# Design Patterns in Java

## A design patterns are **well-proved solution** for solving the specific problem/task.

## Advantage of design pattern:

1. They are reusable in multiple projects.
2. They provide the solutions that help to define the system architecture.
3. They capture the software engineering experiences.
4. They provide transparency to the design of an application.
5. They are well-proved and testified solutions since they have been built upon the knowledge and experience of expert software developers.
6. Design patterns don?t guarantee an absolute solution to a problem. They provide clarity to the system architecture and the possibility of building a better system.

### When should we use the design patterns?

We must use the design patterns **during the analysis and requirement phase of SDLC**(Software Development Life Cycle).

### Categorization of design patterns:

Basically, design patterns are categorized into two parts:

1. Core Java (or JSE) Design Patterns.
2. JEE Design Patterns.

## Core Java Design Patterns

In core java, there are mainly three types of design patterns, which are further divided into their sub-parts:

## 1.Creational Design Pattern

1. Factory Pattern
2. Abstract Factory Pattern
3. Singleton Pattern
4. Prototype Pattern
5. Builder Pattern.

## 2. Structural Design Pattern

1. Adapter Pattern
2. Bridge Pattern
3. Composite Pattern
4. Decorator Pattern
5. Facade Pattern
6. Flyweight Pattern
7. Proxy Pattern

## 3. Behavioral Design Pattern

1. Chain Of Responsibility Pattern
2. Command Pattern
3. Interpreter Pattern
4. Iterator Pattern
5. Mediator Pattern
6. Memento Pattern
7. Observer Pattern
8. State Pattern
9. Strategy Pattern
10. Template Pattern
11. Visitor Pattern

# Creational design patterns

Creational design patterns are concerned with **the way of creating objects.** These design patterns are used when a decision must be made at the time of instantiation of a class (i.e. creating an object of a class).

# Factory Method Pattern

A Factory Pattern or Factory Method Pattern says that just **define an interface or abstract class for creating an object but let the subclasses decide which class to instantiate.** In other words, subclasses are responsible to create the instance of class.

The Factory Method Pattern is also known as **Virtual Constructor.**

#### Advantage of Factory Design Pattern

* 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.

#### Usage of Factory Design Pattern

* When a class doesn't know what sub-classes will be required to create
* When a class wants that its sub-classes specify the objects to be created.
* When the parent classes choose the creation of objects to its sub-classes.

# Abstract Factory Pattern

“**Abstract Factory is a creational design pattern that lets you produce families of related objects without specifying their concrete classes**.”

Abstract Factory Pattern says that just **define an interface or abstract class for creating families of related (or dependent) objects but without specifying their concrete (implementation**) **sub-classes.**That means Abstract Factory lets a class returns a factory of classes. So, this is the reason that Abstract Factory Pattern is one level higher than the Factory Pattern.

An Abstract Factory Pattern is also known as **Kit.**

#### Advantage of Abstract Factory Pattern

* Abstract Factory Pattern isolates the client code from implementation classes.
* It eases the exchanging of object families.
* It promotes consistency among objects.

#### Usage of Abstract Factory Pattern

* When the system needs to be independent of how its object are created, composed, and represented.
* When the family of related objects has to be used together, then this constraint needs to be enforced.
* When you want to provide a library of objects that does not show implementations and only reveals interfaces.
* When the system needs to be configured with one of a multiple family of objects.

# 

# Singleton design pattern in Java

Singleton Pattern says that just **"define a class that has only one instance and provides a global point of access to it".**

There are two forms of singleton design pattern

* **Early Instantiation:** creation of instance at load time.
* **Lazy Instantiation:** creation of instance when required.

#### Advantage of Singleton design pattern

* Saves memory because object is not created at each request. Only single instance is reused again and again.

#### Usage of Singleton design pattern

* Singleton pattern is mostly used in multi-threaded and database applications. It is used in logging, caching, thread pools, configuration settings etc.

#### How to create Singleton design pattern?

To create the singleton class, we need to have static member of class, private constructor and static factory method.

* **Static member:** It gets memory only once because of static, itcontains the instance of the Singleton class.
* **Private constructor:** It will prevent to instantiate the Singleton class from outside the class.
* **Static factory method:** This provides the global point of access to the Singleton object and returns the instance to the caller.

#### If singleton class is loaded by two classloaders, two instance of singleton class will be created, one for each classloader.

### Significance of Serialization in Singleton Pattern

If singleton class is Serializable, you can serialize the singleton instance. Once it is serialized, you can deserialize it but it will not return the singleton object.

To resolve this issue, you need to override the **readResolve() method of Serializable**  that enforces the singleton. It is called just after the object is deserialized. It returns the singleton object.

# Prototype Design Pattern

Prototype Pattern says that **cloning of an existing object instead of creating new one and can also be customized as per the requirement**.

#### Advantage of Prototype Pattern

The main advantages of prototype pattern are as follows:

* It reduces the need of sub-classing.
* It hides complexities of creating objects.
* The clients can get new objects without knowing which type of object it will be.
* It lets you add or remove objects at runtime.

#### Usage of Prototype Pattern

* When the classes are instantiated at runtime.
* When the cost of creating an object is expensive or complicated.
* When you want to keep the number of classes in an application minimum.
* When the client application needs to be unaware of object creation and representation.

# Builder Design Pattern

Builder Pattern says that **"construct a complex object from simple objects using step-by-step approach"**

It is mostly used when object can't be created in single step like in the de-serialization of a complex object.

#### Advantage of Builder Design Pattern

The main advantages of Builder Pattern are as follows:

* 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.

# Object Pool Pattern

Object Pool Pattern says that **" to reuse the object that are expensive to create".**

Basically, an Object pool is a container which contains a specified amount of objects. When an object is taken from the pool, it is not available in the pool until it is put back. **Objects in the pool have a lifecycle: creation, validation and destroy.**

A pool helps to manage available resources in a better way. There are many using examples: especially in application servers there are data source pools, thread pools etc.

#### Advantage of Object Pool design pattern

* It boosts the performance of the application significantly.
* It is most effective in a situation where the rate of initializing a class instance is high.
* It manages the connections and provides a way to reuse and share them.
* It can also provide the limit for the maximum number of objects that can be created.

# Structural design patterns

**Structural design patterns** are concerned with how classes and objects can be composed, to form larger structures.

The structural design patterns **simplifies the structure by identifying the relationships**.

These patterns focus on, how the classes inherit from each other and how they are composed from other classes.

## Types of structural design patterns

There are following 7 types of structural design patterns.

1. [Adapter Pattern](https://www.javatpoint.com/adapter-pattern)Adapting an interface into another according to client expectation.
2. [Bridge Pattern](https://www.javatpoint.com/bridge-pattern)Separating abstraction (interface) from implementation.
3. [Composite Pattern](https://www.javatpoint.com/composite-pattern)Allowing clients to operate on hierarchy of objects.
4. [Decorator Pattern](https://www.javatpoint.com/decorator-pattern)Adding functionality to an object dynamically.
5. [Facade Pattern](https://www.javatpoint.com/facade-pattern)Providing an interface to a set of interfaces.
6. [Flyweight Pattern](https://www.javatpoint.com/flyweight-pattern)Reusing an object by sharing it.
7. [proxy Pattern](https://www.javatpoint.com/proxy-pattern)Representing another object.

# Adapter Pattern

An Adapter Pattern says that just **"converts the interface of a class into another interface that a client wants".**

In other words, to provide the interface according to client requirement while using the services of a class with a different interface.

The Adapter Pattern is also known as **Wrapper.**

#### Advantage of Adapter Pattern

* It allows two or more previously incompatible objects to interact.
* It allows reusability of existing functionality.

#### Usage of Adapter pattern:

It is used:

* When an object needs to utilize an existing class with an incompatible interface.
* When you want to create a reusable class that cooperates with classes which don't have compatible interfaces.
* When you want to create a reusable class that cooperates with classes which don't have compatible interfaces.

# Bridge Pattern

**Separating interface from implementation.**

A Bridge Pattern says that just **"decouple the functional abstraction from the implementation so that the two can vary independently".**

The Bridge Pattern is also known as **Handle or Body.**

#### Advantage of Bridge Pattern

* It enables the separation of implementation from the interface.
* It improves the extensibility.
* It allows the hiding of implementation details from the client

#### Usage of Bridge Pattern

* When you don't want a permanent binding between the functional abstraction and its implementation.
* When both the functional abstraction and its implementation need to extended using sub-classes.
* It is mostly used in those places where changes are made in the implementation does not affect the clients.

# Composite Pattern

A Composite Pattern says that just **"allow clients to operate in generic manner on objects that may or may not represent a hierarchy of objects".**

#### Advantage of Composite Design Pattern

* It defines class hierarchies that contain primitive and complex objects.
* It makes easier to you to add new kinds of components.
* It provides flexibility of structure with manageable class or interface.

#### Usage of Composite Pattern

It is used:

* When you want to represent a full or partial hierarchy of objects.
* When the responsibilities are needed to be added dynamically to the individual objects without affecting other objects. Where the responsibility of object may vary from time to time.

# Decorator Pattern

A Decorator Pattern says that just **"attach a flexible additional responsibilities to an object dynamically".**

In other words, **The Decorator Pattern uses composition instead of inheritance to extend the functionality of an object at runtime.**

The Decorator Pattern is also known as **Wrapper.**

#### Advantage of Decorator Pattern

* 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.

#### Usage of Decorator Pattern

It is used:

* When you want to transparently and dynamically add responsibilities to objects without affecting other objects.
* When you want to add responsibilities to an object that you may want to change in future.
* Extending functionality by sub-classing is no longer practical.

AD