1. **What is the lambda expression of Java 8?**

**Ans:** Lambda expressions were added in Java 8. [They are used to provide the implementation of functional interfaces1](https://www.geeksforgeeks.org/lambda-expressions-java-8/). A lambda expression is a short block of code that takes in parameters and returns a value. [It is similar to methods but does not need a name and can be implemented right in the body of a method2](https://www.w3schools.com/java/java_lambda.asp).

Here is an example of a lambda expression in Java:

interface FuncInterface {

void abstractFun(int x);

default void normalFun() {

System.out.println("Hello");

}

}

class Test {

public static void main(String args[]) {

FuncInterface fobj = (int x)->System.out.println(2\*x);

fobj.abstractFun(5);

}

}

Output:

10

1. **Can you pass lambda expressions to a method? When?**

**Ans:** Yes, you can pass lambda expressions as parameters to methods in Java 8. [To use lambda expressions, it must be a functional interface](about:blank)[1](https://stackoverflow.com/questions/32255681/how-do-i-pass-a-method-as-a-parameter-in-java-8). A functional interface is an interface that contains only one abstract method. [You can use the @FunctionalInterface annotation to indicate that an interface is a functional interface](about:blank)[2](https://www.programmingcube.com/write-a-java-program-to-pass-lambda-expression-as-a-method-argument/).

Here is an example of how to pass a lambda expression as a method parameter in Java:

interface MethodParameter {

String apply(Stuff input);

}

MethodParameter method = (Stuff stuff) -> { return stuff.toString(); };

System.out.println(method.apply(stuff));

1. **What is the functional interface in Java8?**

**Ans:** [A functional interface is an interface that contains only one abstract method](about:blank)[1](https://www.javatpoint.com/java-8-functional-interfaces). [It can have any number of default, static methods but can contain only one abstract method1](https://www.javatpoint.com/java-8-functional-interfaces). [It is also known as Single Abstract Method Interfaces or SAM Interfaces1](https://www.javatpoint.com/java-8-functional-interfaces).

[Java 8 provides predefined functional interfaces to deal with functional programming by using lambda and method references1](https://www.javatpoint.com/java-8-functional-interfaces). Following is the list of functional interfaces which are placed in java.util.function package:

* BiConsumer<T,U>
* Consumer<T>
* Function<T,R>
* Predicate<T>

Here is an example of a functional interface in Java 8:

@FunctionalInterface

interface MyInterface {

void myMethod();

}

In this example, MyInterface is a functional interface because it has only one abstract method. The @FunctionalInterface annotation is used to indicate that an interface is a functional interface.

1. **Why do we use lambda expression in Java?**

**Ans:** Lambda expressions are used to provide the implementation of functional interfaces. They are used to support the functional programming approach, lambda expression, and method reference as well.

Lambda expressions are used in Java for the following reasons:

1. To provide the implementation of functional interfaces.
2. To write more concise and readable code.
3. To enable functional programming in Java.
4. To use method references.
5. **Is it mandatory for a lambda expression to have parameters?**

**Ans:** No, it is not mandatory for a lambda expression to have parameters. A lambda expression can have zero or more parameters. The syntax for a lambda expression with no parameters is as follows:

() -> {

// body of lambda expression

}

Copy

Here is an example of a lambda expression with no parameters:

() -> System.out.println("Hello World");

1. **Explain different types of Errors in Java.**

**Ans:** There are three types of errors in Java:

1. Compile-time errors: These errors are detected by the Java compiler and prevent the code from running because of incorrect syntax such as a missing semicolon at the end of a statement or a missing bracket. These errors are sometimes also referred to as syntax errors.
2. Runtime errors: These errors occur during the execution of the program. Sometimes these are discovered when the user enters invalid data or data that is not relevant. Runtime errors occur when a program does not contain any syntax errors but asks the computer to do something that it is unable to reliably do. During compilation, the compiler has no technique to detect these kinds of errors. It is the JVM (Java Virtual Machine) that detects it while the program is running.
3. Logic errors: These errors occur when a program runs without any syntax or runtime errors but produces incorrect results. These types of errors are also known as semantic errors.

Here are some examples of each type of error:

1. Compile-time error: Missing semicolon at the end of a statement

public class Main {

public static void main(String[] args) {

System.out.println("Hello World")

}

}

1. Runtime error: Dividing by zero

public class Main {

public static void main(String[] args) {

int x = 5;

int y = 0;

int z = x / y;

}

}

1. Logic error: Incorrect calculation

public class Main {

public static void main(String[] args) {

int x = 5;

int y = 10;

int z = x + y;

System.out.println("The sum of x and y is " + z);

}

}

1. **What is an exception in java?**

**Ans:** In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime. When an exception occurs within a method, it creates an object. This object is called the exception object. [It contains information about the exception, such as the name and description of the exception and the state of the program when the exception occurred1](https://www.geeksforgeeks.org/exceptions-in-java/).

Here is an example of how to use exceptions in Java:

public class Main {

public static void main(String[] args) {

try {

int[] arr = new int[5];

arr[10] = 50;

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("An error occurred: " + e.getMessage());

}

}

}

In this example, we are trying to access an element in an array that does not exist. This will cause an ArrayIndexOutOfBoundsException to be thrown. We catch this exception using a try-catch block and print out a message indicating that an error occurred.

1. **How can you handle exceptions in java? Explain with an example.**

**Ans:** In Java, exceptions are handled using a try-catch block. The try block contains the code that might generate an exception. If an exception occurs, it is caught by the catch block. Here is an example of how to use a try-catch block in Java:

public class Main {

public static void main(String[] args) {

try {

int[] arr = new int[5];

arr[10] = 50;

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("An error occurred: " + e.getMessage());

}

}

}

In this example, we are trying to access an element in an array that does not exist. This will cause an ArrayIndexOutOfBoundsException to be thrown. We catch this exception using a try-catch block and print out a message indicating that an error occurred.

There are other ways to handle exceptions in Java as well, such as using the finally block or the throw and throws keywords. You can learn more about these methods by following the links below:

* [Java Exception Handling (With Examples)](https://www.programiz.com/java-programming/exception-handling)
* [Java Exceptions - Try…Catch](https://www.w3schools.com/java/java_try_catch.asp)
* [Exception Handling in Java | Java Exceptions](https://www.javatpoint.com/exception-handling-in-java)

1. **Why do we need exception handling in java?**

**Ans:** Exception handling is an important part of Java programming because it helps to maintain the normal flow of the program’s instructions. When an exception occurs, it disrupts the normal flow of the program and can cause it to terminate abnormally. Exception handling allows us to catch these exceptions and handle them gracefully so that the program can continue running normally.

1. **What is the different between exception and error in Java?**

**Ans:** In Java, Exception is an event that occurs during the execution of the program and interrupts the normal flow of program instructions. These are the errors that occur at compile time and run time. It occurs in the code written by the developers. It can be recovered by using the try-catch block and throws keyword. There are two types of exceptions i.e. checked and unchecked. Errors are problems that mainly occur due to the lack of system resources. It cannot be caught or handled. [It indicates a serious problem1](https://www.javatpoint.com/exception-vs-error-in-java)[2](https://rollbar.com/blog/java-exceptions-hierarchy-explained/).

Here’s an example of Exception in Java:

import java.util.Scanner;

public class ExcptionExample {

public static void main (String args []) {

Scanner sc = new Scanner (System.in);

System.out.print ("Enter a number: ");

int number = sc.nextInt ();

System.out.println ("You have entered: "+number);

}

}

Let’s run the above program and enter a float value deliberately to generate an exception. It shows the InputMismatchExaception. Because the program accepts an integer value. [We observe that the next statement is skipped and the program is terminated](about:blank)[1](https://www.javatpoint.com/exception-vs-error-in-java).

Here’s an example of Error in Java:

public class ErrorExample {

public static void main (String args []) {

//method calling

recursiveDemo (10);

}

public static void recursiveDemo (int i) {

while(i!=0) {

//increments the variable i by 1

i=i+1;

//recursive called method

recursiveDemo (i);

}

}

}

[We observe that on running the program, we get the StackOverflowError, not an exception](about:blank)[1](https://www.javatpoint.com/exception-vs-error-in-java).

1. **Name the different types of exception in Java?**

Ans: In Java, exceptions can be categorized into two ways: Built-in Exceptions and User-Defined Exceptions. Built-in Exceptions are exceptions that are already available in Java libraries. These exceptions are able to define the error situation so that we can understand the reason for getting this error. It can be categorized into two broad categories, i.e., checked exceptions and unchecked exception. Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler. The compiler ensures whether the programmer handles the exception or not. [The programmer should have to handle the exception; otherwise, the system has shown a compilation error](https://www.javatpoint.com/types-of-exception-in-java).

Here are some examples of built-in exceptions in Java:

* ArithmeticException
* ArrayIndexOutOfBoundsException
* ClassNotFoundException
* FileNotFoundException
* IOException
* InterruptedException
* NoSuchFieldException
* NoSuchMethodException
* NullPointerException
* NumberFormatException

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

**Ans:** No, we cannot just use try instead of finally and catch blocks. The try block must be followed by either catch or finally block or both. The catch block is used to handle the exception that is thrown by the try block. The finally block is used to execute the important code of the program. [It is executed whether an exception is handled or not1](https://stackoverflow.com/questions/4559661/java-try-catch-finally-blocks-without-catch).

Here’s an example of try-catch-finally in Java:

import java.util.Scanner;

public class TryCatchFinallyExample {

public static void main (String args []) {

Scanner sc = new Scanner (System.in);

System.out.print ("Enter a number: ");

try {

int number = sc.nextInt ();

System.out.println ("You have entered: "+number);

} catch (Exception e) {

System.out.println ("Invalid input");

} finally {

System.out.println ("Thank you for using this program.");

}

}

}

In the above code, we are trying to read an integer value from the user. If the user enters a non-integer value, then it will throw an exception. The catch block will handle this exception and print “Invalid input”. [Finally, it will print "Thank you for using this program."1](https://stackoverflow.com/questions/4559661/java-try-catch-finally-blocks-without-catch).

1. **What is an interface in Java?**

**Ans:** An interface in Java is a fully abstract class. It includes a group of abstract methods (methods without a body). We use the interface keyword to create an interface in Java. For example, interface Language { public void getType(); public void getVersion(); } Here, Language is an interface. It includes abstract methods: getType () and getVersion (). [Like abstract classes, we cannot create objects of interfaces1](https://www.programiz.com/java-programming/interfaces)[2](https://www.guru99.com/java-interface.html).

[To use an interface in your class, append the keyword “implements” after your class name followed by the interface name](about:blank).

1. **Which modifiers are allowed for methods in an interfaces? Explain with examples.**

**Ans:** In an interface, all methods are public and abstract by default. We cannot use any other modifiers with the methods in an interface.

For example, the following code shows an interface named Language with a method named getType() that has no body:

interface Language {

public void getType();

}

1. **What is the use of interface in java? Or, Why do we use an interface in Java?**

**Ans:** An interface in Java is an abstract type used to specify the behavior of a class. It is a blueprint of a behavior. A Java interface contains static constants and abstract methods. The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not the method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body. Java Interface also represents the IS-A relationship. When we decide a type of entity by its behavior and not via attribute we should define it as an interface. Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body). Interfaces specify what a class must do and not how. [It is the blueprint of the behavior](about:blank) [1](https://www.geeksforgeeks.org/interfaces-in-java/)[2](https://www.baeldung.com/java-interfaces).

For example, let’s say you have two classes: Car and Bike. Both classes have different implementations but they share some common functionality like they both have wheels and they both can move forward. In this case, you can create an interface called Vehicle which will define the common functionality like move() method and wheels variable. [Now both classes can implement this interface and use its functionality 1](https://www.geeksforgeeks.org/interfaces-in-java/).

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

**Ans:** An abstract class is a class that cannot be instantiated and can contain both abstract and non-abstract methods. An interface, on the other hand, is a contract that specifies a set of methods that a class must implement. In an abstract class, some methods can have an implementation while others can be abstract. In an interface, all methods are abstract by default. [Since Java 8, interfaces can have default and static methods as well](about:blank) [1](https://www.javatpoint.com/difference-between-abstract-class-and-interface)[2](https://www.geeksforgeeks.org/difference-between-abstract-class-and-interface-in-java/).

Here are some differences between abstract classes and interfaces:

* Abstract classes can have final, non-final, static and non-static variables. Interfaces have only static and final variables.
* Abstract classes can provide the implementation of an interface. Interfaces cannot provide the implementation of an abstract class.
* An abstract class can extend another Java class and implement multiple Java interfaces. An interface can extend another Java interface only.
* A Java abstract class can have class members like private, protected, etc. [Members of a Java interface are public by default](about:blank) [1](https://www.javatpoint.com/difference-between-abstract-class-and-interface)[2](https://www.geeksforgeeks.org/difference-between-abstract-class-and-interface-in-java/).

Here’s an example:

abstract class Vehicle {

int wheels;

void move() {

System.out.println("Vehicle is moving");

}

abstract void start();

}

interface Car {

void start();

void stop();

}

class BMW extends Vehicle implements Car {

public void start() {

System.out.println("BMW is starting");

}

public void stop() {

System.out.println("BMW is stopping");

}

}

In this example, Vehicle is an abstract class that has a non-abstract method move() and an abstract method start(). Car is an interface that has two methods start() and stop(). The BMW class extends the Vehicle class and implements the Car interface. [It provides the implementation for both the start() and stop() methods](about:blank) [1](https://www.javatpoint.com/difference-between-abstract-class-and-interface)