JAVA Exception:-

Q1. How can you handle exceptions in Java?

Exception handling can be performed using:

- **Try: the set of statements or code which requires monitoring for an exception is kept under this block.**
- **Catch: this block catches all exceptions that were trapped in the try block.**
- **Finally: this block is always performed irrespective of the catching of exceptions in the try or catch block.**

Q2. What is the difference between exception and error in Java?

Errors typically happen while an application is running. For instance, Out of Memory Error occurs in case the JVM runs out of memory. On the other hand, exceptions are mainly caused by the application. For instance, Null Pointer Exception happens when an app tries to get through a null object.

Q3. Why do we need exception handling in Java?

If there is no try and catch block while an exception occurs, the program will terminate. Exception handling ensures the smooth running of a program without program termination.

Q4. Name the different types of exceptions in Java

Based on handling by JVM, there are typically two types of exceptions in Java:

- **Checked: Occur during the compilation. Here, the compiler checks whether the exception is handled and throws an error accordingly.**
- **Unchecked: Occur during program execution. These are not detectable during the compilation process.**

Q5. Can we just use try instead of finally and catch blocks?

No, doing so will show a compilation error. Catch or finally block must always accompany try block. We can remove either finally block or catch block, but never both.

1. Difference between unchecked and checked exceptions in Java

- **Checked Exceptions:** These are exceptions that are checked at compile time. This means the compiler forces the programmer to handle these exceptions explicitly, either by catching them or declaring that the method throws them. Examples include `IOException`, `SQLException`, etc.
- **Unchecked Exceptions:** These are exceptions that are not checked at compile time. They occur at runtime and are typically programming errors or conditions that are outside the control of the program (e.g., `NullPointerException`, `ArrayIndexOutOfBoundsException`, `ClassCastException`). These exceptions do not need to be declared in a method's throws clause.

2. Difference between finally, final, and finalize in Java

- **finally:** It is a block used in exception handling to ensure that a section of code is always executed, whether an exception is thrown or not.
- **final:** It is a keyword used to apply restrictions on class, method, and variable in Java. A `final` class cannot be inherited, a `final` method cannot be overridden, and a `final` variable cannot be reassigned.
- **finalize:** It is a method defined in the `Object` class that performs cleanup operations on an object before it is garbage collected. It is rarely used and generally discouraged due to unpredictable nature and inefficiency in Java's garbage collection mechanism.

3. Define try-with-resources. How does it differ from an ordinary try?

- **Try-with-resources:** It is a Java 7 feature that allows automatic resource management (like streams, connections) by ensuring that each resource is closed at the end of the statement. It simplifies the code and avoids resource leaks.
- **Difference:** In an ordinary `try` statement, you need to explicitly close resources in a `finally` block. With try-with-resources, resources are automatically closed after the try block completes, whether normally or due to an exception, provided they implement `AutoCloseable`.

4. Define RuntimeException. Describe it with an example.

- **RuntimeException:** It is a subclass of `Exception` and represents exceptions that can be avoided with better programming practices. They are unchecked exceptions.

  Example: `NullPointerException`, which occurs when trying to access methods or fields of an object that is `null`.

5. Difference between NoClassDefFoundError and ClassNotFoundException in Java

- **NoClassDefFoundError:** This error occurs when the Java Virtual Machine (JVM) tries to load a class but cannot find the definition of the class at runtime, usually due to a missing class file or a class that was present during compilation but not available during runtime.
- **ClassNotFoundException:** This exception occurs when using methods like `Class.forName()` to load classes dynamically, but the class being loaded is not found in the classpath.

6. Can we throw an exception explicitly or manually?

- **Yes,** you can throw an exception explicitly in Java using the `throw` keyword followed by an instance of an exception class. For example:

  ```
  java
  ```

```
Copy code
throw new IllegalArgumentException("Invalid argument");
```

7. Describe the use of the throw keyword.

- The `throw` keyword is used to explicitly throw an exception within a method or a block of code. It interrupts the normal flow of the program and transfers control to the nearest enclosing `try` block (or `catch` block or method that handles exceptions).

8. Why should we clean up activities such as I/O resources in the finally block?

- Cleaning up activities such as closing I/O resources in the `finally` block ensures that these resources are released regardless of whether an exception occurs or not. This helps in preventing resource leaks and maintaining program stability.

9. Describe OutOfMemoryError in exception handling.

- `OutOfMemoryError` is a subclass of `Error` and not an `Exception`. It occurs when the Java Virtual Machine (JVM) cannot allocate an object because it is out of memory, typically due to a memory leak or insufficient heap space.

10. What is the error of ClassCastException?

- `ClassCastException` occurs when you try to cast an object of one type to another type that is not compatible with the original type.

11. Is there any difference between throw and throws in exception handling in Java?

- Yes, there is a difference:
  - `throw` is used to explicitly throw an exception.
  - `throws` is used in method declarations to specify that the method may throw one or more exceptions, which must be handled by callers or propagated up the call stack.

12. When should we use the printStackTrace() method?

- `printStackTrace()` is used to print the stack trace of an exception to the console. It is helpful for debugging and understanding the sequence of method calls that led to the exception.

13. Provide me with some examples of unchecked exceptions.

- Examples of unchecked exceptions include:
  - `NullPointerException`
  - `ArrayIndexOutOfBoundsException`
  - `IllegalArgumentException`
  - `ArithmeticException`

14. Is it illegal to keep an empty catch?

- While not illegal, keeping an empty `catch` block (`catch (Exception e) {}`) is generally discouraged because it suppresses exceptions without providing any information or handling. It makes debugging and understanding program behavior more difficult.

15. What are the advantages of using exception handling in Java?

- Exception handling in Java provides:
  - Separation of normal code from error-handling code.
  - Propagation of errors up the call stack.
  - Cleaner and more readable code.
  - Ability to gracefully handle errors and recover from exceptional situations.

16. Can checked exceptions occur at compile time?

- Yes, checked exceptions are checked by the compiler at compile time. Methods that throw checked exceptions must either handle them using `try-catch` blocks or declare that they throw those exceptions using the `throws` clause.

17. What happens if a runtime exception occurs?

- If a runtime exception occurs and is not caught and handled by the program, the program terminates abruptly. Runtime exceptions are not required to be caught or declared in a `throws` clause.

18. Describe unreachable catch block error in Java.

- An unreachable catch block error occurs when a `catch` block is written for a specific exception type that cannot be thrown by the code in the corresponding `try` block. This is a compilation error because the catch block will never be executed.

19. In which situation will you not be able to execute the finally block?

- The `finally` block will not be executed in the following situations:
  - If the JVM exits during the execution of the try or catch block.
  - If the thread executing the try or catch block is interrupted or killed.

20. Is it possible to throw a statement inside a static block?

- Yes, it is possible to throw an exception inside a static block. However, care should be taken because static blocks are executed when the class is loaded, and throwing an exception in a static block can prevent the class from being initialized.

21. Define rethrowing.

- Rethrowing refers to catching an exception in one `catch` block and then throwing it again using the `throw` statement. This allows you to handle the exception at one level of the call stack while signaling it to be handled at a higher level.

22. Define user-defined or custom exceptions in Java.

- User-defined or custom exceptions are exceptions created by the programmer by extending the `Exception` class (for checked exceptions) or `RuntimeException` class (for unchecked exceptions). They allow programmers to define specific exception types tailored to their application's needs.

23. What do you understand by a chained exception?

- A chained exception is an exception that is caused by another exception. It provides a way to associate one exception with another, retaining information about the original cause of the error. It is achieved by passing the original exception to the constructor of the new exception.

24. What do you understand about throwables in Java?

- `Throwable` is the superclass of all errors and exceptions in the Java language. It provides methods to get information about an exception (such as its message and stack trace) and to control the flow of execution when an exception occurs.

25. Mention the methods in the `Throwable` class.

- Some methods in the `Throwable` class include:
  - `getMessage()`: Returns the detail message string of this throwable.
  - `printStackTrace()`: Prints this throwable and its backtrace to the standard error stream.
  - `getCause()`: Returns the cause of this throwable.

26. Give me some examples of checked exceptions.

- Examples of checked exceptions include:
  - `IOException`
  - `SQLException`
  - `ClassNotFoundException`

27. Define NumberFormatException exception in Java.

- `NumberFormatException` is an unchecked exception that is thrown when a method in the `Integer`, `Double`, `Float`, `Long`, `Short`, or `Byte` classes is invoked to convert a string to a numeric type, but the format of the input string is not suitable for that numeric type.

28. What do you understand by ArrayIndexOutOfBoundsException?

- `ArrayIndexOutOfBoundsException` is an unchecked exception that occurs when trying to access an element of an array using an index that is either negative or greater than or equal to the size of the array.

29. Suppose there is a catch block in tune with a try block with 3 statements - 1, 2, and 3. Now, imagine that the statement is thrown in statement 2. Will there be an execution of statement 3?

- No, if an exception is thrown in statement 2 and caught in the corresponding `catch` block, statement 3 will not be executed. Control will pass directly to the `catch` block after the exception is thrown.

MultiThreading

1. What is multithreading?

It is a process of concurrently executing multiple threads or parts of a program. Multithreading is used for multitasking purposes. It acquires less memory and provides fast and efficient performance.

2. What is the thread?

A thread is nothing but a lightweight subprocess. It is a different path of execution. Every thread runs in a different stack frame. The process can contain multiple threads. Every thread shares the same memory space.

3. Differentiate between process and thread.

Process	Thread
A program in execution is a process	Thread is a part of a process
Processes are heavyweight	Threads are lightweight
Termination is slow	Termination is fast
Creation takes time	Creation takes less time
Communication between processes needs more time	Communication between threads requires less time
More time for context-switching	Less time for context-switching
Processes do not share memory	Threads share memory
Consumes more resources	Consumes lesser resources

4. What is inter-thread communication?

- Inter-thread communication is nothing but the process of communication between synchronized threads.
- It is used to avoid thread polling in Java.
- The thread is paused running in its critical section, and another thread is allowed to enter the same critical section to be executed.
- inter-thread communication can be achieved by wait(), notify(), and notifyAll() methods.

5. Explain wait() method in Java? Why must it be called from the synchronized block?

It is the method provided by the Object class in Java. It is used for inter-thread communication in Java. java. lang.Object. wait() method is used to pause the current thread and wait until another thread does not call the notify() or notifyAll() method.

Syntax

public final void wait()

6. How will you synchronize a method in Java?

We can use the synchronized modifier in the method's signature or use the synchronized {} construct inside the method.

```
public void swap(){
    synchronized (this){
    //Code
    }
}
public synchronized void swap1EqualsSwap(){
//Code
}
public static synchronized void swap2(){
//Code
}
public static void swap3EqualsSwap2(){
    synchronized (JoinClass.class)  {
          /code
}}
```

7. What are the advantages of multithreading?

- **Improved Performance:** Multithreaded programs can use multi-core processors, making them faster and more efficient.
- **Enhanced Responsiveness:** Multithreading allows for responsive user interfaces (UIs) in applications. Long-running tasks can be moved to background threads, ensuring the UI remains responsive to user input.
- **Resource Utilization:** Multithreading enables better utilization of system resources, as different threads can execute different tasks simultaneously.
- **Parallel Processing:** Multithreading is essential for parallel processing, where a large task is divided into smaller sub-tasks that can be executed concurrently, reducing overall execution time.

8. Discuss the states in the lifecycle of a Thread.

- **New**: A Thread class object is created using a new operator, but the thread is not alive. The thread doesn't start until we call the start() method.
- **Runnable**: The thread is run after calling the start() method. It is waiting in the ready or runnable queue to be allocated to the processor.
- **Running**: In this state, the thread scheduler picks the thread from the ready state and the thread executes.
- **Waiting/Blocked**: In this state, a thread is not running but still alive, or waiting for the other thread to finish.
- **Dead/Terminated**: The thread exits or stops executing due to some unusual happenings.

DIAGRAM:
https://dotnettrickscloud.blob.core.windows.net/article/java/3720240430154253.com-png-to-webp-converter (1)

11. Differentiate between the Thread class and Runnable interface for creating a thread.

- The thread can be created by extending the Thread class
- The thread can be created by implementing the Runnable interface

Thread Class	Runnable Interface
It has multiple methods like start() and run()	It has only abstract method run()
Each thread creates a unique object and gets associated with it	Multiple threads share the same objects
More memory required	Less memory required
Multiple Inheritance is not allowed in Java hence after a class extends the Thread class, it can not extend any other class	If a class implements the runnable interface, it can extend another class.

12. What is join() method?

One thread can call the join() method on another thread. As a result, the first thread that called the method suspends its work and waits for the second thread to finish execution.

The Thread class has three forms of join() method:

1. public final void join() throws InterruptedException
2. public final void join(long millis) throws InterruptedException
3. public final void join(long millis, int nanos) throws InterruptedException

13. Describe the purpose and working of the sleep() method.

sleep() method in Java blocks a thread for a particular time, which means it pauses the execution of a thread for a specific time. There are two overloaded forms of sleep() method in java.lang.Thread class.

1. public static void sleep(long millis) throws InterruptedException
2. public static void sleep(long millis, int nanos) throws InterruptedException

14. List out the differences between wait() and sleep() method.

wait()	**sleep()**
wait() is a method of Object class	sleep() is a method of Thread class
wait() releases lock during synchronization	sleep() method does not release the lock on the object during synchronization
wait() is not a static method.	sleep() is a static method
wait() has three overloaded methods: • wait() • wait(long timeout) • wait(long timeout, int nanos)	sleep() has two overloaded methods: • sleep(long millis)millis: milliseconds • sleep(long millis,int nanos) nanos: Nanoseconds

15.Can we call the run() method instead of start()?

- Yes, calling the run() method directly is valid, but it will not work as a thread instead it will work as a normal object.
- There is no context-switching between the threads.
- When we call the start() method, it internally calls the run() method. It creates a new stack for a thread while directly calling the run() will not create a new stack.

17. What are the two ways of implementing thread in Java?

1. Extending the Thread class

```
class MultithreadingDemo extends Thread {
        public void run()  {
          System.out.println("Welcome to Scholarhat.");
        }
        public static void main(String args[]) {
          MultithreadingDemo obj=new MultithreadingDemo();
            obj.start();
        }
}
```

3. Implementing Runnable interface in Java

```
class MultithreadingDemo implements Runnable {
            public void run(){
              System.out.println("Welcome to Scholarhat..");
            }
             public static void main(String args[]){
              MultithreadingDemo obj=new MultithreadingDemo();
              Threadtobj =new Thread(obj);      tobj.start();
            }
        }
```

18.What's the difference between notify() and notifyAll()?

1.notify()

- It sends a notification and wakes up only a single thread instead of multiple threads that are waiting on the object's monitor.
- The notify() method is defined in the Object class, which is Java's top-level class.
- It's used to wake up only one thread that's waiting for an object, and that thread then begins execution.
- The thread class notify() method is used to wake up a single thread.
  https://dotnettrickscloud.blob.core.windows.net/article/java/3720240430154520.com-png-to-webp-converter

2.notifyAll()

- It sends notifications wakes up all threads and allows them to compete for the object's monitor instead of a single thread.
- The notifyAll() wakes up all threads that are waiting on this object's monitor.
- A thread waits on an object's monitor by calling one of the wait methods.
- The awakened threads will not be able to proceed until the current thread relinquishes the lock on this object.

19. What is the default priority of a thread? Can we change it?

The default priority of a thread is the same as that of its parent. Yes, we can change the priority of a thread at any time using the setPriority() method.

```
public class ThreadPriority {
   public static void main(String[] args) {
      Thread t = Thread.currentThread();
      System.out.println(t.getPriority());
      t.setPriority(8);
      System.out.println(t.getPriority());
   }
}
```

t.setPriority(8) sets the current thread priority to 8. By default, the priority of the main thread is Thread.NORM_PRIORITY, which is generally 5. This value is printed to the console.

20. Is it possible to make constructors synchronized in Java?

No, the synchronized keyword cannot be used with constructors in Java. The synchronization mechanism in Java is designed to manage access to objects that already exist, ensuring that multiple threads do not interfere with each other when accessing a shared resource.

Java Multithreading Interview Questions for Intermediates

21. What is the fundamental difference between a process and a thread?

A process is part of the main program, while threads are subsets of processes. Processes have different address spaces in the memory, while threads share the same address space.

22. What is inter-thread communication?

Communication between synchronized threads is referred to as inter-thread communication. It is a core feature that helps to avoid thread-polling in Java. A particular thread can be paused through inter-thread communication to allow another thread to enter the block.

23. What are some functions used to perform inter-thread communication in Java?

This is one of the most common Java Multithreading interview questions asked in tech interviews. Some common functions used to perform inter-thread communication in Java are - notify(), wait(), and notifyAll().

24. What do you understand about the wait() function in Java?

The wait() function, specified by the object class, is used to pause the current thread and wait until another thread calls the notify() function. Note that the wait() method needs to be called from a synchronized block to avoid an exception from occurring.

25. What do you understand by context switching?

Context switching is a feature through which the current state of a thread is saved for it to be restored and executed later. Through context switching, multiple processes can share the same CPU.

26. What is the function of the join() method?

The join() method causes the current thread to stop running until the thread due congregates and completes its task. It is a commonly used function that facilitates the execution of multiple threads in an organized manner.

27. What is the function of the sleep() method?

This is yet another popular Java Multithreading Interview Question. The sleep() method is used to pause a particular thread from executing and prioritizes another thread that needs to be executed before the current thread executes.

28. What do you understand about Deadlock situations in Java Multithreading?

Deadlock is a situation where each thread is waiting for resources held by other waiting threads. Due to this situation, no threads are executed, causing a pause in program execution and breaking the code at runtime.

29. How do you detect deadlock situations in Java?

Deadlock situations can be detected by running the executable code on cmd and subsequently collecting the thread dump. If deadlock situations are present, the cmd will throw up a message.

30. How can deadlock situations be avoided in Java?

This is one of the most common Java Multithreading interview questions asked in technical interviews. Deadlock situations in Java can be avoided by:

- By way of avoiding nested thread locks and providing locks to only one thread at a time
- By using thread join - the function helps to wait for threads to execute before other threads are executed, thereby preventing multiple threads from waiting for resources used by other threads.

OOP'S CONCEPT

1. What do you understand by the term OOP?

OOPs is an acronym for Object Oriented Programming.

2. What is a class in OOP?

Class is a collection of objects. It is like a blueprint for an object.

3. What is an object in OOP?

An object is a real-world entity having a particular behavior and a state. It can be physical or logical. An object is an instance of a class and memory is allocated only when an object of the class is created.

5. What are the main features of OOPs?

1. **Inheritance**
2. **Encapsulation**
3. **Polymorphism**
4. **Data Abstraction**

6. How is a class different from a structure?

Class	Structure
Classes are of reference types	Structures are of value types
It is allocated to heap memory	It is allocated on stack memory
Allocation is cheaper in the large reference type	Allocation is cheaper in value type than a reference type
It has limitless features	It has limited features
A class is used in large programs	A structure is used in small programs

7. List out a few advantages of using OOPs.

- **Modularity:** OOP promotes modularity by encapsulating data and methods into objects, facilitating easy maintenance and updates.
- **Reusability:** Objects can be reused in different parts of the program or other projects.
- **Readability:** OOP enhances code readability with a clear structure, making it easier for developers to understand and collaborate.
- **Scalability:** OOP accommodates the growth of software projects, allowing the creation of large, complex systems with manageable components.
- **Flexibility:** Polymorphism and inheritance provide flexibility, allowing the creation of versatile and adaptable code.
- **Data abstraction:** OOP hides complex implementation details through data abstraction, exposing only essential features for user interaction.

- **Code maintenance:** OOP simplifies code maintenance by isolating changes to specific objects or classes, reducing the risk of unintended consequences in other parts of the code.

8. What is encapsulation?

Encapsulation helps to wrap up the functions and data together in a single unit. By privatizing the scope of the data members it can be achieved. This particular feature makes the program inaccessible to the outside world.

9. What do you understand about Access Specifiers/Access Modifiers? What are their types?

Access specifiers define how the members (data members and member functions) of a class can be accessed. They allow us to determine which class members are accessible to other classes and functions, and which are not.
There are three access specifiers in object-oriented programming:

1. **public**
2. **private**
3. **protected**

These modifiers are crucial in achieving Encapsulation as they control the visibility of internal details and contribute to the overall security and organization of the code.

Public Access Modifier: This employs a public keyword to create public members (data and functions). The public members are accessible from any part of the program i.e. from inside or outside the class.

Private Access Modifier: This employs a private keyword to create private members (data and functions). The private members can only be accessed from within the class.

Protected Access Modifier: This employs the protected keyword to create protected members (data and function). The protected members can be accessed within the class and from the derived class.

11. Compare classes and objects in OOP.

Class	Object
Class is a logical entity	An object is a physical entity
Class is a template from which an object can be created	An object is an instance of the class

Class is a prototype that has the state and behavior of similar objects	Objects are entities that exist in real life such as a mobile, mouse, or a bank account
Class is declared with a class keyword	An object is created via a new keyword

12. What is meant by constructor?

A constructor is a special member function invoked automatically when an object of a class is created. It is generally used to initialize the data members of the new object. They are also used to run a default code when an object is created. The constructor has the same name as the class and does not return any value.

14. How is a constructor different from a method?

Constructors	**Methods**
The constructor's name should match that of the Class	Methods should not have the same name as the Class name
They are used to create, initialize, and allocate memory to the object.	Methods are used to execute certain statements written inside them.
Constructors are implicitly invoked by the system whenever objects are created.	Methods are invoked when it is called.
They are invoked using a new keyword while creating an instance of the class (object)	Methods are invoked during program execution
A constructor does not have a return type	The method has a return type
A subclass cannot inherit a constructor from its superclass.	Sub-classes are capable of inheriting methods.

15. What is inheritance?

Inheritance in object-oriented programming (OOP) allows one class to inherit the attributes and functions of another.

17. State the advantages of inheritance.

- **Code Reusability:** Inheritance allows you to create new classes that are based on existing classes, so you can reuse existing code and avoid rewriting the same functionality. This saves a lot of time and effort when you're creating new programs.
- **Organization:** Inheritance helps you organize your code in a logical and structured way. You can create a hierarchy of related classes that makes it easy to understand how your code works and how different parts of your program are related.
- **Polymorphism:** Inheritance facilitates polymorphism, which allows you to treat objects of derived classes as objects of their base class, enabling you to write code that operates on base-class pointers or references but can work with objects of derived classes at runtime.
- **Overriding:** Inheritance allows you to override methods in a derived class, which means you can change the way a method works without changing the original code.
- **Abstraction:** Inheritance allows you to create abstract classes that define the basic behavior of a group of related classes while leaving the implementation details to the derived classes.

18. Describe Polymorphism.

Polymorphism is made up of two words, poly means more than one and morphs means forms. So, polymorphism means the ability to take more than one form. This property makes the same entities such as functions, and operators perform differently in different scenarios. You can perform the same task in different ways.

20. How much memory does a class occupy?

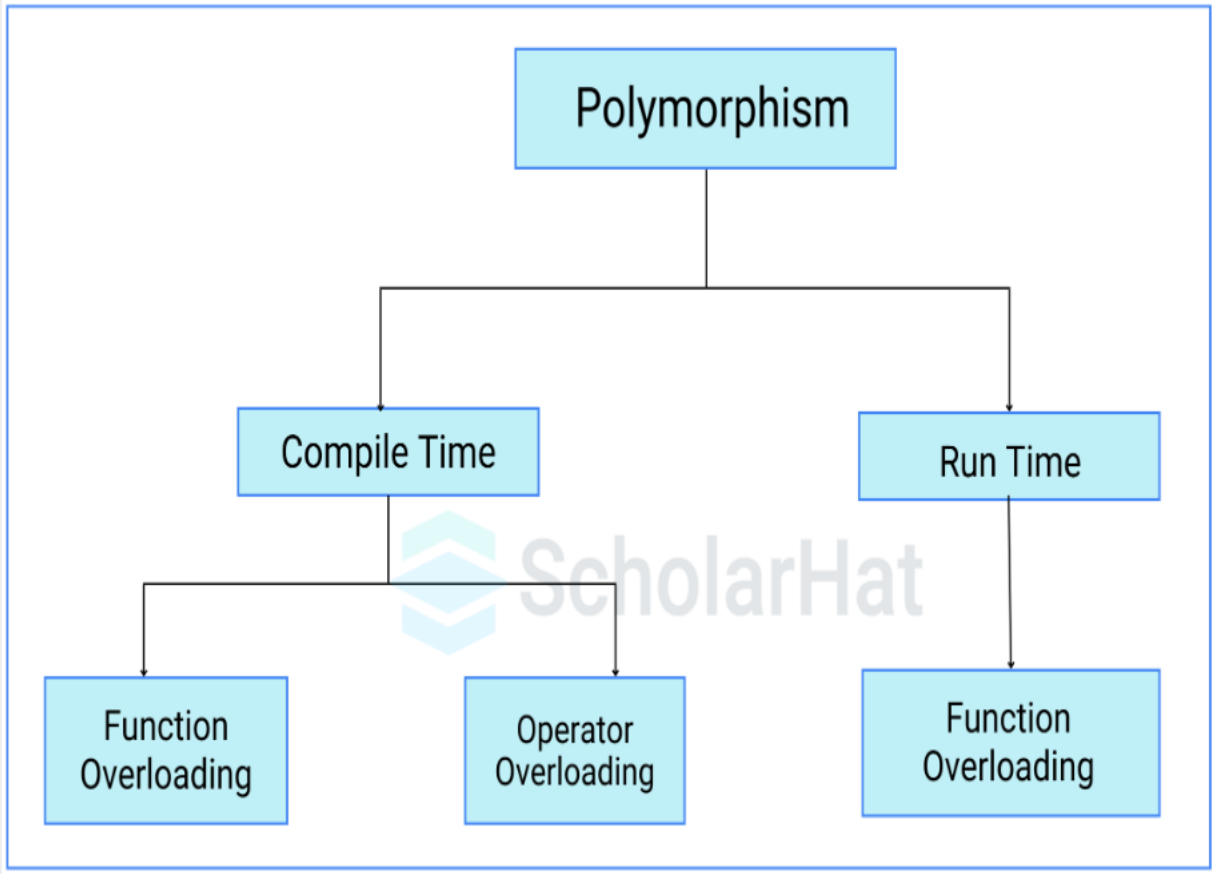
Classes do not use memory. They merely serve as a template from which items are made. Now, objects initialize the class members and methods when they are created, using memory in the process.

Intermediate OOPs Interview Questions & Answers

21. Is it always necessary to create objects from class?

No, it is not always necessary to create objects from a class. A class can be used to define a blueprint for creating things, but it is not required to create an object from a class to use its methods or access its properties.

22. What are the different types of polymorphism?



There are two types of polymorphism. These are:

1. **Compile time Polymorphism:** In this, the compiler at the compilation stage knows which functions to execute during the program execution. It matches the overloaded functions or operators with the number and type of parameters at the compile time.
   The compiler performs compile-time polymorphism in two ways:
   1. **Function Overloading:** If two or more functions have the same name but different numbers and types of parameters, they are known as overloaded functions.
   2. **Operator Overloading:** It is function overloading, where different operator functions have the same symbol but different operands. And, depending on the operands, different operator functions are executed. The operands here are the user-defined data types like objects or structures and not the basic data types.
2. **Runtime Polymorphism:** In this, the compiler at the compilation stage does not know the functions to execute during the program execution. The function call is not resolved during compilation, but it is resolved in the run time based on the object type. This is also known as dynamic binding or late binding. This is achieved through function overriding.
   1. When you redefine any base class method with the same signature i.e. same return type, number, and type of parameters in the derived class, it is known as function overriding.

23. Differentiate between static and dynamic binding.

Parameters	Static Binding	Dynamic Binding
Time of binding	Happens at compile time	Happens at run time
Actual object	The actual object is not used for binding	The actual object is used for binding
Also known as	early binding because binding happens during compilation	late binding because binding happens at run time
Example	Method overloading	Method overriding
Methods of binding	Private, static, and final methods show static binding. Because they cannot be overridden.	Other than private, static and final methods show dynamic binding. Because they can be overridden.

25. What is Abstraction?

Abstraction means providing only the essential details to the outside world and hiding the internal details, i.e., hiding the background details or implementation. Abstraction is a programming technique that depends on the separation of the interface and implementation details of the program.

26. What is an interface?

An interface refers to a special type of class, which contains methods, but not their definition. Only the declaration of methods is allowed inside an interface. To use an interface, you cannot create objects. Instead, you need to implement that interface and define the methods for their implementation.

27. How is an Abstract class different from an Interface?

Abstract Class	Interface
When an abstract class is inherited, however, the subclass is not required to supply the definition of the abstract method until and unless the subclass uses it.	When an interface is implemented, the subclass is required to specify all of the interface's methods as well as their implementation.
An abstract class can have both	An interface can only have abstract

abstract and non-abstract methods.	methods
An abstract class can have final, non-final, static, and non-static variables	The interface has only static and final variables.
Abstract class doesn't support multiple inheritance	An interface supports multiple inheritance.

28. Describe the use of the super keyword.

super keyword in Java is used to identify or refer to parent (base) class.

- We can use super to access the superclass constructors and call methods of the superclass.
- When method names are the same in the superclass and subclass, the super keyword refers to the superclass.
- To access the same name data members of the parent class when they are present in parent and child class.
- Super can be used to make an explicit call to no-arg and parameterized constructors of the parent class.
- Parent class method access can be done using super when the child class has a method overridden.

31. Distinguish between Method Overloading and Method Overriding.

Parameters	Method Overloading	Method Overriding
Definition	Multiple methods in the same class	Method in a subclass with the same name as in the superclass
Inheritance	Not necessary, can be in the same class	Must have the same method name and parameter list as the superclass
Signature	Must have different parameter lists	Must have the same method name and parameter list as the superclass
Return Type	Can be different	Must be the same as the method in the superclass
Usage	Provides multiple methods with the same name but different parameters	Allows a subclass to provide a specific implementation of a method defined in the superclass

32. Is Operator overloading supported in Java?

No, operator overloading is not supported in Java.

33. How will you differentiate a class from an interface?

Class	Interface
A template for creating objects, defining their state (attributes) and behavior (methods)	A collection of abstract methods (and sometimes constants) that a class can implement
Contains concrete implementations of methods and properties	Does not contain any method implementation. Only declares method signatures
A class can extend another class, inheriting its properties and methods	A class implements an interface, agreeing to perform the specific behaviors outlined in the interface
Typically, a class cannot extend more than one class (depends on the language, but it is true for languages like Java)	A class can implement multiple interfaces, allowing for a form of multiple inheritances
To encapsulate data and provide functionality	To define a contract that implementing classes must follow, ensuring a certain level of functionality

35. Is it possible for the static method to use nonstatic members?

In Java, a static method cannot directly access or use non-static (instance) members of a class. This is because static methods are associated with the class itself, rather than with instances of the class. Non-static members, on the other hand, belong to individual instances of the class and require an object reference to be accessed.

36. What is meant by Garbage Collection in OOPs?

Object-oriented programming revolves around entities like objects. Each object consumes memory and there can be multiple objects of a class. So if these objects and their memories are not handled properly, then it might lead to certain memory-related errors and the system might fail.
Garbage Collection (GC) is an automatic memory management process in programming languages. It involves identifying and reclaiming memory occupied by objects no longer in use by the program. Instead of requiring manual memory deallocation, which can lead to

memory leaks and bugs, GC tracks and frees up memory occupied by objects that are no longer reachable or referenced in the program.

37. Why new keyword is used in Java?

When we create an instance of the class, i.e. objects, we use the Java keyword new. It allocates memory in the heap area where JVM reserves space for an object. Internally, it invokes the default constructor as well.

40. What are errors and what are exceptions? Differentiate between them.

Exceptions are runtime errors. Many a time, your program looks error-free during compilation. But, at the time of execution, some unexpected errors or events or objects come to the surface and the program prints an error message and stops executing.
Errors represent serious system or hardware failures, typically leading to program termination, and are usually beyond the program's control.

Differences between Errors and Exceptions

Parameters	Errors	Exceptions
Typical Use	Represents serious system failures	Used for manageable exceptional conditions in programs
Examples	OutOfMemoryError, StackOverflowError	NullPointerException, ArrayIndexOutOfBoundsException
Handling	Often cannot be recovered, leads to program termination	Can be caught and handled to prevent program termination
Cause	Typically caused by the system or hardware	Caused by program conditions that can be handled

41. How does Java implement multiple inheritance?

Multiple inheritance is not supported by Java using classes. To achieve multiple inheritance in Java, we must use the interface. Interfaces ensure that classes follow predetermined behaviors by defining method contracts without implementing them.

42. Explain the difference between abstract class and method.

Abstract Class	Abstract Method
An object cannot be created from the	The abstract method has a signature

abstract class	but does not have a body
Subclass created or inherit abstract class to access members of abstract class	It is compulsory to override abstract methods of the superclass in their sub-class
Abstract classes can contain abstract methods or non-abstract methods	Class containing abstract method should be made the abstract class

43. Distinguish encapsulation from abstraction.

Encapsulation	**Abstraction**
Encapsulation is the process or method of containing the information in a single unit and providing this single unit to the user.	Abstraction is the process or method of gaining information
Main feature: Data hiding. It is a common practice to add data hiding in any real-world product to protect it from the external world. In OOPs, this is done through specific access modifiers	**Main feature:** reduce complexity, promote maintainability, and also provide a clear separation between the interface and its concrete implementation
problems are solved at the implementation level.	problems are solved at the design or interface level
It is a method to hide the data in a single entity or unit along with a method to protect information from outside	It is the method of hiding the unwanted information
encapsulation can be implemented using access modifiers i.e. private, protected, and public	We can implement abstraction using abstract classes and interfaces
implementation complexities are hidden using abstract classes and interfaces	the data is hidden using methods of getters and setters
the objects that result in encapsulation need not be abstracted	The objects that help to perform abstraction are encapsulated
Encapsulation hides data and the user cannot access the same directly	Abstraction provides access to specific parts of data

Encapsulation focus is on “How” it should be done	Abstraction focus is on “what” should be done

44. What is the use of the finalize method?

The finalize method helps to perform cleanup operations on the resources that are not currently used. The finalize method is protected, and it is accessible only through this class or by a derived class.

45. Can a Java application be run without implementing the OOPs concept?

No. Java applications are based on Object-oriented programming models or OOPs concept, and hence they cannot be implemented without it.

46. What are the differentiating points between extends and implements?

Extends	**Implements**
A class can extend another class (a child extending parent by inheriting his characteristics). An interface can also inherit (using the keyword extends) another interface	A class can implement an interface
Sub-class extending superclass may not override all of the superclass methods	Class implementing interface has to implement all the methods of the interface
A class can only extend a single superclass	A class can implement any number of interfaces
Interface can extend more than one interface	An interface cannot implement any other interface
Syntax: class Child extends class Parent	Syntax: class Hybrid implements Rose

47. Is Java a pure Object Oriented language?

Java is not an entirely pure object-oriented programming language. The following are the reasons:

- Java supports and uses primitive data types such as int, float, double, char, etc.
- Primitive data types are stored as variables or on the stack instead of the heap.
- In Java, static methods can access static variables without using an object, contrary to object-oriented concepts.

48. Can you create an instance of an abstract class?

No, an instance of the Abstract class cannot be created.
However, you can create an instance of a concrete subclass that extends the abstract class. This concrete subclass must provide implementations for all the abstract methods defined in the abstract class. Then you can create an instance of this subclass.

- AbstractClass is an abstract class with an abstract method abstractMethod().
- ConcreteClass is a concrete subclass of AbstractClass that provides an implementation for the abstractMethod().
- In the main method, we create an instance of ConcreteClass, which is allowed because it provides implementations for all abstract methods defined in AbstractClass.

49. What are manipulators?

Manipulators are the functions that can be used in conjunction with the insertion (<<) and extraction (>>) operators on an object. Examples are endl and setw.

50. Define the term 'composition' in OOP.

Composition in Object-Oriented Programming refers to a design principle where a class includes instances of other classes to add functionality or behavior. This "has-a" relationship allows for flexible and reusable code.