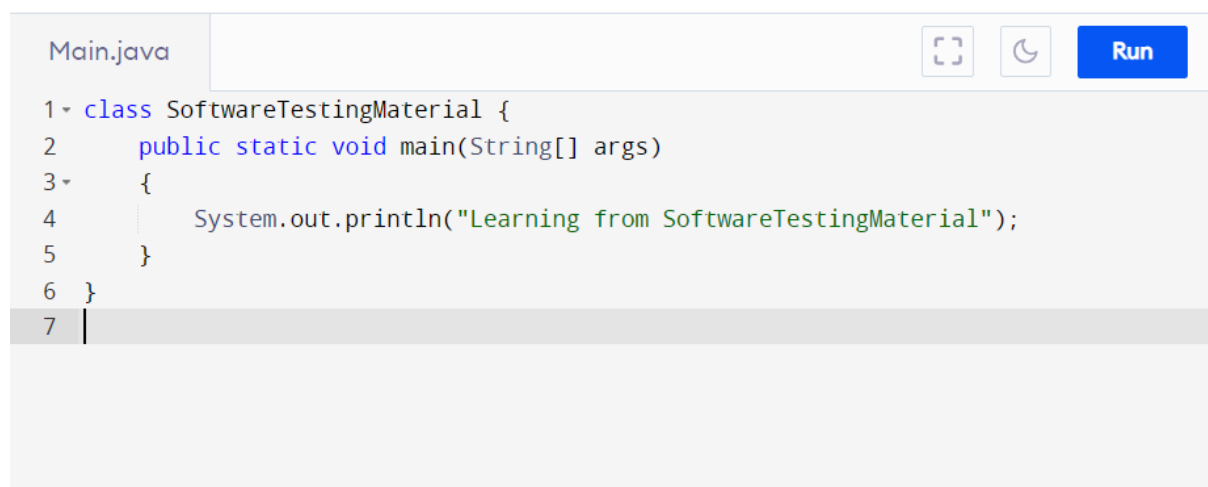


## 1. Explain Java Main Method `public static void main (String[] args)`

When you start [learning Java](#), the first method you encounter is ***public static void main(String [] args)***.

The starting point of any Java Program is the main() method. It is one of the important methods of Java. Technically, the main method is the starting point where the Java program starts its execution. JVM always look for this method signature to start running an application.

***Syntax of the main method is as follows***

A screenshot of a Java IDE window titled 'Main.java'. The code inside is as follows:

```
1 class SoftwareTestingMaterial {  
2     public static void main(String[] args)  
3     {  
4         System.out.println("Learning from SoftwareTestingMaterial");  
5     }  
6 }  
7 |
```

The IDE has a toolbar at the top right with icons for a code editor, a moon icon, and a blue 'Run' button.

**Note:** `public static void main(String[] args)` can also be written as `public static void main(String args[])`. Don't get confused.

Each word has its purpose. Let's break the method signature and see in detail.

### **Public:**

Public is an access modifier. The scope of public access modifier is everywhere. It has no restrictions. Data members, methods and classes that declared public can be accessed from anywhere. If you make a main () method public then it is allowed to be executed by any program globally. If you make a main () method non-public then it is not allowed to be executed by any program.

### **Static:**

JVM cannot create an object of class at run time to access the main method. By declaring the main() method as static, JVM can invoke it without instantiating the class.

If we don't declare the main method as static then JVM cannot call it to execute the program.

### **Void:**

Void means the Method will not return any value. In Java, every method provides the return type whereas Java main method doesn't return any value. Java program starts with main method and terminates once the main method is finished executing. If we make main method to return a value, JVM cannot do anything with the returned value. So there is no need to return any value.

### **main:**

main is the name of Java main method. It is the first thing that runs when you compile and run the Java program. The main method is searched by JVM as the starting point where the Java program starts its execution.

### **String[] args**

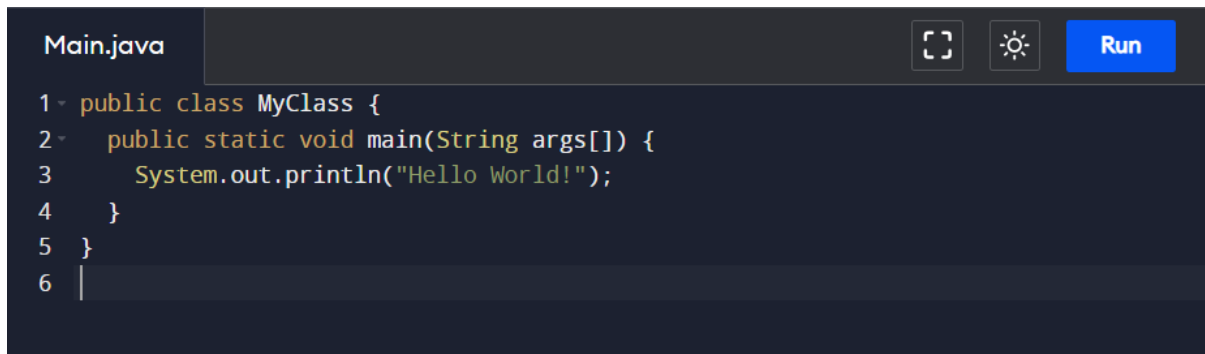
String[] args is a parameter that accepts inputs. The name of the String array is '**args**' but it can be of anything based on users choice. The type of args is always String[] because Java main method accepts only a single argument of the type String array. We can replace the array name with anything then also it works fine. i.e. **String[] raj** or **String[] stm**

.

## **2. What is Java?**

Java is a programming language and computing platform first released by Sun Microsystems in 1995. There are lots of applications and websites that will not work unless you have Java installed, and more are created every day. Java is fast, secure, and reliable. From laptops to datacentres, game consoles to scientific supercomputers, cell phones to the Internet, Java is everywhere!

Java program which prints Hello World!



```
Main.java
1 public class MyClass {
2     public static void main(String args[]) {
3         System.out.println("Hello World!");
4     }
5 }
6 |
```

### 3. Mention some features of Java?

Some of the features which play an important role in the popularity of java are as follows:

- **Simple:** Java is easy to learn. Even though Java is based on C++, it was developed by eliminating poor programming practices of C++.
- **Object-Oriented:** Java is an object-oriented programming language. Everything in Java is an Object.
- **Portable:** Java run time environment uses a bytecode verification process to make sure that code loaded over the network doesn't violate Java security constraints.
- **Platform independent:** Java is platform-independent. Java is a write once, run anywhere language. Without any modifications, we can use a program on different platforms.
- **Secured:** Java is well known for its security. It delivers virus-free systems.
- **High Performance:** Java enables high performance with the use of JIT (Just-In-Time) compilers
- **Multithreaded:** Java multithreaded features allows us to write programs that can perform many tasks simultaneously. The multithreading concept of Java shares a common memory area. It doesn't occupy memory for each thread.

### 4. Is Java 100% Object Oriented Language?

Java is not a pure Object Oriented Language because it supports primitive data type such as byte, boolean , char, double, float, int, long, short. These primitive data types are not object oriented. This is the reason why Java is not 100% object-oriented language.

### 5. What is the difference between Object-oriented programming language and Object-based programming language?

There is a difference between Object Oriented Languages and Object Based languages.

**Object Oriented Languages:**

- Some of the Object Oriented Languages are Java, C#, VB. Net, Smalltalk etc.,
- These languages support all the concepts of OOPs.
- These languages don't have the inbuilt objects.

### Object Based Languages:

- Some of the Object Based Languages are JavaScript, VBScript etc.,
- These languages support all the concepts of OOPs like inheritance and polymorphism.
- These languages have the inbuilt objects, say JavaScript has window object.

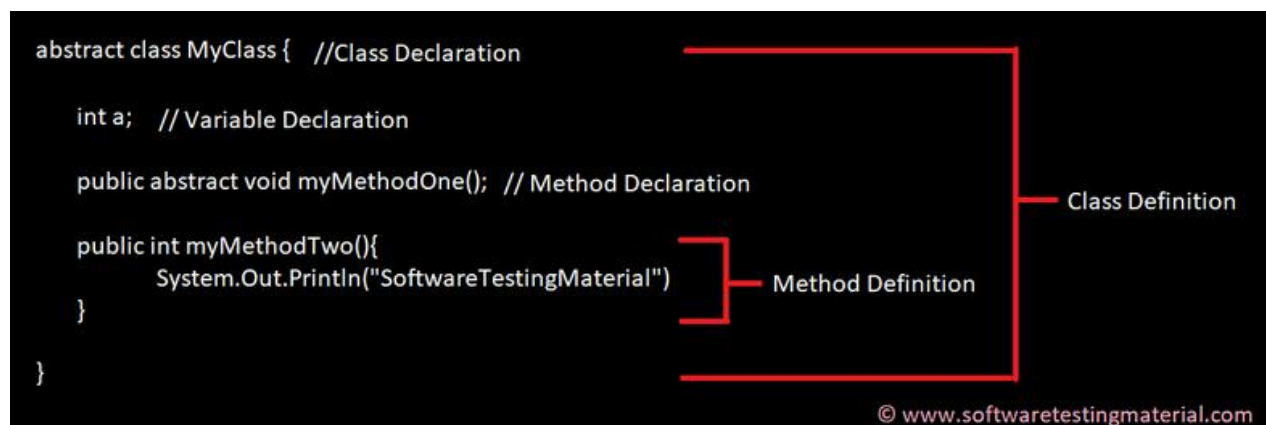
## 6. What is the difference between Declaration and Definition in Java?

**Declaration:** If you just declare a class or method/function or variable without mentioning anything about what that class or method/function or variable looks like is called a declaration in Java.

**Definition:** If you define how a class or method/function or variable is implemented then it is called definition in Java.

When we create an interface or abstract class, we simply declare a method/function but not define it.

For a clear understanding, check the below image



## 7. What is JRE, and why is it required?

JRE stands for "Java Runtime Environment". It comprises of the JVM (Java Virtual Machine), Java platform classes, and supporting libraries. Using JRE, we can only execute already developed applications. We cannot develop new applications or modify existing applications. As the name suggests, JRE only provides Runtime Environment.

## 8. What is JDK, and why is it required?

JDK stands for Java Development Kit. It is a superset of JRE (Java Runtime Environment). Using JDK, we can develop, compile and execute (run) new applications and also we can modify existing applications. We need to install JDK in developers machine where we want to develop new applications or modify existing applications. JDK includes JRE and development tools (environment to develop, debug and monitor Java programs).

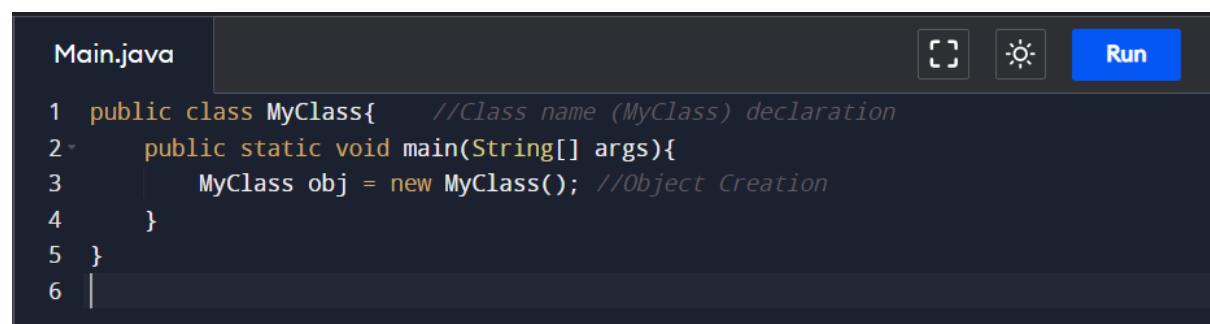
## 9. What is JVM, and why is it required?

JVM stands for Java Virtual Machine. JVM drives the java code. Using JVM, we can run java byte code by converting them into current OS machine language. It makes Java to become a portable language (write once, run anywhere)

## 10. What is an Object in Java?

An object is an instance of a class. Objects have state (variables) and behavior (methods).

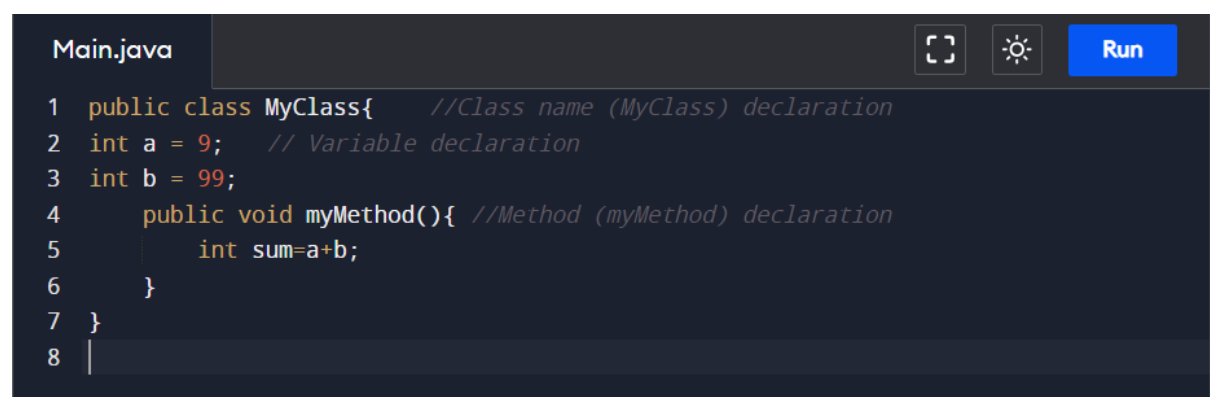
Example: A dog is an object of Animal class. The dog has its states such as color, name, breed, and behaviors such as barking, eating, wagging her tail.



```
Main.java  [Full Screen] [Settings] [Run]
1 public class MyClass{    //Class name (MyClass) declaration
2     public static void main(String[] args){
3         MyClass obj = new MyClass(); //Object Creation
4     }
5 }
6 |
```

## 11. What is a Class in Java?

A class can be defined as a collection of objects. It is the blueprint or template that describes the state and behavior of an object.



```
Main.java  [Full Screen] [Settings] [Run]
1 public class MyClass{    //Class name (MyClass) declaration
2     int a = 9;    // Variable declaration
3     int b = 99;
4     public void myMethod(){ //Method (myMethod) declaration
5         int sum=a+b;
6     }
7 }
8 |
```

## 12. What is Constructor in Java?

Constructor in Java is used in the creation of an Object that is an instance of a Class. The constructor name should be the same as the class name. It looks like a method but it's not a method. It won't return any value. We have seen that methods may return a value. If there is no constructor in a class, then the compiler automatically creates a default constructor.

## 13. What is Local Variable and Instance Variable?

### Local Variable:

A local variable is a variable that we declare inside a Method. A method will often store its temporary state in local variables.

It can be accessible only inside a block, function, or constructor.

#### Main.java

```
1 public void website() {  
2     String websiteName;  
3     double websiteLoadTime;  
4     int webisteAge;  
5 }  
6 |
```

String websiteName, double websiteLoadTime, int websiteAge are Local variables in above example.

### Instance Variable (Non-static):

An instance variable is a variable that is declared inside a Class but outside a Method. We don't declare this variable as Static because these variables are non-static variables.

It can be accessible by all the methods in the class.

```
Main.java
1 class website() {
2     public String websiteName;
3     public double websiteLoadTime;
4     public int webisteAge;|
5 }
6
```

websiteName, websiteLoadTime, websiteAge are Instance variables in above example.

## 14. What are the OOPs concepts?

OOPS Stands for Object-Oriented Programming System. It includes Abstraction, Encapsulation, Inheritance, Polymorphism, Interface, etc.,

### #1. ABSTRACTION

Abstraction is the methodology of hiding the implementation of internal details and showing the functionality to the users.

Let's see an example of data abstraction in Selenium Automation Framework.

In Page Object Model design pattern, we write locators (such as id, name, xpath etc.,) and the methods in a Page Class. We utilize these locators in tests but we can't see the implementation of the methods. Literally we hide the implementations of the locators from the tests.

#### [Abstraction in Java](#)

Abstraction in Java is a methodology of hiding the implementation of internal details and showing the functionality to the users.

**Example:** Mobile Phone.



A layman who is using mobile phone doesn't know how it works internally but he can make phone calls.

Abstraction in Java is achieved using abstract classes and interfaces. Let's see what is Abstract Class and Interface in detail.

### **Abstract Class:**

We can easily identify whether a class is an abstract class or not. A class which contains abstract keyword in its declaration then it is an Abstract Class.

Syntax:

```
abstract class <class-name>{}
```

### **Points to remember:**

1. Abstract classes may or may not include abstract methods
2. If a class is declared abstract then it cannot be instantiated.
3. If a class has abstract method then we have to declare the class as abstract class
4. When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared abstract.



## Abstract Method:



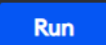
An abstract method is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

```
abstract void myMethod();
```

In order to use an abstract method, you need to override that method in sub class.




Let's see an example:

To create an abstract class, just use the **abstract** keyword before the class keyword, in the class declaration.

```
Main.java   
```

```
1 package abstractClass;
2
3 // Here class is abstract
4 public abstract class AbstractSuperClass {
5
6     // myMethod() is an abstract method
7     abstract void myMethod();
8
9 }
10 |
```

Let's try to instantiate the AbstractSuperClass class in the following way

```
Main.java   
```

```
1 package abstractClass;
2
3 public class AbstractChildOneClass{
4
5     public static void main (String [] args){
6
7         AbstractSuperClass obj = new AbstractSuperClass();
8
9         obj.myMethod();
10
11     }
12
13 }
14 |
```

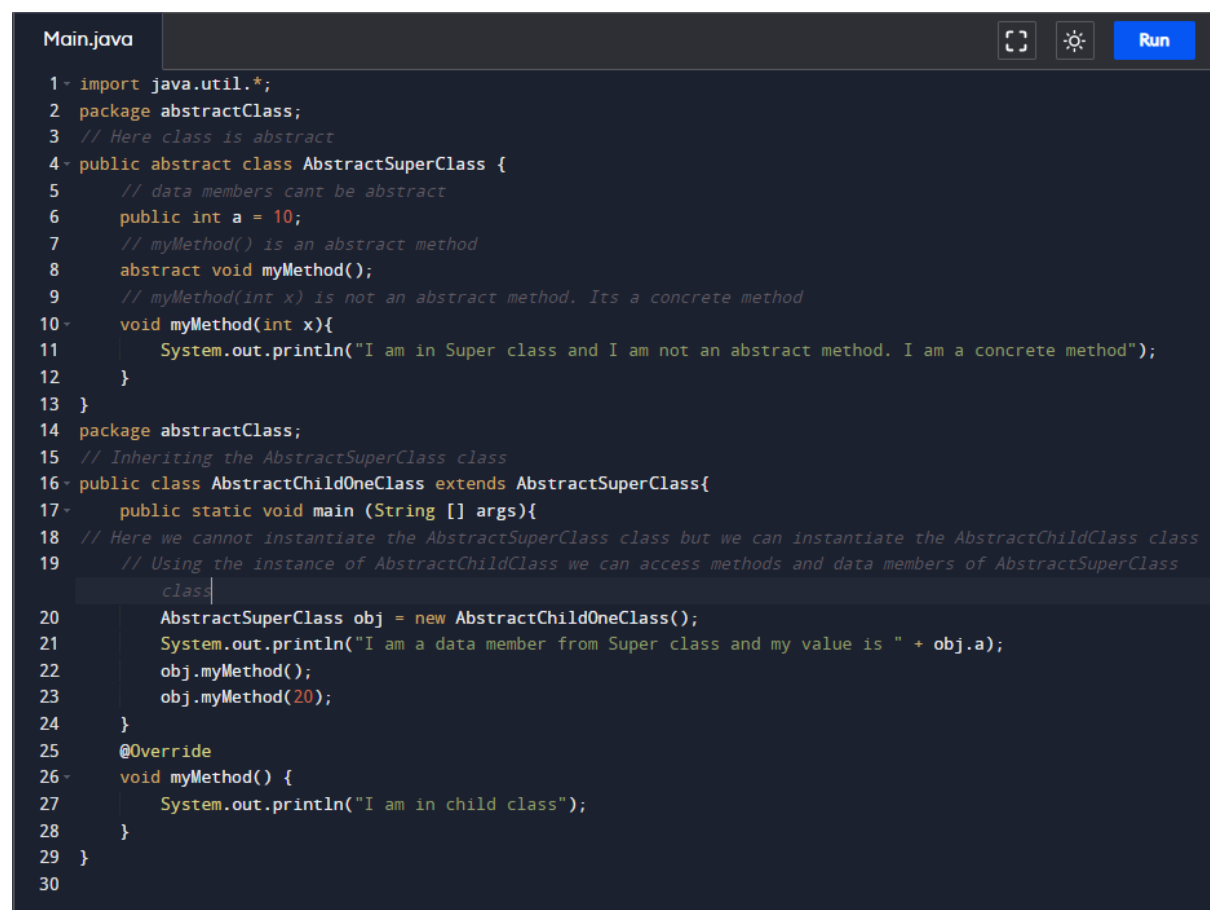
When you compile the above class, the output will be as follows

### Output:

Exception in thread "main" java.lang.Error: Unresolved compilation problem:  
Cannot instantiate the type AbstractSuperClass  
  
at abstractClass.AbstractChildOneClass.main(AbstractChildOneClass.java:7)

Now let's Inherit the abstract class and see.

### Inheriting the Abstract Class



```
Main.java
1- import java.util.*;
2 package abstractClass;
3 // Here class is abstract
4- public abstract class AbstractSuperClass {
5     // data members cant be abstract
6     public int a = 10;
7     // myMethod() is an abstract method
8     abstract void myMethod();
9     // myMethod(int x) is not an abstract method. Its a concrete method
10-    void myMethod(int x){
11        System.out.println("I am in Super class and I am not an abstract method. I am a concrete method");
12    }
13 }
14 package abstractClass;
15 // Inheriting the AbstractSuperClass class
16- public class AbstractChildOneClass extends AbstractSuperClass{
17-    public static void main (String [] args){
18 // Here we cannot instantiate the AbstractSuperClass class but we can instantiate the AbstractChildClass class
19 // Using the instance of AbstractChildClass we can access methods and data members of AbstractSuperClass
20        class {
21            AbstractSuperClass obj = new AbstractChildOneClass();
22            System.out.println("I am a data member from Super class and my value is " + obj.a);
23            obj.myMethod();
24            obj.myMethod(20);
25        }
26        @Override
27        void myMethod() {
28            System.out.println("I am in child class");
29        }
30    }
```

### Output:

I am a data member from Super class and my value is 10  
I am in child class  
I am in Super class and I am not an abstract method. I am a concrete method

We use abstraction when we know that our class should have some methods but we are not sure how exactly those methods should function. Assume, I am creating a class of Vehicle which should have a method called start(). There will be some other subclass of this Vehicle class such as Car, Bike and these two subclasses use start() method. But the implementation of start() method in Car is different from Bike. So in this case I don't implement the start() method in Vehicle class and implement those in subclasses.

In the above program, we have seen abstract method in the abstract class. Abstract classes don't give 100% abstraction since abstract class allows concrete methods. With abstract class we can achieve partial abstraction whereas we can achieve 100% abstraction with an interface which we see in the Interface section.

In Java, abstraction is achieved by interfaces and abstract classes. Using interfaces, we can achieve 100% abstraction.

Let's see interface concept below.

## **INTERFACE**

Earlier we have learnt abstract class. We learnt that with abstract class we can achieve partial abstraction and with interface we can achieve 100% abstraction. Let's see how we can achieve 100% abstraction with Interface in this post.

Abstraction is a methodology of hiding the implementation of internal details and showing the functionality to the users.

An interface in Java looks similar to a class but both the interface and class are two different concepts. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract. We can achieve 100% abstraction and multiple inheritance in Java with Interface.

### **Points to remember:**

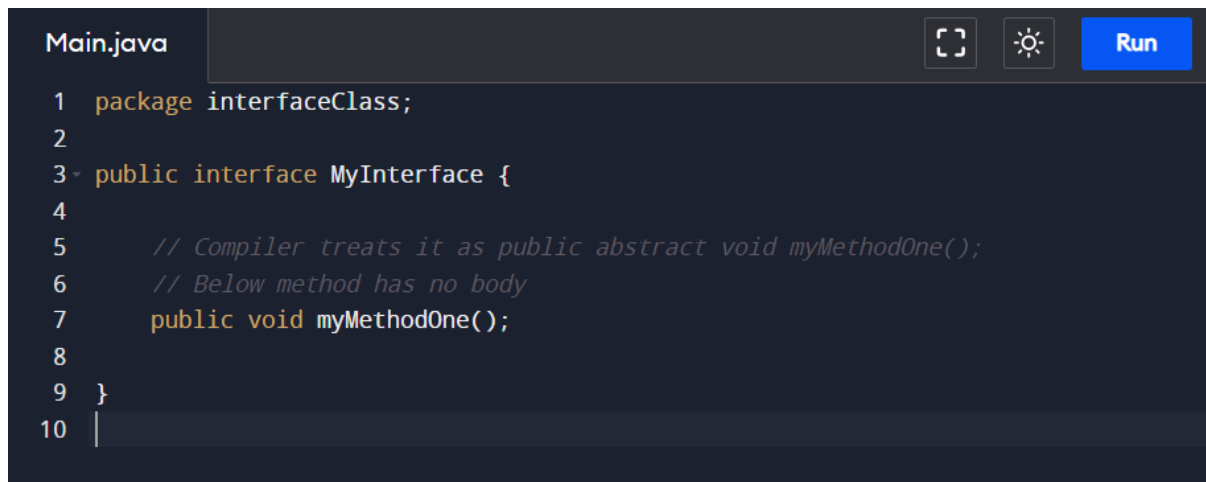
1. Java interface represents IS-A relationship similar to Inheritance
2. Interface cannot be instantiated same like abstract class
3. Java compiler adds public and abstract keywords before the interface methods
4. Java compiler adds public, static and final keywords before data members
5. Interface extends another interface just like a Class extends another Class but a class implements an interface.
6. The class that implements interface must implement all the methods of that

interface.

7. Java allows you to implement more than one interface in a Class

Let's see some example program:

To create an interface – Right click on your project – New – Interface



```
Main.java
1 package interfaceClass;
2
3 public interface MyInterface {
4
5     // Compiler treats it as public abstract void myMethodOne();
6     // Below method has no body
7     public void myMethodOne();
8
9 }
10 |
```

Now create a class and implements it with the above interface



```
Main.java
1 package interfaceClass;
2
3 public class MyClass implements MyInterface{
4
5
6     public static void main(String [] args){
7         MyInterface obj = new MyClass();
8         obj.myMethodOne();
9     }
10
11     // Try to comment the below method myMethodOne and see. You will face a compilation error.
12     // As per the rule, this class must implement the abstract method of interface
13
14     @Override
15     public void myMethodOne() {
16
17         System.out.println("Implementation of myMethodOne");
18     }
19
20 }
21
22 |
```

Interface extends another interface but interface cannot implement another interface.

Now let's see how interface extends another interface by creating two interfaces. If an interface (say MyInterfaceTwo) extends another interface (MyInterfaceOne). Now if a Class implements the interface InterfaceTwo then this class has to provide

implementation of all the methods of both interfaces (MyInterfaceOne and MyInterfaceTwo)

To create an interface – Right click on your project – New – Interface

```
Main.java
1 package interfaceClass;
2
3 public interface MyInterfaceOne {
4
5     // Compiler treats it as public abstract void myMethodOne();
6     // Below method has no body
7     public void myMethodOne();
8
9 }
10 |
```

Let's create another interface

```
Main.java
1 package interfaceClass;
2
3 public interface MyInterfaceTwo extends MyInterfaceOne{
4
5     // Compiler treats it as public abstract void myMethodTwo();
6     // Below method has no body
7     public void myMethodTwo();
8 }
9 |
```

Let's create a Class

```
Main.java
1 package interfaceClass;
2
3 public class MyClass implements MyInterfaceTwo{
4     public static void main(String [] args){
5         MyInterfaceTwo obj = new MyClass();
6         obj.myMethodTwo();
7     }
8     // If you comment below two methods, you can see a compilation error
9     /*This class is just implementing MyInterfaceTwo but
10     it has to implement all the methods of MyInterfaceTwo and MyInterfaceOne as well
11     because MyInterfaceTwo extends MyInterfaceOne*/
12
13     @Override
14     public void myMethodOne() {
15         System.out.println("Implementation of myMethodOne");
16     }
17     @Override
18     public void myMethodTwo() {
19
20         System.out.println("Implementation of myMethodTwo");
21     }
22 }
23
```

Output:

Implementation of myMethodTwo

Automation example of interface

Basic statement we all know in Selenium is **WebDriver driver = new FirefoxDriver();**

**Detailed explanation on why we write [WebDriver driver = new FirefoxDriver\(\);](#) in Selenium.**

WebDriver itself is an Interface. So based on the above statement **WebDriver driver = new FirefoxDriver();** we are initializing Firefox 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* as mentioned earlier and *FirefoxDriver* is a *class*.

An interface in Java looks similar to a class but both the interface and class are two different concepts. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract. We can achieve 100% abstraction and multiple inheritance in Java with Interface.

.

## 2. INHERITANCE

The mechanism in Java by which one class acquires the properties (instance variables) and functionalities of another class is known as Inheritance.

We create a Base Class in the Automation Framework to initialize WebDriver interface, WebDriver waits, Property files, Excels, etc., in the Base Class.

We extend the Base Class in other classes such as Tests and Utility Class.

Here we extend one class (Base Class like WebDriver Interface) into other class (like Tests, Utility Class) is known as Inheritance..

## 3. POLYMORPHISM

Polymorphism allows us to perform a task in multiple ways.

Combination of overloading and overriding is known as Polymorphism. We will see both overloading and overriding below.

### #1. METHOD OVERLOADING

We use **Implicit wait** in Selenium. Implicit wait is an example of overloading. In Implicit wait we use different time stamps such as SECONDS, MINUTES, HOURS etc.,

**Action class** in TestNG is also an example of overloading.

**Assert class** in TestNG is also an example of overloading.

A class having multiple methods with same name but different parameters is called Method Overloading

### #2. METHOD OVERRIDING

We use a method which was already implemented in another class by changing its parameters. To understand this you need to understand Overriding in Java.

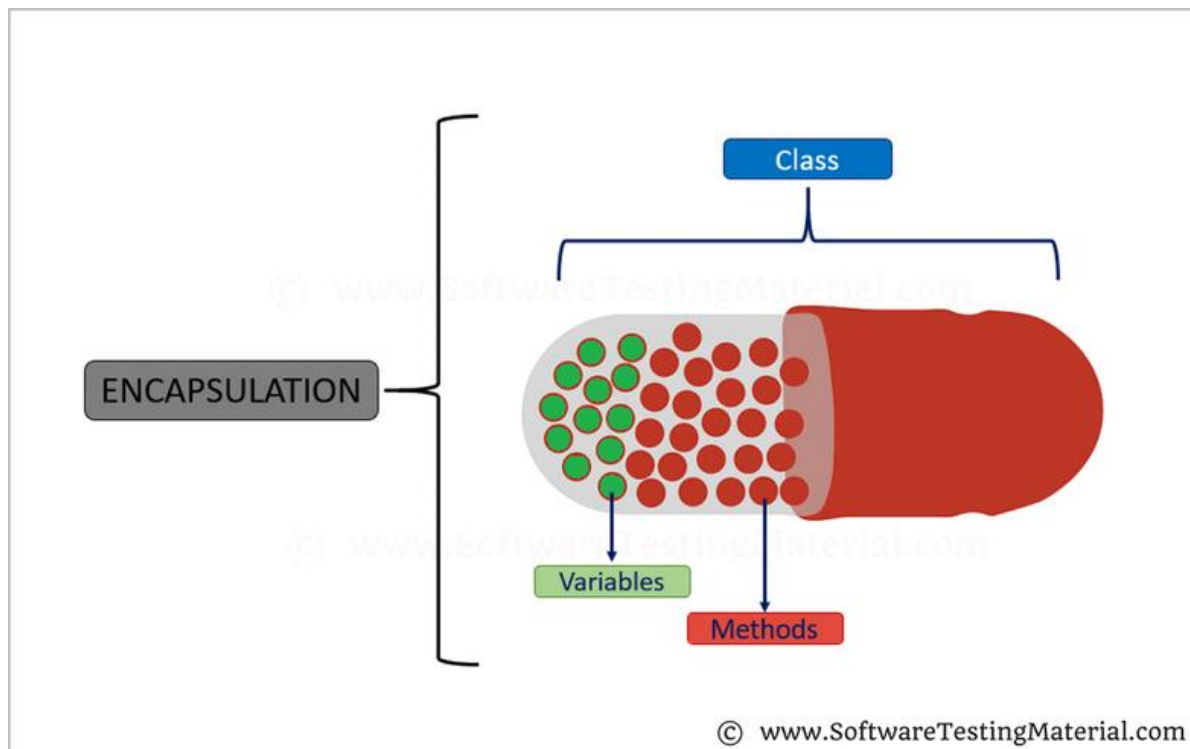
Declaring a method in child class which is already present in the parent class is called Method Overriding. Examples are **get** and **navigate** methods of different drivers in Selenium .

## #4. ENCAPSULATION

All the classes in a framework are an example of Encapsulation. In POM classes, we declare the data members using **@FindBy** and initialization of data members will be done using [Constructor](#) to utilize those in methods.

Encapsulation is a mechanism of binding code and data (variables) together in a single unit.

Encapsulation is a mechanism of binding code and data together in a single unit. Let's take an example of Capsule. Different powdered or liquid medicines are encapsulated inside a capsule. Likewise in encapsulation, all the methods and variables are wrapped together in a single class.



We will see detailed explanation with some example programs about Encapsulation in the post related to Encapsulation.

Let's see how can we implement encapsulation. Set the instance variables private so that these private variables cannot be accessed directly by other classes. Set getter and setter methods of the class as public so that we can set and get the values of the fields.

Let's see a sample program.

```
package encapsulationClass;  
  
public class EncapsulationClassOne {  
  
    // Variables declared as private
```



```

// These private variables can only be accessed by public methods of class
    private int age;
    private String name;

// getter method to access private variable
    public int getAge(){
        return age;
    }

    public String getName(){
        return name;
    }

// setter method to access private variable
    public void setAge(int inputAge){
        age = inputAge;
    }

    public void setName(String inputName){
        name = inputName;
    }
}

```

```

package encapsulationClass;

```

```

public class EncapsulationClassTwo {

    public static void main(String [] args){

        EncapsulationClassOne obj = new EncapsulationClassOne();
        // Setting values of the variables
        obj.setAge(25);
        obj.setName("Rajkumar");

        System.out.println("My name is "+ obj.getName());
        System.out.println("My age is "+ obj.getAge());

    }

}

```

Output:

My name is Rajkumar

My age is 25

In the above example, you can find all the data member (variables) are declared as private. If the data member is private it means it can only be accessed within the same class. No other class can access these private variables of other class. To access these private variables from other classes, we used public getter and setter methods such as `getAge()`, `getName()`, `setAge()`, `setName()`. So, the data can be accessed by public methods when we can set the variables private and hide their implementation from other classes. This way we call encapsulation as data hiding

## 15. What is Inheritance in Java?

Inheritance is a process where one class inherits the properties (methods & fields) of another class.

Inheritance is a process where one class inherits the properties of another class.



Let's say we have two classes namely Parent Class and Child Class. The child class is also known as Derived Class. As per the above definition, the Child class inherits the properties of the Parent Class. The main purpose of Inheritance is to obtain Code Reusability. We can achieve run time polymorphism with inheritance. We will see what is run-time polymorphism in a few minutes.

Assume we have a Class named Laptop, Apple MacBook Pro, Lenovo Yoga. Apple MacBook Pro and Lenovo Yoga classes extend the Laptop Class to inherit the properties of the Laptop Class.

We will see a detailed explanation with some example programs about Inheritance in the post related to Inheritance.

In inheritance, we use two keywords namely **extends** and **implements**

### **Extends:**

We use the extends keyword in Java to allow the child class to inherit all the properties (data members and methods) of the parent class and in addition to these, we can also create new data members and methods. If the properties are private then the child class cannot inherit those properties from the parent class.

```
class ChildClass extends ParentClass
{
}
```

Technically, we say that child class has an **IS-A** relationship with the parent class

```
class QA extends Employee
{
}
```

As per the IS-A relationship, we can say QA **IS-A** Employee

### **Employee Class:**

```
package classFourOops.inheritance;
```

```
public class Employee {

    String name = "Rajkumar";

}
```

### **QA Class:**

```

package classFourOops.inheritance;

public class QA extends Employee{

    String fullName = "Rajkumar SM";

    public static void main(String [] args){

        QA objName = new QA();

        System.out.println(objName.name);
        System.out.println(objName.fullName);

    }

}

```

In the above example, the QA object can access the properties of its own as well as the Employee class.

### **Implements:**

We use implements keyword in Java to inherit the properties from an interface. Interfaces cannot be extended by the classes.

We will see this in the Interface section.

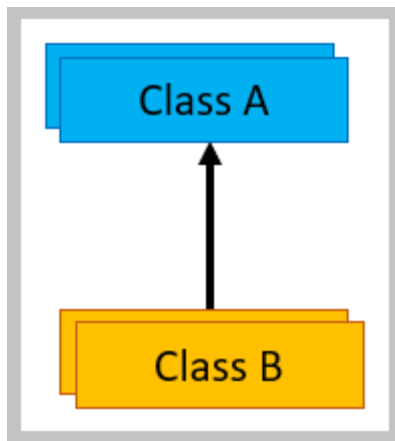
Types of Inheritance:

- |    |                          |             |
|----|--------------------------|-------------|
| 1. | Single                   | Inheritance |
| 2. | Multilevel               | Inheritance |
| 3. | Hierarchical Inheritance |             |

There are two more inheritances as Multiple and Hybrid Inheritance which are supported through interface only. We will see those in the Interface section.

### **Single Inheritance:**

Single Inheritance in Java refers to a child and parent class relationship. In this one class extends another class say Class B extends Class A



```
package classFourOps.singleInheriance;
```

```
public class ClassA {
```

```
    void methodOneClassA(){
        System.out.println("I am a Method One of ClassA");
    }
```

```
}
```

```
package classFourOps.singleInheriance;
```

```
public class ClassB extends ClassA{
```

```
    void methodOneClassB(){
        System.out.println("I am a Method One of ClassB");
    }
```

```
}
```

```
package classFourOps.singleInheriance;
```

```
public class InheritanceTest {
```

```
    public static void main(String args[]){
```

```
        // Class B extends Class A
        // Here I am creating an instance of ClassB
```

```
        ClassB obj = new ClassB();
```

```
        // Using object of ClassB, I can call methods of ClassA and ClassB.. Its
just because ClassB extends ClassA
```

```
        obj.methodOneClassA();
        obj.methodOneClassB();
```

```
    }  
}
```

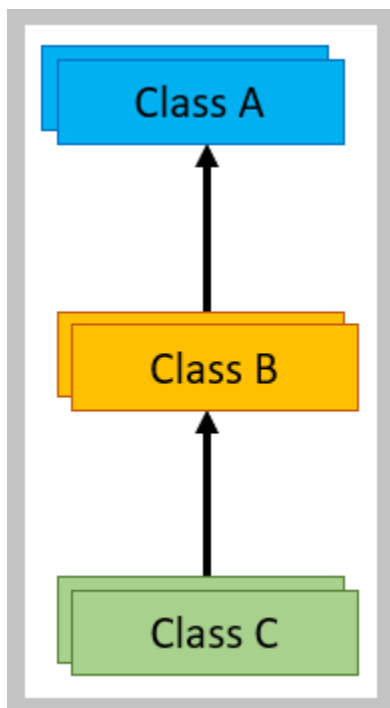
Output:

I am a Method One of ClassA

I am a Method One of ClassB

### **Multilevel Inheritance:**

Multilevel Inheritance in Java refers to a child and parent class relationship. In this a class extends a child class say Class C extends Class B and Class B extends Class A



```
package classFourOps.multipleInheritance;
```

```
public class ClassA {
```

```
    void methodOneClassA(){  
        System.out.println("I am a Method One of ClassA");  
    }
```

```
}
```

```
package classFourOps.multipleInheritance;
```

```
public class ClassB extends ClassA{
```

```

        void methodOneClassB(){
            System.out.println("I am a Method One of ClassB");
        }
    }
package classFourOps.multipleInheriance;

public class ClassC extends ClassB{

    void methodOneClassC(){
        System.out.println("I am a Method One of ClassC");
    }
}
package classFourOps.multipleInheriance;

public class InheritanceTest {
    public static void main(String args[]){

        // Class B extends Class A
        // Class C extends Class B
        // Here I am creating an instance of ClassC

        ClassC obj = new ClassC();

        // Using object of ClassC, I can call methods of ClassA, ClassB and
ClassC..

        // Its just because ClassB extends ClassA and ClassC extends ClassB

        obj.methodOneClassA();
        obj.methodOneClassB();
        obj.methodOneClassC();

    }
}

```

Output:

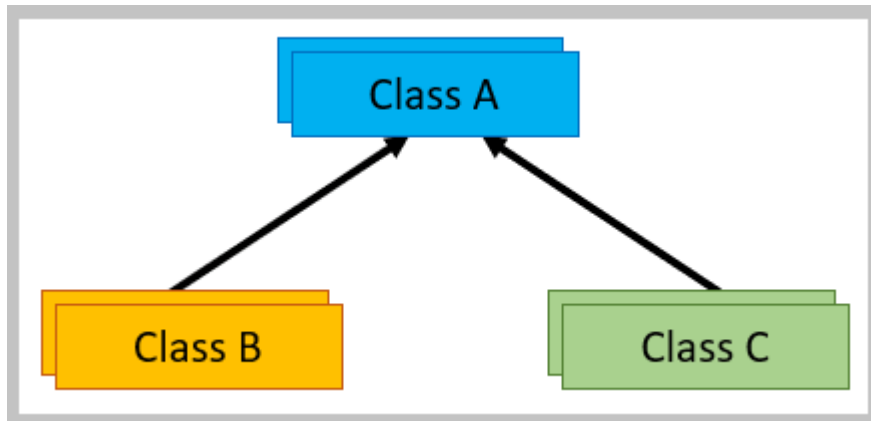
```

I am a Method One of ClassA
I am a Method One of ClassB
I am a Method One of ClassC

```

### **Hierarchical Inheritance:**

Hierarchical Inheritance in Java refers to a child and parent class relationship. In this more than one class extends the same class say Class B, Class C extends Class A



```
package classFourOps.hierarchialInheriance;

public class ClassA {

    void methodOneClassA(){
        System.out.println("I am a Method One of ClassA");
    }

}

package classFourOps.hierarchialInheriance;

public class ClassB extends ClassA{

    void methodOneClassB(){
        System.out.println("I am a Method One of ClassB");
    }

}

package classFourOps.hierarchialInheriance;

public class ClassC extends ClassA{

    void methodOneClassC(){
        System.out.println("I am a Method One of ClassC");
    }

}

package classFourOps.hierarchialInheriance;

public class InheritanceTest {
    public static void main(String args[]){
```



```

// Class B extends Class A
// Class C extends Class A
// Here I am creating an instance of ClassC

ClassC obj = new ClassC();

// Using object of ClassC, I can call methods of ClassA and ClassC..
// Its just because ClassC extends ClassA
// Here ClassB cant call methods of ClassB because there is no
relation between ClassC and ClassB

obj.methodOneClassA();
obj.methodOneClassC();

    }
}

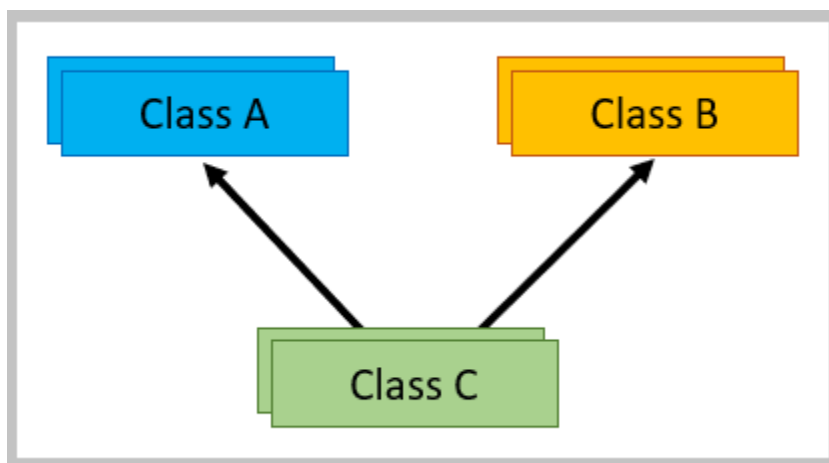
```

Output:

I am a Method One of ClassA  
I am a Method One of ClassC

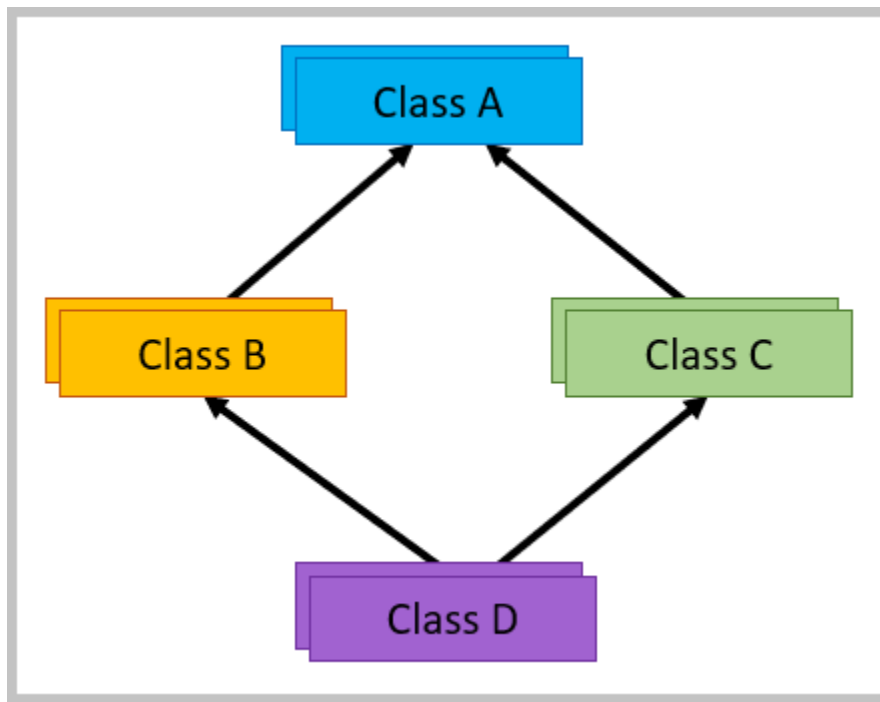
### Multiple Inheritance:

Multiple Inheritance is not possible in Java. In this a child extends two parents classes say Class C extends Class A and Class B.



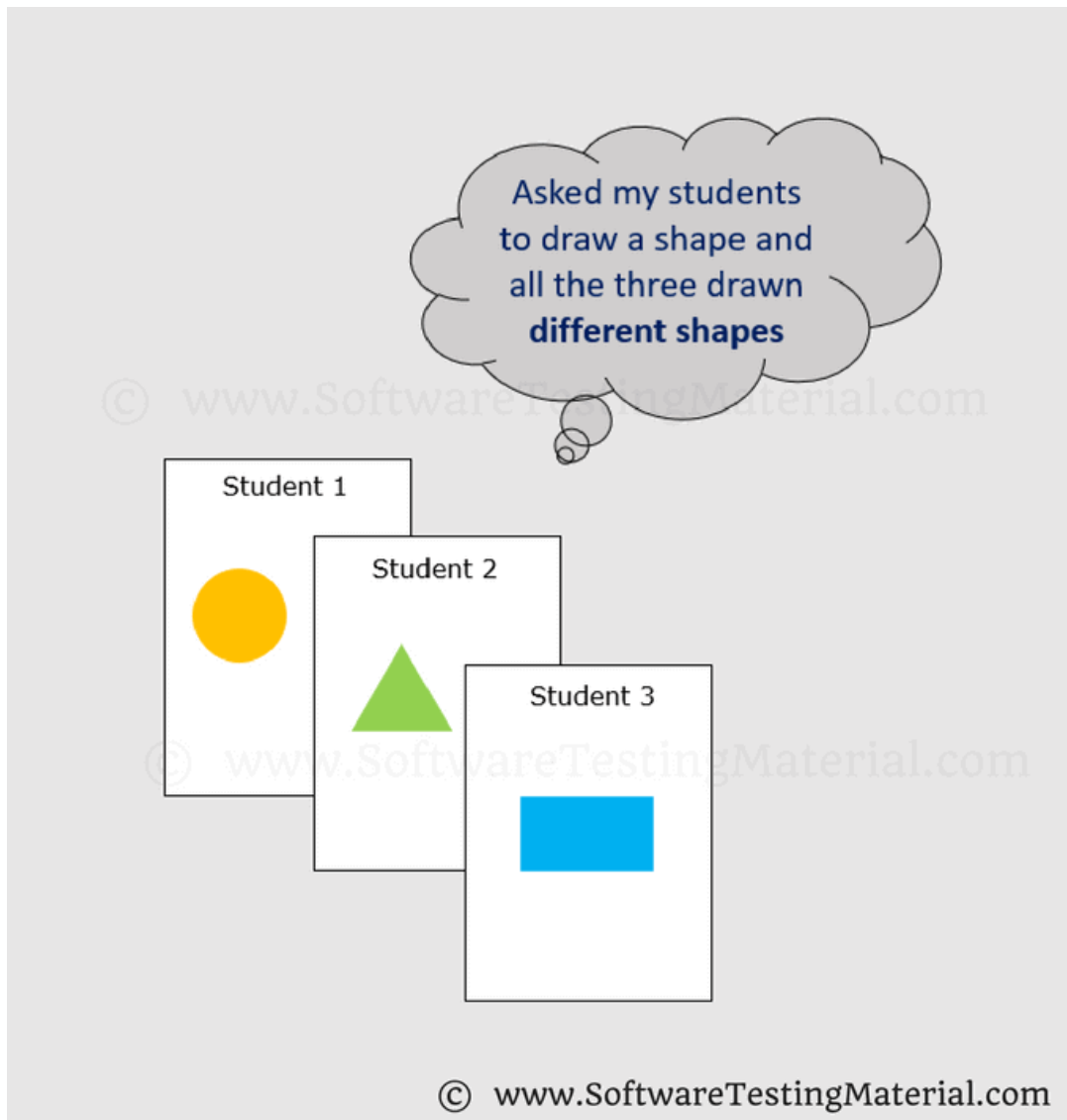
### Hybrid Inheritance:

Combination of more than one type of inheritance.



## 16. What is Polymorphism?

Polymorphism allows us to perform a task in multiple ways. Let's break the word Polymorphism and see it, 'Poly' means 'Many' and 'Morphos' means 'Shapes'.



Assume we have four students and we asked them to draw a shape. All the four may draw different shapes like Circle, Triangle, and Rectangle.

### **17. What are the types of Polymorphism?**

There are two types of Polymorphism in Java

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

We can perform polymorphism by 'Method Overloading' and 'Method Overriding'

### **18. What is Method Overloading?**

A class having multiple methods with the same name but different parameters are called Method Overloading

There are three ways to overload a method.

- Parameters with different data types
- Parameters with a different sequence of data types
- Different number of parameters

## 19. What is Method Overriding?

Declaring a method in child class that is already present in the parent class is called Method Overriding.

In simple words, overriding means to override the functionality of an existing method.

In this case, if we call the method with the child class object, then the child class method is called. To call the parent class method we have to use **super** keyword.

## 20. What is Abstraction in Java?

Abstraction is the methodology of hiding the implementation of internal details and showing the functionality to the users.



Example: Mobile Phone.

A layman who is using a mobile phone doesn't know how it works internally but he can make phone calls.

## **21. What is Abstract Class in Java?**

We can easily identify whether a class is an abstract class or not. A class that contains abstract keyword in its declaration then it is an Abstract Class.

Syntax:

```
abstract class <class-name>{}
```

Points to remember:

- Abstract classes may or may not include abstract methods
- If a class is declared abstract then it cannot be instantiated.
- If a class has abstract method then we have to declare the class as abstract class
- When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared abstract.

## **22. What is Abstract Method?**

An abstract method is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

```
abstract void myMethod();
```

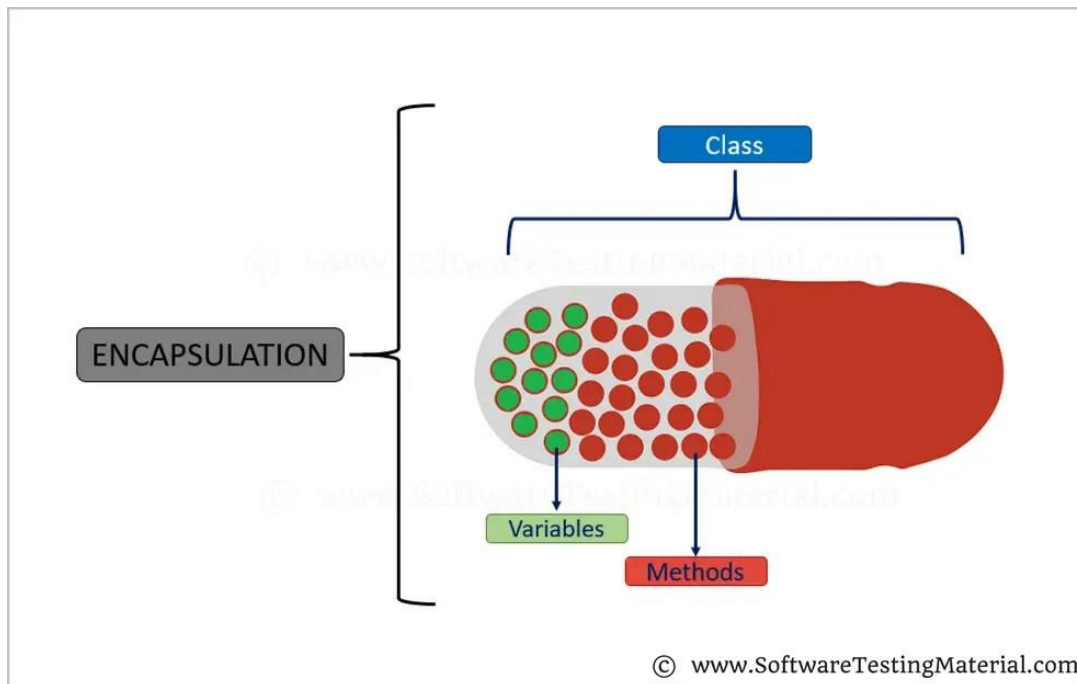
In order to use an abstract method, you need to override that method in sub class.

## **23. What is Interface in Java?**

An interface in Java looks similar to a class but both the interface and class are two different concepts. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract. We can achieve 100% abstraction and multiple inheritance in Java with Interface. Read more on

## **24. What is Encapsulation in Java?**

Encapsulation is a mechanism of binding code and data together in a single unit. Let's take an example of Capsule. Different powdered or liquid medicines are encapsulated inside a capsule. Likewise in encapsulation, all the methods and variables are wrapped together in a single class.



## 25. What is String in Java?

String in Java is an object that represents sequence of characters. An array of characters works same as Java string. String in Java is an immutable (cannot grow) object that means it is constant and cannot be changed once it is created.

### For example:

```
char[ ] c={'S','T','M'};
```

## 26. Why are strings immutable in Java?

In Java, String is immutable to make sure that the string value doesn't change. String literals are usually shared between multiple clients. If the value of the string changes (from "STM" to "stm"), it will affect all reference variables and cause severe discrepancies.

The **String is immutable** in [Java](#) because of the security, synchronization and concurrency, caching, and class loading. The reason of making string final is to destroy the immutability and to not allow others to extend it..

## 27. What is the difference between equals() method and double equal operator (==) in Java?

### equals() method

- This method is defined in the Object class in Java.
- It is used for checking the equality of contents between two objects defined by business logic.

- `public boolean equals(Object o)` is the method provided by the `Object` class.

### double equal operator (==)

- It is a binary operator in Java.
- It is used for comparing addresses (or references), i.e checks if both the objects are pointing to the same memory location.
- Default implementation uses double equal operator `==` to compare two objects.

### 43. Difference between Array and ArrayList?

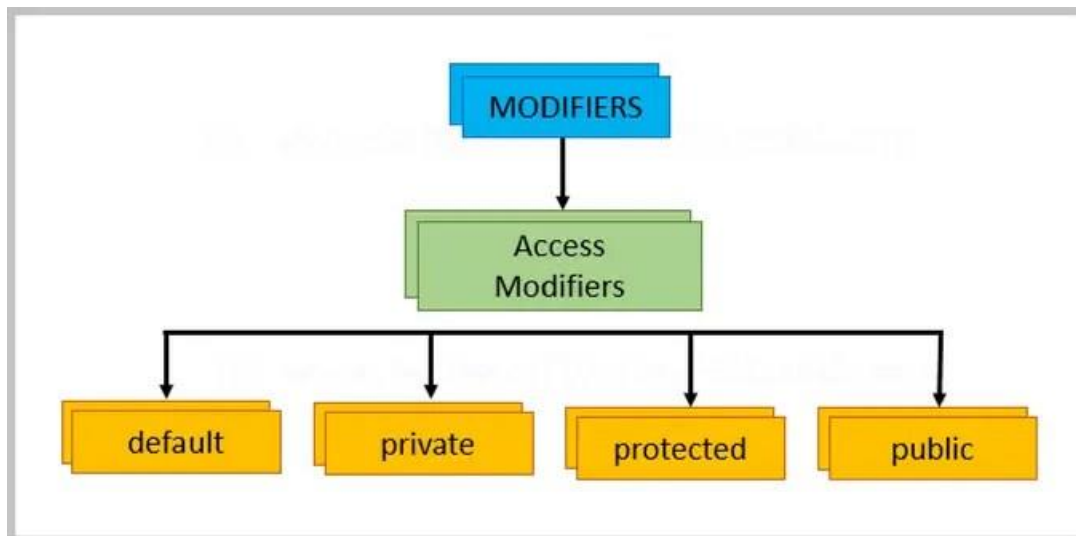
Array	ArrayList
Array is static	ArrayList is dynamic
Size of the array should be given at the time of array declaration. We cannot change the size of array after creating it	Size of the array may not be required. It changes the size dynamically. Capacity of ArrayList increases automatically whenever we add elements to an ArrayList
Array can contain both primitive data types as well as objects	ArrayList cannot contain primitive data types. It contains only objects
Arrays are multidimensional	ArrayList is always single dimension

### 44. Difference between ArrayList and HashSet in Java?

ArrayList	HashSet
ArrayList implements List interface	HashSet implements Set interface
ArrayList allows duplicates	HashSet doesn't allow duplicates
ArrayList is an ordered collection and maintains insertion order of elements	HashSet is an unordered collection and doesn't maintain insertion order
ArrayList is backed by an Array	HashSet is backed by an HashMap instance
ArrayList is an index based	HashSet is object based
In ArrayList, we can retrieve object by calling <code>get()</code> method or remove object by calling <code>remove()</code> method	In HashSet, we can't achieve <code>get()</code> method

### 45. What are the different access modifiers available in Java?

Access modifiers are subdivided into four types such as Default, Public, Private, Protected



**default:** The scope of default access modifier is limited to the package only. If we do not mention any access modifier, then it acts like a default access modifier.

**private:** The scope of private access modifier is only within the classes.

Note: Class or Interface cannot be declared as private

**protected:** The scope of protected access modifier is within a package and also outside the package through inheritance only.

Note: Class cannot be declared as protected

**public:** The scope of public access modifier is everywhere. It has no restrictions. Data members, methods and classes that declared public can be accessed from anywhere.

#### 46. Difference between static binding and dynamic binding?

1. Static binding is also known as early binding whereas dynamic binding is also known as late binding.
2. Determining the type of an object at compile time is Static binding whereas determining the type of an object at run time is dynamic binding
3. Java uses static binding for overloaded methods and dynamic binding for overridden methods.

.

#### 47. Difference between Abstract Class and Interface?



ABSTRACT CLASS	INTERFACE
To declare Abstract class we have to use abstract keyword	To declare Interface we have to use interface keyword
In an Abstract class keyword abstract is mandatory to declare a method as an abstract	In an Interface keyword abstract is optional to declare a method as an abstract. Compiler treats all the methods as abstract by default
An abstract class contains both abstract methods and concrete methods(method with body)	An interface can have only abstract methods
An abstract class provides partial abstraction	An interface provides fully abstraction
An abstract class can have public and protected abstract methods	An interface can have only public abstract methods
An abstract class can have static, final or static final variables with any access modifiers	An interface can have only public static final variables
An abstract class can extend one class or one abstract class	An interface can extend any number of interfaces
Abstract class doesn't support multiple inheritance	Interface supports multiple inheritance

#### 48. What is Multiple Inheritance?

If a class implements multiple interfaces, or an interface extends multiple interfaces then it is known as multiple inheritance.

#### 49. What are the differences between throw and throws in Java?

##### throw keyword

- The throw keyword is used to explicitly throw an exception in the program inside a function or inside a block of code.
- The checked exceptions cannot be propagated with throw only.
- The throw keyword is followed by an instance.
- The throw keyword is used within the method.
- You cannot throw multiple exceptions.

##### throws keyword

- The throws keyword is used in the method signature to declare an exception which might get thrown by the function while executing the code.
- The checked exception can be propagated with throws
- The throws keyword is followed by class.
- The throws keyword is used with the method signature.
- You can declare multiple exceptions, e.g., public void method () throws IOException, SQLException.

## 50. What is functional interface?

Ans) functional interface is nothing but a interface which contains only one abstract method, which can be represented using `@FunctionalInterface` annotation.

Functional interface also called as SAM (single Abstract method)

`@FunctionalInterface`

```
Interface MyFunctionalInterface {  
    Void myMethod ();  
}
```

Advantage of this is we can write the lambda expression using functional interface.

## 51. What is lambda Expression and its benefits?

Ans) Lambda expression was a new feature introduced in the Java 8 support, which can be used with functional interface (SAM (single Abstract method))

The Major advantage of lambda expression is it reduces the extra implementation of class, loading and execution time is saved and obviously it improves the performance of the soft ware

Benefits of the Lambda expression it reduces the boiler plate code and enables the functional programming.

## 52. What is the Difference between Final, Finally and Finalize?

Ans) Final is a keyword it is used to specify the restrictions on the programming components like class, method and variable, if we apply final keyword it can be modified or overridden. Once created cannot be modified

```
Public final class ImmutableClass {  
    //Final variable  
    Public final int Max_size =100;
```

```

//Final method
Public final void displaymessage() {
    System.out.println(" This is a final method")
}
}

```

Finally block it is related to exception handling , the code in the finally block execute no matter what , for example finally block can be used to close the database resources after the execution of the code

```

Try {
    //Some code that may throw error
} catch (Exception e) {
    // Handle exception
} finally {
    //code that will always execute
}

```

**Finalize** belongs to java.lang.obj class and it is a special type of the method in java which is called by the garbage collector before reclaiming the memory, the main purpose of the finalize method is to perform cleaning actions.

```

@Override
Protected void finalize () throws Throwable {
    // Cleanup Operation
    Super.finalize ();
}

```

### 53. What is the differences between Method and Constructor in java?

Method is a block of code, which defines the behavior of object or class, can be called multiple times, and accepts the parameters.

Method is used to perform the tasks or actions

```
Public class Example {  
    // Method definition with return type string  
    Public static String great (String name) {  
        return "Hello" + name + "!" ;  
    }  
}
```

Constructor is a special type of method in java which as a same name as a class name, which is executed automatically while object creation, constructor is used to initialize the instance variable.

Advantage of constructor is more efficient than method because it is called while object creation

```
Public class Example {  
    Private String message;  
    // Constructor  
    Public Example () {  
        this.message =" helo" ;  
    }  
}
```

### 54. What is this word in java?

The keyword 'this' in Java serves a fundamental purpose: it refers to the current object. In other words, 'this' represents the instance of the class where it's used. It's commonly

used to access or modify the fields of the current object, especially when field names are the same as local variable names.

### **55. What is Autoboxing and unboxing?**

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. If the conversion goes the other way, this is called unboxing.

Autoboxing is the process by which primitive data types are automatically converted into their corresponding wrapper objects, while unboxing involves the automatic extraction of the primitive value from the wrapper object. Understanding these mechanisms is essential for enhancing code readability and flexibility

### **56. What is serialization and deserialization?**

Serialization and deserialization are processes that convert data objects into a portable format. Serialization is the process of converting an object's state into a byte stream, which can then be saved to a file, sent over a network, or stored in a database. Deserialization is the reverse process of converting a byte stream back into an object.

### **57. What is an abstract modifier?**

The abstract modifier indicates that the thing being modified has a missing or incomplete implementation. The abstract modifier can be used with classes, methods, properties, indexers, and events

The abstract keyword is a non-access modifier, used for classes and methods. Class: An abstract class is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

### **58. Difference Between this() and super() Constructor**

The this() constructor refers to the current class object. The super() constructor refers immediate parent class object. It is used for invoking the current class method. It is used for invoking parent class methods.

super is used to access methods of the base class while this is used to access methods of the current class. Extending the notion, if you write super() , it refers to constructor of the base class, and if you write this() , it refers to the constructor of the very class where you are writing this code

### **59. Can we use this () and super () in a method?**

The invocation of "this()" or "super()" must be the first line of the constructor. Both are responsible for invoking constructors, so they cannot be used in the same constructor. "this()" can invoke other constructors in the same class, whereas "super()" invokes the constructor of the parent class.

### **60. What is the difference between collection and array?**

Arrays are fixed in size, whereas some Collections are grow-able in nature. Arrays store homogeneous data. Collections store both homogeneous as well as heterogeneous data. In Arrays, there are no underlining data structures, whereas Collections have underlining data structures.

### **61. what is the difference between static and non static in java**

Static: Static members are initialized when the class is loaded into memory, typically during program startup. Initialization happens only once.

Non-Static: Non-static members are initialized when each instance of the class is created, usually using the new keyword. Initialization occurs separately for each object.

Static class is defined using static keyword. Non-Static class is not defined by using static keyword. In static class, you are not allowed to create objects. In non-static class, you are allowed to create objects using new keyword.

### **62. What is call by reference and call by value?**

Call by value and call by reference are two ways to pass data to a function:

Call by value

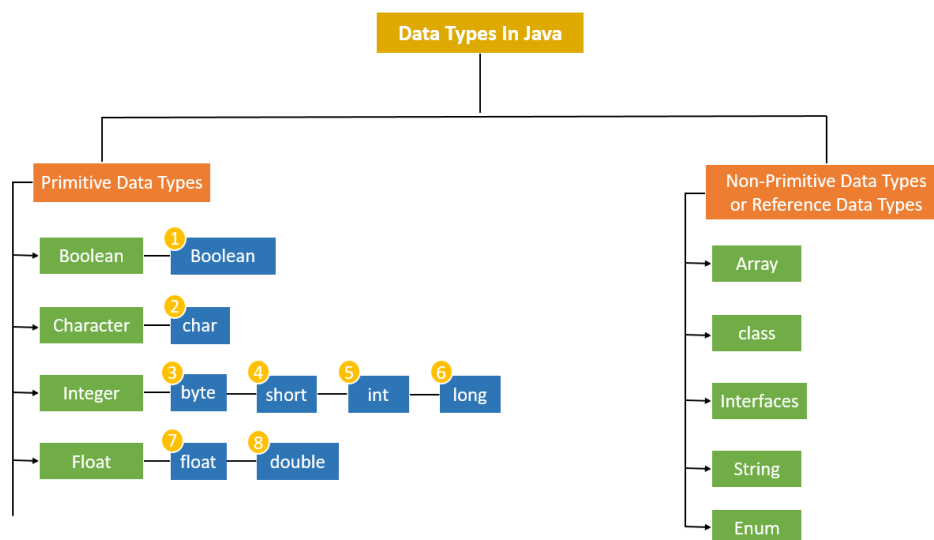
Passes a copy of the data to the function, so any changes made in the function don't affect the original data. The parameter acts as a new local variable within the function, initialized to the value of the argument.

Call by reference

Passes the memory address (reference) of the original data to the function, so the function can directly modify the original data. The argument variable supplied by the caller can be affected by actions within the called function.

Technically, Java is always pass by value, because even though a variable might hold a reference to an object, that object reference is a value that represents the object's location in memory.

### 63.Primitives and Non-Primitives datatypes in Java? String is primitive or non-primitive?



### 64.What is the method of overloading?

Method overloading in Java means having two or more methods (or functions) in a class with the same name and different arguments (or parameters). It can be with a different number of arguments or different data types of arguments

### 65. Why is it important to override hashCode() when you override equals()?

You must override hashCode() in every class that overrides equals(). Failure to do so will result in a violation of the general contract for Object. hashCode(), which will

prevent your class from functioning properly in conjunction with all hash-based collections, including HashMap, HashSet, and Hashtable.

## **66. What is the difference between a checked and unchecked exceptions?**

To summarize, the difference between a checked and unchecked exception is: A checked exception is caught at compile time whereas a runtime or unchecked exception is, as it states, at runtime

Checked Exceptions should be used for predictable, but unpreventable errors that are reasonable to recover from. Unchecked Exceptions should be used for everything else.

## **67. Difference between String Builder and String Buffer?**

StringBuffer is synchronized, meaning its methods are thread-safe and can be safely used in a multithreaded environment. On the other hand, StringBuilder is not synchronized, which makes it faster than StringBuffer, but it is not thread-safe and should not be used in a multithreaded environment.

## **68. What do you mean by POJO why we use POJO?**

POJO stands for Plain Old Java Object. It is an ordinary Java object, not bound by any special restriction other than those forced by the Java Language Specification and not requiring any classpath.

Their primary advantage is their reusability and simplicity.

## **69. How to define dynamic array?**

A dynamic array is a variable-size list data structure that can grow and shrink during runtime. It's also known as a resizable array, growable array, mutable array, or array list. Dynamic arrays are flexible and efficient with memory, and they overcome a limitation of static arrays, which have a fixed capacity

Here are some key features of dynamic arrays:

- Automatic resizing: Dynamic arrays can automatically increase in size when more elements are added. For example, in C++, a dynamic array might double in size when it runs out of space.
- Adding elements: Elements can be added to the end of the array.



- Deleting elements: Elements can be removed from the array using the default `remove()` method, or by calling the `removeAt(i)` method to remove an element at a specific index.

### **70. Can we create the object for the abstract classes?**

We cannot create objects of an abstract class. To implement features of an abstract class, we inherit subclasses from it and create objects of the subclass. A subclass must override all abstract methods of an abstract class.

### **71. Can we create the object for an interface?**

In Java, we cannot create objects of interfaces directly because interfaces are abstract by nature. They cannot be instantiated like regular classes. Attempting to do so will result in a compilation error.

### **72. Can we create constructor of abstract class?**

Abstract classes can have constructors, but they cannot be instantiated directly. The constructors are used when a concrete subclass is created. There may be one or greater abstract methods in an abstract class, which signifies that those methods are not implemented by means of the class.

### **73. Can constructor be overloaded. Explain why?**

Constructors can be overloaded in a similar way as function overloading. Overloaded constructors have the same name (name of the class) but the different number of arguments. Depending upon the number and type of arguments passed, the corresponding constructor is called.

### **74. Can main method be overloaded?**

Yes, We can overload the main method in java but JVM only calls the original main method, it will never call our overloaded main method

### **75. Can main method be overridden?**

No, we cannot override main method of java because a static method cannot be overridden. The static method in java is associated with class whereas the non-static method is associated with an object

#### **76. Can we override static method?**

No, you cannot override static methods in Java. This is because static methods are associated with the class itself, not with an instance of the class. When a subclass inherits a static method from its parent class, it cannot change the method's behavior.

#### **77. Can we overload static method?**

Yes, static methods can be overloaded in Java, but they cannot be overridden. Overloading a static method means having multiple methods with the same name but different parameter lists within the same class or subclass. For example, you can have two or more static methods with the same name, but with different input parameters. However, you cannot overload two methods if they differ only by the static keyword, even if the number and types of parameters are different

#### **78. Can we write non-abstract methods in Interface?**

In Java, prior to Java 8, interfaces only allowed abstract methods, meaning methods without a body. However, with the introduction of default and static methods in Java 8, you can now include non-abstract methods in interfaces. 1. Default Methods: These are methods in an interface that have a default implementation

#### **79. Can we execute a java program without main method?**

Yes, you can compile and execute without main method by using a static block. However, after static block executes, you will get an error saying no main method found. And latest info, you can't do this with Java 7 version. It will not execute.

#### **80. Can we call a non-static variable in static method?**

Non-static variables cannot be used inside static methods. It will throw a compile-time error. Static variables are memory efficient and are created only once per class.

### **81. Can I execute multiple catch blocks without try will it give me compile time error?**

If you use multiple catch blocks for the same type of exception, then it will give you a compile-time error because C# does not allow you to use multiple catch block for the same type of exception. A catch block is always preceded by the try block

### **82. How to achieve serialization and deserialization?**

In Java, serialization is the process of converting an object's state into a byte stream, while deserialization is the process of converting the byte stream back into an object. Serialization and deserialization can be used to: Persist an object, Send an object across a network, Save an object as a file, Store an object in a database, and Transport code from one JVM to another

To serialize an object in Java, you can use the `writeObject()` method from the `ObjectOutputStream` class. To deserialize an object, you can use the `readObject()` method from the `ObjectInputStream` class.

To make a class serializable in Java, you need to implement the `java.io.Serializable` interface. This interface is a marker interface with no fields or methods to implement. You can only serialize an object if you implement the serializable interface, and all references must also be serializable if you want to serialize an object with a reference to another class.

### **83. If we declare the main method as private what will happen?**

The code gets compiled but it does not get executed as the JVM cannot find a private method. The method has to be declared public for the JVM to find it and execute it.

#### **84. How to check whether the array is empty and null?**

The isEmpty() function takes an array as a parameter. Then it will check if the parameter passed to it is null or empty. If the array passed as a parameter is null or empty then it would return a true. If the array passed as a parameter is not null or empty then it would return a false

#### **85. What are the classes available in a list interface?**

Here are the different implementation classes of the List interface in Java:

- AbstractList.
- AbstractSequentialList.
- ArrayList.
- AttributeList.
- CopyOnWriteArrayList.
- LinkedList.
- RoleList.
- RoleUnresolvedList.
- Stack.
- Vector.

The most commonly used implementation of the List interface are ArrayList and LinkedList.

Since both classes above implement the List interface, they make use of the same methods to add, access, update, and remove elements in a collection.

In this tutorial, we'll have a look at how we can add, access, update, and remove elements in a collection using the ArrayList.

#### **86. What is the use of constructor in java?**

A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes.

Constructor in java is used to create the instance of the class. Constructors are almost similar to methods except for two things - its name is the same as the class name

and it has no return type. Sometimes constructors are also referred to as special methods to initialize an object

### **87. What is Hashmap? Can we store objects in Hashmap and how to retrieve them?**

HashMap is a data structure that implements the Map interface and uses a hash table for storing key-value pairs. It allows the insertion of elements based on key-value pairs and provides constant-time performance for basic operations like adding or retrieving elements.

Value in hashmap can be any wrapper class, custom objects, arrays, any reference type or even null .

### **88. Difference between Hash Map and Hash Set?**

The main difference between them is that HashSet stores unique elements without any associated values, while HashMap stores key-value pairs where the keys are unique identifiers and the values are associated data.



# HashMap vs HashSet

## Comparison Chart

HashMap	HashSet
It is a general-purpose implementation of the Map interface which stores key/value pairs.	It is a standard implementation of the Set interface backed by a hash table.
It works on hashing principle to map identifying values internally.	It internally uses HashMap as a backing data structure to add or store objects.
It allows only one null key but allows any number of null values.	It allows only one null value.
It does not allow duplicate keys but it is allowed to have duplicate values.	It cannot have duplicate elements by the definition of a Set.

### 95. Is it possible to override Static method?

No, it's not possible to override a static method in Java:

Static methods are associated with the class, not an instance

.When a subclass inherits a static method from its parent class, it can't modify the static method's behavior.

Static methods are bound at compile time, while overriding relies on dynamic binding at runtime

. If you try to override a static method in a child class, the child's class method will be hidden and the parent's class method will be called based on the object reference.

Declaring a static method with the same signature in a subclass isn't considered overriding

. The method in the derived class will be hidden by the method in the base class.

You can, however, overload a static method in Java. Overloading means having multiple methods with the same name but different parameter lists within the same class or subclass

## **89. What is meant by Thread?**

A thread in Java is the direction or path that is taken while a program is being executed. Generally, all the programs have at least one thread, known as the main thread, that is provided by the JVM or Java Virtual Machine at the starting of the program's execution.

## **90. What is singleton class in java?**

The Singleton's purpose is to control object creation, limiting the number to one but allowing the flexibility to create more objects if the situation changes. Since there is only one Singleton instance, any instance fields of a Singleton will occur only once per class, just like static fields.

A singleton class in Java ensures only one instance of itself exists. To make one, create a class with a private static instance variable and a private constructor. Then, provide a public static method to access the instance. This method checks if the instance exists; if not, it creates one.

## **91. Is Hashmap thread safe?**

The reason is that the HashMap object referenced by the static variable cache is shared by all calls to isPrime() , and HashMap is not threadsafe. If multiple threads mutate the map at the same time, by calling cache

This HashMap class extends AbstractMap class that implements the Map interface. HashMap stores the entries in the form of a key-value pair and allows at max one null key and multiple null values. Java HashMap is not thread-safe and hence it should not be used in multithreaded applications

## **92. What is static, How to set value of static variable**

When you declare a variable or a method as static, it belongs to the class, rather than a specific instance. This means that only one instance of a static member exists, even if you create multiple objects of the class, or if you don't create any.

Whenever a variable is declared as static, this means there is only one copy of it for the entire class, rather than each instance having its own copy. A static method means it can be called without creating an instance of the class

### **93. Can we overload private methods?**

No, we cannot override private methods because the scope of private methods is limited to the class and we cannot access them outside of the class which they are defined in. Just like static and final methods, private methods in Java use static binding that is done at compile time

### **94. Is it possible to extend Final Class?**

Final classes cannot be extended or inherited. If we try to inherit a final class, the compiler throws an error during compilation. We can simply define a final class using the final keyword and write the class body code according to our needs.

### **96. Is it possible to overload main method?**

Yes, you can overload 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. That means main method acts as an entry point for the Java interpreter to start the execution of the application.

### **97. Is it possible to initialize a variable present in an Interface?**

In Java, interface variables are **static** and **final** by default. This means they must be initialized on the first line or in the constructor. Since interfaces don't have constructors, you need to initialize final variables directly. However,



instance variables (fields) are not part of an interface's contract. If you want to declare variables in an interface, consider using an **abstract class** instead.

Every variable of an interface must be initialized in the interface itself. The class that implements an interface can not modify the interface variable, but it may use as it defined in the interface.

### **98. What would happen, if multiple inheritance is possible, in Java?**

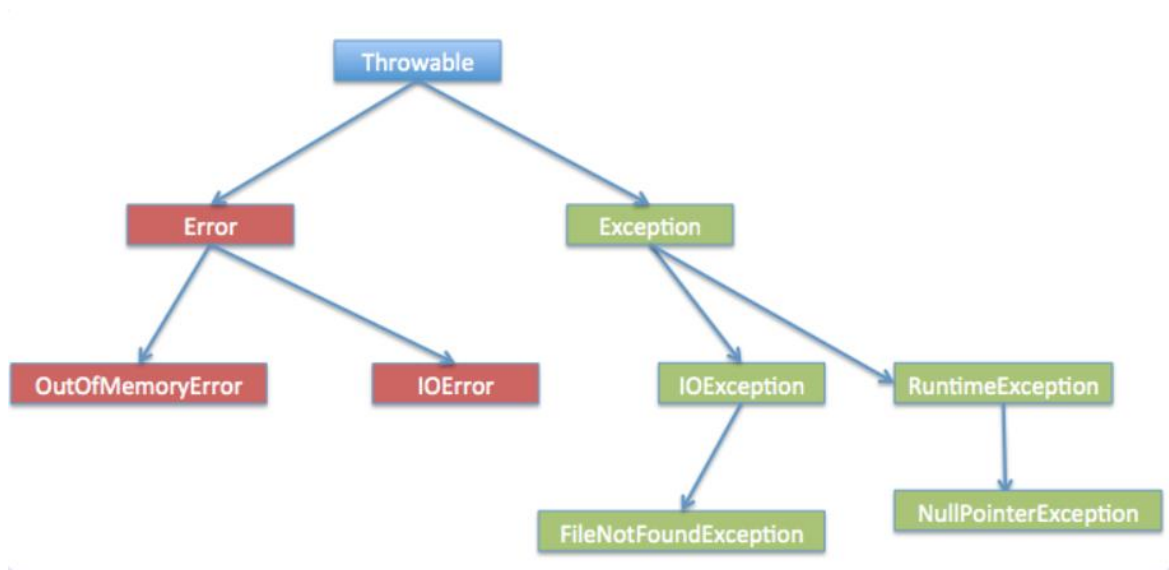
When a class inherits from more than one parent class, it can lead to complex hierarchies that are challenging to manage. The main issue is tracking the origin of methods and attributes, especially when the parent classes have methods or attributes with the same names but different implementations.

Java doesn't support multiple inheritances in classes because it can lead to diamond problem and rather than providing some complex way to solve it, there are better ways through which we can achieve the same result as multiple inheritances

### **99. Explain Exceptions hierarchy in java?**

An exception in Java is an event that disrupts the normal functioning of a program during execution. It can occur at compile time or run time and it can be of many types. These exceptions are unwanted and have a fundamental hierarchy

Java Exceptions are hierarchical and inheritance is used to categorize different types of exceptions. Throwable is the parent class of Java Exceptions Hierarchy and it has two child objects – Error and Exception. Exception s are further divided into Checked Exception s and Runtime Exception s



### 100. Explain Set and Map in Java?

Both interfaces are used to store the collection of objects as a single unit. The main difference between Set and Map is that Set contains only data elements, and the Map contains the data in the key-value pair, so Map contains key and its value.

### 101. How and when to use interface?

Interfaces are useful in many situations, including:

Defining a standard behavior

When multiple classes work similarly, interfaces can create a standard behavior.

Capturing similarities

Interfaces can capture similarities between unrelated classes without forcing a class relationship.

Declaring methods

Interfaces can declare methods that one or more classes are expected to implement.

Achieving security

Interfaces can hide certain details and only show important details of an object.

Modeling multiple inheritance

Interfaces can model multiple inheritance, which allows a class to have more than one superclass.

Defining interactions

Interfaces can define interactions between a hardware device, software program, and a user

### **102. Can we instantiate an interface?**

An interface can't be instantiated directly. Its members are implemented by any class or struct that implements the interface. A class or struct can implement multiple interfaces. A class can inherit a base class and also implement one or more interfaces.

### **103. Can we override constructor?**

Constructor looks like method but it is not. It does not have a return type and its name is same as the class name. But, a constructor cannot be overridden

### **104. What is the `System.out.println()` and use of it?**

In Java, `System.out.println()` is a statement which prints the argument passed to it. The `println()` method display results on the monitor.

### **105. Can we use multiple catches? When can we use multiple catches?**

Yes you can have multiple catch blocks with try statement. You start with catching specific exceptions and then in the last block you may catch base Exception . Only one of the catch block will handle your exception.

### **106. Different between POI and JXL?**

JXL doesn't support drawing shapes; Apache POI does. JXL supports most Page Setup settings such as Landscape/Portrait, Margins, Paper size, and Zoom. Apache POI supports all of that plus Repeating Rows and Columns. JXL doesn't support Split Panes; Apache POI does.

### **107. How to prevent the override method in Java?**

#### **Different Ways to Prevent Method Overriding in Java**

- Using a static method.
- Using private access modifier.

- Using default access modifier.
- Using the final keyword method.

### **108. Why is the main method static?**

The main() method is static in Java and C# to allow the JVM or compiler to invoke it without instantiating the class that contains it. This is necessary because there is no class object present when the Java runtime starts, and the JVM initiates program execution without creating objects.

Marking the main() method as static also saves memory that would otherwise be used by an object created just to call the main() method. If the main() method is not static, the JVM will be unable to call it and the program will result in an error.

### **109. What is the use of static variables?**

Static variables, by contrast, are variables associated with a class itself, rather than a particular instance of that class. Static variables are used to keep track of information that relates logically to an entire class, as opposed to information that varies from instance to instance.

### **110. How will you access default and protected class?**

A class with default access has no modifier preceding it in the declaration. Default access is simple in that **only the code within the same package can access code with default access**. In other words, a class with default access can be seen only by classes within the same package.

The protected members are inherited by the child classes and can access them as its own members. But we can't access these members using the reference of the parent class. We can access protected members only **by using child class reference**.

### **111. Why Object creation not possible in Abstract classes?**

Because an abstract class is an incomplete class (incomplete in the sense it contains abstract methods without body and output.) We cannot create an instance or object, the same way you say for an interface.

You can't create an object of an abstract class type. However, you can use pointers and references to abstract class types. You create an abstract class by declaring at least one pure virtual member function. That's a virtual function declared by using the pure specifier ( = 0 ) syntax

## 112. What All of the classes in the Java Collection Framework have?

Classes in Java's Collection Framework have different functionalities:

- HashSet  
Implements the Set interface and extends AbstractSet, allowing only unique elements to be stored
- ArrayList  
Part of the java.util package, this class provides dynamic arrays that can be useful in programs that require a lot of array manipulation
- LinkedList  
Part of the java.util package, this class implements the LinkedList data structure, which is a linear data structure where elements are not stored in contiguous locations
- TreeSet  
Implements the NavigableSet interface, which provides functionality to navigate through the SortedSet, which in turn provides functionality to keep elements sorted
- HashMap  
Part of the Java Collections framework, this class provides the functionality of storing key/value pairs, where keys are used as unique identifiers to associate each value on the map
- PriorityQueue  
Based on the priority heap, the elements of the priority queue are ordered according to the natural ordering, or by a Comparator provided at queue construction time
- TreeMap

Implements the Map interface and NavigableMap, and stores key-value pairs in the natural sorting order of the keys or a sorting order defined with the help of a Comparator

- **ArrayDeque**

Also known as Array Double Ended Queue (Array Deck), this class provides a way to apply a resizable-array in addition to the implementation of the Deque interface

- **Vector**

Part of the Java Collections Framework, this class provides a dynamic array implementation of the List interface, and was added in the original release of Java (Java 1.0)

### **113. Will Java provide default constructor by own ? How**

A default constructor is a constructor without parameters. If a class has no constructors, the Java compiler automatically provides a default constructor. This is called an implicit default constructor. You can also define a default constructor explicitly.

Every time an object is created using the `new()` keyword, at least one constructor is called. It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

### **114. Difference between Arraylist and Linked List, In which situation they are used ? Difference between List list = new ArrayList() and ArrayList list = new ArrayList();**

Certainly! Let's explore the differences between ArrayList and LinkedList in Java:

#### **ArrayList:**

**Data Structure:** ArrayList uses a dynamic array to store elements. It's part of the collection framework and resides in the `java.util` package.

#### **Advantages:**

Supports all types of objects due to generics.

Dynamic resizing (automatic expansion) when needed.

**Disadvantages:**

Slower manipulation (e.g., removal) due to internal array traversal and memory shifting.

Inefficient memory utilization.

**Use Cases:**

When frequent random access (get operations) is required.

For one, two, or multi-dimensional lists.

**LinkedList:**

**Data Structure:** LinkedList uses a doubly linked list to store elements. Each element is a separate object with data and address parts.

**Advantages:**

Efficient insertions and deletions (no memory shifting).

Good memory utilization.

**Use Cases:**

When frequent insertions or deletions are needed.

For single, double, or circular linked lists.

Regarding your second question, both `List<String> list = new ArrayList<String>()` and `ArrayList<String> list = new ArrayList<String>()` create an ArrayList of strings. The difference lies in the variable type:

The first one (`List<String> list`) uses the interface type `List`, which allows you to switch to other list implementations (like `LinkedList`) without changing your code.

The second one (`ArrayList<String> list`) explicitly specifies the `ArrayList` type

## **115. Difference between HashMap and MultiMap?**

So in a `MultivaluedMap` you can insert 0, 1, 2, 3 or more objects related to the same key. In a `Map` you can insert exactly 1 object related to a key. This is the

difference, it can be useful if you need to store many values related to a single key, if you have only one value they are similar

### **116. In which situation the method should be static and when non static?**

In Java, methods can be static or non-static. Static methods belong to a class and can be called without an instance, while non-static methods belong to an object and are loaded when an instance of the class is created.

Here are some situations where you might want to use a static method:

When the method doesn't require object state manipulation

Static methods can't reference instance member variables, so they're a good choice for methods that don't need to manage state.

When the method operates on a shared state

For example, a static counter that's shared by all instances of a class should be operated on by a static method.

When the method should be accessible from outside the class

For example, if you want to access a field from outside a class, you can make it static and access it directly on the class itself.

Here are some situations where you might want to use a non-static method:

When the method only makes sense for an individual object: For example, if a method has to be different for each object of a class, it should be non-static.

### **117. How does HashMap is implemented using key value pair?**

In Java, HashMap is implemented using an array of buckets, where each bucket is essentially a linked list of key-value pairs. The process works as follows: Hashing: When you insert a key-value pair into the HashMap, Java computes the hash code of the key using the `hashCode()` method



**118. Suppose you have class and abstract class in class there is a user defined constructor and main method which one will get executed first?**

In Java, when you have both a class and an abstract class, and both contain a user-defined constructor and a main method, here's the order of execution:

**Constructor Execution:**

The constructor of the class (non-abstract) will be executed first.

If the class extends an abstract class, the abstract class's constructor is also called (before the class constructor).

Constructors are called in a top-down manner (from the base class to the derived class).

main **Method Execution:**

The main method is the entry point for executing a Java program.

It is a static method, so it doesn't require an instance of the class.

The main method is executed after the constructors have completed their execution.

In summary:

Abstract class constructor (if applicable).

Class constructor.

main method.

Remember that the main method is where your program starts, and it's called by the Java Virtual Machine (JVM) when you run your program. The constructors, on the other hand, are called when you create an object of the class

## 119. Difference between Comparable and Comparator?

`Comparable` is implemented by a class to define its *natural* ordering (e.g., sorting strings alphabetically). It has a single method, `compareTo()`. `Comparator` is a separate interface that defines a comparison between two objects. It has a `compare()` method and is used when you need custom sorting logic or when you can't modify the class being sorted.

## 120. What are cases where you will use a `finally` block?

The `finally` block in a `try-catch` statement is used for code that *must* execute regardless of whether an exception is thrown or caught. This is typically used for resource cleanup (e.g., closing files, database connections).

## 121. What is method hiding in Java?

Method hiding occurs when a subclass defines a static method with the same signature as a static method in its superclass.<sup>1</sup> It's not overriding (which applies to instance methods), but rather hiding the superclass's method. The method called depends on the reference type, not the object type.

## 122. Default vs. public access modifier:

`public` means the member is accessible from anywhere. `default` (no explicit modifier) means the member is accessible only within the same package.

## 123. What is an interface in Java, and how is it used?

An **interface** in Java is a blueprint for a class. It contains abstract methods (no implementation) and constants. A class implements an interface and provides concrete implementations for its methods.

Example:

```
interface Vehicle {
    void start();
    void stop();
}

class Car implements Vehicle {
    public void start() {
        System.out.println("Car starts");
    }
    public void stop() {
        System.out.println("Car stops");
    }
}
```

## 124. Explain method overloading and method overriding with examples.

### Method Overloading:

- Occurs **within the same class**.
- Methods have the **same name** but **different parameters** (number, type, or order).

Example:

```
class Calculator {
    int add(int a, int b) {
        return a + b;
    }
    double add(double a, double b) {
        return a + b;
    }
}
```

### Method Overriding:

- Occurs **between parent and child classes**.
- The child class provides its own implementation of a method defined in the parent class.

Example:

```
class Animal {
    void sound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Dog barks");
    }
}
```

## 125. Difference between Array, ArrayList, and Map

Feature	Array	ArrayList	Map
<b>Storage Type</b>	Fixed-size.	Dynamic resizing.	Key-Value pairs.
<b>Data Type</b>	Stores similar data types.	Stores similar data types.	Stores unique keys with values.
<b>Indexing</b>	Indexed by numbers.	Indexed by numbers.	Keys act as indices.
<b>Example</b>	<pre>int[] arr = new int[5];</pre>	<pre>ArrayList&lt;String&gt; list;</pre>	<pre>Map&lt;String, String&gt; map;</pre>

## 126. Final vs Static Keywords

Feature	final	static
Purpose	Prevents changes.	Belongs to the class, not instances.
Usage	Variables, methods, classes. Variables, methods, blocks.	
Example	<code>final int MAX = 100; static int count = 0;</code>	

## 127. Difference between Interface and Abstract Class

Feature	Interface	Abstract Class
Purpose	Defines a contract (100% abstraction).	Can have both abstract and concrete methods.
Multiple Inheritance	Supports.	Does not support.
Keyword	<code>interface.</code>	<code>abstract.</code>

## 128. Which open-source tools allow us to read and write MS Excel files using Java?

Answer:

- **Apache POI:** Used to read and write Microsoft Excel files (.xls and .xlsx).
- **JExcelAPI:** An older library for Excel operations.

## 129. Difference between Comparable and Comparator?

Answer:

Feature	Comparable	Comparator
Purpose	Defines natural ordering.	Defines custom ordering.
Interface	<code>java.lang.Comparable.</code>	<code>java.util.Comparator.</code>
Method	<code>compareTo().</code>	<code>compare().</code>
Modification	Changes the class's implementation.	Can be used externally to customize sorting.

### 130. What are cases where you will use the finally block?

**Answer:**

- Closing resources like files or database connections.
- Cleaning up memory (e.g., nullifying large objects).
- Logging final messages irrespective of the success or failure of the code.

### 131. Parent class of all exceptions in Java?

**Answer:**

`java.lang.Throwable.`

- Subclasses: `Error` and `Exception`.

### 132. Can you use multiple catch blocks with a try block?

**Answer:**

Yes, multiple catch blocks can be used to handle different exceptions.

**Example:**

```
try {
    int a = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Arithmetic Exception");
} catch (Exception e) {
    System.out.println("General Exception");
}
```

### 133. Can you create an object of an interface/abstract class? Explain.

**Answer:**

- No, you **cannot directly create an object** of an interface or abstract class because they are incomplete by design (methods lack implementation).
- **Example using interface:**
- ```
interface Animal {
```
- ```
    void sound();
```
- ```
}
```
- ```
Animal obj = new Animal() { // Anonymous class
```
- ```
    public void sound() { System.out.println("Roar"); }
```
- ```
};
```
- ```
obj.sound();
```

### 134. Why is String immutable in Java?

**Answer:**

- **Reasons for immutability:**
  - **Security:** Prevents modification of sensitive data like database URLs.
  - **Caching:** Allows reuse in the String Pool for memory efficiency.
  - **Thread-safety:** Makes Strings safe to use in multithreaded environments.
- **Behavior:** Any modification creates a new object, leaving the original unchanged.

### 135. Purpose of LinkedHashMap in Java? Have you used it in a framework?

**Answer:**

- **Purpose:** Maintains the insertion order of key-value pairs.
- **Use Case:** Logging order-sensitive data or maintaining API response structure.
- **Example in a framework:** Storing test data in a predictable order for API validation.

### 136. What is the Singleton design pattern in Java? Advantages of it?

**Answer:**

- **Singleton Pattern:** Ensures a class has only one instance globally.
- **Implementation:**

```
public class Singleton {  
    private static Singleton instance;  
    private Singleton() {}  
    public static Singleton getInstance() {  
        if (instance == null) {  
            instance = new Singleton();  
        }  
        return instance;  
    }  
}
```

**Advantages:** Memory efficiency, controlled access to a single instance

### 137. Why does an abstract class have no main methods?

An abstract class can have a `main` method, but it is not mandatory or typical because an abstract class is intended to be inherited by subclasses. The `main` method typically belongs to concrete classes that are meant to be executed, while abstract classes serve as blueprints for those subclasses.