INB370 Assignment 2 FAQ

Evidently this document is a collection of questions and answers to help people out. Some of these questions have been over-asked and thus over-answered on FB and elsewhere. This is a collection and summary. There is a small amount of new material here, so all should have a browse. In general, I have stolen the student question and my answer, and done some minor editing. I have also grouped according to category.

UNIT TESTING

Q: So I've just got a general question about unit testing in general. For the first assignment I missed quite a few Actions tests, and I noticed in the JUnit logs there the markers had a lot of "OffByOne" tests, of which I had none (explains why I lost quite a few marks here!).

What are these off by one tests doing/testing for, exactly? For example: "badActions_Fails_Constructor_DefaultRelease_OffByOne"?

A: Off by one errors are one side of the boundary cases. Classical example is a for loop that should iterate 10 times but actually runs 9 times because you get the condition wrong. So, if you have a condition – and this may apply to situations other than loops - and you test the last time it is satisfied and the first time it isn't, you should have it covered. Check with someone at a prac if you aren't sure.

Q: For the assignment 2 tests are we just supposed to fill out the ones provided in the code drop or are we supposed to do them and add more?

A: Code drop test cases were auto generated. Use your own, create many more, and feel free to change the signature of the test methods.

Q: Should I do test cases for the finalState, initialState and getStatus methods?

*A: Some general comments: Unit tests are seldom provided for getters, setters and toString methods. finalState, initialState and getStatus are really toString methods in disguise. So no, tests for these methods are not required. My own practice, BTW, is to include stubs for *all* methods and for trivial methods to insert an AssertTrue(true). The idea here is that it documents the fact that I have looked at the method and made a decision not to test it - so there is no confusion. Again, if in doubt, check with one of us.*

STATES AND STATE TRANSITIONS

An opening comment: CarPark handles the physical shifting of vehicles. Vehicle and its subclasses manage the internal state of a vehicle. That means there is a division of labour. But I have also managed the states that exist:

- *New, Queued, Parked, Archived*

Now, the possible transitions are:

- *N>Q. N>P, N>A*
- *Q>P, Q>A*
- *P>A*

Of these states, N is a simple, momentary state, and the vehicle must leave it immediately. The Archive state is obviously final. So for that reason, we have only really to manage the Queued and Parked states explicitly.

Q: I'm having quite a bit of trouble with one small thing in the CarPark class. It seems so straight forward, yet it's got me caught. In the instance that a new vehicle is created and the carpark is full and so is the queue the vehicle is immediately turned away. Therefore it must go from a state of N -> A.

What I can't work out though is how to do this? There doesnt appear to be any relevant method in vehicle or carpark. Seems to me like there needs to be (or should be) a public method in Vehicle called archiveNewVehicle(). But obviously there isnt.

A: N is a momentary state. Vehicles never stay in that state and never return to it. So, there are no methods to make that transition as such. Archiving is a final operation, from which there is no return. The explicit methods in Vehicle are therefore about queue and park. The method archiveNewVehicle in CarPark handles the mechanics, but you need to realise that while we set an arrival time, we don't ever record a parking time or an exit time or a queue entry time. That gives all the info we could ever want, and we can use that to assess anything about state and satisfaction.

Q1: If a vehicle parks, is the vehicle always 'satisfied'? I would assume that once a vehicle parks, they're satisfied, but the javadoc sounds like they still might not be satisfied if they park? If so, what could cause them to be unsatisfied once they've park?

Q2: I'm debating with some fellow students and Lawrence about what states of satisfaction the Vehicles are in before parking. Can you please clarify for us if the vehicle is satisfied when new, or waiting in the queue? The way the spec reads can indicate vehicles are dissatisfied until parked, or they could possibly be satisfied until turned away or queueing too long

A: The javadoc is as below, which says that we are certainly satisfied once parked. We are also clearly dissatisfied if we are turned away or queued too long. But the state while queuing patiently – and for the moment when you have a new vehicle - are indeterminate in the spec. Both interpretations are reasonable, but we need to make a call to enable testing. So, for consistency with the way this is written, I will say that the vehicle is satisfied until it is dissatisfied – so a call while it is in a queue but not parked should return satisfied. I will make a minor adjustment to the javadoc to reflect this.

```
/**
 * Boolean status indicating whether customer is satisfied or not
 * Satisfied if they park; dissatisfied if turned away, or queuing for
 *  too long
 * Note that calls to this method may not reflect final status
 * @return true if satisfied, false if never in parked state or
 * if queuing time exceeds max allowable
 */
```

EXCEPTIONS:

Q: So in the documentation for some of the exceptions thrown for a particular method, are they talking about exceptions thrown by that method, or ones thrown and propagated upwards from a method that it calls?

A: *Check the wording. If there is no information really (timing constraints etc) then it is propagating up from below. If there is a clear statement, then you probably have to throw it yourself. If in doubt, check with us.*

For example:

```
/**
 * Archive vehicles which have stayed in the queue too long
 * @param time int holding current simulation time
 * @throws VehicleException if one or more vehicles not in the correct
 *         state or if timing constraints are violated
 */
public void archiveQueueFailures(int time) throws VehicleException
```

In this one, the wording is vague and we are propagating from below.

```
/**
 * Method to archive new vehicles that don't get parked or queued and
  * are turned away
 * @param v Vehicle to be archived
 * @throws SimulationException if vehicle is currently queued or parked
 */
public void archiveNewVehicle(Vehicle v) throws SimulationException
```

This is explicit and requires a throw. Note the difference between queued, which means physically in the queue, and in a queued state, which means that the internal state of the vehicle is set that way.

Q: A question concerning the archiveDepartingVehicles method for the exception:

```
@throws SimulationException if one or more departing vehicles
are not in the car park when operation applied
```

My interpretation is:

```
inCarPark(v) == v.isParked() || v.isQueued()
```

Is this correct, or is this a different case?

A: As noted above, there is a split in responsibilities between CarPark – which handles the physical movement of the Vehicles, and Vehicle and its subclasses, which handle the internal state. Mostly, Simulation exceptions come from the CarPark methods, but a small number quite reasonably emerge from the Vehicle classes. I have tried as far as possible to make this clear from the language. If I say "in a queued state" that means it is a Vehicle responsibility. Here, "not in the car park" means not physically in the car park data structure, despite all our beliefs to the contrary. In my implementation, this exception is actually thrown from a call to unparkVehicle, which checks whether v is physically in the data structure. How you throw this will depend on how you implement the method, but it won't be based on the internal Vehicle state.

Q: There seems to be a contradiction in the JavaDoc for the Vehicle classes between the constructor and the method enterParkedState. The declarations are:

```
 Vehicle():@throws VehicleException if arrivalTime is <= 0

enterParkedState(int parkingTime, int Duration):
@throwsVehicleException if [stuff deleted] parkingTime < 0
```

Now, the parkingTime is the minute at which the vehicle was able to park and the simulation requires that a vehicle cannot arrive before time=1. enterParkedState allows parking at time=0. How do you park at time=0 if you can't arrive before time=1?

A: For consistency, we should either have arrivalTime <0 and parkingTime <0 or both <= 0. What we presently have is inconsistent as pointed out in the question. There is an issue of whether you leave open the option of parking at time 0 and leave that control to the other classes , but I think the best thing to do is to make both <=0. So, Vehicle() remains the same; enterParkedState is revised as follows:

```
@throwsVehicleException if parkingTime <= 0
```

The actual javadoc has been updated.

THE ARGUMENTS AND THE MAIN PROGRAM

Q: Do we call the GUI from the existing main program? Or do we develop another main?

A: The GUISimulator is in essence a GUISimulationRunner, though that name became a bit cumbersome. So I see the existing main program as a step toward the new one, with the command line arguments being part of that. See the spec part 2 for some discussion of the role of those command line arguments as starting values for the GUI.

GUI PROGRAMMING:

Q: Am I permitted to use anonymous action listeners in the GUI? For instance, I prefer using:

```
button.addActionListener(new ActionListener() {
     public void actionPerformed(ActionEvent event) {
     //Code here..
     }
});
```

Rather than having extra inner classes that extend ActionListener.

A: Yes, this style is fine. But refactor common code into private methods where possible.

THE CARPARK, SMALL CAR AND MOTORCYCLE SPACES

Q: Hi, I would like to clarify something regarding the vehicles taking parks that are not allocated specifically to them (like a motorcycle in small car parking space).

Does the class keep track of whether the vehicle takes a park designated for itself or another available (eg, if a small car is in a small car park or a normal car park, or a motorcycle is in a motorcycle spot or a small car park)? How should we keep track of which parks are taken?

A: This one is covered in the opening to the lecture from week 10. Have a good look at it. The main issue is that we simplify. Assume for a moment that all the small car spaces are taken, say 20 of them, but that there are 25 small cars in the car park. That means 5 of the normal car spaces are taken. If there are say, 55 normal cars in the car park, then we might then have 70-55-5 = 10 general car park spaces available, if 70 were the maximum. Let's assume that a small car leaves. It might have been in a small car space; it might have been in a general space. But we assume the following:

- *There are now 4 small cars in general spaces*
- *There are 11 general spaces now available*
- *All 20 small car spaces are still all taken*

And we keep track of these things by keeping track of the totals. The same argument is made for the motorcycles in small car spaces. So you never explicitly model the individual parking space.

SOURCE CONTROL

Q: Hey so my partner and I are trying to set up the git version control, are we using git hub? and if so is there a simple way to set up a 2 person project?

A: People in the class are in general using github or bitbucket for their remote repository. Both provide guides for new users. The bitbucket intro doc is especially good - amongst the best in the industry. The relevant URLS are as follows:

- *https://help.github.com/*
- *https://confluence.atlassian.com/display/BITBUCKET/Bitbucket+101*

Note that egit has a commit and push button which will help you manage things.