# Exception Handling

Exception : *It is an object which is thrown at runtime and disrupts normal flow of program.*

**Why Exception Handling?**

To maintain normal execution of application.

**Checked Exception :** The exceptions which are checked by compiler at runtime for smooth execution of program is called checked exception. ***These are most commonly occurring exceptions so it must be handled by programmer using try catch or throws***. Whether programmer handling or not compiler always going to check. In unchecked compiler is never going to check whether programmer is handling or not.

Note : throws followed by some exception is checked exception only.

***Unchecked Exception*** : The exceptions which are not checked by compiler are called unchecked exceptions. In case of unchecked exception compiler wont check whether Programmer is handling exception or not.

EX : NullPionterException, ArithmeticException, ArrayIndexOutOfBoundsException, NumberFormatException, ClassCastException

Note-1 : whether exception is checked or unchecked compulsory it will occur only at runtime. There is no chance of occurring any exception at compile time. Exceptions are objects which are thrown at run time only.

Note-2 : Runtime exception and its child classes, error and its child classes are the unchecked except this remaining are checked exceptions.

Note-3: Errors are caused by lack of system resource not by programmer. Handled by system admin or server admin

Note-4 : When a statement throws an exception in the try block, the remaining part of the try block will not be executed. Program control comes out of the try block and enters directly into catch block.

NOTE :

Integer i=new Integer("wipro");//No compilation issue. NumberFormatException at Runtime.

There are five keywords in Java to handle exceptions.

1. Try
2. Catch
3. Finally
4. Throw

5. Throws

```
ackage exceptionalHandling;

public class ExceptionHandling {

	public static void main(String[] args) {
		String str=null;
		try{
		int length=str.length();
		}
		catch (Exception e) {// catch accepts exception object as argument
			System.out.println(e);//e prints : java.lang.NullPointerException
			e.printStackTrace();//prints the stack trace
		}

	}

}
```

***What happens when exception is not handled?***

**JVM performs following steps when an exception is not handled-**

- Prints out exception description.
- Prints the stack trace (Hierarchy of methods where the exception occurred).
- Causes the program to terminate

**Single Statement throwing multiple exceptions**

*From Java 7 onward, there is one more way for handling multiple exceptions. Multiple exceptions thrown by the try block can be handled by a single catch block using **pipe** (|) **operator**.*

```
package exceptionalHandling;

public class ExceptionHandling {

	public static void main(String[] args) {
		String[] s = {"abc", "123", null, "xyz"};   //String array containing one null object

     for (int i = 0; i < 6; i++)
     {
        try
        {
            int a = s[i].length() + Integer.parseInt(s[i]);

            //This statement may throw NumberFormatException, NullPointerException and
ArrayIndexOutOfBoundsException
        }

        catch(NumberFormatException | NullPointerException | ArrayIndexOutOfBoundsException ex)

        {

           System.out.println("Now, this block handles NumberFormatException, NullPointerException
and ArrayIndexOutOfBoundsException");
        }
    }
         }

}
```

**Catch block must be ordered from most specific to general otherwise compile time error saying unreachable catch block.**

```
package exceptionalHandling;

public class ExceptionHandling {

	public static void main(String[] args) {

		try
		{
			String str=null;
			int length=str.length();
		}
		catch (Exception e) {//already null pointer exception is handled here.
			System.out.println(e);

		}
		catch (NullPointerException e) {

		}
	}

}
```

**Nested Try Block**

Nested try blocks are useful when different statements of try block throw different types of exceptions

```
package exceptionalHandling;

public class ExceptionHandling {

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

```
                    try
                    {
                                String str=null;
                                int l=str.length();
                                try{

                                }
                                catch (ArithmeticException e) {
                                            int res=320/0;

                                }


                    }
                    catch (NullPointerException e) {
                                e.printStackTrace();
                    }


                    }

}
```

***NOTE :*** It is always advisable to go with one try and multiple catch. If we go for nested try catch then exception should be propagated from one catch to another catch which is more time consuming.

***Finally Block:***

Finally block is used to write important codes like clean up, closing file, closing connection etc.
Finally block is always executed whether Exception is not occurred or exception occurred but not handled or exception handled and occurred.
If we don't handle exception JVM executes finally block if present.
Note : For one try block there may be zero or more catch blocks but only one finally block.
Note : The finally block will not execute if program encounters system.exit();

```
package exceptionalHandling;

public class ExceptionHandling {

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

		int result=20/0;
		System.exit(0);//will
		}
		catch (Exception e) {
			System.out.println("inside catch");
			System.exit(0);
		}
		finally {
			System.out.println("inside finally..");
		}

	} o/p : finally block will be executed as after result statement exception is
thrown and caught in catch block.
```

## Try Block

Try Block must be written inside method.

It must be followed by either catch of finally block.

```
try{
//code that may throw exception
}catch(Exception_class_Name ref){}
```

OR

```
try{
//code that may throw exception
}finally{}
```

### Try- Catch-Finally

```
/* package codechef; // don't place package name! */

import java.util.*;
import java.lang.*;
import java.io.*;

/* Name of the class has to be "Main" only if the class is public. */
class Codechef
{
                public static void main (String[] args) throws java.lang.Exception
                {
                                try {
                                   System.out.println("Inside Try Block");
                                   int a=20/0;


                                } catch(Exception e) {
                                   System.out.println(e);

                                }
                                finally
                   {
                   System.out.println("In finally Block");
                   }


                    System.out.println("outside try-catch-finally ");
                }


}
```

**Please note if method returns value the statement after finally block becomes unreachable code(compilation error)**

**Output :**

```
Inside Try Block
java.lang.ArithmeticException: / by zero
In finally Block
```

outside try-catch-finally

## Return Value From Try-catch-Finally Blocks

If a methods returns a value and also has try-catch-finally block then these two rules come into picture.
**Rule 01 :**
If a finally block returns a value then try and catch block may or may not return value.

```
public class ReturnValueFromTryCatchFinally
{
    public static void main(String[] args)
    {
        System.out.println(methodReturningValue());
    }

    static int methodReturningValue()
    {
        try
        {
            //This block may or may not return a value as finally block is returning a value
        }
        catch (Exception e)
        {
            //This block may or may not return a value as finally block is returning a value
        }
        finally
        {
            return 20;
        }
    }
```

**Rule 02 :**
If a finally block is not returning a value then both try and catch blocks must return a value.

```
public class ReturnValueFromTryCatchFinally
{
    public static void main(String[] args)
    {
        System.out.println(methodReturningValue());
    }

    static int methodReturningValue()
    {
        try
        {
            return 10;
        }
        catch (Exception e)
        {
            return 20;
        }
        finally
        {
            //Now, This block is not returning a value so
            // both try and catch blocks must return a value
        }
    }
}
```

These are the two rules imposed if your function contains try, catch and finally block
Rule : 1 : If finally block returns a value then try and catch blocks may or may not return a value.
Rule : 2 : If finally block returns a value then try and catch blocks may or may not return a value.
Rule : 3 : finally block overrides any return values from try and catch blocks.
Rule : 4 : finally block will be always executed even though try and catch blocks are returning the control
Rule : 5 : If try-catch-finally blocks are returning a value according to above rules, then you should not keep any statements after finally block. Because they become unreachable and in Java, Unreachable code gives compile time error