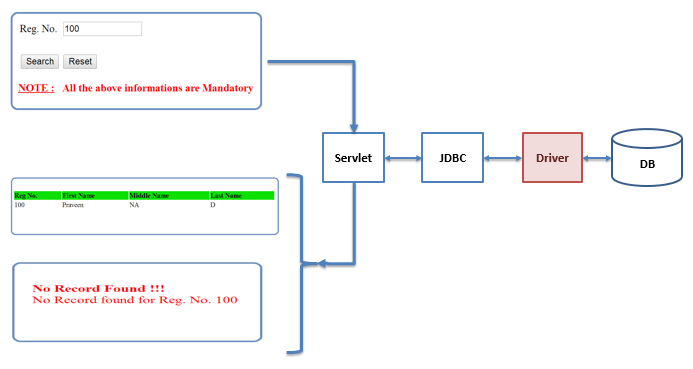
**Design Pattern**

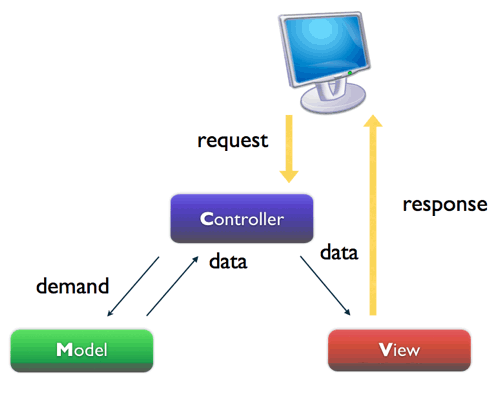


* Design pattern is a proven solution to a commonly occurring problem
* Design patterns promote easier program changes and code reusability.

They also help us to achieve loosely coupled architecture

* Few commonly used Design Patterns are :
  + Singleton Design Pattern
  + Model View Controller (MVC) Design Pattern
  + Data Transfer Object (DTO) Design Pattern
  + Data Access Objects (DAO) Design Pattern
  + Factory Design Pattern

**Model View Controller (MVC) Design Pattern:**

****

* This design pattern divides entire request & response flow into three components (Model, View, Controller) & it defines the interactions between them

1. Model (Data Model)

It’s the core part of the application & consists of the application business logic. In J2EE application plain java classes are used as models.

1. View

The View captures the information from the user & also displays the data to the user. It does not worry about what that data means, how to get that data or where to store that data etc. In J2EE application HTML / JSP’s are used as views

1. Controller

The controller controls both model & view i.e. it controls the entire request and response flow. It takes the request from view, calls the Model to get some data and passes the data to the View for displaying to the user. It’s like a principal of the college or like a real estate broker

* MVC Design Pattern helps us in achieving Readability, Maintainability, Flexibility, Reuseablity and Scalability of the application
* Each layer (i.e. Model layer, View layer & Controller layer) is independent of one another. So change in one layer doesn’t **affect much** on the other layer (i.e application is flexible)
* Easy to maintenance of the code and thus helps in future improvements (i.e application is scalable)
* Since each layer has distinct roles, the components of each layer are reusable (i.e application is easily maintainable)
* MVC architecture helps, development team to work on different parts of the application in parallel with minimal dependencies on each other.

**Disadvantages of MVC Design Pattern**

In case of a large scale application (like banking / shopping cart application) where there are many, many screens & hence We may end up in writing N number of controllers, which are hard to maintain or One controller which would grow to be very large and later is hard to maintain if that one controller handled all actions for the entire application

**Java Bean or POJO**

* If a Java class follow the below rules then it is called as Java Bean or Plain Old Java Object

(POJO),

1. public default constructor (this allows easy instantiation)
2. private class level variables
3. public getters & setters (this allows easy inspection and updating of an object state)
4. and does not have any other methods

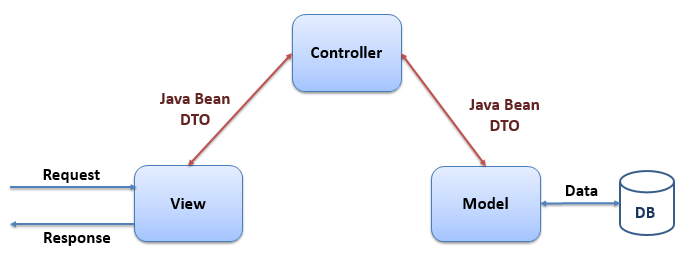
* The one & only use of a Java Bean is to help us to transfer the data from one program to

another program

**Plain Old Java Object (POJO)**

* It is an ordinary Java Object, not a special object
* The only difference between JavaBeans & POJO is that, JavaBeans must implement Serializable and have a no-argument constructor, whereas POJO does not have these restrictions

**Data Transfer Object (DTO) Design Pattern**

****

* Data transfer object (DTO), also known as Value Objects (VO), is a design pattern that

help us the transfer the data between one program to another program or one layer to

Another layer in the form of Java Object using Java Beans / POJO

* This way of passing the data is easy as compared to other ways like,

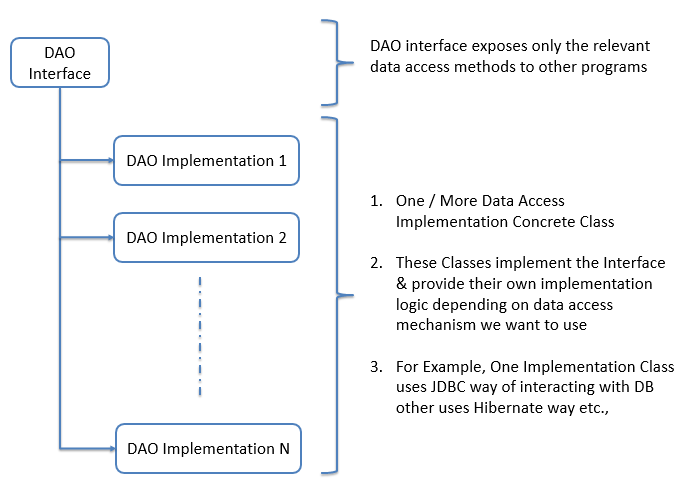
Arrays, ArrayList, HashMap etc.,

* Also this design pattern helps us to write error / bug free data transfer code in loosely

coupled fashion

* DTOs are often used in conjunction with DAOs to retrieve data from a database & send it to calling programs / clients.
* DTO does not have any other business logic (i.e. any other methods) other than getters & setters
* DTOs are often Serializable, in order to transfer data across different JVM’s

**Data Access Object Design Pattern:**

****

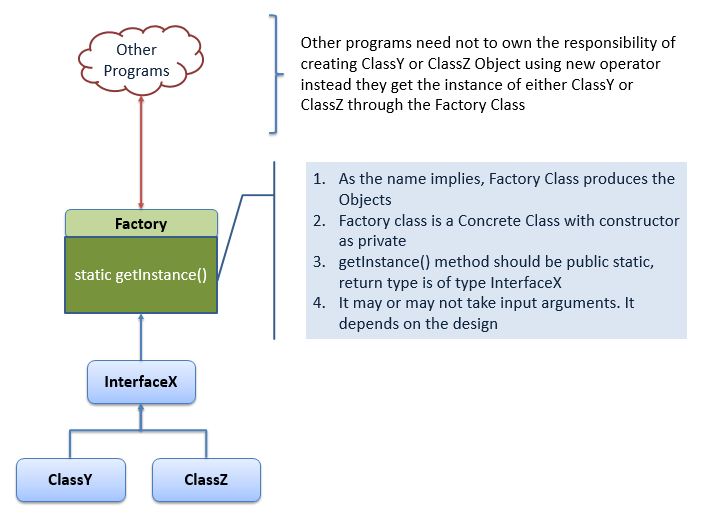
Different Data Access Mechanisms are: JDBC, Hibernate, Spring-JDBC, TopLink, JPA (Java Persistence API) or JDO (Java Data Objects) etc.,

* The purpose of DAOs is to help us to read from and write data to the database.
* It defines data access methods in an Interface &the corresponding implementation logic will be present in a concrete class which implement this interface. Therefore we can have “N” number of implementation class to align with different type of data access mechanism we want to use. Hence we can have “N” number of implementation classes with each having different data access mechanism
* As shown in the diagram, the DAO interface exposes only the relevant data access methods to the other programs by hiding implementation details
* With DAO design pattern it is quite easy to swap from one implementation to another with minimal impact on applicationbecause the interface exposed by the DAO to clients does not change when the underlying data handling logic changes

**Note:** We may have more than one DAO in our web application. It depends on the design of the application

For example : One DAO to handle user account related DB interactions, Another one for generating reports or documents, One more for data load operations etc.,

**Factory Design Pattern**



* Factory design pattern helps us to create objects without exposing the instantiation logic to the other programs. Hence other programs are totally decoupled from the implementation details
* This pattern is one of most commonly used design pattern in Java &provides one of the best ways to create an object

For ex: In JDBC creating the Connection Object makes use of Factory design pattern

Connection con = DriverManager.getConnection(“dbURL”);

Where Connection is an Interface

DriverManager is a concrete class with only one private default constructor

getConnection() is a public static method

* Whenever other programs needs an object, instead of creating it directly using the new operator, it approaches the factory for an object. The factory creates an object and then returns to the other programs by up-casting it to its interface type