TECHTORIAL ACADEMY
JAVA INTERVIEW PREPARATION DOCUMENT

TECHTORIAL ACADEMY
JAVA INTERVIEW PREPARATION DOCUMENT

1. Versions of java you worked with? What version of java do you currently use in your framework?

Version Name	Release Date
Java SE 7	July 2011
Java SE 8	**March 2014**
Java SE 9	September 2017
Java SE 10	March 2018
Java SE 11	September 2018
Java SE 15	September 2020

2. Difference between JRE, JDK, JVM ?

JRE stands for **Java Runtime Environment** which we usually download as a Java software. The JRE consists of the Java Virtual Machine, Java platform classes, and supporting libraries. The JRE is the runtime component of Java software and is all we need to run any Java application.

JDK stands for **Java Development Kit** is a superset of the JRE and includes everything that the JRE contains. Additionally, it comes with the compilers and debuggers tools required for developing Java applications.

JVM stands for **Java Virtual machine**. It translates and executes the Java bytecode. It's the entity which transforms Java to become a "portable language" (i.e. write once, run anywhere). Java compiler generates bytecode for all the Java code and converts into class files.

3. Difference between '= =' operator , '=' sign and .equals() method?

= **we are assigning value**
== **comparison operator (reference and address comparison)**
When we use it with primitives it is checking the value. When we use the == sign for the Object like String, it will compare whether the reference of the object is pointing to the same object in the memory or not.
We can use the = = sign for the wrapper class object as well.

.equals() → Equals method in String will compare whether the value of the String is **exactly** the same or not. Equals method will also compare the **value** of the object in **Wrapper** classes as well. However, if the object has **no equals()** method implemented, it will come from the **Object** class and if it comes from the object class, it will check if the **references** of the **object** are pointing to the same object in java memory or not.

4. What is the main method? Why do we need one in java? Do we have to have a main method in java?

Main method is the starting point of an application. JVM starts execution by invoking the main method of some specified class, passing it a single argument, which is an array of strings. Whenever we execute a program, the main() is the first function to be executed. We can call other functions from main to execute them. It is not mandatory to have a main method in java, without main() our Java code will compile but won't run.

5. Explain public static void main (String args[])?

public: it is an access specified that means it will be accessible by any Class. **static:** is a keyword to call this method directly using class name without creating an object of it.
void: it is a return type i.e. it does not return any value.
main(): it is the name of the method which is searched by JVM as a starting point for an application with a particular signature only. it is the method where the main execution occurs.
string args[]: it's a command line arguments passed to the main method.

* * *

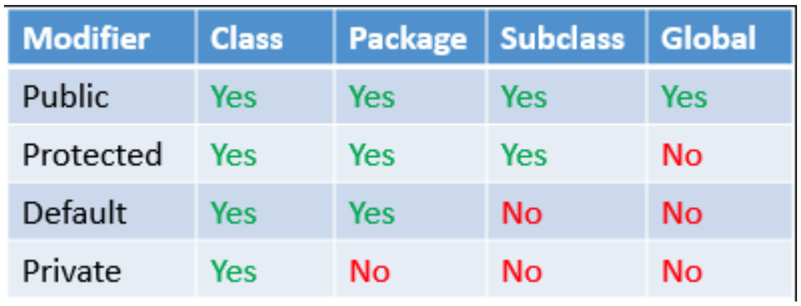
6. What is System.out. println()?

System : System is class in java.
Out : out is one instance variable name in java. Data type of this variable is PrintStream.
println() : it is one method. The PrintStream class has a println method. That's why we can call using the 'out' reference name.

7. What are Access Modifiers (Private,public,protected)? How did you use them?

Java provides access modifiers to set access levels for classes, variables, methods and constructors.
public: A class or interface may be accessed from outside the package. Constructors, inner classes, methods and field variables may be accessed wherever their class is accessed.

protected: Accessed by other classes in the same package or any subclasses of the same package or different package.
private: Accessed only within the class in which they are declared. We usually use it for the encapsulation.
default: When no access modifier is specified for a class , method or data member – It is said to be having the default access modifier by default.



Modifier	Class	Package	Subclass	Global
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No

8. How can we access variables without creating an object instance of it? What are Instance variables and how do you use it? What is the difference between local and instance variables?

Local Variables: Variables which are declared **inside a method or constructor or blocks are called local variables.** Local variables are created when a method is called and destroyed when the method exits.
Instance Variables: Variables which are declared **inside the class, but outside a method, constructor or any block are called instance variables.** We can access instance variables by creating an Object of the class they belong to. Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.

Example of Instance Variable from the framework

```
WebDriver driver;
```

9. Difference between Instance Variable and static Variable? What is a static keyword in java? Where did you use static in your framework?

1. Static variables are declared with the **static** keyword in a class, but outside a method, constructor or a block. By declaring a variable as a static we can access it from different classes without creating an Object - those variables called **class variables** and also known as **static variables.**
Whereas, **Instance** variables are declared in a class, but outside a method, constructor or any block. To access instance variables we need to create an object of the Class they belong to.

```
public class Test {

String Url = "https://www.techtorial.com/";// Instance variable

static String Url2 = "https://www.orangeHRM.com/";// Static variable

public static void main(String[] args) {

System.out.println(Url2);
Test t1 = new Test();
System.out.println(t1.Url);
}
}
```

2. Class variables only have one copy that is shared by all the different objects of a class, whereas every object has its own **personal copy of an instance variable**. So, **instance variables across different objects can have different values** and when we make changes to the instance variable they don't reflect in other instances of that class **whereas class variables across different objects can have only one value.**

3. Static variables are created when the program starts and destroyed when the program stops whereas instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.

Static keyword in java:

● Static keyword means that the variable or method **belongs to the class** and **shared between all** instances (Objects).
● Using static keyword we can access class variables and methods **without object reference**

● Static methods **can not call** Non Static members

Usage of static keyword in framework:

In our utility package we have a class where we store common methods, such as wait, switch between frames, clicking on buttons, selecting values from drop down. So those methods are written using static keywords and we can easily access them in our program.

```
public static WebElement waiting(WebElement element) {
WebDriverWait wait = new WebDriverWait(driver, 30);
return wait.until(ExpectedConditions.elementToBeClickable(element)); }
```

In our Base Class we have static variables

```
public class BaseClass {

public static WebDriver driver;
}
```

In our Configs Reader we have static variables

```
public class ConfigsReader {
public static Properties prop;
}
```

10. What is a constructor? Use of constructor in class? Can you make the constructor static? Can we overload a constructor?

What is constructor

A constructor in java is a block of code similar to a method. Constructor called when an instance of a class is created. A constructor is a special method whose task is to initialize the object of its class. **Constructors cannot be abstract, final, static.**

Rules to create constructor:

1. Constructor name and class name must be the same

2. Constructor do not have any return type.

3. Constructor may or may not have parameters.

Usage of Constructor

The main use of constructor is to **initialize** the instance variables. Constructors are special functions which are called automatically when we create objects of the class. So once we create an object of the class all the variables get initialized, and we don't need to write extra code for initialization of variables.

Constructor is the property of an object while static has nothing to do with object. That's why there is nothing like a static constructor. But we have a static block to do the similar task as constructor i.e. initialization of fields etc.

11. What is the difference between constructor and method?

● Constructor must not have a return type whereas methods must have a return type.

● Constructor name is the same as the class name where as method may or may not the same class name.

● Constructor will be called automatically whenever an object is created whereas the method invokes explicitly.

● Constructor compiler provides **default constructor** whereas method compiler doesn’t provide.

Example of constructor from framework

creating constructor to initialize instance variables

```
public class HomePage {

   public HomePage(WebDriver driver){
      PageFactory.initElements(driver,this);
   }

   @FindBy(xpath = "//h4")
   public WebElement loginText;
}
```

WE CAN OVERLOAD CONSTRUCTOR using different parameters. When we create the constructor, the default constructor will not be available anymore.

12. Super vs super()? this vs this()? Can super() and this() keywords be in the same constructor?

this vs this()

● **this keyword** is used to refer to the **current object** and differentiate between local and instance variables

```
public class City {
   private String name;
   private double population;
   private int areaCode;
   private String mayor;
   private String localTime;

   public City(String name, double population, int areaCode, String mayor, String localTime) {
      this.name = name;
      this.population = population;
      this.areaCode = areaCode;
      this.mayor = mayor;
      this.localTime = localTime;
   }
}
```

● **this()** is used to access one constructor from another where both constructors belong to the same class.

```
public class City{
```

```
City() {
System.out.println("This a no-argument constructor");
}

City(String a) {
this(); // → this one calls the no-argument constructor. It must be in the first line.
System.out.println("One argument constructor);
} }
```

super vs super()

Both are used in a subclass as a way to invoke or refer to its superclass.

● **super keyword** is used to call super class(parent class/ base class) variables and methods by the subclass object when they are overridden by subclasses.

● **super()** is used to call super class constructor from subclass constructor.

```
 public class SuperKeyword1 extends SuperKeyword{

SuperKeyword1(){

super(4);

System.out.println("This is a child default constructor");

}
```

We can use **super()** and **this()** only in the constructor, not anywhere else, any attempt to do so will lead to a compile-time error. **This()** and **super()** are always have to be in first line within constructor and for that reason we **CANNOT** use them within the same constructor. We have to keep either super() or this() as the first line of the constructor but NOT both simultaneously.

13. Difference between an abstract class and interface? Can we create an object for an abstract class? interface? When to use abstract class and interface in Java?

Interface is a blueprint for your class that can be used to implement a class. Interface is a collection of public static methods and public static final variables

An abstract class is the class which is declared with abstract keyword and can contain defined(concrete) and undefined(abstract) methods.

Abstract Class	Interface

The abstract keyword is used to declare abstract class	The interface keyword is used to declare interface
Abstract class does not support multiple inheritance	Interface support multiple inheritance
Abstract class contains Constructors	Interface doesn't contain Constructors
An abstract class Contains both incomplete (abstract) and complete member and Abstract class can have abstract and non-abstract methods.	An interface Contains only incomplete member (signature of member) and Interface can have only abstract methods.
An abstract class can contain access modifiers for the methods, properties(variables)	An interface cannot have access modifiers by default everything is assumed as public
Abstract class can have final, static and non-static variables.	Interface has only final static variables.
These properties can be reused commonly in a specific application.	These properties are commonly usable in any application of java environment.
Abstract class may contain either variables or constants.	Interface should contain only constants.
The default access specifier of abstract class methods are default.	The default access specifiers of interface methods are public.
These class properties can be reused in other class using extends keyword.	These properties can be reused in any other class using implements keyword.
For the abstract class there is no restriction like initialization of variables at the time of variable declaration.	For the interface it should be compulsory to initialization of variables at the time of variable declaration.
There are no any restrictions for abstract class variables.	For the interface variable can not declare variable as private, protected
There are no restrictions for abstract class method modifiers that means we can use any modifiers.	For the interface method cannot declare method as protected, static, private, final

We cannot create an object of interface or an abstract class !

● An abstract class is good if you think you will plan on using inheritance since it provides a common base class implementation to derived classes.
● An abstract class is also good if you want to be able to declare non-public members. In an interface, all methods must be public.

● If you think you will need to add methods in the future, then an abstract class is a better choice. Because if you add new method headings to an interface, then all of the classes that already implement that interface will have to be changed to implement the new methods.

Practical Example of an Interface:

Basic statement we all know in **Selenium** and **Java** is

```
WebDriver driver = new ChromeDriver();

List<String> names=new ArrayList<>();

Set <Integer> numbers= new HashSet<>();
```

WebDriver itself is an Interface. We are initializing the Chrome browser using Selenium WebDriver. It means we are creating a reference variable (driver) of the interface (WebDriver) and creating an Object. Here **WebDriver** is an **Interface** and **ChromeDriver** is a class.

List and Set is an interface in java. We need to instantiate the object using ArrayList for List and HashSet, LinkedHashSet and TreeSet for Set.

Practical Example of an Interface:

```
public static byte[] takeScreenshot() {

TakesScreenshot ts = (TakesScreenshot) driver;
byte[] screen = ts.getScreenshotAs(OutputType.BYTES);

return screen;
}

public static void scrollDown(int pixels) {
JavascriptExecutor js = (JavascriptExecutor) driver;
js.executeScript("window.scrollBy(0," + pixels + ")");
}
}
```

Practical Usage of Abstraction

Page Object Model design pattern, we write locators (such as id, name, xpath etc.,) in a Page Class.

We utilize these locators in tests but we can't see these locators in the tests. Literally we hide the locators from the tests.

14. Explain OOPS concepts? Is java 100% object oriented?

OOP concepts in Java are the main idea behind Java's Object Oriented Programming. They are an **abstraction, inheritance, polymorphism and encapsulation.**

Inheritance is a mechanism in which one object acquires all the properties and behaviors of a parent object.
Polymorphism is the ability of an object to take on many forms.
Abstraction is the methodology of hiding the implementation of internal details and showing the functionality to the users.
Encapsulation is a mechanism of binding code and data together in a single unit.

No, **Java is not 100% object oriented**, since it has primitive data types, which are different from objects.

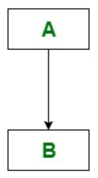
15. What is inheritance and benefits of it? Types of inheritance? How do you use it in your code?

Inheritance

● The process of acquiring properties (variables) &methods (behaviors) from one class to another class is called inheritance.
● We are achieving the inheritance concept by using extends keyword. Also known as is-a relationship.
● Extends keyword is providing the parent-child relationship between two classes.
● The main objective of inheritance is code extensibility whenever we are extending the class automatically code is reused.

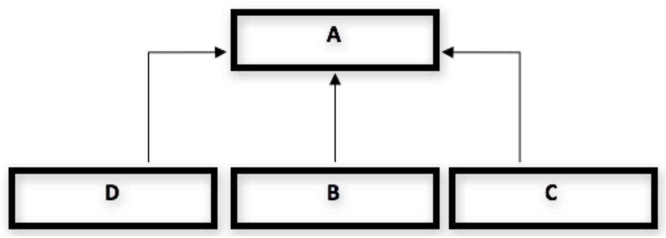
Types of Inheritance:

● **Single Inheritance** - single base class and single derived class.

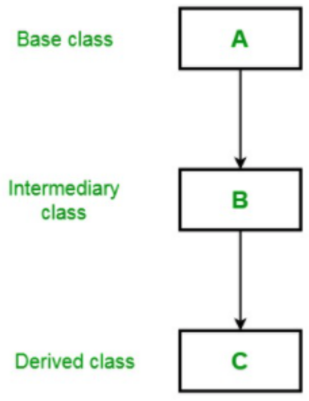


Single Inheritance

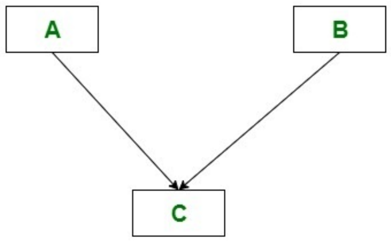
● **Hierarchical Inheritance** - when a class has more than one child classes (sub classes)



● **Multilevel Inheritance** - single base class, single derived class and multiple intermediate base classes.

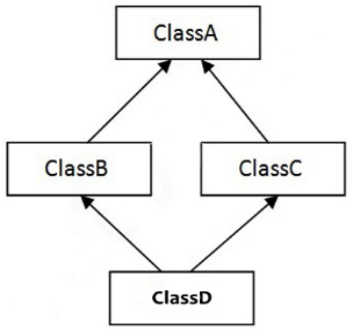


● **Multiple Inheritance** - multiple classes and single derived class (Possible through interface only)



Multiple Inheritance

● **Hybrid Inheritance** - combination of both Single and Multiple Inheritance (Possible through interface only)



4) Hybrid Inheritance

Usage of inheritance in real time project

In our current Cucumber framework we have a TestBase class where we initialize the WebDriver interface. And after we extend the Base Class in other classes such as Pages to initialize page elements and to the Common methods where we have functions to work with Web Browser.

16. What is polymorphism? Types of polymorphism?

Polymorphism is the ability of an object to take on many forms. Polymorphism allows us to perform a task in multiple ways.

Combination of overloading and overriding is known as Polymorphism.

There are two types of Polymorphism in Java

1. Compile time polymorphism (Static binding) – Method overloading 2. Runtime polymorphism (Dynamic binding) – Method overriding

17. Method overloading & overriding? How do you use it in your framework? Any example or practical usage of Run time polymorphism?

Method overloading in Java occurs when two or more methods in the same class have the exact same name but different parameters (remember that method parameters accept values passed into the method).
Overloading: Same method name with different arguments **in the same class**

Practical Example of Overloading

```
public static void switchToFrame(String idOrName) {

try {

driver.switchTo().frame(idOrName);
} catch (NoSuchFrameException e) {
System.out.println("Frame is not present");
}
}

public static void switchToFrame(WebElement element) {
try {
driver.switchTo().frame(element);
} catch (NoSuchFrameException e) {
System.out.println("Frame is not present");
}
}

public static void switchToFrame(int index) {
try {
driver.switchTo().frame(index);
} catch (NoSuchFrameException e) {
System.out.println("Frame is not present");
}
}
```

18. Method overriding

Declaring a method in child class which is already present in the parent class is called Method Overriding.In simple words, overriding means to override the functionality of an existing method.
With method overriding a child class can give its own specific implementation to an inherited method

without modifying the parent class method.

***** Rules to Override the Methods *****

1. The method in the child class must have the same signature as the method in the parent class
2. The method in the child class must be at least as accessible or more accessible than the method in the parent class.
3. The method in the child class may not throw a checked exception that is new or broader than the class of any exception thrown in the parent class method.
4. Return type of the method in the child class must be the same as the parent class or covariant return types.
5. NOTE: between primitive data type there is no covariant relationship however **Number** is parent **class** of wrapper class object.

Practical Usage:

1. Implementation of WebDriver interface.

```
WebDriver driver = new FirefoxDriver();
WebDriver driver = new ChromeDriver();
```

2. Implementation of iTestListener interface.

```
 public class Listener implements ITestListener {

@Override
public void onTestStart(ITestResult result) {//for method
System.out.println("Starting Test: "+result.getName());
}

@Override
public void onTestSuccess(ITestResult result) {
System.out.println("Test case passed: "+result.getName());
}

@Override
public void onTestFailure(ITestResult result) {
System.out.println("Test case failed: "+result.getName());
}

@Override
public void onTestSkipped(ITestResult result) {
System.out.println("Test case skipped: "+result.getName());
}
```

2. Selenium WebDriver provides an interface WebDriver, which consists of abstract methods getDriver() and closeDriver(). So any implemented class with respect to browsers can override those methods as per their functionality, like ChromeDriver implements the WebDriver and can override the getDriver() and closeDriver().

19. Can we override/overload the main method? Explain the reason? Can you override the static method? Can we overload and override private methods?

We cannot override a static method, so we cannot override the main method. However, you can overload the main method in Java. But the program doesn't execute the overloaded main method when you run your program; you have to call the overloaded main method from the actual main method. Practically I do not see any use of it and we don’t use it in my framework.

```
public class MainMethodOverload {
public static void main(String[] args) {
main(5);
}
public static void main(int r) {
System.out.println("Hello");
}
}
```

Static methods are bound with class; it **is not possible to override static methods. When we implement a static method under the child class, it is hiding the method, not overriding.**

```
class Parent {
static void m1() {
System.out.println("parent m1()");
}
}
class Child extends Parent {
static void m1() {
System.out.println("child m1()");
}

public static void main(String[] args) {
Parent p = new Child();
p.m1();
} }
```

In java it is **not possible to override private** methods because these methods are specific to classes, not visible in child classes.

20. What is binary search and how it is working?

Binary search will compare the target value starting from the middle of the list and it keeps continuing to compare every time from the middle element until finding the last two values. It is very fast comparing to linear search.

21. What is encapsulation?

It is the technique of making the fields in a class private and providing access to the fields via public methods. If a field is declared private, it cannot be accessed by anyone outside the class, thereby hiding the fields within the class. Therefore encapsulation is also referred to as data hiding. The main benefit of encapsulation is the **ability to modify our implemented code without breaking the code of others** who use our code. Encapsulation gives maintainability, flexibility and extensibility to our code.

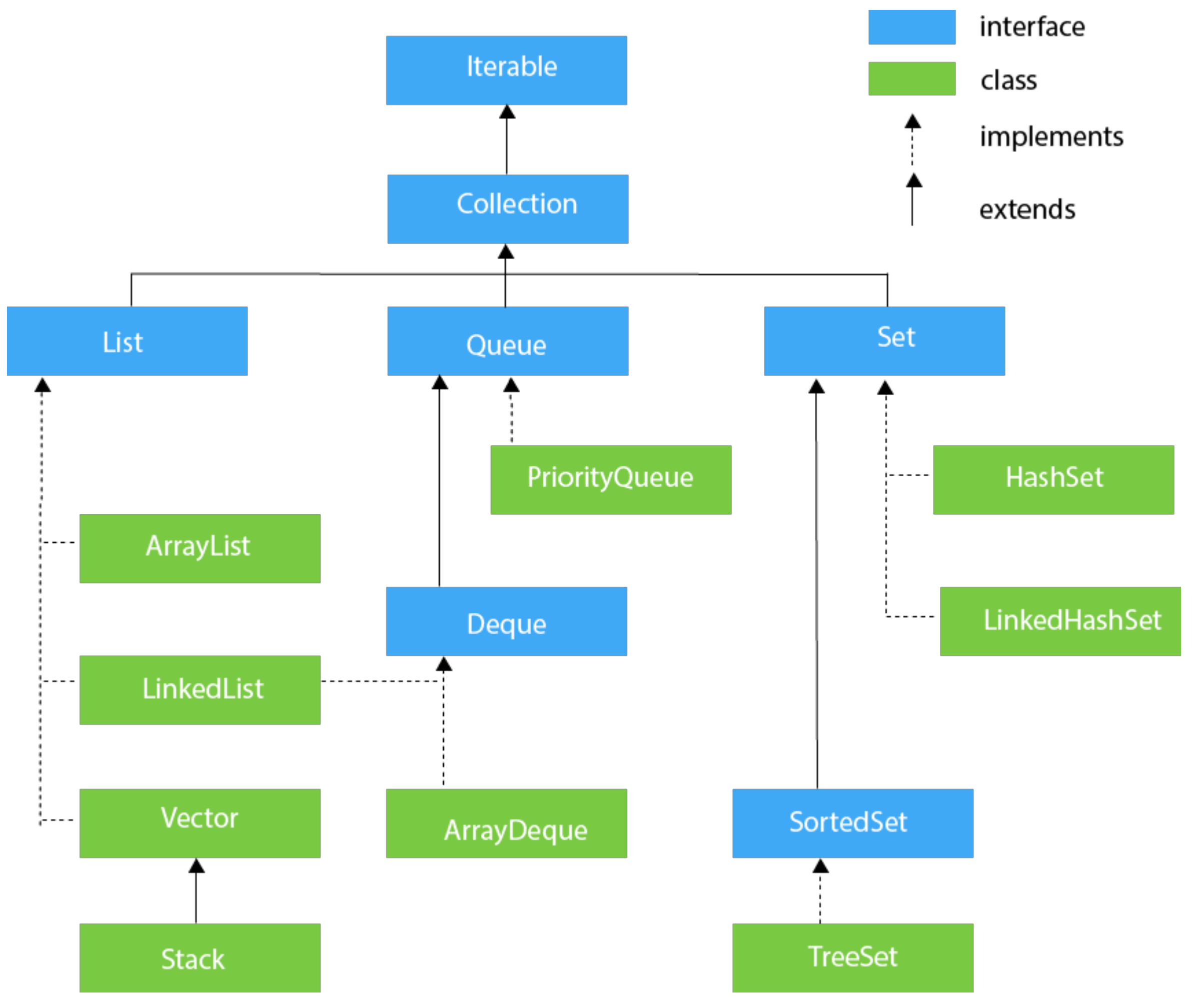
22. What are the primitives and wrapper classes?

Primitives are data types in Java. There are a total of **8 primitive data types in Java: byte, short, int, long, float, double, char, boolean.**

Every primitive data type has a class dedicated to it and these are known as wrapper classes. **These classes wrap the primitive data type into an object** of that class.

23. What is collection in Java and what type of collections have you used?

Java provides Collection Framework which defines several classes and interfaces to represent a group of objects as a single unit.



Maps are not part of collection but built based on the collection concepts

Mostly in my current project we use List and Map.

Practical Example of List & Map usage

```
public static List<Map<String, String>> getResultSetData(String sql) throws SQLException {
statement = conn.createStatement();
resultSet = statement.executeQuery(sql);

List<Map<String, String>> rsList = new ArrayList<>();
ResultSetMetaData rsMetaData = resultSet.getMetaData();
int cols = rsMetaData.getColumnCount();
while (resultSet.next()) {
Map<String, String> rsData = new HashMap<>();
for (int i = 1; i <= cols; i++) {
```

```
rsData.put(rsMetaData.getColumnName(i),
resultSet.getObject(i).toString());
}
rsList.add(rsData);
}
return rsList;
}
```

Practical Example of Map usage

```
public void i_enter_invalid_username_and_password_I_see_errorMessage(DataTable
wrongCredentials) {

List<Map<String, String>> maps = wrongCredentials.asMaps();
}
```

Practical Example of List usage

```
public void i_see_following_labels(DataTable addEmpLabels) {

List<String> expectedLabels = addEmpLabels.asList();
```

24. What is Array and Arraylist (List)? Difference between them?

● Arrays are fixed in size but ArrayLists are dynamic in size.
● Array can contain both primitives and objects but ArrayList can contain only object elements.
● To find the size on an Array we use ArrayName.length and for arrayList we use ArrayListName.size()
● Array use assignment operators to store elements but ArrayList use add() to insert elements.
● Array can be multi dimensional , while ArrayList is always single dimensional. **ArrayList:**

```
ArrayList names = new ArrayList();
names.add("Daniela");
names.add("Patrick");
```

How to print all values from arrayList
//1. using for loop

```
for (int i=0; i<names.size();i++) {
System.out.println(names.get(i));
}
```

```
//2. use advance for loop/enhanced/for each loop

for (String value: names) {
System.out.println(value);
}

//3 using Iterator
Iterator<String> it=names.iterator();//create/initialize Iterator
while(it.hasNext()) {
String name=it.next();
System.out.println(name);
}




//4 using while loop
int count=0;
while(names.size()>count) {
System.out.println(names.get(count));
count++;
}

Array:
String[] array=new String[3];
array[0]="Jordan";
array[1]="Jack";
array[2]="Jack";
for(int i=0; i<array.length; i++) {
System.out.println(array[i]);
}
```

25. Difference between ArrayList vs LinkedList?

ArrayList and LinkedList, both implement List interface and provide capability to store and get objects as in ordered collections. Both are non synchronized classes and both allow duplicate elements.

ArrayList

● ArrayList internally uses a dynamic array to store the elements.
● Manipulation with ArrayList is slow because it internally uses an array. If any element is removed

from the array, all the bits are shifted in memory. ● ArrayList is better for storing and accessing data.

LinkedList

● LinkedList internally uses a doubly linked list to store the elements (consist on value + pointer to previous node and pointer to the next node)
● Manipulation with LinkedList is faster than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
● LinkedList is better for manipulating data.

//Create linked list

```
LinkedList linkedList = new LinkedList();
//Add elements
linkedList.add("A");

linkedList.add("B");
System.out.println(linkedList);
//Add elements at specified position  linkedList.add(2, "C");
linkedList.add(3, "D");
System.out.println(linkedList);
//Remove element
linkedList.remove("A"); //removes A  linkedList.remove(0); //removes B
System.out.println(linkedList);
```

Additional:
ArrayList vs Vector?

Both implement List Interface and maintains insertion order
ArrayList is not synchronized, so it is fast.
Vector - is synchronized, so it is slow.

26. Difference between HashSet vs LinkedHashSet vs TreeSet ?

HashSet

1. HashSet class implements Set interface
2. In HashSet, we store objects(elements or values).
3. HashSet does **not allow duplicate** elements that mean you cannot store duplicate values in HashSet.
4. HashSet permits to have a single **null** value.
5. HashSet is **not synchronized**.

LinkedHashSet

1. Java LinkedHashSet class contains unique elements only like HashSet.
2. Java LinkedHashSet class provides all optional set operation and permits null elements.
3. Java LinkedHashSet class is not synchronized.
4. Java LinkedHashSet class maintains insertion order.

TreeSet

1. Java TreeSet class contains unique elements only like HashSet.
2. Java TreeSet class access and retrieval times are quiet fast.
3. Java TreeSet class doesn't allow null elements.
4. Java TreeSet class is non synchronized.
5. Java TreeSet class maintains ascending order.

Bonus:

ArrayList vs HashSet?

Both ArrayList and HashSet are non synchronized collection class

Both ArrayList and HashSet can be traversed using Iterator

ArrayList

- ArrayList implements List interface
- ArrayList allows duplicate values
- ArrayList maintains the order of the object in which they are inserted
- In ArrayList we can add any number of null values
- ArrayList is index based

HashSet

- HashSet implements Set interface
- HashSet doesn’t allow duplicates values
- HashSet is an unordered collection and doesn’t maintain any order
- HashSet allow one null value
- HashSet is completely object based

27. What is Map/ HashMap vs LinkedHashMap vs TreeMap? How did you use it in your framework?

Java Map Interface. A map contains values on the basis of key, i.e. key and value pairs. Each key and value pair is known as an entry. Map is a collection of entry objects. A Map Contains unique keys. A Map is useful if we have to search, update or delete elements on the basis of a key. The Map interface is implemented by different Java classes, such as HashMap, HashTable, and TreeMap.

HashMap: it makes no guarantees concerning the order of iteration, HashMap doesn't maintain the insertion order of elements.

LinkedHashMap: It orders its elements based on the order in which they were inserted into the set (insertion-order).

TreeMap: It stores its elements in a red-black tree, orders its elements based on their values; it is substantially slower than HashMap.

```
Map<String, Integer> groceryMap=new LinkedHashMap<>();
groceryMap.put("Milk", 1);
groceryMap.put("Bread", 2);
groceryMap.put("Ice Cream", 10);
groceryMap.put("Cookie", 5);
groceryMap.put("Tea", 3);

for(Map.Entry<String, Integer> entry : groceryMap.entrySet()) {
if (entry.getKey().contains("Tea")) {
entry.setValue(2);
}  }

for (String key:groceryMap.keySet()) {
if(key.contains("Cookie")) {
groceryMap.replace(key, 3);
} }

System.out.println(groceryMap);
```

28. Difference between HashTable and HashMap ?

Both **HashMap** and **Hashtable** implement Map Interface

HashMap

● HashMap is non synchronized, so it is not-thread safe
● HashMap is fast
● HashMap allows one null key and multiple null values

```
HashMap <Integer, String> days=new HashMap<Integer, String>();
```

```
days.put(1, "Monday");
days.put(1, "Monday");
days.put(2,"Tuesday");
days.put(null, "Wednesday");
days.put(null, "Thursday");

for (Map.Entry obj: days.entrySet()) {
System.out.println(obj.getKey()+" "+obj.getValue());
}
```

Output:

```
null Thursday
1 Monday
2 Tuesday
```

Hashtable

● Hashtable is synchronized, so it is thread-safe
● Hashtable is slow
● Hashtable doesn't allow any null key or value

```
Hashtable <Integer, String> days=new Hashtable<Integer, String>();

days.put(1, "Monday");
days.put(1, "Monday");
days.put(2,"Tuesday");
days.put(null, "Wednesday");
days.put(null, "Thursday");

for (Map.Entry obj: days.entrySet()) {
System.out.println(obj.getKey()+" "+obj.getValue());
}
```

Output:

```
Exception in thread "main" java.lang.NullPointerException
```

29. How can you handle exceptions? Types of exceptions you faced in your project? What is the parent of all exceptions?

An Exception is a problem that can occur during the normal flow of execution. Depending on the situation, we can use try catch finally blocks.
In **try block:** Code that might throw some exceptions

In **catch block:** We define exception type to be caught and what to do if an exception happens in TRY block code.

Throwable class is parent of all Exceptions:

```
try {
int a=10;
int b=0;
int c=a/b;
}catch (ArithmeticException e) {
System.out.println(e.getMessage());
}
```

Types of Exception:

1. Checked Exception - are the exceptions that are checked at compile time. Example of checked exceptions:

- **ClassNotFoundException** - Class not found
- **InstantiationException** - Attempt to create an object of an abstract class or interface
- **FileNotFoundException -** Attempt to open file that doesn't exist or open file to write but have only read permission

2. Unchecked Exception - are the exceptions that are not checked at compile time, they are Runtime Exceptions.

Exception faced as part of java perspective:

- **ArithmeticException** - Arithmetic error, such as divide-by-zero.
- **ArrayIndexOutOfBoundsException** - Array index is out-of-bounds.
- **NullPointerException** - Invalid use of a null reference.
- **IllegalArgumentException** - Illegal argument used to invoke a method.

Example of exception handling from current framework:

```
public static void createConnection() {
try {
conn =
DriverManager.getConnection(ConfigsReader.getProperty("oracleDbUrl"),
ConfigsReader.getProperty("oracleDbUser"),
ConfigsReader.getProperty("oracleDbPassword"));

} catch (SQLException e) {
e.printStackTrace(); } }
```

30. How many catch blocks can we have? Which catch block will get executed if you get an ArithmeticException?

There can be any number of catch blocks for a single try block and it is not necessary that each try block must be followed by a catch block. It should be followed by either a catch block or a finally block. However only the catch block encountered first on the call stack that satisfies the condition for the exception will be executed for that particular exception, rest will be ignored.

```
try {
int a=10;
int b=0;

int c=a/b;
}catch (ArithmeticException e) {
System.out.println(e.getMessage());
}catch (Exception e) {
System.out.println(e.getMessage());
}
```

31. What is the difference between throw and throws?

Throws :

● is used to declare an exception, which means it works similar to the try-catch block.
● is used in method declaration.
● is followed by exception class names.
● you can declare multiple exception with throws
● throws declare at method it might throws Exception
● used to handover the responsibility of handling the exception occurred in the method to the caller method.

1 Example:

```
public void readPropFile() throws FileNotFoundException, IOException{

Properties prop=new Properties();

FileInputStream fis=new FileInputStream("fileNamePath.properties"); prop.load(fis);

}
```

2 Example:

```
public class Test {
public static void main(String[] args) throws InterruptedException {
```

```
Test test = new Test();
test.company();
}

void studentDetails() throws InterruptedException {
System.out.println("Sumair is sleeping");
Thread.sleep(3000);
System.out.println("Please do not disturb.....");
}

void test() throws InterruptedException {
studentDetails();
}

void company() throws InterruptedException {
test(); } }
```

Throw :

- is used in the method body to throw an exception
- throw is followed by an instance variable
- you cannot declare multiple exceptions with throw
- The throw keyword is used to handover the instance of the exception created by the programmer to the JVM manually.
- throw keyword is mainly used to throw custom exceptions.

Example

```
class Test {
public static void main(String args[]) {
licenseAge(15);
}

public static void licenseAge(int age) {
try {
if (age >= 18) {
System.out.println("You are eligible for driver license");
} else {
throw new ArithmeticException("Person is less than 18 years old and not eligible to drive the car"); }
} catch (Exception e) {
 System.out.println(e.getMessage());
```

} }

32. What is the difference between final,finally and finalize?

Final keyword:

- **Used to apply restrictions on class, methods, and variables.**
- Used to declare constant values. The variable declared as final should be **initialized** only once and cannot be changed.
- Used to prevent **inheritance**. Java classes declared as **final cannot be extended**.
- Used to prevent method **overriding**. Methods declared as final **cannot be overridden**.

Example 1:

final int b=30;
b=37; //cannot change the value of final variable

Example 2:

public **final class Test** {
public static void main(String args[]) {

System.out.println("I am parent");
}

//you will get an error "Cannot subclass final class"
public class Child **extends Test**{
 public static void main(String[] args) {
System.out.println("I am a child");
 } }

Example 3:

public class Test {
public final void testFinalKey() {
System.out.println("Parent final method");
} }

//you will get an error "**Cannot override the final method**"

public class Child extends Test{
public void testFinalKey() {

```
System.out.println("Child final method");
}
}
```

Finally block :

● The finally block **always** executes when the try block exits. This ensures that the finally block is executed even if an unexpected exception occurs. We can have **multiple catches** but we can have only **one finally block**.

```
try {
Properties prop = new Properties();
FileInputStream fis = new FileInputStream("FilePath");
prop.load(fis);
} catch (Exception e) {
System.out.println("I am an exception block");
} finally {
System.out.println("I am final block");
System.out.println("Running script after exception");
}
```

Finalize() method :

● finalize() is a protected method of java.lang.object class and it is inherited by every class we create in java.
● finalize() method is used to perform some clean up operations on an object before it is removed from memory.
NOTE: we need to override the implementation of finalize method from object class, if we want to execute the specific code after the object is garbage collected.

33. What is the difference between String and StringBuffer? String and StringBuilder? What is mutable and immutable? StringBuffer vs StringBuilder?

The most important difference between **String and StringBuffer** in java is that **String object is immutable** whereas **StringBuffer object is mutable.** Once a String Object is created **we cannot change** it and everytime we change the value of a String there is actually a new String Object getting created.

For example we cannot reverse string directly, only through using StringBuffer class.
There are **2 ways to make String mutable:**

1. by using **StringBuffer**
2. by using **StringBuilder**.

The StringBuffer and StringBuilder Class are mutable means we can change the value of it without creating a new Object. Objects of StringBuilder and StringBuffer Classes live inside **heap memory.**

immutability vs. mutability

✓ String is immutability class it means once we are creating String objects it is not possible to perform modifications on existing objects. (String object is fixed object)

✓ StringBuffer and StringBuilder are mutable classes. It means once we are creating StringBuffer/ StringBuilder objects on that existing object it is possible to perform modification.

```
StringBuilder car=new StringBuilder();
  car.append("Honda");
  StringBuilder car1=car.append(" Civic");
  System.out.println(car);
  System.out.println(car1);
```

```
// OUTPUT:
Honda Civic
Honda Civic
```

- **Since only one object is created in the memory, the output will be Honda Civic for car and car1.**

StringBuffer vs StringBuilder?

Both Classes are mutable, except **StringBuffer is thread-safe (synchronized)** and **StringBuilder is not thread-safe (non synchronized)** which makes **StringBuilder faster compared to StringBuffer.**

34. What is singleton and have used the singleton concept in your project ?

A singleton class is a class that can have only one object (an instance of the class) at a time. After the first time, if we try to instantiate the Singleton class, the new variable also points to the first instance created. So whatever modifications we do to any variable inside the class through any instance, it affects the variable of the single instance created.

● Singleton pattern restricts the instantiation of a class and ensures that only one instance of the class exists in the Java virtual machine.

● The singleton class must provide a global access point to get the instance of the class.
● Singleton pattern is used for logging, drivers objects

Example:

```
public class SingletonExample {
//static member holds only one instance of the singleton class
private static SingletonExample singletonInstance;

//creating private constructor to prevent instantiation
private SingletonExample(){

}
//create public method to return an instance of the class
public static SingletonExample getInstance() {
singletonInstance=new SingletonExample();
return singletonInstance;
}
}
```

In my current project I do not use the concept of singleton class.

35. What is a garbage collector and what is a Garbage Collector?

Garbage collection is the process of looking at heap memory and identifying which objects are in use and which are not and deleting unused objects. Once an object is created it uses some memory and the memory remains allocated until there are references for the use of the object.

Since garbage collection is not guaranteed, we use **System.gc():** to call the garbage collector specifically.

36. What is Java regular expression?

Regular Expressions or Regex (in short) is an API for defining String patterns that can be used for searching, manipulating and editing a text.
Regular Expressions are provided under the java.util.regex package.

```
 String given="ertwsFADSF::IJ67585498testtest8732484375efds*&(&*^)5%^";

String replaced=given.replaceAll("[^A-Za-z]","");
```

37. What is the difference between pass-by-value and pass-by-reference?

Passing by value means that the value of the function parameter is copied into another location of your memory, and when accessing or modifying the variable within your function, only the copy is accessed/modified, and the original value is left untouched. Passing by value is how your values are passed on most of the time.

Passing by reference means that the memory address of the variable (a pointer to the memory location) is passed to the function. This is unlike passing by value, where the value of a variable is passed on. In the examples, the memory address of myAge is 106. When passing myAge to the function increaseAgeByRef, the variable used within the function (age in this example) still points to the same memory address as the original variable myAge (Hint: the & symbol in front of the function parameter is used in many programming languages to get the reference/pointer of a variable).

38. What is autoboxing and unboxing?

1. **Auto-boxing** **Autoboxing** is the automatic conversion that the **Java** compiler makes between the primitive types and their corresponding object wrapper classes. For example, converting an int to an Integer, a double to a Double, and so on
2. Converting an object of a wrapper type (Integer) to its corresponding primitive (int) value is called **unboxing**. The Java compiler **applies unboxing** when an object of a wrapper class is:

- Passed as a parameter to a method that expects a value of the corresponding primitive type.
- Assigned to a variable of the corresponding primitive type.

39. What's the difference between IS-A and HAS-A relationships?

- IS-A is based on inheritance"This thing is a type of that thing

- HAS-A relationships are based on usage

o Ex: class A HAS -A B if code in Class A has a reference to an instance of class B

- You are calling a Halter instance variable to use jump method that is coming from horse class - what this does is that it is means that Horse HAS-A Halter

- Horse class has a Halter, because Horse declares an instance variable of type Halter. When code invokes tie() on the Horse object's Halter instance variable -}
- Abstract class have constructors while interface don't have one

40. What is thread safe or Synchronized?

- Thread safety is very important, and it is the process to make our program safe to use in a multithreaded environment. There are different ways through which we can make our program thread safe.
- **Synchronization** is the easiest and most widely used tool for thread safety.
- JVM guarantees that synchronized code will be executed by only one thread at a time.
- JAVA keyword **synchronized** is used to create synchronized code and internally it uses locks on Object or Class to make sure only one thread is executing the synchronized code.
- I mean Java synchronization works on locking and unlocking of the resource, so no thread enters into synchronized code.
- We can use synchronized keyword in two ways, one is to make a complete method synchronized and another way is to create a synchronized block.

JAVA CODING INTERVIEW QUESTIONS

1. Write a program to swap 2 numbers without a temporary variable? Swap 2 strings without a temporary variable?

//swap numbers

int a=5;

```
int b=10;
 a=a+b;// first this should be there a=5+10=15
 b=a-b; // b= 15-10=5
 a=a-b; //a=15-5=10
System.out.println(a);
System.out.println(b);
```

//swap strings

```
String x="Hello";
String y="Techtorial";
x=x+y; //HelloWelcome
y=x.substring(0,(x).length()-y.length());
x=x.substring(y.length());
System.out.println(x);
System.out.println(y);
```

2. Write a java program to find the second largest number in the array? Maximum and minimum number in the array?

// second largest number in the array
1. easiest way

```
 int[] numArray= {12,13,12,15,0, -1};
 Arrays.sort(numArray);
 System.out.println(numArray[numArray.length-2]);
```

2. Interview solution

```
      int[] array = {100, 300, 200, 450,350};
     int largest = array[0];
     int secondLargest = 0;
       for (int i = 0; i < array.length; i++) {
           if (array[i] > largest) {
               secondLargest = largest;
               largest = array[i];
           } else if (array[i] > secondLargest && array[i] != largest) {
               secondLargest = array[i];
           }
```

```
        }
        System.out.println("The largest number=" + largest + " and secondLargest=" +
secondLargest);
```

3. Find out how many alpha characters present in a string?

```
 String given="ertwsFADSF::IJ67585498testtest8732484375efds*&(&*^)5%^";

 String replaced=given.replaceAll( "[^A-Za-z]"  ,  "" );
 System.out.println(replaced.length());
```

NOTE: replaceAll method takes the **regex** as a parameter. Once we use the regex A-Z or a-z finds all the alphabetical characters in the String. **'^' symbol** means '**not**'. In this example [^A-Za-z] replace all the non alphabetical characters with empty space " ".

4. What is the return type of split() method?

NOTE: Return type of split method is String array. We can use for loop or foreach loop to print the values from the array.

```
String techtorial="Techtorial interview preparation document. ";

String [] words=techtorial.split(" "); // will split the string with the spaces.
System.out.println(words.length);

for (String string : words) {
System.out.println(string);
}
```

5. Write a java program to reverse String? Reverse a string word by word?

Reverse String:
Using Reverse Function: StringBuffer

```
String word= "Hello Techtorial";

StringBuffer sb=new StringBuffer(word);
System.out.println(sb.reverse());
```

Using Reverse Function: StringBuilder

```
String word= "Hello Techtorial";
```

```
StringBuilder sb=new StringBuilder(word);
System.out.println(sb.reverse());
```

Without Using Reverse Function:

```
String toReverse="Hello Techtorial";

// 1 way using charAt();

String reversed="";
for (int i=toReverse.length()-1; i>=0; i--) {
reversed=reversed+toReverse.charAt(i);
}
System.out.println("Reversed String is: "+reversed);

//2 way using to charArray();

String reversed1="";
char[] array=toReverse.toCharArray();
for (int i=array.length-1; i>=0; i--) {
reversed1+=array[i];
}

System.out.println("Reversed String is: "+reversed1);

//3 way using substring();

String reversed2="";
for (int i=toReverse.length(); i>=1; i--) {
reversed2+=toReverse.substring(i-1, i);
}
System.out.println("Reversed String is: "+reversed2);

// 4 way using Java Recursion

public static String reverseString(String str){
        if (str.isEmpty())
           return str;
        //Calling Function Recursively
        return reverseString(str.substring(1)) + str.charAt(0);
}
```

```
//OUTPUT: lairothceT olleH
```

Reverse a string word by word

```
String str = "I love java";
String reversed = "";
String[] array = str.split(" ");

for (int i = array.length - 1; i >= 0; i--) {
reversed = reversed + array[i]+" ";
}
System.out.println(reversed); // output: java love I
```

******Reverse each word in the sentence******

```
 String str = "I love java";
        String reversed = "";
        String[] array = str.split(" ");
        for (int i=0;i<array.length;i++) {
            String word="";
            for(int j=array[i].length()-1;j>=0;j--){
                word+=array[i].charAt(j);
            }
            reversed = reversed + word+" ";
        }
        System.out.println(reversed); // OUTPUT: I evol avaj
```

6. Write a Java Program to find whether a String is palindrome or not.

```
String original="madam";

String reversed="";

for (int i =original.length()-1; i>=0; i--) {

reversed=reversed+original.charAt(i);

}

if (original.equals(reversed)) {

System.out.println("Given String is Palindrome");

}else {
```

```
System.out.println("Given String is NOT Palindrome");

}
```

7. Write a java program to check whether a given number is prime or not?

```
     int num = 29;
    boolean flag = false;
    for (int i = 2; i <= num / 2; ++i) {
      // condition for non prime number
      if (num % i == 0) {
        flag = true;
        break; }  }

    if (!flag){
      System.out.println(num + " is a prime number.");
    }else{
      System.out.println(num + " is not a prime number.");
  }
```

8. Write a Java Program to print the first 10 numbers of Fibonacci series.

```
int a=0,b=1,c;

for (int i=0; i<10; i++) {

System.out.print(a+" ");
c=a+b;
a=b;
b=c;
}
```

9. How can you remove all duplicates from ArrayList?

```
ArrayList aList=Arrays.asList(“John”, “Jane”, “James”, “Jasmine”, “Jane”, James”);

// 1 way
HashSet set=new HashSet(aList);

// 2 way
HashSet set=new HashSet();
for (Object name : arr) {
set.add(name);
}
```

```
System.out.println(hset);
```

10. How to sort the array without the sort method?

```
     int [] nums={3,7,6,2,9};

    for(int i=0;i<nums.length;i++){

        int temp=0;
        for(int k=i+1;k<nums.length;k++){
            if(nums[i]>nums[k]){
                temp=nums[i];
                nums[i]=nums[k];
                nums[k]=temp;
            }
        }
    }

    System.out.println(Arrays.toString(nums));
```

11. How to find count of each letter in the String?

```
public static Map<Character,Integer> countLetter(String str){
    Map<Character, Integer> result=new LinkedHashMap<>();
        // techtorial
        for(int i=0;i<str.length();i++){
            char ch=str.charAt(i);
            if(result.containsKey(ch)){
                int count=result.get(ch); //1
                result.replace(ch,++count);
            }else{
                result.put(ch,1);
            }
        }
        return result; }
```

12. How to reverse the array?

```
int nums [] ={45,62,7,67, 23,35,11};

    for(int i=0;i<nums.length/2;i++){//i=3
        int temp=nums[i]; //7
        nums[i]= nums[nums.length-i-1]; // 35
        nums[nums.length-i-1]=temp; // 7
    }
     System.out.println(Arrays.toString(nums));
```

13. How to find the missing number in an array?

```
// this example only for the numbers between 0 to 9 and only finds the one missing number

int [] arr={0,4,2,3,6,8,1,9,7};

  // find missing number from given array

 // 0, 1,2,3,4... 9

        Arrays.sort(arr);

        System.out.println(Arrays.toString(arr));

        int num=0;

        for(int i=0;i<arr.length;i++){

            if(arr[i]!=num){

                System.out.println(num);

                break;

            }

            num++;  }
```

14. How to find the perfect number?

Perfect Number: Perfect number, a positive integer that is equal to the sum of its proper divisors.

The smallest perfect number is 6, which is the sum of 1, 2, and 3. Other perfect numbers are 28, 496.

```
static boolean isPerfect(int n) {
   // To store sum of divisors
   int sum = 1;
   // Find all divisors and add them
   for (int i = 2; i * i <= n; i++)
   {
      if (n % i==0)
      {
         if(i * i != n)
            sum = sum + i + n / i;
         else
            sum = sum + i;
      }  }
   // If sum of divisors is equal to
   // n, then n is a perfect number
   if (sum == n && n != 1) {
      return true; }

   return false;
}
```

15. How to print the start tree?

```
 *
* *
```

```
    * * *
   * * * *
  * * * * *
int row=5;
for(int i=1;i<=row;i++){
    for(int k=1;k<=row-i;k++){
        System.out.print(" ");
    }
    for(int l=1;l<=i;l++){
        System.out.print("* ");
    }
```

16. How to find the sum of digits from a given number?

```
int number=4577;
int originalNum=number;
int sum=0;
// 4577- 457 - 45 - 4 - 0
while (number>0){
    sum=sum+(number%10);
    number=number/10;
}
System.out.println("The sum of "+originalNum+" is equals to "+sum);
```

OUTPUT: The sum of **4577** is equals to **23**

JAVA SHORT QUESTIONS

1- Tell me 5 String methods?

- charAt(indexNumber);
- indexOf(String);
- startsWith(String);
- endsWith(String);
- equals(String);

2- What is the return type split() method?

Answer: String array.

3- How to call the method without creating the object?

Answer: using static keyword

4- What is the return type of keyset() method?

Answer: It returns Set<ObjectType> of keys from the map.

5- What is the return type of entrySet() method?

Answer: Set<Map.Entry<Object, Object>>

6- Can we override the static method?

Answer: We can not override the final, private and static methods. We can hide the static method.

7- How do you create the object in java?

Answer: Using a 'new' keyword.

8 - Can you store the null values inside the HashTable?

Answer: We can not store null keys and values inside the HashTable.

9- Can you store the null values or keys inside the TreeMap?

Answer: We can not store the null keys but we can store the multiple null values.

10- How do you handle the exception in java?

Answer: Using try-catch block or declaring the exception in the method signature.

11- How do you convert a String to Wrapper class object?

Answer: Using valueOf method.

12- How do you convert a String to primitive data type?

Answer: Using parse methods like parseInt(), parseDouble, parseLong() etc.

13- How do you convert a primitive data type to String?

Answer: Using valueOf method from String class or using concatenation. For Example:

String.valueOf('c');   or   String str= " "+'c';