

CONCURRENT PROGRAMMING PROJECT | DESIGN PATTERN DELIVERABLE.

GROUP 2

Group Members Mayougo Akingni | Gossadina Eliphaz | Ating Eric |Erwan Roberto

UCAC-ICAM, October 2021.

DESIGN PATTERN CHOICE JUSTIFICATION.

In software engineering, a **design pattern** is a general repeatable solution to a commonly occurring problem in software design. A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

After carrying out a number of research, we choose the following design patterns for our project.

1. MODEL VIEW CONTROLLER (MVC) MODEL.

The **Model-View-Controller (MVC)** is an architectural pattern that separates an application into three main logical components: the **model**, the view, and the controller. Each of these components are built to handle specific development aspects of an application. Based on the various criteria given to us for this project, we choose the MVC model for the following reasons;

* A main advantage of MVC is separation of concern. Separation of concern means we divide the application Model, Control   and View.
* We can easily maintain our application because of separation of concern.
* In the same time we can split many developers work at a time. It will not affect one developer work to another developer work.
* It supports TTD (test-driven development). We can create an application with unit test. We can write won test case
* Latest version of MVC Support default responsive web site and mobile templates.
* We can create own view engine. It is syntax is very easy compare to traditional view engine.

1. BUILDER DESIGN PATTERN

This DP provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created. It is mostly used when object can't be created in single step like in the de-serialization of a complex object reason why we choose it for our project. The main advantages of using builder DP are;

* It provides clear separation between the construction and representation of an object.
* It provides better control over construction process.
* It supports to change the internal representation of objects.

1. FACTORY METHOD PATTERN

**Factory method** is a creational design pattern which solves the problem of creating product objects without specifying their concrete classes. The main reason we choose this model for our project was for the following criteria.

* It will help us have a high level of flexibility in our code.
* Factory Method Pattern allows the sub-classes to choose the type of objects to create.
* It promotes the loose-coupling by eliminating the need to bind application-specific classes into the code. That means the code interacts solely with the resultant interface or abstract class, so that it will work with any classes that implement that interface or that extends that abstract class.

1. SINGLETON PATTERN

**Singleton** is a creational design pattern, which ensures that only one object of its kind exists and provides a single point of access to it for any other code. We choose this pattern for two main reasons;

* Saves memory because object is not created at each request. Only single instance is reused again and again.
* Singleton pattern is mostly used in multi-threaded and database applications. It is used in logging, caching, thread pools, configuration settings etc.

1. DECORATOR PATTERN

The Decorator Pattern uses composition instead of inheritance to extend the functionality of an object at runtime. Using decorators you can wrap objects countless number of times since both target objects and decorators follow the same interface reason why it is also known as **Wrapper.**  Advantages of this pattern include;

* It provides greater flexibility than static inheritance.
* It enhances the extensibility of the object, because changes are made by coding new classes.
* It simplifies the coding by allowing you to develop a series of functionality from targeted classes instead of coding all of the behavior into the object.

1. STRATEGY PATTERN

Strategy is a behavioral design pattern that turns a set of behaviors into objects and makes them interchangeable inside original context object.

The original object, called context, holds a reference to a strategy object and delegates it executing the behavior. In order to change the way the context performs its work, other objects may replace the currently linked strategy object with another one. Its benefits include;

* It provides a substitute to subclassing.
* It defines each behavior within its own class, eliminating the need for conditional statements.
* It makes it easier to extend and incorporate new behavior without changing the application.