# Creational Design Pattern

## What is design pattern?

In JavaScript, a design pattern is a reusable solution to a common problem that occurs in software design.

They help developers create code that is more maintainable, scalable, and robust.

## Creational Pattern

Creational design patterns are a subset of design patterns in software development.

They deal with the process of object creation, trying to make it more flexible and efficient.

It makes the system independent and how its objects are created, composed, and represented.

## Types of creational pattern

- Singleton Method
- Factory Method
- Abstract Factory Method
- Builder Method
- Prototype Method

## Singleton Method

Singleton pattern is a design pattern which restricts a class to instantiate its multiple objects.

It is nothing but a way of defining a class.

Class is defined in such a way that only one instance of the class is created in the complete execution of a program or project.

## Key Components (singleton method)

Private Constructor

Private Static Instance

Public Static Method to Access the Instance

Lazy Initialization

## Advantage

Controlled access to sole instance

Reduced namespace

Permits a variable number of instances

## Disadvantage

Concurrency Issues: If not implemented carefully, Singletons can introduce concurrency issues in multi-threaded applications.

Singleton Pattern Overuse: Due to its convenience, developers might overuse the Singleton pattern, leading to an abundance of Singleton instances in an application.

## Factory Method

- The Factory Design Pattern is a creational pattern that allows for the creation of objects without exposing the creation logic to the client.
- It involves creating a separate factory function that is responsible for creating instances of various related objects based on a specified input.
- In modern software development, the Factory Design Pattern plays a crucial role in creating objects while abstracting away the specifics of their creation.

## Key Components

- Base Class
- Sub Class
- Factory Class
- Client part to call factory class

## Advantage

Abstraction and Encapsulation

Simplified Object Creation

Code Reusability

## Disadvantage

Complexity for Simple Cases

Understanding the Factory Logic

Potential Performance Overheads

## Abstract Factory Pattern:

- Abstract Factory Pattern is to abstract the process of object creation by defining a family of related factory methods, each responsible for creating a different type of object.
- These factory methods are organized within an abstract factory interface or class, and the client code uses this interface to create objects.

## Components

- Abstract Factory: This is an interface or an abstract class that declares a set of factory methods for creating a family of related objects.
- Concrete Factory: These are the implementations of the abstract factory interface. Each concrete factory is responsible for creating a specific family of related objects.
- Abstract Products: These are interfaces or abstract classes that declare the common set of methods for the objects within a family.
- Concrete Products: These are the actual implementations of the abstract products. Each concrete product is specific to a particular family and is created by a corresponding concrete factory.
- Client: The client code interacts with the abstract factory and abstract product interfaces.

## Advantage

Consistency: Abstract factories patterns ensure that the objects created within a family are compatible with each other.

Flexibility: Abstract Factory pattern allows us to switch between different families of objects by using different concrete factories.

Extensibility: Basically, we can add new concrete factories to support new product families without modifying existing client code.

## Disadvantage

Increased Complexity: We need to define interfaces for factories and their product families, create concrete factory implementations, and manage the relationships between factories and products.

Code Duplication: While implementing concrete factories patterns for different product families, we end up with duplicate code for creating similar types of objects across multiple factories.

Limited Use Cases: Abstract Factory pattern is most valuable in scenarios where there are multiple related product families with interchangeable components.

## Builder Pattern

The Builder design pattern is a creational design pattern used to construct complex objects by separating the construction process from the actual representation.

It’s especially useful when an object requires multiple steps or configurations to be created.

The Builder Pattern is a design pattern used to construct complex objects step-by-step.

## Components

- Product class:The object being built. It holds all the data and attributes that you want to include in the final object.
- Builder Class: A helper class that provides methods to set various attributes of the User object. It ensures that the object is created step-by-step and allows for customization.
- Usage : The process of using the Builder to set attributes and create the final Product object. It involves calling methods to set attributes and then building the final object.

## Advantage

Improved Object Creation: The Builder pattern allows for the step-by-step construction of an object, making the creation process more understandable and manageable.

Flexible Object Construction: It provides flexibility by allowing different configurations for constructing objects.

Code Readability:

## Disadvantage

Code Overhead: Implementing the Builder pattern requires creating a separate builder class or methods, which can introduce additional code and complexity.

Not Suitable for Simple Objects: For simple objects that do not have a significant number of properties or configurations, the Builder pattern may be overly complex.

## Prototype Pattern:

- A Prototype Method is a JavaScript design pattern where objects share a common prototype object, which contains shared methods.
- The prototype method allows you to reuse the properties and methods of the prototype object, and also add new ones as needed.

## Components:

**Prototype:** An existing object that serves as a template for creating new objects.

**Cloning:** Creating a new object by copying the prototype object.

## Advantage

Efficient Object Creation: Objects can be created by copying existing ones, saving time and memory.

Dynamic Updates: Changes made to a prototype are reflected in all instances, allowing for dynamic updates.

## Disadvantage

Potential for Overwriting: Modifying the prototype can unintentionally affect all instances, causing unexpected behaviour.

Complexity: Managing and updating prototypes and their relationships can become complex as the application grows.