Exception Handling in Java

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

What is Exception in Java?

**Dictionary Meaning:** Exception is an abnormal condition.

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

What is Exception Handling?

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](https://www.javatpoint.com/java-tutorial).

Hierarchy of Java Exception classes

The java.lang.Throwable class is the root class of Java Exception hierarchy inherited by two subclasses: Exception and Error. The hierarchy of Java Exception classes is given below:



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.

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| 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...");

  }

}

# Java Exception Propagation

An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method. If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack. This is called exception propagation.

#### **Note: By default Unchecked Exceptions are forwarded in calling chain (propagated).**

## Exception Propagation Example

**TestExceptionPropagation1.java**

**class** TestExceptionPropagation1{

**void** m(){

**int** data=50/0;

  }

**void** n(){

    m();

  }

**void** p(){

**try**{

    n();

   }**catch**(Exception e){System.out.println("exception handled");}

  }

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

   TestExceptionPropagation1 obj=**new** TestExceptionPropagation1();

   obj.p();

   System.out.println("normal flow...");

  }

}

### What is String in Java?

Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

### How to create a string object?

There are two ways to create String object:

1. By string literal
2. By new keyword

### 1) String Literal

Java String literal is created by using double quotes. For Example:

1. String s="welcome";

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

1. String s1="Welcome";
2. String s2="Welcome";//It doesn't create a new instance



In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "Welcome" in string constant pool that is why it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.

#### **Note: String objects are stored in a special memory area known as the "string constant pool".**

### Why Java uses the concept of String literal?

To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

### 2) By new keyword

1. String s=**new** String("Welcome");//creates two objects and one reference variable

In such case, [JVM](https://www.javatpoint.com/jvm-java-virtual-machine) will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

### Java String Example

**StringExample.java**

**public** **class** StringExample{

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

String s1="java";//creating string by Java string literal

**char** ch[]={'s','t','r','i','n','g','s'};

String s2=**new** String(ch);//converting char array to string

String s3=**new** String("example");//creating Java string by new keyword

System.out.println(s1);

System.out.println(s2);

System.out.println(s3);

}}