Fundamentals of Computing with Java:

# Role Player Game

Contents

[Role Player Game 1](#_Toc480990240)

[Write functioning java code showing understanding of core concepts 2](#_Toc480990241)

[Effectively design OO classes and interfaces, for real world problems 3](#_Toc480990242)

[Design code selecting appropriate visibility for class members, re-using code where appropriate 5](#_Toc480990243)

[Make use of external libraries where appropriate, and explain the rationale for doing so 6](#_Toc480990244)

[Test code by developing simple automated tests, and explain the rationale for doing so 7](#_Toc480990245)

[Android Client Project – Museum App – Team 14 8](#_Toc480990246)

[Write functioning java code showing understanding of core concepts 8](#_Toc480990247)

[Effectively design OO classes and interfaces, for real world problems 10](#_Toc480990248)

[Design code selecting appropriate visibility for class members, re-using code where appropriate 12](#_Toc480990249)

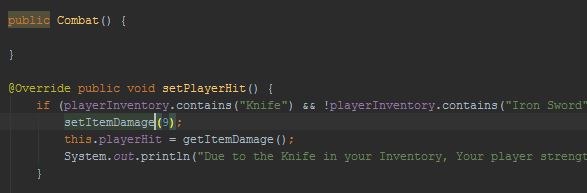
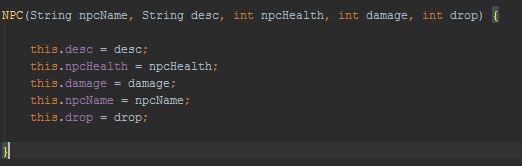
[Make use of external libraries where appropriate, and explain the rationale for doing so 14](#_Toc480990250)

[Test code by developing simple automated tests, and explain the rationale for doing so 16](#_Toc480990251)

[References 17](#_Toc480990252)

# Write functioning java code showing understanding of core concepts

I covered many of the core concepts of Java when creating the RPG. For example, I used many different data types and made full use of methods, constructors, operators, Scanners, if statements, for loops and more. I will provide some examples of this:

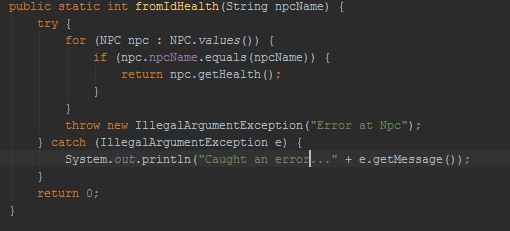
In almost every class I took use of constructors to create my objects. I used both empty constructors and those with parameters to good effect, shown below. I would use empty constructors when I didn’t need to specify the characteristics of the object and just access its methods.

C:\Users\Jack\AppData\Local\Microsoft\Windows\INetCache\Content.Word\arraylist.jpgC:\Users\Jack\AppData\Local\Microsoft\Windows\INetCache\Content.Word\collection.jpgI tried to use as many types of data storage as possible, some from what I have learnt in lecturers such as simple lists, lists and array Lists but also types I have found outside of lectures such as Enums and Collections. For example, I used a simple list to hold my directions. This is because simple lists are of fixed size and North, East, West and South are not going to change. I used Array Lists to add and remove rooms that have been visited to and collections to store my non-player characters (NPC’s). Collections were useful due to their expandability. By using the collections package, I could make my NPC lists immutable and unmodifiable but still not have a fixed size.

D:\Desktop\list.JPG

My last example of core java concepts is my use of flow control, if statements and error catching. I used many if statements along with while loops throughout the program to make sure the user is entering the right answer and that the correct objects are being generated. For example:

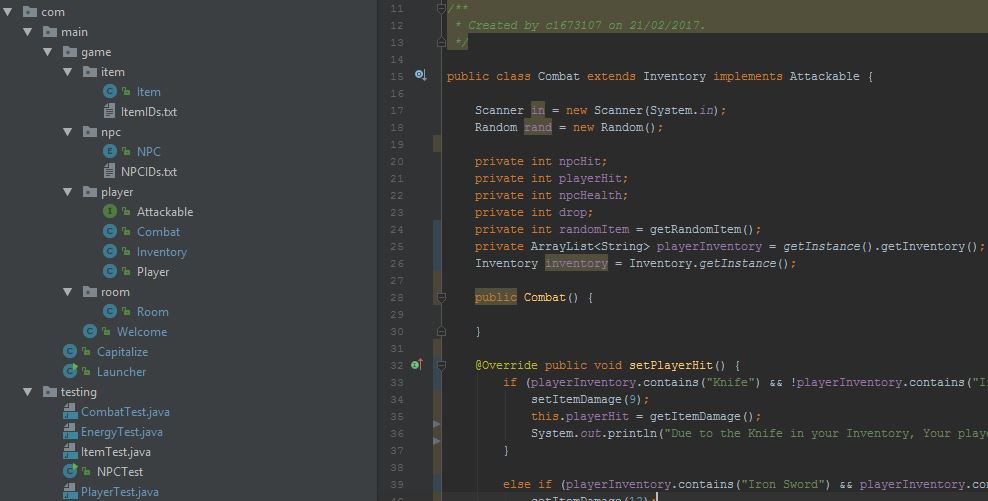




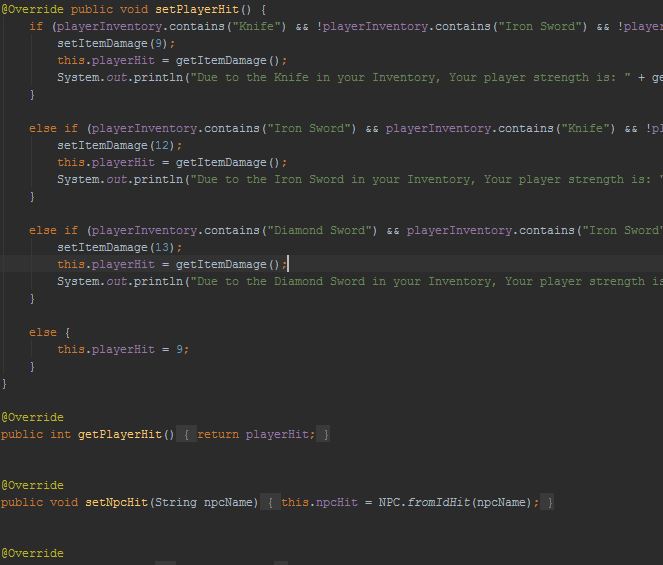
The while loop and if statements ensure the player enters a proper name and this error catching above will help me find and understand an error if it is thrown.

I also had many different classes which each had its own purpose. By using the constrictors and inheritance my I could link these together.

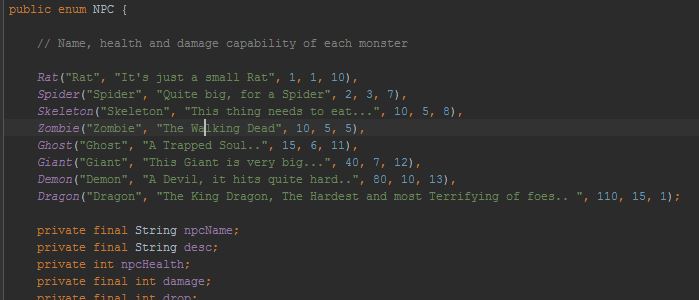
# Effectively design OO classes and interfaces, for real world problems

I believe that my code is of a high Object Oriented Standard. I do not do all the work in one class but spread it over many specific classes with specific functions in different packages shown below.

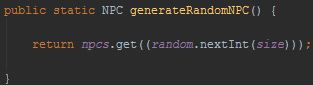
Throughout my code, I make use of inheritance, constructors and Interfaces to make my program of a high OO standard. As shown in the example screenshot above the class combat extends my class Inventory and implements the interface Attackable. In the combat class I need to be able to view, add to and remove from the inventory. Without the use of inherence here I would have to use a constructor and create an object of inventory. The benefit of this is that I have direct access to the inventory methods without the need of a constructor. The reason I extend here is because both classes the class combat is related to the fields and methods in Inventory, so instead of re-creating the methods in the combat class I can inherit and use the ones I have already made.

The below screenshot is one of the interfaces that I made.

The above interface contains methods that set and get the player health, hit and NPC health and hit. I implement the interface in the combat class. I chose an interface here because I wanted it so for an NPC to be attackable in a sense they need all these variables set. If I implement the interface I must implement all the interface methods which accomplishes this.

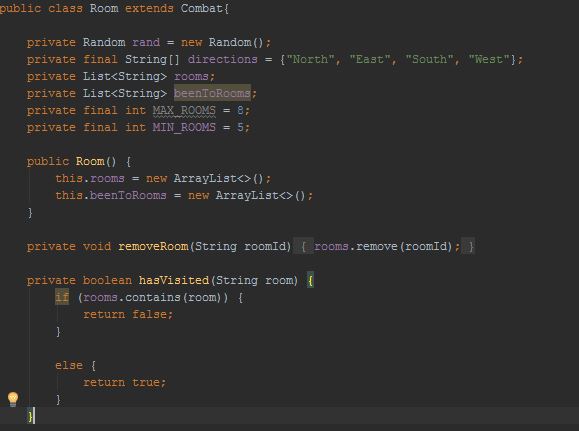
I have also made sure I maximise the use of objects throughout the program. An example of this would be my NPC object.

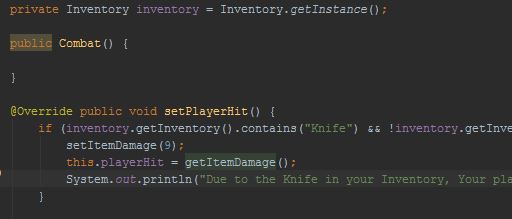
Creating an NPC object means I can have multiple NPC’s all with different health, strength and drops. If I hadn’t have used objects I would have to have many different fields for each NPC.



# Design code selecting appropriate visibility for class members, re-using code where appropriate

I designed my game with encapsulation in mind. As a result, I ensured that my fields had the visibility ‘private’ rather than ‘public’. This meant that the only way I could reach those fields would be from within that class. If I wanted to use them in another class, I would have to use the class with those fields getter and setter methods either by using a constructor or inheritance. For example, in the below class, Room, I made all my fields private.

 I also took visibility into consideration for my class methods. If I was going to need to use those methods outside the class I gave them the access modifier of public. However, there were some methods that would only need to be used within that class such as the ‘hasVisited’ method and so made it private.

In the above case, playerInventory is an array list in my inventory class but is private. I can reach it however by creating an object of inventory and using the get method (getInventory).

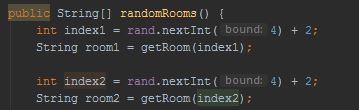
By making the variables private, my code is now more maintainable, independent and useable. [[1]](#endnote-1)If in the future, I needed to change the type of one of my fields, perhaps the inventory array List. By changing the type in the class that contains the encapsulated variable, the type will change for every class and method that uses it. I can make a change to the variable in the one class and it will not affect the other classes at all.

I did originally have a lot of fields set to public however I realised that this could cause many problems in the future. This is because you can access that public field anywhere so if there is an issue, it could be very hard to find out where it is. I also find it is easier to understand where things are being accessed from if they are private fields.

# Make use of external libraries where appropriate, and explain the rationale for doing so

For the game project, I decided that it wasn’t necessary to use external libraries. This is because there are a few negatives to using external libraries. Firstly, external libraries may not be updated very often and as this developer website states ‘it may be difficult to contact the owner regarding bugs’[[2]](#endnote-2). (Gigster, n.d.)Furthermore, parts of the library can be removed at any time or deprecated if not updated. It is possible to prevent this, but I thought it wasn’t worth the hassle for this project. In other words, the potential gain from using one of these libraries just wouldn’t be worth it.

C:\Users\Jack\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ran.jpgAnother reason I chose not to use external libraries is because the internal libraries within java was all I needed to achieve full functionality of the game. External libraries were just not needed. I used quite a lot of internal libraries, for example the Scanner Library and the Random Library. Examples of this can be seen below:

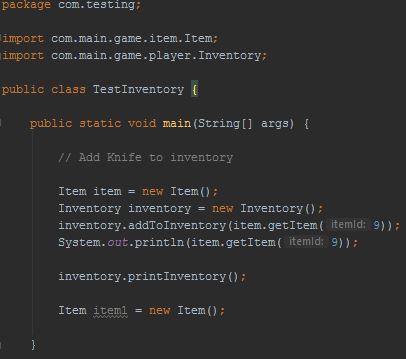


In the above case I use the random library to generate a random digit between 2 and 5. This is because out of the 8 rooms in my game, in a certain part of the level I want the user to be moving around random rooms. This makes it a little bit more interesting.C:\Users\Jack\AppData\Local\Microsoft\Windows\INetCache\Content.Word\name.jpg

Also in the above screenshot is an example of the Scanner library I used. This allows a player to enter their name. This is so their final score is saved into a SQLite database so high scores can be presented. The SQlite is also an internal library shown in the next screenshot which was not covered in our lectures.

I do understand the purpose of external libraries and will consider them again for the next project. They can be very useful if you are using your own code snippet very often in a project. Being able to extract from a library may prove useful here.

# Test code by developing simple automated tests, and explain the rationale for doing so

For the RPG, I chose to not use Junit testing. Instead I chose to create multiple testing files for each class using a method where I would create objects of a class and use the System.out.println() method to check the methodology was correct. This is shown below.

Testing in this way did work for me. In the above example I am testing the inventory and item classes. I create objects of both Item and Inventory and then combine the 2 methods together to add an item to the inventory. I used the System.out.println() method to check my getItem method was returning the correct item string ‘Knife’ (has an Id of 9), and then that the printInventory() method worked which in turn is just a print statement.

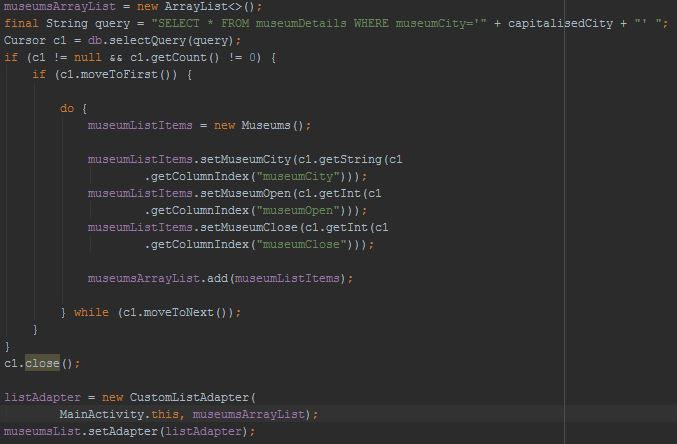
I chose to test like this because at the time I was unsure on how I was going to put all this together and use all my classes. Being able to structure my tests like this allowed me to visualise what my program would look like and how it would be structured in the final Launcher.java class.

However, it would have been better to have included automated testing through Junit on top of this. This is because by using the Junit automated testing, you can fully make sure there are no bugs and much easier see what it right and wrong. For example, when I print the item of item Id 9, it would not throw up and error if ‘Apple’ was returned instead of ‘Knife’. I would have to manually spot the mistake. Through automated testing, the program would ensure that what is retuned is in fact ‘Knife’. Of course, I could use if statement to check that is ‘Knife’ but it would be very long winded and would be much easier and quicker to use the Junit automated testing library.

# Android Client Project – Museum App – Team 14

# Write functioning java code showing understanding of core concepts

I covered a lot of the Java and Android core concepts to write functioning java code. Some of the key concepts I will demonstrate were my use of databases, data types, scope and casting and shared preferences.

I chose to use SQlite over MySQL because firstly it gives us easy access without the need of a server, like MySQL, and because it is saved on the device so the app can be used offline (client requirement). As a team, we used these SQlite databases all over the app and I was the main orchestrator in the SQlite databases. In the below example, I take us of two database initiator and helper classes I created and pull the museum records out of the database file and into a custom list view I created. I chose to make a custom list view over using the inbuilt list view because it allowed greater customizability and more data to be displayed.

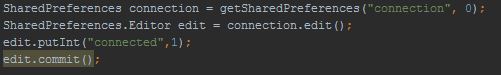
I used many different data types in the project. I used array lists in the above screenshot to pass information over to the list adapter. The reason for using array lists is because they can grow unlike more simple lists and arrays.

Another concept I took use of a lot was scope. Fields that I was going to need to use and access all over the program I specified at the top rather than directly in the methods. This is good practice. I also made use of the Java casting ability when using TextViews, ListViews and Buttons. C:\Users\Jack\AppData\Local\Microsoft\Windows\INetCache\Content.Word\casting.jpg

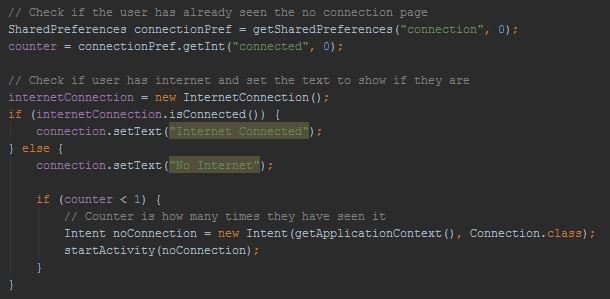
The reason I need to cast here is because the method finds the ID in my layout file and I then need to cast the returned Id to the ListView.

I also used shared preferences like in the below screenshot so that so the user is not spammed with the no connection page. When they are shown the page, they will not see it again.

When they have no connection:



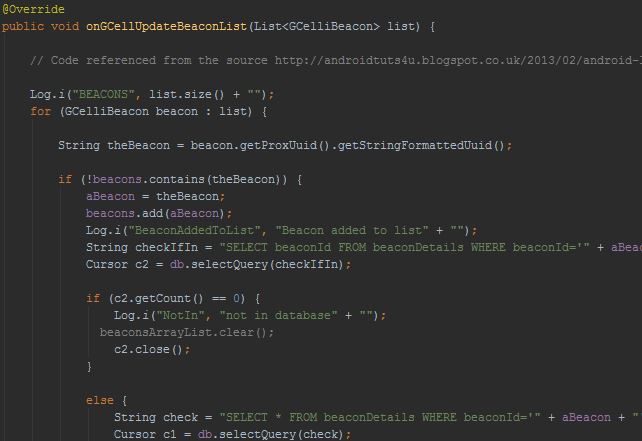
Check if shown page before:



# Effectively design OO classes and interfaces, for real world problems

The android app I produced took full advantage of the benefits from object oriented design. I implemented interfaces and inherited from other classes where needed.

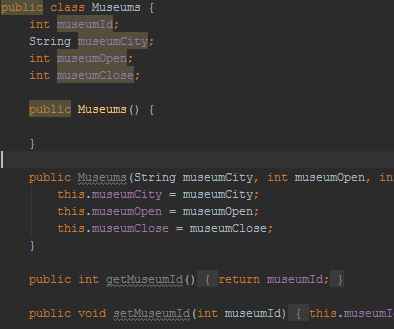
For example, the two interfaces I used in this project were Filterable and GCellBeaconManagerScanEvents: C:\Users\Jack\AppData\Local\Microsoft\Windows\INetCache\Content.Word\implements1.jpgC:\Users\Jack\AppData\Local\Microsoft\Windows\INetCache\Content.Word\imlem3.jpg



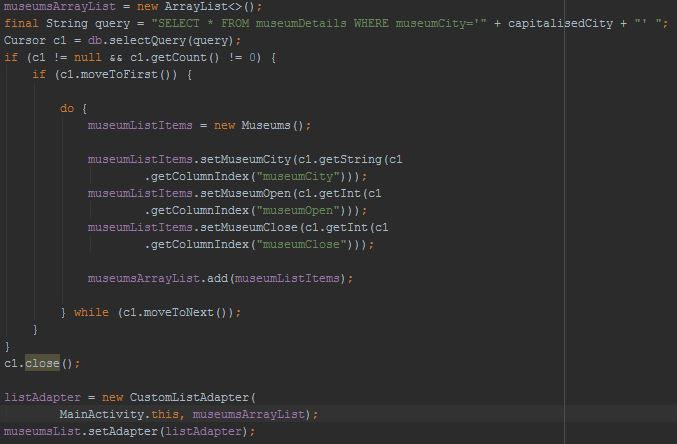
I used the interface GCellBeaconManagerScanEvents because this was required to deliver the Bluetooth beacons in which the client wanted. As it is an interface I must implement all its methods like the one above. Researching into all its methods allowed me to go beyond the scope of lectures. For example, I get the beacons ID and check it with allowed beacons in databases to ensure proper security. I can then fetch all the beacons data from the database and show it in a ListView if it is in the database.

I inherited the classes AppCompatActivity and BaseAdapter. I chose to use AppCompatActivity over others such as Activity for several reasons. Firstly, it is backwards compatible with many lower SDK versions and it uses ToolBar over the deprecated ActionBar. It also naturally inherits from FragmentActivity which would allow me to easily use a Fragment Manager.

The teams effective design in Object Oriented Programming is also shown by our used of packages and objects. All our classes are nearly ordered into relevant packages. I linked classes together in different packages using imports and used constructors to access methods in other classes. Such as in the screenshot on the next page.

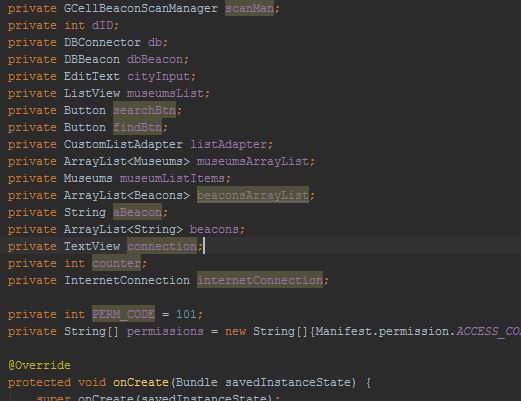
This museum class contains getter and setter methods to help me add and remove museums from the database.

I used the empty constructor above to access these getter and setters in whatever class I needed to, such as in the Main Activity.



As shown, I initialise a new object of museums and set each field in museums class with what is in the cursor of the database. Creating museum objects means I can have museums of different names and with different opening and closing times.

# Design code selecting appropriate visibility for class members, re-using code where appropriate

For fields outside the scope of the methods of a class as shown in the below screenshot, I chose to give visibility (or the access modifier) private. I created some fields outside the scope of the methods if I needed to reuse those variables. 

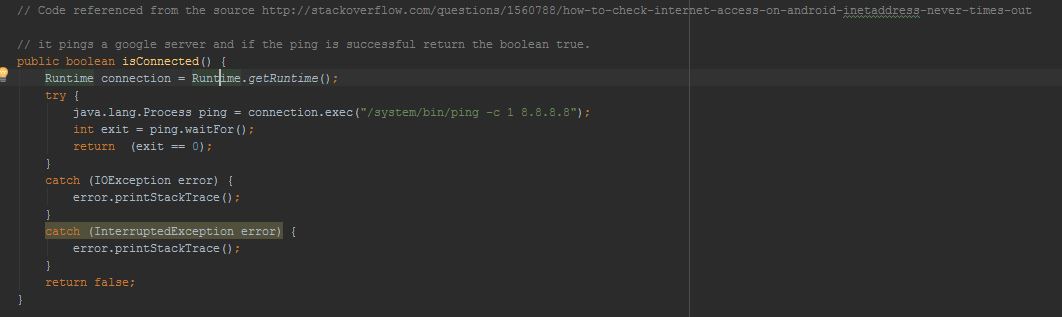
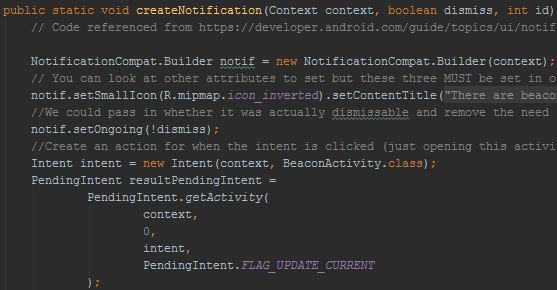
For example, in this case, these fields are created outside of the methods. I re-use these variables in multiple instances of the code. The string aBeacon for instance I need to access in many different methods. Methods which add this to a database and which create a notification. If I had initialised these fields in the method scope I wouldn’t be able to access it anywhere other than in the method it is initialised in.

I made the fields private because my code is now more maintainable, independent and useable. [[3]](#endnote-3)If in the future, I needed to change the type of one of my fields, the type will change for every class and method that uses it. By benefiting from encapsulation, I can make a change to the variable in the one class and it will not affect the other classes at all.

Having fields set to public could cause many future issues especially if developers join a project late into its development. This is because you can access that public field anywhere so if there is an issue, it could be very hard to find out where it is.

I set the access modifier visibility of my methods to public so that I can access them in multiple classes, as shown in the following screenshot. In evaluation however, I could change some of the methods to private as not all the methods are required outside the classes they are made in.

The notification method is only used in that class and so could be set to private. The connection method must be public however as it is called in different classes.



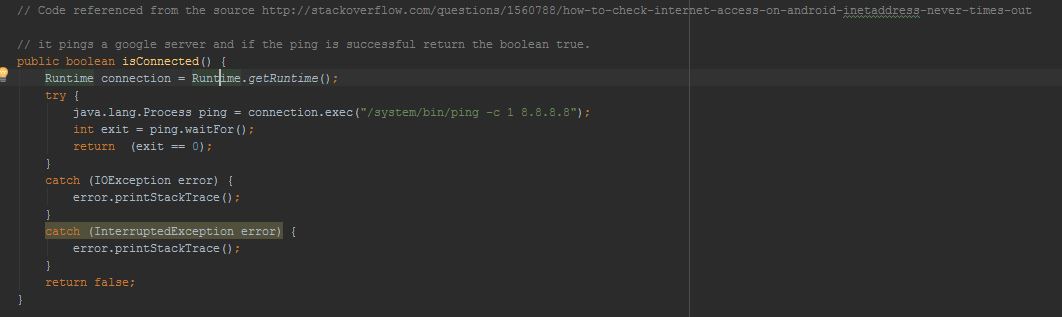
# Make use of external libraries where appropriate, and explain the rationale for doing so

I have used two different external libraries to create the app for the client. The GCellBeacon Library and Junit.

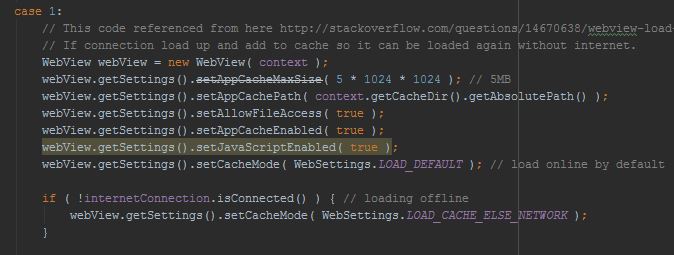
I chose to use the GCellBeacon external library mainly because it is a well-known library which was recommended by my University. I do know that other libraries exist such as AltBeacon and Google beacons. However, for this project, the GCellBeacon library was perfect for what we needed to do. The beacons provided by my University for this project were also built to run off this library so it was essential.

I did however know the benefits of using an external library. For example, they help reduce file sizes and clean the code. This is because instead of re-using code everywhere it can come from the library. You can also test external libraries dependant of the rest of your code allowing you to better identify where bugs exist.

I chose to use the Junit external library as it enables me to take advantage of automated testing which makes it much easier to identify and correct bugs. I will talk about this in greater detail in the next section however.

It wasn’t just external libraries that I used, I also took full advantage of the internal libraries Android and Java provides. I used many libraries such as Toast, View, Log and many more. In the next example is a class which uses the two internal libraries Runtime and Process.

These go beyond the coverage of my lectures.

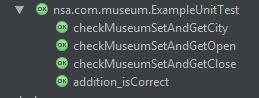
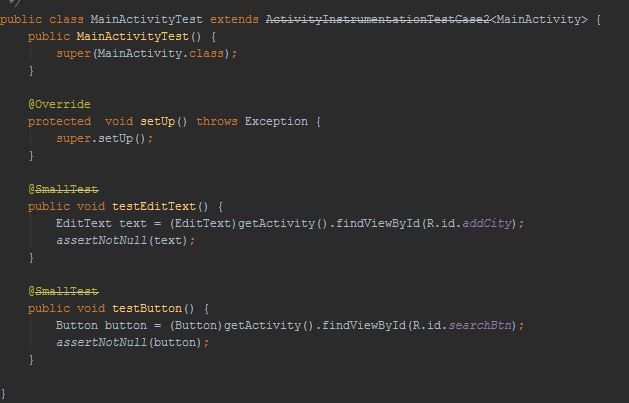
I use the Runtime library to execute the connection command on the environment interface. The Process library allows me to make the thread wait until the process is complete. When the ping is complete if it is successful, 0 will be returned. I was then able to use this Boolean method to display to the user if they have internet connection and attempt to cache webpages. 

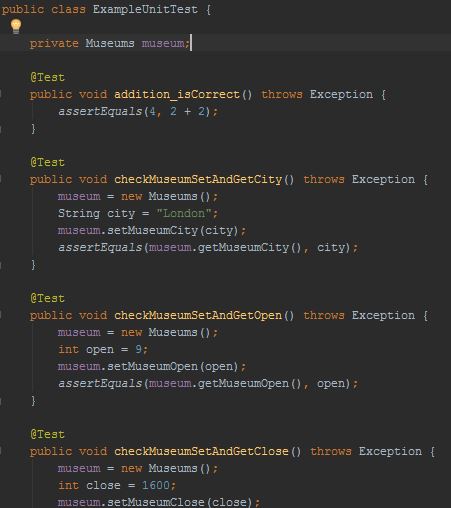
# Test code by developing simple automated tests, and explain the rationale for doing so

I had 3 main methods of testing in this project. The first was using log statements. I often used logs to test which bits of the code were working when I ran into bugs. I used logs because it is an easy and quick way to test. I could use the android studio script filter to see all the logs printed. The second method I used was just manually trying out the app by running it. By doing it this way I could visually see what was working and what wasn’t. My final and best method however was automated testing through the Junit external library.

This method was better than the others because it is much more accurate and thorough in testing. Being able to write the code to continually test methods and classes each time the app is run is much quicker than having to print log statements for every possibility and manually try out every possible scenario in the app. Through these Junit tests I could create scenarios to see how the app would react and test things much quicker than I would be able to do on my own.

The shots below show me using unit tests and instrumentation tests. Both types of these tests are crucial. The unit tests allowed me the test all the methods of my classes. Whereas the instrumentation tests allowed me to check the more GUI side of android such as buttons were working correctly. As shown below, in the unit test I check the Museum class by creating a museum object and then ensuring that my methods returned the correct values. For the instrumentation tests, I check that the button and edit text field on various pages are not null. The speed at which these tests are completed is so quick it made this method of testing extremely efficient and allowed me to spot bugs that I might not have ever checked for without this testing approach.





# References

**For this documentation and all project code**

TutorialsPoint 2017. Java Encapsulation [Online]. Available at:https://www.tutorialspoint.com/java/java\_encapsulation.htm[Accessed: 16 March 2017].

Sa, J. 2017. Benefits of Encapsulation [Online]. Available at:http://www.cems.uwe.ac.uk/~jsa/UMLJavaShortCourse09/CGOutput/Unit3/unit3(0809)/page\_13.htm [Accessed: 16 March 2017].

SoftwareEngineering, W. (2017). *Why do we need private variables?*. [online] Softwareengineering.stackexchange.com. Available at: https://softwareengineering.stackexchange.com/questions/143736/why-do-we-need-private-variables [Accessed 26 Apr. 2017].

duffymo, Stack Overflow. (2017). *Unmodifiable List in java*. [online] Stackoverflow.com. Available at: http://stackoverflow.com/questions/16356232/unmodifiable-list-in-java [Accessed 26 Apr. 2017].

www.tutorialspoint.com. (2017). *java.util.Collections.unmodifiableList() Method Example*. [online] Available at: https://www.tutorialspoint.com/java/util/collections\_unmodifiablelist.htm [Accessed 26 Apr. 2017].

Stackoverflow.com. (2017). *Java - ArrayList is returning Empty*. [online] Available at: http://stackoverflow.com/questions/43393196/java-arraylist-is-returning-empty?noredirect=1#comment73848343\_43393196 [Accessed 26 Apr. 2017].

Java2s.com. (2017). *Makes the first letter caps and the rest lowercase. : String Format « Data Type « Java Tutorial*. [online] Available at: http://www.java2s.com/Tutorial/Java/0040\_\_Data-Type/Makesthefirstlettercapsandtherestlowercase.htm [Accessed 26 Apr. 2017].

Stackoverflow.com. (2017). *Activity, AppCompatActivity, FragmentActivity, and ActionBarActivity: When to Use Which?*. [online] Available at: http://stackoverflow.com/questions/31297246/activity-appcompatactivity-fragmentactivity-and-actionbaractivity-when-to-us [Accessed 26 Apr. 2017].

Docs.oracle.com. (2017). *Process (Java Platform SE 7 )*. [online] Available at: https://docs.oracle.com/javase/7/docs/api/java/lang/Process.html [Accessed 26 Apr. 2017].

Developer.android.com. (2017). *AppCompatActivity | Android Developers*. [online] Available at: https://developer.android.com/reference/android/support/v7/app/AppCompatActivity.html [Accessed 26 Apr. 2017].

Gigster.com. (2017). *Should you use external libraries in your code? What are the benefits and drawbacks? What precautions can be taken?*. [online] Available at: https://gigster.com/java-interview-questions/java-library-advanced [Accessed 26 Apr. 2017].

Developer.android.com. (2017). *Notifications | Android Developers.* [online] Available at: https://developer.android.com/guide/topics/ui/notifiers/notifications.html [Accessed 26 Apr. 2017].

Developer.xamarin.com. (2017). *Style a Button - Xamarin*. [online] Available at: https://developer.xamarin.com/recipes/android/resources/general/style\_a\_button/ [Accessed 26 Apr. 2017].

Driller., J. (2017). *WebView load website when online, load local file when offline*. [online] Stackoverflow.com. Available at: http://stackoverflow.com/questions/14670638/webview-load-website-when-online-load-local-file-when-offline [Accessed 26 Apr. 2017].

Krishna, A. (2013*). Android List View using Custom Adapter and SQLite*. [online] Androidtuts4u.blogspot.co.uk. Available at: http://androidtuts4u.blogspot.co.uk/2013/02/android-list-view-using-custom-adapter.html [Accessed 26 Apr. 2017].

Patel, N. (2017). *Convert ImageView in bytes android?.* [online] Stackoverflow.com. Available at: http://stackoverflow.com/questions/20700181/convert-imageview-in-bytes-android [Accessed 26 Apr. 2017].

Stackoverflow.com. (2017). *Null pointer error with hideSoftInputFromWindow*. [online] Available at: http://stackoverflow.com/questions/19069448/null-pointer-error-with-hidesoftinputfromwindow [Accessed 26 Apr. 2017].

1. # Code Quality Coursework – By Jack Allcock (myself)

   [↑](#endnote-ref-1)
2. https://gigster.com/java-interview-questions/java-library-advanced

   Code Quality Coursework – By Jack Allcock (myself)

   # Appendix

   **GitLab Links:**

   https://gitlab.cs.cf.ac.uk/c1673107/RPGGameProject

   https://gitlab.cs.cf.ac.uk/CM6122\_project/cm6122\_museum\_team14

   **GitLab SSH’s:**

   git@gitlab.cs.cf.ac.uk:c1673107/RPGGameProject.git

   git@gitlab.cs.cf.ac.uk:CM6122\_project/cm6122\_museum\_team14.git [↑](#endnote-ref-2)
3. [↑](#endnote-ref-3)