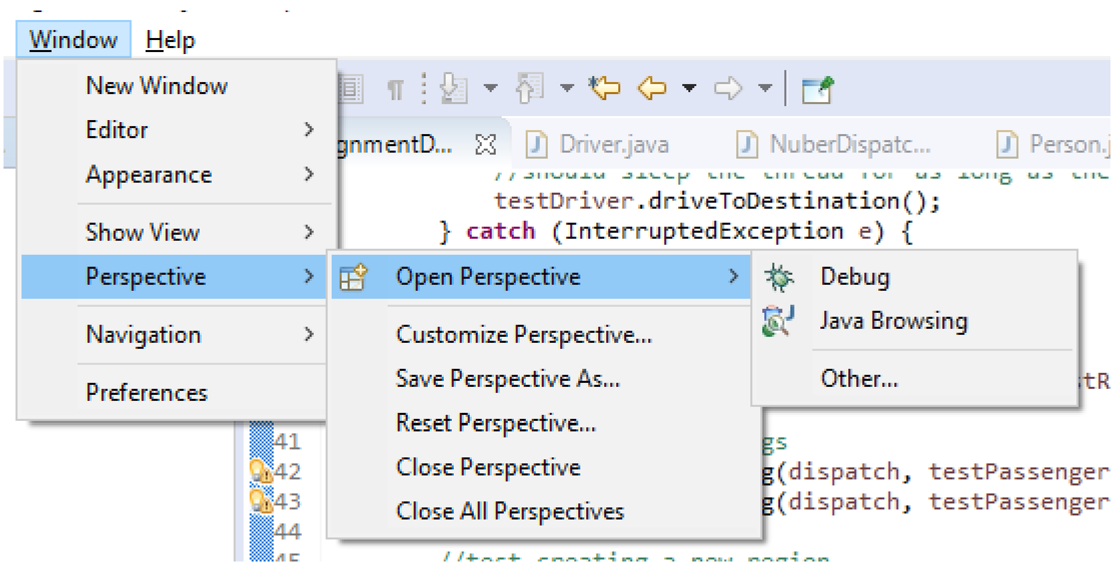
Mark	
10	Driver allocation works with multiple threads adding/requesting drivers
5	The Driver class sleeps as expected for driving to passenger, and delivering passenger
5	Booking IDs are generated correctly and handle cross-thread access
10	Simultaneous live bookings are limited to the maximum allowed for a given region
15	Individual Bookings are created and processed correctly, regardless of timeouts, max thread limitations per region, and driver quantities available
5	Bookings provide a BookingResult when complete
5	Drivers are placed back into the idle driver queue once a booking is completed
5	Program terminates correctly when all jobs are completed
5	Program confirms and runs within the provided driver
5	Dispatch can accurately report on the number of bookings awaiting a driver
20	Manual code review to ensure appropriate design and function (e.g. appropriate use of synchronised functions)
Total: 100	

Important Notes and Plan of Attack

- Part of this assignment is to examine the provided skeleton and identify how it's designed and how your solution can fit within that structure. As long as your solution fits within the provided framework, it is a valid solution. However, please revise the slides, lectures, and practicals to identify the best approaches to use. *Should you manually create threads? Should an object be Runnable or Callable, or neither? What data structures (if any) should be used? When should things block/wait versus returning? How can you leverage existing classes from the standard Java libraries to make things easier for yourself?* **You do not need to create everything from scratch**. The standard libraries are there for a reason!
- Get things to compile first. Add return statements, variables, etc. just to get the program to compile initially.
- Start with the smaller functions, then add bigger/cross-class functionality.
- There will be a point where you need to have multiple things working, but you can't fully test them until they are all either almost working, or full working. This can be frustrating, however try and think about what's happening and why rationally, and then try and test your theory.
- Please add your own test code, or comment our sections of the existing driver. Our own driver will be used for final marking, so your code must work with that.
- Just because your code runs without showing any exceptions, it does not mean none are occurring. When you create additional Runnable/Callable bookings, **they may be failing silently**. Future objects can return a failed result, and still return true for the isDone() function.
- In addition, just because the code runs, it doesn't mean the code is correct. Running the same project with different thread counts, max timeouts, faster CPU, etc. can all potentially impact on deadlocks, collisions, etc. Your solution should run, as expected, regardless of the parameters passed into it, or the speed of the machine it's run on. The assignment driver has some concurrency, but it doesn't test everything, so you need to be confident in your solution as we'll be marking based on whether the solution is correct, not just whether it runs the provided driver.
- You can be too cautious, over-designing your code to handle concurrency where not required can impact the performance of the code, and will negatively impact your marks. Try and ensure you handle the concurrency only where required.
- You can add your own functions if required, however the solution uses those provided. You can modify the return types of functions, their synchronous status, and if they throw any exceptions, however any changes made must still compile and run with the provided driver code.
- If in doubt, ask for help

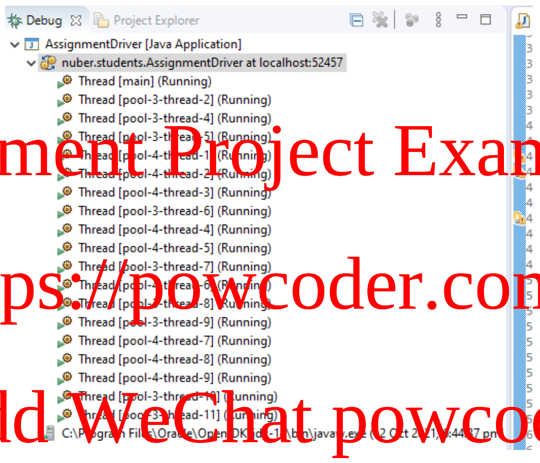
Debugging

Make sure you only have one instance of the program running. One of the problems with threading is that if you're not careful, you might think your program is finished running, but it's actually still running in the background. If you go into the Window menu > Perspective > Open Perspective > Debug (Figure 3), it should show you the Debug panel (Figure 4). The Debug panel is where you can see if you have multiple copies of you program running, select them, and then click the red terminate button in the toolbar. It also lets you see other information when **running in debug mode**, for example all the threads that are live.



```
testDriver.driveToDestination();
} catch (InterruptedException e) {
```

Figure 3 Opening the Debug perspective



```
AssignmentDriver [Java Application]
  nuber.students.AssignmentDriver at localhost:52457
    Thread [main] (Running)
    Thread [pool-3-thread-2] (Running)
    Thread [pool-3-thread-4] (Running)
    Thread [pool-3-thread-5] (Running)
    Thread [pool-4-thread-1] (Running)
    Thread [pool-4-thread-2] (Running)
    Thread [pool-4-thread-3] (Running)
    Thread [pool-3-thread-6] (Running)
    Thread [pool-4-thread-4] (Running)
    Thread [pool-4-thread-5] (Running)
    Thread [pool-3-thread-7] (Running)
    Thread [pool-4-thread-6] (Running)
    Thread [pool-3-thread-8] (Running)
    Thread [pool-3-thread-9] (Running)
    Thread [pool-4-thread-7] (Running)
    Thread [pool-4-thread-8] (Running)
    Thread [pool-4-thread-9] (Running)
    Thread [pool-3-thread-10] (Running)
    Thread [pool-3-thread-11] (Running)
  C:\Program Files\Oracle\OpenJDK\jdk-1[illegible]\bin\javaw.exe ([illegible] Oct 2021, [illegible]:44:[illegible] pm
```

Figure 4 Viewing all the threads for the current debug run of the program

Getting Help

You can work on the assignment during your practical once you have completed that week's practical tasks. Consulting times are available in person or online on Mondays, including for external students (please check the course site for the latest times and locations). If you can't make the consulting times, please contact Will (will.meakin@unisa.edu.au) in the first instance to organise another time.

In addition to consulting, there's the course forums and Slack that are monitored.

If you have questions about the structure of the existing program, please don't be afraid to ask, the worst outcome is we'll let you know that we can't answer that question!

Submission and Due Date

Please upload your assignment’s zipped “src” directory ONLY. You do not need to submit any other folders or files.

Submissions that are not submitted as above, **additional directories outside of src, non-Zipped formats, etc. will lose marks for not following the instructions.**

Due Date

Please see the course site for submission due date.

Late Policy and Extensions

Unless you have an approved extension for the assignment, there is no late policy. **This means late submission, regardless of the time, will result in 0 marks.**

To make sure that you have a backup, and to ensure you don’t miss the deadline, **please upload versions of your submission regularly to the submission page**. You can update your submission as many times as you would like until the due date. Please be aware that we can only mark what is submitted on the course site.

If you have a legitimate reason and can provide required documentation for an extension, I am more than happy to grant an extension if required. **Please be aware that we have had instances recently of students submitting fake and altered medical certificates.** Any certificate’s authenticity may be checked by us, should you request an extension.

Leaving the assignment until the last minute is not a reason for an extension, nor is workload or work commitments. By enrolling in the course, you have made a commitment to *spend time each week* on developing yourself through the course materials. You are expected to commence this assignment well before the due date.

Your progress on the assignment will be taken into account should you request any last-minute extensions.

Academic Integrity

Please note that this is NOT a group assignment. Whilst you can discuss the problems in the assignment at a high level with others, your specific implementation should reflect your own thoughts. All content in the assignment that you submit should be your own.

Plagiarism and academic integrity checks will be run on submissions, so please ensure that your submission is your own. At the end of the day, academic integrity will impact you the most if you finish your degree without the knowledge and skills that you claim to have demonstrated.

We are seeing significant increases in academic integrity issues and cheating in general, so we are monitoring both submissions and the general interwebs for cheating. We have caught students posting anonymously on online forums prior and identified them, so please don't take the risk.

If you are unsure about whether you can use something, or how to reference something, please contact a staff member and we can advise. **There is no penalty for asking.**

Appendix – Sample Output

Please note that this output only reflects the sample solution based on the additional logging that was added. Your output will vary based on when and where you place your log commands, as well as the final parameters for the simulation, and the real time scheduling by the CPU.

Sample run with debug out

The following is an example run for

```
      new Simulation(regions, 5, 10, 1000, true);
```

```
Creating Nuber Dispatch
Creating 2 regions
Creating Nuber region for South
Creating Nuber region for North
Done creating 2 regions
1:null:null: Creating booking
2:null:null: Creating booking
3:null:null: Creating booking
4:null:null: Creating booking
1:null:P-Harold: Starting booking, getting driver
3:null:P-Jerry: Starting booking, getting driver
5:null:null: Creating booking
2:null:P-Isabella: Starting booking, getting driver
4:null:P-Stephen: Starting booking, getting driver
4:D-Olivia:P-Stephen: Starting, on way to passenger
3:D-Kenneth:P-Jerry: Starting, on way to passenger
2:D-Vincent:P-Isabella: Starting, on way to passenger
5:null:P-Larry: Starting booking, getting driver
6:null:null: Creating booking
1:D-Debra:P-Harold: Starting, on way to passenger
5:D-Jack:P-Larry: Starting, on way to passenger
7:null:null: Creating booking
6:null:P-Ruth: Starting booking, getting driver
8:null:null: Creating booking
7:null:P-Diane: Starting booking, getting driver
9:null:null: Creating booking
8:null:P-Gerald: Starting booking, getting driver
10:null:null: Creating booking
9:null:P-Brandon: Starting booking, getting driver
10:null:P-Virginia: Starting booking, getting driver
11:null:null: Creating booking
11:null:Test: Rejected booking
Active bookings: 10, pending: 6
5:D-Jack:P-Larry: Collected passenger, on way to destination
3:D-Kenneth:P-Jerry: Collected passenger, on way to destination
5:D-Jack:P-Larry: At destination, driver is now free
6:D-Jack:P-Ruth: Starting, on way to passenger
2:D-Vincent:P-Isabella: Collected passenger, on way to destination
4:D-Olivia:P-Stephen: Collected passenger, on way to destination
1:D-Debra:P-Harold: Collected passenger, on way to destination
Active bookings: 9, pending: 5
4:D-Olivia:P-Stephen: At destination, driver is now free
10:D-Olivia:P-Virginia: Starting, on way to passenger
3:D-Kenneth:P-Jerry: At destination, driver is now free
8:D-Kenneth:P-Gerald: Starting, on way to passenger
2:D-Vincent:P-Isabella: At destination, driver is now free
1:D-Debra:P-Harold: At destination, driver is now free
```

```
7:D-Debra:P-Diane: Starting, on way to passenger
9:D-Vincent:P-Brandon: Starting, on way to passenger
6:D-Jack:P-Ruth: Collected passenger, on way to destination
10:D-Olivia:P-Virginia: Collected passenger, on way to destination
10:D-Olivia:P-Virginia: At destination, driver is now free
7:D-Debra:P-Diane: Collected passenger, on way to destination
6:D-Jack:P-Ruth: At destination, driver is now free
Active bookings: 3, pending: 1
8:D-Kenneth:P-Gerald: Collected passenger, on way to destination
9:D-Vincent:P-Brandon: Collected passenger, on way to destination
9:D-Vincent:P-Brandon: At destination, driver is now free
7:D-Debra:P-Diane: At destination, driver is now free
8:D-Kenneth:P-Gerald: At destination, driver is now free
Active bookings: 0, pending: 1
Simulation complete in 4036ms
```

Sample run without debug out

The following is an example run for

```
      new Simulation(regions, 5, 10, 1000, false);
```

```
Creating Nuber Dispatch
Creating 2 regions
Creating Nuber region for South
Creating Nuber region for North
Done creating 2 regions
Active bookings: 10, pending: 6
Active bookings: 8, pending: 4
Active bookings: 2, pending: 1
Active bookings: 0, pending: 1
Simulation complete in 4040ms
```