**Java Design Patterns**

**Introduction**

A developer will build software/an application to meet/solve the requirements of an enterprise or a business firm using some programming language. While developing the applications they might use any programing language of their chose like C, C++ or Java etc.

These programming language provides API’s to the developer in building the components, but they never document the best practices, bad practices or design considerations that a developer needs to follow

What a pattern exactly does. It helps you in identifying the recurring problems and provides a pre-built solution that can be applied at its best in solving those problems.

The first efforts in documenting the problem and their solution have been done in 1970’s by Christopher Alexander. He is a civil engineer and architect, has document various in his area in several books.

The software community subsequently adopted the idea of pattern based on his work. Patterns in the software were popularized by the book Design Patterns:

Elements of reusable Object-Oriented Software by Erich Gamma, Richard Helm, Ralph, and John Vlissides (also known as the Gang of Four, or GOF)

**What is a pattern?**

Patterns are about documenting a solution for a well know (recurring) problem in a particular context. It can also be defined as recurring solution to a problem in a context. Let me elaborate the things First, what is a context? A context is an environment, surroundings or situations under which something exits. What is a problem? A problem is something that needs to be resolved, Solution? It is the answer to the problem in a context that helps resolve the issue.

**GOF Pattern Catalog**

When it comes to GOF pattern they grouped together related patterns and came up with catalog of patterns as shown below.

**Creational Patterns**

It provides guidelines to instantiate a single or group of objects so called Creational Patterns

* **Singleton**
* **Abstract Factory**
* Builder
* **Factory Method**
* Prototype
* **Factory**

**Structural Patterns**

It provides a manner to define relationship between classes

* Adapter
* Bridge
* Composite
* Decorator
* Façade
* Flyweight
* **Proxy**

**Behavioral Patterns**

It defines communication between classes and objects

* Chain of responsibility
* **Command**
* Interpreter
* Iterator
* Mediator
* Memento
* Observer
* State
* **Strategy**
* Template Method
* Visitor

Above all are core / GOF patterns like we do have design patterns in **Presentation Tier**, **Business Tier** and **Integration Tier**

For example

**Presentation Tier –** Intercepting Filter, View Helper etc.

**Business Tier –** Business Delegate, Service Locator etc.

**Integration Tier –** Data access Object, Service Activator etc.

**1. Singleton Design Pattern**

A singleton design pattern is the most popular and the older design pattern within the design pattern catalog.

if you create a class as singleton, an application allows only one instance of that class. generally, we create a class as singleton when we want global point of access to that instance.

**How to create a class as singleton?**

1. Declare the constructor of the class as private.

2. Declare a static method.

3. Declare a static member of the same class type in the class.

**When to use singleton**

1. When a class has absolutely zero state.

2. When a class has some state and it has some methods.

* the methods of the class are using the state of the class. but the state the class contains is

completely read-only.

3. When a class has some state and it has some methods.

* the methods of the class are using the state of the class. the state the class is not read-only

rather the state is a sharable state.

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

Ex: -

1. Runtime

2. Business Delegate

3. Service Locater

**2.Factory Method Design pattern**

Every factory class has a method, it contains the logic for creating the object of another class,

so it is called factory method. generally, these methods will be declared as static to let you call

without creating the object of factory.

factory method is used for creating the object for family of related classes within

the hierarchy.

**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.

**3.Abstract factory**

Abstract factory can be treated as a super factory or a factory of factories.

using factory design pattern, we abstract the creation process of another class.

using the Abstract factory pattern we Abstract the creation of family of classes.

**Advantage of Abstract Factory Pattern**

Abstract Factory Pattern isolates the client code from concrete (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.

**Diff bet Abstract Factory and factory-method**

1. abstract-factory pattern delegates the responsibility of object instantiation to another object via composition.

2. factory-method pattern uses inheritance and relies on subclasses to handle the desired object instantiation.

**4.Strategy Design pattern**

Strategy design pattern lets you build software as loosely coupled collection of interchangeable parts, in contract with tightly coupled system. this loosely coupling makes your software much flexible, extensible, maintainable and reusable.

Strategy design pattern recommends mainly three principles every application should follow to get the benefits those are as

1. favor composition over inheritance

2. always design to interfaces never design to concrete classes.

3. code should be open for extension and closed for modification.

**5.** **Command**

Command design pattern is the one behavioral design pattern from Gang of design patterns. It is used to encapsulate a request as an object and pass to an invoker. Invoker doesn’t know how to service the request but uses encapsulated command object to perform the action.

Typically, in a command design pattern there are five actors involved these are as follow

1. Command: - It is an interface with execute method. It acts as a contract.
2. Client: - Client instantiates a concrete command object and associate it with a receiver.
3. Invoker: - He instructs the command to perform an action.
4. Concrete Command: - Associate a binding between receiver and action.
5. Receiver: - It is the object that knows the actual steps to perform the action.

Let’s consider an example to understand it. For example, Power-On and Power-Off are the commands, to turn on/off the television. These commands are received by the television. You will issue these commands using the remote controller who acts as an Invoker. Client is the person who uses this remote control.

The advantage of this invoker is decoupled by the action performed by the receiver. The invoker has no knowledge of the receiver. The invoker issues a command wherein the command performs the action on a receiver. The invoker doesn’t know the details of the action being performed. So changes to the receiver action don’t affect the invoker action.

# 6. Proxy Pattern

Simply, proxy means an object representing another object.

According to GoF, a Proxy Pattern "provides the control for accessing the original object".

So, we can perform many operations like hiding the information of original object, on demand loading etc.

Proxy pattern is also known as **Surrogate or Placeholder.**

#### **Advantage of Proxy Pattern**

* It provides the protection to the original object from the outside world.