**1.what is abstract class in java?**

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

Java abstract class is a class that can not be initiated by itself, it needs to be subclassed by another class to use its properties.

An abstract class is declared using the “abstract” keyword in its class definition.

**Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class

Eg:

**abstract** **class** Shape {

**public** **abstract** **void** draw();

}

**class** Circle **extends** Shape {

@Override

**public** **void** draw() {

System.out.println("Drawing a circle");

}

}

**class** Rectangle **extends** Shape {

@Override

**public** **void** draw() {

System.out.println("Drawing a rectangle");

}

}

**public** **class** AbstractClass{

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

Shape circle = **new** Circle();

Shape rectangle = **new** Rectangle();

circle.draw(); // Output: Drawing a circle

rectangle.draw(); // Output: Drawing a rectangle

}

}

**2. Inheritance in java**

**3:Types of Inheritances:**

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only.

**4. why multiple Inheritance is not allowed in java**

Multiple Inheritance is a feature of an object-oriented concept, where a class can inherit properties of more than one parent class. The problem occurs when there exist methods with the same signature in both the superclasses and subclass. On calling the method, the compiler cannot determine which class method to be called and even on calling which class method gets the priority.

**5.multithreading**

**Multithreading in**[Java](https://www.javatpoint.com/java-tutorial) is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

 Multithreading can also handle multiple requests from the same user.

* **Multitasking** is a computer's ability to execute two or more concurrent programs. Multithreading makes multitasking possible when it breaks programs into smaller, executable threads. Each thread has the programming elements needed to execute the main program, and the computer executes each thread one at a time.
* **Multiprocessing** uses more than one CPU to speed up overall processing and supports multitasking.

Java Multithreading is mostly used in games, animation, etc.

**6.life cycle of thread?**

In Java, a thread always exists in any one of the following states. These states are:

1. New
2. Active
3. Blocked / Waiting
4. Timed Waiting
5. Terminated

**7.** **can we start the thread twice?**

No. After starting a thread, it can never be started again. If you does so, an IllegalThreadStateException is thrown. In such case, thread will run once but for second time, it will throw exception.

**8.what are the methods in object class**

In general, a **method** is a way to perform some task. Similarly, the **method in Java** is a collection of instructions that performs a specific task. It provides the reusability of code. We can also easily modify code using **methods**.

**9.class I want to print class name and package name how will u do?**

Eg:

class HelloWorld {

public static void main(String[] args) {

HelloWorld main = new HelloWorld();

String className = main.getClass().getName();

System.out.println(className);

}

}

10.**if I create a object of object class and if want to call the clone method is that possible?**

No (The Object class does not implement Cloneable. Hence, we **cannot** call the clone () method for the object of the Object class.)

11.why to use hcode and equals methods

The equals() and hashcode() are the two important methods provided by the **Object** class for comparing objects.

Eg:

**class** Test\_hash\_equal{

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

        String a = "Andrew";

        String b = "Andrew";

**if**(a.equals(b)){   //checking the equality of objects using equals() methods

            System.out.println("a & b are equal variables, and their respective hashvalues are:" + " "+ a.hashCode() + " & " + b.hashCode());

        }

        String c = "Maria";

        String d= "Julie";

**if**(!c.equals(d)){    //checking  the equality of objects using equals() method

            System.out.println("\nc & d are Un-equal variables, and their respective hashvalues are:" + " "+ c.hashCode() + " & " + d.hashCode());

        }

    }

}

**Output:**

a & b are equal variables, and their respective hash values are: 1965574029 & 1965574029

c & d are Un-equal variables, and their respective hash values ar

12.what is the return type of clone method

Object

13.clone method will it do deep cloning or sallow cloning

Shallow copy:

* Default implementation while using the clone() method a shallow copy of the object is created. It means it creates a new instance and copies all the fields of the object to that new **instance where both are referencing to the same memory in heap memory.**

Eg: class GFG

{

int i,j;

}

main(String[] args)

{

// **Copying**

GFG object1 = new GFG() ;

object.i = 5;

object.j = 6;

**GFG object2 = object1 ;** // shallow copying object

**}**

14. if we need deep cloning what we should we do

Deep Copy

When we do a copy of some entity to create two or more than two entities such that changes in one entity are not reflected in the other entities, then we can say we have done a deep copy. In the deep copy, a new memory allocation happens for the other entities, and reference is not copied to the other entities. Each entity has its own independent reference. The following example demonstrates the same.

15. A list has 1000 of elements and if I want to pick one element how to pick that

We have get method

16. What will you pass in the get method?

Index

17. What is the data type of index?

Integer

18. How many elements a list can have?

List can store more that 100000 elements. The list capacity is only bound by the JVM memory capacity or Integer.MAX\_VALUE whichever is less.

19. What is the upper limit of interger?

20. One list has 10 interger how will you sort it?

Using collections and utility class we use sort method.

Is there any other way?

We can pass this array list to a tresset and duplicates will be removed

21. What is serialization?

**Serialization in Java** is a mechanism of writing the state of an object into a byte-stream. It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies.

The reverse operation of serialization is called deserialization where byte-stream is converted into an object. The serialization and deserialization process is platform-independent, it means you can serialize an object on one platform and deserialize it on a different platform.

For serializing the object, we call the **writeObject()** method of ObjectOutputStream class, and for deserialization we call the **readObject()** method of ObjectInputStream class.

We must have to implement the Serializable interface for serializing the object.

22. What is externaliazation?

Java serialization is not much effective when we have bloated objects with multiple attributes and properties. Here, externalization comes into role. It allows us o customize the serialization. For example, if we have implemented the Serialization interface in a class, we can externalize it using the **writeExternal()** method. When users reconstruct an externalized object from their end, an instance of the object will be created using the **readExternal()** method.

Thus, the externalization provides custom serialization, where we can manage our object stream and decide what to store in it.

23. What are the feature jdk 8 has?

Java 8 provides following features for Java Programming:

* Lambda expressions,
* Method references,
* Functional interfaces,
* Stream API,
* Default methods,
* Base64 Encode Decode,
* Static methods in interface,
* Optional class,
* Collectors class,
* ForEach() method,
* Nashorn JavaScript Engine,
* Parallel Array Sorting,
* Type and Repating Annotations,
* IO Enhancements,
* Concurrency Enhancements,
* JDBC Enhancements etc.

24. What is functional interface?

[Java 8 Functional Interfaces - javatpoint](https://www.javatpoint.com/java-8-functional-interfaces)