|  |
| --- |
| Chain of Responsibility |
| * Gives more than one object an opportunity to handle a request by linking receiving objects together. * Example: JDK implementation of exception handling. Use throws keyword to pass the request to the next exceptional handler. |
| Mediator Pattern |
| * **Too many relationships exist and common point of control or communication is needed.** * Example, group chat program. All users need to have a reference of mediator. Mediator maintains a list of users. When one of the users sends a message, the user actually uses the reference of the mediator to loop through all the users and send out the message to everybody. ([Detail](http://www.journaldev.com/1730/mediator-design-pattern-in-java-example-tutorial))           ChatMediator mediator = new ChatMediatorImpl();          User user1 = new UserImpl(mediator, "Pankaj");          User user2 = new UserImpl(mediator, "Lisa");          mediator.addUser(user1);          mediator.addUser(user2);            user1.send("Hi All"); |
| Observer Pattern |
| * **Other objects will be notified when one object’s state is changed.** * Example: can be found in almost every GUI env. When buttons or other components are placed in application, the application typically registers as a listener for those controls. When a user triggers an event, such as clicking a button, the control iterates through its registered observers and sends a notification to each. (one button can have multiple listeners. ) |
| Strategy Pattern |
| 1. Defines multiple algorithms that can be swapped to carry out a specific behavior 2. Example in JDK, [Arrays.sort().](http://www.programcreek.com/2013/11/arrays-sort-comparator/) By passing in different comparator, the sorting behavior is different.   *Dog d1 = new Dog(2, 50);*  *Dog d2 = new Dog(1, 30);*  *Dog d3 = new Dog(3, 40);*  *Dog[] dogArray = {d1, d2, d3};*  *printDogs(dogArray);*  *Arrays.sort(dogArray, new DogSizeComparator());*  *printDogs(dogArray);*  *Arrays.sort(dogArray, new DogWeightComparator());*  *printDogs(dogArray);* |
| Template Method |
| * Identifies the framework of an algorithm, allowing implementing classes to define the behavior. * Example in JDK. Any interface defined in java, the concrete classes need to implement the abstract method defined in the interface. |
| State Pattern |
| * This allows you easily change an object’s behavior at runtime based on internal state. * Example: [MP3 Player](http://java.dzone.com/articles/design-patterns-state), press play button. (State object has a reference to the context object too) |
| Command Pattern |
| * It allows the requester of a particular action to be decoupled from the object that performs the action. * Command will have common interface which has an abstract function execute all concrete command must impl. * Example: JDK: java.lang.Runnable |
| Iterator Pattern |
| * Allows access to the elements of an aggregate object without allowing access to its underlying representation. * Example: JDK, ArrayList and iterator. ArrayList will have iterator created. Client get reference of iterator by ArrayList. |
| Visitor Pattern |
| * Allows for one or more operations to be applied to a set of unrelated objects at runtime. * Each of the unrelated objects has a method to accept the visitor, and calls the visit operation in visitor in accept method. ([Detail](http://java.dzone.com/articles/design-patterns-visitor)) |
| Adapter Pattern |
| * Permits classes with disparate interfaces to work together by creating a common object. * Example, JDK : Arrays.asList() / java.io.InputStreamReader(InputStream) |
| Bridge Pattern |
| * Defines an abstraction object structure independently of the implementation object structure in order to limit coupling * [Example](http://www.programcreek.com/2011/10/java-design-pattern-bridge/): Java: JVM is the abstract implementor, the concrete JVM of specific OS is the concrete implementor. OS is the abstraction. Another example is the remote control as described below. |
| Façade Pattern |
| * Supplies a single interface to a set of interfaces within a system. * Example, in web services, one web service might provide access to a number of smaller services that have been hidden from the caller by the facade. Our CDX API service for getMembership is actually a façade. |
| Proxy Pattern |
| * Allows for object level access control by acting as a pass through entity or a place holder object. * Example: we have an address validation webservice called by CRM system. By providing an ESB proxy, we decoupled the CRM system from the actual address validation. The ESB is a pass through which uses the same WSDL of the implementation web service. |
| Factory Method Pattern |
| * Simply a method that returns an actual type. * Factory Methods are usually called within Template Methods. * Simply a method that returns an actual type.   + java.lang.Object. toString()   + java.lang.Class .forName() |
| Abstract Factory |
| * To provide a contract for creating families of related objects without having to specify their concrete classes. * Spring framework’s BeanFactory |
| Builder |
| * Allows for the dynamic creation of objects based upon easily interchangeable algorithms. (StringBuilder or [Example](http://www.tutorialspoint.com/design_pattern/builder_pattern.htm)) |
| Singleton |
| * Ensures that only one instance of a class is allowed within a system. |
| Fly Weight |
| * It is used to share large number of objects in an efficient way. * Used when Many similar objects are used and the storage cost is high * Example: the representation of a character in a word processor. ([Another Example](https://dzone.com/articles/design-patterns-flyweight)) |
| Memento |
| * Allows for capturing and externalizing an object's internal state so that it can be restored later * The Originator is the object that knows how to save itself: the class that you want to make stateful. * The Caretaker is that object that deals with when, and why, the Originator needs to save or restore itself. * The Memento holds the information about the Originator's state, and cannot be modified by the Caretaker. |
| Composite |
|  |
| Decorator |
| * Allows for the dynamic wrapping of objects in order to modify their existing responsibilities and behaviors. |
|  |