The aim of this assignment is to develop Christchurch airport parking lots, and to do so, I had considered the following points as my beginning options.

My source code consists of three packages (airport, airport.gui, junit) to specify each different purpose. Each parking lot has the same parking lot structure, and their methods were implemented from their interface ***Parkable***. However, the parking fee for each parking lot is going to be charged by different rates. To do so, joda time and joda money have been added to the build path. I used ArrayLists in my *ParkingSystem* whenever a collection was necessary. Using ArrayLists allows to store new objects freely to the Collection without having to resize its size each time whenever new items are added to the list. However, ArrayLists do not work with primitive types, which could be inconvenient since I needed to assign data types for each time I need the arraylists. Also the contains method for *ArrayLists* takes O(n) since it uses sequential search, which is time consuming. Therefore, if the number of *parkinglots* and vehicles are too large to inspect, hashMap can be used instead because hashMap takes O(1).

Vehicle
In 'Vehicle' class, I have two constructors, one with the parameters: java.lang.String regNo, BillableEntity owner' the other has the parameter java.lang.String regNo. To use both constructors, I have designed boolean hasOwner() to check if the 'owner' has been known, otherwise the constructor with the parameter regNo will be used. Having two construtors is useful as I can differentiate between the case when the owner was known and the case when the car registration number was known. Then toString() will print out the registration number; or the registration number, customer's name and address only if we know the customer's name and address. Also, overloading equals(Object o) is necessary to check if two vehicle objects are identical. Otherwise, the equals method has the behaviour of == and thus will only compare memory locations.

Parking System
The parkOptions() method will be called to show which parking lot options are available. I created the SimpleWelcome JFrame as my first page, and showed the parking lots name and brief description for each type of parking lot. The inLot method will be called whenever admit button is pressed, and this will prevent any car from being duplicating in each parking lot, so we have no identical cars in different parking lots at the same time. Collection lots() returns Collections,unmodifiableCollection(parkingLots). This is because the parkingLots is read- only and unmodifiable, so they cannot be modified by strangers. vehicleFor checks if a current car is already been seen by parking system. Otherwise, the car is added to the list of seen vehicles. This allows me to keep track on the vehicles that have ever seen before.

Parking Lots
All the parking lots have the same methods since these are all implementing the interface Parkable. Properties for the parking lots are initialised, and a Constructor takes the name and initialCapacity as its parameters. In each 'Money computeCharge 'method, I used the 'withZoneRetainFields' method to find the millisecond time in another timezone. I changed the timezone to UTC, which does not have the daylight saving because CIAL parking calculator does not use daylight saving.I converted the interval with milliseconds since it is the smallest time unit so that it is easy to change to higher units such as second or minute. I used car.equals() in the admit() method to check if any car with the same registration number exists in the same parking lot. To use equals(Vehicle v),I overload equals() method in Vehicle. The toString() method simply returns the name of each parking lot, this allows me to print the name only instead of instead of the default object string, <class_name>@<memory_location>.

Advisor & ParkSmart
Advisor was made to find the cheapest parking lot In the 'Advisor', I have a 'compareMoney' method to store parking fee for each parking lot, and this method will be called in updateParkSmart() method in the 'ParkSmart' class. HashMap was used to store each parking lot fee because hashMap stores both the key and value, and the keys will be the name of the parking lot and the values will be the fee of each parking lot. Then, the HashMap will be accessed in updateParkSmart(). Since Money in Map<String, Money> is not

comparable(The datatype money is comparable), I converted the datatype Money into int by calling getAmountMajorInt. Then, I added each parking fee which matches the key by calling the get(key) method, and the values will be added to ArrayList<Integer> arr which I created beforehand. I highlighted the cheapest parking lot by using swing HighLighter. Collections.min method return the integer minimum value in the Collection. So, I converted the integer value to money and then checked if the minimum charge is the same as any parking fee displayed on JTextField, then the minimum value will be highlighted.

BillableEntity & FreeTracker

In the 'BillableEntity', I have a toString() method to print a customer name and address when the customer was asked their information. To have the information, I used JOptionPane.showInputDialog in the SimpleStatusTranscriptPanel, so we can have customers' information whenever a car is admitted. The FreeTracker class is to allow the calculator and transcript to work together. Instead of a checkout button, a 'Paid' button has been added to checkout only if a parking fee has been charged. The clearButton clear the transcript.

Junit test

Each parking lot has two kinds of tests. ParkinglotTestCompute is to test the computerCharge method because there are many different possibilities for parking fee. I made private DateTime from so the departure time is always a fixed value and the parking fee changes whenever any new DateTime to is assigned. Then, the 'DateTime to' was created for each different arrival time case to assert the Money expected and Money actual(lot.computeCharege(from,to)) are equals(assertEquals(expected,actual)). Similarly ,the ParkingLotTest is to test the rest of the methods.

Gui

I added three new Gui classes, FreeTracker, NosyParker and ParkSmart. Also, all my GUI classes use keyListener to enable keyboard input to work as same as the mouse click on buttons. Since the keylistener for the buttons is only enabled when the cursor is in registration text field, I set the default frame focus to that text field.

- **SimpleWelcome**

The SimpleWelcome class is the main JFrame to welcome customers, and it shows which parking lots we have and also a brief description for each parking lot. The start button brings up the NosyParker class, and the NosyParker class has all the other Panels. The exitButton quits the program but I used showConfirmDialog to confirm once more before the program is shut down.The parking lot types on the welcome page are coming from the parkOption method.