- Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.
- Abstract Factory patterns work around a super-factory which creates other factories. This factory is also called as factory of factories. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.
- This pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.
- A Builder class builds the final object step by step. This builder is independent of other objects.
- Adapter pattern works as a bridge between two incompatible interfaces. This type of design pattern comes under structural pattern as this pattern combines the capability of two independent interfaces.

  This pattern involves a single class which is responsible to join functionalities of independent or incompatible interfaces. A real life example could be a case of card reader which acts as an adapter between memory card and a laptop. You plugin the memory card into card reader and card reader into the laptop so that memory card can be read via laptop.

  We are demonstrating use of Adapter pattern via following example in which an audio player device can play mp3 files only and wants to use an advanced audio player capable of playing vlc and mp4 files.
- Bridge is used when we need to decouple an abstraction from its implementation so that the two can vary independently. This type of design pattern comes under structural pattern as this pattern decouples implementation class and abstract class by providing a bridge structure between them.

  This pattern involves an interface which acts as a bridge which makes the functionality of concrete classes independent from interface implementer classes. Both types of classes can be altered structurally without affecting each other.

  We are demonstrating use of Bridge pattern via following example in which a circle can be drawn in different colors using same abstract class method but different bridge implementer classes.
- Composite pattern is used where we need to treat a group of objects in similar way as a single object. Composite pattern composes objects in term of a tree structure to represent part as well as whole hierarchy. This type of design pattern

comes under structural pattern as this pattern creates a tree structure of group of objects.

- Decorator pattern allows a user to add new functionality to an existing object without altering its structure. This type of design pattern comes under structural pattern as this pattern acts as a wrapper to existing class.

  This pattern creates a decorator class which wraps the original class and provides additional functionality keeping class methods signature intact.

  We are demonstrating the use of decorator pattern via following example in which we will decorate a shape with some color without alter shape class.

- Facade pattern hides the complexities of the system and provides an interface to the client using which the client can access the system. This type of design pattern comes under structural pattern as this pattern adds an interface to existing system to hide its complexities.

  This pattern involves a single class which provides simplified methods required by client and delegates calls to methods of existing system classes.

- proxy pattern, a class represents functionality of another class. We are going to create an *Image* interface and concrete classes implementing the *Image* interface. *ProxyImage* is a a proxy class to reduce memory footprint of *RealImage* object loading.

- the chain of responsibility pattern creates a chain of receiver objects for a request. This pattern decouples sender and receiver of a request based on type of request. This pattern comes under behavioral patterns.

  In this pattern, normally each receiver contains reference to another receiver. If one object cannot handle the request then it passes the same to the next receiver and so on.

- Command pattern is a data driven design pattern and falls under behavioral pattern category. A request is wrapped under an object as command and passed to invoker object. Invoker object looks for the appropriate object which can handle this command and passes the command to the corresponding object which executes the command.

- Mediator pattern is used to reduce communication complexity between multiple objects or classes. This pattern provides a mediator class which normally handles all the communications between different classes and supports easy maintenance of the code by loose coupling. Mediator pattern falls under behavioral pattern category.

- Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified, its depenedent objects are to be notified automatically. Observer pattern falls under behavioral pattern category.
- State pattern, we create objects which represent various states and a context object whose behavior varies as its state object changes.

MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate application's concerns.

- **Model** - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.
- **View** - View represents the visualization of the data that model contains.
- **Controller** - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.
-