**What is an exception?**

An Exception is an unwanted event that interrupts the normal flow of the program execution. When an exception occurs program execution gets terminated immediatly.

**Types of exceptions:**

There are two types of exceptions in Java:

**1) Checked exceptions**

**2) Unchecked exceptions**

**1) Checked Exceptions or Compiletime exceptions:**

Exceptions which are checked at the compile-time are called checked exceptions.

**Checked Exceptions Examples:**

1. ClassNotFoundException
2. InterruptedException
3. InstantiationException
4. IOException
5. SQLException
6. IllegalAccessException
7. FileNotFoundException, etc

**2) Unchecked exceptions or RuntimeException:**

Unchecked exceptions are the exceptions that are checked at run time.

**Unchecked Exceptions Examples:**

1. ArithmeticException
2. ClassCastException
3. NullPointerException
4. ArrayIndexOutOfBoundsException
5. NegativeArraySizeException
6. ArrayStoreException
7. IllegalThreadStateException
8. SecurityException, etc.
9. NumberFormatException

## List of Exceptions:

1) ArithmeticException:

Arithmetic abnormalities like divide by zero results in ArithmeticException.

2) ArrayIndexOutOfBoundsException:

ArrayIndexOutOfBoundsException is thrown when an array element is accessed using an illegal index. The index used is either beyond the size of the array or is negative.

3) ClassNotFoundException:

If the class definition is not found then the ClassNotFoundException is raised.

4) FileNotFoundException:

FileNotFoundException is given when the file does not exist or does not open.

5) IOException:

IOException is thrown when the input-output operation fails or is interrupted.

6) InterruptedException:

Whenever a thread is doing processing or sleeping or waiting, then it is interrupted by throwing InterruptedException.

7) NoSuchFieldException:

If a class does not contain a specified field or variable, then it throws NoSuchFieldException.

8) NoSuchMethodException:

When the method being accessed is not found, then NoSuchMethodException is raised.

9) NullPointerException:

NullPointerException is raised when a null object is referred. This is the most important and most common exception in Java.

**10)** NumberFormatException:

This exception is raised when a method could not convert a string into a numeric format.

11) RuntimeException:

Any exception that occurs at runtime is a RuntimeException.

12) StringIndexOutOfBoundsException:

The StringIndexOutOfBoundsException is thrown by String class and indicates that the index is beyond the size of the String object or is negative.

13) EOFException:

EOFException is a part of the java.io package and is thrown when the end of file is reached and the file is being read.

14) IllegalArgumentException:

IllegalArgumentException is thrown when illegal or invalid arguments are passed to the method. For example, the wrong data format, null value when non-null is required or out of range arguments.

**Try-catch-finally**

A program can catch exceptions by using a combination of the try, catch, and finally blocks.

* The try block identifies a block of code in which an exception can occur.
* The catch block identifies a block of code, known as an exception handler, that can handle a particular type of exception.
* The finally block identifies a block of code that is guaranteed to execute, and is the right place to close files, recover resources, and otherwise clean up after the code enclosed in the try block.
* For each try block, there can be zero or more catch blocks. Multiple catch blocks allow us to handle each exception differently.

try {

//code

}

catch (ExceptionType1 e1) {

// catch block

}

finally {

// finally block always executes

}

### **Throw vs Throws:**

throws clause is used to declare an exception and throw keyword is used to throw an exception explicitly.

1. If we see syntax wise then throw is followed by an instance variable and throws is followed by exception class names.
2. The keyword throw is used inside method body to invoke an exception and throws clause is used in method declaration (signature).

**For example** :

throw

throw new Exception("You have some exception")

throw new IOException("Connection failed!!")

throws

public int myMethod() throws IOException, ArithmeticException, NullPointerException {}

1. You cannot declare multiple exceptions with throw. You can declare multiple exception e.g. public void method()throws IOException,SQLException.
2. checked exceptions can not be propagated with throw only because it is explicitly used to throw an particular exception. checked exception can be propagated with throws.

### Example: Exception handling using Java throw

class Main {

public static void divideByZero() {

// throw an exception

throw new ArithmeticException("Trying to divide by 0");

}

public static void main(String[] args) {

divideByZero();

}

}

### Example: Java throws keyword

import java.io.\*;

class Main {

// declareing the type of exception this method throws

public static void findFile() throws IOException {

// code that may generate IOException

File newFile = new File("test.txt");

FileInputStream stream = new FileInputStream(newFile);

}

public static void main(String[] args) {

try {

findFile(); // caller should handle the exception

}

catch (IOException e) {

System.out.println(e);

}

}

}

**Exception vs Error:**

**Error** and **Exception** both extend **Throwable**, but mostly Error is thrown by JVM in a scenario which is fatal and there is no way for the application program to recover from that error. For instance **OutOfMemoryError**.

Though even application can raise an Error but its just not a good a practice, instead applications should use checked exceptions for recoverable conditions and runtime exceptions for programming errors.

## **Writing your own exception class**

* Create a new class whose name should end with Exception like ClassNameException.
* Make the class extends one of the exceptions which are subtypes of the java.lang.Exception class.
* Generally, a custom exception class always extends directly from the Exception class.
* Create a constructor with a String parameter which is the detail message of the exception. In this constructor, simply call the super constructor and pass the message.

The following is a custom exception class which is created by following the above steps:

public class StudentNotFoundException extends Exception {

public StudentNotFoundException(String message) {

super(message);

}

}

And the following example shows the way a custom exception is used is nothing different than built-in exception:

public class StudentManager {

public Student find(String studentID) throws StudentNotFoundException {

if (studentID.equals("123456")) {

return new Student();

} else {

throw new StudentNotFoundException(

"Could not find student with ID " + studentID);

}

}

}

And the following test program handles that exception:

public class StudentTest {

public static void main(String[] args) {

StudentManager manager = new StudentManager();

try {

Student student = manager.find("0000001");

} catch (StudentNotFoundException ex) {

System.err.print(ex);

}

}

}

Run this program and you will see this output:

StudentNotFoundException: Could not find student with ID 0000001