

Exceptions in Java

What is an Exception?

An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e at run time, that disrupts the normal flow of the program's instructions.

Types of error

1. Compile time errors e.g (syntax error) (; "----- ,)
2. Run time error:
 - a. when program successfully compiled and run but we did not get desired output because of some sort of logical mistake in our code. And these errors can be solve after reviewing the code.
 - b. There are some sort of run time Errors that terminate the program e.g divided by 0, stack over flow, interpreter display a message and program will be terminated called exception.
 - c. Exception is unusual condition that error occurs during run time execution. Interpreter create an exception object for that error and throw.
 - d. so here we need to handle these exception objects that are thrown when an runtime error occurs so that remaining code in program can run.

Exception handling mechanism

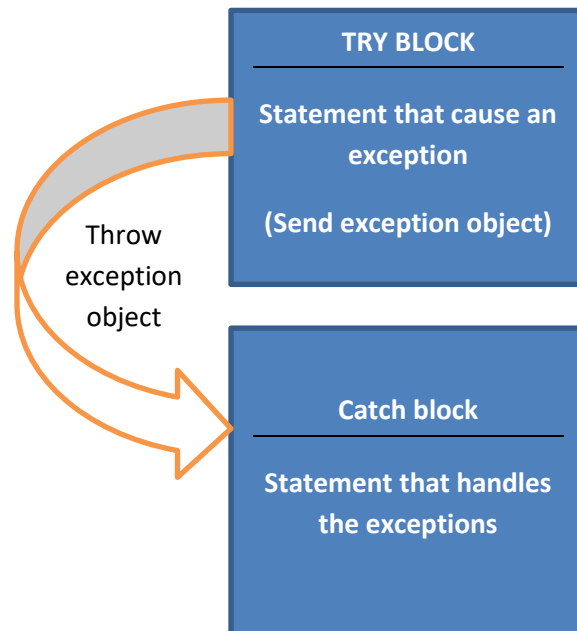
1. **Hit the exception:** find exception
2. **Throw the exception:** object create and throw
3. **Catch the exception:** catch thus exception
4. **Handle the exception:** Appropriate action perform for this exception

Exception object creator

```
Try{  
Statement;}
```

Exception handler

```
Catch(Exception_type obj){  
Statement;}
```



Advantage of Exception Handling

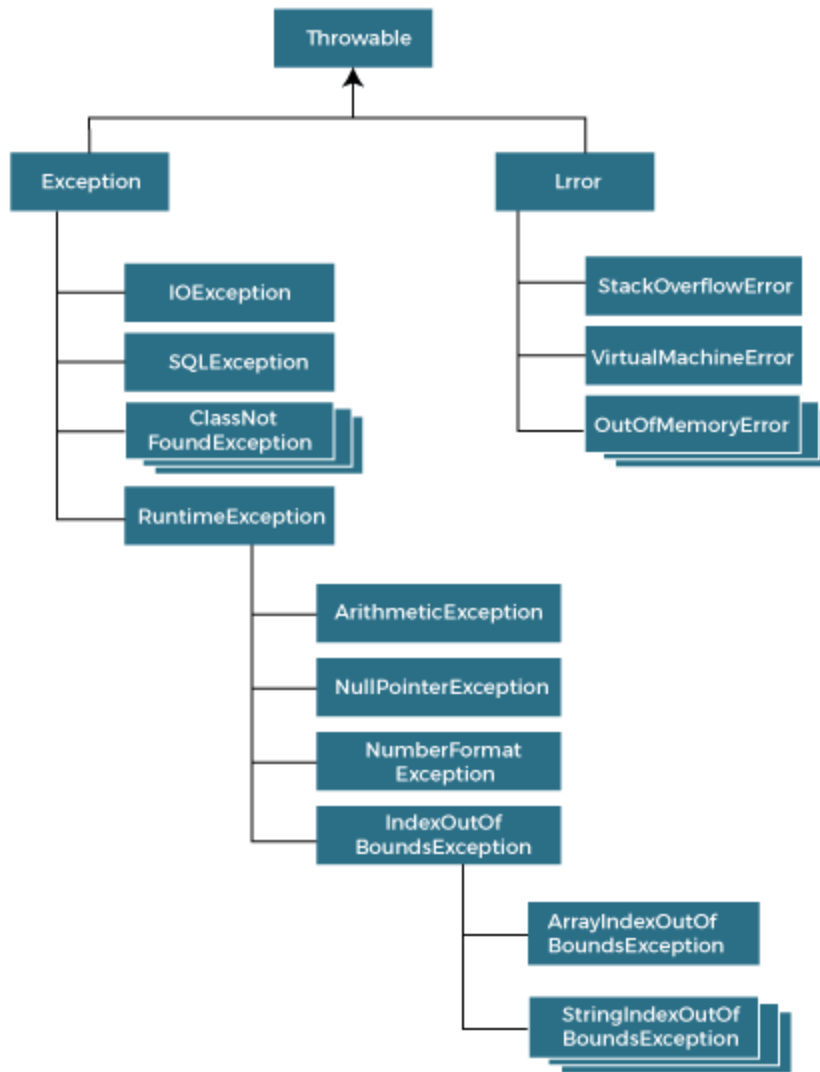
The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions. Let's consider a scenario:

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](#).

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

```

1. public class JavaExceptionExample{
2.     public static void main(String args[]){
3.         try{
4.             //code that may raise exception
5.             int data=100/0;
6.         }catch(ArithmeticException e){System.out.println(e);}
7.         //rest code of the program
8.         System.out.println("rest of the code...");
9.     }
10. }
```

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

In the above example, 100/0 raises an ArithmeticException which is handled by a try-catch block.

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](#), 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](#) 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 its 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`