Types of Design Pattern

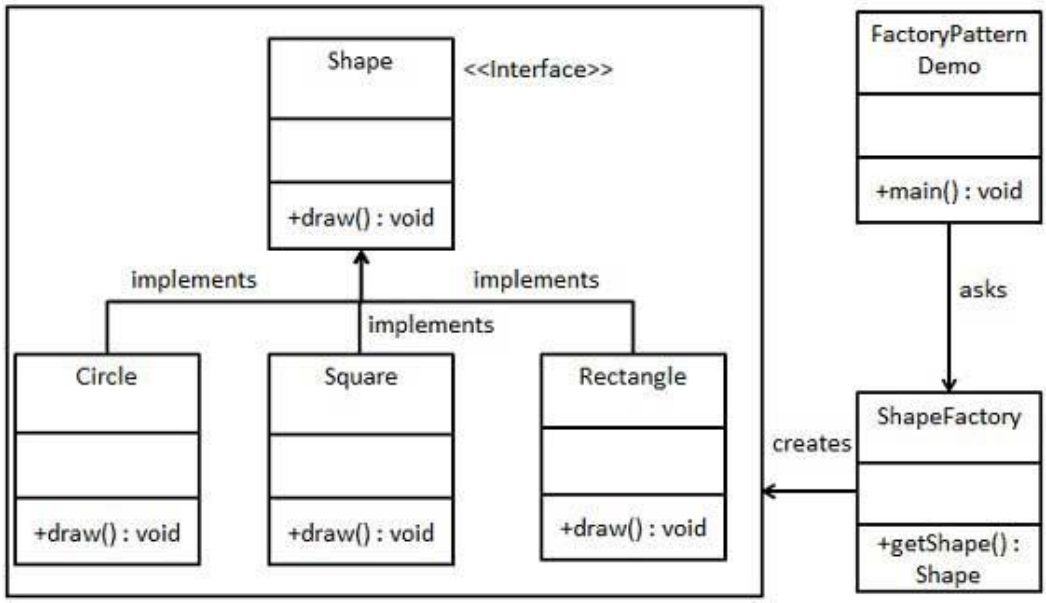
As per the design pattern reference book **Design Patterns - Elements of Reusable Object-Oriented Software** , there are 23 design patterns. These patterns can be classified in three categories: Creational, Structural and behavioral patterns. We'll also discuss another category of design patterns: J2EE design patterns.

S.N.	Pattern & Description
1	**Creational Patterns** These design patterns provides way to create objects while hiding the creation logic, rather than instantiating objects directly using new opreator. This gives program more flexibility in deciding which objects need to be created for a given use case.
2	**Structural Patterns** These design patterns concern class and object composition. Concept of inheritance is used to compose interfaces and define ways to compose objects to obtain new functionalities.
3	**Behavioral Patterns** These design patterns are specifically concerned with communication between objects.
4	**J2EE Patterns** These design patterns are specifically concerned with the presentation tier. These patterns are identified by Sun Java Center.

Factory Pattern

Factory pattern is one of most used design pattern in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

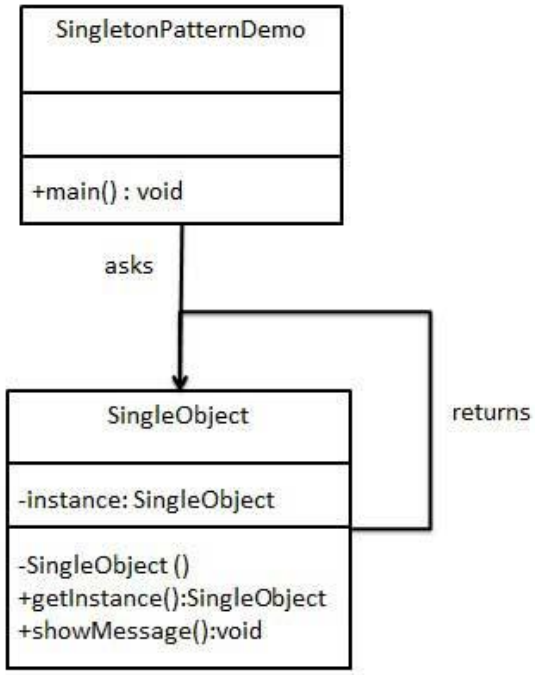
In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.



Singleton Pattern

Singleton pattern is one of the simplest design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best way to create an object.

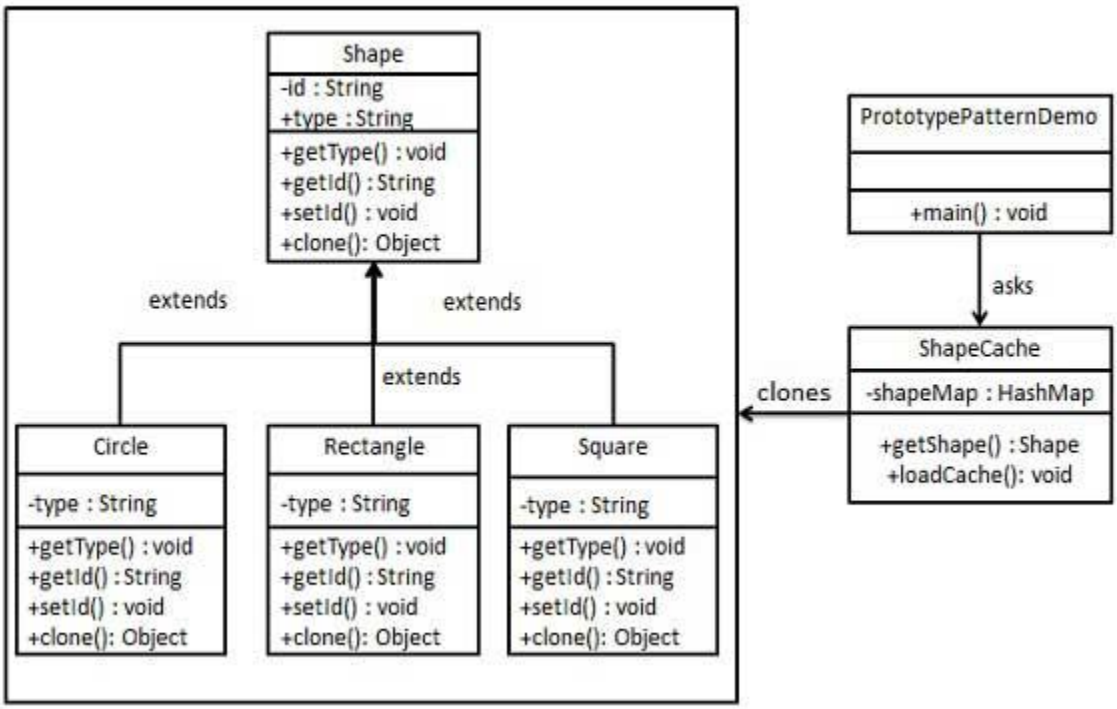
This pattern involves a single class which is responsible to creates own object while making sure that only single object get created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.



Prototype pattern

Prototype pattern refers to creating duplicate object while keeping performance in mind. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

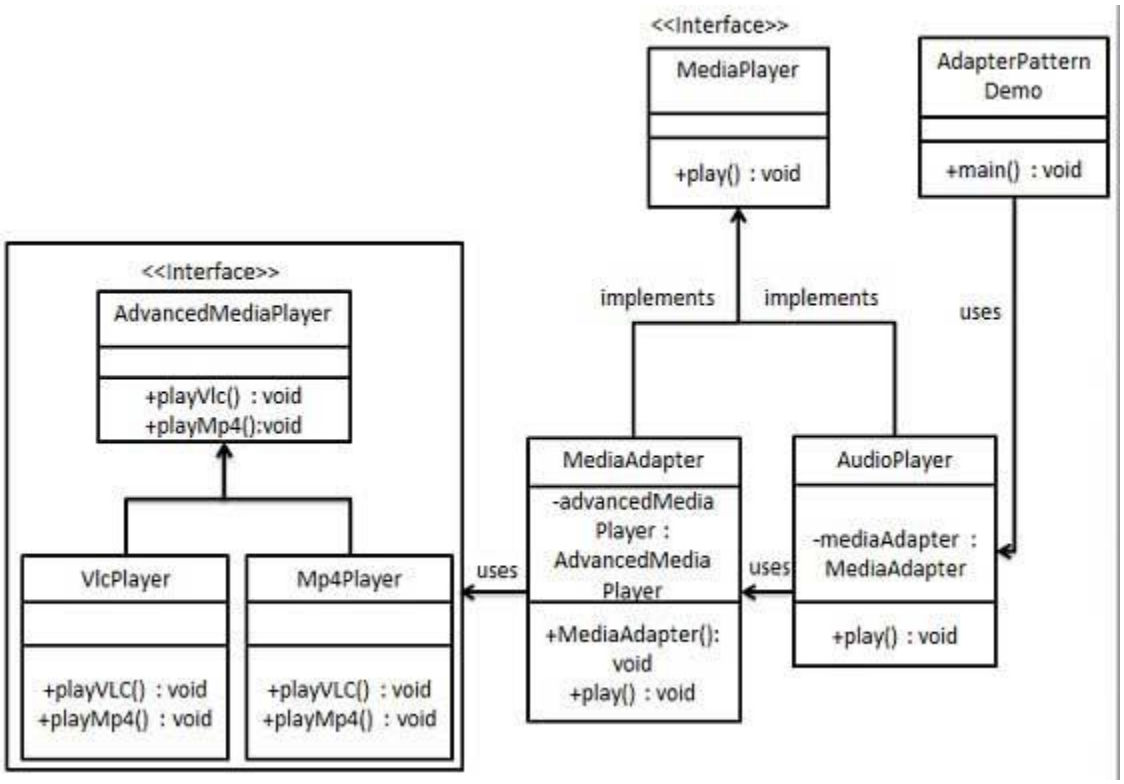
This pattern involves implementing a prototype interface which tells to create a clone of the current object. This pattern is used when creation of object directly is costly. For example, a object is to be created after a costly database operation. We can cache the object, returns its clone on next request and update the database as and when needed thus reducing database calls.



Adapter pattern

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.

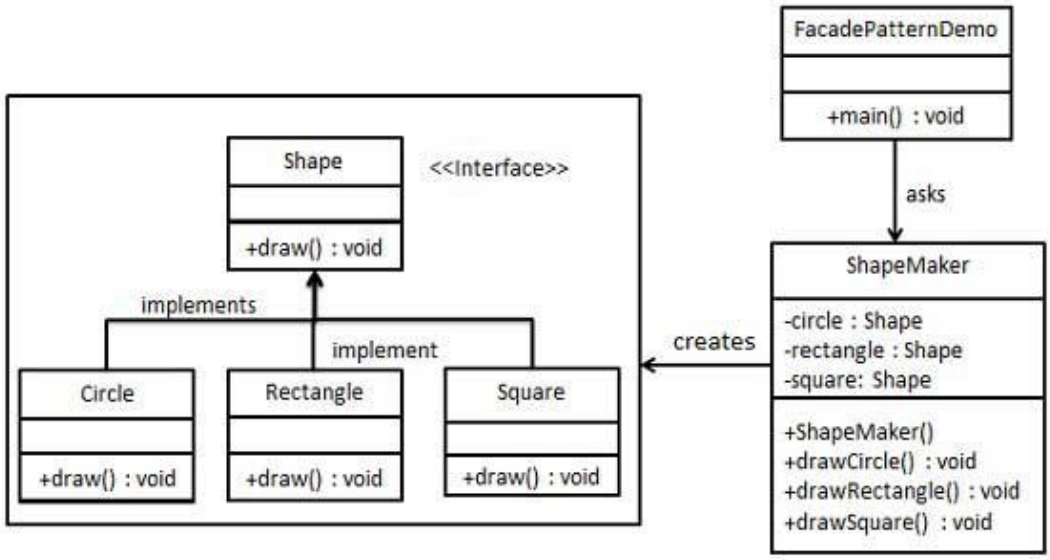
This pattern involves a single class which is responsible to join functionalities of independent or incompatible interfaces. A real life example could be a case of card reader which acts as an adapter between memory card and a laptop. You plugins the memory card into card reader and card reader into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



Facade pattern

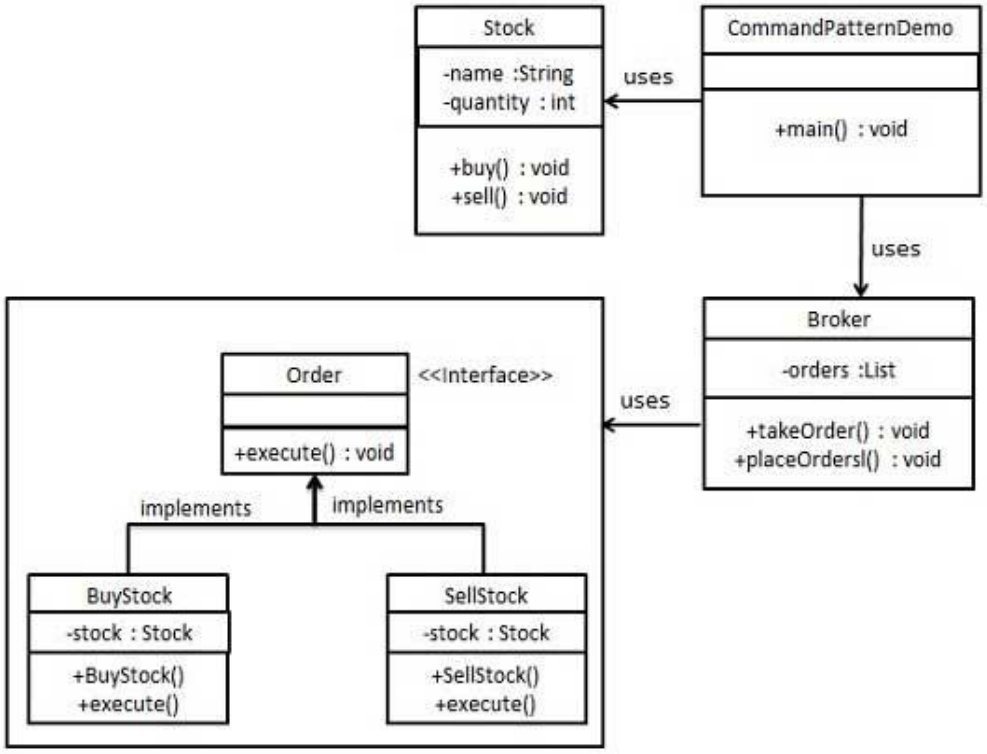
Facade pattern hides the complexities of the system and provides an interface to the client using which the client can access the system. This type of design pattern comes under structural pattern as this pattern adds an interface to exiting system to hide its complexities.

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



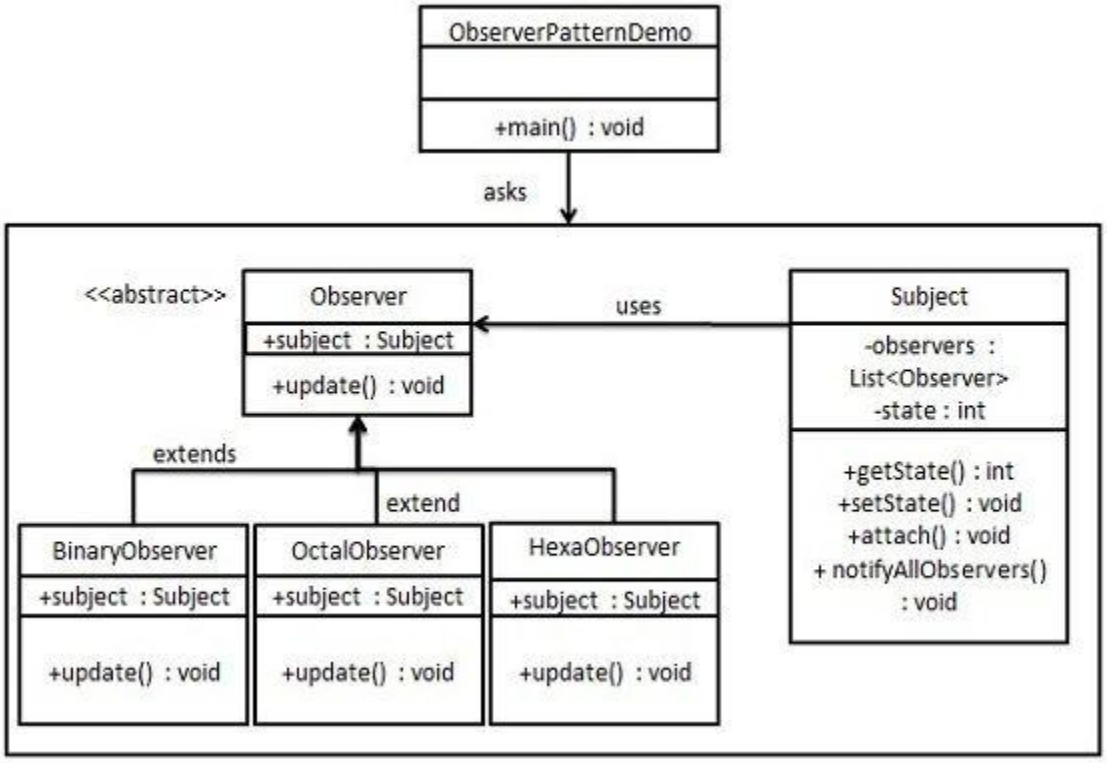
Command pattern

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



Observer pattern

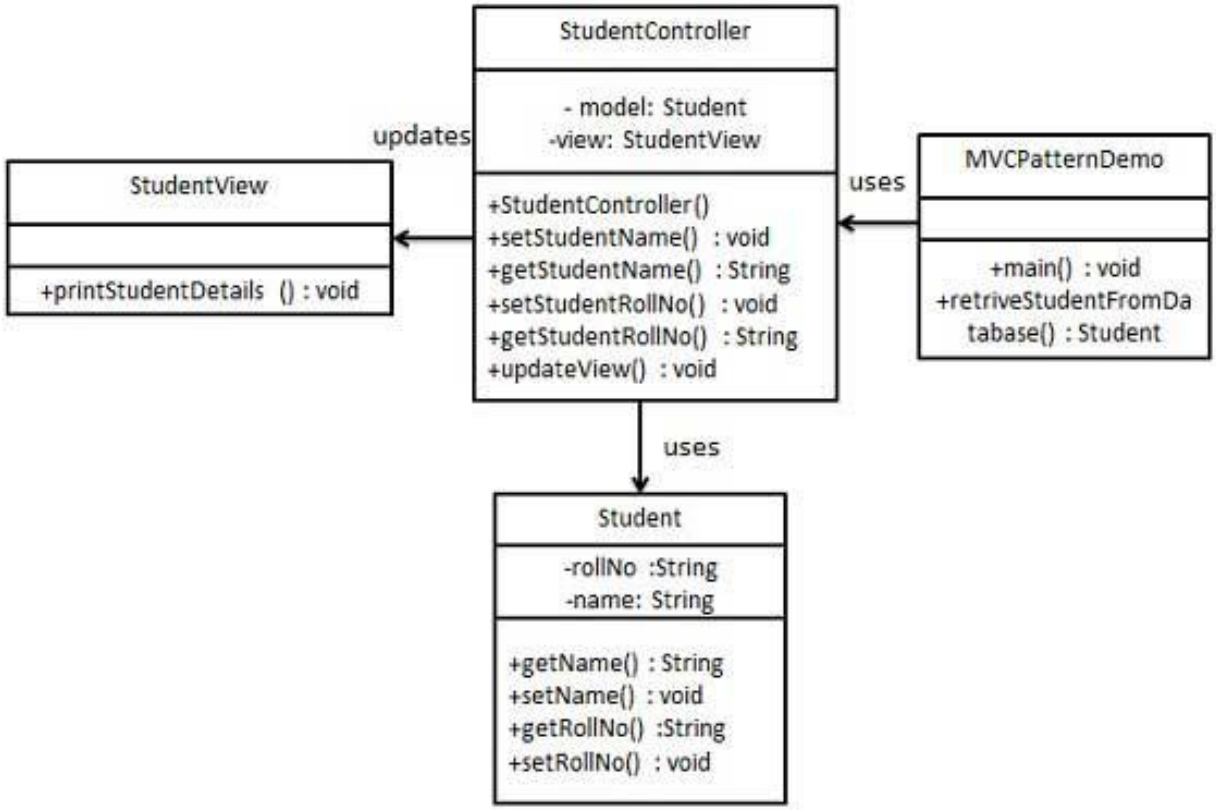
Observer pattern is used when there is one to many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically. Observer pattern falls under behavioral pattern category.



MVC Pattern

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

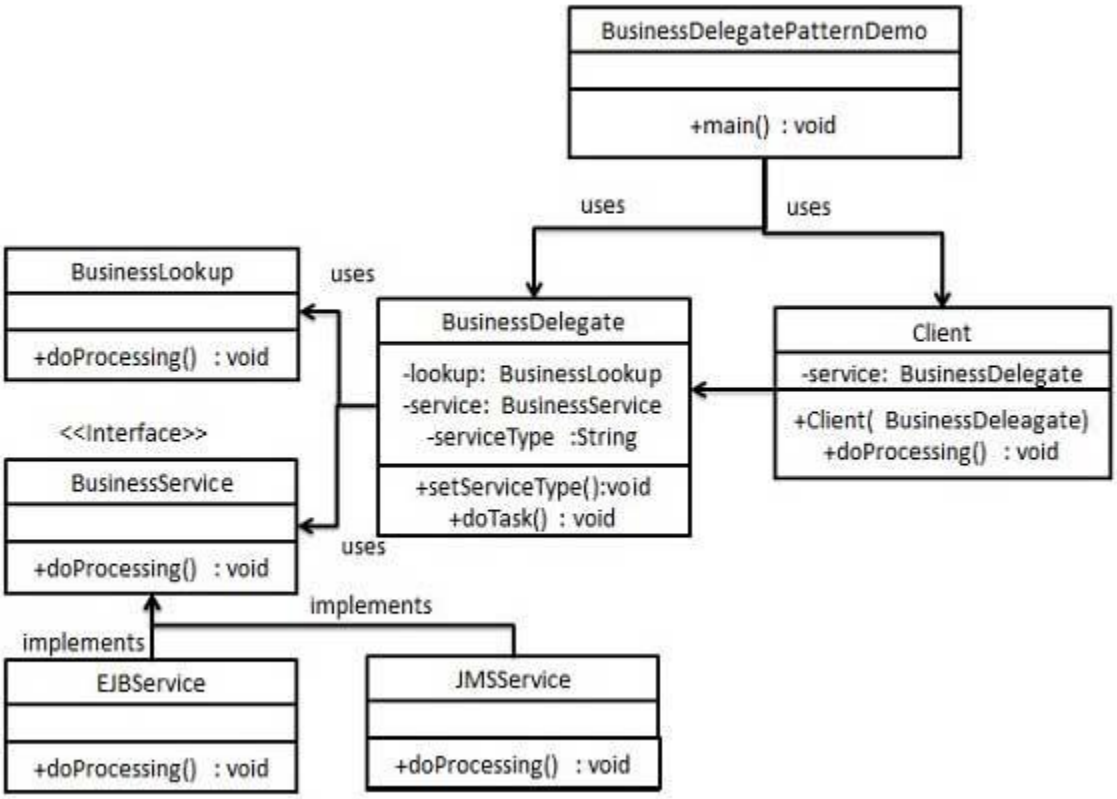
- **Model** - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.
- **View** - View represents the visualization of the data that model contains.
- **Controller** - Controller acts on both Model and view. It controls the data flow into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



Business Delegate Pattern

Business Delegate Pattern is used to decouple presentation tier and business tier. It is basically use to reduce communication or remote lookup functionality to business tier code in presentation tier code. In business tier we've following entities.

- **Client** - Presentation tier code may be JSP, servlet or UI java code.
- **Business Delegate** - A single entry point class for client entities to provide access to Business Service methods.
- **LookUp Service** - Lookup service object is responsible to get relative business implementation and provide business object access to business delegate object.
- **Business Service** - Business Service interface. Concrete classes implement this business service to provide actual business implementation logic.



Data Access Object Pattern

Data Access Object Pattern or DAO pattern is used to separate low level data accessing API or operations from high level business services. Following are the participants in Data Access Object Pattern.

- **Data Access Object Interface** - This interface defines the standard operations to be performed on a model object(s).
- **Data Access Object concrete class** -This class implements above interface. This class is responsible to get data from a datasource which can be database / xml or any other storage mechanism.
- **Model Object or Value Object** - This object is simple POJO containing get/set methods to store data retrieved using DAO class.