# Software Design Patterns

**Name - Kartikay Goel**
**Roll Number - 180101033**
**IIT Guwahati**

## Abstract

This paper provides an overview of the various Design Patterns used in software development. It will cover topics such as what a design pattern is, why we need one, and how to classify them. The patterns needed for the project allotted to us i.e. Automated Live Class Performance Evaluation System will be given special focus and attention. The focus will be on creational design patterns, structural design patterns, behavioural patterns and architectural patterns.

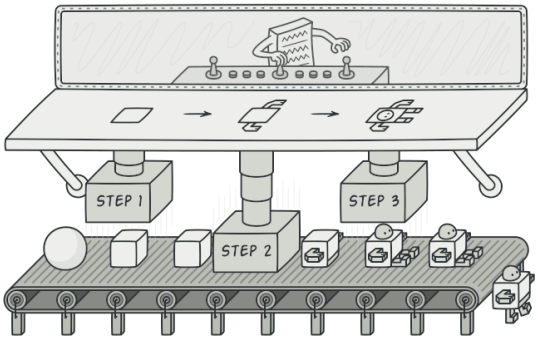
## Introduction

A software design pattern is a reusable and generic solution to a common software design issue in a particular situation. It is not a completely defined architecture that can be coded immediately. Instead, it's a summary or roadmap for resolving an issue that can be applied to a variety of circumstances. Design Patterns are best practises that a programmer should use when designing an application to solve common issues. Understanding why design patterns are used is important for us to use them. By providing tried-and-true architecture paradigms, design trends may help to speed up the development process. With these paradigms, it's simple to create highly coherent modules with low coupling. They isolate potential complexity in system requirements, making the system as a whole easier to understand and handle. Furthermore, design patterns facilitate efficient communication among designers.

*Structural Design Patterns, Creational Design Patterns, Behavioral Design Patterns,* as well as *Architectural Design Patterns*, these are the four sub-categories in which the design patterns have been categorised. These categories are based on the type of problem they solve.

## Creational Patterns

Creational design patterns are based on two major ideas. The first is details about the system's concrete grades. Another is to hide the development and combination of these concrete classes' instances. Builder method, Singleton method, Factory method, Abstract Factory method, and some others are examples of creational patterns. Below is a quick overview of the Builder and Factory methods which have been used in the Automated Live Class Performance Evaluation Application.

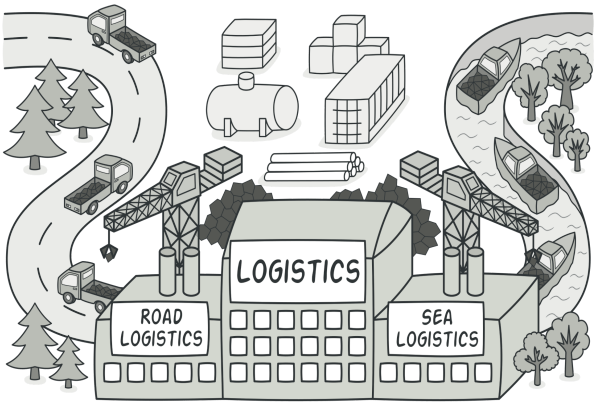



***Builder Method Design Pattern:*** The Builder design pattern is a step-by-step creational design pattern that helps you to create complex artefacts. You may use the template to generate different forms and representations of an object while also using the same construction code. The example of making a pizza is well-suited for explaining builder method design patterns. The toppings which are added to pizza cannot be arranged in any random order, or else the entire thing would be sticky. In its place, a step-by-step technique is used. This usually begins by determining the size of the pie and, as a result, the amount of dough to use. After that, the sauce is added, accompanied by the cheese, maybe some pepperoni, and so forth.

In Java, Method Chaining is used to execute the builder method. Method-chaining is enforced by a series of

methods. These methods return the object relation for a class instance. Invoking methods in a series is possible by basing the next method invocation on the return value of the previous method in the chain. Thus, models used in Automated Live Class Performance Evaluation Application can be constructed using builder method with the help of method chaining.

***Factory Method Design Pattern:*** According to the Factory Method Pattern, what you have to do to create an object is describe an interface or abstract class and then let the subclasses determine which class to instantiate. Subclasses, in other words, are in charge of constructing the class case. Virtual Constructor is another name for the Factory Method Pattern. Sub-classes will choose the type of object they want to generate using the Factory Method Pattern. It encourages loose coupling by removing the need for application-specific classes to be bound into the code. That is, the code only deals with the resulting interface or abstract class, and it can operate with all classes that enforce or extend the interface or abstract class. When a class doesn't know what subclasses would be needed to construct, when a class requires its subclasses to define the objects to be generated, or when the parent classes choose the construction of objects for their subclasses, the factory design pattern is used.




## Structural Patterns

Structural patterns explain how to put objects and groups together to form larger constructs that are both scalable and practical. Adapter, Bridge, Composite, Decorator, and other structural patterns are examples. The Adapter Method is briefly discussed in the following section:

***Adapter Pattern:*** The adapter design pattern is a structural architecture pattern that allows structures with incompatible interfaces to work together. Since the real world is full of adapters, this pattern is easy to learn. A USB to Ethernet adapter is a good option. We need this because one end has an Ethernet port and the other has a USB interface. We need a converter to convert one to the other since they are incompatible. Adapters not only support objects with different interfaces in communication, but they can also convert data into different formats. An interface is assigned to the adapter that is compatible with one of the pre-existing objects. Using this protocol, the current object can securely call the adapter's methods. When the adapter gets a call, it sends it to the second object, but in the format of the second object. Adapters are a clever way to link objects to the View in Java. RecyclerView and PagerAdapter are examples of the Adapter pattern that are used in Automated Live Class Performance Evaluation Application.

*A USB to ethernet adapter*

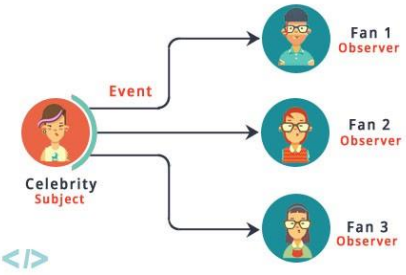
## Behavioural Patterns

Behavioral design patterns are patterns that identify common communication patterns among objects. As a result, these trends broaden the range of contact options. Command, Iterator, Mediator, Memento, Observer, and other behavioural trends are examples. The Command and Observer interface patterns are discussed briefly below.

***Command Pattern:*** According to the Command Pattern, "Encapsulate a request in an object and transfer it to the invoker object as a command. The invoker object searches for a suitable object that can handle this

command and passes the command to that object, which then executes the command ". It's often referred to as an event or a transaction. It distinguishes between the object that executes the procedure and the object that invokes it. Since current classes aren't updated, it's simple to add new commands. When you need to parameterize objects based on an action taken, when you need to generate and execute requests at various times, or when you need to enable rollback, this pattern is used.

***Observer Pattern:*** Observer is a behavioural design pattern that allows you to choose a subscription mechanism that will notify other objects of any incidents that happen to the object you're watching. This system is now used by every organisation. The company sends an email to all those who have subscribed to the company if a new product is introduced to the market. The business keeps track of subscribers who are involved in the new offering. As a result, the buyer does not have to keep checking the store's website to see whether a new product will be available. The class in Java has three methods: subscribe, unsubscribe, and alert, as well as a list of all subscribers.




## Architectural Patterns

Architectural patterns are utilized when the task is to arrange and modularize the code of the project. Regardless of the medium, these patterns can be used everywhere. Model View Controller (MVC), Model-View-ViewModel (MVVM), Model View Presenter (MVP), and other architectural patterns are among the most common. The following is a quick overview of MVC architecture:

### *MVC Architecture:*

MVC or Model View Controller as it is popularly called, is a software design pattern for developing web applications.
MVC pattern consists of the following components-:

1. **Model −** This is responsible for maintaining data and is the lowest level of the pattern
2. **View −** It is responsible for displaying the data to the user (completely or partially)
3. **Controller −** It is the software code that controls the interactions between the Model and View.

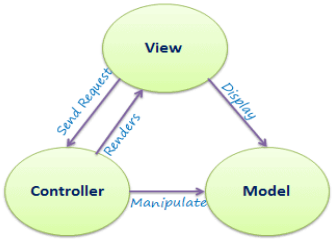



Figure 1 shows the relation between the Controller, the Model, and the View. Model and View in the MVC pattern can never interfere with each other. The Model data will be shown on View by the Controller. If any data changes, the Model will inform the Controller, and the Controller will update the View. When a user interacts with a View, the Controller is notified, and the Model data is either requested or manipulated.

## Conclusion

Software that is more nuanced and advanced is being produced nowadays. Software developers must modify and upgrade large pieces of code in their architecture as customer expectations evolve and expand. Design Patterns include tried-and-true strategies for completing a variety of activities and overcoming dynamic difficulties while undertaking even the most simple tasks of daily life. Adopting various design trends based on one's needs will boost productivity and performance. These architecture trends are among the most comprehensive resources for software design available to programmers.