Java Exception Handling

Last Updated : 14 Jan, 2025

Exception handling in Java allows developers to manage runtime errors effectively by using mechanisms like **try-catch block**, **finally block**, **throwing Exceptions**, **Custom Exception handling,** etc.

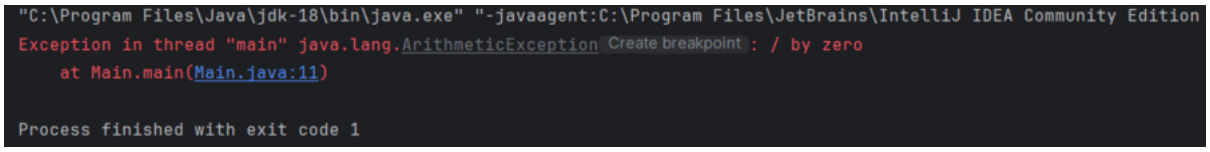
An **Exception** is an unwanted or unexpected event that occurs during the execution of a program (i.e., at **runtime**) and disrupts the normal flow of the program's instructions. It occurs when something unexpected things happen, like accessing an invalid index, dividing by zero, or trying to open a file that does not exist.

Exception in Java is an error condition that occurs when something wrong happens during the program execution.

Example: Showing an Arithmetic Exception or you can say divide by zero exception.

```java
import java.io.*;

class Geeks {
    public static void main(String[] args)
    {
        int n = 10;
        int m = 0;

        int ans = n / m;

        System.out.println("Answer: " + ans);
    }
}
```

Output:

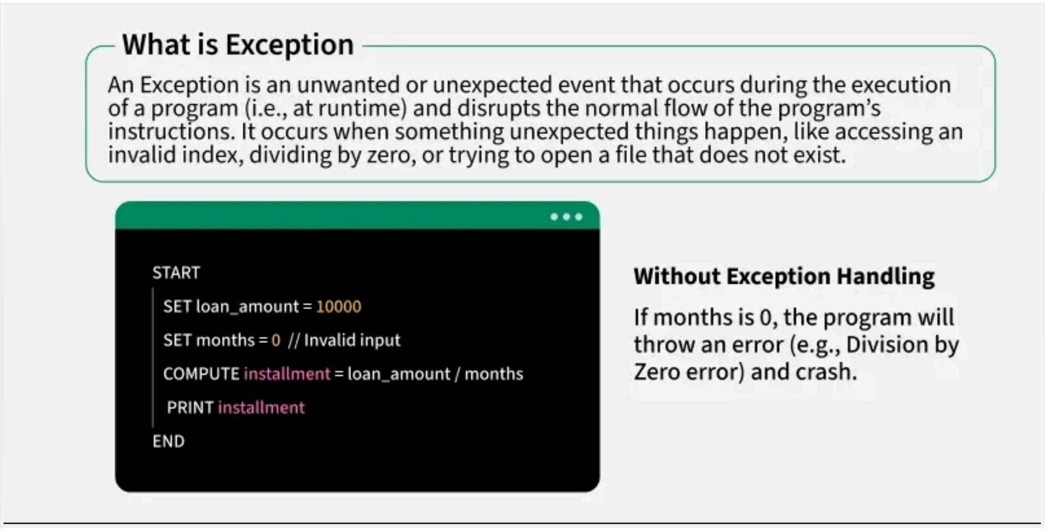


```
"C:\Program Files\Java\jdk-18\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition
Exception in thread "main" java.lang.ArithmeticException Create breakpoint : / by zero
    at Main.main(Main.java:11)

Process finished with exit code 1
```

> ***Note:*** *When an exception occurs and is not handled, the program terminates abruptly and the code after it, will never execute.*

Exception Handling in Java

Exception handling in Java is an effective mechanism for managing runtime errors to ensure the application's regular flow is maintained. Some Common examples of exceptions include ClassNotFoundException, IOException, SQLException, RemoteException, etc. By handling these exceptions, Java enables developers to create robust and fault-tolerant applications.



What is Exception

An Exception is an unwanted or unexpected event that occurs during the execution of a program (i.e., at runtime) and disrupts the normal flow of the program's instructions. It occurs when something unexpected things happen, like accessing an invalid index, dividing by zero, or trying to open a file that does not exist.

```
START
  SET loan_amount = 10000
  SET months = 0 // Invalid input
  COMPUTE installment = loan_amount / months
  PRINT installment
END
```

Without Exception Handling

If months is 0, the program will throw an error (e.g., Division by Zero error) and crash.

1 / 3

Example: The below Java program modifies the previous example to handle an **ArithmeticException** using try-catch, and finally blocks and keep the program running.

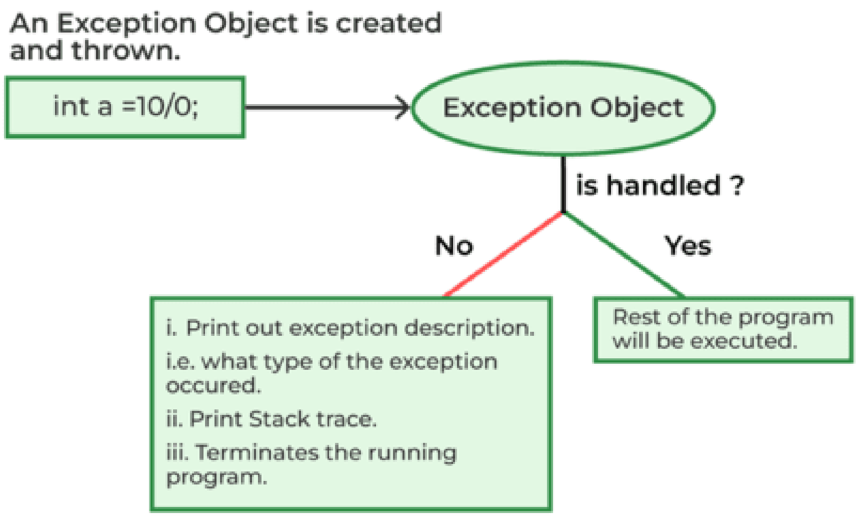
```java
// Java program to demonstrates handling
// the exception using try-catch block
import java.io.*;

class Geeks {
    public static void main(String[] args)
    {
        int n = 10;
        int m = 0;
        try {

            // Code that may throw an exception
            int ans = n / m;
            System.out.println("Answer: " + ans);
        }
        catch (ArithmeticException e) {

            // Handling the exception
            System.out.println(
                "Error: Division by zero is not allowed!");
        }
        finally {
            System.out.println(
                "Program continues after handling the exception.");
        }
    }
}
```

Output

```
Error: Division by zero is not allowed!
Program continues after handling the exception.
```

> ***Note:*** *With the help of exception handling we can detect and handle the exceptions gracefully so that the normal flow of the program can be maintained.*

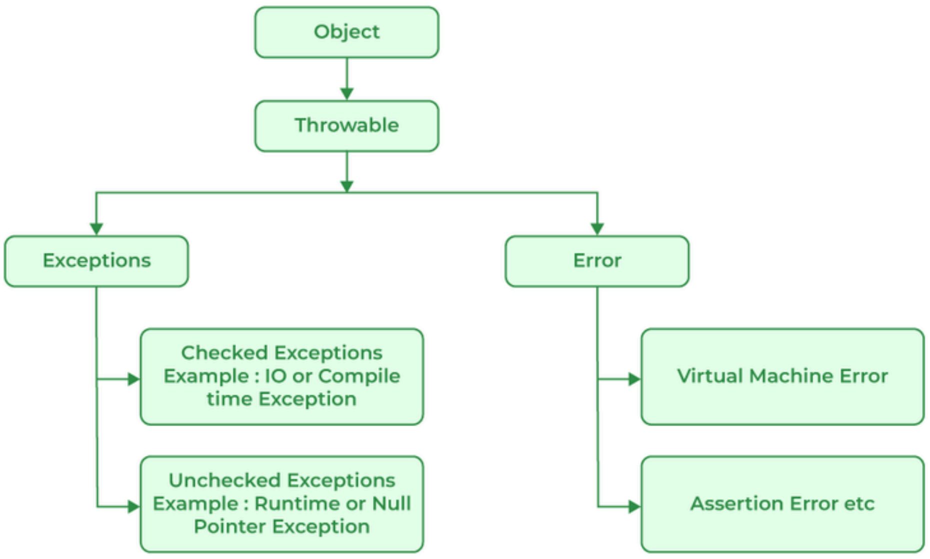
The summary is depicted via visual aid below as follows:


An Exception Object is created and thrown.
int a =10/0;
Exception Object
is handled ?
No
Yes
i. Print out exception description.
i.e. what type of the exception occured.
ii. Print Stack trace.
iii. Terminates the running program.
Rest of the program will be executed.

Java Exception Hierarchy

All exception and error types are subclasses of the class **Throwable**, which is the base class of the hierarchy. One branch is headed by **Exception**. This class is used for exceptional conditions that user programs should catch. NullPointerException is an example of such an exception. Another branch, **Error** is used by the Java run-time system(JVM) to indicate errors having to do with the run-time environment itself(JRE). StackOverflowError is an example of such an error.

The below figure demonstrates the exception hierarchy in Java:


Object
Throwable
Exceptions
Error
Checked Exceptions Example : IO or Compile time Exception
Unchecked Exceptions Example : Runtime or Null Pointer Exception
Virtual Machine Error
Assertion Error etc

Major Reasons Why an Exception Occurs

Exceptions can occur due several reasons, such as:

- Invalid user input
- Device failure
- Loss of network connection
- Physical limitations (out-of-disk memory)
- Code errors
- Out of bound
- Null reference
- Type mismatch
- Opening an unavailable file
- Database errors
- Arithmetic errors

Errors represent irrecoverable conditions such as Java virtual machine (JVM) running out of memory, memory leaks, stack overflow errors, library incompatibility, infinite recursion, etc. Errors are usually beyond the control of the programmer, and we should not try to handle errors.

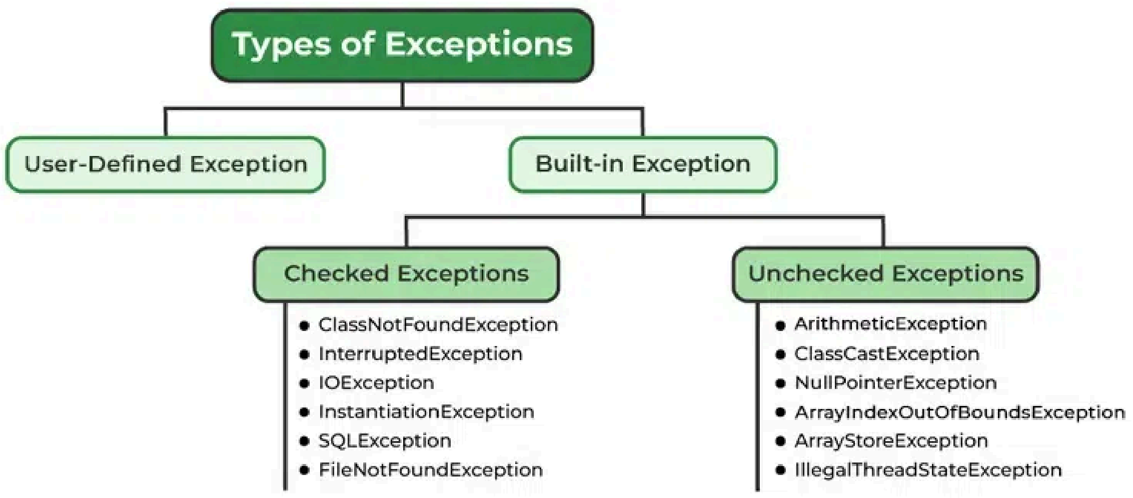
Difference Between Exception and Error

Aspect	Error	Exception
Definition	An Error indicates a serious problem that a reasonable application should not try to catch.	Exception indicates conditions that a reasonable application might try to catch
Cause	Caused by issues with the JVM or hardware.	Caused by conditions in the program such as invalid input or logic errors.
Examples	OutOfMemoryError StackOverFlowError	IOException NullPointerException

To know more differences about Exception and Errors, refer to this article: Exception vs Errors in Java.

Types of Java Exceptions

Java defines several types of exceptions that relate to its various class libraries. Java also allows users to define their own exceptions.



Exceptions can be categorized in two ways:

1. **Built-in Exceptions**
   - Checked Exception
   - Unchecked Exception

2. **User-Defined Exceptions**

1. Built-in Exception

Build-in Exception are pre-defined exception classes provided by Java to handle common errors during program execution.

1.1 Checked Exceptions

Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler. Examples of Checked Exception are listed below:

1. **ClassNotFoundException:** Throws when the program tries to load a class at runtime but the class is not found because its not present in the correct location or it is missing from the project.
2. **InterruptedException:** Thrown when a thread is paused and another thread interrupts it.
3. **IOException:** Throws when input/output operation fails
4. **InstantiationException:** Thrown when the program tries to create an object of a class but fails because the class is abstract, an interface, or has no default constructor.

5. **SQLException:** Throws when there's an error with the database.
6. **FileNotFoundException:** Thrown when the program tries to open a file that doesn't exist

1.2 Unchecked Exceptions

The unchecked exceptions are just opposite to the checked exceptions. The compiler will not check these exceptions at compile time. In simple words, if a program throws an unchecked exception, and even if we didn't handle or declare it, the program would not give a compilation error. Examples of Unchecked Exception are listed below:

1. **ArithmeticException:** It is thrown when there's an illegal math operation.
2. **ClassCastException:** It is thrown when you try to cast an object to a class it does not belongs to.
3. **NullPointerException:** It is thrown when you try to use a null object (e.g. accessing its methods or fields)
4. **ArrayIndexOutOfBoundsException:** It occurs when we try to access an array element with an invalid index.
5. **ArrayStoreException:** It **h**appens when you store an object of the wrong type in an array.
6. **IllegalThreadStateException:** It is thrown when a thread operation is not allowed in its current state

> ***Note:*** *For checked vs unchecked exception, see Checked vs Unchecked Exceptions*

2. User-Defined Exception

Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, users can also create exceptions, which are called "user-defined Exceptions".

Methods to Print the Exception Information

Method	Description
printStackTrace()	Prints the full stack trace of the exception, including the name, message, and location of the error.
toString()	Prints exception information in the format of the Name of the exception.
getMessage()	Prints the description of the exception.

Try-Catch Block

A try-catch block in Java is a mechanism to handle exception. The try block contains code that might thrown an exception and the catch block is used to handles the exceptions if it occurs.

```
try {

    // Code that may throw an exception

} catch (ExceptionType e) {

    // Code to handle the exception


}
```

finally Block

The finally Block is used to execute important code regardless of weather an exception occurs or not.

Note: finally block is always executes after the try-catch block. It is also used for resource cleanup.

```
try {

    // Code that may throw an exception

} catch (ExceptionType e) {

    // Code to handle the exception

}finally{

// cleanup code


}
```

Handling Multiple Exception

We can handle multiple type of exceptions in Java by using multiple catch blocks, each catching a different type of exception.

```
try {

    // Code that may throw an exception

} catch (ArithmeticException e) {

    // Code to handle the exception

} catch(ArrayIndexOutOfBoundsException e){

    //Code to handle the anothert exception

}catch(NumberFormatException e){

     //Code to handle the anothert exception
```

}

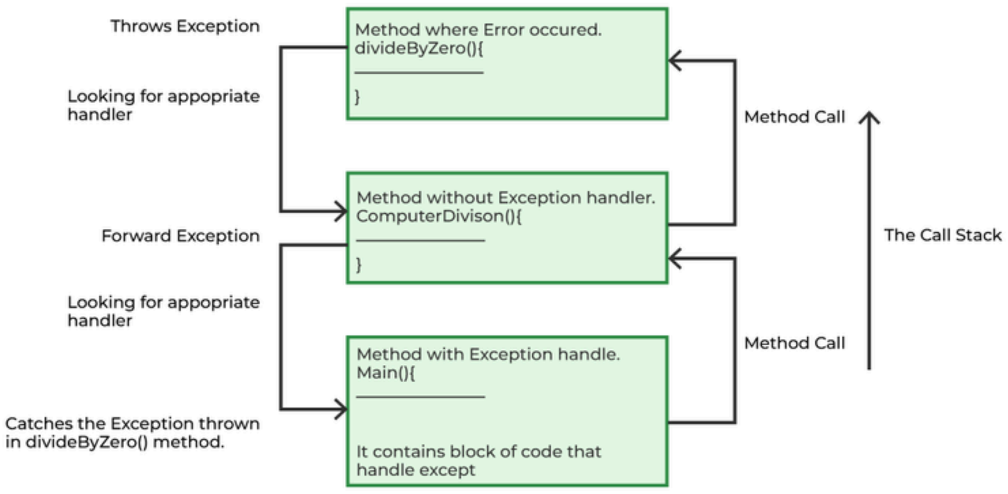
How Does JVM Handle an Exception?

Default Exception Handling: When an Exception occurs, the JVM Creates an exception object containing the error name, description, and program state. Creating the Exception Object and handling it in the run-time system is called throwing an Exception. There might be a list of the methods that had been called to get to the method where an exception occurred. This ordered list of methods is called Call Stack. Now the following procedure will happen.

1. The run-time system searches the call stack for an Exception handler
2. It starts searching from the method where the exception occurred and proceeds backward through the call stack.
3. If a handler is found, the exception is passed to it.
4. If no handler is found, the default exception handler terminates the program and prints the stack trace.

Exception in thread "abc" Name of Exception : Description
... . // Call Stack

Look at the below diagram to understand the flow of the call stack.

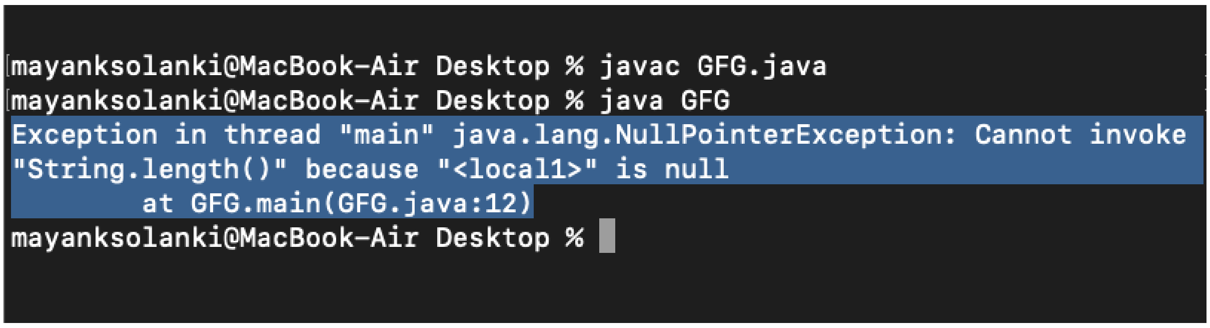

Throws Exception
Looking for appopriate handler
Forward Exception
Looking for appopriate handler
Catches the Exception thrown in divideByZero() method.
Method where Error occured. divideByZero(){ }
Method without Exception handler. ComputerDivison(){ }
Method with Exception handle. Main(){ It contains block of code that handle except
Method Call
Method Call
The Call Stack

The Call Stack and searching the call stack for exception handler.

Illustration:

```java
class Geeks{

    public static void main(String args[])
    {
        // Taking an empty string
        String s = null;

        // Getting length of a string
        System.out.println(s.length());
    }
}
```

Output:



```
mayanksolanki@MacBook-Air Desktop % javac GFG.java
mayanksolanki@MacBook-Air Desktop % java GFG
Exception in thread "main" java.lang.NullPointerException: Cannot invoke
"String.length()" because "<local1>" is null
        at GFG.main(GFG.java:12)
mayanksolanki@MacBook-Air Desktop %
```

Let us see an example that illustrates how a run-time system searches for appropriate exception handling code on the call stack.

Example:

```java
// Class
// ExceptionThrown
class Geeks {

    // It throws the Exception(ArithmeticException)
    // Appropriate Exception handler is not found
    // within this method
    static int divideByZero(int a, int b)
    {

        // this statement will cause ArithmeticException
        // (/by zero)
        int i = a / b;

        return i;
    }

    // The runTime System searches the appropriate
    // Exception handler in method also but couldn't have
    // found. So looking forward on the call stack
    static int computeDivision(int a, int b)
    {

        int res = 0;

        // Try block to check for exceptions
        try {

            res = divideByZero(a, b);
        }

        // Catch block to handle NumberFormatException
        // exception doesn't matches with
        // ArithmeticException
        catch (NumberFormatException ex) {

            System.out.println(
                "NumberFormatException is occurred");
        }
        return res;
    }

    // Found appropriate Exception handler
    // i.e. matching catch block.
    public static void main(String args[])
    {

        int a = 1;
        int b = 0;

        // Try block to check for exceptions
        try {
            int i = computeDivision(a, b);
        }

        // Catch block to handle ArithmeticException
        // exceptions
        catch (ArithmeticException ex) {

            // getMessage() will print description
            // of exception(here / by zero)
            System.out.println(ex.getMessage());
        }
    }
}
```

Output

```
/ by zero
```

How Programmer Handle an Exception?

Customized Exception Handling: Java exception handling uses five keywords: try, catch, throw and throws, and finally. Code that might cause an exception goes in the try block. If an exception occurs, it is caught using catch. We can throw exceptions manually with throw, and methods must declare exceptions they can throw using throws. The finally block is used for code that must run after try, whether an exception occurs or not.

Tip: *One must go through control flow in try catch finally block for better understanding.*

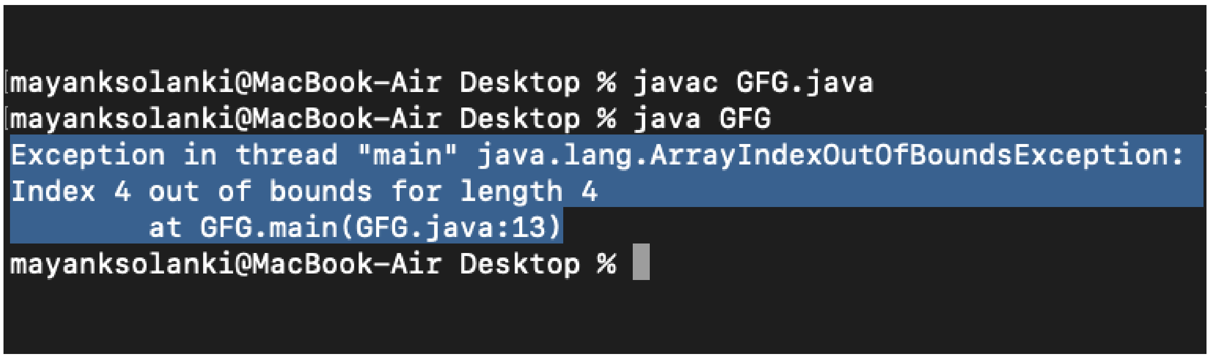
Need for try-catch clause (Customized Exception Handling)

Consider the below program in order to get a better understanding of the try-catch clause.

Example:

```java
// Java Program to Demonstrate
// Need of try-catch Clause
class Geeks {

    public static void main(String[] args) {

        // Taking an array of size 4
        int[] arr = new int[4];

        // Now this statement will cause an exception
        int i = arr[4];

        // This statement will never execute
        // as above we caught with an exception
        System.out.println("Hi, I want to execute");
    }
}
```

Output:



```
mayanksolanki@MacBook-Air Desktop % javac GFG.java
mayanksolanki@MacBook-Air Desktop % java GFG
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
Index 4 out of bounds for length 4
        at GFG.main(GFG.java:13)
mayanksolanki@MacBook-Air Desktop %
```

Explanation: In the above example, an array is defined with size i.e. we can access elements only from index 0 to 3. But we trying to access the elements at index 4 (by mistake) that is why it is throwing an exception. In this case, JVM terminates the program abnormally. The statement System.out.println("Hi, I want to execute"); will never execute. To execute it, we must handle the exception using try-catch. Hence to continue the normal flow of the program, we need a try-catch clause.

Advantages of Exception Handling

- Provision to Complete Program Execution
- Easy Identification of Program Code and Error-Handling Code
- Propagation of Errors
- Meaningful Error Reporting
- Identifying Error Types

Next Article

Java Checked vs Unchecked Exceptions

Similar Reads

Comparison of Exception Handling in C++ and Java

Both languages use to try, catch and throw keywords for exception handling, and their meaning is also the same in both languages. Followi...

4 min read

Java - Exception Handling With Constructors in Inheritance

Java provides a mechanism to handle exceptions. To learn about exception handling, you can refer to exceptions in java. In this article, we discuss...

7 min read

Output of Java program | Set 12(Exception Handling)

Prerequisites : Exception handling , control flow in try-catch-finally 1) What is the output of the following program? public class Test { public...