Top of Form

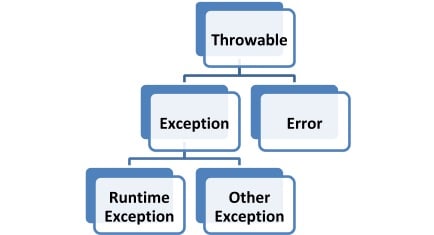
**Java**is an object-oriented programming language. It provides support for various mechanisms such as exception handling. This feature of Java enables developers to manage the runtime errors caused by the exceptions.

Exceptions are the unwanted errors or bugs or events that restrict the normal execution of a program. Each time an exception occurs, program execution gets disrupted. An error message is displayed on the screen.

There are several reasons behind the occurrence of exceptions. These are some conditions where an exception occurs:

* Whenever a user provides invalid data.
* The file requested to be accessed does not exist in the system.
* When the **Java Virtual Machine** (JVM) runs out of memory.
* Network drops in the middle of communication.

The parent class of all the exception classes is the **java.lang.Exception** class. Figure 1 illustrates the different types of Java exceptions.



If we talk about the **Exception** class, it is a subclass of the built-in **Throwable** class. There is another subclass which is derived from the Throwable class i.e. **Error** as illustrated in Figure 1. The error can be defined as an abnormal condition that indicates something has gone wrong with the execution of the program. These are not handled by Java programs.

**Checked And Unchecked Exceptions in Java**

Java supports checked and unchecked exceptions.

Checked exceptions for all exceptional events that you can anticipate and that a well-written application should be able to handle. A checked exception extends the Exception class. A method that throws a checked exception or that calls a method that specifies a checked exception need to either specify or handle it.

Unchecked exceptions extend the RuntimeException. You should use them for internal errors that you can’t anticipate and that, most often, the application can’t recover from. Methods can but don’t need to handle or specify an unchecked exception. Typical examples that throw unchecked exceptions are:

* the missing initialization of a variable which results in a NullPointerException or
* the improper use of an API that causes an IllegalArgumentException

#### ****Checked Exceptions****

1. ClassNotFoundException: Class not found.
2. CloneNotSupportedException: Attempt to clone an object that does not implement the Cloneable interface.
3. IllegalAccessException: Access to a class is denied. IllegalAccessException signals that a particular method could not be found.
4. InstantiationException: Attempt to create an object of an abstract class or interface.
5. InterruptedException: One thread has been interrupted by another thread.
6. NoSuchFieldException: A requested field does not exist.
7. NoSuchMethodException: A requested method does not exist.
8. ReflectiveOperationException - Superclass of reflection -related exceptions(Added by JDK 7.)

#### ****Runtime - Unchecked Exceptions****

1. ArithmeticException: Arithmetic error, such as divide-by-zero.
2. ArrayIndexOutOfBoundsException: Array index is out-of-bounds.
3. ArrayStoreException: Assignment to an array element of an incompatible type.
4. ClassCastException: Invalid cast.
5. EnumConstantNotPresentException: An attempt is made to use an undefined enumeration value
6. IllegalArgumentException: Illegal argument used to invoke a method.
7. IllegalMonitorStateException: Illegal monitor operation, such as waiting on an unlocked thread.
8. IllegalStateException: Environment or application is in incorrect state.
9. IllegalThreadStateException: Requested operation not compatible with the current thread state.
10. IndexOutOfBoundsException: Some type of index is out-of-bounds.
11. NegativeArraySizeException: Array created with a negative size.
12. NullPointerException: Invalid use of a null reference.
13. NumberFormatException: Invalid conversion of a string to a numeric format.
14. SecurityException: Attempt to violate security.
15. StringIndexOutOfBounds: Attempt to index outside the bounds of a string.
16. TypeNotPresentException: Type not found. (Added by J2SE 5.)
17. UnsupportedOperationException: An unsupported operation was encountered.

Bottom of Form

Top of Form

#### Checked Exceptions

#### SQLException

This type of exception occurs while executing queries on a database related to the SQL syntax. For example, consider the following code snippet:

public void setClientInfo(String sname, String svalue) throws **SQLClientInfoException** {

try {

checkClosed();

((**java.sql.Connection**) this.mc).setClientInfo(sname, svalue);

} catch (**SQLException** sqlEx) {

try {

checkAndFireConnectionError(sqlEx);

} catch (SQLException sqlEx2) {

SQLClientInfoException client\_Ex = new SQLClientInfoException();

client\_Ex.initCause(sqlEx2);

throw client\_Ex;

}

}

}

Output: This code will generate a SQLException.

#### ****IOException****

This type of exception occurs while using file I/O stream operations. For example, consider the following code snippet:

import java.io.\*;

public class sample\_IOException {

private static String filepath = "D:\User\guest\Desktop\File2.txt";

public static void main(String[] args) {

BufferedReader br1 = null;

String curline;

try {

br1 = new BufferedReader(new FileReader(filepath));

while ((curline = br1.readLine()) != null) {

System.out.println(curline);

}

} catch (IOException e) {

System.err.println("IOException found :" + e.getMessage());

} finally {

try {

if (br1 != null)

br1.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

Output: This code will generate an IOException.

#### ClassNotFoundException

This type of exception is thrown when the JVM is not able to find the required class. It may be due to a command-line error, a classpath issue, or a missing .class file. For example, consider the following code snippet:

public class sample\_ClassNotFoundException {

private static final String CLASS\_TO\_LOAD = "main.java.Utils";

public static void main(String[] args) {

try {

Class loadedClass = Class.forName(CLASS\_TO\_LOAD);

System.out.println("Class " + loadedClass + " found!");

} catch (ClassNotFoundException ex) {

System.err.println("ClassNotFoundException was found: " + ex.getMessage());

ex.printStackTrace();

}

}

}

Output: This code will generate a ClassNotFoundException.

#### InvocationTargetException

This type of exception wraps an exception thrown by an invoked method or a constructor. The thrown exception can be accessed with the help of the getTargetException method. For example, consider the following code snippet:

package main.samplejava;

import java.lang.reflect.InvocationTargetException;

import java.lang.reflect.Method;

public class Example {

@SuppressWarnings("unused")

private int test\_sample(String s1) {

if (s1.length() == 0)

throw new IllegalArgumentException("The string should have at least one character!");

System.out.println("Inside test\_sample: argument's value equals to: "" + s1 + """);

return 0;

}

public static void main(String... args) {

try {

Class<?> c1 = Class.forName("main.samplejava. Example");

Object t1 = c1.newInstance();

Method[] declared\_Methods = c1.getDeclaredMethods();

for (Method method : declared\_Methods) {

String methodName = method.getName();

if (methodName.contains("main"))

continue;

System.out.format("Invoking %s()%n", methodName);

try {

method.setAccessible(true);

Object returnValue = method.invoke(t1, "");

System.out.format("%s() returned: %d%n", methodName, returnValue);

} catch (InvocationTargetException ex) {

System.err.println("An InvocationTargetException was caught!");

Throwable cause = ex.getCause();

System.out.format("Invocation of %s failed because of: %s%n",

methodName, cause.getMessage());

}

}

} catch (ClassNotFoundException | InstantiationException | IllegalAccessException ex) {

System.err.println("The following exception was thrown:");

ex.printStackTrace();

}

}

}

Output:

Invoking testMethod()

An InvocationTargetException was caught!

Invocation of testMethod failed because of: The string must contain at least one character!

Output: This code will generate an InstantiationException.

Bottom of Form

Top of Form

#### Unchecked Exceptions:

#### NullPointerException

This type of exception occurs when you try to access an object with the help of a reference variable whose current value is null or empty. For example, consider the following code snippet:

// Program to demonstrate the NullPointerException

class SampleNullPointer {

public static void main(String args[]) {

try {

String a1 = null; // null value

System.out.println(a1.charAt(0));

} catch(NullPointerException e) {

System.out.println("NullPointerException is found in the program.");

}

}

}

Output: NullPointerException is found in the program.

#### ArrayIndexOutofBound

This type of exception occurs when you try to access an array with an invalid index value. The value you are providing is either negative or beyond the length of the array.

For example, consider the following code snippet:

// Program to demonstrate the ArrayIndexOutOfBoundException

class sample\_ArrayIndexOutOfBound {

public static void main(String args[]) {

try {

int b[] = new int[6];

b[8] = 2; // we are trying to access 9th element in an array of size 7

} catch(ArrayIndexOutOfBoundsException e) {

System.out.println ("The array index is out of bound");

}

}

}

Output: The array index is out of bound

#### IllegalArgumentException

This type of exception occurs whenever an inappropriate or incorrect argument is passed to a method. For example, if a method is defined with non-empty string as parameters. But you are providing null input strings. Then, the [IllegalArgumentException](https://docs.oracle.com/javase/7/docs/api/java/lang/IllegalArgumentException.html) is thrown to indicate the user that you cannot pass a null input string to the method.

Consider the following code snippet to demonstrate this type of exception:

import java.io.File;

public class Sample\_IllegalArgumentException {

public static String createRelativePath(String par, String f\_name) {

if (par == null)

throw new IllegalArgumentException("You cannot provide null parent path!");

if (f\_name == null)

throw new IllegalArgumentException("Please enter the complete filename!");

return par + File.separator + f\_name;

}

public static void main(String[] args) {

// This command will be successfully executed.

system.out.println(IllegalArgumentExceptionExample.createRelativePath("dir1", "file1"));

system.out.println();

// This command will throw an IllegalArgumentException.

System.out.println(IllegalArgumentExceptionExample.createRelativePath(null, "file1"));

}

}

Output: This code will generate an IllegalArgumentException.

#### IllegalStateException

This type of exception occurs when the state of the environment does not match the operation being executed. For example, consider the following code snippet, which demonstrates this type of exception:

/\*\*

\* This code will publish the current book.

\* If the book is already published, it will throw an IllegalStateException.

\*\*/

public void pub() throws IllegalStateException {

Date pub\_at = getPub\_at();

if (pub\_at == null) {

setPub\_at(new Date());

Logging.log(String.format("Published '%s' by %s.", getTitle(), getAuthor()));

} else {

throw new IllegalStateException(

String.format("Cannot publish '%s' by %s (already published on %s).",

getTitle(), getAuthor(), pub\_at));

}

}

Output: This code will generate IllegalStateException.

If a publication date already exists in the system, then it will produce an IllegalStateException that indicates that the book cannot be published again.

#### NumberFormatException

This type of exception occurs when you pass a string to a method that cannot be converted to a number. For example, consider the following code snippet:

// Program to demonstrate the NumberFormatException

class Sample\_NumberFormat {

public static void main(String args[]) {

try {

// "Test" is not a number

int n = Integer.parseInt ("Test") ;

System.out.println(n);

} catch(**NumberFormatException** e) {

System.out.println("Number format exception");

}

}

}

Output: This code will generate NumberFormatException.

#### ArithmeticException

This type of exception occurs when you perform an incorrect arithmetic operation. For example, if you divide any number by zero, it will display such an exception. Let us consider the following code snippet:

// Program to demonstrate the ArithmeticException

class Sample\_ArithmeticException {

public static void main(String args[]) {

try {

int p = 30, q = 0;

int r = p/q; // It cannot be divided by zero

System.out.println ("Result = " + r);

} catch(ArithmeticException e) {

System.out.println ("Number cannot be divided by 0");

}

}

}

Output: This code will generate an ArithmeticException.

Bottom of Form

Top of Form

#### Java Custom Exception

If you are creating your own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.

By the help of custom exception, you can have your own exception and message.

Let's see a simple example of java custom exception.

1. **class** InvalidAgeException **extends** Exception{
2. InvalidAgeException(String s){
3. **super**(s);
4. }
5. }
6. **class** TestCustomException1{
8. **static** **void** validate(**int** age)**throws** InvalidAgeException{
9. **if**(age<18)
10. **throw** **new** InvalidAgeException("not valid");
11. **else**
12. System.out.println("welcome to vote");
13. }
15. **public** **static** **void** main(String args[]){
16. **try**{
17. validate(13);
18. }**catch**(Exception m){System.out.println("Exception occured: "+m);}
20. System.out.println("rest of the code...");
21. }
22. }

Output:Exception occured: InvalidAgeException:not valid

rest of the code...

**Another Example**

// A Class that represents use-defined expception

class MyException extends Exception

{

public MyException(String s)

{

// Call constructor of parent Exception

super(s);

}

}

// A Class that uses above MyException

public class Main

{

// Driver Program

public static void main(String args[])

{

try

{

// Throw an object of user defined exception

throw new MyException("GeeksGeeks");

}

catch (MyException ex)

{

System.out.println("Caught");

// Print the message from MyException object

System.out.println(ex.getMessage());

}

}

}

Bottom of Form

Top of Form

Bottom of Form

Top of Form

##### **Reference Links:**

ONLINE NOTES LINKS:

[https://www.tutorialspoint.com/java/java\_exceptions.htm](https://www.tutorialspoint.com/java/java_exceptions.htm#:~:text=An%20exception%20(or%20exceptional%20event,exceptions%20are%20to%20be%20handled)

<https://www.javatpoint.com/exception-handling-in-java>

<https://www.geeksforgeeks.org/exceptions-in-java/>

<https://www.w3resource.com/java-tutorial/types-of-exception.php>

<https://www.protechtraining.com/content/java_fundamentals_tutorial-exceptions>

VIDEO LINKS:

<https://www.javatpoint.com/exception-handling-in-java>

<https://www.edureka.co/blog/java-exception-handling>

<https://www.youtube.com/watch?v=4my7mKFaNQs>

<https://www.youtube.com/watch?v=W-N2ltgU-X4>

<https://marcus-biel.com/advanced-exception-handling-in-java/>

Bottom of Form

Help for current page