EXCEPTION HANDLING

What is exception handling?

The **Exception Handling in Java** is one of the powerful mechanisms to handle the runtime errors so that the normal flow of the application can be maintained.

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime. Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

**Advantage of Exception Handling:**

The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions. Let's consider a scenario:

1. statement 1;
2. statement 2;
3. statement 3;
4. statement 4;
5. statement 5;//exception occurs
6. statement 6;
7. statement 7;
8. statement 8;
9. statement 9;
10. statement 10;

Suppose there are 10 statements in a Java program and an exception occurs at statement 5; the rest of the code will not be executed, i.e., statements 6 to 10 will not be executed. However, when we perform exception handling, the rest of the statements will be executed. That is why we use exception handling in java.

### Types of Java Exceptions:

There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception. However, according to Oracle, there are three types of exceptions namely:

1. Checked Exception
2. Unchecked Exception
3. Error

## Difference between Checked and Unchecked Exceptions

### 1) Checked Exception

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

### 2) Unchecked Exception

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

### 3) Error

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

## **Java Exception Keywords**

Java provides five keywords that are used to handle the exception. The following table describes each.

**Try**: The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.

**Catch**: The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.

**Finally**: The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.

**Throw**: The "throw" keyword is used to throw an exception.

**Throws**: The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

## **JAVA EXCEPTION HANDLING EXAMPLE**

Let's see an example of Java Exception Handling in which we are using a try-catch statement to handle the exception.

**JavaExceptionExample.java**

**public** **class** JavaExceptionExample{

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

**try**{

      //code that may raise exception

**int** data=100/0;

   }**catch**(ArithmeticException e){System.out.println(e);}

   //rest code of the program

   System.out.println("rest of the code...");  }}

**Output:**

Exception in thread main java.lang.ArithmeticException:/ by zero

rest of the code...

## **Common Scenarios of Java Exceptions**

There are given some scenarios where unchecked exceptions may occur. They are as follows:

### 1) A scenario where ArithmeticException occurs

If we divide any number by zero, there occurs an ArithmeticException.

1. **int** a=50/0;//ArithmeticException

### 2) A scenario where NullPointerException occurs

If we have a null value in any [variable](https://www.javatpoint.com/java-variables), performing any operation on the variable throws a NullPointerException.

1. String s=**null**;
2. System.out.println(s.length());//NullPointerException

### 3) A scenario where NumberFormatException occurs

If the formatting of any variable or number is mismatched, it may result into NumberFormatException. Suppose we have a [string](https://www.javatpoint.com/java-string) variable that has characters; converting this variable into digit will cause NumberFormatException.

1. String s="abc";
2. **int** i=Integer.parseInt(s);//NumberFormatException

### 4) A scenario where ArrayIndexOutOfBoundsException occurs

When an array exceeds to it's size, the ArrayIndexOutOfBoundsException occurs. there may be other reasons to occur ArrayIndexOutOfBoundsException. Consider the following statements.

1. **int** a[]=**new** **int**[5];
2. a[10]=50; //ArrayIndexOutOfBoundsException

**TryCatchExample6.java**

1. **public** **class** TryCatchExample6 {
3. **public** **static** **void** main(String[] args) {
4. **int** i=50;
5. **int** j=0;
6. **int** data;
7. **try**
8. {
9. data=i/j; //may throw exception
10. }
11. // handling the exception
12. **catch**(Exception e)
13. {
14. // resolving the exception in catch block
15. System.out.println(i/(j+2));
16. }
17. }
18. }

**Output:**

**25**