Design Patterns

# Introduction:

Design patterns are a very useful tool used by object oriented software developers in order to aid them in battling common problems. Design patterns aim to create solutions to problems which can be reused, their purpose is also to simplify the implementation as well as add clarity to the code so that it’s easier to understand. There are 3 main types of design patterns: Creational, Structural and Behavioural. I will cover some of the design patterns that can be used in the Android framework and can potentially aid the development of my game.

# Design Patterns:

## Creational:

**Builder:**The builder pattern is a commonly used in order to create multiple complex objects out of basic objects collectively. It makes it much easier to create them in mass amounts, the implementation makes sure our code is much cleaner and reduces the repetition. It allows us to use the same objects for various representations, or representations of other complex objects, which might consist of different basic objects.

**Dependency Injection:**Dependency injection is a pattern where the dependency is not given to an object on creation; it is rather “injected” into it externally. In simple terms, this basically means that we are passing instance variables to our objects externally after they are created rather than on creation. There are multiple ways we can inject dependencies into our objects, two most basic methods to “inject” dependencies are through the constructors or setters.  
Dependency injection is especially useful when testing, it allows us to isolate classes in order so that a stub object can be passed and appropriate functionality can be easily tested.

**Singleton:**This design pattern is used where only a single instance of a class is needed to be globally accessed. The class, with the way it’s written, implements a static method which returns an instance of the class, which can be initialized only once in the entire program, the constructor cannot be invoked. This is useful for modelling real life objects which can only exist as a single instance, singleton allows the instance to be accesses from anywhere within the program, it has a global access.

## Structural:

**Adapter:**The purpose of the adapter design pattern is to create a bridge between two classes which are not compatible and make it possible for them to work. The way that this is achieved is by converting our current into an interface which we want the user to see. It’s very useful when working with classes that are not compatible but provide essential functionality for us, we can “adapt” them through a use of specifically written methods so that they convert their interface into one more suitable to achieve our goal.

**Facade:**The goal behind the facade design pattern is to hide the underlying complexity of the system from the user. It’s about building an interface which will contain what the user needs without the unnecessary complexity behind it. We can think of it just like the literal real world use of facade, it’s a cover up, a fancy clean interface to hide all the nasty inner workings of a program.

Facade allows us to decouple the system; we can provide methods which return from a variety of different classes. This adds clarity through the use of a new interface on top of our system, without the user having to see all the depth behind our implementation. It provides exactly what the user needs, rather than flooding them with unnecessary detail.

## Behavioural:

**Command:**The command design pattern aims to decouple the object the object that invokes an operation from the one that executes it. The invoker received a command hidden in an object and then passes it to the appropriate object which handles a given request. This pattern is useful when wanting to issue a command without knowing the receiver; the pattern handles our request and forwards the command to an appropriate object so that it can be dealt with.

**Observer:**The observer pattern is used in order to be able to monitor changes in other objects. Whenever a change is detected, an observable object will notify all of its observers and the changes could be acted upon through the use of various methods. The observer pattern aims to decouple objects so that they are easy to change and maintain.

# Sources:

<https://www.raywenderlich.com/109843/common-design-patterns-for-android>

<http://www.jamesshore.com/Blog/Dependency-Injection-Demystified.html>

<https://www.tutorialspoint.com/design_pattern/adapter_pattern.htm>

<https://sourcemaking.com/design_patterns/command>