Exploring and Using Creational, Structural

and Behavioural Design Patterns

*Ryan Edwards*

*Introduction*

Within this report we will explore the different uses of three different design patterns. The three different design patterns are Creational, Structural and Behavioural. This report will contain what they are and what they do within object orientated.

Creational

Creational Design Patterns is the way in which objects are created. The Creational Design patterns are design patterns that are tasked with dealing with object creation mechanisms based on the surrounding outcomes which it designs the objects based off of. It creates objects in a manner suitable to the situation. Creational design patterns solve this problem by somehow controlling this object creation

Some examples of Creational Design Patterns:

Factory Method – The Factory Method Design Pattern is one of the most used design patterns within Java. The Factory Method is the interface for creating an object, but let’s subclasses decide which class to instantiate. The Factory method lets a class defer instantiation to subclasses. To achieve this, we rely on the factory method which provides us with the objects, hiding the actual implementation details. The created objects are accessed using a common interface

Builder Design – The Builder Design Pattern is a creational pattern. That is designed to deal with the construction of more complex and difficult objects. When the complexity of creating object increases, the Builder pattern can separate out the instantiation process by using another object to construct the object. This builder can then be used to create many other similar representations using a simple step-by-step approach.

Singleton – Singleton is the method of creating a class that is globally available to all other classes. But when in herniated and altered by the main class whatever changes it makes it will change them on every class that has the singleton class within.

Here is some example code to input and use a Singleton:

public class EagerSingleton {

/\*\* private constructor to prevent others from instantiating this class \*/

private EagerSingleton() {}

/\*\* Create an instance of the class at the time of class loading \*/

private static final EagerSingleton instance = new EagerSingleton();

/\*\* Provide a global point of access to the instance \*/

public static EagerSingleton getInstance() {

return instance;

}

}

Structural

The structural design pattern is all about the relationship between the classes and subclasses within. It does this by giving them easy to realise identity’s. These design patterns control how the classes inherit from each other and how they are composed from other classes

Some examples of Structural Design Patterns:

Decorator Design - Allows behaviour to be added to an individual object, dynamically, without affecting the behaviour of other objects from the same class. There are many times to use a decorator design pattern one of them is when you want to add responsibilities to an object that you may want to change in future

Adapter Design – The Adapter Design allows the interface of an existing class to be used as another interface. It is often used to make existing classes work with others without modifying their source code.

Flyweight –A flyweight pattern is designed to minimizes memory usage by sharing as much data as possible with other similar objects as its self. It is a way to use objects in large numbers when a simple design pattern and objects would use a large amount of memory.

Behaviour

Behavioural design patterns are design patterns that identify common communication patterns between objects. By doing so, these patterns increase flexibility in carrying out this communication. Behavioural class patterns use inheritance to distribute behaviour between classes

Command – The Command pattern within the behaviour pattern turns a request into a stand-alone object that contains all information from the request that was made. This lets you parameterize methods with different requests, delay or queue a request’s execution, and support undoable operations.

State – State is a behavioural design pattern that lets an object alter its behaviour when its internal state changes. It appears as if the object changed its class. The State pattern is closely related to the concept of a Finite-State Machine with showing the current state of a running machine in an interface with which the user can see the outcomes.

Observer – that lets you define a subscription mechanism to notify multiple objects about any events that happen to the object they’re observing.