HowToDoInJava

Design Patterns

Last Updated: October 22, 2020

Design patterns, as name suggest, are solutions for most commonly (and frequently) occurred problems while designing a software. These patterns are mostly "evolved" rather than "discovered". A lot of learning, by lots of professional, have been summarized into these design patterns. None of these patterns force you anything in regard to implementation; they are just guidelines to solve a particular problem – in a particular way – in particular contexts. Code implementation is your responsibility.

Being so much of importance, let's learn these design patterns (in context of java) in more detail.

1. Creational Design Patterns

Creational patterns often used in place of direct instantiation with constructors. They make the creation process more adaptable and dynamic. In particular, they can provide a great deal of flexibility about which objects are created, how those objects are created, and how they are initialized.

Design Pattern Name	Builder
Purpose	Builder design pattern is an alternative way to construct complex objects and should be used only when we want to build different types of immutable objects using same object building process.
Design Pattern Name	Prototype
Purpose	Prototype design pattern is used in scenarios where application needs to create a large number of instances of a class, which have almost same state or differ very little.

Design Pattern Name	Factory
Purpose	Factory design pattern is most suitable when complex object creation steps are involved. To ensure that these steps are centralized and not exposed to composing classes.
Design Pattern Name	Abstract factory
Purpose	Abstract factory pattern is used whenever we need another level of abstraction over a group of factories created using factory pattern.
Design Pattern Name	Singleton
Purpose	Singleton enables an application to have one and only one instance of a class per JVM.

2. Structural Design Patterns

Structural design patterns show us how to glue different pieces of a system together in a flexible and extensible fashion. These patterns help us guarantee that when one of the parts changes, the entire application structure does not need to change.

Design Pattern Name	Adapter
Purpose	An adapter convert the interface of a class into another interface clients expect. It lets classes work together that couldn't otherwise because of incompatible interfaces.
Design Pattern Name	Bridge
Purpose	Bridge design pattern is used to decouple a class into two parts – *abstraction* and it's *implementation* – so that both can evolve in future without affecting each other. It increases the loose coupling between class abstraction and it's implementation.
Design Pattern Name	Composite

Purpose	Composite design pattern helps to compose the objects into tree structures to represent whole-part hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.
Design Pattern Name	Decorator
Purpose	Decorator design pattern is used to add additional features or behaviors to a particular instance of a class, while not modifying the other instances of same class.
Design Pattern Name	Facade
Purpose	Facade design pattern provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.
Design Pattern Name	Flyweight
Purpose	Flyweight design pattern enables use sharing of objects to support large numbers of fine-grained objects efficiently. A flyweight is a shared object that can be used in multiple contexts simultaneously. The flyweight acts as an independent object in each context.
Design Pattern Name	Proxy
Purpose	In proxy design pattern, a proxy object provide a surrogate or placeholder for another object to control access to it. Proxy is heavily used to implement lazy loading related usecases where we do not want to create full object until it is actually needed.

3. Behavioral Design Patterns

Behavioral patterns abstract an action we want to take on the object or class that takes the action. By changing the object or class, we can change the algorithm used, the objects affected, or the behavior, while still retaining the same basic interface for client classes.

Design Pattern Name	Chain of responsibility
Purpose	Chain of responsibility design pattern gives more than one object an opportunity to handle a request by linking receiving objects together in form of a chain.
Design Pattern Name	Command
Purpose	Command design pattern is useful to abstract the business logic into discrete actions which we call commands. These command objects help in loose coupling between two classes where one class (invoker) shall call a method on other class (receiver) to perform a business operation.
Design Pattern Name	Interpreter
Purpose	Interpreter pattern specifies how to evaluate sentences in a language, programatically. It helps in building a grammar for a simple language, so that sentences in the language can be interpreted.
Design Pattern Name	Iterator
Purpose	Iterator pattern provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
Design Pattern Name	Mediator
Purpose	Mediator pattern defines an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets us vary their interaction independently.
Design Pattern Name	Memento
Purpose	Memento pattern is used to restore state of an object to a previous state. It is also known as snapshot pattern.
Design Pattern Name	Observer

Purpose	Observer pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. It is also referred to as the publish-subscribe pattern.
Design Pattern Name	State
Purpose	In state pattern allows an object to alter its behavior when its internal state changes. The object will appear to change its class. There shall be a separate concrete class per possible state of an object.
Design Pattern Name	Strategy
Purpose	Strategy pattern is used where we choose a specific implementation of algorithm or task in run time – out of multiple other implementations for same task.
Design Pattern Name	Template method
Purpose	Template method pattern defines the sequential steps to execute a multi-step algorithm and optionally can provide a default implementation as well (based on requirements).
Design Pattern Name	Visitor
Purpose	Visitor pattern is used when we want a hierarchy of objects to modify their behavior but without modifying their source code.

Happy Learning !!

5 thoughts on “Design Patterns”

Ravi

March 6, 2019 at 12:07 pm

Very good information really good starting for the learners.

Prakash

April 10, 2017 at 10:41 am

> And Facade, observer too.

Prakash

April 10, 2017 at 10:37 am

> Hi Lokesh ,
>
> Awesome blog . But could you include brief description of Strategy pattern with examples .

Sachin Bhardwaj

February 20, 2017 at 9:19 am

> Awesome Blog Lokesh. Thanks ! Simple and Best. Really helpful. Please update remaining TODO soon.

Ravi

August 9, 2016 at 5:29 pm

> Hi,

Very clear information. Really helpful to understand the Design patterns with real time examples.
Please update the remaining TODO

Comments are closed.

Search …