\***OBJECT ORIENTED PRGM**: creating objects that contains both data and methods.

**CLASS**:- its is a blueprint for creating objects

class consists of both data members and member function

**OBJECT**:- instance of class ,real time entity.

\*\***5 OOPS CONCEPTS**

\*\***ABSTRACTION**: the process of hiding implementation details and showing only essentials information to the user. Abstraction lets you focus on what the object does instead of how it does it.

**FOR EXAMPLE:-**We all use an ATM machine for cash withdrawal, money transfer, retrieve min-statement, etc. in our daily life. But we don't know internally what things are happening inside ATM machine when you insert an ATM card for performing any kind of operation.

\*\***ENCAPSULATION**: combining the both data members and member fuction in single unit.

**FOR EXAMPLE:-**When you log into your email accounts such as Gmail, Yahoo Mail, or Rediff mail, there is a lot of internal processes taking place in the backend and you have no control over it.

\*\***POLYMORPHISM:**- Ability to have more than one form.

**FOR EXAMPLE**:-A real-life example of polymorphism, a person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So the same person posses different behavior in different situations.

\* **STATIC BINDING:-**When type of the object is determined at compiled time(by the compiler), it is known as static binding.

\* **DYNAMIC BINDING:-**When type of the object is determined at run-time, it is known as dynamic binding.

\***METHODOVERLOADING:-**same method names with different parameters.

\***OVERRRIDDING:-**same method names and same parameters used in both parent and child classes.

\*\***MESSAGE PASSING:-**in Java is like sending an object i.e. message from one thread to another thread. It is used when threads do not have shared memory and are unable to share monitors or semaphores or any other shared variables to communicate.

\*\***INHERITANCE:-**inherit the properties form base class to derived class

\*\***SINGLE INHERITANCE:-** one parent class and one derived class.

\*\***MUTILEVEL INHERITANCE:-** one parent class and two derived classes.

\*\***HIERARCHIAL INHERITACE:-** one parent class and no. of derived classes.

\*\***EXTEND KEYWORD:-** to inherit the properties of base class through dervied class by using "EXTEND".

\*\*\*\*IN JAVA MUTIPLE AND HYBRID INHERITANCE POSSIBLE BY USING "INTERFA \*\***INTERAFACE CLASS:-** it has only one abstract method. we can have default and static methods, Private methods in an interface.we can not create object to interfaces.To inherit the properties from child class to parent class by using the "IMPLEMENTS"

**FOR EXAMPLE:-** interface class k

{

abstract sum();

}

class b implements k

{

public void sum()

}

\*\***ABSTRACT CLASS**:-A class which is declared as abstract is known as an abstract class. alteast one abstract method, we cannot create the objects to abstract classes+

**CONSTRUCTOR**:- special method ,class name and the method name must be equal.no return types. Every time an object is created using the new() keyword, at least one constructor is called.

\*\***default constructor**:- A constructor is called "Default Constructor" when it doesn't have any parameter.

\*\***paramerterised constructor**:- A constructor which has a specific number of parameters is called a parameterized constructor.

**Static Keyword**:-

**Arrays:-** array is a collection of elements with similar data types. Stored in a continuous memory allocation.

**Types of arrays:-**

One dimensional

2 dimensional

Mutli -dimensional

**Exceptional Handling:-** Used to handle run time exceptions.

**Types of exceptions:-**

1.**Checked:-**Exceptions that occur at compile time. These exceptions can be handled by try()

Catch() blocks

1. ClassNotFound
2. IllegalAccessException
3. NoSuchFiledException

2.**Unchecked:-**Exceptions that occur at runtime.

a. ArithmeticException

b. ArrayIndexOutOfBounds

c. NegativeArraySizeException

**try():-**

A try block contains a block of programmable statements with in which an exception might occur.

\*it contains the code that might throw an exception.

**Catch():-**

Catch() block must be associated with the try block. The corresponding catch block executes if an exception of a particular type occurs within try block.

\*it contains the exception handler for exceptions in try block.

**throw():-**

\* throw keyword is used throw an exception explicitly in the code, inside the function or the block of code.

**throws():-**

\*throws is used in the method declaration in order to explicitly specify the exceptions at a particular method might throw.

\*The throws keyword **indicates what exception type may be thrown by a method**. There are many exception types

**Diff b/w throw and throws**

\*The throws keyword is used to declare which exceptions can be thrown from a method, while the throw keyword is used to explicitly throw an exception within a method or block of code

**finally():-**The finally keyword is used **to create a block of code that follows a try block**. A finally block of code always executes, whether or not an exception has occurred.

**Multithreading:-**

\*Thread is a light-weight process. which shares a common memory space.

\*The process of executing multiple thread simultaneously.

**Thread life cycle:-**

**New:-**whenever we create object for thread then the thread is in new state.

**Active:-**when we call start() method the thread moves from new to active state where body gets attached.

**Blocked:-**when we call sleep(),wait(),join() methods then the thread goes from active to blocked state. Here body gets attached but its temporarily out of service.

a. **running:-**when the thread started execution then it is in running state.

b. **runnable:-** when we instantiate 3 threads then the one of the them is in running state and remaining are in runnable state.

**Death:-**completion of the thread process it is permanently out of service. Then it is called normal death of the thread.

**Maven:-**

\*it is a build tool used for adding the dependencies.

\*it mainly concentrates on logic not on the configuration.

\*it has 2 repositories:

1.**Local repository:-** it stores dependencies which are already used.

2.**Remote repository:-** It is used to fetch dependencies from the remote repository.

**Collections:-**

\*The Java collections framework is a set of classes and interfaces that implement commonly reusable collection data structures.

**Set:-** Set in Java Set is an interface which extends Collection. It is an unordered collection of objects in which duplicate values cannot be stored. Basically, Set is implemented by HashSet, LinkedHashSet or TreeSet (sorted representation). Set has various methods to add, remove clear, size, etc to enhance the usage of this interface.

**Hash Set:-**

\*HashSet is **commonly used if we have to access elements randomly**.

\*HashSet stores the elements by using a mechanism called hashing. HashSet contains unique elements only. HashSet allows null value.

\***HashSet is an unordered & unsorted collection of the data set**

**Sorted Set:-**A set is used to provide a particular ordering on its element.

**Linked Hash Set:-** it is an ordered and sorted collection of HashSet.

**Tree Set:-** TreeSet **provides an implementation of the Set interface that uses a tree for storage**. Objects are stored in a sorted and ascending order. Access and retrieval times are quite fast, which makes TreeSet an excellent choice when storing large amounts of sorted information that must be found quickly.

**Queues:-**

**Dequeue:-(Double ended queue)**

**Array Dequeue:-**

**Blocking Queue:-**

**List:-** List in Java provides the facility to maintain the ordered collection. It contains the index-based methods to insert, update, delete and search the elements. It can have the **duplicate elements** also. We can also store the **null elements** in the list.

\***ArrayList:-** Java **ArrayList** class uses a dynamic [*array*](https://www.javatpoint.com/array-in-java) for storing the elements. It is like an array, but there is no size limit. We can add or remove elements anytime. So, it is much more flexible than the traditional array

\***Iterator:-**An Iterator is **an object that can be used to loop through collections, like ArrayList and HashSet.**

An Iterator is an interface in Java and we can traverse the elements of a list in a forward direction

**ListIterator:-**  A ListIterator is an interface that extends the Iterator interface and we can traverse the elements in both forward and backward directions.

a.**LinkedList:-**

* Single Linked List:-
* Double linked List:-

**Map:-**Map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains unique keys. A Map is **useful if you have to search, update or delete elements on the basis of a key**.

**Hash Map:-**  **Elements of a HashMap are not in order, totally random,**

**Linked Hash Map:-**

**Elements of LinkedHashMap are ordered.**

The entries of a LinkedHashMap are in key insertion ordr, which is the order in which the keys are  inserted in the Map.

**Sorted Map:-** SortedMap is **an interface in the collection framework**. This interface extends the Map interface and provides a total ordering of its elements (elements can be traversed in sorted order of keys).

**Diff b/w Set and Map?**

A Set is an interface in Collection hierarchy that cannot contain duplicate elements whereas a Map is an interface that maps unique keys to values.

**Diff b/w HashSet and LinkedHashSet?**

HashSet is an unordered & unsorted collection of the data set, whereas the LinkedHashSet is an ordered and sorted collection of HashSet.

**Diff b/w HashMap and LinkedHashMap??**

The key difference between HashMap and LinkedHashMap is order. **Elements of a HashMap are not in order, totally random, whereas elements of LinkedHashMap are ordered**. The entries of a LinkedHashMap are in key insertion order, which is the order in which the keys are inserted in the Map.

**Diff b/w set and list??**

**Servlets:-**

\*Servlet is a technology which is used to create a web application.  
\*Servlet is an API that provides many interfaces and classes including documentation.  
\*Servlet is an interface that must be implemented for creating any Servlet.

**Life cycle of servlet:-**

**Init():-**Used to invoke servlet.

The **Servlet.init()** method is called by the Servlet container to indicate that this Servlet instance is instantiated successfully and is about to put into service.'

**Service():-**Invoked by the container to perform the actual task. It can be any http specific methods(doGet,doPost)

\* The service () method of the Servlet is invoked to inform the Servlet about the client requests

**Destroy():-**The **destroy()** method runs only once during the lifetime of a Servlet and signals the end of the Servlet instance.

**Methods:-**

**doGet():-**To get the information from the server.

**doPost():-**To give the information to the server.

**Dispatcher Servlet:-**

The DispatcherServlet is **an actual Servlet (it inherits from the HttpServlet base class), and as such is declared in the web.xml of your web application**. You need to map requests that you want the DispatcherServlet to handle, by using a URL mapping in the same web.xml file.

This is the**important configuration file where we need to specify the ViewResolver and View components**.

**JSP(Java server pages):-**

JavaServer page (JSP) is **a template for a Web page that uses Java code to generate an HTML document dynamically**. JSPs are run in a server-side component known as a JSP container, which translates them into equivalent Java servlets

JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

**JSP life cycle:-**

Translation of JSP page to Servlet  
Compilation of JSP page(Compilation of JSP into [test.java](http://test.java/))  
Classloading ([test.java](http://test.java/) to test.class)  
Instantiation(Object of the generated Servlet is created)  
Initialization(jspInit() method is invoked by the container)  
Request processing(\_jspService()is invoked by the container)  
JSP Cleanup (jspDestroy() method is invoked by the container)

**Diff b/w servlet and jsp??**

Servlet is a java code.JSP is a HTML based code.Writing code for servlet is harder than JSP as it is HTML in java.JSP is easy to code as it is java in HTML.Servlet plays a controller role in the hasMVC approach.JSP is the view in the MVC approach for showing output.Servlet is faster than JSP.JSP is slower than Servlet because the first step in JSP lifecycle is the translation of JSP to java code and then compile.Servlet can accept all protocol requests.JSP only accepts HTTP requests.In Servlet, we can override the service() method.In JSP, we cannot override its service() method.

**JPA(java persistence API):-**

\*JPA is a Java specification that is used to access, manage, and persist data between Java object and relational database.

* **Entity manager:-** The entity manager implements the API and encapsulates all of them within a single interface.
* Entity manager is used to read, delete and write an entity.
* An object referenced by an entity is managed by entity manager.

**Entity manager factory:-**

The **EntityManagerFactory** interface present in **java.persistence** package is used to provide an entity manager.

EntityManagerFactory emf=Persistence.createEntityManagerFactory("Student\_details");

* **Persistence -** The Persistence is a bootstrap class which is used to obtain an EntityManagerFactory interface.
* **createEntityManagerFactory() method -** The role of this method is to create and return an EntityManagerFactory for the named persistence unit. Thus, this method contains the name of persistence unit passed in the Persistence.xml file.

2) Obtaining an entity manager from factory.

1. EntityManager em=emf.createEntityManager();

* **EntityManager -** An EntityManager is an interface
* **createEntityManager() method -** It creates new application-managed EntityManager

3) Intializing an entity manager.

1. em.getTransaction().begin();

* **getTransaction() method -** This method returns the resource-level EntityTransaction object.
* **begin() method -** This method is used to start the transaction.

4) Persisting a data into relational database.

1. em.persist(s1);

* **persist() -** This method is used to make an instance managed and persistent. An entity instance is passed within this method.

5) Closing the transaction

1. em.getTransaction().commit();

6) Releasing the factory resources.

1. emf.close();
2. em.close();

* **close() -** This method is used to releasing the factory resources.

**JPQL:-** JPQL is **Java Persistence Query Language defined in JPA specification**. It is used to create queries against entities to store in a relational database. JPQL is developed based on SQL syntax. But it won't affect the database directly.

**JDBC:-**java data base connectivity.

**Java Database Connectivity** (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access a database. It is a Java-based data access technology used for Java database connectivity. It is part of the Java Standard Edition platform, from Oracle Corporation.

\*JDBC is a Java API that is used to connect and execute the query to the database. JDBC API uses JDBC drivers to connect to the database. JDBC API can be used to access tabular data stored into any relational database.

1.Fundamental Steps in JDBC  
2.Import JDBC packages.  
3.Load and register the JDBC driver.  
4.Open a connection to the database.  
5.Create a statement object to perform a query.  
6.Execute the statement object and return a query resultset.  
7.Process the resultset.  
8.Close the resultset and statement objects.  
9.Close the connection.

**JSTL(Java Standard Tag Library):-**

The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.

**Spring Core:-**

Spring is an open-source lightweight framework widely used to develop enterprise applications.

**Root context:-**

The root context in a Spring application is **the ApplicationContext that is loaded by the ContextLoaderListener.**

**Child Context:-**

The child-context in a Spring application is the ApplicationContext that is loaded by a DispatcherServlet.

**@Bean:-**when u start a application spring IOC will search for @configuration annotation then it will search for the annotation @Bean inside this @configuration class once it finds the @Bean then immediately spring ioc will create the object for the it and maintain main its life cycle for itself.

**@lazy:-**when we give @bean annotation then spring IOC will create an object for it. But we don’t know whether we are using it or not which increases the heap. So, to avoid this we can use the @lazy annotation so that the spring boot or bean create the object on demand only. Otherwise, just stay silent.

[**@Profile:-**](mailto:J.@Profile:-) it is used whenever u want to load some configurations specific to the environment.

[**@Autowired:-**](mailto:C.@Autowired:-) it will create the object for the class or interface and it is used for the communication and object response.

[**@Configuration:-**](mailto:.@Configuration:-)It is used when we want to use java base configuration. Java base configuration means we don’t want to handle the Spring Bean Lifecycle either using annotation or XML. So when we use @configuration it indicate that class can be used by a IOC container as a source of bean definition.

**Spring Boot:-Spring Boot is built on top of the conventional spring framework, widely used to develop**

**REST APIs**.

Spring Boot is **an open source, microservice based Java web framework**.

The Spring Boot framework creates a fully production-ready environment that is completely configurable using its prebuilt code within its codebase.

**Advantages of spring boot:-**

**\*** Fast and easy development of Spring-based applications;  
**\***No need for the deployment of war files;  
**\***The ability to create standalone applications;  
**\***Helping to directly embed Tomcat, Jetty, or Undertow into an application;  
**\***No need for XML configuration;  
**\***Reduced amounts of source code

**Custom exception:-**

Customized Exception Handling in Spring Boot Exceptions are one of many structures that maintain the control flow of a running application.

\* Custom exceptions**provide you the flexibility to add attributes and methods that are not part of a standard Java exception**.

**Spring MVC:-**

Spring MVC is a Model View, and Controller based web framework widely used

to develop web applications.

**Spring MVC life cycle/flow of MVC:-**



\*As displayed in the figure, all the incoming request is intercepted by the DispatcherServlet that works as the front controller.  
\*The DispatcherServlet gets an entry of handler mapping from the XML file and forwards the request to the controller.  
\*The controller returns an object of ModelAndView.  
\*The DispatcherServlet checks the entry of view resolver in the XML file and invokes the specified view component.

**@requestBody:-** The @RequestBody annotation **allows us to retrieve the request's body**. We can then return it as a String or deserialize it into a Plain Old Java Object (POJO). Spring has built-in mechanisms for deserializing JSON and XML objects into POJOs, which makes this task a lot easier as well.

**@responseBody:-** @ResponseBody annotation **tells a controller that the object returned is automatically serialized into JSON and passed back into the HttpResponse object**. In the developer console of our browser or using a tool like Postman, we can see the following response:

**@RequestMapping:-** @RequestMapping Annotation which is used to map HTTP requests to handler methods of MVC and REST controllers. In Spring MVC applications, the DispatcherServlet (Front Controller) is responsible for routing incoming HTTP requests to handler methods of controllers.

**@Controller:-** It is used to denote a Component as a Spring MVC Controller.

**@Rest Controller:-** it is the web layer where we can expose the Rest API.

@RestController is a sibling convenience annotation for creating Restful controllers. In the following application, we demonstrate the usage of @Controller. The application returns current data and time to the client. This is the project structure of the Spring application. This is the Maven pom.xml file.

**@Component scan:-** :-It simply scans the bean so that it will be visible to IOC(Inversion of Control)container.

Here we can provide base packages or base Classes in attribute field

**@component:-** It is main or base stereotype annotation and rest three are derived from these Annotation. But we are using all the 3 becoz we need to identify the class functionality and role of a Class.

[B.@Service:-It](mailto:B.@Service:-It) is used to write business logic.

[C.@RestController/@Controller:- It](mailto:C.@RestController/@Controller:-%20It) is the web layer where we can expose the Rest API.

[D.@Repository:- It](mailto:D.@Repository:-%20It) is used to write the database logic.

**@Transactional:-**

Transactional annotation **provides the application the ability to declaratively control transaction boundaries on CDI managed beans**, as well as classes defined as managed beans by the Java EE specification.

**@Request Param:-** RequestParam annotation is used to read the form data and bind it automatically to the parameter present in the provided method. So, it ignores the requirement of **HttpServletRequest** object to read the provided data.

\*Spring @RequestParam annotation is used to fetch the value of a parameter in the form request. In Spring MVC, "request parameters" map to query parameters, form data

**@Entity:-** @Entity - This is a marker annotation which indicates that this class is an entity. This annotation must be placed on the class name.

**@ModelAttribute:-** The @ModelAttribute is an annotation that binds a method parameter or method return value to a named model attribute and then exposes it to a web view.

**@BindingResult:-**  [ BindingResult] is Spring’s object that holds the result of the validation and binding and contains errors that may have occurred. The BindingResult must come right after the model object that is validated or else Spring will fail to validate the object and throw an exception.

**@valid:-** @Valid annotation **ensures the validation of the whole object**. Importantly, it performs the validation of the whole object graph.

\* The @Valid annotation can be added to variables in a rest controller method to validate them.

**@Validation:-** The Spring MVC Validation is used to restrict the input provided by the user. To validate the user's input, the Spring 4 or higher version supports and use Bean Validation API. It can validate both server-side as well as client-side applications.

**@range:-**

**@Size:-** It determines that the size must be equal to the specified [value.@size](mailto:value.@size)

**Diff b/w size and range??**

**@SessionAttribute:-** SessionAttribute annotation **retrieve the existing attribute from the session**. This annotation allows you to tell Spring which of your model attributes will also be copied to HttpSession before rendering the view.

**@Scope:-**The @Scope annotation indicates the name of the scope for the application. It can be used as a type-level or method-level annotation. If we declare the @Scope as a type-level, it must be used in conjunction with @Component or @Configuration annotation else used with @Bean annotation.

**Diff b/w @controller and @rest controller**

@Controller and @RestController in Spring MVC/BOOT. **The @Controller is a annotation to mark class as Controller Class in Spring While @RestController is used in REST Web services.**

**Diff b/w merge and persist??**

Persist should be called only on new entities, while merge is meant to reattach detached entities

**Diff b/w Statement vs prepared statement??**

Statement is used for executing simple SQL Statements whereas PreparedStatement is used for executing dynamic and pre-compiled SQL Statements.

**Diff b/w @not null and @not empty??**

\*NotNull﻿ The @NotNull annotation is, actually, an explicit contract declaring that: **A method should not return null**. **Variables (fields, local variables, and parameters) cannot hold a null value**.

\*NotNull: a constrained CharSequence, Collection, Map, or Array is valid as long as it's not null, but it can be empty. @NotEmpty: a constrained CharSequence, Collection, Map, or Array is valid as long as it's not null, and its size/length is greater than zero.

## View resolver/resolver:-

The ViewResolver **provides a mapping between view names and actual views**. The View interface addresses the preparation of the request and hands the request over to one of the view technologies.

**Request handler:-**

Request handlers are **the functions that handle the client request and construct a response**.

**Functional interface:-** An Interface that contains exactly one abstract method is known as functional interface. It can have any number of default, static methods but can contain only one abstract method

How to create custom repository validations used and

how to validate Modelattribute and Front end controller

How to convert bean to entity class

Diff b/w size and range

How view will auto populate the data.

**The controller annotated with @RequestMapping can have custom class argument(s) annotated with @ModelAttribute.**