**Creational Patterns**

How do you want to create objects?

* [Factory Method/Template](https://stackabuse.com/creational-design-patterns-in-java#factorymethod)

It is a method that can able to generate and return Object of its own class or some other class.

java.util.Calendar#getInstance()

Example : In an organisation HR works as factory method. Here development team request type of resource need to HR. Based on request type, HR provide resource to Development team.

* [Abstract Factory](https://stackabuse.com/creational-design-patterns-in-java#abstractfactory)

It defines how to create factory of factories design pattern. Here each factory will follow factory method pattern.

javax.xml.parsers.DocumentBuilderFactory#newInstance()

Example : HP, Samsung and Dell laptops are uses Intel and AMD processor.

* [Builder](https://stackabuse.com/creational-design-patterns-in-java#builder)

It will take the all raw values form user (huge data) and creates one appropriate class object to handle it.

java.lang.StringBuilder

Example : Kitchen is a **Factory**, Chef is a **Builder** where waiter tell to chef "pizza with cheese, onion". Here chef exposed attributes but hidden implementation.

* [Prototype](https://stackabuse.com/creational-design-patterns-in-java#prototype)

For each request it will give one separate object. It will implements Clonable.

java.lang.Object#clone()

Example : initial status of chess game

* [Singleton](https://stackabuse.com/creational-design-patterns-in-java#singleton)

For first request it will create one object and from second request onwards it will references already created object

java.lang.Runtime#getRuntime()

Example : President of a country

**Structural Patterns**

This design patterns is all about Class and Object composition i.e. How do you want structure the software component.

* [Adapter](https://stackabuse.com/structural-design-patterns-in-java#adapter)

It deals with mismatches between interfaces , to resolve this we create one class implementing expected interface and extends the mismatched class. It indirectly converts user data to expected data.

Example : Power Adapters

* java.util.Arrays#asList()
* [Bridge](https://stackabuse.com/structural-design-patterns-in-java#bridge)

It deals with mismatches between abstract classes. Need to extend expected abstract class and pass mismatched class as construtor parameter to extended class.

* [Filter](https://stackabuse.com/structural-design-patterns-in-java#filter)

Modifying data before it reaching target class. Just like servlet filter.

* [Composite](https://stackabuse.com/structural-design-patterns-in-java#composite)

Treating group of objects as same manner or similar. Ex: Entity class and child class(OneToMany).

Example : File System in Operating Systems, Directories are composite and files are leaves. System call Open is single interface for both composite and leaf.

* [Decorator](https://stackabuse.com/structural-design-patterns-in-java#decorator) (gives additional features)

Dynamically generating class object as per the situation. Ex: Animal interface has multiple implementation get() will generates different class objects as per situation.

Example : 1) Adding discounts on an order 2) gun is a deadly weapon on it's own. But you can apply certain "decorations" to make it more accurate, silent and devastating.

* All subclasses of java.io.InputStream, OutputStream, Reader and Writer have a constructor taking an instance of same type.
* [Facade](https://stackabuse.com/structural-design-patterns-in-java#facade)

Implementing top level interface (high level), to work with this class we don’t need strong knowledge on implementation. Ex: Zoo Keeper and Animal feed App.

Example : Control Panel

* javax.faces.context.ExternalContext, which internally uses ServletContext, HttpSession, HttpServletRequest, HttpServletResponse, et
* [Flyweight](https://stackabuse.com/structural-design-patterns-in-java#flyweight)

It is memory management technique. Instead of creating multiple objects create one object and pass its references Ex: String constant pool and SimpleDateFormate.

Example : Dial tone

* java.lang.Integer#valueOf(int) (also on Boolean, Byte, Character, Short and Long)
* [Proxy](https://stackabuse.com/structural-design-patterns-in-java#proxy)

Creating temporary class object that represents real data, indirectly it has implementation how contact with real object and real data.

java.rmi.\*, the whole API actually.

Example : check book leaf, credit card, debit card are proxy for Money and a customer representative to order a product

**Behavioral Patterns**

This design patterns is all about Class's objects communication i.e How do you want a behavior in software?

* [Interpreter](https://stackabuse.com/behavioral-design-patterns-in-java#interpreter)

Created using interface . is used to convert data from one formate to another Ex: 9 to nine, nine to 9, Language conversion using Locale.

* [Template Method/Pattern](https://stackabuse.com/behavioral-design-patterns-in-java#templatemethod)

Deals with abstract classes, it is used to eliminate boiler plate problem. It already have implemented methods programmers need to implement only method with specific implementation. It will call all codes in sequence.

JMSTemplate HibernateTemplate and JdbcTemplate in Spring

* [Chain of Responsibility](https://stackabuse.com/behavioral-design-patterns-in-java#chainofresponsibility)

Strongly linked chain , decoupling is very difficult. Here responsibility is shared between various objects like 1) gatherring 2) processing data 3) Storing Data

Example : Loan or Leave approval process

* javax.servlet.Filter#doFilter()
* [Command](https://stackabuse.com/behavioral-design-patterns-in-java#command)

Takes all the incoming request data and creates one class object to store all these values.

Example : The "Guest Check" at a diner is an example of a Command pattern. The waiter or waitress takes an order or command from a customer and encapsulates that order by writing it on the check. The order is then queued for a short order cook. Note that the pad of "checks" used by each waiter is not dependent on the menu, and therefore they can support commands to cook many different items.

* All implementations of java.lang.Runnable
* [Iterator](https://stackabuse.com/behavioral-design-patterns-in-java#iterator)

Creating our own Iterator to create our own sequence by implementing Iterator interface and its methods.

Example : Next/Previous buttons on TV

* All implementations of java.util.Iterator & java.util.Enumeration
* [Mediator](https://stackabuse.com/behavioral-design-patterns-in-java#mediator)

It is like middle ware service and acts as adapter pattern. Reduces the direct interaction with client app. Ex user and Chat, kafka

Client🡪 middleware service🡪web server

Example : Air Traffic Controller(ATC)

* [Memento](https://stackabuse.com/behavioral-design-patterns-in-java#memento)

Storing the data for latter use Ex: calculating Execution time

Memento(contains data)🡪Originator (know where to store and retrive)🡪CareTaker (stored data)

Example : save the state in a game & Undo/Redo operation in Windows

* All implementations of java.io.Serializable
* [Observer](https://stackabuse.com/behavioral-design-patterns-in-java#observer)

If one object state changes then it should give alert to remaining people (OneToMany) Ex: Ceo alert goes to all employees , Tl alert goes to his team mates.

* [State](https://stackabuse.com/behavioral-design-patterns-in-java#state)

It deals with dynamically changing its behaviour or internal state .

Ex: order received, confirmed , delivered

Example : A fan wall control

* [Strategy](https://stackabuse.com/behavioral-design-patterns-in-java#strategy)

It dynamically changes its state and internal algorithoms as per state.

Ex: cancelling order form flipkart.

Example : Modes of transportation

* java.util.Comparator#compare(), executed by among others Collections#sort().
* javax.servlet.http.HttpServlet, the service() and all doXXX() methods take HttpServletRequest and HttpServletResponse and the implementor has to process them (and not to get hold of them as instance variables!).
* javax.servlet.Filter#doFilter()
* [Visitor](https://stackabuse.com/behavioral-design-patterns-in-java#visitor) (Defines a new operation to a class without change)

This pattern is used to track or monitor the group of objects just like counting visitors hit in web site. To listen all these things it should be accepted by all the group objects (Ex: monitoring only admin hits).

Example : Taxi

**J2EE Patterns**

* [MVC Pattern](https://stackabuse.com/java-j2ee-design-patterns/#mvcpattern)

This method is used to create loosely coupled nature in web applications. Here group of same items will be grouped together.

* [Business Delegate Pattern](https://stackabuse.com/java-j2ee-design-patterns/#businessdelegatepattern)

Reduces no.of direct hits to view layer. (separating business layer and view layer)

* [Composite Entity Pattern](https://stackabuse.com/java-j2ee-design-patterns/#compositeentitypattern)

It is graph of objects. If one object is replaced then it will replace remaining with it’s dependencies. Spring will replace Entire Ejb.

* [Data Access Object Pattern](https://stackabuse.com/java-j2ee-design-patterns/#dataaccessobjectpattern)

It is standard way to communicate with database. You should create interface and it’s implementation to achieve it. Ex: EmployeeDaoImpl with methods getAllEmployee(); GetEmployeeById();

* [Front Controller Pattern](https://stackabuse.com/java-j2ee-design-patterns/#frontcontrollerpattern)

It is first listener class that hears all the incoming requests and after processing it sends to its destination.

* [Intercepting Filter Pattern](https://stackabuse.com/java-j2ee-design-patterns/#interceptingfilterpattern)

Preprocessing data or request before it reaching its destination just like filters in servlets.

* [Service Locator Pattern](https://stackabuse.com/java-j2ee-design-patterns/#servicelocatorpattern)

Using this pattern we can decouple any method execution and awake our own service methods. Ex: caching in Dao class methods.

* [Transfer Object Pattern](https://stackabuse.com/java-j2ee-design-patterns/#transferobjectpattern)

Using this pattern we can transfer Object we can transfer data through over the network like implementing object as Serializable .