**JAVA**

**Inheritance**: <https://www.javatpoint.com/inheritance-in-java>

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent classes.

The idea behind inheritance in Java is that you can create new classes

that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Types of Inheritance:

1. Single level -> class sub-class extends super-class
2. Multi-level -> class sub-class implements super

**Polymorphism**: <https://www.geeksforgeeks.org/polymorphism-in-java/>

Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations.

In Java polymorphism is mainly divided into two types:

* Compile-time Polymorphism - Method overloading
* Runtime Polymorphism - Method overriding

**Method Overloading**: If a [class](https://www.javatpoint.com/object-and-class-in-java) has multiple methods having same name but different in parameters, it is known as Method Overloading.

**Method overriding:**

If subclass (child class) has the same method as declared in the parent class, it is known asmethod overriding in Java.

**Encapsulation**: <https://www.geeksforgeeks.org/encapsulation-in-java/>

Binding the data with methods under a single unit Is known as Encapsulation. The variables of classes will be hidden from the other classes and can be accesses through the methods of their current class, therefore it is also known as data hiding.

To achieve encapsulation:

Declare variables as a private

Provide public setters and getters methods to modify and view the variables values.

**Abstraction:** [**https://www.geeksforgeeks.org/abstract-keyword-in-java/**](https://www.geeksforgeeks.org/abstract-keyword-in-java/)

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Two ways to achieve:

* + Abstract class
  + Interface

Points to remember:

An abstract class must be declared with an abstract keyword.

It can have abstract and non-abstract methods.

It cannot be instantiated.

It can have constructors and static methods also.

It can have final methods which will force the subclass not to change the implementation.

**Singleton Design pattern:**

Singleton design pattern restricts the instantiation of object and ensure that class has only one instance and provides a global point of access to it.

There are two types:

Early Instantiation -> creation of instances at load time.

Lazy Instantiation -> creation of instance when required.

Saves memory because object is not created at each request. Only single instance is used again and again.

Singleton pattern is mostly used in Logger class, multi-threaded and database applications.

**Eager Design pattern:**

1. Create private static instance for the object
2. Private constructor.
3. Public static getInstance method to return Instance.

**Lazy Design pattern:**

1. Private constructor.
2. Public static getInstance method.
3. Then we will check if the Instance is created or not, if not created then will create instance inside the getInstance method
4. Finally return the Instance.

**Spring Boot:**

Default port for spring boot is localhost:8080. For this you need to add 'spring boot web dependency'.

If you want to change the port number then you need to add property in Resources-> Application.yml - > server.port: 1010

After running application, spring boot search for index.html file from resource/static and execute in default browser.

To restart automatically, check the box in Preferences -> compiler -> auto build

**Spring MVC**: (Model view Controller)

MVC is an architectural design pattern.

Using MVC, we can develop our applications with **loosely coupling**

Using Spring MVC we can develop below 2 types of applications:

* + - * + Web Applications
        + Distributed applications

HTTP request HTTP response

1 10

Dispatcher Servlet (Front Controller)

2 3 4 5 6 7 8 9

View

View Resolver

Controller

Handler Mapper

**HTTP Request** -> **dispatcher servlet** (Dispatcher servlet acts as a Front controler.

The responsibility of Dispatcher servlet is to build

Spring container and then identify the) -> **controller**,which can handle the current request by using @RequestMapping annotation and the responsibility of controller is to build model by calling,the service layer logicand service layer will call the -> **DAO layer** and DAO layer will call -> **databse** and get the dataand delegate that data to Service layer.

Then service layer will delegate it to controller

layerController layer will build the model and add

data to the model object and return both View name

and Model object to Front controller

Then Front controller will contact view tresolver

ro resolve the view name and delegate the model to the view.