Java Interview Questions

1. What do you understand by Java?

Java is an object-oriented programming language especially designed to focus on objects. It is a high-level programming language created by Sun Microsystems' James Gosling in 1995. Java is a quick, safe, and dependable programming language that is utilized in a variety of games, gadgets, and apps.

2. What is the difference between JDK and JRE?

Java Development Kit (JDK) is an acronym for Java Development Kit. It includes the tools and libraries needed to create Java programs. JRE stands for Java Runtime Environment, and it includes compilers and debuggers for compiling Java programs. This is a feature of JDK. JRE is a set of libraries and a virtual machine (JVM) that are necessary to run a Java program.

3. What are the features of the Java Programming language?

- Object-oriented: Java follows an object-oriented technique and employs a collection of objects from a class to perform any operation.
- Secure: It is extremely secure because it does not produce any security problems such as stack overflow. This is due to the fact that it does not employ the concept of pointers.
- Robust: Java is strong and dependable because it detects mistakes early on and avoids concerns like garbage collection, memory allocation, and exception handling.
- Simple: Java is simple to learn and does not necessitate the use of advanced programming concepts such as pointers or multiple inheritance.
- Platform-independent: Because the code we compile and execute on any operating system is the same, Java is platform-independent. This means that the code can be compiled on one system and run on another.
- Multithreaded: Supports the multithreading idea, which means we can run two or more programmes at the same time, maximising CPU utilisation.

- Great performance: Java makes advantage of a Just-in-Time compiler, which ensures high performance.
- Dynamic: Java is more dynamic when we compare to other programming languages like C or C++. This is because it can carry a large amount of run time information which we can use to verify runtime object access.

4. What are the different types of memory areas allocated by JVM?

In java, JVM allocates memory to different processes, methods, and objects. JVM is also platform-dependent. JVM provides the environment to execute the java file. Some of the memory areas allocated by JVM are:

- ClassLoader: It is a component of JVM used to load class files.
- Class (Method) Area: It stores per-class structures such as the runtime constant pool, field and method data, and the code for methods.
- Heap: Heap has created a runtime and it contains the runtime data area in which objects are allocated.
- Stack: Stack stores local variables, reference variable,s and partial results at runtime. It also helps in method invocation and return value. Each thread creates a private JVM stack at the time of thread creation.
- Program Counter Register: This memory area contains the address of the Java virtual machine instruction that is currently being executed.
- Native Method Stack: This area is reserved for all the native methods used in the application.

5. How Java platform is different from other platforms?

The following are the primary distinctions between the Java platform and other platforms:

The Java platform is a software-only platform that works on top of other hardware-based platforms; other platforms are primarily hardware software or hardware-only, and can only be run on hardware. On any operating system, a programmer can write Java code. Java byte code can run on any platform that supports it.

Other languages, on the other hand, require libraries that have been developed for a certain platform in order to run.

6. Can we write the main method as public void static instead of public static void?

No, you can't write it in this manner. Any method must specify the modifiers first, followed by the return value. Modifiers can be rearranged in any sequence.

Instead of public static void main(), we can use static public void main().

7. What is core java?

Sun's designation for Java SE, the standard edition, and a group of related technologies such as the Java Virtual Machine, CORBA, and so on is "Core Java". This is largely to distinguish itself from Java ME or Java EE, for example. It's also worth noting that they're referring to a library collection rather than a programming language.

8. What is an array in java?

A container object that stores a fixed number of values of a single type is called an array. The length of an array is determined at the time of its creation. Its length is set when it is created.

9. What is the difference between byte and char data types in Java?

Both the byte and char are numeric data types in Java that may represent integral integers in a range, but they are fundamentally different. The main distinction between the byte and char data types is that the former is used to store raw binary data, whilst the latter is used to store characters or text data. Character literals can be stored in a char variable,

For example, char ch ='x'; A character literal is enclosed in single quotes. A byte variable may store any value between -128 and 127, but a char variable can store any value between 0 and 255.

10. Why do we need a constructor in Java?

Java is an object-oriented programming language that allows us to build and manipulate things. A constructor is comparable to a method in that it is a piece of code. It's used to make an object and set the entity's initial state. A constructor is a one-of-a-kind function with the same name as the class. There is no other method to create an object without a constructor. Every object in Java has a default function constructor by default.

11. How many types of constructors does Java support?

The following constructors are supported by Java:

- Default or non-parametrized Constructors
- Parameterized Constructors
- Copy constructor

12. Why constructors cannot be final, static, or abstract in Java?

We know that the static keyword refers to a class rather than a class's object. The static constructor is not used because a constructor is called when a class object is created. Another point to consider is that if we specify a static function constructor, we won't be able to access or call it from a subclass. Because we know that while static is permitted within a class, it is not permitted within a subclass.

When we mark a method as final, we're saying we don't want any other classes to override it. The constructor, however, cannot be altered (according to the Java Language Specification). As a result, designating it as final is pointless.

When we mark a method as abstract, we're indicating that it doesn't have a body and should be implemented in a child class. When the new keyword is used, however, the constructor is called implicitly. As a result, it requires a body.

13. How to implement constructor chaining using the Java super keyword?

Using the super keyword, we can create constructor chaining by calling the parent class constructor from a subclass with distinct arguments. It's crucial to remember that the super statement should always come first in the child constructor. The compiler does not create another implicit call to the parent class when we use an explicit constructor call.

```
class Shapes {Shapes() {System.out.println("This is a shape");}Shapes(String name) {System.out.println("Shape name is: " + name);}}public class Main extends Shapes {Main() {super("Triangle");System.out.println("Triangle constructor");}public static void main(String[] args) {Main obj = new Main();}}
```

Shape name is Triangle

Triangle constructor

14. How aggregation and composition are different concepts?

Aggregation refers to a connection in which the child object can exist without the parent object. For example, if you eliminate the Bank, the Employee will still exist.

Composition, on the other hand, suggests a relationship in which the child cannot exist without the parent. The human and heart, for example, are not different entities.

The aggregation relation is a "has-a" relationship, while the composition relation is a "part-of" relationship.

Aggregation is a weak association, whereas composition is a strong association.

Aggregation

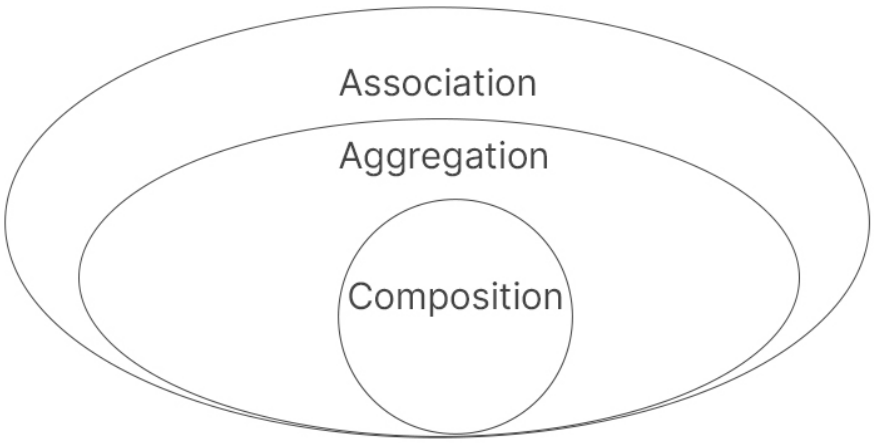
```
class Address{int street_no;String city;String state;int pin;Address(int street_no, String city, String state, int pin ){this.street_no = street_no;this.city = city;this.state = state;this.pin = pin;}}

public class Main{String name;Address ad;Main(String name){this.name = name;}public static void main (String[] args) {Main obj = new Main("Saurav");Address ad = new Address(12, "Mumbai", "Maharashtra", 400008);}}
```

Composition

```
class Job {private long salary;private int id;public long getSalary() {return salary;}public void setSalary(long salary) {this.salary = salary;}public int getId() {return id;}public void setId(int id) {this.id = id;}}

class Person {

//composition has-a relationshipprivate Job job;public Person(){this.job=new Job();job.setSalary(100000);}public long getSalary() {return job.getSalary();}

}

public class Main {

public static void main(String[] args) {Person person = new Person();long salary = person.getSalary();}

}
```

15. What is an association?

The term "association" refers to a connection in which each object has its own lifecycle and there is no owner. Consider the relationship between a teacher and a student. Numerous students can be associated with a single teacher, and a single student can be associated with multiple teachers, but the objects do not share ownership, and each has its own lifecycle. One-to-one, one-to-many, many-to-one and many-to-many relationships are all possible. Composition and Aggregation are the two forms of association.



16. If there are no pointers in Java, then why do we get NullPointerException?

The object reference is Java's pointer counterpart. When we use a., we're referring to an object. As a result, the JVM makes use of pointers, whereas programmers only see object references.

When we try to access a function or member variable on an object reference that links to a null object, we receive a NullPointerException.

17. What is object cloning in Java?

The process of creating an exact clone of an object in Java is known as object cloning. It basically means being able to generate a new item that is in the same state as the original. To do this, Java has the clone() method, which can be used to make use of this functionality. This method creates a new instance of the current object's class and then populates all of its fields with the same values as the relevant fields. The java.lang marker interface is used to clone an object. To avoid any runtime exceptions, Cloneable must be implemented. One thing to keep in mind is that Object clone() is a protected function that must be overridden.

18. Is it possible to use this() and super() both in the same constructor?

In the constructor, we can't utilize both keywords. this() and super() keyword must be the initial statements in the constructor, according to the Java rule. As a result, we can't utilize both in a constructor.

19. What is the difference between this() and super() in Java?

Below are the key differences between this and super.

This	Super
1. The current class object is represented by this().	1. The current instance of a parent/base class is represented by super().
2. used to call the same class's default constructor	2. To invoke the parent/base class's default constructor
3. Used to access the current class's methods.	3. Used to access parent class method.
4. Used for pointing the current class instance	4. Used for pointing the superclass instance
5. Must be the first line of a block	5. Must be the first line of a block

20. What is the difference between break and continue statements?

Below are the key differences between break and continue.

Break	Continue
1. Switch and loop (for, while, do while) statements are supported.	1. Only works with loop statements.
2. When it is executed, it causes the switch or loop statements to terminate.	2. It does not end the loop, but instead causes it to jump to the next iteration.
3. It immediately terminates the innermost enclosing loop or switch.	3. When a continue is used within a loop that is nested with a switch, the next loop iteration is executed.

21. What is a copy constructor in Java?

A member function called copy constructor is used to initialize an object with another object of the same class. Because all objects in Java are passed by reference, there is no need for a copy constructor. Furthermore, Java does not provide automated pass-by-value pass-through.

class Main {int empId;String empName;public Main(int id, String name) {this.empId = id;this.empName = name;}//Copy constructorpublic Main(Main emp) {empId = emp.empId;empName = emp.empName;}public void displayDetails() {System.out.println("Employee ID:" + empId);System.out.println("Employee Name:" + empName);}public static void main(String[] args) {Main e1 = new Main(85, "Saurav");//Create a copy of another objectMain e2 = new Main(e1);e1.displayDetails();e2.displayDetails();}}

Employee ID:85

Employee Name:Saurav

Employee ID:85

Employee Name:Saurav

22. Is it possible to use a non-static variable in a static context?

No, non-static variables cannot be referenced directly in a static environment; instead, objects must be used.

23. Why do we mark main method as static in Java?

In Java, the main method is designated as static so that the JVM may call it to start the application. As a result, marking the main method static in Java is a well-known practice. However, if the static is removed, there will be uncertainty. It's possible that a Java process won't know which method of a class to call to begin the program. As a result,this practice assists in determining the beginning code for a program in a class that is supplied as an argument to the Java process.

24. What is Inheritance?

In Object-Oriented Programming, inheritance is a crucial notion. Certain qualities and behaviors are shared by some items. We can place the common behavior and features in a base class, also known as a super class, by using inheritance. Then this base class is inherited by all objects having similar behavior. IS-A relationship is also a representation of it. Code reuse, method overriding, and polymorphism are all aided by inheritance.

25. What are the different types of inheritance in Java?

- Single Inheritance: In single inheritance, one class inherits the properties of another, implying that there will only be one parent and one child class or derived class.
- Multilevel Inheritance: This sort of inheritance occurs when a class is derived from another class that is itself derived from another class, i.e. when a class has more than one parent class but at different levels.
- Hierarchical Inheritance: Hierarchical inheritance occurs when a class has many child classes (subclasses), or when more than one child class inherits from the same parent class.
- Hybrid Inheritance: A hybrid inheritance is one that combines two or more inheritance types.

26. Why is multiple inheritance not supported in java?

Multiple inheritance occurs when a child class inherits properties from several other classes. Extending several classes in Java is not possible.

Multiple inheritance has the drawback of making it difficult for the compiler to decide which method from the child class to execute at runtime if multiple parent classes have the same method name.

As a result, multiple inheritance in Java is not supported. The Diamond Problem is a well-known name for the issue. If you have any difficulties answering these java interview questions, please leave a remark below.

```
import java.io.*;
```

class classP {public void display() {System.out.println("Class P");}}class classQ {public void display() {System.out.println("Class Q");}}class Main extends classP, classQ { public static void main(String[] args) {Main obj = new Main();obj.display();}}

```
Main.java:13: error: '{' expected
class Main extends classP, classQ {
 ^
1 error
```

27. How will you implement method overloading in Java?

Multiple methods with the same name but different parameters might exist in a Java class. It's known as Method Overloading. To implement method overloading, we must construct two methods in a class with the same name and perform one or more of the following:

- By changing the number of parameters.
- By changing the data type of parameter.

28. What is Polymorphism?

"One interface, many implementations" is a precise description of polymorphism. Polymorphism is the ability to ascribe a distinct meaning or usage to something in different situations - especially, the ability to have many forms for an item such as a variable, a function, or an object. There are two types of polymorphism

- Compile Time Polymorphism: Method overloading is compile-time polymorphism

class Calculator

{

int add(int a, int b) {

return a + b;

}

int add(int a, int b, int c) {

```
return a + b + c;

}

}

public class Main

{

public static void main(String args[]) {

Calculator obj = new Calculator();

System.out.println(obj.add(15, 20));

System.out.println(obj.add(14, 36, 30));

}

}
```

35

80

- Run time polymorphism or Dynamic Polymorphism: Inheritance and interfaces are used to implement runtime time polymorphism.

```
class A {

public void my() {

System.out.println("Overridden Method");

}
```

```
}

public class Main extends A {

public void my() {

System.out.println("Overriding Method");

}

public static void main(String args[]) {

A obj = new Main();

obj.my();

}

}
```

29. Can we change the scope of the overridden methods in the subclass?

Yes, we can adjust the scope of the subclass's overridden methods as long as we don't make it less accessible.

- Protected, default, and public can all be changed from private.
- Protected can be changed to default or public.
- It is possible to alter a default modifier to public.
- Only public can be declared with the public modifier.

30. What is the difference between method overloading and method overriding in Java?

In Method Overloading, methods of the same class shares the same name but each method must have a different number of parameters or parameters having different types and order.

Method Overloading is to “add” or “extend” more to the method’s behavior.

It is a compile-time polymorphism.

The methods must have a different signature.

It may or may not need inheritance in Method Overloading.

In Method Overriding, the subclass has the same method with the same name and exactly the same number and type of parameters and same return type as a superclass.

Method Overriding is to “Change” existing behavior of the method.

It is a run time polymorphism.

The methods must have the same signature.

It always requires inheritance in Method Overriding.

31. What is encapsulation in Java?

Encapsulation is a technique for combining data (variables) and code (methods) into a single entity. The data is hidden from the rest of the world and may only be accessed through the methods of the current class. This aids in the protection of data from unnecessary alteration.

In Java, we can achieve encapsulation by:

- Declaring a class's variables as private; and
- Providing public setter and getter methods to edit and inspect the variables' values.

32. What is an abstraction in Java?

Abstraction is the process of hiding some implementation details of an object from the outside world and only displaying the object's key qualities.

It differs from Java's Abstract class. The abstraction process identifies commonality while masking the implementation's complexity. It allows us to concentrate on the interface with the outer world.

There are two approaches to create abstraction:

- Classes that are abstract (0-100 percent of abstraction can be achieved)
- Interfaces (it is possible to achieve 100% abstraction).

33. How to create singleton class in java?

A singleton class is one for which only one object can be created. You can make a singleton class by making its constructor private, which restricts the object's creation. Provide a static method for getting an object instance, so you may handle object creation entirely within the class. We're using a static block to create an object in this example.

```
public class Main { private static Main myObj;static{myObj = new Main();}private Main(){}public static Main getInstance(){return myObj;}public void test(){System.out.println("Hello");}public static void main(String a[]){Main ms = getInstance();ms.test();}}
```

34. Can you override a private or static method in Java?

In Java, you can't override a private or static method. If you construct an identical method in a child class with the same return type and method arguments, the superclass method will be hidden; this is known as method hiding. A private method cannot be overridden in a subclass since it is not accessible there. You can do this by creating another private method in the child class with the same name.

35. Explain the difference between static and dynamic binding

Binding is a technique that establishes a connection between a method's call and its actual implementation. At both compile and run time, object forms can be resolved. Static binding occurs when the link between the method call and the method implementation is resolved at compile-time, whereas dynamic binding occurs when the connection is resolved at run time.

Access Modifier	Within class	Within Package	Outside package without inheritance	Outside package with inheritance
Default	✔	✔	✘	✘
Private	✔	✘	✘	✘
Protected	✔	✔	✘	✔
Public	✔	✔	✔	✔

36. What do you mean by an interface in Java?

In Java, an interface is a blueprint for a class or a collection of abstract methods and static constants. Each method in an interface is public and abstract, but it lacks a constructor. As a result, an interface is essentially a collection of connected methods with empty bodies.

Example:

```
public interface Animal {

public void eat();

public void sleep();

public void run();

}
```

37. Is it allowed to mark a method abstract as well as a final?

No. Marking a method abstract as well as the final will be a contradiction in terms. A child class must implement an abstract method. And you can't override a final method. As a result, in Java, a method might be final or abstract.

38. Is it allowed to mark an interface method as static?

Yes, we may define static and default methods in an interface starting with Java 8. It was not permitted before to Java 8, because All methods in an interface are explicitly abstract and hence you cannot define them as static or final because static or final methods cannot be abstract.

39. What is the difference between abstract class and interface in Java?

Below are the key differences between abstract class and interface

Abstract Class	Interface
An abstract class can include both complete default code and information that must be customized.	An interface can only give the signature, not any code.
Non-abstract methods can exist in an abstract class.	An Interface's methods are all abstract.
A class can only extend one abstract class in the case of an abstract class	Several interfaces can be implemented by a single class.
Instance variables can exist in an abstract class.	Instance variables are not allowed in an interface.
The abstract keyword is used to declare an abstract class.	The interface keyword is used to declare the interface.

40. What are the main uses of 'this' keyword?

- This keyword can be used in the following ways.

- This can be used to refer to the current instance variable of the current class.
- This can be used to call the method of the current class (implicitly)
- This() can be used to call the constructor of the current class.
- In the method call, this can be given as an argument.
- This can be provided to the constructor as an argument.
- This can be used to get the current instance of the class from a method.

41. Can a class be marked final in java?

In Java, a class can be tagged as final. A class that has been declared as final cannot be prolonged. We cannot extend a final class and does not support inheritance.

```
final class Vehicle {

public void speed() {

System.out.println("Default speed");

}

}

public class Car extends Vehicle{

public static void main(String[] args) {

}

}
```

The type Car cannot subclass the final class Vehicle

42. Is it allowed to declare main method as final?

In Java, the main () method can be declared final. There are no errors thrown by the compiler.

When we use the final keyword to declare a method as final, that method becomes the final method. The final method is mostly used in Java because it is not overwritten.

43. What is the purpose of package in Java?

A package is a container for a collection of classes, interfaces, and sub-packages. It is frequently a hierarchical framework for storing data. This makes it easy to manage the relevant classes and sub-packages.

A Package also protects classes and interfaces against unauthorized access. A package also aids in the avoidance of naming conflicts.

44. What is a static import in Java?

Static import is similar to normal import declaration. Normal import allows us to import classes from packages without using package qualifiers. A static import allows us to import static members from a class without using a class qualifier.

import static java.lang.Math.*;import static java.lang.System.*;public class Main {public static void main(String[] args) {out.println(sqrt(16));out.println(abs(6.9));out.println(pow(5, 2));}}

4.0

6.9

25.0

45. What is the purpose of serialization?

Serialization can be used for a variety of purposes, including:

- Communication: It is used to send data between two machines across a network.
- Persistence: We can save the state of an object in a database and recover it later.

- Caching: To boost performance, serialization can be utilized for caching. It could take us 10 minutes for object creation, but just 10 seconds to de-serialize it.
- Cross JVM Synchronization: It can be utilized in the same way across multiple JVMs with various architectures.

46. What are the uses of Reflection in Java?

Reflection is a runtime API for inspecting and changing the behavior of methods, classes, and interfaces. Reflection informs us about the class to which an object belongs, as well as the methods of that class that can be used with the object. We can call methods at runtime using reflection, regardless of the access specifier provided.

47. What is Garbage Collection in Java?

The method through which Java applications do automated memory management is known as garbage collection. Java applications are compiled into bytecode that may be executed by a Java Virtual Machine, or JVM. Objects are produced on the heap, which is a part of memory devoted to the Java application, while it runs on the JVM. Some items will become obsolete over time. To free up memory, the garbage collector detects these useless objects and deletes them.

48. What are the different types of garbage collectors in Java?

Garbage collection is Java software that aids in the management of implicit memory. Because the new keyword in Java allows you to build dynamically formed objects, which will use memory once they are created. When the operation is finished and there are no more references to the object, Java uses garbage collection to delete it and free up the memory it has taken up. Garbage collectors come in four types in Java:

- Serial Garbage Collector
- Parallel Garbage Collector
- CMS Garbage Collector
- G1 Garbage Collector

49. Why do we use finalize() method in Java?

To do any cleaning before Garbage Collection, Java provides the finalise() function. This function is found in the Object class and is used by the JVM internally. In the case of Garbage Collection, developers are able to use this approach for any custom cleaning.

This method may not be invoked if an Object is not Garbage Collected. The JVM never calls this procedure more than once.

50. What is the difference between a Inner class and sub class in Java?

Any private instance variable of the outer class can be accessed by the nested inner class. We can use the access modifiers private, protected, public, and default, just like any other instance variable.

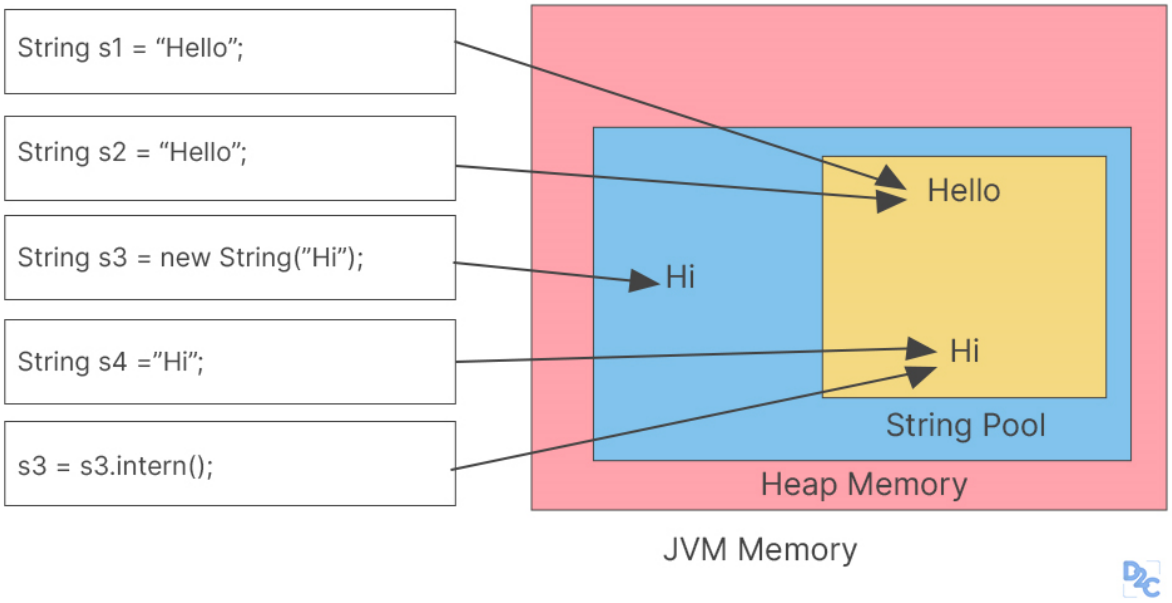
class Outer { class Inner { public void show() { System.out.println("In a nested class method"); } } } class Main { public static void main(String[] args) { Outer.Inner in = new Outer().new Inner(); in.show(); } }

A class that inherits a method or methods from a superclass is known as a subclass.

class Car {//...} class HybridCar extends Car {//...}

51. What is Java String Pool?

The term "string pool" in Java refers to a group of Strings stored in heap memory. When a new item is produced, the String pool first checks to see if the object is already present in the pool. If it is present, the same reference is returned to the variable; otherwise, a new object in the String pool is created and the appropriate reference is returned.



public class Main {

public static void main(String[] args) {String s1 = "Hello";String s2 = "Hello";String s3 = new String("Hi");String s4 = "Hi";System.out.println("s1 == s2 : " +(s1==s2)); // trueSystem.out.println("s3 == s4 : " +(s3==s4)); // false}}

s1 == s2 : true

s3 == s4 : false

52. Why a String object is considered immutable in java?

String objects are immutable in Java, which basically implies that their state cannot be changed after they are created. Java produces a new string object whenever you try to edit the value of that object instead of modifying the values of that particular object. Java Because string objects are often stored in the String pool, they are immutable. Because String literals are frequently shared among numerous clients, one client's behavior may have an impact on the others. It improves the application's security, caching, synchronization, and speed.

53. What is bufferreader in java?

The Java.io.BufferedReader class reads text from a character-input stream, buffering characters so that characters, clusters, and lines can be read efficiently. The following are the main features of BufferedReader: The cradle size can be set or the default size can be used.

54. What is the basic difference between a String and StringBuffer object?

Immutability is a property of the String class. The equals() method of the Object class is overridden by the String class. As a result, the equals() method can be used to compare the contents of two strings. When executing a concatenation operation, the String class is slower.

StringBuffer is a mutable class. The equals() function of the Object class is not overridden by the StringBuffer class. When performing concatenation operations, the StringBuffer class is faster.

public class Main{ public static void main(String[] args){ StringBuffer buffer=new StringBuffer("Hello"); buffer.append(" World"); System.out.println(buffer); } }

55. What is the use of toString() method in java

The function toString() function returns the object's String representation. When you print an object, the Java compiler calls the object's function toString() function internally. So, depending on the implementation, overriding the function toString() function produces the desired output, which may be the state of an object, for example. We may, however, override this function to get the format we wish to output.

56. Difference between String Buffer and StringBuilder

Below are the key differences between StringBuffer and StringBuilder

No.	StringBuffer	StringBuilder
1)	StringBuffer is thread-safe since it is synchronized. It indicates that two threads can't call the StringBuffer functions at the same time. .	StringBuilder is not thread safe since it is not synchronised. It indicates that two threads can call StringBuilder's methods at the same time..

2)	StringBuffer is *less efficient* than StringBuilder.	StringBuilder is *more efficient* than StringBuffer.
3)	In Java 1.0, the StringBuffer class was introduced	In Java 1.5, the StringBuilder class was introduced.

57. What is the difference between Error and Exception?

An error is an unrecoverable condition that occurs during the execution of a program. For example, an OutOfMemoryException. You can't fix JVM issues in the middle of a game. Although an error can be caught in the catch block, the application's execution will be halted and will not be recovered.

Exceptions, on the other hand, are conditions that arise as a result of faulty input, human error, or other factors. It disrupts the program's normal flow of execution. For example, if the given file does not exist, a FileNotFoundException will be issued. If you try to use a null reference, you'll get a NullPointerException.

58. In Java, what are the differences between a Checked and Unchecked Exception?

- Checked Exception: Checked exceptions are all classes that extend the Throwable class except RuntimeException and Error. Checked exceptions are checked during the compilation process. IOException, SQLException, and other exceptions are examples.
- Unchecked Exception: Unchecked exceptions are the classes that extend RuntimeException. At compilation time, unchecked exceptions are not checked. ArithmeticException, NullPointerException, and other exceptions are examples.

59. What are the differences between throw and throws?

Below are the key differences between throw and throws keyword

throw keyword	throws keyword
To explicitly throw an exception, use the throw keyword.	To declare an exception, use the throws keyword.

Throwing only will not propagate checked exceptions.	Throws can be used to propagate checked exceptions.
Throw is followed by an instance.	Throws is followed by class.
Throw is used within the method.	The method signature includes the keyword throws.
You cannot throw multiple exception	Multiple exceptions can be declared, for example, public void method() throws IOException, SQLException.

60. What is the use of finally block in Java?

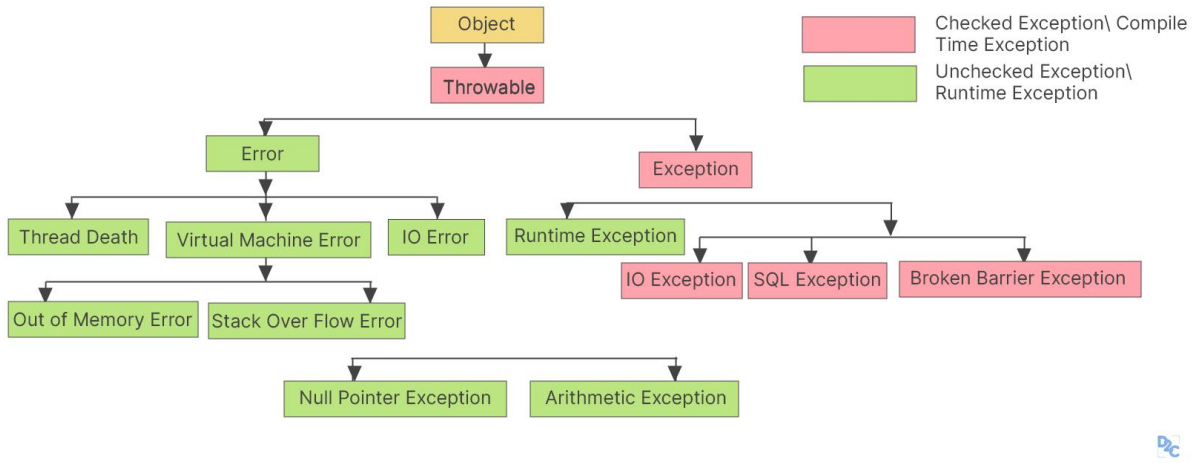
A finally block is always performed in Java, according to the standard. whether or not an error happens, whether or not an exception is handled not. It aids in the cleaning of transactions such as rollbacks and closes. Make a connection, close a file, and so forth.

```
try {

int var = 6;

} catch (Exception exception) {

System.out.println("Exception occurred");

} finally {

System.out.println("Execution of finally block");

}
```

61. What is the exception hierarchy in java?

The following is the order of precedence:

All Exception classes have a parent class called Throwable. Checked exceptions and UncheckedExceptions or RunTimeExceptions are the two sorts of exceptions. Both types of exceptions belong to the Exception class, whereas errors are divided into two categories: Virtual Machine and Assertion errors.



62. Do we have to always put a catch block after a try block?

The rule of placing a catch block after a try block is not enforced in Java. After a try block, we may write a catch or finally block. In the catch block, we describe any exceptions we want to capture.

63. How Multi-threading works in Java?

Multi - threading in Java is the process of running two or more threads at the same time to maximize CPU utilization. Multithreaded programs have two or more threads running at the same time. As a result, Concurrency is another name for it in Java. Each thread runs in the same direction as the others. Because several application threads do not allocate distinct memory areas, memory is saved. It also takes less time to transition between threads' contexts. To maintain a multi-threading environment, Java includes methods such as start(), notify(), wait(), sleep(), and so on.

64. What is a Thread? What are the two ways to create a thread?

A thread is the smallest unit of specified instructions that a scheduler can run independently. Every Java program execution will contain at least one thread, referred to as the main thread.

When the JVM starts executing the application, it creates this main thread. The main thread is used to call the program's main() function. Threads can be created in Java in one of two ways:

By implementing the Runnable interface

```
class Main implements Runnable{ public void run(){ System.out.println("Thread runs."); } public static void main(String args[]){ Thread ib = new Thread(new Main()); ib.start(); } }
```

Thread runs.

By extending the Thread class.

```
class Main extends Thread{ public void run(){ System.out.println("Thread runs."); } public static void main(String args[]){ Main ib = new Main(); ib.start(); } }
```

Thread runs.

65. What is a Thread’s priority and how it is used in scheduling?

Every Thread in Java has a priority. This priority is given a numerical value. When scheduling, the priority value is utilized to select the thread with the highest priority for execution. Threads with a higher priority receive more execution priority than threads with a lower priority.

The task scheduler prioritizes the threads with the highest priority first, then the threads with the lowest priority.

66. Explain the thread lifecycle in Java?

A thread's life cycle has five phases, and java can be in any of them. The thread can be found in any of the following states:

- New: After it has been generated, the thread is in a new state. Until you start the execution procedure, it will remain in the new state.

- Running: When a thread is in the runnable state, it is ready to run at any time or has already started running.
- Waiting: Another thread may be running in the system while the current thread is waiting to execute. As a result, the thread enters the waiting state.
- Dead: When a thread's execution is complete, its status is converted to dead, which indicates it is no longer considered active.

67. What is synchronization?

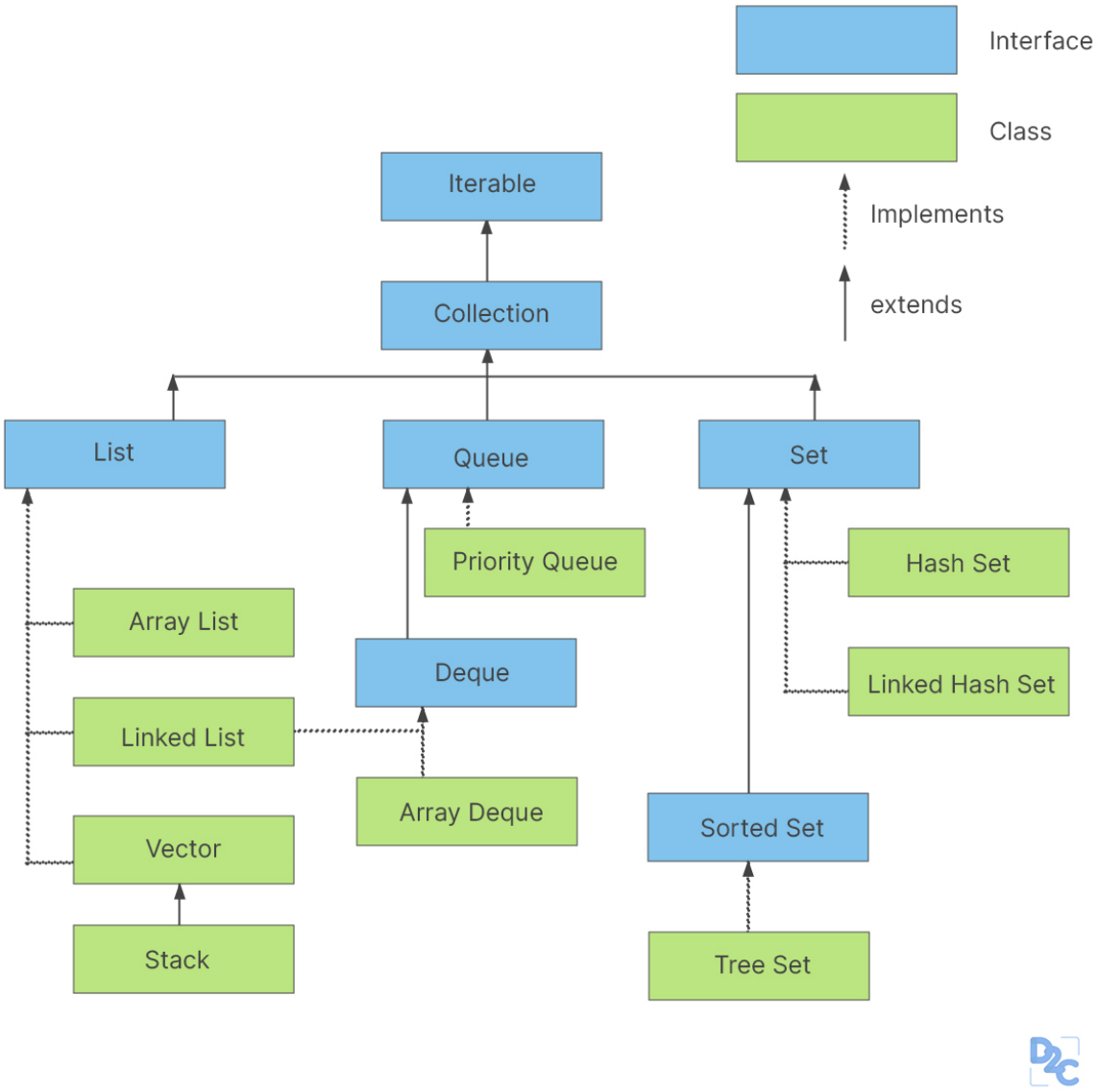
Multi-threading is referred to as synchronization. Only one thread can execute a synchronized block of code at a time. Because Java allows many threads to run simultaneously, two or more threads may access the same fields or objects. Synchronization is a mechanism that ensures that all concurrent threads are running at the same time. The synchronization prevents memory consistency issues caused by an incorrect shared memory view. When a method is marked as synchronized, the monitor for the method's object is held by the thread. If the synchronized method is being executed by another thread, the thread is blocked until that thread releases the monitor.

68. What is a collection class in Java? List down its methods and interfaces

The collection is a framework in Java that works as an architecture for storing and managing a collection of objects. You may use Collections to conduct a variety of operations like searching, sorting, insertion, manipulation, and deletion. The following are some of the features of the Java collection framework:

- Classes
- Methods
- Interfaces

The Java Collection hierarchy is depicted in the graphic below.



69. What is the difference between Collection and Collections Framework in Java?

A Collection is a Java object that holds several elements of the same type in one unit. A single Collection object can access all of these aspects.

Collections Framework in Java is a library that provides a standard architecture for building, updating, and accessing many sorts of collections. There are standard ways for working on a Collection object in the Collections framework that are often used by developers.

70. What are the differences between a List and Set collection in Java?

	List	Set
1.	The list implementation allows us to add the same or duplicate elements.	The set implementation doesn't allow us to add the same or duplicate elements.
2.	The insertion order is maintained by the List.	It doesn't maintain the insertion order of elements.
3.	List allows us to add any number of null values.	Set allows us to add at least one null value in it.
4.	The List implementation classes are LinkedList and ArrayList.	The Set implementation classes are TreeSet, HashSet and LinkedHashSet.

71. In Java, how will you decide when to use a List, Set or a Map collection

Set should be your first choice if you don't want duplicate values in your database because none of its classes allow them.

List (ArrayList) is a better alternative if frequent search operations based on index values are required.

The List is also a suitable collection interface if the insertion order needs to be maintained.

If you need to store the key and value mappings in the database, Map is your best option.

72. What is the difference between Arraylist and vector in Java?

Below are the key differences between Arraylist and vector

Arraylist	Vector
There is no synchronization in the Array List.	Vector is synchronized.
Because it is non-synchronized, the Array List is quick.	Because Vector is thread-safe, it is sluggish..

When an element is added to the Array List, it increases its Array size by 50%.	By default, a vector's array is doubled in size.
The increment size is not specified in the Array List.	The increment size is defined by a vector.
Array List can only use Iterator for traversing an Array List.	Vector can use both Enumeration and Iterator for traversing.

73. What are the differences between Comparable and Comparator?

Comparable <T> is a Java interface that specifies the type of objects with which this object can be compared. A single sorting sequence is provided by Comparable. To put it another way, we can sort the collection by a single attribute such as id, name, or price. To sort elements, Comparable provides the compareTo() method. In java.lang package, there is a class called Comparable.

Comparator <T> is likewise an interface, with T denoting the type of objects that this comparator may compare. Multiple sorting sequences are available in the Comparator. To put it another way, we can sort the collection based on different criteria such as id, name, and price. To order elements, the Comparator provides the compare() method. The java.util package includes a Comparator.

74. What are the differences between a HashMap and a Hashtable in Java?

Below are the key differences between HashMap and HashTable

HashMap	Hashtable
HashMap is not a synchronized collection. If it is used in a multi-thread environment, it may not provide thread safety.	A Hashtable is a synchronized collection. Not more than one thread can access a Hashtable at a given moment of time..

One null key and numerous null values are allowed in HashMap	There is no such thing as a null key or value in a hashtable.
HashMap is a new class introduced in JDK 1.2.	Hashtable is a legacy class.
A HashMap implementation by LinkedHashMap maintains the insertion order of elements	A TreeMap sorts the mappings based on the ascending order of keys.
Iterator traverses the HashMap.	Enumerator and Iterator traverse the hashtable.
HashMap inherits AbstractMap class.	Hashtable inherits Dictionary class.

75. What are the differences between a HashMap and a TreeMap?

Below are the key differences between HashMap and TreeMap

HashMap	TreeMap
The Java HashMap implementation of the Map interface is based on hashtables	Java TreeMap is a Map interface implementation based on a Tree structure.
HashMap implements Map, Cloneable, and Serializable interfaces.	TreeMap implements NavigableMap, Cloneable, and Serializable interface.
A single null key and numerous null values are allowed in HashMap	TreeMap does not allow null keys, however multiple null values are allowed.
HashMap allows heterogeneous elements because it does not perform sorting on keys.	TreeMap allows homogeneous values as a key because of sorting.

The HashMap class uses the hash table.	TreeMap internally uses a Red-Black tree, which is a self-balancing Binary Search Tree.
A HashMap uses equals() method to compare keys.	A TreeMap uses compareTo() method for maintaining natural ordering.
A HashMap gives constant time performance for operations like get() and put().	A TreeMap gives order of log(n) time performance for get() and put() methods.

76. What are Wrapper classes in Java?

Wrapper classes are used in Java to allow primitive types to be accessed as objects. Boolean, Integer, Double, Float, and other primitive types have corresponding Wrappers classes – Boolean, Integer, Double, Float, and so on. The automatic conversion of primitive data types into their equivalent Wrapper type is known as boxing and the opposite operation is known as unboxing.

The java.lang package contains many of these Wrapper classes.

In Java 5.0, the idea of Autoboxing and Unboxing for Wrapper classes was introduced.

Primitive Data Type	Wrapper Class
Char	Character
Byte	Byte
Short	Short
int	Integer
Long	Long
Float	Float
Double	Double
Boolean	Boolean

77. What is the purpose of the native method in Java?

When applied to a method, the native keyword indicates that the method is implemented in native code using JNI (Java Native Interface).

As a result, native methods enable Java developers to use platform-specific APIs directly.

78. What is the difference between Shallow Copy and Deep Copy in Java?

When we copy an entity to generate two or more entities, we can say we've done a shallow copy because changes in one entity are mirrored in the other entities as well. In shallow copy, new memory allocation for the other entities is never done, and the sole reference to the other entities is duplicated. When we make a deep copy of an entity to produce two or more entities where changes in one entity do not affect the other, we are referring to it as a deep copy. For the other entities, a fresh memory allocation occurs during the deep copy, and references to the other entities are not duplicated.

79. What is the difference between Collection and Collections Framework in Java?

A Collection is a Java object that holds several elements of the same type in one unit. A single Collection object can access all of these aspects.

Collections Framework in Java is a library that provides a standard architecture for building, updating, and accessing many sorts of collections. There are standard ways for working on a Collection object in the Collections framework that are often used by developers.

80. Why Map interface does not extend the Collection interface in the Java Collections Framework?

Because Map requires both a key and a value, it is incompatible with the Collection interface. For example, if we wish to add a key-value pair, we will use put (Object key, Object value).

To add an element to a HashMap object, you'll need two parameters. Add(Object o) has only one parameter in the Collection interface.

The other reason is that Map supports valueSet, keySet, and other relevant methods with views that differ from the Collection interface.

81. What is the difference between an Iterator and ListIterator in Java?

In Java, there are two interfaces for traversing data structures: Iterator and ListIterator. The following are the distinctions between the two:

Only a List can be traversed with ListIterator. Iterator, on the other hand, can traverse Lists, Sets, and Queues, among other things.

An Iterator only moves through the elements in one direction. It just happens. The components of a ListIterator can be traversed in both backward and forward directions.

We can't get the index of a Data Structure element via an iterator. When traversing a ListIterator, we can use methods like nextIndex() and previousIndex() to determine the index of an entry.

Iterator cannot be used to replace the value of an existing element. The method set(e) in ListIterator is used to replace the value of the last element returned by the next() and previous() methods.

82. What is the difference between a Set and a Map in Java?

In Java, Set is used to construct the mathematical Set. It can't have values that are repeated. Adding the same components to a set is not possible. Only the unique value is stored in each class that implements the Set interface. Using the keyset() and entryset() methods, we can quickly iterate the Set components. The Set interface does not keep track of insertion order. Some of its classes, such as LinkedHashSet, however, keep the insertion order.

The map function is used to map data in a database. It is possible for different keys to have the same value. The map has a unique key and values that are repeated. One or more keys in a Map can have the same value, but two keys cannot. It is not possible to iterate across map elements. To iterate the elements, we must convert Map to Set. The Map does not keep track of the insertion sequence. Some Map classes, such as TreeMap and LinkedHashMap, do the same thing.

83. What is Hash Collision? How Java handles hash-collision in HashMap?

Two separate items may have the same HashCode in a Hashing scenario, yet they may not be equal. As a result, when storing two separate objects with the same Hash Code in a HashMap, Java will encounter an issue. Hash Collision is the term for a situation like this.

Hash Collision can be resolved or avoided using a variety of ways. In HashMap, however, in the event of a Hash Collision, Java simply replaces the Object at the old Key with a new Object.

84. What is the difference between Queue and Stack data structures?

A FIFO data structure is a queue. First In, First Out is the acronym for First In, First Out. It means that the piece that was added first will be the first to be withdrawn from the queue. A line for purchasing tickets at a train station is an example of a Queue in the real world. The individual who enters the queue first is serviced first.

A LIFO data structure is a stack. Last In First Out (LIFO) is an acronym meaning Last In First Out. The piece inserted last is the first to be deleted from the collection. Elements are added or deleted from the top of the stack in a Stack. The back button on a browser is an example of Stack in action. We can go back one by one only and it works in the reverse order of adding webpages to history.

85. What is a classloader?

The Java Classloader is a component of the Java Runtime Environment (JRE) that loads classes into the Java Virtual Machine when they are needed (JVM).

Three types of class loaders are utilised when the JVM is started:

1. Bootstrap Classloader: It loads the rt.jar classes from the core Java API file.

2. Classloader for Extensions: It loads jar files from the lib/ext subdirectory.

3. System/Application Classloader: It loads jar files from the CLASSPATH environment variable's provided directory.

Classes can come from a local file system, a remote file system, or even the internet.

86. What is the covariant return type?

It is possible for a child class to have a differentt return type for an overriding method, but the child's return type must be a sub-type of the parent's return type. In terms of return type, the overriding method becomes variant.

```
class A{Object m1(){return new Object();}}

class B extends A{String m1(){return new String("Saurav");}}

class Main{public static void main (String[] args) {B obj = new
B();System.out.println(obj.m1());}}
```

Saurav.

87. How to create an immutable class in java?

- Make the class final to prevent it from being extended.
- Make all fields private in order to prevent direct access.
- Variables should not have any setter methods.
- Make all changeable fields final, so that their value may only be changed once.
- Create a constructor that does a deep copy and initialises all fields.
- In the getter methods, clone objects to return a copy instead of the original object reference.

88. Can you have virtual functions in Java?

In Java, all non-static methods are by default virtual functions. Only methods marked with the keyword final, which cannot be overridden, along with private methods, which are not inherited, are non-virtual.

89. Mention the uses of the synchronized block

The synchronized block is used because:

- It aids in the locking of an object for each shared resource.The synchronized block has a smaller scope than the method.

90. Distinguish between static loading and dynamic class loading?

Static class loading is the process of creating objects and instances using the new keyword. At compilation time, the class definition is retrieved and the object is instantiated.

class TestClass {public static void main(String args[]) {TestClass tc = new TestClass();}}

Dynamic Class loading: The Class.forName() function is used to load classes dynamically. When the name of the class is unknown at build time, dynamic class loading is used.

Class.forName (String className);

91. What are different scenarios causing "Exception in thread main"?

The following are some examples of frequent main thread exceptions:

- Exception in thread main java.lang.UnsupportedClassVersionError: This exception occurs when you try to run a java class that was compiled with a different JDK version.
- Exception in thread main java.lang.NoClassDefFoundError: This exception comes in two flavors. The first is where you give the whole name of the class, including the.class extension. When Class is not identified, the second situation occurs.
- Exception in thread main java.lang.NoSuchMethodError: main: When you try to launch a class that doesn't have a main function, you'll get this exception.
- Exception in thread "main" java.lang.ArithmeticException: When an exception is thrown from the main method, the exception is sent to the console. The first section specifies that the main method throws an exception, the second part publishes the exception class name, and the third part displays the exception message after a colon.

92. Explain the Externalizable interface.

Controlling the serialization process is easier with the Externalizable interface. The readExternal and writeExternal functions are part of an "extensible" interface. Externalizable

serialization imposes additional responsibilities on the programmer, but it typically results in superior performance. Because of the entire control over serialization logic, it's easier to evaluate and adjust the class structure.

93. Can we execute any code, even before the main method? Explain.

Yes, we may run any code before calling the main function. When constructing objects at class load time, we will use a static piece of code in the class. Any statements included in this static block of code block will be performed at the same time as the class is loaded, even before any objects are created in the main function.

```
public class Main {static int k;static {System.out.println("Inside static block");k =
10;}public static void main(String[] args) {System.out.println("Inside main
method");System.out.println("Value of k: " + k);}}
```

```
Inside static block

Inside main method

Value of k: 10
```

94. What is the difference between transient and volatile variables in Java?

Transient: The transient modifier instructs the Java object serialization subsystem to skip the field while serializing a class instance. The field will be initialized to the default value when the object is deserialized, which is null for a reference type and zero or false for a primitive type.

Volatile: The volatile modifier instructs the JVM that updates to the field should always be synchronously flushed to memory, while reads should always be read from memory. This implies that fields defined as volatile in a multi-thread program can be reliably accessed and modified without the need for native or standard library-based synchronization.

95. What is a functional interface in java?

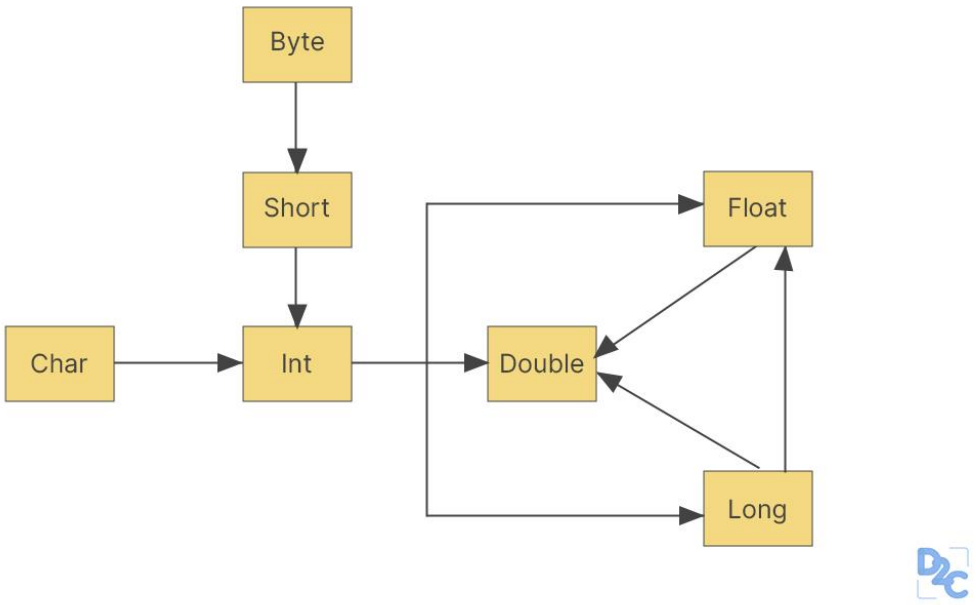
In Java, a functional interface is one that has just one abstract (unimplemented) method. In addition to the one unimplemented method, a functional interface can include default and static methods with implementations. java.lang.Runnable and java.util.concurrent are new in Java 8. Both Callable and Functional interfaces are quite popular.

96. What are the uses of the Java super keyword?

- Access the parent class variable
- To invoke the parent class method
- To invoke the parent class constructors with and without argument.

97. What is method overloading with type promotion?

Method overloading with type promotion means, when no matching parameter type method is found, it implicitly promotes the data type to the next level and calls that corresponding method. For example, if we call a method with a short parameter data type and if there is no matching method with a short type but if int is present, then it calls the method with int parameter datatype.



class Main{void show(int a){System.out.println("int method");}void show(String a){System.out.println("string method");}public static void main(String[] args) {Main s = new Main();s.show('a');}}

98. What is Java instanceOf operator?

Java instanceOf operator is a comparison operator that checks if the object is an instance of some type of class. The return value is either true or false. If there is no value, then it returns false.

public class Main {public static void main(String[] args) {String value = "";Boolean k;k = (value instanceof String);System.out.println(k);}}

true

99. What is the use of System class and Runtime class?

The System class can be used to access system resources such as Standard input and output. For the System class, we are unable to construct an instance. Standard input, Standard output, and Standard error are all supported.

The Runtime class is an Object subclass that gives information about the current running environment. To retrieve a reference to the current runtime object, we can utilise the getRuntime() method. It has a number of methods, such as gc(), exit(), and halt().

100. What is the difference between abstraction and encapsulation?

Abstraction	Encapsulation
Abstraction is the process of hiding implementation details from the user and only displaying functionality.	The technique of encapsulating code and data into a single unit is known as encapsulation.
Abstraction allows you to concentrate on the object's function rather than how it performs it.	Encapsulation gives you control over your data while also protecting it from outside threats.
In the Design Level, abstraction solves the problem.	The problem in the Implementation Level is solved through encapsulation.

Interfaces and Abstract Classes are used to implement abstraction.	Access Modifiers are used to implement encapsulation (private, default, protected, public)
Abstraction refers to the use of interfaces and abstract classes to hide implementation difficulties.	Encapsulation refers to the use of setters and getters to hide data.

101. What are the different types of access specifiers in Java?

There are four different types of access specifiers in Java:

- public: Any class in any package can see it. For a class, variable, or method, we can declare a public access type.
- protected: It can be accessed from any class in the same package or from the subclass of the declared class. We can gain access to the outside world through inheritance.
- default: The scope is contained within the package and does not require the use of any keywords.
- private: It can only be accessible by members of the same class.

Java's easy-to-use syntax and built-in capabilities, as well as the reliability it gives to applications, are the primary reasons for its growing popularity in the software community. These 101 questions will cover up the important topics of Java and will surely help you in acing the interviews.