DCU School of Computing
Assignment Submission

Student Name(s):	Bernard O'Connor, Cillian Mc Neill
Student Number(s):	14367821 , 14352621
Programme:	BSc in Computer Applications
Project Title:	Assignment 1: University CarPark
Module code:	CA4006
Lecturer:	Dr. Martin Crane
Project Due Date:	26/03/2018

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying is a grave and serious offence in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion, or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references.

I have not copied or paraphrased an extract of any length from any source without identifying the source and using quotation marks as appropriate. Any images, audio recordings, video or other materials have likewise been originated and produced by me or are fully acknowledged and identified.

This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. I have read and understood the referencing guidelines found at http://www.library.dcu.ie/citing&refguide08.pdf and/or recommended in the assignment guidelines.

I understand that I may be required to discuss with the module lecturer/s the contents of this submission.

I/me/my incorporates [✓ we/us/our] in the case of group work, which is signed by all of us.

Signed: Bernard O'Connor

2018 CA4006 University CarPark Report

Authors

- Cillian Mc Neill: 14352621
- Bernard O’Connor: 14367821

How To Run

- cd to the directory with Main.java
- javac Main.java
- java Main

Features Implemented / Represented

See last page for Class Diagram.

Time to Travel Through Automatic Barriers

As per the assignment spec, we have three sets of entrances and three sets of exits. We use fair Semaphores with 3 permits each to control the threads accessing these entrances / exits. We set the fairness to true to represent the drivers queueing in an orderly fashion.

We use a final constant, BARRIER_TRAVEL_TIME, as the minimum amount of time it takes a driver to pass through a barrier (using a sleep).

Drivers Delayed While Trying to Exit

For when a car is exiting the CarPark, we simulate the driver occasionally taking a longer than normal amount of time to exit, to simulate them having an issue and possibly having to walk to the security office to get it fixed. The calculateTimeToLeaveCarPark(Car car) function checks the car’s “isUnlucky” condition, which is randomly set for each car in the Car Class. If a car is “unlucky” then they take a random multiple of BARRIER_TRAVEL_TIME (up to 10 times) to exit the car park.

The drivers who got delayed are represented in the “Delayed Exit” bar, the drivers who weren’t delayed are represented in the “Undelayed Exit bar”.

Drivers Double Parking

To represent drivers occasionally double parking, i.e a car taking two spaces instead of one, we randomly set a variable in the Car Class to either take up one space or two spaces, with a weighting in favour of one space.

When a driver goes to take a space, they must acquire the appropriate amount of permits (the number of spaces they are taking) from the Semaphore “spacesSem”.

Drivers Searching For A Spot

We keep track of all drivers who have entered the CarPark and are currently searching for a parking space. We increment the atomic integer tracking the count as they enter, and until they can get a spot (by acquiring the Semaphore permit mentioned above), they are considered to be actively looking for a spot.

Impatient Drivers Giving Up on Search And Leaving (to go to another CarPark for example)

We have decided to add our own use case to the assignment. We wanted to simulate drivers getting impatient while searching for a spot for a certain amount of time, and just give up and leave instead of parking. We achieve this in “takeSpace(Car car)” with a tryAcquire “spacesSem” for that has the amount of permits they want (car.getSpace), and a final constant FED_UP_TIME in milliseconds. If the driver can’t get all the permits they need before the FED_UP_TIME expires, then they leave the CarPark without parking.

We represent these drivers in the “Total Fed Up” bar.

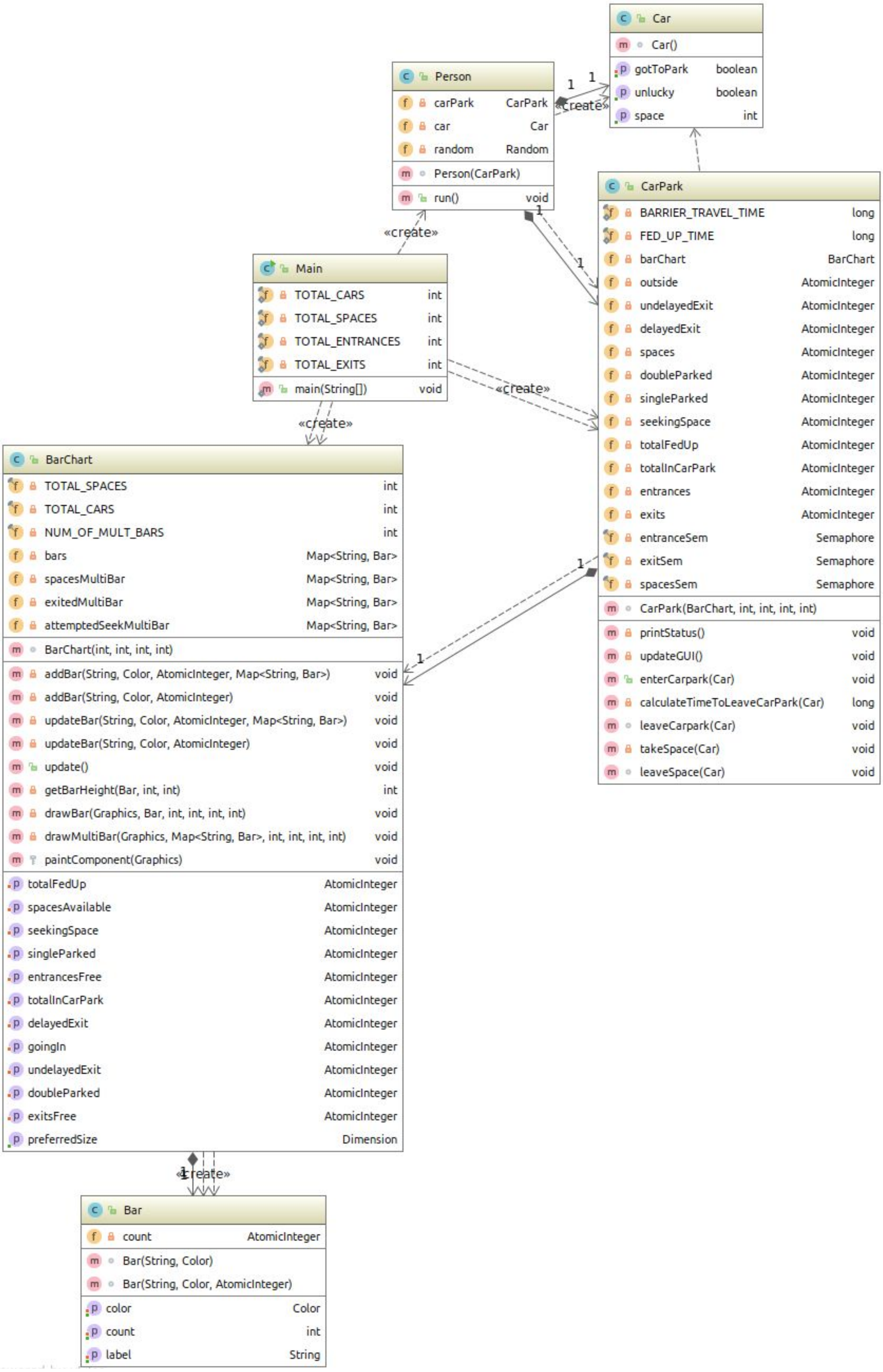
A Count of Current Total of All Drivers in the CarPark

We actively keep a track of every driver who is currently in the CarPark, whether they are searching for a spot, have single or double parked, have parked and now are trying to exit, or who have given up trying to a spot and are trying to exit.
This is represented in “Total In CarPark” bar.

Use Atomic Integers to track the values for the Bars

We utilize atomic integers to keep track of the numeric values of the bars to help safeguard to allow us to atomically increment and decrement the values. We discovered the hard way that the atomic integer set() method is not atomic, instead you must iteratively call the increment / decrement methods if you need to change the value by more than 1;

Class Diagram:



Car
Car()
gotToPark boolean
unlucky boolean
space int
Person
carPark CarPark
car Car
random Random
Person(CarPark)
run() void
1
1
«create»
CarPark
BARRIER_TRAVEL_TIME long
FED_UP_TIME long
barChart BarChart
outside AtomicInteger
undelayedExit AtomicInteger
delayedExit AtomicInteger
spaces AtomicInteger
doubleParked AtomicInteger
singleParked AtomicInteger
seekingSpace AtomicInteger
totalFedUp AtomicInteger
totalInCarPark AtomicInteger
entrances AtomicInteger
exits AtomicInteger
entranceSem Semaphore
exitSem Semaphore
spacesSem Semaphore
CarPark(BarChart, int, int, int, int)
printStatus() void
updateGUI() void
enterCarpark(Car) void
calculateTimeToLeaveCarPark(Car) long
leaveCarpark(Car) void
takeSpace(Car) void
leaveSpace(Car) void
«create»
1
1
Main
TOTAL_CARS int
TOTAL_SPACES int
TOTAL_ENTRANCES int
TOTAL_EXITS int
main(String[]) void
«create»
«create»
1
1
BarChart
TOTAL_SPACES int
TOTAL_CARS int
NUM_OF_MULT_BARS int
bars Map<String, Bar>
spacesMultiBar Map<String, Bar>
exitedMultiBar Map<String, Bar>
attemptedSeekMultiBar Map<String, Bar>
BarChart(int, int, int, int)
addBar(String, Color, AtomicInteger, Map<String, Bar>) void
addBar(String, Color, AtomicInteger) void
updateBar(String, Color, AtomicInteger, Map<String, Bar>) void
updateBar(String, Color, AtomicInteger) void
update() void
getBarHeight(Bar, int, int) int
drawBar(Graphics, Bar, int, int, int, int) void
drawMultiBar(Graphics, Map<String, Bar>, int, int, int, int) void
paintComponent(Graphics) void
totalFedUp AtomicInteger
spacesAvailable AtomicInteger
seekingSpace AtomicInteger
singleParked AtomicInteger
entrancesFree AtomicInteger
totalInCarPark AtomicInteger
delayedExit AtomicInteger
goingIn AtomicInteger
undelayedExit AtomicInteger
doubleParked AtomicInteger
exitsFree AtomicInteger
preferredSize Dimension
1
«create»
Bar
count AtomicInteger
Bar(String, Color)
Bar(String, Color, AtomicInteger)
color Color
count int
label String