OOPS

- ⇨ It's an Object-oriented programming system.
- ⇨ OOPS is concept not language.
- ⇨ Java is using this concept that why java is object-oriented programming language.
- ⇨ This concept is **designed** from our **day-to-day life**.
- ⇨ Because of this concept language **learning** and **understanding** will be easier.
- ⇨ This concept **deal** with classes and object.
- ⇨ This concept is designed and developed by some other person and it is implemented in various programming languages.
- ⇨ While development of project we should follow this concept, so that, code will be optimized.
- ⇨ There are Four major pillars in this concept: 1) Inheritance 2) Polymorphism 3) Abstraction 4) Encapsulation

1. What is Data Hiding

- ⇨ Outside person should **not access** our internal data directly.
- ⇨ So, we declaring data member is **private** we can **implement data hiding**.
- ⇨ The main advantage of data hiding is **Security**.

2. What is Abstraction?

- ⇨ **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.
- ⇨ We can use and apply **functionality** with their **essential knowledge** only, **no need to learn in detail** or in depth.
- ⇨ We can achieve security.
- ⇨ In java, we achieve abstraction by using "interface" and "abstract class".
- ⇨ There are several places where we will use abstraction
  - i) While development of project structure or layout.
  - ii) For creating an API.
  - iii) For Third Party Communication.

- ⇨ **API :-**
- ⇨ Application Programming Interface.
- ⇨ It is bunch of **rules**, **guidelines** and **specification**.
- ⇨ Through API uniform will be maintained.

- ⇨ **There are three different types of API :-**
  1) **In-built API**
     - There rules, guidelines are **written** by **Sunmicrosystem** and their **implementation** is also written by Sunmicrosystem.
     - e.g Multi-threading API, AWT API etc.
  2) **In-built API**
     - There rules, guidelines are written by Sunmicrosystem, but their implementations are written by **different vendors**.

- e.g JDBC API, Servlet API, JSP API, etc.

3) **Third-Party API**
   - There rules, guidelines are written by **different vendors** and their implementation is also written by **different vendors**.
   - e.g Spring API, Hibernate API, etc.

3. Why Java class doesn't support multiple inheritance?

⇨ There are chances of **ambiguity**, for avoiding that ambiguity Java doesn't support multiple inheritance.

4. What is Abstract Class?

⇨ **Hide internal implementation just the highlight the set of services** this concept is known as abstraction.
⇨ Abstract Class means its **partial** implemental class.
⇨ **No** one can **create object** of abstract class.
⇨ If class have any one method is **un-implemented** then this class must be had abstract class.
⇨ If class have all **implemented** method only then also make a class as a abstract.
⇨ Only **child class** is **implemented** the all-abstract class method.
⇨ If child class is **unable** to implement then that create **another child** to implement abstract method.
⇨ Constructor is present inside the abstract class, but constructors even when they are only called from **their concrete subclasses**. (If we do not define any constructor inside the abstract class then JVM will give a default constructor to the abstract class.)
⇨ Abstract method defines by abstract keyword. (public abstract void m1();).
⇨ Abstract method has no body(m1() {}) only use semicolon (;).
⇨ **Advantages: -**
   - We can achieve **security** using abstraction.
   - **Enhancement** become very easy.
   - **Maintainability** our application is goes to be improved.
   - **Modularity** our application is goes to be improved.

⇨ E.g ATM machine, Car, Mobile, etc.

5. What is Interface?

⇨ **Interface** it is a bunch of rule and guideline.
⇨ An interface refers to a special type of class, which contains methods, but not their definition.
⇨ Only the **declaration** of methods is allowed inside an interface.
⇨ To use an interface, you **cannot create objects**.
⇨ Instead, you need to implement that interface and define the methods for their implementation.
⇨ In interface all **variable** is by-default public, final and static.

- ⇨ In interface all **methods** are ended with **semicolon** (;).
- ⇨ It is by-default **public** and **abstract** only.
- ⇨ e.g public abstract void m2();
- ⇨ We can achieve **multiple inheritance** through interface, it means one interface can extend multiple interfaces.

6. Difference between abstract class and interface

Abstract class	interface
Abstract keyword used to make class abstract.	**Interface** it is a bunch of rule and guideline.
An abstract class **extends single class** only and can implements multiple interfaces.	An interface can **extends** at a time multiple interface.
Abstract class can have abstract and non-abstract **methods.**	**Interface can have** only abstract **methods. Since Java 8, it can have** default and static methods **also.**
Abstract class doesn't support multiple inheritance**.**	**Interface** supports multiple inheritance**.**
Abstract class can have final, non-final, static and non-static variables	**Interface has** only static and final variables
Abstract class can provide the implementation of interface**.**	**Interface** can't provide the implementation of abstract class
The abstract keyword **is used to declare abstract class**	**The** interface keyword **is used to declare interface**
An abstract class **can extend another Java class and implement multiple Java interfaces.**	**An** interface **can extend another Java interface only.**
An abstract class **can be extended using keyword "extends".**	**An** interface **can be implemented using keyword "implements".**
A Java abstract class **can have class members like private, protected, etc.**	**Members of a Java interface are public by default.**
Example: **public abstract class Shape{** **public abstract void draw();** **}**	Example: **public interface Drawable{** **void draw();** **}**

7. What is Encapsulation?

- ⇨ Wrapping/grouping of **data member** and **methods** called as Encapsulation.
- ⇨ We can achieve it by making data members “private”. (private is use for data hiding).
- ⇨ Every java class is best example of encapsulation.
- ⇨ **Encapsulation = Data Hiding + Abstraction.**

- ⇨ POJO class is good example of encapsulation.
- ⇨ In a class if it has every data member as a “private” then such class is called as tightly encapsulated.
- ⇨ Drawback is line of code will increases that why performance will reduce.
- ⇨ E.g. Engine, Gear box within Car etc.

8. Difference between Abstraction and Encapsulation?

Abstraction	Encapsulation
Hiding internal implementation and sharing set of services.	Wrapping of data member and methods.
We can achieve abstraction by using “Interface” and “Abstarct Class”.	We can achieve it by making data members “private”.
It increases code.	It decreases code.
It solves problem at Design level.	It solves problem at implementation level.

9. What is Inheritance?

- ⇨ Parent and Child class relationship is called inheritance.
- ⇨ It is also called as IS-A relationship.
- ⇨ It can be achieved by using “extends” keyword, by making Parent-Child relationship.
- ⇨ It helps in reusability of code.
- ⇨ Parent is a Super Class and Child is a Sub-Class
- ⇨ There are Five types of inheritance
  (1) Single Level (2) Multi Level (3) Multiple (Not Support) (4) Hybrid (Not Support) (5) Hierarchical
- ⇨ E.g New Car version inherits properties from old versions.

10. What is Polymorphism?

- ⇨ It means **one name many forms**.
- ⇨ It makes **reusability** simple and also makes code understanding easy.
- ⇨ There are **two types** of Polymorphism:
  - ○ **Run**-time Polymorphism (**Dynamic** Binding, **Overriding**)
    - ▪ Decision making at Runtime by using runtime object.
    - ▪ Used for adding additional functionality into existing one.
    - ▪ Useful only in Parent-Child Relationship.
  - ○ **Compile**-time Polymorphism (**Static** Binding, **Overloading**)
- ⇨ Best example of polymorphism is “**println**” method of “**printstream**” class.

11. What is Compiler Rule, Method Running Rule, Variable Running Rule?

- ⇨ **Compiler Rule :-** Compiler always check **method and variable** from **reference class** and it will goes to their parent class.

- ⇨ **Method Running Rule :-** Method running always **start from constructor type** and it will goes to their parent class.

- ⇨ **Variable Running Rule :-** Variable running always **start from reference class** and it will goes to their parent class.

12. Difference between Overloading & Overriding?

Overloading	Overriding
It is a complier time polymorphism.	It is run-time polymorphism.
It is static binding.	It is dynamic binding.
In overloading method name is same but different parameter or signature.	In overriding method name parameter must be same.
We can overload method and constructor both.	We can override method but we can't override constructor.
Method overloading is happen in same class but some time it can be happen parent – child class.	Method overriding always happen in parent – child class
Constructor overloading happen in same class.	we cannot override constructor.
We can overload static, private and final method.	We can't override static, private and final method.
Access modifier (public) and return type (void) does not matter.	Access modifier should be same or greater than access modifier.
E.g. "println" method of Printstream class.	E.g 'equals' method of object class and 'equals' method of String class.

13. Difference between IS-A relationship & HAS-A relationship?

IS-A relationship	HAS-A relationship
If we want **total functionality** of **class** then we should go for Is-A relationship.	If we want **part of** the **functionality** then we should go for Has-A relationship.
It is also known as inheritance, it is acquiring parent class properties.	It is acquiring class properties by creating instance of that class.
It is achieved by "**extends**" keyword.	It is achieved by "**new**" keyword.

Method with same name, signature but different return types can't be created in parent-child class.	Method with same name and different return type can be created.

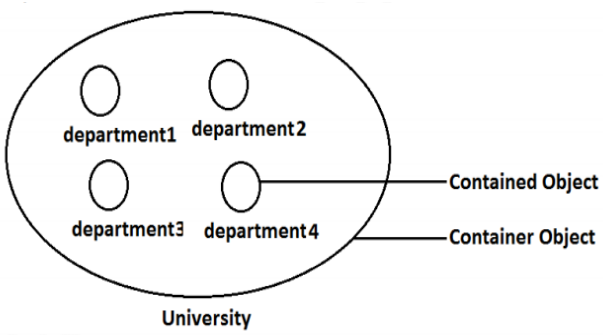
14. Can we write main in abstract class?

- ⇨ **Yes**, you can use the main() method in abstract class.
- ⇨ The main() method is a **static** method so it is **associated with Class**, **not with object or instance.**
- ⇨ The abstract is applicable to the object so there is no problem if it contains the main method.

15. What is Composition and Aggregation?

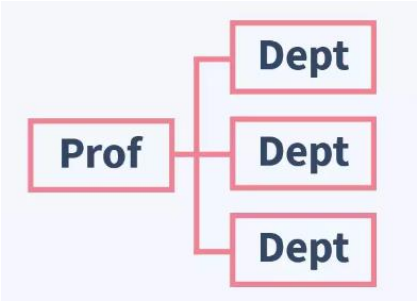
❖ **Composition :-**

- ⇨ Without existing container object if there is no chance of existing contained object then container and contained object are strongly associated and this strongly association is called composition.
- ⇨ **For Example :-**
  - ○ University consist of several department without existing university there is no chance of existing department hence **university on the department are strongly associated** on this association is nothing but composition.



❖ **Aggregation:-**

- ⇨ Without existing container object if there is a chance of existing contained object then contained object then container and contained object are **weakly associated** and this weak association is nothing but aggregation.
- ⇨ **For Example :-**
  - ○ Department consist of several professor without existing department there may be a chance of existing professor object. Hence department and professor object are weakly associated and this weak association is nothing but aggregation.



16. What is Marker Interface?

- ⇨ It is also called "**tag**" interface.
- ⇨ Interface which has **blank body** i.e called "marker" or "tag" interface.
- ⇨ It is used for is **decision making purpose**.
- ⇨ **JVM** will **take decision** through marker interface.
- ⇨ There are several marker interfaces given by Sunmicrosystem.
- ⇨ eg Serializable, Cloneable, Remote, etc.

17. Which OOPs concepts is used in your project?

- ⇨ We use override the method. Secondary reference/extends , service method override service-implementation, setter and getter used.
- ⇨ Interface
- ⇨ Abstraction (Data Hiding)
- ⇨ Polymorphism (Overloading and Overriding Method)
- ⇨ Inheritance
- ⇨ Encapsulation (Data Hiding)

18. Purpose of Marker Interface?

- ⇨ For decision making.

19. Which Tag interface prevent in java?

- ⇨ Serializable, Cloneable

20. Who will provide ability to provide tag interface?

- ⇨ **JVM** will **take decision** through marker interface.

21. How Marker interface internally works?

- ⇨ **JVM** will **take decision** through marker interface

22. What is the access modifier present java?

- ⇨ **There are 4 access modifiers present in java.**
- ⇨ **public :-** It can access anywhere.
- ⇨ **private :-** It can access within a class only.
- ⇨ **default :-** It can access within a package only
- ⇨ **protected :-** It can access within a package and outside the package of subclass. It can not applied on the class.

- ⇨ **private < default < protected < public**

23. Explain the Static Keyword?

- ⇨ Static is keyword.
- ⇨ We can **apply static keyword** with variables, methods, blocks and classes if we trying to override, we will get compile time error.
- ⇨ The static **variable** gets memory only once in the class area at the time of class loading. **It can be used to refer to the common properties of all objects**, for example, the company name of employees, college name of students, etc.
- ⇨ A static **method** belongs to the class rather than the object of a class. It can be invoked without the need for creating an instance of a class. It can access static data member and can change the value of it. It can be used to setup database connection.
- ⇨ Static **block** is used to initialize the static data member. It is executed before the main method at the time of class loading.

24. What is non-static block?

- ⇨ Non-static means used in **runtime**.
- ⇨ It is used for non-initializing content.
- ⇨ Before calling constructor, non-static block is executed.

25. Why is the main method static in java?

- ⇨ The main method is always static because static members are those methods that belong to the classes, **not to an individual object**.
- ⇨ So, if the main method will not be static then for **every object it is available**. And that is **not acceptable by JVM**.
- ⇨ JVM **calls** the main method based on the class name **itself**. **Not by creating the object**.
- ⇨ Because there must be **only one main method** in the java program as the **execution starts from the main method**. So, for **this reason** the main method is static.

26. Can the static methods be overridden?

- ⇨ **No**
- ⇨ Declaration of static methods having the **same signature** can be done in the **subclass** but **run time polymorphism cannot take place in such cases**.
- ⇨ Overriding or dynamic polymorphism occurs during the runtime, but the **static methods** are loaded and looked up at the **compile time statically**. Hence, these methods cannot be overridden.

27. Difference between static methods, static variables, and static classes in java.

- ⇨ **Static Methods and Static variables** are those methods and variables that belong

to the class of the java program, not to the object of the class. This gets memory where the class is loaded. And these can directly be called with the help of class names.

- ⇨ **Static classes** - A class in the java program cannot be static except if it is the inner class. If it is an inner static class, then it exactly works like other static members of the class

28. Explain the Final?

- ⇨ Final is **keyword**.
- ⇨ We can apply final keyword with variable, method and class.

❖ **final variable:**

- ⇨ When a variable is declared as final in Java, the **value can't be modified** once it has been assigned.
- ⇨ If **any value has not been assigned** to that variable, then **it can be assigned only by the constructor of the class.**

❖ **final method:**

- ⇨ A method declared as **final cannot be overridden** by its children's classes.
- ⇨ A **constructor cannot be marked as final** because whenever a **class is inherited**, the constructors are not inherited.
- ⇨ Hence, marking it final doesn't make sense. Java throws compilation error saying - **modifier final not allowed here**

❖ **final class:**

- ⇨ **No classes can be inherited** from the class declared as final.
- ⇨ But that final class can extend other classes for its usage.

29. Is it possible that the 'finally' block will not be executed? If yes then list the case.

- ⇨ **Yes**. It is possible that the 'finally' block will **not be executed**.
- ⇨ The cases are-
  - Suppose we use **System.exit()** in the above statement.
  - If there are fatal errors like **Stack overflow, Memory access error**, etc.

30. Do final, finally and finalize keywords have the same function?

- ⇨ All three keywords have their own utility while programming.

❖ **Final:**

- It is the **keyword**, If any **restriction is required** for classes, variables, or methods, the final keyword comes in handy.
- Inheritance of a final class and overriding of a final method is restricted by the use of the final keyword.
- The variable value becomes fixed after incorporating the final keyword.
- **Example:- final int a=100;**
  a = 0; // error
- The second statement will throw an error.

❖ **Finally:**

- It is the **block** present in a program where all the codes written inside it get executed irrespective of handling of exceptions.
- **try() {} catch(exception e) {} finally(){}**

❖ **Finalize:**

- Prior to the garbage collection of an object, the finalize **method** is called so that the **clean-up activity is implemented**.
    - **public static void main**(String[] args) {
    - String example = **new** String ("Finalize Method");
    - example = **null**;
    - ***System.gc();*** // Garbage collector called
    - }
    - **public void finalize**() {// Finalize called}

31. Define Copy constructor in java.

⇨ Copy Constructor is the **constructor used when we want to initialize the value to the new object from the old object of the same class**.

⇨ Here we are **initializing the new object value from the old object value** in the constructor. Although, this can also be achieved with the help of object **cloning.**

32. Explain constructor and their rule?

⇨ Its use to initialization of instance variable of class.

⇨ Constructor get call automatically whenever new object of class will get create.

⇨ Constructors are **two types** Parameterized and non-parameterized.

⇨ **Rule**: -

- Constructor name must same as a class name.
- Constructor must have no explicit return type.
- If java class have no constructor is define then complier put default constructor.
- If our java source file has any types of constructors is present then complier will not put another constructor.

33. Types of constructor

⇨ Types of constructors depend upon languages

⇨ Private Constructor

⇨ Default Constructor

⇨ Copy Constructor

⇨ Static Constructor

⇨ Parameterized Constructor

34. Can the main method be Overloaded?

- ⇨ **Yes**, It is possible to overload the main method.
- ⇨ We can create as many overloaded main methods we want.
- ⇨ However, **JVM has a predefined calling method** that JVM will **only call the main method** with the definition of -

```
class Main
 {
   public static void main(String args[]) {
      System.out.println(" Main Method");
   }
   public static void main(int[] args){
      System.out.println("Overloaded Integer array Main Method");
   }
   public static void main(char[] args){
      System.out.println("Overloaded Character array Main Method");
   }
   public static int main(double[] args){
      System.out.println("Overloaded Double array Main Method");
   }
   public static void main(float args){
      System.out.println("Overloaded float Main Method");
   }
}
```

35. Explain Object class method?

Method	Description
public final Class **getClass**()	returns the Class class object of this object. The Class class can further be used to get the metadata of this class
public int **hashCode**()	returns the hashcode number for this object.
public boolean **equals**(Object obj)	compares the given object to this object
protected Object **clone**() throws CloneNotSupportedException	creates and returns the exact copy (clone) of this object
public String **toString**().	returns the string representation of this object.
public final void **notify**().	wakes up single thread, waiting on this object's monitor
public final void **notifyAll**()	wakes up all the threads, waiting on this object's monitor.

public final void **wait**(long timeout)throws InterruptedException	causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void **wait**(long timeout, int nanos)throws InterruptedException	causes the current thread to wait for the specified milliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void **wait**()throws InterruptedException	causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).
protected void **finalize**()throws Throwable	is invoked by the garbage collector before object is being garbage collected.

36. What is Object?

- An object is an instance of a class.
- Objects have states and behaviours. Example: A dog has states - color, name, breed as well as behaviours – eat, bark, smell.
- Ways we can create an object :
  - **By using new Operator** :
    Test t = new Test();
  - **By using new Instance**() :(Reflection Mechanism) Test t=(Test)Class.forName("Test").newInstance();
  - **By using Clone**() : Test t1 = new Test();
    Test t2 = (Test) t1. clone();
    By using Factory methods :
  - Runtime r = Runtime.getRuntime();
    DateFormat df = DateFormat.getInstance();
  - By using Deserialization :
    FileInputStream fis = new FileInputStream("abc.ser");
    ObjectInputStream ois = new ObjectInputStream(fis);Test t = (Test)ois.readObject();

37. Explain super keyword?

- Super Keyword is non-static keyword (used in runtime).
- Super() by default present in first line.
- Multiple super keyword is not allow.
- The super keyword is used to access hidden fields and overridden methods or attributes of the parent class.

- ⇨ By using super keyword, we call parent class constructor, method and veriables.

❖ **Super Keyword in case of Constructor calling**

- ⇨ It is use for calling parent class constructor.
- ⇨ It explicitly writes first line of the child class constructor.
- ⇨ No need to write explicitly super keyword in constructor for calling parent.
- ⇨ When we have to call parent class **parameterized** constructor we have to write **compulsory super()** in child class constructor.
- ⇨ At least one constructor in every class must be have super keyword.

❖ **Super Keyword in case of Method calling**

- ⇨ It is use for calling parent class Method.

❖ **Super Keyword in case of Veriable calling**

- ⇨ It is use for calling parent class variable.

38. Explain this keyword?

- ⇨ Super Keyword is non-static keyword (used in runtime).
- ⇨ We can replace super keyword by using this keyword.

❖ **This Keyword in case of Constructor calling**

- ⇨ It is use for calling same class **another** constructor.
- ⇨ We cannot wite super and this at a time in single constructor for calling constructor.
- ⇨ We can write (n-1) this keyword in class constructor.(n= this and 1= super)

❖ **This Keyword in case of Method calling**

- ⇨ It is use for calling same class **another** Method.

❖ **This Keyword in case of Veriable calling**

- ⇨ It is use for calling same class **global** Veriable.

39. What is the difference between OOP and SOP?

- ⇨ **Object-oriented programming** involves concepts of objects and classes. Everything is considered as an object which has specific properties and behaviors which are represented in a class. Object-oriented programming provides encapsulation and abstraction in the code. Ex: – Java Programming language.
- ⇨ **Structure-oriented programming** involves the concepts of functions and structures. Everything is considered functionality and structures, represented using functions—Ex: – C Programming language
- ⇨ It is use for calling parent class Veriable.

40. What are the limitations of OOPs?

- ⇨ Larger Program size – Programs can become lengthy if written using object-oriented programming concepts compared to procedure-oriented programming.
- ⇨ Slower execution – As the number of lines of code to be executed is more comparatively, the execution time is also more.
- ⇨ Not suitable for all types of Problems.
- ⇨ Testing time is also higher for OOP Solutions.

41. What are the differences between class and objects in Java?

Class	Object
Class is a logical entity	Object is physical entity
Class is a template from which object can be created	Object is an instance of the class
Class is a prototype that has the state and behavior of similar objects	Objects are entities that exist in real life such as mobile, mouse, or intellectual objects such as bank account
Class is declared with class key word like class Classname { }	Object is created via new keyword as Employee emp = new Employee();
During class creation, there is no allocation of memory	During object creation, memory is allocated to the object
There is only one-way class is defined using the class keyword	Object creation can be done many ways such as using new keyword, newInstance() method, clone() and factory method.
Real-life examples of Class can be a •A recipe to prepare food. •Blue prints for an automobile engine.	Real-life examples of Object can be •A food prepared from recipe. •Engine constructed as per blue-prints.

42. What is a ClassLoader?

- ⇨ Java Classloader is the program that belongs to JRE (Java Runtime Environment). The task of ClassLoader is to load the required classes and interfaces to the JVM when required.
- ⇨ Example- To get input from the console, we require the scanner class. And the Scanner class is loaded by the ClassLoader.

43. What part of memory - Stack or Heap - is cleaned in garbage collection process?

- ⇨ Heap.

44. Can class support multiple inheritance?

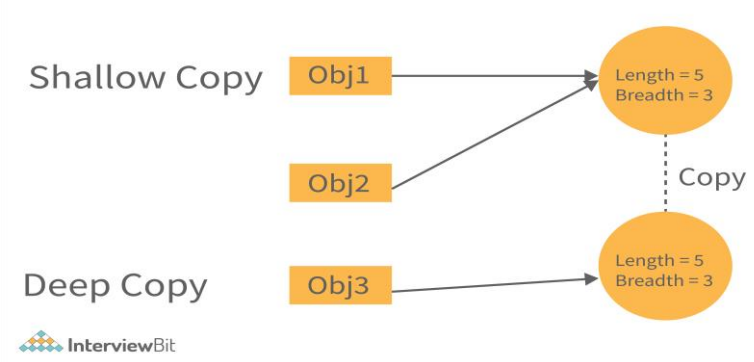
- ⇨ **No**

45. Can interface support multiple inheritance?

- ⇨ **Yes**

46. What are shallow copy and deep copy in java?

- ⇨ To copy the object's data, we have several methods like deep copy and shallow copy.
- ⇨ Shallow copy - The shallow copy only creates a new reference and points to the same object. Example:-- Rectangle obj2 = obj1;
- ⇨ Deep Copy - In a deep copy, we create a new object and copy the old object value to the new object. Example –
    - **Rectangle obj3 = new Rectangle();**
    - **Obj3.length = obj1.length;**
    - **Obj3.breadth = obj1.breadth;**



47. Why An Interface Cannot Have Constructor In Java?

- ⇨ Interface **cannot** have constructor in java, because interface **not have any instance member** so nothing to construct.
- ⇨ Now the question comes how interface can be inherited **without constructor because subclass constructor call super class constructor**.
- ⇨ We have two cases here.
    - **First case, interface extends other interface:** in this situation there is no issue because as we know that **no interface have constructor and that's why no super class constructor will be called.**
    - **Second case**, **a class implements an interface:** in this situation there is **no inheritance because class implements interface not extends interface.**

48. Create An Object Without Using New Operator In Java

- ⇨ **Yes**, we can create an object without using new operator in java**.**
- ⇨ **Using newInstance() Method**
    - If we know the name of the class and it has a public default constructor than we

can create an object in the following way.

- **MyObject object = (MyObject) Class.forName("Without Using New Ope").newInsta**

49. Can We Make a Constructor Final In Java?

- **No**, we cannot make constructor as final in java.
- For methods, final keyword is used to prevent them to be **overridden** by subclass. Constructors are also the **special kind of methods** but as we know that constructor **cannot be inherited in subclass**, hence there is no use of final keyword with constructor.

50. Private Constructor In Java?

- We can create private constructor in java.
- It is used to **restrict** the **instantiation** of a class.
- We **cannot create an object outside of the class**, if we create the **private** constructor.
- It is used to **implement Singleton pattern**.
- The main purpose of singleton pattern is to **control object creation** i.e. keep only one instance of a class at any time.

51. What is Narrowing Concept?

- Value should be **applicable in multiple overloading method**.
- Where the value is applicable then **method should be call in parent-child relation**.
- According to narrowing rule **child class** parameter **get first priority to execute**.

What is a singleton class in Java?

- I any java class if we are allow to create only one object such type of class is called as **singleton Class**.
- **Advantages**:-
  - If several people have same requirement, then it is **not recommended to create a separate object for every requirement**.
  - We have to create only one object and we can **reuse same object for every similar requirement** so that **performance** and **memory utilization** will be **improve**.

52. How to create singleton class in Java?

- We have to use private constructor and private static variable and public factory method.
- Private constructor main use to create our own singleton classes.

53. Class is not final but we are not allowed to create child class how it is possible?

- By declaring every constructor as private we can restrict child class create.
- Class P {private p(){}}
- Class C extends P {c(){}} impossible to create child class.

54. Java Immutable Class

- ⇨ In Java, when we create an **object** of an immutable class, we **cannot change its value**.
- ⇨ For example, **String** is an immutable class. Hence, we **cannot change the content** of a **string once created**.
- ⇨ Besides, we can also **create** our **own custom immutable classes**. Here's what we need to do to create an immutable class.
- ⇨ declare the **class as final** so it cannot be extended
- ⇨ **all class members** should be **private** so they cannot be accessed outside of class
- ⇨ **shouldn't** contain any **setter** methods to change the value of class members
- ⇨ the **getter** method should return the **copy of class members.**
- ⇨ class members are only initialized using constructor

```
// class is declared final
final class Immutable {
  // private class members
  private String name;
  private int date;

  Immutable(String name, int date) {
    // class members are initialized using constructor
    this.name = name;
    this.date = date;
  }
  // getter method returns the copy of class members
  public String getName() {  return name; }
  public int getDate() { return date; }
}
class Main {
  public static void main(String[] args) {
    // create object of Immutable
    Immutable obj = new Immutable("Programiz", 2011);
    System.out.println("Name: " + obj.getName());
    System.out.println("Date: " + obj.getDate());
  }
}
```

55. Static related program?

- ⇨ **A static method can be accessed without creating an object of the class first:**

```
public class Main {
  // Static method
  static void myStaticMethod() {
    System.out.println("Static methods can be called without creating objects");
```

```
  }
  // Public method
  public void myPublicMethod() {
    System.out.println("Public methods must be called by creating objects");
  }
  // Main method
  public static void main(String[ ] args)
 {
    myStaticMethod(); // Call the static method
    // myPublicMethod(); This would output an error

    Main myObj = new Main(); // Create an object of Main
    myObj.myPublicMethod(); // Call the public method
  }
}
```

56. Final related program?

⇨ Set a variable to final, to prevent it from being overridden/modified:

```
public class Main {
  final int x = 10;

  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 25; // will generate an error: cannot assign a value to a final
variable
    System.out.println(myObj.x);
  }
}
```

Exception Handling

1. **What is Exception?**
   - ⇨ In our application, there are chances to **occur abnormal condition**, because of that abnormal condition **normal flow** of program gets **disturbed** or sometime program will be **terminated abnormally**, such situation is called as an exception
   - ⇨ All types of exceptions only occur at **runtime**
   - ⇨ So, avoiding such abnormal termination we need to handle exception.
   - ⇨ Exception will occur because of **wrong user input, file mismatch, logical mistake**.
   - ⇨ There are two types of exception
     - Check Exception (Complier Time Exception)
     - Uncheck Exception (Runtime Exception)

2. **What is Exception Handling?**
   a. To avoid occurring exception we need to provide alternative way to execute rest of program, this is called as exception handling.
   b. Keywords to handle exception: try, catch, finally, throw, throws.
   - ⇨ **try** : The code or set of statements that may raise exception should be try block.
   - ⇨ **catch** : This block catches the **exceptions thrown** in the try block.
   - ⇨ **finally** : This block of code is **always executed** whether an exception has occurred in the try block or not except in one scenario explained in below question.
   - ⇨ **throw** : Its used to throw exception at runtime. Its used in custom exception.
   - ⇨ **throws** : Its used to declare that could be throw while execution program.

3. **What is Error?**
- ⇨ The error is occurred at runtime.
- ⇨ This type of error we need to solve not handle such type of exception is called as error.
- ⇨ Error are Three types
  I. Virtual Machine Error (StackOverFlowError, OutOfMemoryError)
  II. Assertion Error
- ⇨ Ex. Stack Overflow error, virtual machine out of memory.

4. **Default Exception handling in Java?**
   a. Step-1: Problem Analyzed.
   b. Step-2: Problem Finding.
   c. Step-3: Create Object of an Exception found.
      Eg ArithmeticException e=new ArithmeticException ();
   d. Step-4: Throw e;
   e. Step-5: JVM will catch.
   f. Step-6: JVM will display Exception message.

5. Difference between Exception and Error?

Exception	Error
Classified as checked and unchecked type.	Classified as an unchecked type.
It belongs to java.lang.Exception.	It belongs to java.lang.error.
We need to handle the exceptions.	We need to solve the errors.
It can occur at run-time & compile-time both.	It doesn't occur at compile-time, only occurs at run-time.
E.g. NullPointerException, SqlException, etc.	E.g. OutOfMemoryError , AbstractMethodError, etc.

6. Explain Exception hierarchy?

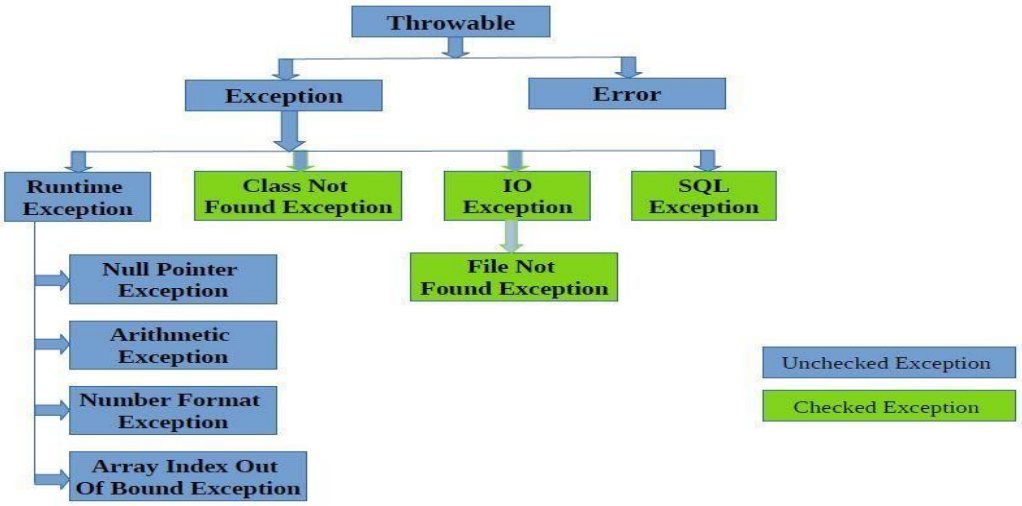
- ⇨ Throwable class access route for exception hierarchy.
- ⇨ Throwable class define two child classes
  - I. Exception
  - II. Error

❖ **Exception:-**

- ⇨ Most of the times exception are caused by our program and these are recoverable.
- ⇨ **Example:-** If our programming requirement is to read data from remote file locating at USA at runtime if remote is not available then we will get runtime exception saying FileNotFoundException.
- ⇨ If FileNotFoundException occurs we can provide location file continue rest of the program normally.

❖ **Error:-**

- ⇨ Most of the times error are caused by our program and these are recoverable.
- ⇨ Errors are non-recoverable.
- ⇨ If OutOfMemoryError occurs being a programmer we cannot do anything and program will be terminated abnormally.
- ⇨ System admin or server admin is responsible to increase heap memory.



7. Difference between Checked Exception & Unchecked Exception?

Checked Exception	Unchecked Exception
Exceptions that **compiler forces** user to write handling code before compilation, it is called checked exception.	Exceptions that **compiler doesn't forces** user to write handling code are called checked exception.
Compulsory need to handle them.	Not compulsion to handle them.
It **increases** code.	It **reduces** code.
Eg. IOException, SqlException, FileNotFoundException, etc.	All Runtime Exception as well as all Errors are example of unchecked exception.

8. What are methods in Exception class?

- ⇨ public String **getMessage**()
  - Returns a detailed message about the exception that has occurred.
- ⇨ public Throwable **getCause**() Returns the cause of the exception.
- ⇨ public String **toString**()
- ⇨ Returns the name of the class concatenated with the result of **getMessage**().
- ⇨ public void **printStackTrace**()
- ⇨ **Prints** the result of **toString**() along with the stack trace, the error output stream.
- ⇨ public **StackTraceElement** [] **getStackTrace**()
  - Returns an array containing each element on the stack trace.
- ⇨ public Throwable **fillInStackTrace**()
  - Fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace.

9. What is use of finally block?

- ⇨ "finally" block code always be executed either there is problem inside "try" block or there is no problem inside "try" block. In both situation, "finally" block code will be executed.
- ⇨ Resources which are open inside "try" block, those resources should be closed inside "finally" block. E.g. If database connection open then it should be closed inside "finally" block.
- ⇨ Syntax :
  - Try(){} catch(){} finally{} is valid
  - Try(){} catch(){} catch(){} finally{} is valid
  - Try(){} finally{} catch(){} not valid
  - Try(){} finally{} finally{} not valid.
- ⇨ We cannot write two or more "finally" block for one "try" block, only one "finally" block can be written with "try" block.
- ⇨ We cannot write "finally" block before "catch" block, we can write it only after "catch" block like try-catch-finally.

10. How to stop executing finally block?

a. Before finally block get executed, System.exit(0); line get executed.
b. Before finally block get executed, unending loop get executed.

11. What is final, finally, finalized?

a. The **final keyword** in java is used to **restrict** the user. The java final keyword can be used in many contexts. Final can be used for variables, methods, class.
    i. Final **variable** once assigned can't be changed after.
    ii. Final **method** can't be rewritten, can't be inherited.
    iii. Final **class** can't be accessed by creating child of it.
b. The **finally** is a **block** that always be executed either there is exception occur inside "try" block or there is no exception occur inside "try" block.
c. In both situations, "finally" block code will be executed.
d. The **finalize**() **method** is called by the garbage collector on an object when garbage collection determines that there are no more references to the object, it is used to perform clean-up activity.

12. What is throws keyword?

- ⇨ By using "throws" keyword we can give a **chance to caller method to handle** the exception.
- ⇨ The "throws" keyword is used for **propagating**(publicity) the exception.
- ⇨ Whenever **unchecked** exception will occur, it will automatically propagate.
- ⇨ Whenever **checked** exception will occur, we need to **write** "**throws**" keyword for propagating the exception.
- ⇨ The "throws" keyword is used for **delegating**(handover) the exception.
- ⇨ Throws key is **followed** by exception class if can be **one or more multiple exception class**.
- ⇨ Throws keyword is always come with **method signature or constructor signature**.
- ⇨ e.g. public void m1( ) throws IO Exception, SQL Exception{ }

13. What is throw keyword?

- ⇨ The "throw" keyword is used for occurring our **own** exception.
- ⇨ Whenever we want occur **custom exception** that time, we will use throws keyword.
- ⇨ Throw keyword should be **inside** the method.
- ⇨ Throw keyword used **runtime** exception.
- ⇨ Throw keyword is forward by exception object.
- ⇨ Syntax :

```
public void m1( )
{
```

```
Arithmetic Exception e=new Arithmetic Exception( ) ;throw e;}
OR
public void m2( )
{
throw new Arithmetic Exception( ) ;
}
```

14. Difference between throw & throws?

throw	throws
It is used to occur custom exception.	It is used to give a chance to caller method to handle the exception.
This keyword is used while creating new instance of exception class.	This keyword is used with method name to declare the exception that need to propagate.
Can be used only with single exception at a time.	Can be used with multiple exceptions at a time.
You cannot throw multiple exceptions.	You can declare multiple exceptions

15. Why to create Customise Exception?

a. Customise Exception tells user user-friendly message, as user can't understand programming language code for exception.

16. Tell me Customise Exception for any Checked Exception?

a. First create user-define class for customise exception, extend it to corresponding checked exception class, and also create parameterized constructor that accepts string input and forward that variable using super keyword.
b. Now use throw keyword and create new instance of custom class you have created where you want to handle checked exception in your application.

17. Can we create Customise Exception for Checked Exception?

a. Yes, we can but not necessary, because customise exception used for telling user user-friendly message as user doesn't know programming language.
b. But programmer is aware of programming codes, so programmer doesn't require customise exception.

18. New Exception related feature in JDK 1.7 version?

a. Try with Resource.

19. Which type of statements can be written in try with resources?

a. We can write those classes/interfaces which have implemented/extended Autoclosable interface only.
b. For user-defined class/interface, we need to implement/extend Autoclosable interface explicitly. So that we can write that class in try with resources context.

20. Difference between ClassCasteException and NoClassDefFound Error?

ClassCasteException	NoClassDefFound Error
It occurs when we are trying to type cast parent object to Child.	It occurs when it is unable to find required .class file at runtime.
It can be avoided by correcting type while type casting, by changing in code.	It can be avoided by providing required .class file.
It is child class of Runtime Exception.	It is child class Error.

21. How do you handle Exception in your project?

⇨ We created separate package for Exception classes needed in project where handling code was written, then at a time of exception handling scenario in project used throws keyword.

```
@RestControllerAdvice
@ResponseStatus
public class ExcepationController
{
    @ExceptionHandler(value = AccountNotFoundException.class)
      public ResponseEntity<Object> exception(AccountNotFoundException
exception)
      {
        return new ResponseEntity<Object>("Account not found",
HttpStatus.NOT_FOUND);
      }
}
```

22. Throws keyword with method overriding, in case of unchecked exception?

⇨ If parent class method throws unchecked exception or does not throws any exception then at the time of overriding no need to write throws keyword.

23. Throws keyword with method overriding, in case of checked exception?

⇨ If parent class method throws checked exception then at that time of overriding no need to write keyword.
⇨ If we want then we can write same exception class or their child class but not parent class.

24. Different Scenarios in Exception Handling?

a. > Once problem will occur inside "try" block, then remaining line of code of "try" block will not be executed.

- If there is no problem inside "try" block, then "catch" block will be skipped. In "catch" block we can write same class of exception and its parent classes only.
- We can write two or more "catch" for one "try" block, but we can write child classes of exception in first catch blocks and then parent class exception, we cannot write first parent exception before child exception.

b. > If "try" with "finally" block and "try" have return statement, then before control return back to caller first of all "finally" block code will be executed.

- If "try" with "finally" block, "try" have return statement and if "finally" block change return statement value, then it will be changed only for "finally" block, it will not be changed for caller.
- If "try" and "finally" both have return statement, then "finally" block statement will be executed.

c. "throws" with method calling, in case of unchecked exception classes :

- For unchecked exceptions, it is not needed to write explicitly throws exception, the exception propagation is done implicitly to the caller.
- In case of checked exception classes, needed to write explicitly throws exception for propagating exception to the caller.

d. "throws" keyword with method Overriding:

- (In case of unchecked exceptions) If Parent class method throws unchecked exception or doesn't throws any exception then at the time of overriding no need to write throws keyword, if we want we can write ay of unchecked exception class but not checked exception class.
- (In case of checked exceptions) If Parent class method throws checked exception then at the time of overriding no need to write throws keyword, if we want then we can write same exception class or their child class but not parent class.

e. "throws" keyword with Super Class constructor and Sub Class constructor:

- (For unchecked exception) If Parent class constructor doesn't throws any exception or "throws" unchecked exception then child class constructor no need to write
  "throws keyword, if we want then any of exception class they can write.
- (For checked exception) If parent class constructor throws exception then at a time of inheritance child class constructor must write "throws" keyword with same exception class or their parent class.

25. Can we write only try block without catch and finally blocks?

- ⇨ **No**,
- ⇨ It shows **compilation** error.
- ⇨ The try block must be followed by either catch or finally block. You can remove either catch block or finally block but not both.

25. There are three statements in a try block – statement1, statement2 and statement3. After that there is a catch block to catch the exceptions occurred in the try block. Assume that exception has occurred in statement2. Does statement3 get executed or not?

- ⇨ **No.**
- ⇨ Once a try block **throws an exception**, remaining statements will **not be executed**.
- ⇨ **control** comes directly to catch block.

26. What is unreachable catch block error?

- ⇨ When you are keeping multiple catch blocks, the order of catch blocks must be from most specific to most general ones.
- ⇨ i.e sub classes of **Exception** must come **first** and **super classes later**.
- ⇨ If you keep **super classes first and sub classes later**, compiler will show unreachable catch block error.
- ⇨ **Example :-**

```
public class ExceptionHandling
{
public static void main(String[] args)
   {
      try {
         int i = Integer.parseInt("abc");
         //This statement throws NumberFormatException
      }
      catch(Exception ex) {
         System.out.println("This block handles all exception types");
      }
       catch(NumberFormatException ex)
      {
         //Compile time error
         //This block becomes unreachable as
         //exception is already caught by above catch block
      }
   }
}
```

27. What is Re-throwing an exception in Java?

- Exceptions raised in the try block are handled in the catch block. If it is unable to handle that exception, it can re-throw that exception using throw keyword. It is called re-throwing an exception.

```
try
{
   String s = null;
   System.out.println(s.length());   //This statement throws
NullPointerException
}
catch(NullPointerException ex)
{
   System.out.println("NullPointerException is caught here");

   throw ex;   //Re-throwing NullPointerException
}
```

28. What is ClassCastException in Java?

- ClassCastException is a **RunTimeException**.
- which occurs when **JVM unable to cast an object of one type to another type**.

29. Can we override a super class method which is throwing an unchecked exception with checked exception in the sub class?

- No.
- If a **super** class method is throwing an **unchecked** exception, then it can be **overridden** in the **sub** class with **same exception** or any other unchecked exceptions but cannot be overridden with checked exceptions.

30. Do you know try-with-resources blocks? Why do we use them? When they are introduced?

- Try-with-resources blocks are introduced from Java 7 to auto-close the resources like File I/O streams, Database connection, network connection etc… used in the try block.
- You need not to close the resources explicitly in your code.
- Try-with-resources implicitly closes all the resources used in the try block.

31. What are the benefits of try-with-resources?

- The main benefit of try-with-resources is that it avoids resource leaks that could happen if we don't close the resources properly after they are used.
- Another benefit of try-with-resources is that it removes redundant statements in the code and thus improves the readability of the code.

32. What is a SQLException in Exception Handling?

- ⇨ An exception that provides information **related** to **database** access **error** or other errors is called SQL exception.

33. What is NumberFormatException in java?

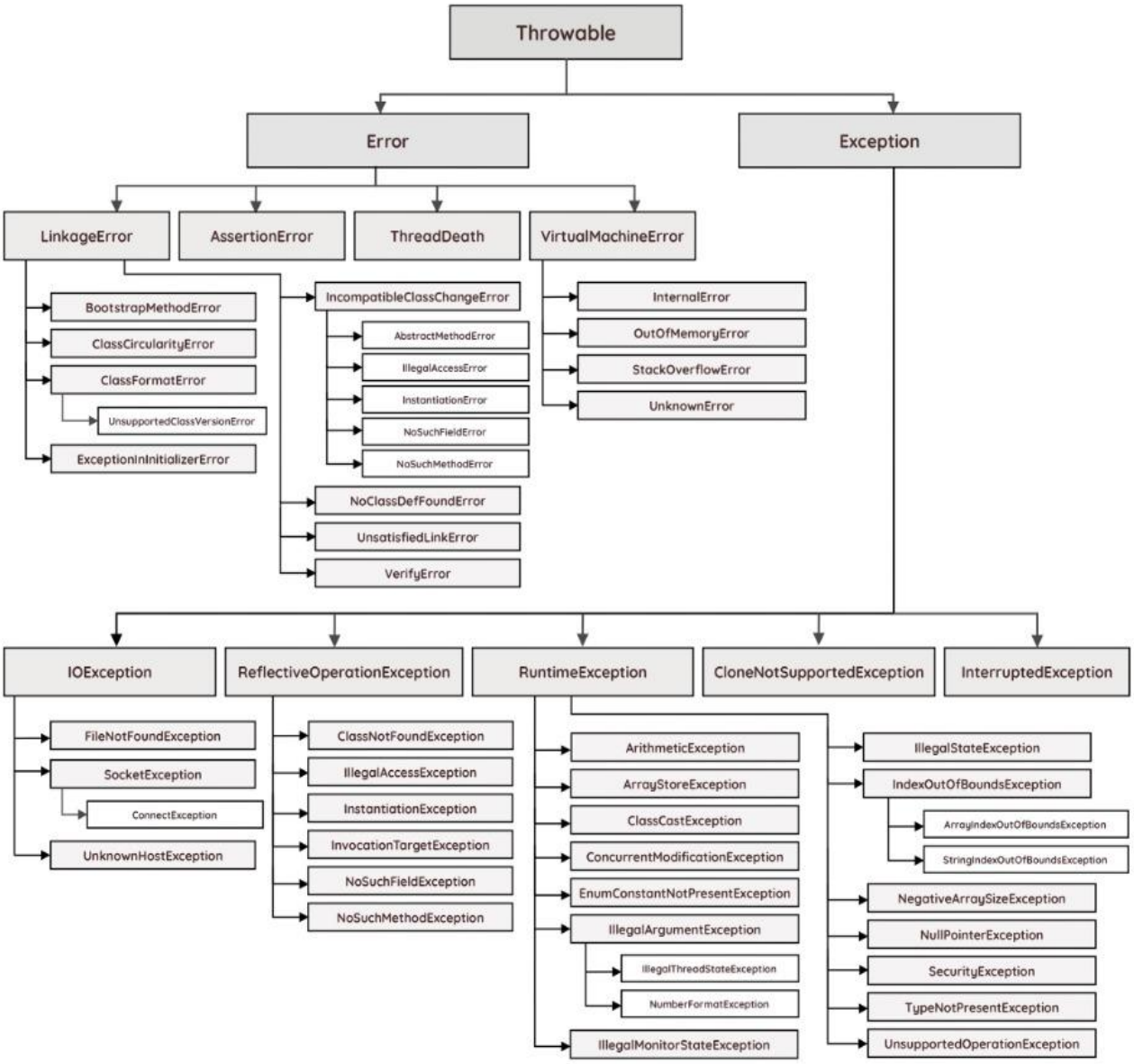
- ⇨ NumberFormatException is thrown when you **try to convert a String into a** number.

34. What is ArrayIndexOutOfBoundsException in java?

- ⇨ ArrayIndexOutOfBoundsException arises while **trying to access an index** of the **array that does not exist** or out of the bound of this array

35. What will happen if an exception is thrown by the main method?

- ⇨ When an exception is thrown by the main method then JVM **terminates** the **program**.
- ⇨ As a result, you will find the exception message and stack trace in the system console.


Throwable
Error
Exception
LinkageError
AssertionError
ThreadDeath
VirtualMachineError
BootstrapMethodError
ClassCircularityError
ClassFormatError
UnsupportedClassVersionError
ExceptionInInitializerError
IncompatibleClassChangeError
AbstractMethodError
IllegalAccessError
InstantiationError
NoSuchFieldError
NoSuchMethodError
NoClassDefFoundError
UnsatisfiedLinkError
VerifyError
InternalError
OutOfMemoryError
StackOverflowError
UnknownError
IOException
ReflectiveOperationException
RuntimeException
CloneNotSupportedException
InterruptedException
FileNotFoundException
SocketException
ConnectException
UnknownHostException
ClassNotFoundException
IllegalAccessException
InstantiationException
InvocationTargetException
NoSuchFieldException
NoSuchMethodException
ArithmeticException
ArrayStoreException
ClassCastException
ConcurrentModificationException
EnumConstantNotPresentException
IllegalArgumentException
IllegalThreadStateException
NumberFormatException
IllegalMonitorStateException
IllegalStateException
IndexOutOfBoundsException
ArrayIndexOutOfBoundsException
StringIndexOutOfBoundsException
NegativeArraySizeException
NullPointerException
SecurityException
TypeNotPresentException
UnsupportedOperationException

Array

1) What is ArrayStoreException in java? When you will get this exception?

- ⇨ The ArrayStoreException is a run time exception which occurs when you try to store non-compatible element in an array object.
- ⇨ The type of the elements must be compatible with the type of array object.
- ⇨ For example, you can store only string element in an array of string. If you try to insert integer element in an array of string, you will get ArrayStoreException at run time.

⇨ Example:

```
public class MainClass
  {
        public static void main(String[] args)
         {
   Object[] stringArray = new String[5];
         stringArray[1] = "JAVA";
        stringArray[2] = 100;
        }
        }
```

2) Can you pass the negative number as an array size?

- ⇨ No. You can't pass the negative integer as an array size.
- ⇨ If you pass, there will be no compile time error but you will get NegativeArraySizeException at run time.

3) Can you change the size of the array once you define it? OR Can you insert or delete the elements after creating an array?

- ⇨ No. You can't change the size of the array once you define it.
- ⇨ You can not insert or delete the elements after creating an array.
- ⇨ Only you can do is change the value of the elements.

4) What is an anonymous array? Give example?

- ⇨ Anonymous array is an array without reference. For example,

```
Public class A{
    Public static void main(String[] args)
 {
  System.out.println(new int[]{1, 2, 3, 4, 5}.length);    //Output : 5
 System.out.println(new int[]{21, 14, 65, 24, 21}[1]);   //Output : 14
} }
```

5) There are two array objects of int type. one is containing 100 elements and another one is containing 10 elements. Can you assign array of 100 elements to an array of 10 elements?

⇨ Yes, you can assign array of 100 elements to an array of 10 elements provided they should **be of same type**. While assigning, compiler checks only type of the array not the size.

6) What are the differences between Array and ArrayList in java?

Basis	Array	ArrayList
Definition	An array is a dynamically-created object. It serves as a container that holds the constant number of values of the same type. It has a contiguous memory location.	The ArrayList is a class of Java Collections framework. It contains popular classes like Vector, HashTable, and HashMap.
Static/ Dynamic	Array is static in size.	ArrayList is dynamic in size.
Resizable	An array is a fixed-length data structure.	ArrayList is a variable-length data structure. It can be resized itself when needed.
Initialization	It is mandatory to provide the size of an array while initializing it directly or indirectly.	We can create an instance of ArrayList without specifying its size. Java creates ArrayList of default size.
Performance	It performs fast in comparison to ArrayList because of fixed size.	ArrayList is internally backed by the array in Java. The resize operation in ArrayList slows down the performance.
Primitive/ Generic type	An array can store both objects and primitives type.	We cannot store primitive type in ArrayList. It automatically converts primitive type to object.
Iterating Values	We use for loop or for each loop to iterate over an array.	We use an iterator to iterate over ArrayList.
Type-Safety	We cannot use generics along with array because it is not a convertible type of array.	ArrayList allows us to store only generic/ type, that's why it is type-safe.
Length	Array provides a length variable which denotes the length of an array.	ArrayList provides the size() method to determine the size of ArrayList.
Adding Elements	We can add elements in an array by using the assignment operator.	Java provides the add() method to add elements in the ArrayList.

Single/ Multi-Dimensional	Array can be multi-dimensional.	ArrayList is always single-dimensional.

7) What are the different ways of copying an array into another array?

- ⇨ There are four methods available in java to copy an array.
- ⇨ Using for loop
- ⇨ Using Arrays.copyOf() method
- ⇨ Using System.arraycopy() method
- ⇨ Using clone () method

8) What is ArrayIndexOutOfBoundsException in java? When it occurs?

⇨ ArrayIndexOutOfBoundsException is a **run time** exception which occurs when your program **tries to access invalid index of an array** i.e. negative index or index higher than the size of the array.

```
public class A {
    public static void main(String[] args)
    {
        int[] x = new int[2];
        x[0]=10;
        x[1]=20;
        x[2]=30;

        System.out.println(x[0]);
        System.out.println(x[1]);
        System.out.println(x[2]);
    } }
}
```

Collection

1) What is Collection?

- ⇨ Collection is an **Interface**.
- ⇨ Collection Means **group of object**.
- ⇨ Collection represents a **single unit of objects as a group**.
- ⇨ Collection provides operations that you perform on a data such as **searching**, **sorting**, **insertion** and **deletion** on the group of objects.
- ⇨ Collection Size is **Growable** in nature.
- ⇨ Collection it can hold both **Homogeneous and Heterogeneous** Element.

- ⇨ The Collection is a framework that provides an **architecture to store and manipulate** the group of objects.
- ⇨ Collection Framework is a **grouping of classes and interfaces** that is used to store and manage the objects.
- ⇨ It provides various **classes** like **Vector, ArrayList, HashSet, Stack**, etc. Java Collection framework can also be used for **interfaces** like **Queue, Set, List**, etc.

2) Which Collection object you have used in your Project?

- ⇨ List - ArrayList for getting lists of object from Database.
- ⇨ Set – HashSet for mapping POJOs using hibernate. (One-to-Many & Many-to-One)

3) Explain List interface?

- ⇨ List interface **extends** collection interface used to **store sequence of elements** in collection.
- ⇨ We can even **store duplicate** elements in list.
- ⇨ It is a factory of **ListIterator** interface.
- ⇨ Through the ListIterator, we can iterate the list in **forward** and **backward** directions**.**
- ⇨ The **implementation classes** of List interface are **ArrayList**, **LinkedList**, Stack and **Vector**.
- ⇨ It contains the **index-based methods** to insert, update, delete and search the elements.
- ⇨ We can also **store the null** elements in the list.
- ⇨ List is an **ordered** collection.
- ⇨ The main difference between List and non list interface are **methods based on position.**

4) Explain ArrayList ?

- ⇨ ArrayList is a **class** of **implements** List Interface.
- ⇨ ArrayList is an **ordered collection** which extends **AbstractList**.
- ⇨ We use ArrayList mainly when we need **faster access and fast iteration** (FAFI) of elements in list.
- ⇨ We can insert **nulls** in to arraylist.
- ⇨ Arraylist is nothing but a **growable array**.
- ⇨ public class ArrayList extends **AbstractList** implements List, **RandomAccess**, **Cloneable**, java.io.**Serializable**{} From java 1.4 ArrayList implements.
- ⇨ RandomAccess interface which is a marker interface which supports fast and random access.
- ⇨ **Advantages** : 1) Faster and easier access. 2) Used for Random access of elements.
- ⇨ **Drawbacks** : 1) We cannot insert or delete elements from middle of list

5) How Arraylist works?

- ⇨ When we **create object** of ArrayList, it create ArrayList instance with **default capacity 10**.
- ⇨ ArrayList capacity **increases** with formula :–
  New Capacity = ((3/2) x Old Capacity)+1
- ⇨ When ArrayList increments with new capacity then data from old ArrayList is **copied** into new instance and old instance is destroyed.
- ⇨ When we **add** or **delete** data into the ArrayList then **multiple data shift** operations are performed.
- ⇨ ArrayList follows **Indexing**.

6) List down the primary interfaces provided by Java Collections Framework?

1. public interface **List** <E>extends **Collection**<E>

- ⇨ *List Interface*: java.util.List is an extended form of an array that contains ordered elements and may include duplicates.
- ⇨ It supports the index-based search, but elements can be easily inserted irrespective of the position.
- ⇨ The List interface is implemented by various classes such as ArrayList, LinkedList, Vector, etc.

2. public interface **Set**<E> extends **Collection**<E>

- ⇨ *Set Interface*: java.util.Set refers to a collection class that cannot contain duplicate elements.

- ⇨ Since it **doesn't** define an order for the elements, the index-based search is **not** supported.
- ⇨ It is majorly used as a **mathematical** set abstraction model.
- ⇨ The Set interface is **implemented** by various classes such as HashSet, TreeSetand LinkedHashSet.

3. public interface **Queue** <E> extends **Collection**<E>

- ⇨ *Queue Interface*: java.util.Queue in Java follows a FIFO approach i.e. it orders the elements in First In First Out manner.
- ⇨ Elements in Queue will be added from the rear end while removed from the front.

4. public interface **Map**<E> extends **Collection**<E>

- ⇨ *Map Interface*: java.util.Map is a **two-dimensional** data structure in Java that is used to store the data in the form of a Key-Value pair.
- ⇨ The key here is the **unique** hashcode and value represent the element.
- ⇨ Map in Java is another form of the Java Set but can't contain duplicate elements.

7) What is the need for overriding equals() method in Java?

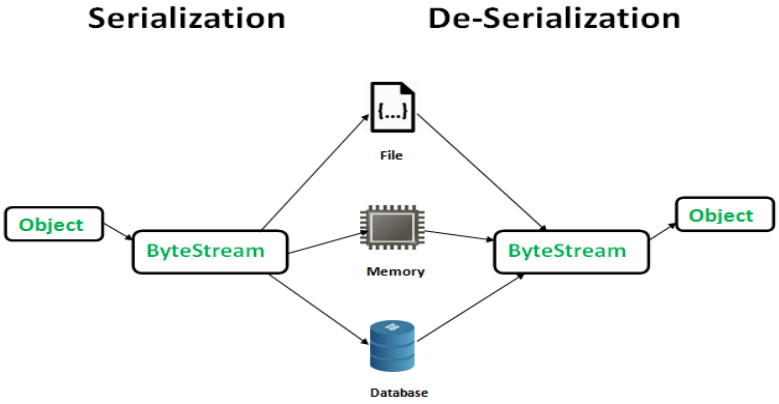
- ⇨ The initial implementation of the equals method helps in checking whether **two objects are the same or not**.
- ⇨ But in case you want to **compare the objects based on the property** you will have to override this method.

8) Why Collection doesn't extend the Cloneable and Serializable interfaces?

- ⇨ The Collection interface in Java specifies a **group of objects called elements**.
- ⇨ Each **concrete implementation** of a Collection can choose its **own way** of how to **maintain** and **order** its elements.
- ⇨ Thus, there is no use of extending the Cloneable and Serializable interfaces.
- ⇨ For **example**, the **List** implementations allow **duplicate** elements but **Set** implementations don't.
- ⇨ Many implementations have a method for public cloning. But it isn't practical to include it in all Collection implementations as the Collection is **abstract** and **implementation** is all that matters.

9) What is Serialization In Java?

- ⇨ Serialization can be defined as a process by which we **convert the object state into its equivalent byte stream to store the object into the memory** in a **file or persist** the object.
- ⇨ When we want to **retrieve** the object from its saved state and **access** its contents, we will have to **convert the byte stream back to the actual Java object** and this process is called **deserialization**.



10) How we synchronize Arraylist?

- ⇨ By default ArrayList Object is **Non-Synchronized** but we can get **Synchronized Version ArrayList Object** by using the following Method of Collections Class.
- ⇨ **For** Example.

```
public class A
{
  Public static void main(String[] args)
  {
   List<String> list = Collections.synchronizedList(new ArrayList<String>());
                                    list.add("Pune");
                                list.add("KarveNagar");

                synchronized(list) { Iterator itr = list.iterator();
                        while (itr.hasNext()) { System.out.println(itr.next());}
                                        }
                }}
```

11) Explain all Constructors of Arraylist?

1) **public ArrayList(int initialCapacity):**
   - ⇨ It creates an **Empty ArrayList Object** with specified Initial Capacity.
   - ⇨ When the constructor is used, we can **provide some initial capacity** rather than depending on the default capacity as defined in the ArrayList class.
   - ⇨ For example:-> **List<String> myList = new ArrayList<String>(7);**

```
public ArrayList(int initialCapacity)
 {
  if (initialCapacity > 0) { this.elementData = new Object[initialCapacity]; }
else if (initialCapacity == 0) {this.elementData = EMPTY_ELEMENTDATA; }
```

```
        else
{ throw new IllegalArgumentException("Illegal Capacity: " + initialCapacity); }
        }
```

Where **EMPTY_ELEMENTDATA** is defined as:

```
private static final Object[
    ] EMPTY_ELEMENTDATA = {};
```

⇨ It's clear that if the supplied **capacity** is **more than zero**, the elementData array will be formed with that capacity, but if the provided **capacity is zero**, the elementData array will be constructed with an empty Object array. When the first element is inserted, the ArrayList will expand.

2). public ArrayList(): It creates an Empty ArrayList Object with Default Initial Capacity 10.

The default constructor of ArrayList class is used to create an ArrayList as following,
List myList = new ArrayList();

```
public ArrayList()  {this.elementData = DEFAULTCAPACITY_EMPTY_ELEMENTDATA;
}
```

Where **DEFAULTCAPACITY_EMPTY_ELEMENTDATA** is defined as

```
private static final Object[] DEFAULTCAPACITY_EMPTY_ELEMENTDATA
    = {};
```

As mentioned above initially it will be initialized with an empty array, it will grow only when first element is added to the list.

3). public ArrayList(Collection<? extends E> c):

⇨ It creates an **equivalent** ArrayList Object for the given Collection Object.
⇨ If we want to construct a list containing the elements of the specified collection we can use this constructor.
⇨ In this constructor implementation checks for the length of the collection passed as parameter, if length is greater than zero then Arrays.copyOf method is used to copy the collection to the elementData array.

```
elementData = Arrays.copyOf(elementData, size, Object[].class);
```

12) Why Arraylist is fast for retrieval operation?

⇨ It implements **Random Access Interface**, hence Arraylist is fast for retrieval operation.

13) ArrayList Vs LinkedList

- ArrayList and LinkedList both implements List interface and their methods and results are almost identical.
- However there are **few differences between** them which make **one better over another depending on the requirement**.

Search:-

- ⇨ **ArrayList** search operation is pretty **fast** compared to the **LinkedList** search operation.
- ⇨ **get(int index)** in **ArrayList** gives the performance of **Low Time complexity O(1)** while LinkedList performance is **High Time complexity O(n).**
- ⇨ **Reason**: ArrayList maintains index based system for its elements as it uses array data structure implicitly which makes it faster for searching an element in the list.
- ⇨ On the other side LinkedList implements doubly linked list which requires the traversal through all the elements for searching an element.

Deletion:-

- ⇨ LinkedList remove operation gives O(1)) in best case performance while ArrayList gives variable performance: O(n) in worst case

  Conclusion:

  LinkedList element deletion is faster compared to ArrayList.

  Reason:
- ⇨ LinkedList's each element maintains two pointers (addresses) which points to the both neighbor elements in the list.
- ⇨ Hence removal only requires change in the pointer location in the two neighbor nodes (elements) of the node which is going to be removed.
- ⇨ While In ArrayList all the elements need to be shifted to fill out the space created by removed element.

Inserts Performance:-

- ⇨ LinkedList add method gives O(1) performance while ArrayList gives O(n) in worst case. Reason is same as explained for remove.

Memory Overhead:-

- ⇨ ArrayList maintains indexes and element data while LinkedList maintains element data and two pointers for neighbor nodes hence the memory consumption is high in LinkedList comparatively.

14) How add () of set works?

- ⇨ Its return type is **Boolean**, it returns true or false value.
- ⇨ It uses equals() method of **object class**.
- ⇨ It implements HashMap internally.

15) How LinkedList works? (Why insertion & deletion is fast in LinkedList?)

- ⇨ When we create an **object** of LinkedList and **add** an element to it.
- ⇨ It **stores element** as a **node** in which previous & next node address is also **stored**.
- ⇨ **Node format =**

prev. node addr.	(value)	next node addr.

- ⇨ Due to previous & next node address is stored, hence while **updation** or **insertion & deletion** operation data **shift** operation **need not to perform** and it makes LinkedList fast.

16) Define LinkedList?

- ⇨ The underlying data structure is **Double LinkedList**.
- ⇨ It is one of implemented **class** of List interface in collection framework.
- ⇨ It **allows** duplicate **values**.
- ⇨ Heterogeneous object are allowed.
- ⇨ Insertion order is **preserved** & **indexing** is maintained.
- ⇨ It implements **Clonable**, **Serializable** interfaces but not recommended.
- ⇨ It follows **doubly** linked list structure.
- ⇨ It is mostly preferable for **insertion & deletion** operations.

17) Why set doesn't allow duplicates?

- ⇨ Set **internally** uses **HashMap**.
- ⇨ **HashMap** object is **created** in every **Set implemented class**.
- ⇨ Here HashMap stores **key** as all **inserted** elements and **value** as a **dummy** object created with new keyword.
- ⇨ As HashMap doesn't **accept duplicate keys**, so set don't allow duplicate values.

18) What is Set in Java Collections framework and list down its various implementations?

- ⇨ A Set **refers** to a collection that cannot contain duplicate elements.
- ⇨ It is mainly used to model the **mathematical set abstraction**.
- ⇨ The Java platform provides three general-purpose Set implementations which are:
  1. HashSet 2. TreeSet 3. LinkedHashSet

19) Can you add a null element into a TreeSet or HashSet?

- ⇨ In HashSet, **only one null** element can be added.
- ⇨ In TreeSet it **can't** be added as it makes use of NavigableMap for storing the elements.
- ⇨ This is because the **NavigableMap is a subtype of SortedMap** that doesn't allow null keys.

- So, in case you **try to add null** elements to a **TreeSet**, it will **throw** a **NullPointerException**.

20) Explain the emptySet() method in the Collections framework?

- The Collections.emptySet() is used to **return the empty immutable** Set while **removing the null** elements.
- The set returned by this method is serializable.
- **Syntax: public static final** <T> **Set**<T> emptySet()

21) What is the HashSet class in Java and how does it store elements?

- java.util.HashSet class is a member of the Java collections framework which **inherits** the **AbstractSet** class and **implements** the Set interface.
- It **implicitly** implements a **hashtable** for **creating and storing** a collection of unique elements.
- Hashtable is an instance of the HashMap class that uses a **hashing** mechanism for **storing** the information within a HashSet.
- Hashing is the process of **converting** the informational content into a unique value that is more popularly known as hash code.
- This hashcode is then used for indexing the data associated with the key.
- The entire process of transforming the informational key into the hashcode is performed internally.

22) What is Map?

- Map is used for store different object in the **pair** of “key” and “value”.
- In map, “key” should be **unique**.
- **Insertion** order will **not** be maintained in Map.

23) How HashMap works?

- When we create HashMap object, HashMap instance as per **default capacity 16** buckets is created.
- When we perform add **(put())** operation, it accepts data in key & value format.
- Internally hashing technique is used, that **generates hashcode** for key and also **calculate index** to find bucket location for **inserting** data in HashMap instance.
- It will store element at that location as a **node** format.

prev. node addr.	Key	Value	next node addr

- Now when we perform **retrieval** (get ()) operation, it **asks for key**.
- Again hashing technique is used and bucket location is identified, then equals () method is used to compare key content and if it returns true then value is retrieved.

24) What is Hash Collision?

- ⇨ In HashMap, if **two keys have same hashcodes** then such situation is called as hash collision.
- ⇨ In such case, while adding data, **doubly** LinkedList is created to insert data.

prev. node addr.	Key1	Value1	next node addr	Key2	Value2	null

- ⇨ And **retrieval** operation is **performed** using **equals** () method.

25) What happens when we put same keys in Map?

- ⇨ If we add a key-value pair where the key exists already, put method **replaces the existing value** of the key with the new value.

26) What is Contract between equals () & hashcode ()?

- ⇨ If equals () returns **true**, then **objects** must have **same** hashcodes.
- ⇨ If equals () returns **false**, then objects **may or may not** have same hashcodes.
- ⇨ If hashcodes of **objects** are **same**, then we **can't conclude** output of equals ( ), it may be **true or may be false**.
- ⇨ If hashcodes of **objects** are **different**, then **output** of equals ( ) must be **false**.

27) What is Identity Hashmap?

- ⇨ In IdentityHashMap JVM will Use == Operator to Identify Duplicate Keys, which is meant for **Reference Comparison**.
- ⇨ e.g. Integer i=new Integer (5);
  Integer i1=new Integer (5);
  Map m=new IdentityHashMap();
  m.put(i, "java");
  m.put(i1, "c++");
  System.out.println(m); => {5=java, 5=c++}

28) What is WeakHashmap?

- ⇨ In Case of WeakHashMap if an Object doesn't contain any References, then it is Always Eligible for Garbage Collector Even though it is associated with WeakHashMap.
- ⇨ Garbage Collector Dominates WeakHashMap.
- ⇨ Both null values and null keys are supported in WeakHashMap.
- ⇨ It is not synchronized.

29) What is fail safe & fail fast iterator?

- ⇨ Using iterators we can traverse over the collections objects. The iterators can be either fail-safe or fail-fast.
- ⇨ Fail-safe iterator means they will not throw any exception even if the collection is modified while iterating over it.
- ⇨ Whereas Fail-fast iterator throw an exception (ConcurrentModificationException) if

the collection is modified while iterating over it.

- ⇨ Fail-Fast Iterators internal working:
- ⇨ Every fail fast collection has a modCount field, to represent how many times the collection has changed/modified.
- ⇨ So at every modification of this collection we increment the modCount value. For example the modCount is incremented in below cases:
- ⇨ When one or more elements are removed.
- ⇨ When one or more elements are added.
- ⇨ When the collection is replaced with other collection.
- ⇨ When the collection is sorted.
- ⇨ So everytime there is some change in the collection structure, the mod count is incremented.
- ⇨ Now the iterator stores the modCount value in the initialization as below: int expectedModCount = modCount;
- ⇨ Now while the iteration is going on, expectedModCount will have old value of modCount. If there is any change made in the collection, the modCount will change and then an exception is thrown
- ⇨ Unlike the fail-fast iterators, fail-safe iterators traverse over the clone of the collection. So even if the original collection gets structurally modified, no exception will be thrown.

Collection	Interface	Implements Collection	Allow Duplicate	Allow Null value	Ordered	Synchronized	Thread Safe
ArrayList	List	Yes	Yes	Yes	Yes (Insertion)	No	No
LinkedList	List	Yes	Yes	Yes	Yes	No	No
Vector	List		Yes	Yes	Yes	Yes	Yes
HashSet	Set	Yes	No	Yes	No	No	No
LinkedHashSet	Set			Yes		No	No
TreeSet	Set	Yes	No	No	Yes (Natural)	No	No
SortedSet	Set						
HashMap	Map	No	Just for values	Yes	No	No	No
HashTable				No	No	Yes	Yes
LinkedHashMap	Map			Yes	Yes	No	No
SortedMap	Map						
TreeMap	Map	No	Just for values	No	Yes (natural order of keys)	No	No
Array Deque	Queue	Yes	Yes	No	Yes (FIFO or LIFO)	No	No

Differences – Java Collections Interview Questions

1) Differentiate between an Array and an ArrayList.

Array	ArrayList
An Array is a collection of similar data types stored in contiguous memory locations.	An ArrayList is a class of Java Collections framework.
It is Fixed in Size.	It is Dynamic in Size.
It can hold only **Homogeneous** Data Elements.	It can hold both **Homogeneous** and **Heterogeneous** Elements.
A for **loop or for each loop** is used to **iterate** over an array.	An **iterator** is used to **iterate** over ArrayList.
It is **fast** in comparison to ArrayList as it has a fixed size.	ArrayList is **Slower** than Array.
It is strongly typed	It is loosely types
Array is multi-dimensional.	ArrayList is Single-dimensional
No need to box and unbox the elements	**Needs to box** and unbox the elements

2) Explain Difference between Array & Collection?

Array	Collection
It is Fixed in Size.	It is Growable in nature.
It can hold only **Homogeneous** Data Elements.	It can hold both **Homogeneous** and **Heterogeneous** Elements.
Array can hold both **Primitive** and **Object** type.	Collection can hold only **Object** type.
With Respect **to Memory** Arrays are **Not** Recommended to Use.	With Respect to **Memory** Collections are Recommended to Use.
With Respect to **Performance**, Arrays are Recommended to Use.	With Respect to **Performance**, Collections are **Not** Recommended to Use.
There is **no underlying** data structure for arrays that why readymade method support is not available.	Every collection class is implemented based on some standard data structure hence readymade method support is available.
It is multi-dimensional array.	It is always Single-dimension

3) Difference between Arraylist & Vector?

Arraylist	Vector(1.0V)
It is not Synchronized.	It is Synchronized.
It is not Thread Safe	It is Thread Safe
Its Default Capacity :- 10 and It increases after by (Old+Old*(3/2))+1	Its Default Capacity :- 10 and It increases after by Old+Old*100%
Performance is high.	Performance is low.
Enumeration cannot be used.	Enumeration can be used.

Enumeration is use for to iterate the value (1.0V).

4) Differentiate between ArrayList and LinkedList.

ArrayList	LinkedList
Seaching Operation is **fast** compare to LinkedList	**Seaching** Operation is **slow** compare to ArrayList
Get() performance is **fast** compare to LinkedList because of ArrayList maintain Index base system.	**Get()** performance is **Slow** compare to ArrayList because of LinkedList implements **Doubly** LinkedList.
Deleting is **Slow** compare to LinkedList.	**Deleting** is **fast** compare to ArrayList.
ArrayList need to **shifted** to fill the space created by remove operation.	LinkedList pointer location in two neighbor **node** of the node which is going to be remove.
Add() is **high time complexity** for insertion.	Add() is **low time complexity** for insertion.
Memory **consumption** is **low** in ArrayList because of indexing.	Memory **consumption** is **High** in LinkedList because of maintain data and two pointers for neighbor node.
Implements **dynamic** array internally to store elements	Implements **doubly** linked list internally to store elements
Manipulation of elements is **slower**	Manipulation of elements is **faster**
ArrayList act only as a **List**	LinkedList act as a **List** and a **Queue**

5) Differentiate between List and Set.

List	Set
An **ordered** collection of elements	An **unordered** collection of elements
Preserves the insertion order	**Doesn't preserves** the insertion order
Duplicate values are allowed	Duplicate values are **not allowed**
Any number of **null** values can be **stored**	**Only one null** values can be stored
ListIterator can be used to traverse the List in **any direction**	ListIterator **cannot be used** to traverse a Set
Contains a **legacy** class called vector	**Doesn't** contains any legacy class

6) Differentiate between List and Map.

List	Map
Belongs to java.**util** package	Belongs to java.**util** package
Extends the Collection interface	**Doesn't** extend the Collection interface
Duplicate elements are allowed	Duplicate keys are **not** allowed but duplicate values are
Multiple null values can be stored	Only one null key can be stored but **multiple** null values are allowed
Preserves the **insertion** order	**Doesn't** maintain any insertion order
Stores elements based on **Array Data Structure**	Stores data in **key-value pairs** using various hashing techniques

7) What is Difference between Collection & Collections?

Collection	Collections
It is the root **interface** of the Collection framework	It is a utility **class**
It can be used to Represent a Group of **Individual** Objects as a Single Entity.	It is used to **sort** and **synchronize** the collection elements.
It provides the **methods** that can be used for data **structure**.	It provides the **methods** which can be used for **various operations** on a **collection**.
Is used to represent a group of objects as a single entity	It is used to define **various utility** method for collection objects

It is used to **derive** the data structures of the Collection framework	It contains various **static** methods which help in data structure manipulation

8) Difference between Comparable & Comparator?

Comparable	Comparator
This interface is from java.**lang** package.	This interface is from java.**util** package.
It is used for **Default** sorting.	It is used for **Custom** sorting.
It has only one method i.e **compareTo**.	It has two methods i.e **compare** & **equals**.
Programmer decides how sorting is to be done.	**User** decides how sorting is to be done.
Modifies the actual class	Doesn't modifies the actual class

9) Difference between Hashmap & Synchronised (or Concurrent) Hashmap?

HashMap	Synchronised or Concurrent HashMap
It is non-Synchronized in nature.	It is Synchronized in nature.
It is not Thread-safe.	It is thread-safe.
Performance is high.	Performance is low.
It can throw ConcurrentModificationException.	It doesn't throw ConcurrentModificationException

10) Difference between Hashmap & Hashtable?

HashMap	Hashtable
It is **not** Synchronised.	It is Synchronised.
It allows **multiple** threads at a time.	It allows **single** thread at a time.
It is **not** thread safe.	It is thread safe.
Null key (once) & Null value is **allowed**.	**Null** key & Null value is **not** allowed.
Its performance is **fast**.	Its performance is **slow**.
Inherits **AbstractMap** class	Inherits **Dictionary** class
Can be traversed by **Iterator**	Can be traversed by **Iterator** and **Enumeration**

11) Difference between HashSet, LinkedHashSet, and TreeSet.

	HashSet	LinkedHashSet	TreeSet
How they work internally?	HashSet uses HashMap internally to store it's elements.	LinkedHashSet uses LinkedHashMap internally to store it's elements.	TreeSet uses TreeMap internally to store it's elements.
Order Of Elements	HashSet doesn't maintain any order of elements.	LinkedHashSet maintains insertion order of elements. i.e elements are placed as they are inserted.	TreeSet orders the elements according to supplied Comparator. If no comparator is supplied, elements will be placed in their natural ascending order.
Performance	HashSet gives better performance than the LinkedHashSet and TreeSet.	The performance of LinkedHashSet is between HashSet and TreeSet. It's performance is almost similar to HashSet. But slightly in the slower side as it also maintains LinkedList internally to maintain the insertion order of elements.	TreeSet gives less performance than the HashSet and LinkedHashSet as it has to sort the elements after each insertion and removal operations.
Insertion, Removal And Retrieval Operations	HashSet gives performance of order O(1) for insertion, removal and retrieval operations.	LinkedHashSet also gives performance of order O(1) for insertion, removal and retrieval operations.	TreeSet gives performance of order O(log(n)) for insertion, removal and retrieval operations.
How they compare the elements?	HashSet uses equals() and hashCode() methods to compare the elements and thus removing the possible duplicate elements.	LinkedHashSet also uses equals() and hashCode() methods to compare the elements.	TreeSet uses compare() or compareTo() methods to compare the elements and thus removing the possible duplicate elements. It doesn't use equals() and hashCode() methods for comparision of elements.
Null elements	HashSet allows maximum one null element.	LinkedHashSet also allows maximum one null element.	TreeSet doesn't allow even a single null element. If you try to insert null element into

			TreeSet, it throws NullPointerException.
Memory Occupation	HashSet requires less memory than LinkedHashSet and TreeSet as it uses only HashMap internally to store its elements.	LinkedHashSet requires more memory than HashSet as it also maintains LinkedList along with HashMap to store its elements.	TreeSet also requires more memory than HashSet as it also maintains Comparator to sort the elements along with the TreeMap.
When To Use?	Use HashSet if you don't want to maintain any order of elements.	Use LinkedHashSet if you want to maintain insertion order of elements.	Use TreeSet if you want to sort the elements according to some Comparator.

12) Differentiate between Iterable and Iterator.

Iterable	Iterator
Iterable is an interface	Iterator is an interface
Belongs to java.**lang** package	Belongs to java.**util** package
Provides **one single abstract** method called **iterator**()	Provides **two abstract methods** called **hasNext**() and **next**()
It is a representation of a series of **elements** that can be traversed	It represents the **object** with iteration state

13) Differentiate between Set and Map.

Set	Map
Belongs to java.**util** package	Belongs to java.**util** package
Extends the Collection interface	**Doesn't** extend the Collection interface
Duplicate values are not allowed	Duplicate keys are not allowed but duplicate values are
Only one null values can be stored	Only **one** null key can be **stored** but **multiple** null values are **allowed**
Doesn't maintain any **insertion** order	Doesn't maintain any insertion order

14) Differentiate between HashSet and HashMap.

HashSet	HasMap
Based on Set **implementation**	Based on Map **implementation**
Doesn't allow any duplicate elements	Doesn't allow any duplicate keys but duplicate values are allowed
Allows only a single null value	Allows only one null key but any number of null values
Has **slower** processing time	Has **faster** processing time
Uses HashMap as an **underlying** data structure	Uses various hashing **techniques** for data manipulation

15) Differentiate between HashMap and TreeMap.

HashMap	TreeMap
Doesn't preserves any ordering	Preserves the natural ordering
Implicitly implements the hashing principle	Implicitly implements the Red-Black Tree Implementation
Can store only one null key	Cannot store any null key
More memory usage	Less memory usage
Not synchronized	Not synchronized

16) Differentiate between Iterator and ListIterator.

Iterator	ListIterator
Can only perform **remove** operations on the Collection elements	Can perform **add**, **remove** and **replace** operations the Collection elements
Can traverse List, Sets and maps	Can traverse **only Lists**
Can traverse the Collection in **forward** direction	Can traverse the collection in **any** direction
Provides **no method** to **retrieve** the index of the element	Provides **methods** to **retrieve** the index of the elements
iterator() method is available for the **entire** Collection Framework	listIterator() is **only** available for the collections implementing the **List** interface

17) Differentiate between HashSet and TreeSet.

HashSet	TreeSet
Uses HasMap to store elements	Uses Treemap to store elements
It is **unordered** in nature	By default, it stores elements in their **natural ordering**
Has **faster** processing time	Has **slower** processing time
Uses **hasCode**() and **equals**() for comparing	Uses **compare**() and **compareTo**() for comparing
Allows **only** one null element	**Doesn't** allow any null element after 1.7
Takes up **less** memory space	Takes up **more** memory space

18) Differentiate between Queue and Deque.

Queue	Deque
Refers to single-ended queue	Refers to double-ended queue
Elements can be added or removed from only one end	Elements can be added and removed from either end
Less versatile	More versatile

19) Differentiate between failfast and failsafe.

failfast	failsafe
Doesn't allow **modifications** of a collection while iterating	Allows modifications of a collection while iterating
Throws ConcurrentModificationException	**Don't** throw any exceptions
Uses the **original** collection to traverse over the elements	Uses a **copy** of the original collection to traverse over the elements
Don't require extra memory	Require **extra** memory

20) Differentiate between Queue and Stack.

Queue	Stack
Based on FIFO (First-In-First-Out) principle	Based on LIFO (Last-In-First-Out) principle
Insertion and deletion takes place from two opposite ends	Insertion and deletion takes place the same end
Element insertion is called enqueue	Element insertion is called push
Element deletion is called dequeue	Element deletion is called pop
Two pointers are maintained one point to the first element and the other one points the last element on the list	Only one pointer is maintained which points to the top element on the stack

21) Differentiate between PriorityQueue and TreeSet.

PriorityQueue	TreeSet
It is a type of Queue	It is based on a Set data structure
Allows duplicate elements	Doesn't allows duplicate elements
Stores the elements based on an additional factor called priority	Stores the elements in a sorted order

22) Differentiate between the Singly Linked List and Doubly Linked List.

Singly Linked List(SLL)	Doubly Linked List(DLL)
Contains nodes with a data field and a next node-link field	Contains nodes with a data field, a previous link field, and a next link field
Can be traversed using the next node-link field only	Can be traversed using the previous node-link or the next node-link
Occupies less memory space	Occupies more memory space
Less efficient in providing access to the elements	More efficient in providing access to the elements

23) Differentiate between Iterator and Enumeration.

Iterator	Enumeration
Collection element can be removed while traversing it	Can only traverse through the Collection
Used to traverse most of the classes of the Java Collection framework	Used to traverse the legacy classes such as Vector, HashTable, etc
Is fail-fast in nature	Is fail-safe in nature
Is safe and secure	Is not safe and secure
Provides methods like hasNext(), next() and remove()	Provides methods like hasMoreElements() and nextElement()

My SQL

1) Write all SQL Queries?

- ⇨ show databases;
- ⇨ create database fbj142;
- ⇨ use fbj142;
- ⇨ create table employee(id int, ename varchar(20));
- ⇨ desc employee;
- ⇨ insert into employee values(2,'A'), (3,'B'), (4,'C')
- ⇨ **insert into employee values(1,'Darshan Lohiya');**
- ⇨ **select * from employee;**
- ⇨ **select * from employee where ename='B';**
- ⇨ **update employee set ename='Darsh' where id=1;**
- ⇨ **delete from employee where id=4;**
- ⇨ **alter table employee add column salary double;**

- The **SELECT** statement is used to select data from a database. The data returned is stored in a result-set.
- The **WHERE** clause is used to filter records.
- It is used to extract only those records that fulfill a specified condition.
- The **UPDATE** statement is used to modify the existing records in a table.
- The **DELETE** statement is used to delete existing records in a table.
- The **INSERT** INTO statement is used to insert new records in a table.

2) What is MySQL database or Server?

- ⇨ MySQL is **database management system (DBMS)**.
- ⇨ DBMS is a **software used** to **store** and **manage data**.
- ⇨ **Database** consists of a structured **set of tables** and each row of a **table** is a **record**.

SQL

- ⇨ SQL stand for **structure query language**
- ⇨ SQL is a language which is **used to operate your database**.
- ⇨ SQL is the **basic language** used for all the databases.
- ⇨ SQL is used in the **accessing**, **updating**, and **manipulation** of **data** in a database.

Adavatage:

- cross platform- it can run any platform
- it is used with multiple language (java, python, .net, php)
- open source

Disadvantages

- Scalability in MySQL is a redundant task.
- MySQL serves good for large databases mostly.
- There are issues of the instability of software.

3) What is the default port for MySQL Server?

- ⇨ The default port for MySQL Server is 3306.
- ⇨ Another standard default port is 1433 in TCP/IP for SQL Server.

4) How to Test for NULL Values?

- It is not possible to test for NULL values with comparison operators, such as =, <, or <>.
- We will have to use the IS NULL & IS NOT NULL operators instead

5) What Can SQL do?

- ⇨ SQL can **execute** queries against a database
- ⇨ SQL can **retrieve** data from a database
- ⇨ SQL can **insert** records in a database
- ⇨ SQL can **update** records in a database
- ⇨ SQL can **delete** records from a database
- ⇨ SQL can **create** new databases
- ⇨ SQL can create new tables in a database
- ⇨ SQL can create stored procedures in a database

6) What is difference between SQL and MySQL?

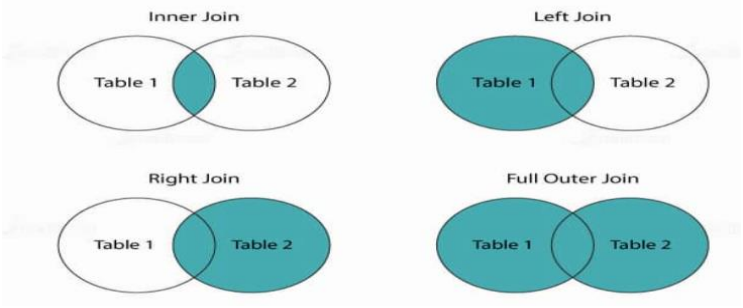
- ⇨ SQL is a query **language**, whereas MySQL is a relational **database** that uses SQL to query a database.
- ⇨ You can use SQL to access, update, and manipulate the data stored in a database. However, MySQL is a database that stores the existing data in a database in an organized manner.

7) What is Normalization?

- ⇨ Normalization is the process of organizing data in a database. This includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating redundancy and inconsistent dependency
  - **Join**:- A join clause is used to combine rows from two or more tables, based on a related column between them.
  - **Inner join**:- The inner join keyword selects records that have matching values in both tables.
  - **Left join**:- The Left join keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.
  - **Right join**:- The Right join keyword returns all records from the right

table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

- **Full outer join**:- The Full outer join keyword returns all records when there is a match in left (table1) or right (table2) table records.
- **Self-join**:- A self-join is a regular join, but the table is joined with itself.
- **Join and subquery**:- both are used to combine data from difference table into a single table



8) What is a Query?

- ⇨ A query is a request for data or information from a database table or combination of tables.
- ⇨ A database query can be either a select query or an action query.

9) Difference between union and join

Union	*Join*
Union is **combination** of rows	**Marge** the column
Not be necessary to have same column name. but no. of column and database column should be same	Combine the rows from two or more tables base on a related common column between them
Select city from table1 union Select city from table2	Select a.city, b.name from table1 a join table2 b on a.id=b.custId

10) Difference Between Delete and truncate?

Key	**Delete**	**Truncate**
Basic	It is used to **delete specific data**	It is used to **delete the entire data** of the table
Where clause	We can use with where **clause**	It **can't** be used with where clause
Locking	It **locks** the table now before deleting the row	It locks the entire table
Rollback	We can rollback the changes	We can't rollback the changes
performance	It is **slower** than truncate	It **faster** than delete

11) What is persist method?

- ⇨ The persist operation must be **used only for new entities**.
- ⇨ From JPA perspective, an entity is new when it has never been associated with a database row, meaning that there is **no table record in the database to match the entity in question**.
- ⇨ The persist method is intended for adding a new entity instance to the persistence context, i.e. transitioning an instance from transient to persistent state.
- ⇨ We usually call it when **we want to add a record to the database** (persist an entity instance)

12) What is cursor in sql database?

- ⇨ A database cursor is an identifier associated with a group of rows. It is, in a sense, a pointer to the current row in a buffer.
- ⇨ You must use a cursor in the following cases: Statements that return more than one row of data from the **database server**: A **SELECT** statement requires a select cursor.
- ⇨ Cursor is a Temporary Memory or Temporary Work Station.
- ⇨ It is allocated by Database Server at the Time of Performing DML operations on Table by User. Cursors are used to store Database Tables.
- ⇨ There are 2 types of Cursors: **Implicit Cursors, and Explicit Cursors**.
- ⇨ **Cursor**:-A cursor in sql is temporary work areas created in system memory when sql statement is executed a cursor is set of the together with pointer that identifies a current row
- ⇨ FIRST is used to fetch only the first row from cursor table.
  LAST is used to fetch only last row from cursor table.
  NEXT is used to fetch data in forward direction from cursor table.
  PRIOR is used to fetch data in backward direction from cursor table.
  ABSOLUTE n is used to fetch the exact n^{th} row from cursor table.
  RELATIVE n is used to fetch the data in incremental way as well as detrimental way.

  Syntax : FETCH NEXT/FIRST/LAST/PRIOR/ABSOLUTE n/RELATIVE n FROM cursor_name

13) which is last line we write in before stop the cursor?

- FETCH FIRST FROM s1
- FETCH LAST FROM s1
- FETCH NEXT FROM s1
- FETCH PRIOR FROM s1
- FETCH ABSOLUTE 7 FROM s1
- FETCH RELATIVE -2 FROM s1

14) Difference between primary key Foreign key and unique key ?

Primary key	Foreign key	Unique key
Unique identifier for row of the table	It is used to **link two tables** together	**Unique** identifier for row of the table when **primary key is not present**
Cannot null	Foreign key column cannot access null value	Can be null
Only one primary key can be **present in the table**	Table can have **more than one foreign key**	**Multiple unique key** can be present in the table
Selection using primary key creates clustered index	Foreign key is not clustered index by default we can make clustered index	Selection using primary key creates non-clustered index

Clustered index:- **Clustered indexes** sort and store the data rows in the table or view based on their key values.

15) What is trigger?

⇨ A trigger is special type of **stored procedure that automatically run when an event occur** in the database server. Stored program which are automatically fixed/execute when same events occurred.

Type of trigger

1. ***DML***:- DML trigger can be categorized into two type (insert, update, delete)trigger
   - DML denotes Data Manipulation Language which includes commands such as SELECT, INSERT, etc.

a. ***After trigger***: -fires after the triggering action

b. ***Instead of trigger***: - fired instated of fired triggering action

2. ***DDL***: - (create, alter, drop)
   - DDL is the abbreviation for Data Definition Language dealing with database schemas, as well as the description of how data resides in the database. An example of this is the CREATE TABLE command.
3. ***Login trigger***: -

Database Features of NoSQL

- NoSQL Databases can have a common set of features such as:
- Non-relational data model.
- Runs well on clusters.
- Mostly open-source.
- Built for the new generation Web applications.

- Is schema-less

16) Difference between RDBMS vs NoSQL

RDBMS	NoSQL
1. Data is stored in a relational model, with rows and columns. 2. A row contains information about an item while columns contain specific information, such as 'Model', 'Date of Manufacture', 'Color'. 3. Follows fixed schema. Meaning, the columns are defined and locked before data entry. In addition, each row contains data for each column. 4. Supports vertical scaling. Scaling an RDBMS across multiple servers is a challenging and time-consuming process. 5. Atomicity, Consistency, Isolation & Durability(ACID) Compliant	1. Data is stored in a host of different databases, with different data storage models. 2. Follows dynamic schemas. Meaning, you can add columns anytime. 3. Supports horizontal scaling. You can scale across multiple servers. Multiple servers are cheap commodity hardware or cloud instances, which make scaling cost-effective compared to vertical scaling. 4. Not ACID Compliant.

17) Logging in spring boot?

⇨ In Java, **Logging** is an API that provides the ability to trace out the errors of the applications. When an application generates the logging call, the Logger records the event in the LogRecord. After that, it sends to the corresponding handlers or appenders. Before sending it to the console or file, the appenders format that log record by using the formatter or layouts

⇨ **List of Java Keywords**

⇨ A list of Java keywords or reserved words is given below:

1. **abstract:** Java abstract keyword is used to declare abstract class. Abstract class can provide the implementation of interface. It can have abstract and non-abstract methods.
2. **boolean:** Java Boolean keyword is used to declare a variable as a Boolean type. It can hold True and False values only.
3. **break:** Java break keyword is used to break loop or switch statement. It breaks the current flow of the program at specified condition.
4. **byte:** Java byte keyword is used to declare a variable that can hold an 8-bit data values.
5. **case:** Java case keyword is used to with the switch statements to mark blocks of text.

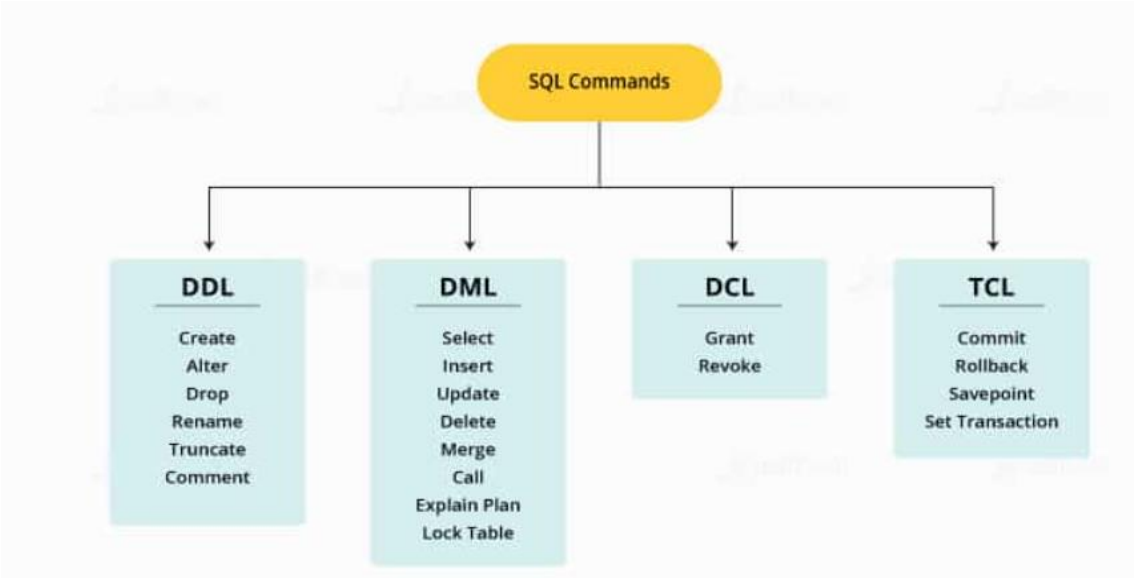
6. **catch:** Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.
7. **char:** Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters
8. **class:** Java class keyword is used to declare a class.
9. **continue:** Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.
10. **default:** Java default keyword is used to specify the default block of code in a switch statement.
11. **do:** Java do keyword is used in control statement to declare a loop. It can iterate a part of the program several times.
12. **double:** Java double keyword is used to declare a variable that can hold a 64-bit floating-point numbers.
13. **else:** Java else keyword is used to indicate the alternative branches in an if statement.
14. **enum:** Java Enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.
15. **extends:** Java extends keyword is used to indicate that a class is derived from another class or interface.
16. **final:** Java final keyword is used to indicate that a variable holds a constant value. It is applied with a variable. It is used to restrict the user.
17. **finally:** Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether exception is handled or not.
18. **float:** Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.
19. **for:** Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some conditions become true. If the number of iteration is fixed, it is recommended to use for loop.
20. **if:** Java if keyword tests the condition. It executes the if block if condition is true.
21. **implements:** Java implements keyword is used to implement an interface.
22. **import:** Java import keyword makes classes and interfaces available and accessible to the current source code.
23. **instanceof :** Java instanceof keyword is used to test whether the object is an instance of the specified class or implements an interface.
24. **int :** Java int keyword is used to declare a variable that can hold a 32-bit signed integer.
25. **interface :** Java interface keyword is used to declare an interface. It can have only abstract methods.
26. **long :** Java long keyword is used to declare a variable that can hold a 64-bit integer.
27. **Native:** Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).
28. **new :** Java new keyword is used to create new objects.

29. **null:** Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.
30. **package:** Java package keyword is used to declare a Java package that includes the classes.
31. **private:** Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.
32. **protected:** Java protected keyword is an access modifier. It can be accessible within package and outside the package but through inheritance only. It can't be applied on the class.
33. **public:** Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.
34. **return:** Java return keyword is used to return from a method when its execution is complete.
35. **short:** Java short keyword is used to declare a variable that can hold a 16-bit integer.
36. **static:** Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is used for memory management mainly.
37. **strictfp:** Java strictfp is used to restrict the floating-point calculations to ensure portability.
38. **super:** Java super keyword is a reference variable that is used to refer parent class object. It can be used to invoke immediate parent class method.
39. **switch:** The Java switch keyword contains a switch statement that executes code based on test value. The switch statement tests the equality of a variable against multiple values.
40. **synchronized:** Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.
41. **this:** Java this keyword can be used to refer the current object in a method or constructor.
42. **throw:** The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exception. It is followed by an instance.
43. **throws:** The Java throws keyword is used to declare an exception. Checked exception can be propagated with throws.
44. **transient:** Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.
45. **try:** Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.
46. **void:** Java void keyword is used to specify that a method does not have a return value.
47. **volatile:** Java volatile keyword is used to indicate that a variable may change asynchronously.

48. **while:** Java while keyword is used to start a while loop. This loop iterates a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

18) What do DDL, DML, and DCL stand for?

- **DDL** is the abbreviation for **Data Definition Language** dealing with database schemas, as well as the description of how data resides in the database. An example of this is the CREATE TABLE command.
- **DML** denotes **Data Manipulation Language** which includes commands such as SELECT, INSERT, etc.
- **DCL** stands for **Data Control Language** and includes commands like GRANT, REVOKE, etc.



19) What is the difference between CHAR and VARCHAR?

- When a table is created, CHAR is used to define the **fixed length** of the **table** and **columns**. The length value could be in the range of 1–255.
- The **VARCHAR** command is used to **adjust the column and table lengths** as required.

20) What are Heap Tables?

- Heap tables are **in-memory tables used for high-speed temporary storage**.
- But **TEXT** or **BLOB** fields are **not allowed** within them.
- They also do not support AUTO INCREMENT**.**

21) What is the difference between the DELETE TABLE and TRUNCATE TABLE commands in MySQL?

- Basically, DELETE TABLE is a logged operation, and **every row deleted is logged**. Therefore, the process is usually slow.
- TRUNCATE TABLE also **deletes rows in a table**, but it will **not log any** of the rows deleted.
- TRUNCATE TABLE process is faster here in comparison.
- TRUNCATE TABLE can be rolled back and is functionally similar to the DELETE statement without a WHERE clause.

22) What are the TRUNCATE, DELETE and DROP statements?

- **DELETE** statement is used to delete rows from a table**.**

```
DELETE FROM Candidates
WHERE CandidateId > 1000
```

- **TRUNCATE** command is used to delete all the rows from the table and free the space containing the table.

```
TRUNCATE TABLE Candidates;
```

- **DROP** command is used to remove an object from the database. If you drop a table, all the rows in the table are deleted and the table structure is removed from the database.

```
DROP TABLE Candidates;
```

- Write a SQL statement to wipe a table 'Temporary' from memory.
- Write a SQL query to remove first 1000 records from table 'Temporary' based on 'id'.
- Write a SQL statement to delete the table 'Temporary' while keeping its relations intact

23) What is the difference between the primary key and the candidate key?

- The **primary** key in MySQL is used to **identify every row of a table in a unique manner**.
- For one table, there is only one primary key.
- The candidate keys can be used to **reference the foreign keys.**
- One of the candidate keys is the primary key.

24) Is there an object-oriented version of MySQL library functions?

- Yes.
- **MySQLi** is the object-oriented version of MySQL, and it interfaces in PHP.

25) What is the storage engine used for MySQL?

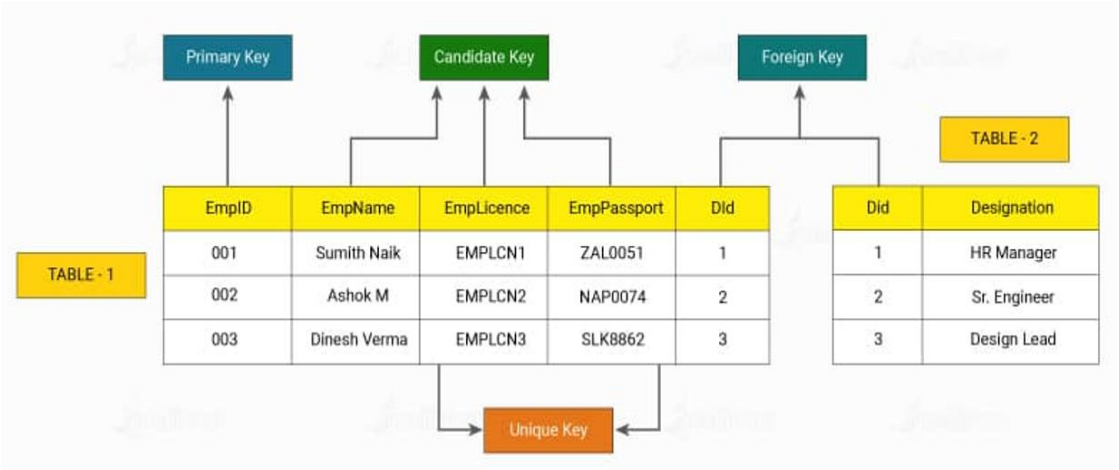
- Storage tables are named table types.
- The data is stored in the files using multiple techniques such as indexing, locking levels, capabilities, and functions.

26) What are some of the common MySQL commands?

Command	Action
ALTER	To alter a database or table
BACKUP	To back-up a table
\c	To cancel Input
CREATE	To create a database
DELETE	To delete a row from a table
DESCRIBE	To describe a table's columns
DROP	To delete a database or table
EXIT(ctrl+c)	To exit
GRANT	To change user privileges
HELP (\h, \?)	Display help
INSERT	Insert data
LOCK	Lock table(s)
QUIT(\q)	Same as EXIT
RENAME	Rename a Table
SHOW	List details about an object
SOURCE	Execute a file
STATUS (\s)	Display the current status
TRUNCATE	Empty a table
UNLOCK	Unlock table(s)
UPDATE	Update an existing record
USE	Use a database

27) What is the difference between primary key and unique key?

- ⇨ While both are used to enforce the uniqueness of the column defined, the primary key would create a **clustered index**.
- ⇨ whereas the unique key would create a **non-clustered index** on the column.
- ⇨ The primary key does not allow ‘NULL’, but the unique key does**.**



28) What are the differences between a primary key and a foreign key?

Primary Key	Foreign Key
It helps in the unique identification of data in a database	It helps establish a link between tables
There can be only one primary key for a table	There can be more than one foreign key for a table
Primary key attributes cannot have duplicate values in a table	Duplicate values are acceptable for a foreign key
Null values are not acceptable	Null values are acceptable
We can define primary key constraints for temporarily created tables	It cannot be defined for temporary tables
The primary key index is automatically created	The index is not created automatically

29) What are the String Data Types in MySQL?

Type Name	Meaning
CHAR	fixed-length nonbinary(character) string
VARCHAR	variable-length nonbinary string
BINARY	fixed-length binary string
VARBINARY	variable-length binary string
TINYBLOB	Very small BLOB(binary large object)
BLOB	Small BLOB
MEDIUMBLOB	Medium-sized BLOB
LONGBLOB	Large BLOB
TINYTEXT	A very small nonbinary string

Type Name	Meaning
TEXT	Small nonbinary string
MEDIUMTEXT	Medium-sized nonbinary string
LONGTEXT	Large nonbinary string
ENUM	An enumeration; each column value is assigned, one enumeration member
SET	A set; each column value is assigned zero or more set members
NULL	NULL in SQL is the term used to represent a missing value. A NULL value in a table is a value in a field that appears to be blank. This value is different than a zero value or a field that contains spaces.

30) SQL Composite Key

- ⇨ A composite key is a **combination of two or more columns** in a table that can be used to uniquely identify each row in the table when the columns are combined uniqueness is guaranteed, but when it taken individually it does not guarantee uniqueness.
- ⇨ Sometimes **more than one attributes are needed to uniquely identify an entity**.
- ⇨ A primary key that is made by the **combination of more than one attribute** is known as a **composite key**.
- ⇨ Composite key is a key which is the **combination of more than one field or column** of a given table. It may be a candidate key or primary key.

31) What are Aggregate and Scalar functions?

- ⇨ An **aggregate function** performs operations on **a collection of values to return a single scalar value**.
- ⇨ Aggregate functions are often used with the GROUP BY and HAVING clauses of the SELECT statement.
- ⇨ Following are the widely used SQL aggregate functions:

- **AVG**() - Calculates the mean of a collection of values.
- **COUNT**() - Counts the total number of records in a specific table or view.
- **MIN**() - Calculates the minimum of a collection of values.
- **MAX**() - Calculates the maximum of a collection of values.
- **SUM**() - Calculates the sum of a collection of values.
- **FIRST**() - Fetches the first element in a collection of values.
- **LAST**() - Fetches the last element in a collection of values.

> ➢ **Note**: All aggregate functions described above ignore NULL values except for the COUNT function.

⇨ A **scalar function** returns a **single value based on the input value**.
⇨ Following are the widely used SQL scalar functions:

- **LEN**() - Calculates the total length of the given field (column).
- **UCASE**() - Converts a collection of string values to uppercase characters.
- **LCASE**() - Converts a collection of string values to lowercase characters.
- **MID**() - Extracts substrings from a collection of string values in a table.
- **CONCAT**() - Concatenates two or more strings.
- **RAND**() - Generates a random collection of numbers of a given length.
- **ROUND**() - Calculates the round-off integer value for a numeric field (or decimal point values).
- **NOW**() - Returns the current date & time.
- **FORMAT**() - Sets the format to display a collection of values.

32) Second Highest Salary in MySQL and SQL Server

⇨ **Show databases;**

```
mysql> show databases;
+--------------------------------+
| Database                       |
+--------------------------------+
| information_schema             |
| car_loan_final_project         |
| fbj142                         |
| imageuploading_dynamicpath     |
| mysql                          |
| springboot_jdbc                |
| test                           |
+--------------------------------+
```

⇨ **Desc employee;**

```
mysql> desc employee;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int(11)     | YES  |     | NULL    |       |
| ename  | varchar(20) | YES  |     | NULL    |       |
| salary | double      | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
```

⇨ **Insert employee values(1,'A',5600.1);**

```
mysql> insert employee values(1,'A',5600.1);
Query OK, 1 row affected (0.04 sec)
```

⇨ **Select * from employee;**

```
mysql> select * from employee;
+------+-------+--------+
| id   | ename | salary |
+------+-------+--------+
|    1 | A     | 5600.1 |
|    5 | B     | 6600.1 |
|    2 | C     | 7600.1 |
|    3 | D     | 8600.1 |
|    4 | F     | 9600.1 |
+------+-------+--------+
```

⇨ **SELECT MAX(salary) FROM employee WHERE salary < (SELECT MAX(salary) FROM employee);**

```
mysql> SELECT MAX(salary) FROM employee WHERE salary < (SELECT MAX(salary) FROM employee);
+-------------+
| MAX(salary) |
+-------------+
|      8600.1 |
+-------------+
```

⇨ **Select Max(salary) as salary from employee;**

```
mysql> Select Max(salary) as salary from employee;
+--------+
| salary |
+--------+
| 9600.1 |
+--------+
```

⇨ **Select MIN(salary) as salary from employee;**

```
mysql> Select MIN(salary) as salary from employee;
+---------+
| salary  |
+---------+
| 5600.1  |
+---------+
```

⇨ **SELECT * FROM `employee` ORDER BY `salary` DESC LIMIT 1 OFFSET 2;**

```
mysql> SELECT * FROM `employee` ORDER BY `salary` DESC LIMIT 1 OFFSET 2;
+------+-------+--------+
| id   | ename | salary |
+------+-------+--------+
|    2 | C     | 7600.1 |
+------+-------+--------+
```

A. **PRIMARY KEY constraint?**

⇨ A table in SQL is indexed by default based on its primary key

B. **FOREIGN KEY constraint?**

⇨ Foreign Key is automatically created when two tables are joined.

C. **What is a Query?**

⇨ A request for data from single or multiple tables in the database.

D. **Query to select all records with "bar" in their name?**

⇨ SELECT * FROM people WHERE name LIKE "%bar%";

E. **SQL query used to fetch unique values from a field?**

⇨ SELECT DISTINCT column_name FROM table_name;

F. **An SQL query to delete a table from the database and memory while keeping the structure of the table intact?**

⇨ TRUNCATE TABLE table_name;

G. **How many columns can be used for creating Index?**

⇨ Maximum of 16 indexed columns can be created for any standard table.

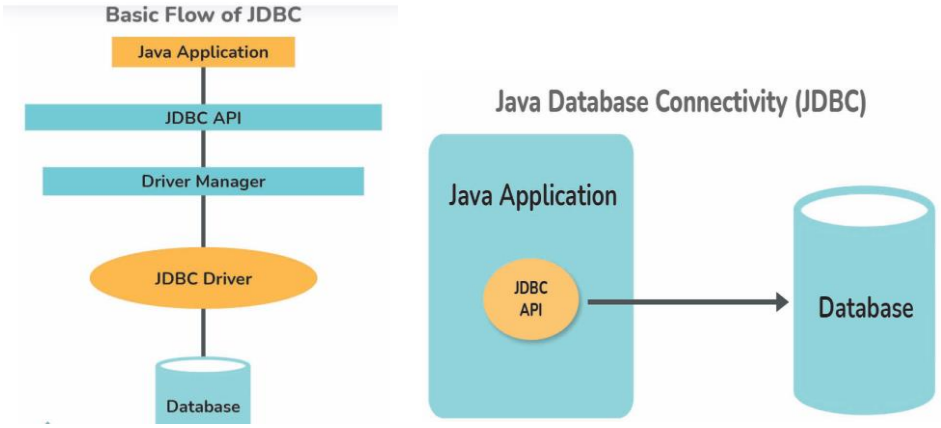
JDBC

1) What is JDBC? Why JDBC?

- ⇨ JDBC stand for **Java Database Connectivity**.
- ⇨ It is a Standard **API** provide by Oracle for java application.
- ⇨ JDBC is part of "j2se" (Java Second Standard Edition).
- ⇨ JDBC is an database independent API. By using JDBC API we can perform basic crud operation.
- ⇨ API for **communicating** to relational databases, API has java **classes** and **interfaces** using that developer can **easily interact with database**.

❖ Why JDBC?

- ⇨ If we store data **permanently** and **get any time** so we can use the concept of jdbc.
- ⇨ Hibernate base depends on jdbc.
- ⇨ Java application connect to database by using jdbc.



- ⇨ Database cannot understand java language so that why we use "SQL" because database understand SQL language.

2) What are the Main Components Of Jdbc?

- ⇨ The life cycle of a servlet consists of the following phases:
- ⇨ **DriverManager**:
  - Driver Manager is a class.
  - Manages a list of database drivers.
  - Matches connection requests from the java application with the proper database driver using communication subprotocol.
  - The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.
- ⇨ **Driver**:
  - The database communications link, handling all communication with the database.
  - Normally, once the driver is loaded, the developer need not call it

explicitly.

- ⇨ **Connection**:
  - Interface with all methods for contacting a database.
  - The connection object represents communication context,
  - i.e., all communication with database is through connection object only.
- ⇨ **Statement**:
  - Encapsulates an SQL statement which is passed to the database to be parsed, compiled, planned and executed.
- ⇨ **ResultSet**:
  - The ResultSet represents set of rows retrieved due to query execution.

3) Write down communication steps with database?

- ⇨ Load Driver class
  - **Class.forName("com.mysql.jdbc.Driver");**
- ⇨ Establish jdbc connection / Create a connection
  - **url = "jdbc:mysql://localhost:3306/emptable"**
  - **user = "root"**
  - **password= "root"**
  - **Connection con = DriverManager.getConnection("url", "user", "password");**
- ⇨ Construct SQL query
  - **String sql = create table_name(column 1 datatype, column 2 datatype, column 3 datatype);**
- ⇨ Establish statement object
  - **Statement smt = con.createStatement();**
  - If we want requestset
  - **String s = "Select * for Employee";**
  - **Requestset rs = smt.executeQuery(s);**
- ⇨ Submit SQL query
  - **smt.execute(sql);**
- ⇨ Close Connection
  - **con.close;**
  - **smt.close;**

4) How to check connection is created or not?

- **url = "jdbc:mysql://localhost:3306/emptable"**
- **user = "root"**
- **password= "root"**
- **Connection con = DriverManager.getConnection("url", "user", "password");**
- **If(con.isclosed()) { sop("Connection is closed")} else{sop("Connection is created")};**

5) Write down the all jdbc query?

1. **Create Table:-**
   - **String q = "create table student(rollno int primary key, name varchar(30), addr varchar(20)";**
2. **Insert Data:-**
   - **String q = "insert student values(1, "Darshan", "Pune");**
3. **Update Data:-**
   - **String q = "update student set name= "Darsh" where rollno=1;**
4. **Delete Data :-**
   - **String q = "delete form student where rollno=1;**

6) Write a program for insert, update, delete?

```
public class Darshan
{
public static void main(String[] args) throws SQLException,
ClassNotFoundException
{
        Class.forName("com.mysql.jdbc.Driver");

          Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/employee","root",
"root");

      //  String s ="create table Darshan(age int, name varchar(20), salary
      varchar(5), mobno varchar(10), primary key(age))";

              String x="insert into Darshan values(45,'Dhananjay',
'56000','965965')";

//            String sss="update Darshan set age=25, name = Darsh, where
mobno ='965965'";

//String sss="delete from Darshan where specialization = 'Heart'";
```

```
            Statement smt = con.createStatement();

//          smt.execute(s);
            smt.execute(x);
//            smt.execute(sss);
            con.close();
            System.out.println("--Doctor table Create---");
            }
        }
```

7) Explain JDBC API components.

- **Interfaces:**
  - **Connection**: The object of Connection is created by using **getConnection**() method of DriverManager class. **DriverManager** is the factory for connection.
  - **Statement**: The object of the Statement is created by using **createStatement**() method of the Connection class. The **Connection** interface is the factory for Statement.
  - **PreparedStatement**: The PreparedStatement object is created by using **prepareStatement**() method of **Connection** class. It is used for executing the parameterized query.
  - **ResultSet:** The ResultSet object maintains a cursor pointing to a table row. At first, the cursor points before the first row. The **executeQuery**() method of the Statement interface returns the object of ResultSet.
  - **ResultSetMetaData**: The ResultSetMetaData interface object contains the details about the data(table) such as number of columns, name of the column, column type etc. The **getMetaData**() method of ResultSet returns the ResultSetMetaData object.
  - **DatabaseMetaData**: It is an interface that has methods to get metadata of a database, like name of the database product, version of database product, driver name, name of the total number of views, name of the total number of tables, etc. The **getMetaData**() method that belongs to Connection interface returns the DatabaseMetaData object.
  - **CallableStatement**: CallableStatement interface is useful for calling the stored procedures and functions. We can have business logic on the database through the usage of stored procedures and functions, which will be helpful for the improvement in the performance as these are pre-compiled. The **prepareCall**() method that belongs to the Connection interface returns the object of CallableStatement.

❖ **Classes**:

⇨ **DriverManager**: It pretends to be an interface between the user and drivers. DriverManager keeps track of the available drivers and handles establishing a connection between a database and the relevant driver. It contains various methods to keep the interaction between the user and drivers.

⇨ **BLOB:** BLOB stands for Binary Large Object. It represents a collection of binary data such as images, audio, and video files, etc., which is stored as a single entity in the DBMS(Database Management System).

⇨ **CLOB:** CLOB stands for Character Large Object. This data type is used by multiple database management systems to store character files. It is the same as BLOB except for the difference, instead of binary data, CLOB represents character stream data such as character files, etc**.**

8) What is the difference between Statement and PreparedStatement?

Statement	PreparedStatement
One Object of statement we can submit multiple SQL query.	One object of prepared statement we can submit only one SQL query.
The query is **compiled every time we run the program**.	The query is **compiled only** once.
It is used in the situation **where we need to run the SQL query without providing parameters at runtime**.	It is used when **we want to give input parameters to the query at runtime**.
Performance is **less** compared to PreparedStatement.	Provides **better** performance than Statement, as it executes the pre-compiled SQL statements.
It is suitable for executing DDL statements such as CREATE, ALTER, DROP and TRUNCATE.	It is suitable for executing DML statements such as INSERT, UPDATE, and DELETE.
It **cannot** be used for storing/retrieving images and files in the database.	It can be used for storing/retrieving images and files in the database.
It executes static SQL statements.	It executes pre-compiled SQL statements.
Less secured as it enforces SQL injection.	More secured as they use bind variables, which can prevent SQL injection.

9) How to call Stored Procedures in JDBC?

- ⇨ We can **execute the SQL** Stored procedures through the **CallableStatement** interface.
- ⇨ The CallableStatement **object** can be created using the **prepareCall**() method of the Connection interface.

10) What is the ResultSet interface?

- ⇨ ResultSet interface is **used to store the output data after the SQL query execution**.
- ⇨ The object of ResultSet maintains the **cursor point** at the result data.
- ⇨ As a default, the cursor points before the **first row of the result data**.
- ⇨ We can traverse the data in the resultset objects as well.

Syntax:

Statement Interface:

- ⇨ Statement smt = conn.createStatement();
- ⇨ ResultSet resultset = smt.executeQuery(“Select * from EMPLOYEE”);

PreparedStatement Interface:

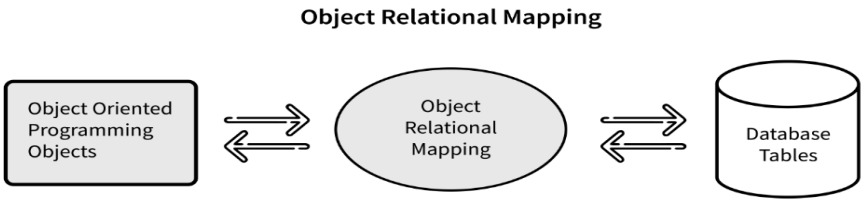
- ⇨ PreparedStatement ps = conn.prepareStatement(insert_query);
- ⇨ ResultSet resultset = ps.executeQuery(“Select * from EMPLOYEE”);

Hibernate

1. **What is Hibernate & ORM tool?**

(ORM)

- ⇨ **Object Relationship Mapping** tool.
- ⇨ It is a programming technique that maps the **object to the data stored in the database**.
- ⇨ It overcomes the **mismatch** between **OOP language & database**.
- ⇨ It **reduces** developer's efforts, time and cost.
- ⇨ It provides **connectivity** to the database.
- ⇨ There are several **ORM tool available** in market like Hibernate, I-Batis, My-Batis, TopLink etc



(Hibernate)

- ⇨ Hibernate is Java **Framework** that simplifies development of Java Application to interaction with database.
- ⇨ Hibernate creates SQL (Structured Query Language) queries at runtime according to database.
- ⇨ It provides **automatic table creation** feature.
- ⇨ It provides hibernate **relationship mapping**
  - **IS-A** relationship (Default, Single Table, Joint, table per Class)
  - **HAS-A** relationship (one to one, one to many, many to one, many to many).
- ⇨ It provides **primary key auto increment** feature.
- ⇨ It **converts all checked exception into uncheck** exception.
- ⇨ It provides **cache** mechanism. (First and Second Level cache)
- ⇨ It provides **HQL** (Hibernate Query Language) query and Criteria API.
- ⇨ It provides **criteria builder** API.
- ⇨ It provides **validation** features.

2. **What is disadvantage of hibernate?**

- ⇨ **Can't** perform **multiple insert** operations.
- ⇨ **Debugging** is **difficult** as compare to JDBC.
- ⇨ Contain lots of **boiler plate** code.
- ⇨ Can't be used for small type of applications.
- ⇨ **Slow** execution as it performs SQL queries at runtime.

3. Difference between get () & load ()?

Get ()	load ()
Eager Loading	**Lazy** Loading
It always **hit** the database.	It does **not hit** the database.
It gives you an actual object	It gives you a proxy object
If value is absent in database then it returns **null**.	If value is absent in database then hibernate exception (ObjectNotFoundException) occurs.
It always **hit** database.	It may or may not be hit to database.
If you are not sure that object exist then use get() method	If you are sure that object exist then use load() method

4. What are some of the important interfaces of Hibernate framework?

⇨ Hibernate core interfaces are:

A. Configuration :-

- ⇨ It configuration **xml.file** (Used in convert JDBC to hibernate)
- ⇨ Represent **configuration or properties file** required by the hibernate.

B. SessionFactory (Interface):-

- ⇨ The Session factory is a factory of **session and client of connection provider**.
- ⇨ It **holds second level cache** of data.
- ⇨ The **org.hibernate.SessionFactory** interface provides factory **method** to get the object of session.
- ⇨ It is thread safe.
- ⇨ Only one session in application.

C. Session :-

- ⇨ It is used to **physical connection between database**.
- ⇨ Which perform **CRUD** operation is not thread safe.
- ⇨ **Method** of Session is get(), load(), delete(), persist(), fetch(), merge(), store()

D. Criteria :-

- ⇨ It is used to **create and execute the objec**t mention criteria queries to retrieve the object.

E. Query :-

- ⇨ SQL query and HQL.
- ⇨ **Retrieve string data from database and create an object**.

F. Transaction :-

- ⇨ It's represent unit of work with the **database**.

- ⇨ In such case **if one step is failed the whole transaction is failed**.
- ⇨ most of the **RDBMS** support transaction functionality
- ⇨ it is used the acid property (Atomicity Consistency Isolation Durability)
- ⇨ A **transaction** is associated with **Session and instantiated** by calling session. beginTransaction().
- ⇨ **@Transaction**:-The transactional annotation itself defines the scope of a single database transaction
- ⇨ It maintains abstraction from the transaction implementation (JTA:-java transaction API), JDBC).

5. What is singleton class how to ways breaking singleton how to avoid?

❖ **What is singleton class:-**

- ⇨ Restricts the instantiation(means object) a **class to only ONE single instance**
- ⇨ **Hide the constructor** of the **class to ensure** that only one instance of the single class ever exists
- ⇨ Provides **global access** to that **instance** an operation that **returns the sole instance** of the class.

❖ **How to implement:-**

- ⇨ Declaring all **constructors** of the class to be **private**
- ⇨ Providing a **static method** that returns a reference to the instance
- ⇨ It is set of operation used to perform **a logical unit of work**

6. What is HQL?

- ⇨ Its Object-Oriented programming **query** similar to SQL.
- ⇨ But **instead** of operating on **table** and **column**.
- ⇨ HQL work with **persistence** object and there properties.
- ⇨ HQL query is database **independent**.
- ⇨ Keyword like select, from and where etc. and not case sensitive.

7. How to disable 1st level cache?

- We **can't disable cache** but we can **clear all cache using clear( )** method of session, then it works as disabled.

8. How to enable 2nd level cache?

- ⇨ Add **third party jar files**. (ehcache)
- ⇨ Add Annotation below to Entity class.

```
@Cache(usage=CacheConcurrencyStrategy.READ_WRITE)
```

⇨ **Include two extra lines in settings in 'HibernateUtil.class'**

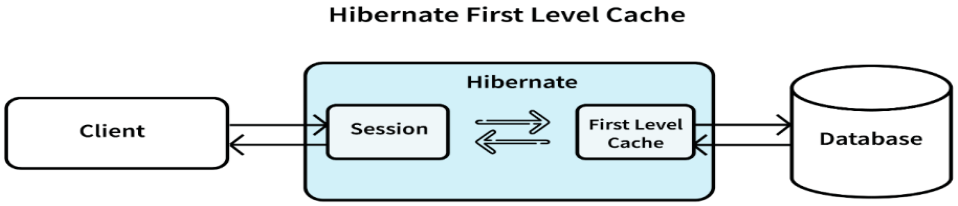
map.put(Environment.USE_SECOND_LEVEL_CACHE,"true");

map.put(Environment.CACHE_REGION_FACTORY, "org.hibernate.cache.ehcache.EhCacheRegionFactory");

9. Explain First Level Cache and Second level cache?

❖ **First Level Cache:**

- ⇨ This level is **enabled by default**.
- ⇨ The first level cache resides in the **hibernate session object**.
- ⇨ We can check the in session any object store or not by using **session.contains**(pass the ref object);
- ⇨ Since it belongs to the session object, the scope of the data stored here will not be available to the entire application as an application can make use of multiple session objects.



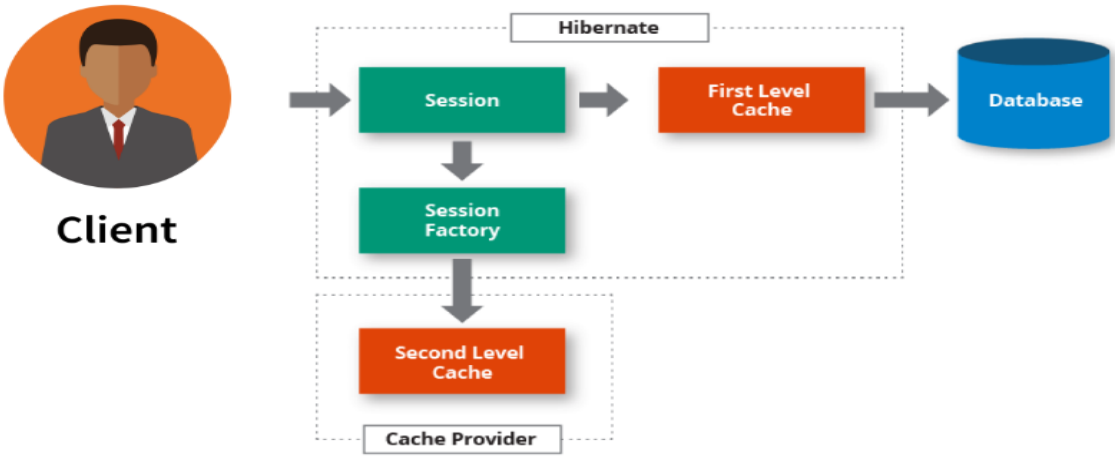
❖ **Second Level Cache:**

- ⇨ For Second Level cache we need to use third party vender jar files.
- ⇨ Second Level cache it means global cache memory will get created and all session object pointing to that global cache memory.
- ⇨ It means record only one time retrieve from database and every time use same data.
- ⇨ Foe enable second level cache we need to add two new configuration.

map.put(Environment.USE_SECOND_LEVEL_CACHE,"true");

map.put(Environment.CACHE_REGION_FACTORY, "org.hibernate.cache.ehcache.EhCacheRegionFactory");

- ⇨ We add **dependency** (**Ehcache and Hibernate-ehcache**) in pom.xml file.
- ⇨ Second level **cache** resides in the **SessionFactory** object and due to this, the data is accessible by the entire application.
- ⇨ This is **not** available by **default**.
- ⇨ It has to be **enabled explicitly.**
- ⇨ EH (Easy Hibernate) Cache, Swarm Cache, OS Cache, JBoss Cache are some example cache providers.



10.What is the difference between first level cache and second level cache?

First Level Cache	Second Level Cache
This is **local** to the **Session** object and cannot be shared between multiple sessions.	This cache is maintained at the **SessionFactory** level and shared among all sessions in Hibernate.
This cache is **enabled** by **default** and there is **no way to disable it.**	This is **disabled** by **default**, but we can enable it through configuration.
The first level cache is available only **until** the **session** is open, once the **session is closed, the first level cache is destroyed.**	The second-level cache is available through the application's **life cycle**, it is only **destroyed and recreated** when an application is **restarted**.

11.Store Procedure?

- ⇨ Store Procedure is set of **re-compile one or more SQL query** which is store (or present) in database side.
- ⇨ It increases performance of our application.
- ⇨ Reusability of processor or query.

```
StoreProcedureQuery spq = session.createStoreProcedureCall("featchAllData", Student.class);
List<Student> list = spq.getResultList();
```

12.Write down HQL Query?

❖ **Insert :-**

- ⇨ Query q = session.createQuery("**insert** into employee(empid, empname)" + "Select rollno, name, addr from Student");
- ⇨ int i = q.executeUpdate();

❖ **Update :-**

- ⇨ Query q = session.createQuery("**update** employee set empname **=:** nam where empid **=:** eid");

- **Delete :-**
  - ⇨ Query q = session.createQuery("**delete** from employee where empid=: eid");

13.What can we write in HiberntaeUtil class

```
public class HibernateUtil
{
        private static SessionFactory sf;
        private static StandardServiceRegistry ssr;

        public static SessionFactory getdata()
        {
                Map<String, Object> map = new HashMap<String, Object>
();

                map.put(Environment.DRIVER, "com.mysql.jdbc.Driver");
                map.put(Environment.URL,
"jdbc:mysql://localhost:3306/pratical_task_7");
                map.put(Environment.USER, "root");
                map.put(Environment.PASS, "root");

                map.put(Environment.DIALECT,
"org.hibernate.dialect.MySQL5Dialect");
                map.put(Environment.HBM2DDL_AUTO, "update");
                map.put(Environment.SHOW_SQL, "true");

                if(sf==null)
                {
                ssr=new
StandardServiceRegistryBuilder().applySettings(map).build();

                MetadataSources mds = new MetadataSources(ssr);

                mds.addAnnotatedClass(Student.class);

                Metadata md = mds.getMetadataBuilder().build();

                sf = md.getSessionFactoryBuilder().build();
                }
        return sf;
        }
}
```

14.What is Session?

- ⇨ A session is an **object** that maintains the **connection between** Java object **application** and **database**.
- ⇨ Session also has **methods** for storing, retrieving, modifying or deleting data from database using methods like persist(), load(), get(), update(), delete(), etc.
- ⇨ Additionally, it has **factory methods to return Query, Criteria, and Transaction objects**.

15.What is Session Factory?

- ⇨ SessionFactory is an **interface**.
- ⇨ It contains all **DB related property** details (pulled from either hibernate.cfg.xml file or hibernate.properties file)
- ⇨ SessionFactory is a factory for **Session objects**. (Allow to create any number of session objects developer wants)
- ⇨ We can **create one SessionFactory per database** in any application.
- ⇨ It is usually created during application start up.
- ⇨ It **holds 2nd level cache data**.

16.What are methods present in Session?

- It **wraps** the JDBC connection.
- It is factory of **Transaction, Query and Criteria**.
- The session object provides an **interface** between the **application** and **data** stored in the database.
- It holds **a first-level cache** (mandatory) of data.
- The org.hibernate.Session interface provides methods **to insert, update and delete** the object.

(Session Method)

- beginTransaction()
- **save**()
- **update**()
- **persist**()
- **delete**()
- **saveOrUpdate**()
- **createQuery**()
  - Create a new instance of Query for the given HQL query string.
- **createSQLQuery**()
  - Create a new instance of SQLQuery for the given SQL query string.
- **merge**()

17. In Many-to-Many, how many tables are created by-default and if mapped by is used?

- ⇨ By-default it will create 4 tables.
- ⇨ For mapped by it will create 3 tables.

18. What is Dirty Checking?

- If we **get record & we set again** then it is **updated without calling update method**, its because of dirty checking.
- This can be avoided by using @Immutable annotation.

19. What does hbm2ddl does?

- ⇨ It validates number of column.

20. Difference between SQL and HQL?

SQL	HQL
It is **not** support object-oriented query.	It is **support** object-oriented query.
SQL **depends** on table **name and column name.**	HQL **depends** on **entity class** and database or persistence object and their properties.
SQL query database **dependent**	HQL query database **independent**.
SQL is **faster** than HQL.	HQL is **slower** than SQL. Because automatically HQL convert in SQL.
It is **difficult** to convert and maintain.	It is **easy** to convert and maintain.
Ex. select * from student (use table name)	Ex. from student (use entity name)

21. Difference types of annotation in hibernate?

Annotations	Use of annotations
@Entity	Used for declaring any POJO **class** as an entity for a database
@Table	This annotation specifies the table name and its optional. If you does-not use @Table annotation hibernate will use the class name as the table name by default.
@Id	Used for declaring a **primary key inside our POJO class**
@GeneratedValue	Hibernate automatically generate the values with reference to the internal sequence and we don't need to set the values manually. Ex. @GeneratedValue(strategy=GenerationType.AUTO)

@Column	It is used to specify **column mappings**. It means if in case we don't need the name of the column that we declare in POJO but we need to refer to that entity you can change the name for the database table.
@Transient	Tells the hibernate, **not to add/save this particular column**. Cannot send the data in database.
@Temporal	This annotation is used to **format the date for storing** in the database
@Lob	Used to tell hibernate that it's a **large** object and is not a simple object
@OrderBy	This annotation will tell hibernate to OrderBy as we do in SQL. For example – we need to order by student first name in ascending order @OrderBy("firstName asc")

21. Difference between save() & persist()?

Save ()	Persist ()
Its **return** type is **Serializable** object.	Its **return** type is **void**.
It can save object within **transaction boundaries** and **outside boundaries**.	It can **only** save object within the **transaction boundaries**.
It is only supported by **Hibernate**.	It is supported by **Hibernate** and also by **JPA** (Java Persistence API).
It will create a **new row in the table for detached object**.	It will **throw** persistence **exception** for **detached object**.

23. What is Query Cache in Hibernate?

- ⇨ Hibernate implements a **separate cache region** for **queries** resultset that integrates with the Hibernate second-level cache.
- ⇨ This is also an optional feature and requires a few more steps in code.

24. What is Hibernate Caching?

- ⇨ Hibernate caches Query data and makes the **application run faster**.
- ⇨ If used correctly, the hibernate cache can be very useful in achieving **the faster application running performance.**
- ⇨ The main idea lying behind the **cache is reducing the number of database queries**, which results in reduced throughput time of the application.

25. Explain about Hibernate Proxy and how it helps in Lazy loading?

- ⇨ Hibernate uses a proxy object in order to support *Lazy loading*.
- ⇨ When you **try loading data from tables**, Hibernate **doesn't load all the mapped objects**.
- ⇨ After you **reference** a **child object** through getter methods, if the **linked entity is not present in the session cache**, then the proxy code will be entered to the database and load the linked object.
- ⇨ It uses Java assist to effectively and dynamically generate sub-classed implementations of your entity objects.

26. What are the different states of a persistent entity? Or What are the states of an object? Explain them?

- ⇨ It may exist in one of the following 3 states:

❖ **Transient**:

- ⇨ When we created new **object** and **set** the values this object will goes to transient state.
- ⇨ But transient state is **not associated** with the **Session** and **database**.
- ⇨ The transient objects exist in the **heap memory**.
- ⇨ They are **independent** of Hibernate.

❖ **Persistent**:

- ⇨ When we **save the object in session.save();** that time this **object is associated** with **Persistent state** and **Session** variable and also associate with **database**.
- ⇨ You can make a transient instance persistent by associating it with a Session.
- ⇨ Persistent means data is store in database in the form of table and object is associated with session variable.
- ⇨ If we change the value of object in persistent state so data can be change in database.
- ⇨ we can say that an object is in the persistence state when we use **save**( ), **persist**( ), **update**( ), **saveOrUpdate**( ), **lock**( ), **merge**( )

❖ **Detached**:

- ⇨ If you **session.close()** the Hibernate **Session**, the persistent instance will become a **detached instance**.
- ⇨ Detached state means if we close or clear the session object will store in detsched state.
- ⇨ But if we call **session.update()** then again session is open and data is associated with session and persistent state.

27. Explain TRANSACTION MANAEMENT?

1) Transaction represent single unit of work in such case if one step fails the whole transaction fails.
2) Transaction is associate with session and instantiated by calling session.beginTransaction();
3) (This is described by ACID property)
4) Atomicity:-means either all successful or not(all operation done successfully)
5) Consistency:-insure bringing the database from one consistent state to another consistent state (reliable data).
6) Isolation:-ensure that transaction is isolated from another transaction.(separation)
7) Durability:-means once the transaction has been committed it will remain so event of errors
8) Transaction interface method:-
9) void begin() starts a new transaction.
10) void commit() ends the unit of work unless we are in FlushMode.NEVER.
11) void rollback() forces this transaction to rollback.
12) void setTimeout(int seconds) it sets a transaction timeout for any transaction started by a subsequent call to begin on this instance.
13) boolean isAlive() checks if the transaction is still alive.
14) void registerSynchronization(Synchronization s) registers a user synchronization callback for this transaction.
15) boolean wasCommited() checks if the transaction is commited successfully.
16) boolean wasRolledBack() checks if the transaction is rolledback successfully.

Session session = factory.openSession ();
Transaction transaction = session.beginTransaction ();
transaction. Commit ();

28. Explain Criteria builder?

⇨ *Hibernate* provides alternate ways of manipulating objects and in turn data available in RDBMS tables. One of the methods is *Criteria* API, which allows you to build up a *criteria query* object programmatically where you can apply filtration rules and logical conditions.
⇨ Entity Manager em = emf.createEntityManager ();
⇨ CriteriaBuilder cb=em.getCriteriaBuilder ();

- Criteria builder API and HQL both object oriented
- Both are used entity class name in state of table or column

29. How to achieve inheritance in hibernates?

⇨ Inheritance in spring is used when we data that comes in program repeatedly then we declare that field only once used it multiple times by giving parent child relation.

1) **By default**
2) **Single table**
3) **Join table** :- @inheritance(strategy=inheritanceType.joined)
4) **Table per class**:- @inheritance (strategy=inheritance Type. Table_per_class) In table per class is used parent class marge into child class entity

30. Used of cascade type?

⇨ Cascade is keyword is often (अनेकदा) appearing on the collection mapping to manage the state of the collection automatically

Method of cascade type:-

⇨ **CascadeType.PERSIST**:- Cascade type persist means that save () method or persist () operations cascade to related entities टिकून राहाणे,

⇨ **CascadeType.MERGE**:- Cascade type marge means that related entities are merged when the owning (मालकीचे) entity is merged.

⇨ **CascadeType.REFRESH**:- Cascade type refresh does the same thing for the refresh () operation

⇨ **CascadeType.REMOVE**:- Cascade type remove removes all related entities association with this setting when the owning entity is deleted.

⇨ **CascadeType.DETACH**:-cascade type detach detaches all related entities if a manual detach occurs. अलग-थलग

⇨ **CascadeType.ALL**:-cascade type all is shorthand for all of the above cascade operations.

31. How you have used Hibernate (XML, Annotations, Java Based)?

⇨ (Always Answer as) Annotation Based.

32. How to increment primary key auto generated?

- By using identity and auto increment
- Used GeneratedType.AUTO
- Used GeneratedType.IDENTITY
- Used GeneratedType.PRIMARY
- Used GeneratedType. SEQUENCE
- Used GeneratedType.All/persist/marge/remove/refresh/detach
- Used Annotation @generatedValue(strategy=GeneratedType.IDENTITY)
- MappedBy :- we cannot create the extra table of entity class
- If you used hibernate in spring mvc application so we used for configuration
  @bean
  Session factory:-

@bean
Datasource :-

@bean
Transaction manager

33. What is process for Automatic ID generation from any random number?

- @SequenceGenerator(name = "mySeqGen", sequenceName = "mySeq", initialValue = 500, allocationSize=1)
- @GeneratedValue(generator = "mySeqGen")

34. How to show SQL queries at run time?

- ⇨ While specifying hibernate properties, **add Show_SQL property as a true**,
- ⇨ e.g. <property name="show_sql">true</property>

35. Named Queries-Definition, syntax, advantages?

- ⇨ Instead of writing multiple case, HQL query or Native query we can write in a single place which is entity class.
- ⇨ Because of that HQL query and Native query will not be scattered.
- ⇨ It is easy to reuse and maintain.
- ⇨ There are Four Annotations for Named query :
  1. @NamedQuery.
  2. @NamedQueries.
  3. @NamedNativeQuery.
  4. @NamedNativeQueries.

36. What is Named SQL Query?

- ⇨ Named Query using which you can define at a **central location and use them anywhere** in the code.
- ⇨ You can **create named queries** for both **HQL** as well as for **Native SQL**.
- ⇨ These Named Queries can be defined in Hibernate **mapping files** with the help of **JPA annotations @NamedQuery and @NamedNativeQuery.**

37. Native Queries-Definition, syntax, advantages?

- ⇨ You can use native SQL to express database queries if you want to utilize database-specific features.
- ⇨ Hibernate 3.x allows you to specify handwritten SQL, including stored procedures, for all create, update, delete, and load operations.
- ⇨ Your application will create a native SQL query from the session with the createSQLQuery() method.
- ⇨ You need to pass a string containing the SQL query to the

createSQLQuery() method.

38. How do you create an immutable class in hibernate?

- ⇨ If we are using the **XML form** of configuration, then a class can be made immutable by **markingmutable=false.**
- ⇨ The default value is **true** there which **indicating** that the **class was not created by default**.
- ⇨ In the case of using **annotations**, immutable classes in hibernate can also be **created** by using **@Immutable annotation.**

39. What is meant by Hibernate tuning?

- ⇨ **Optimizing the performance** of Hibernate **applications** is known as Hibernate tuning.
- ⇨ The performance tuning **strategies** for Hibernate are:
  i. SQL Optimization
  ii. Session Management
  iii. Data Caching

40. What is the benefit of Native SQL query support in Hibernate?

- ⇨ Hibernate provides an option to execute Native SQL queries through the use of the ***SQLQuery*** object.
- ⇨ For normal scenarios, it is however not the recommended approach because you might lose other benefits like Association and Hibernate first-level caching.
- ⇨ Native SQL Query comes handy when you want to execute database-specific queries that are not supported by Hibernate API such query hints or the *Connect* keyword in Oracle Database.

41. Explain Hibernate configuration file and Hibernate mapping file?

- ⇨ **Hibernate configuration file:**
  It contains database specific configurations and is used to **initialize SessionFactory**.
  It provides database credentials or JNDI resource information in the hibernate configuration XML file.
  Dialect information is another important part of the hibernate configuration file.

- ⇨ **Hibernate Mapping file:**
  It is used to define the database table column **mappings and entity bean fields**.
  We use JPA annotations for mappings, but when we are using the third party classes sometimes XML mapping files becomes handy and we cannot use annotations.

POST ():-

This method is used to send some data to the server for example file update data etc. using HTML forms The used of post method to send a form data to the server is as follows

Put() :-

The put method is used to replace all the current representation of the target resource with the uploaded content.

Connect:-

This method establishes a tunnel to the server which is identifies by given URL This method is used by the client it establishes a network to a web server over HTTP. The below example request a connection with a web server which is running on the host

Option:-

This method describes the options of communication for the target resource

Delete ():-

This method request the server to delete a file at a location that is specified by the given URL

The below example request the server to delete the first HTML file at the root of the server

Spring Framework

1. How Spring Works?

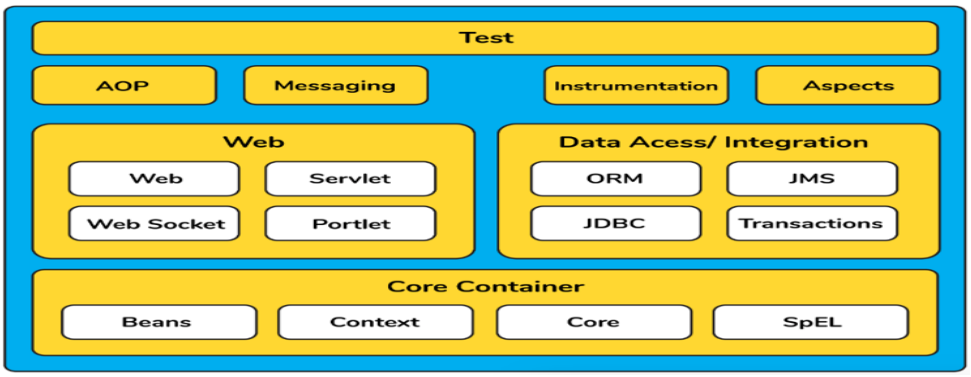
- ⇨ A web application (layered architecture) commonly includes three layers:
- ⇨ Presentation/view layer (UI) - This is the outermost layer which handles the presentation of content and interaction with the user.
- ⇨ Business logic layer - The central layer that deals with the logic of a program.
- ⇨ Data access layer - The deep layer that deals with data retrieval from sources.
- ⇨ Each layer is dependent on the other for an application to work. In other words, the presentation layer talks to the business logic layer, which talks to the data access layer. Dependency is what each layer needs to perform its function. A typical application has thousands of classes and many dependencies.
- ⇨ Without a Spring Framework, application code tends to be tightly coupled (interdependent), which is not considered good coding practice. Loose coupling is ideal because loosely coupled components are independent, meaning changes in one will not affect the operation of others.

2. Why do we use spring?

- ⇨ Spring is **Modularizes** framework (applicable for particular layer in application).
- ⇨ It is **light weight** (Its support pojo model).
- ⇨ It is **open source** (there is no need to purchase any license).
- ⇨ It is **complete end** (applicable for all layers in application).
- ⇨ We can achieve **loose coupling (One class is not depend on another)**.
- ⇨ **Non-invasive** framework (doesn't force to extend or implement any base class or interface).
- ⇨ We can **develop easy to test kind of applications**.
- ⇨ Spring framework is also called the **framework of frameworks**
- ⇨ It provides **support to various** other frameworks such as Struts, Hibernate, Tapestry, EJB, JSF etc.

❖ Advantage of Spring

- ⇨ Spring is non-invasive
- ⇨ Spring is light weight
- ⇨ Loose coupling (we cannot change in java project change in xml file)
- ⇨ Spring has its own container
- ⇨ Spring integrate another framework/tool easily.
- ⇨ It supports all layer of application (web base, desktop base)



3. **Which server Spring has by-default?**
   ⇨ Spring **doesn't have server**, we need to **add it externally**.

4. **In which scenario your application will work as egger loading?**
   ⇨ Egar loading is a **design pattern** in which **data initialization occurs** on the spot.
   ⇨ **For Singleton scope :-** Only one object for bean will be create.
   ⇨ **Application Context :-** eager-loading (ie object will created when xml file loads)

5. **List out the modules of spring?**
   ⇨ There is total 6 modules in spring-
   - Spring **core** module
   - Spring **application context** (J2EE) module
   - Spring **web MVC** module
   - Spring **AOP** (aspect oriented programing) module
   - Spring **ORM** (object relational mapping) module
   - Spring **JDBC** (DAO :- data access object) module

6. **What is the benefit and feature of spring framework?**

Benefit

   ⇨ Light weight, loose coupling, open source etc
   ⇨ It provides Container.
   ⇨ It provides IOC Container.

Feature

   ⇨ **Simplicity**:- It provide Loose coupling and Light weight.
   ⇨ **Testability :-** Easy to test.
   ⇨ **Light weight**

7. **What is the purpose of Container?**
   ⇨ It create **instance/object of bean as per request.**

8. **What is IOC in spring?**
   ⇨ IOC stands for **Inversion of Control**.
   ⇨ It is **heart** of spring framework.
   ⇨ It **converts object** from **tight coupling** to **loose coupling** with the help of Container**.**
   ⇨ **Tight** Coupling -> Change in **one class forces to change in other** (dependent) class.
   ⇨ **Loose** Coupling -> Change in **one class doesn't forces to change** in other (dependent) class.
   ⇨ IOC Container is a **framework** to **manage** automatic **dependency injection** throughout application.
   ⇨ There are two types of IOC
      1. Core container 2. J2EE Container

9. Type of container and its function?

- ⇨ Bean factory (Core) Container.
- ⇨ Application Context Container.
  - ❖ **Core Container :-** (interface)
  - ⇨ Bean factory container
  - ⇨ used to develop **desktop** (Core based application).
  - ⇨ **Resource** manual add.
  - ⇨ **No** any **annotation** features.

  - ❖ **J2EE Container:-** (interface)
  - ⇨ **Application** Context
  - ⇨ Used to develop enterprise/**web** base application.
  - ⇨ Automatic Integrate.
  - ⇨ We can **achieve Annotation** features.

  - ❖ **Function of Container: -**
  - ⇨ **Create instance** of pojo class.
  - ⇨ **Manage life cycle** of pojo class.
  - ⇨ **Dependency injection** of pojo class.

10. Difference between bean factory and application context?

Bean Factory (interface)	Application Context(interface)
It is **core** Container	It is **J2EE** Container
Is implements class is **xmlbeanfactory**	It implements class is I. ClassPathxmlApplicationContext II. FilesystymxmlApplicationContext III. AnnotationconfigApplicationContext IV. Xmlwaysapplicationcontext
It **doesn't** support **annotation**	Is **supports annotation**
Post resource should be add **manually**	Post resource should be add **automatically**
It supports **Lezzy** loading	It supports **Eager** loading
Resource rs=new classpathresousrce(spring configure file path); BeanFactory fb=new XmlBeanFactory(rs); In bean factory add external resource	ApplicationContext ap=new ClassPathXMLApplicationContext(spring configuration file path); In ApplicationContext not add external resource

11. What is Dependency and what is Dependency Injection?

- ⇨ **Data member or properties** are **dependency** and **assigning values** to those data memberis called as dependency injection.

12. What is Dependency Injection?

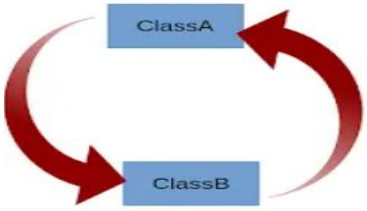
- ⇨ DI is **design pattern.**
- ⇨ In our application **data members** are nothing but DI.
- ⇨ We can **achieve two ways** of DI
  - Setter based DI
  - Constructor based DI
- ⇨ There are **3 types of DI**
  - Primitive DI
  - Secondary Type / Object Type DI
  - Collection Type DI.

13. What is default scope use in spring framework?

- ⇨ Singleton

14. What is Circular dependency in spring?

- ⇨ Circular dependency in Spring happens **when two or more beans require instance of each other** through constructor dependency injections.
- ⇨ For **example**: There is a Class-A that requires an instance of ClassB through constructor injection and Class-B requires an instance of class A through constructor injection.
- ⇨ By using setter injection we will solve the problem.
- ⇨ With **Setter injection Spring** sets properties and resolves dependencies as late as possible, when the bean is actually created thus avoiding exception "**BeanCurrentlyInCreationException**".



15. Difference between Setter based and Constructor based?

Setter Based DI	Constructor Based DI
It allows **partial** injection.	It does **not allow partial** injection.
It **overrides** constructor based DI.	It **doesn't** override setter based DI.
It is **mutable**.	It is **immutable**.
It **increases** line of code.	It **decreases** line of code.
It works better for **few** properties.	It works better for **many** properties.

16. What are Spring Beans?

⇨ It **manages** the container of POJO.
⇨ They are the **objects** that form the **backbone** of the **user's application**.
⇨ Beans are **managed** by the Spring IoC container.
⇨ They are **instantiated**, **configured**, **wired** and **managed** by a Spring IoC container
⇨ Beans are **created with the configuration metadata** that the users supply to the container.

17. What are bean scopes?

⇨ There are five bean scopes.

- **Singleton**:- Only same object for bean is created
- **Prototype** :- Produced new instance each and every time bean is requested
- **Request**:-This scope of the bean definition to an http request. only valid in the context of a web-aware spring application Context
- **Session**: - This scope of the bean definition to an http Session. only valid in the context of a web-aware spring application Context
- **Web socket**: - Single instance will be created and available during complete lifecycle of web socket.
- **Global-session:-** This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.
- **Application**:-a single instance will be created and available during complete lifecycle of servlet Context.

18. How to make Application Context Lazy loading?

⇨ By using scope as Prototype, we can make application context to act as a lazy loading.

19. What does autowire supports by-default?

⇨ By-default autowire supports **byType**.

20. What is Auto wiring?

- Auto wiring is a concept where there is **no need to inject secondary dependencies,** spring **container** will **inject** secondary dependencies on its **own**.
- Auto wiring **supports only secondary dependencies or Reference Type**.
- Auto wiring **can't be used to inject primitive and string values**.
- It works with **reference only**.
- By **default**, Auto wiring is **disabled**.

- If we **enable** Auto wiring spring container will **take care of injecting Secondary type dependencies.**

Type of Auto wiring achieved:-

- **Byname**:-
  - This byname mode **injects the object dependency according to name** of the **bean It internally calls setter method.**
  - In this case bean **id** and **reference** name **must be same**.
- **byType** :-
  - This byType mode **injects the object dependency according to type** of the bean
  - In this case **bean id and reference** name **may be same**.
  - Note: if multiple beans with same class are found ambiguity will occur. To resolve this add tag auto wire-candidate=”false**”** (org.springframework.beans.factorynotUniqueBeanDefinitionException).
- **Constructor**: -
  - The constructor mode **injects the dependency by calling the constructor** of the **class** it calls the constructors having **large number of parameters**.
  - Internally we used “**byType**” but while injecting property it uses constructor based injection.

21. Used of @Qualifier annotation?

- The @Qualifier annotation is used to **resolve the auto wiring conflict**, when there are multiple beans of same type.
- The @Qualifier annotation can be used on any **class annotated with @Component of on method annotated with @bean.**
- we can **eliminate the issue of which bean needs to be injected**.
- If it is @autowired annotation is not work properly to the application that time we can use @qualifier annotation.
- public class FooService {
  @Autowired
  @Qualifier("fooFormatter")
  private Formatter formatter; }

22. What is front controller?

- It is also known as **Dispatcher** Servlet.
- It **manages** entire process.
- It finds **appropriate class** as per request.

23. Can we inject null and empty string values in spring?

- ⇨ **Yes**
- ⇨ we can inject null and empty values. In XML configuration, null value is injected using `<null>` element.
- ⇨ Find the code snippet to inject null value.

```
⇨ <property name="a"><null/></property>
```

- ⇨ Find the code snippet to inject empty value.

```
⇨ <property name="b" value=""/>
```

24. How to enable Autowiring?

- ⇨ Add Spring context and <context:annotation-config /> in bean configuration file.
- ⇨ Include 'AutowiredAnnotationBeanPostProcessor' directly in bean configuration file.
  <bean class = "org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcess or"/>

25. What are Stereotype annotations? Explain them?

- ⇨ @**Controller**: Identifies class as a controller class & marks it as a bean.
- ⇨ @**Service**: Identifies class as a service class & tells it has **business** logic.
- ⇨ @**Repository**: Identifies class as a **Dao layer class** & tells it has a **database** connection.

26. Can we write @Repository for business logic class and @Service for dao logic class?

- ⇨ Yes we can shuffle @Repository & @Service.
- ⇨ It is just to understand other programmers to identify business logic class and dao layer class.
- ⇨ But we can't replace @Controller class.

27. What are annotations used in spring?

- ⇨ @RequestMapping:
  - It is used at class level or method level.
  - It is used to map URL.
- ⇨ @RequestParam:
  - It brings single variable from client side.
  - E.g. Username, Password get from client side.
- ⇨ @ModalAttribute:
  - It brings complete POJO class from client side.
  - E.g. Registration form data from client side.

28. In which scenario you applicable will work as eggar loading?

- ⇨ Egger loading is a design pattern in which data initialization occur on the spot.
- ⇨ For Singleton scope:- Only one object will bean will be created.
- ⇨ **Application Context** :- eggar loading (i.e object is created when cml file load).

29. What is server-side validations or spring validations?

- ⇨ Some annotations we can use as validations.
- ⇨ Annotations like-
  - @NotEmptyValidation
  - @SizeAnnotation
  - @EmailAnnotation
  - @NotNullAnnotation
  - @Valid => Data from client side using request, then this checks data is as per validations or not.

  - Singleton (same instance per IOC container)
  - Prototype (any number of instances per IOC container)
  - Request (Valid for Spring based applications, used for httprequest)
  - Session (Valid for Spring based applications, used for httpsession)
  - Global Session (Valid for Spring based applications, used for global httpsession)

30. What is lookup method?

- Its method level annotation. We same result excepted if we doesn't get same result then we should go with lookup method
- If Student bean scope is singleton and address class bean scope is prototype so Output is not correct, we cannot get expected so to get expected O/p we have to use Lookup Method.
- Example:- Student class and Address Has-A relationship.

	Singleton :- stu Singleton :- addr	Prototype :- stu Prototype :- addr	Prototype :- stu Singleton :- addr	Singleton :- **stu** Prototype :- addr
stu1	101	201	301	**401**
addr1	Pune	Karve-Nagar	Mumbai	**Amt**
stu2	101	202	302	**401**
addr2	Pune	Katraj	Mumbai	**Amt**
stu3	101	203	303	**401**
addr3	Pune	Deccan	Mumbai	**Amt**

- Because **student** is based **class** so it acts like singleton both in case of 4th column and row.
- For avoiding this problem we will use lookup() method.
- Only XML based.

31. Difference between Bean Factory Container & Application Context Container?

Bean Factory	Application Context
It is **core or legacy** container.	It is **J2EE** container, It extends **bean factory** & have additional advance properties.
It is used to develop **desktop** based applications.	It is used to develop **web based & desktop based** applications.
It is **lazy** loading.	It is **eager** loading.
It **doesn't** support **annotations**.	It **supports** annotations.

32. Difference Between spring boot and spring mvc

Spring boot	*Spring mvc*
It is module of spring for packaging the application	Spring mvc module view controller base framework
No need to build configuration manually(used DI)	It provided ready to used feature
development Descriptor is not required	Development Descriptor is required.
It reduces development time and increases productivity.	Take more time to active same

33. Difference between spring and Spring Boot?

Spring	*Spring Boot*
It used java j2ee framework for building the application	It used rest API for develop spring boot framework
Primary feature is dependency injection	Primary feature is auto wiring
Develop loosely couple application	Develop Standalone application with less configuration
Manually define dependency for the spring project format	Come with the concept of starter in pom.xml file that internally take care of download

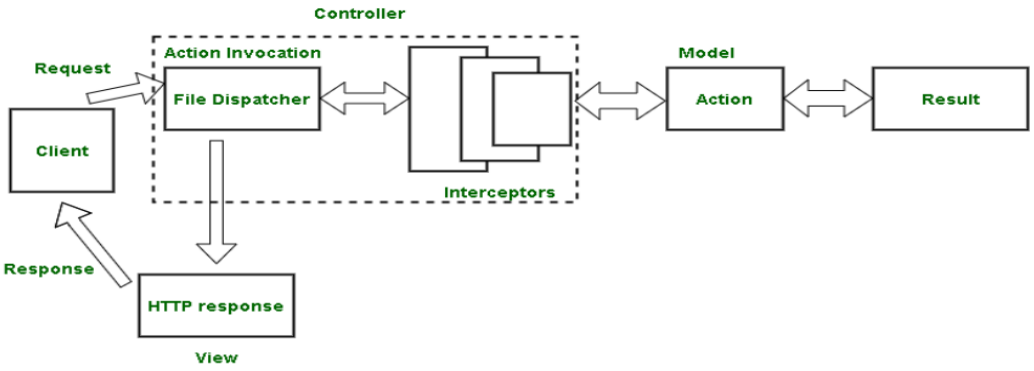
34. Difference between Spring and Struts architecture

Spring	**Struts**
It is a **lightweight** framework.	It is a heavyweight framework.
It does **not support tag library**.	It **supports** tag library **directive**.
It has **loosely** coupled modules.	It has **tightly** coupled programming modules.

It is integrated with ORM Technologies using which, lesser coding is required after and before the main logic.	It supports manual coding.
It has a **layered** MVC architecture containing 3 layers for **modelling, viewing and controller.**	It does **not** have a **layered architecture.**

35. What is Struts framework?

- ⇨ Struts is a framework **based on MVC architecture** that stands for model view and Controller architecture.
- ⇨ It is an **open-source** platform and is **used** to develop **enterprise edition web applications.**
- ⇨ It has a **request handler and response handler** because it is based on request-based Framework which handles the request from the user.
- ⇨ AJAX, REST and SOAP are supported by Struts.



36. What is the @Controller annotation used for?

- ⇨ The @Controller is a stereotype Spring MVC annotation to define a Controller.

37. Can you create a controller without using @Controller or @RestController annotations?

- ⇨ **Yes**! You can create a controller without @Controller or @RestController annotations by annotating the Spring MVC Controller classes using the **@Component annotation**. In this case, the real job of request mapping to handler method is done using the @RequestMapping annotation

38. What is MVC?

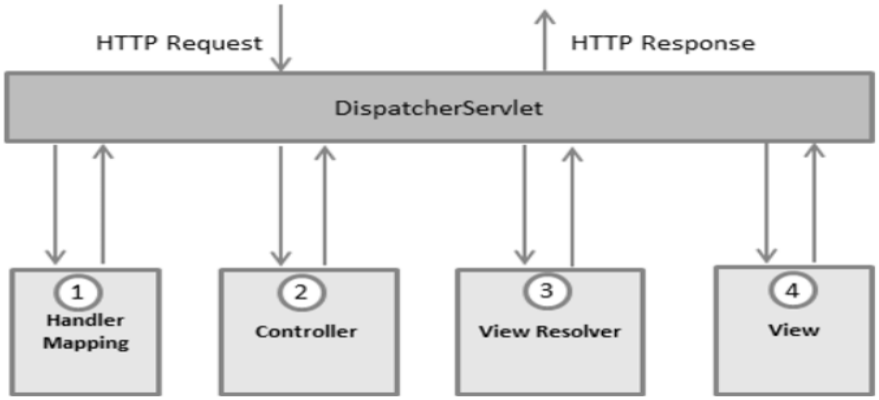
- ⇨ This **module of spring framework** helps us to use MVC in spring.
- ⇨ MVC pattern allows us to **separate the different aspect** of the application program i.e. **input logic, business logic, and UI** logic.
- ⇨ It also provide the **loose coupling between these aspects**.
- ⇨ **M: Model**: - It will bind the all data of application i.e. POJO
- ⇨ **V: View** (Html, Jsp): - It generates the HTML output with the help of model that the client's browser can interprete.

⇨ **C: Controller** (Servlet):- It processes the user request and build model passes it to view for processing.

39. What are the benefits of Spring MVC framework over other MVC frameworks?

- **Clear separation of roles** – There is a **specialized** dedicated object for every role.
- **Reusable business code logic** – With Spring MVC, there is **no need for duplicating** the code. Existing objects can be used as commands instead of replicating them in order to extend a particular framework base class.
- Spring MVC framework provides **customizable binding and validation**.
- Also provides **customizable locale and theme resolution**.
- Spring MVC supports customizable **handler mapping and view resolution** too.

40. Explain Spring MVC architecture?



⇨ Following is the sequence of events corresponding to an incoming HTTP request to Dispatcher Servlet
⇨ After receiving an HTTP request, Dispatcher Servlet consults the Handler Mapping to call the appropriate Controller.
⇨ The Controller takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and returns view name to the Dispatcher Servlet.
⇨ The Dispatcher Servlet will take help from View Resolver to pick up the defined view for the request.
⇨ Once view is finalized, The DispatcherServlet passes the model data to the view which is finally rendered on the browser.

41. What is an InternalResourceViewResolver in Spring MVC?

⇨ The **InternalResourceViewResolver** is a class which is used to resolve internal view in Spring MVC.
⇨ Here, you can define the properties like prefix and suffix where prefix contains the location of view page and suffix contains the extension of view page.
⇨ For example:-

```
1. <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalReso
   urceViewResolver">
2.       <property name="prefix" value="/WEB-INF/jsp/"></property>
3.       <property name="suffix" value=".jsp"></property>
4.    </bean>
```

42. How to declare a class as a controller class in Spring MVC?

- ⇨ The @Controller annotation is used to declare a class as a controller class.
- ⇨ It is required to specify this annotation on the class name.
- ⇨ For example:-

```
@Controller
class Demo
{

}
```

43. How to map controller class and its methods with URL?

- ⇨ The **@RequestMapping** annotation is **used to map the controller class and its methods**.
- ⇨ You can specify this annotation on the class name as well as method name with a particular URL that represents the path of the requested page.
- ⇨ For example:-

```
@Controller
@RequestMapping("/ form")
class Demo
{
@RequestMapping("/show")
public String display()
{ }
}
```

44. Name the annotations used to handle different types of incoming HTTP request methods?

- ⇨ The following annotations are used to handle different types of incoming HTTP request methods: -
- ⇨ @GetMapping
- ⇨ @PostMapping
- ⇨ @PutMapping
- ⇨ @PatchMapping
- ⇨ @DeleteMapping

45. What is the purpose of @PathVariable annotation in Spring MVC?

- ⇨ The @PathVariable annotation is **used to extract the value of the URI template**.
- ⇨ It is passed within the parameters of the **handler** method.
- ⇨ For example :-
- ⇨ @RequestMapping("/show/{id}")
- ⇨ public String handler(@PathVariable("id") String s, Model map)
- ⇨ {
- ⇨ }

46. What is the role of @ResponseBody annotation in Spring MVC?

- ⇨ The @ResponseBody annotation is **used to serialize the returned object automatically in JSON** and bind it with the Http response body.
- ⇨ Here, it not required to invoke the model.
- ⇨ For example :-

```
@RequestMapping("/show")
@ResponseBody
public ResponseHandler display(@RequestBody ShowForm form)
 {
   return new ResponseHandler("display form");
  }
```

47. What is the role of the Model interface in Spring MVC?

- ⇨ The **Model** interface works as a **container that contains the data of the application**.
- ⇨ Here, data can be in any form such as objects, strings, information from the database, etc.

48. What do you mean by ModelAndView in Spring MVC?

- ⇨ The **ModelAndView** is a **class** that **holds both Model and View**
- ⇨ Where the model represents the **data**, and view represents the **representation** of that data.
- ⇨ This class returns the model and view in the **single return value.**

49. What is ModelMap in Spring MVC?

- ⇨ The **ModelMap** is a class that provides the implementation of Map. It extends the LinkedHashMap class. It facilitates to pass a collection of values as if they were within a Map.

50. What are the ways of reading data from the form in Spring MVC?

- The following ways to read the data from the form are: -
- **HttpServletRequest interface** - The HttpServletRequest is a java interface present in javax.servlet.http package. Like Servlets, you can use HttpServletRequest in Spring to read the HTML form data provided by the user.
- **@RequestParam annotation** - The @RequestParam annotation **reads the form data** and **binds it automatically** to the parameter present in the provided method.

- **@ModelAttribute annotation** - The @ModelAttribute **annotation binds** a method parameter or its **return** value to a **named model attribute**.

51. What are the types of stereotype annotations? Explain?

- ⇨ **@Controller: -**
  - Identifies class as a controller class and **marks it as a bean**.
- ⇨ **@Service: -**
  - Identifies class as a Service class and tells it has **business logic**.
- ⇨ **@Repository: -**
  - Identifies class as a Dao layer class and tells it has a **database connection.**
- ⇨ **@Component: -**
  - This annotation is responsible for **converting** a **java class to the bean** so that it can be recognized by spring and used in the application context.

52. What is the purpose of @Request annotation?

- ⇨ One of the most important annotations in spring is the @RequestMapping Annotation which is **used to map HTTP requests to handler methods of MVC and REST controllers**.
- ⇨ To configure the mapping web request, we write

@RequestMapping(maapng between the rest and handler method).

- ⇨ In Spring MVC applications, the **DispatcherServlet** (Front Controller) is responsible for **routing incoming HTTP requests to handler methods of controllers.**
- ⇨ Its applicable for class level as well as method level in the controller class.

53. How we can integrate with another framework such as hibernate?
Where will configure to integrate hibernate framework?

- ⇨ if we are going to integrate the hibernate application with spring, we **don't need to create the hibernate.cfg.xml file**.
- ⇨ We can provide all the information in the **applicationContext.xml file**.

54. What is @ModelAttribute used for?

- ⇨ @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.

55. What is the difference between @PathVeriable and @RequestParam?

@PathVeriable	**@RequestParam**
We can send data through URI.	We can send data through URL.
It is from spring framework.	It is form JAX-RS
It will work in spring MVC and REST	It will work only REST.
Its dynamic or runtime URI.	Its Static

56. What is the difference between @Controller and @RestController?

@Controller	@RestController
@Controller is used to mark classes as Spring **MVC** Controller.	@RestController annotation is a special controller used in **RESTful Web services**, and it's the combination of @Controller and @ResponseBody annotation.
It is a **specialized** version of **@Component annotation.**	It is a specialized version of **@Controller annotation.**
In @Controller, we can **return** a **view** in Spring Web MVC.	In @RestController, we **cannot** return a view.
@Controller annotation indicates that the class is a "controller" like a web controller.	@RestController annotation indicates that class is a controller where @RequestMapping methods assume @ResponseBody semantics by default.
In @Controller, **we need to use @ResponseBody** on every handler method.	In @RestController, **we don't need to use @ResponseBody** on every handler method.
It was added to Spring **2.5 version**.	It was added to Spring **4.0 version**.

57. What is difference between @Service @Repository @Controller?

@Service Annotation	@Repository Annotation	@Controller Annotation
@Service annotation is used with classes that provide some business functionalities.	@Repository Annotation is used to indicate that the class provides the mechanism for storage, retrieval, update, delete and search operation on objects.	@Controller annotation indicates that a particular class serves the role of a controller.
@Service Annotation is a specialization of @Component Annotation.	@Repository Annotation is also a specialization of @Component Annotation.	@Controller annotation is also a specialization of @Component annotation.
It is used to mark the class as a service provider.	It is used to mark the interface as DAO (Data Access Object) provider.	It's used to mark a class as a web request handler.

It is a stereotype for the service layer.	It is also a stereotype for the DAO layer.	It is also a stereotype for the presentation layer (spring-MVC).
Switch can be possible. But it is not recommended.	Switch can be possible. But it is not recommended.	We cannot switch this annotation with any other like @Service or @Repository.
It is a Stereotype Annotation.	It is also a Stereotype Annotation.	It is also a Stereotype Annotation.

58. Spring MVC Crud Operation?

```
@Controller
public class HomeController
{
@Autowired
HomeService hs;

@RequestMapping("/")
public String prelogin()
{
System.out.println("login page called");
return "login";
}

@RequestMapping("/regi")
public String preRegister()
{
return "registation";
}

@RequestMapping("/reg")
public String regdata(@ModelAttribute Student s)
{
hs.savedata(s);
System.out.println("Hello Reg Data");
return "login";
}

@RequestMapping("/login")
public String loginCheck(@RequestParam("uname") String uname,
@RequestParam("password") String pass, Model m)
{
int id = hs.loginCheck(uname, pass);
```

```
List<Student> slist = hs.displayAll();

if(id>0) {
m.addAttribute("data", slist);
return "success";
}
else { return "login"; }
}

@RequestMapping("/delete")
public String deleteData(@RequestParam("uid") int uid, Model m)
{
hs.deleteData(uid);
List<Student> slist = hs.displayAll();
m.addAttribute("data", slist);
return "success";
}

@RequestMapping("/edit")
public String editdata(@RequestParam("uid") int uid,Model m)
{
Student st=hs.editpage(uid);
m.addAttribute("data",st);
return "edit";
}

@RequestMapping("/update")
public String updateData(@ModelAttribute Student s, Model m)
{
hs.updatedata(s);
List<Student> slist = hs.displayAll();
m.addAttribute("data", slist);
return "success";
}
}

@Repository
public class HomeDaoImpl implements HomeDao{

@Autowired
private SessionFactory sf;

@Override
public void savedata(Student s)
{
```

```
Session ss=sf.openSession();
Transaction tx=ss.beginTransaction();
ss.save(s);
tx.commit();
}

@Override
public int loginCheck(String uname, String pass)
{
Session ss=sf.openSession();
Query query = ss.createQuery("from Student where uname='"+uname+"'
and password='"+pass+"'");
Student stu = (Student) query.getSingleResult();

if(stu!=null) { return 1; }
else { return 0; }
}

@Override
public List<Student> displayAll()
{
Session ss=sf.openSession();
Query query = ss.createQuery("from Student");
List<Student> list = query.getResultList();

return list;
}

@Override
public void deleteData(int uid)
{
Session ss= sf.openSession();
Query q= ss.createQuery("delete from Student where uid = "+uid+" ");

Transaction tx = ss.beginTransaction();
int i = q.executeUpdate();
System.out.println(i);
tx.commit();
ss.close();
}

@Override
public void updatedata(Student s)
{
Session ss=sf.openSession();
```

```
Query query = ss.createQuery("update Student set
name='"+s.getName()+"',
uname='"+s.getUname()+"',password='"+s.getPassword()+"',mobileno='"
+s.getMobileno()+"' where uid="+s.getUid()+"");

Transaction tx=ss.beginTransaction();

int i = query.executeUpdate();

tx.commit();
ss.close();
}

@Override
public Student editpage(int uid)
{
Session ss=sf.openSession();
Query query=ss.createQuery("from Student where uid="+uid+"");
Student st = (Student)query.getSingleResult();
return st;
}
}
```

Springboot

1. **What is the SpringBoot?**
   - ⇨ Spring boot is not a framework it provide add-on feature of spring application.
   - ⇨ Spring Boot make it easy to create **standalone application** we just run spring boot application.
   - ⇨ Spring boot application are **light weight** in term of code and configuration.
   - ⇨ It is provided **autoconfiguration** feature.
   - ⇨ SpringBoot is **not spring framework**.
   - ⇨ Zero configuration.

2. **What are the features of springboot?**
   - ⇨ Create Standalone application (both web and desktop application).
   - ⇨ It provides embed tomcat server.
   - ⇨ Starter Dependency: - with the help of this feature, springboot aggregates common dependencies together and eventually improves productivity.
   - ⇨ Spring Initializer: - which can create an internal project structure for you.
   - ⇨ Logging and security.
   - ⇨ SpringBoot automatically configuration.

3. **How many ways to created Springboot application?**
   - ⇨ Using Spring Initializer (https://start.spring.io/)
   - ⇨ Using Spring STS (Spring Tool Suite)
   - ⇨ Using Springboot CLI
   - ⇨ Using Spring IDE (Eclipse)

4. **Is the parent class download any dependency?**
   - ⇨ **No**.
   - ⇨ Parent tag will **take care** our application and **maintain** our application.

5. **Why to use springboot rather than spring MVC?**
   - ⇨ Springboot has embedded Tomcat server, **no need to add external server**.
   - ⇨ It has starter form **Maven Dependency**. (Web Starter, JPA Starter, etc.)
   - ⇨ It **automatically configure** spring application by using @Autoconfigure based on the dependencies.
   - ⇨ It has annotations that's why it **skips XML configurations**.

6. **How does springboot application works?**
   - ⇨ Create **new project** of springboot application.
   - ⇨ First specify dependency starter in **pom.xml**
   - ⇨ Then **dependencies** are **added** to project.
   - ⇨ Executes program from **main ()** where auto configuration done.
   - ⇨ Executes **SpringApplication.run()** that launches program.

7. **What does @SpringBootApplication do?**
   - ⇨ It marks class as a bean.
   - ⇨ It has 3 annotations working inbuilt are @Configuration, @ComponentScan and @EnableAutoConfiguration.
   - ⇨ @Configuration = It tells that class is spring bean.
   - ⇨ @EnableAutoConfiguration = It configures application as per jars added.
   - ⇨ @ComponentScan = Scans other packages in application.

8. **How to disable the autoconfiguration?**
   - ⇨ @EnableAutoConfiguration

9. **SpringApplication.run () statement does?**
   - ⇨ It bootstraps and launches the application.

10. **What is Actuator?**
    - ⇨ Spring boot provided actuator to **monitor and manage** our application when aur application is on production environment.
    - ⇨ Actuator is a tool has HTTP endpoint.
    - ⇨ Actuator provides some end point such as **Health, info, Environment.**
      - **Heath: -** It will check health **status** of aur application.
      - **Info: -** It display information about aur application.
      - **Environment: -** It display current environment property.
    - ⇨ For implementing actuator concept, we need to add one dependency is **SpringBootActuator.**

11. **What is spring boot dependency management?**

⇨ Spring Boot manages dependencies and configuration **automatically**.
⇨ Each release of Spring Boot provides a list of dependencies that it supports.
⇨ The list of dependencies is available as a part of the **Bills of Materials (spring-boot-dependencies)** that can be used with **Maven**.
⇨ So, we **need not to specify the version of the dependencies** in our configuration.
⇨ **Spring Boot manages itself**.
⇨ Spring Boot upgrades all dependencies automatically in a consistent way when we update the Spring Boot version.

12. **What is the use of Devtool?**
    - ⇨ It provides **fast** application **restart**, **live Reloading** and configuration for enhanced development experience.
    - ⇨ Live reload server is running on port 35729.

13. **What is spring data JPA use in our project?**
    - ⇨ Pom.xml add dependency
    - ⇨ Interfaces

14. What is spring profile?

- ⇨ Spring profile helps **segregating** your application, **configuration** and make them available only in certain **environment**.
- ⇨ Each **environment** has its own **specific requirement** i.e. database connection, pipeline, and deployment configuration.
- ⇨ In spring boot we can use **spring.profile.active** in aur application.properties file.
- ⇨ In spring we maintain profiles with the help of properties and file files.
- ⇨ Spring boot profiles group parts of the application configuration and make it be available only in certain environment.

15. Difference between Spring and Spring Boot?

Spring	Spring Boot
Spring Framework is a widely used Java EE framework for building applications.	Spring Boot Framework is widely used to develop REST APIs.
It aims to simplify Java EE development that makes developers more productive.	It aims to shorten the code length and provide the easiest way to develop Web Applications.
The primary feature of the Spring Framework is dependency injection.	The primary feature of Spring Boot is Autoconfiguration. It automatically configures the classes based on the requirement.
It helps to make things simpler by allowing us to develop loosely coupled applications.	It helps to create a stand-alone application with less configuration.
The developer writes a lot of code (boilerplate code) to do the minimal task.	It reduces boilerplate code.
To test the Spring project, we need to set up the sever explicitly.	Spring Boot offers embedded server such as Jetty and Tomcat, etc.
It does not provide support for an in-memory database.	It offers several plugins for working with an embedded and in-memory database such as H2.
Developers manually define dependencies for the Spring project in pom.xml.	Spring Boot comes with the concept of starter in pom.xml file that internally takes care of downloading the dependencies JARs based on Spring Boot Requirement.

16. Difference between Spring MVC and Spring Boot?

Spring Boot	Spring MVC
Spring Boot is a module of Spring for packaging the Spring-based application with sensible defaults.	Spring MVC is a model view controller-based web framework under the Spring framework.
It provides default configurations to build Spring-powered framework.	It provides ready to use features for building a web application.
There is no need to build configuration manually.	It requires build configuration manually.
There is no requirement for a deployment descriptor.	A Deployment descriptor is required.
It avoids boilerplate code and wraps dependencies together in a single unit.	It specifies each dependency separately.
It reduces development time and increases productivity.	It takes more time to achieve the same.

17. How to manage @transation annotation?

- ⇨ Transient annotation use when we wont any field in pojo which is not create a field in table.

18. How do we monitor & manages our application?

- ⇨ To get production ready feature we use springboot **actuator**.
- ⇨ It has **inbuilt HTTP** endpoints.
- ⇨ We can create our endpoints.
- ⇨ Enable this using springboot actuator.
- ⇨ E.g. of endpoints /bean (name of beans in application), /health (up or down of application).

19. What is springboot profile (@profile)?

- ⇨ It provide ways to divide application to spare them for particular environment.
- ⇨ We specify @Profile to tell code is written for particular environment.
- ⇨ Environments like Development environment, Production environment, Testing environment, etc.
- ⇨ Each environment has different configurations.

20. What is spring data JPA?

- ⇨ It has 3 repository interfaces in that-
  - CrudRepository (It is parent interface, used for CRUD operations)
  - PagingAndSortingRepository (It extends CrudRepository, used for Paging & Sorting purpose)
  - JPARepository (It extends PagingAndSortingRepository, used for all above and its additional services)

21. How to create custom queries?

- ⇨ Using @Query annotation.
- ⇨ E.g. @Query("select c from City c where c.name like %?1") List<City> findByNameEndsWith(String chars);

22. How did you manage session in your project?

- ⇨ We have used spring securities.
- ⇨ But I haven't implemented it, my senior has implemented it.

23. How to configure Hibernate in springboot?

- ⇨ Add JPA & MySQL database dependency.
- ⇨ In application.properties, write hibernate configurations to connect database.

24. How application.properties file added to container?

- ⇨ @EnableAutoConfiguration annotation is responsible for connection between database& application while configuring hibernate by adding application.properties file to container from its fixed path or specified path.

25. Spring Boot Crud Operation?

```
@Controller
public class HomeController {
	@Autowired
	HomeService hs;

	@RequestMapping("/")
	public String prelogin() {
		System.out.println("Pre Login Method Call");
		return "login";
	}

	@RequestMapping("/regi")
	public String preRegister()
	{
		System.out.println("Pre Register Method Call");

		return "registation";
	}

	@RequestMapping("/register")
	public String registerPage(@ModelAttribute Student s)
	{
		System.out.println("Return Login Call");

		hs.savedata(s);
```

```
		return "login";
	}

	@RequestMapping("/log")
	public String logincheck(@RequestParam("uname") String uname,
@RequestParam("pass") String pass, Model m)
	{
		hs.logincheck(uname, pass);

		Iterable<Student> list = hs.displayAllData();

		m.addAttribute("data", list);

		return "success";
	}

	@RequestMapping("/delete")
	public String deletePage(@ModelAttribute Student s,Model m)
	{
		hs.deleteData(s);

		Iterable<Student> list = hs.displayAllData();

		m.addAttribute("data", list);

		return "success";
	}

	@RequestMapping("/edit")
	public String editPage(@RequestParam("id") int id , Model m)
	{
		Student s = hs.editData(id);

		m.addAttribute("data", s);

		return "edit";
	}

	@RequestMapping("/update")
	public String updatePage(@ModelAttribute Student s , Model m)
	{
		hs.savedata(s);

		Iterable<Student> list = hs.displayAllData();
```

```
			m.addAttribute("data", list);
			return "success";
		}
}

@Service
public class HomeServiceImpl implements HomeService
 {
		@Autowired
		HomeRepository hr;

		@Override
		public void savedata(Student s)
		{
			hr.save(s);
		}

		@Override
		public void logincheck(String uname, String pass)
		{
			hr.findAllByUnameAndPass(uname, pass);
		}

		@Override
		public Iterable<Student> displayAllData()
		{
			return hr.findAll();
		}

		@Override
		public void deleteData(Student s)
		{
			hr.delete(s);
		}

		@Override
		public Student editData(int id)
		{
			return hr.findAllById(id);
		}
}
```

RESTfull Web Service

1. **What is web service?**
   - ⇨ It is medium or way of communicating application over the network.
   - ⇨ It is a client-server application or application component for communication.
   - ⇨ The method of communication between two devices over the network.
   - ⇨ It is a software system for the interoperable machine to machine communication.
   - ⇨ It is a collection of standards or protocols for exchanging information between two devices or application.

2. **What are the advantages of web service?**
   - ⇨ Interoperability: Web services are accessible over network and runs on HTTP/SOAP protocol and uses XML/JSON to transport data, hence it can be developed in any programming language. Web service can be written in java programming and client can be PHP and vice versa.
   - ⇨ Reusability: One web service can be used by many client applications at the same time.
   - ⇨ Loose Coupling: Web services client code is totally independent with server code, so we have achieved loose coupling in our application.
   - ⇨ Easy to deploy and integrate, just like web applications.
   - ⇨ Multiple service versions can be running at same time.

3. **What are the types of web service and difference between them?**
   - ⇨ SOAP (Simple object Access Protocol) & REST (Representational State transfer) are the types of web services.

SOAP	REST
It supports XML data format only.	It supports XML, HTML, Plain text, JSON, etc. data formats.
It is protocol based.	It is architecture based.
SOAP cannot use REST.	REST can use SOAP.
It has its own security.	Have to add security.
It is less preferred.	It is more preferred.
Need JAX-WS API to implement SOAP.	Need JAX-RS API to implement REST.

4. **What are the ways to implement Restful web service? Which one you have used?**

   In JAX-RS, there are 3 types to implement Restful web service.
   - Jersey.
   - REST easy.
   - Spring with REST.
   - ⇨ We have used spring with REST type.

5. **By using spring with REST, how do we create Restful web service?**

- ⇨ Add REST dependency in pom.xml. (spring-boot-starter-data-rest)
- ⇨ Add @RestController annotation for controller class.
- ⇨ Map the methods written in controller class.

6. What is REST? Why do you choose Restful web service?

- ⇨ REST is the acronym for REpresentational State Transfer. REST is an architectural style for developing applications that can be accessed over the network.
- ⇨ REST is a stateless client-server architecture where web services are resources and can be identified by their URIs (Uniform Resource Identifiers).
- ⇨ We choose Restful web service because-
  - It is architectural style not protocol.
  - It is fast and has no restrictions like SOAP.
  - It uses data formats like XML, HTML, Plain Text, JSON, etc.
  - It also can use SOAP.
  - It is programming language and platform (OS) independent.

7. Annotations used in Restful web service?

- ⇨ @RestController
- ⇨ @GetMapping
- ⇨ @PostMapping
- ⇨ @DeleteMapping
- ⇨ @PutMapping
- ⇨ @PathVariable
- ⇨ @RequestBody
- ⇨ @ResponseBody

8. Difference between @Controller & @RestController?

@Controller	@RestController
It is used to mark a class as Spring MVC Controller.	It is used in RESTFul web services and the equivalent of @Controller + @ResponseBody.
It returns view.	It returns data.
Added in Spring 2.5 version.	Added in Spring 4.0 version.

9. Difference between @PathVariable & @PathParam?

@PathVariable	@PathParam
It is from spring framework.	It is from JAX-RS.
It will work in spring MVC and REST	It will work in REST only.
It is annotation on a method argument to bind it to the value of a URI template variable.	It is a parameter annotation which allows you to map variable URI path fragments into your method

	call.

10. **Have you Consumed or Produced Restful web service? Explain?**
    - ⇨ Consume or Produce in Restful web service means specifying which type of data to be requested (it is just telling type of data accepted or send through application)

E.g. While consuming data type we use syntax, consumes="application/JSON", it specify that certain service accepts only JSON data format.

- ⇨ Produce is associated with @GetMapping annotation where we mention type of data while we are sending it when that method/service is called.
- ⇨ Consume is associated with @PostMapping annotation where we mention type of data is to be accepted for saving it into the database when that method/service is called.
- ⇨ E.g. Film Producer makes (Produce) Movies & Viewers watch (Consume) those movies.

11. **How to consume and produce data using xml format?**
    - ⇨ First use @XmlRootElement on POJO class or need to add Xml data supporting dependency.
    - ⇨ Then in Controller methods specify,
        - (For Produce) produces="application/xml"
        - (For Consume) consumes="application/xml"

12. **In Restful web service, how to convert JSON format data to Object format or vice versa?**
    - ⇨ Object to JSON format:
        - ObjectMapper mapper=new ObjectMapper( );
        - String jsonResult=mapper.writeValueAsString(object);
    - ⇨ JSON to Object format:
        - ObjectMapper mapper=new ObjectMapper( );
        - Student s=mapper.readValue (jsonResult, Student.class);

13. **How to manage session in Restful web service?**
    - ⇨ We can't manage session in Restful web service.

14. **Can we manage session in Restful web service?**
    - ⇨ We can't manage session in Restful web service.
    - ⇨ Because Restful web services are stateless means for every request it takes it as a new request.

15. **How do you apply or implement security in web service?**
    - ⇨ By using spring security we have achieved security in web services.
    - ⇨ But I have not implemented it, my senior has implemented it.

16. **Difference between API & web services?**

API	Web Services
It is an interface between two different applications.	It is interaction between applications over the network.
All APIs are not web services.	All web services are APIs.
API can be used for any style of communication.	Web service uses styles like REST, SOAP for communication.
It can be used by a client who understands JSON or XML.	It can be used by any client who understands XML.
API has a light-weight architecture.	Web Services does not have a light-weight architecture.

17. **How to test web service?**
    - ⇨ Using ARC & POSTMAN.
    - ⇨ Locally –
        - Create war file.
        - Deploy on local Tomcat server.

18. **How do you mock the web service?**
    - ⇨ Mocking Web service is testing application using dummy data.

    We haven't done it.

19. **What are HTTP methods?**
    - ⇨ GET, POST, DELETE, PUT, etc.
    - ⇨ GET - The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.
    - ⇨ POST - The POST method is used to submit an entity to the specified resource, often causing a change in state.
    - ⇨ PUT - The PUT method replaces all current representations of the target resource with the request payload.
    - ⇨ DELETE - The DELETE method deletes the specified resource.
    - ⇨ PATCH - The PATCH method is used to apply partial modifications to a resource.

20. **What are components in HTTP?**
    - ⇨ Request, Body, Header, etc

Design Pattern

1. **Explain with example Factory Method Design Pattern?**
   - ⇨ It is used if a java class is having private constructor and outside of that class object creation of that class is not possible.
   - ⇨ Factory method is java method, having a capability of constructing and returning its own java class object.
   - ⇨ E.g. Example for pre defined instance factory method is:
     String s1=”welcome to durga software solutions”; // Existing Object String s2=s1.substring(3,7); // New Object i.e. it gives come
   - ⇨ E.g. (Sample Code) class

```
Test
{
        int x;
//private zero argument constructor
        private Test()
        {
          System.out.println("Test: 0-arg Con");
        }
//static factory method
        public static Test staticFactoryMethod()
        {
         Test t=new Test();t.x=5;
          return t;
        }
//instance factory method
        public Test instanceFactoryMethod()
        {
         Test t=new Test();
         t.x=this.x+5; return t;
        }
        public String toString()
        {
         return "x="+x;
        }
    }

    public class FactoryEx
    {
        public static void main(String[] args) throws Exception
        {
         Test t1=Test.staticFactoryMethod();
         System.out.println(t1); // which internally calls toString()
```

```
    Test t2=t1.instanceFactoryMethod();
    System.out.println(t2);
  }}
```

2. **Explain with example Singleton Class Design Pattern?**
   - ⇨ Creating multiple objects for java class with same data is wastage of memory.
   - ⇨ So we can use Singleton java class, which says create only one object for java class and use it for multiple times on each JVM in order to minimize the memory wastage and to increase the performance.
   - ⇨ Singleton java class is java class, which allows us to create only one object per JVM.
   - ⇨ Sometimes it's important to have only one instance for a class.
   - ⇨ E.g. Most of the JDBC driver classes are given as singleton java classes for better utilization.
   - ⇨ E.g. (Sample Code)

```
class Demo
{
        private static Demo instance;
        private Demo()
        {
          System.out.println("Demo : 0-arg Con");
        }
//static factory method
        public static Demo getInstance()
        {
// Singleton Logic
          if (instance == null) instance
          = new Demo();return
          instance;
        }
//override Object class clone()
        public Object clone()
        {
          return instance;
        }
}
```

3. **How to break Singleton Class?**
   - ⇨ To Break Singleton Using Reflection

```
import java.lang.reflect.Constructor;

public class Test {
  public static void main(String[] args) throws Exception
    {Demo s = Demo.getInstance();
    Class clazz = Demo.class;
```

```
    Constructor cons =
    clazz.getDeclaredConstructor();
    cons.setAccessible(true);
    Demo s2 = (Demo) cons.newInstance();
  }
}
```

- ⇨ The singleton will only work per classLoader. To break it, use multiple classLoaders.
- ⇨ One more way is to Serialize the Object store it in a file and Deserialize it to get a new Object.

4. **What is Class Loader?**
   - ⇨ The Java ClassLoader is a part of the Java Runtime Environment that dynamically loads Java classes into the Java Virtual Machine. The Java run time system does not need to know about files and file systems because of classloaders.
   - ⇨ Java classes aren't loaded into memory all at once, but when required by an application. At this point, the Java ClassLoader is called by the JRE and these ClassLoaders load classes into memory dynamically.

5. **What is real time example of Singleton Class Design Pattern?**
   - ⇨ Music Player Software, It has class contains services like Play the song, Stop the Song, Change the song, etc. Single instance of this class can be used for playing multiple songs in this application.

Serialization

1. **What is Serialization?**
   - ⇨ The process of saving (or) writing state of an object to a file is called serialization.
   - ⇨ But strictly speaking it is the process of converting an object from java supported form to either network supported form (or) file supported form.
   - ⇨ By using FileOutputStream and ObjectOutputStream classes we can achieve serialization process.

2. Have you used Serialization in your project?
   -

3. **What is real time technical example of Serialization?**
   - ⇨ Flipkart/Amazon application, all buying section contains images of products, that are stored in binary format in their database using serialization technique. When client download them from their website then these

images are deserialised into client's device.

4. **How to serialize particular states of objects of class?**
   - ⇨ By using Externalization process.
   - ⇨ To provide Externalizable ability for any object compulsory the corresponding class should implements externalizable interface.
   - ⇨ Externalizable interface is child interface of serializable interface.

Agile

1. **Do you know Agile? Have you implemented in your project?**
   - ⇨ The Agile Method and methodology is a particular approach to project management that is utilized in software development, it is iterative approach to software delivery that builds software incrementally from the start of the project, instead of trying to deliver it all at once near the end.
   - ⇨ We have not implemented 100 % agile, we have followed agile.
2. **What is Sprint?**
   - ⇨ It is of maximum 2 weeks or minimum 2 days.
   - ⇨ It is time duration, in which decided work is completed.
3. **What is Story?**
   - ⇨ Whichever task is assigned is called as Story.
4. **Who is Scrum Master?**
   - ⇨ Manager who carry all agile process from team.
   - ⇨ In our case team manager was Scrum Master as we can't afford resource for that as we were small scale industry.
5. **Who is Product Owner?**
   - ⇨ It is person who interacts with client.
   - ⇨ Gets requirements from client and convert it into stories in document format. So that it could be assigned to developers.
6. **What is your daily routine?**
   - ⇨ On every start of week, at start time of office at 9 AM, daily stand up or scrum meeting was held approximately 15 min or can be extended to 30 min.
   - ⇨ Scrum master manages this meeting and assigns stories to developers.
   - ⇨ In this meeting, road blocks or problems occurred in stories are discussed and resolved.
7. **What is Grooming Session?**
   - ⇨ It held by Product Owner.
   - ⇨ Also upcoming stories are explained to developers.
8. **What is Retrospective Call?**
   - ⇨ It happens at Sprint end.
   - ⇨ All issues that stopped working of stories are discussed and taken care of not repeating them in next sprint.
   - ⇨ Kudos & Thanksgiving done.

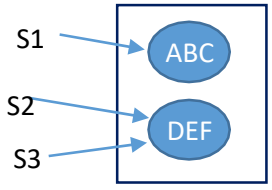
9. **What is Demo Call?**
   - ⇨ It held at the end of sprint.
   - ⇨ Its call with client to show demo of story completed.
   - ⇨ Client suggestions are added in next sprint stories by product Owner.
10. **How did you gather requirements from Client?**
   - ⇨ We had resource as a product owner, for gathering requirement who also divided it into stories to assign it to developer.
11. **How did Stories assign to you?**
   - ⇨ We used to receive mails from product owner with stories attached in document format.
   - ⇨ In big scale companies JIRA like softwares are used to handle all agile process.

GitHub

1. **Tell all the steps followed in Github in your project?**
   - ⇨ git clone url (by-default points master branch)
   - ⇨ git checkout sprint1
   - ⇨ git branch (new branch name)
   - ⇨ git checkout new branch (now it points to new branch created) (now make changes in code, after that follow below commands)
   - ⇨ git status
   - ⇨ git add
   - ⇨ git commit –m “commit message”
   - ⇨ git push origin “branch_name”
2. **What was Master Branch?**
   - ⇨ Branch name “develop”, that contains clean code which goes for execution.
   - ⇨ Team leader was developing new master branch for each sprint and then we clone that.

String

1. **What is String Constant Pool (SCP) concept?**
   - ⇨ When we create String object without using new operator the objects are created in SCP (String constant pool) area.
     - String s1=”ABC
     - String s2=”DEF”;
     - String s3=”DEF”;
   - ⇨ When we create object in SCP area then just before object creation it is always checking previous objects.
   - ⇨ If the previous object is available with the same content then it won’t create new object that reference variable pointing to existing object.



- ⇨ If the previous objects are not available then JVM will create new object.
- ⇨ SCP area does not allow duplicate objects.

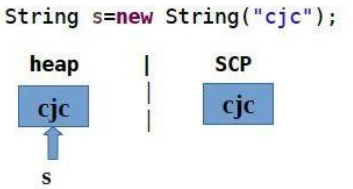
2. **When does object get created in SCP?**
   - ⇨ When String object is created with new keyword.
   - ⇨ When only String variable with content is created.
3. **Why String is immutable?**
   - ⇨ String object is immutable means its internal state remains constant after it has been entirely created. This means that once the object has been assigned to a variable, we can neither update the reference nor mutate the internal state by any means.
   - ⇨ The String is widely used in Java applications to store sensitive pieces of information like usernames, passwords, connection URLs, network connections, etc. It's also used extensively by JVM class loaders while loading classes.
   - ⇨ Hence securing String class is crucial regarding the security of the whole application in general.
   - ⇨ Being immutable automatically makes the String thread safe since they won't be changed when accessed from multiple threads.
   - ⇨ As we saw previously, String pool exists because Strings are immutable. In turn, it enhances the performance by saving heap memory.
   - ⇨ E.g. Database connection properties are stored in String, so that no one can update that object to hack database data and database security can be achieved.
4. **Difference between String & StringBuffer?**

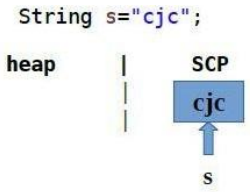
String	StringBuffer
String class is immutable.	StringBuffer class is mutable.
String is slow and consumes more memory when you concat too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when you cancat strings.
String class overrides the equals() method of Object class.	StringBuffer class doesn't override the equals() method of Object class.

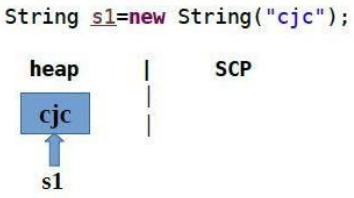
5. **What are the String Scenarios related to String object creation while using Concat()?**
   - ⇨ Concatenation of String :

```
String s="cjc";
String s1=s.concat("class");
```

⇨ String scenarios :







6. **How to use SCP objects in intern () method?**
   - ⇨ It can be used to return string from memory, if it is created by new keyword. It creates exact copy of heap string object in string constant pool.
   - ⇨ E.g. (Sample Code)

```
public class InternExample2 {
  public static void main(String[] args) {
    String s1 = "Javatpoint";
    String s2 = s1.intern();
    String s3 = new String("Javatpoint");
    String s4 = s3.intern();
    System.out.println(s1==s2); // True
    System.out.println(s1==s3); // False
    System.out.println(s1==s4); // True
    System.out.println(s2==s3); // False
    System.out.println(s2==s4); // True
    System.out.println(s3==s4); // False
  }
}
```

7. **How to create immutable class?**
   - ⇨ The instance variable of the class is final i.e. we cannot change the value of it after creating an object.
   - ⇨ The class is final so we cannot create the subclass.
   - ⇨ There is no setter methods i.e. we have no option to change the value of the instance variable.

Logical Problems

1. **Find Even-Odd Number?**

```
Public class Test1
{
 public static void main(String[ ] args)
  {
   Scanner sc=new Scanner(System.in);
   int n=sc.nextInt( );
   if (n%2==0)
    {
         System.out.println(n+"is Even number.");
    } else {
         System.out.println(n+"is Odd number.");
          }
 }
}
```

2. **Find factorial of number?**

```
Public class Test2
{
 public static void main(String[ ] args)
  {
    Scanner sc=new Scanner(System.in);
    int n=sc.nextInt( );
    int sum=1;

    for(int i=1 ; i<=n ; i++)
     {
         sum=sum x i ;
     }
         System.out.println("Factorial of " + n + "is : "+ sum);
 }
}
```

3. **Find Prime Number?**

```
Public class Test
{
 public static void main(String[ ] args)
  {
    Scanner sc=new Scanner(System.in);
    int n=sc.nextInt( );
```

```
   int count=0;
   for(int i=1 ; i<=n ; i++)
    {
         if( n%i == 0)
         {
           count++;
         }
         if(count == 2)
         {
          System.out.println(n + “is prime number.”);
         } else {
          System.out.println(n + “is not a prime number.”);
               }
    }

 }
}
```

4. **Find Armstrong number?**

E.g. 153 => $1^3+5^3+3^3 = 153$, 1634 => $1^4+6^4+3^4+4^4 = 1634$, etc.

```
Public class Test
{
 public static void main(String[ ] args)
 {
  Scanner sc=new Scanner(System.in);
  int n=sc.nextInt( );
  int m=n;
  int count=0;
 while(m>0)
  {
        m=m/10;count++;
  }
 int m=n;
 int sum=0, k=1;
 while(m>0)
  {
   m=m/10; int
   c=count;
   for(int i=1 ; i<=c ; i++)
   {
        k=k x m sum=sum +
        k;
   }
```

```
   }
  if(sum == n)
  {
        System.out.println(n + “is Armstrong number.”);
  } else {
         System.out.println(n + “is not a Armstrong number.”);
              }
 }
}
```

5. **Find Palindrome Number?**

E.g. 121, 3443, 24542, etc.

```
Public class Test
{
 public static void main(String[ ] args)
  {
   Scanner sc=new Scanner(System.in);
   int n=sc.nextInt( );
   int nrev=0, n1 = n;
   while(n>0)
   {
        nrev = nrev x 10 + (n % 10);
        n=n/10;
   }
  if(n1 == nrev)
  {
        System.out.println(n1 + “is Palindrome number.”);
  } else {
         System.out.println(n1 + “is not a Palindrome number.”);
              }
 }
}
```

6. **Find Perfect Number?** E.g.

6 => (3,2,1) => 3+2+1 = 6

```
Public class Test
{
 public static void main(String[ ] args)
  {
   Scanner sc=new Scanner(System.in);
   int n=sc.nextInt( );
   int sum=0;
   for(int i=1 ; i<n ; i++)
    {
```

```
        if(n % i == 0)
        {
          sum=sum + i ;
        }
    }
    if(sum == n)
    {
        System.out.println(n + “is Perfect number.”);
  } else {
          System.out.println(n + “is not a Perfect number.”);
        }

 } }
```

7. **Find Neon Number?**

E.g. 9^2 =81 => 8+1 = 9

```
Public class Test
{
 public static void main(String[ ] args)
 {
   Scanner sc=new Scanner(System.in);
   int n=sc.nextInt( );
   int m=n x n, sum=0;
   while(m>0)
   {
        sum = sum + m%10 ;
        m=m/10;
    }
    if(sum == n)
    {
        System.out.println(n + “is Neon number.”);
  } else {
          System.out.println(n + “is not a Neon number.”);
        }

 }
}
```

8. **Find Spy Number?**

E.g. 1124 => 1+1+2+4 = 1 x 1 x 2 x 4 = 8

```
Public class Test
{
 public static void main(String[ ] args)
 {
   Scanner sc=new Scanner(System.in);
```

```
   int n=sc.nextInt( );
   int sum=0, mult=1;
   while(n>0)
  {
        sum = sum + n%10 ; mult
        = mult x n%10 ;n=n/10;
   }
   if(sum == mult)
    {
        System.out.println(n + “is Spy number.”);
    } else {
          System.out.println(n + “is not a Spy number.”);
          }

 }
}
```

9. **Swap the numbers?**

```
Public class Test
{
 public static void main(String[ ] args)
  {
   Scanner sc=new Scanner(System.in);
   int n1=sc.nextInt( );
   int n2=sc.nextInt( );int
   temp = n1; n1=n2;
   n2=temp;
 }
}
```

10. **Find Automorphic number?**

E.g. $\underline{5}^2 = 2\underline{5}$, $\underline{6}^2 = 3\underline{6}$, $\underline{76}^2 = 57\underline{76}$, etc.

```
Public class Test
{
 public static void main(String[ ] args)
  {
   Scanner sc=new Scanner(System.in);
   int n=sc.nextInt( );
   int m = n x n, n1=n, temp=1;
   while(n1>0)
   {
        n1=n1/10 ;
        temp = temp x 10 ;
   }
```

```
   m = m % temp ;if(n ==
      m)
    {
        System.out.println(n + “is Automorphic number.”);
    } else {
         System.out.println(n + “is not a Automorphic `snumber.”);
         }
 }
}
```

11. Print Fibonacci Series?

E.g. 1, 1, 2, 3, 5, 8, 13 . . . Public class Test

```
{
 public static void main(String[ ] args)
  {
        System.out.println(“1, ”);int j =
        1, k = 0 ;

        for(int i = 0; i<=10; i++)
        {
              int m = j + k ;
              System.out.println( m + “ , ”);k = j
              ;
              j = m ;
        }
  }
}
```

Angular

1) What is module in Angular 4?

- ⇨ Module in Angular refers to a place where you can group the components, directives, pipes, and services, which are related to the application.
- ⇨ In case you are developing a website, the header, footer, left, center and the right section become part of a module.
- ⇨ To define module, we can use the NgModule.
- ⇨ Every Angular app has at least one module, the root module.

2) How to create project using NPM?

- ⇨ Step-1: Install angular cli:
  - o npm install - g @angular/cli

Step-2: Create new project by this command:

- Choose yes for routing option and, CSS or SCSS.

- ng new myNewApp

Step-3: Go to your project directory:

- cd myNewApp

Step-4: Run server and see your application in action:

- ng serve

3) What is NPM?

⇨ NPM stands for Node Package Manager.
⇨ NPM is used to fetch any packages (JavaScript libraries) that an application needs for development, testing, and/or production, and may also be used to run tests and tools used in the development process.
⇨ NPM is 3rd party library or 3rd party package.
⇨ NPM is the package manager for the Node JavaScript platform. It puts modules in place so that node can find them, and manages dependency conflicts intelligently.
⇨ Most commonly, it is used to publish, discover, install, and develop node programs.

4) How to create modules, components, services, interfaces, classes in Angular 4 using NPM?

=> *for Module:

- ng g module newModule

*For Components:

- ng g component new-module/new-component

*for Service

- ng g service newservice
- ng g s newservice

* for Interface

- ng generate interface interfaceName

*for Class

- ng g class className

(Manually)

- Go to component (where new component to create) □ Right Click to create component files □ **.ts** file is compulsory needed, other than that **.html**, **.scss** & **.spec.ts** files are also created while creating component □ add **@Component** Decorator in **.ts** file that contains metadata of component.

5) What is component in Angular 4?

⇨ Components are simply classes with decorators that mark their types and provide metadata which guide Angular to do things.

⇨ Every Angular application always has at least one component known as root component.

⇨ Each component defines a class that contains application data and logic, and is associated with an HTML template that defines a view to be displayed in a target environment.

6) How to run Angular 4 project?

⇨ By using **ng serve** command in Angular CLI.

7) What is service in Angular 4?

⇨ Angular services are singleton objects that get instantiated only once during the lifetime of an application.

⇨ They contain methods that maintain data throughout the life of an application, i.e. data does not get refreshed and is available all the time.

⇨ The main objective of a service is to organize and share business logic, models, or data and functions with different components of an Angular application.

8) What is Angular CLI?

⇨ Angular CLI is a command-line interface (CLI) to automate your development workflow.

⇨ It allows you to:

- Create a new Angular application
- Run a development server with LiveReload support to preview your application during development
- Add features to your existing Angular application
- Run your application's unit tests
- Run your application's end-to-end (E2E) tests
- Build your application for deployment to production.

9) How to call the data from service to component?

⇨ Through method (Has-A-Relationship).

10) What is Decorator in Angular 4?

⇨ Decorators are an excellent way to add metadata information to a class / method / property / parameter.

⇨ Angular uses quite a lot of decorators. There are decorators for classes, properties, methods and even parameters. Some of the important decorators are:

- NgModule
- Component
- Injectable

- Input
- Output

⇨ The Component decorator is used to decorate a class as an angular component and adds additional metadata information like the template, selector name, styles etc. to it.

11) Difference between Angular 1 vs 2 vs 4 vs 5 vs 6?

⇨ Angular 2
- Released in 2016
- Complete rewrite of Angular 1
- Written entirely in typescript
- Component-based instead of Controller
- More testable as component-based
- Support for Mobile/Low-end devices

⇨ Angular 3:
- Why we don't have Angular 3?

⇨ Angular is being developed in a MonoRepo it means a single repo for everything. @angular/core, @angular/compiler, @angular/router etc are in the same repo and may have their own versions.

⇨ The angular router was already in v3 and releasing angular 3 with router 4 will create confusion

⇨ To avoid this confusion they decided to skip the version 3 and release with version 4.0.0 so that every major dependency in the MonoRepo are on the right track.

⇨ **Angular 4**
- Released in 2017
- Changes in core library
- Angular 4 is simply the next version of angular 2, the underlying concept is the same & is an inheritance from Angular 2
- Lot of performance improvement is made to reduce size of AOT compiler generated code
- Typescript 2.1 & 2.2 compatible — all feature of ts 2.1 & 2.2 are supported in Angular 4 application
- Animation features are separated from @angular/core to @angular/animation don't import @animation packages into the application to reduce bundle size and it gives the performance improvement.
- Else block in *ngIf introduced:

— Instead of writing 2 ngIf for else , simply add below code in component template:

*ngIf="yourCondition; else myFalsyTemplate"

"<ng-template #myFalsyTemplate>Else Html</ng-template>"

⇨ **Angular 5**

- Released 1st November 2017
- Build optimizer: It helps to removed unnecessary code from your application
- Angular Universal State Transfer API and DOM Support — by using this feature, we can now share the state of the application between the server side and client side very easily.
- Compiler Improvements: This is one of the very nice features of Angular 5, which improved the support of incremental compilation of an application.
- Preserve White space: To remove unnecessary new lines, tabs and white spaces we can add below code(decrease bundle size)

// in component decorator you can now add:
"preserveWhitespaces: false"
// or in tsconfig.json:
"angularCompilerOptions": { "preserveWhitespaces": false}`

- Increased the standardization across all browsers: For internationalization we were depending on `i18n` , but in ng 5 provides a new date, number, and currency pipes which increases the internationalization across all the browsers and eliminates the need of i18n polyfills.
- exportAs: In Angular 5, multiple names support for both directives and components
- HttpClient: until Angualar 4.3 @angular/HTTP was been used which is now depreciated and in Angular 5 a new module called HttpClientModule is introduced which comes under @angular/common/http package.
- Few new Router Life-cycle Events being added in Angular 5: In Angular 5 few new life cycle events being added to the router and those are:
- ActivationStart, ActivationEnd, ChildActivationStart, ChildActivationEnd, GuardsCheckStart, GuardsCheckEnd, ResolveStart and ResolveEnd.
- Angular 5 supports TypeScript 2.3 version.
- Improved in faster Compiler support:
- A huge improvement made in an Angular compiler to make the development build faster. We can now take advantage of by running the below command in our development terminal window to make the build faster.
  ng serve/s — aot

⇨ **Angular 6**

- Released on April 2018
- This release is focused less on the underlying framework, and more on tool-chain and on making it easier to move quickly with angular in the future
- No major breaking changes
- Dependency on RxJS 6 (this upgrade have breaking changes but CLI

command helps in migrating from older version of RxJS)
- Synchronizes major version number of the:
  – Angular framework
  – Angular CLI
  – Angular Material + CDK
- All of the above are now version 6.0.0, minor and patch releases though are completely independent and can be changed based on a specific project.
- Remove support for <template> tag and "<ng-template>" should be used.
- Registering provider: To register new service/provider, we import Service into module and then inject in provider array. e.g:

```
// app.module.ts
import {MyService} from './my-service';
...
providers: [...MyService]
...
```

- But after this upgrade you will be able to add provided In property in injectable decorator. e.g:

```
// MyService.ts@Injectable({ providedIn: 'root'})
export class MyService{}
```

- The way ngModelChange event works:
  - Let's understand this with output produced by older and this version:
  - // Angular 5:
    <input [(ngModel)]='name'
    (ngModelChange)='onChange($event)' /> onChange(value){
    console.log(value); } // Would log updated value

    <input #modelDir='ngModel' [(ngModel)]='name'
    (ngModelChange)='onChange(modelDir)' />
    onChange(ngModel: NgModel){ console.log(ngModel.value); } //
    Would log old value, not updated
  - // Angular 6:

```
onChange(ngModel: NgModel){ console.log(ngModel.value); } // Would log
updated value
```

CLI Changes: Two new commands have been introduced
– ng update <package>
* Analyse package.json and recommend updates to your application
* 3rd parties can provide update scripts using schematics
* automatically update code for breaking changes
* staying update and low maintenance
– ng add
* add new capablities to your applicaiton
* e.g ng add @angular/material : behind the scene it add bit of necessary code and changes project where needed to add it the thing we just told it to add.
* Now adding things like angular material, progressive web app, service workers

& angular elements to your existing ng application will be easy.
CLI + Material starter templates: Let angular create code snippet for your basic components. e.g:
– Material Sidenav
* ng generate @angular/material:material-nav — name=my-nav
Generate a starter template including a toolbar with app name and then the side navigation & it's also responsive
– Dashboard
* ng generate @angular/material:material-dashboard — name=my-dashboard Generates Dynamic list of cards
– Datatable
* ng generate @angular/material:material-table — name=my-table Generates Data Table with sorting, filtering & pagination
It uses angular.json instead of .angular-cli.json
Support for multiple projects: Now in angular.json we can add multiple projects initial release of Angular Elements which gives us ability to use our angular components in other environments like a Vue.js application. Its potential is truly amazing but unfortunately this release only works for angular application, we need to wait for next release to wrap out angular component into custom element and use it with framework like Vue.js

12) Difference between http and httpclient?

- ⇨ Http: The HttpClient is used to perform HTTP requests and it imported from @angular/common/http.
- ⇨ Httpclient: The HttpClient is more modern and easy to use the alternative of HTTP. HttpClient is an improved replacement for Http.
- ⇨ https://blog.hackages.io/angular-http-httpclient-same-but-different-86a50bbcc450

13) What is Rxjs?

- ⇨ RxJS (Reactive Extensions for JavaScript) is a library for reactive programming using observables that makes it easier to compose asynchronous or event-based code.
- ⇨ RxJS can be used both in the browser and in the server-side using Node.

14) How to create single page application in Angular 4?

- ⇨ Single-Page Applications (SPAs) are Web apps that load a single HTML page and dynamically update that page as the user interacts with the app. SPAs use AJAX and HTML5 to create a fluid and responsive Web apps, without constant page reloads. However, this means much of the work happens on the client side, in JavaScript.
- ⇨ By using Child Routing Process.

15) What is Observable?

- ⇨ RxJS introduces Observables, a new Push system for JavaScript. An Observable

is a Producer of multiple values, "pushing" them to Observers (Consumers). A Function is a lazily evaluated computation that synchronously returns a single value on invocation.

- ⇨ Observable are just that — things you wish to observe and take action on. Angular uses the Observer pattern which simply means — Observable objects are registered, and other objects observe (in Angular using the subscribe method) them and take action when the observable object is acted on in some way.

16) What is Structural Directive & Different Structural Directives in Angular?

- ⇨ Structural directives are responsible for HTML layout. They shape or reshape the DOM's structure, typically by adding, removing, or manipulating elements. As with other directives, you apply a structural directive to a host element.
- ⇨ Each structural directive does something different with that template.
- ⇨ There are 3 built-in structural directives – NgIf, NgFor and NgSwitch.

17) What is routing?

- ⇨ To handle the navigation from one view to the next, you use the Angular router. The router enables navigation by interpreting a browser URL as an instruction to change the view.

18) How routing works?

- ⇨ An Angular application that uses Angular Router only has one router service instance: It's a singleton. Whenever and wherever you inject the Router service in your application, you'll get access to the same Angular Router service instance.
- ⇨ To enable routing in our Angular application, we need to do three things:
    a. create a routing configuration that defines the possible states for our application
    b. import the routing configuration into our application
    c. add a router outlet to tell Angular Router where to place the activated components in the DOM.

19) What is Angular 4 digest cycle?

- ⇨ Digest cycle is what Angular JS triggers when a value in the model or view is changed. The cycle sets off the watchers which then match the value of model and view to the newest value. Digest cycle automatically runs when the code encounters a directive.

20) How to pass data from one component to another?

- ⇨ Components can communicate with each other in various ways, including:
    a. Parent to Child: via @Input()
    b. Child to Parent: via @Output() and EventEmitter

c. Child to Parent: via @viewChild()
d. Unrelated Components: via a Service

⇨ https://fireship.io/lessons/sharing-data-between-angular-components-four-methods/