**EXCEPTIONS**

**Definition:**

Exception is an Event that gets triggered when JVM is not able to execute a statement. It can be handled using try catch block

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.

### 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

Program:

public class Demo1 {

public static void main(String[] args) {

int i=10;

int j;

try

{

j=i/0;

System.out.println("inside try block");

}

catch(ArithmeticException e)

{

e.printStackTrace();

System.out.println("inside catch block");

}

}

}

Here, j=i/0 is triggering in try block so we are catching it in catch block

O/P  
java.lang.ArithmeticException: / by zero

inside catch block

**NOTE**:

**Finally**, block is used along with try catch(Try-catch-finally) mostly or can be used along with try alone(try-finally). This block gets executed irrespective of a statement generating an exception or not , we can have

Try-catch-finally OR try-finally

Program:

public class Demo5 {

public static void main(String[] args) {

int i=10;

int j;

try{

System.out.println("inside try block");

j=i/0;

}

catch(ArithmeticException e)

{

System.out.println("inside catch block");

}

finally

{

System.out.println("inside finally block");

}

System.out.println("i= "+ i);

}

}

O/P:

inside try block

inside catch block

inside finally block

i= 10

**Stack Overflow**

Program:

public class Demo13 {

public static void main(String[] args) {

// TODO Auto-generated method stub

Demo13 d1=new Demo13();

d1.test1();

}

void test1(){

test2();

}

void test2()

{

test3();

}

void test3()

{

test4();

}

void test4(){

int i=10/0;

}

}

O/p:

Exception in thread "main" java.lang.ArithmeticException: / by zero

at com.qsp.ExceptionDemo.Demo13.test4(Demo13.java:23)

at com.qsp.ExceptionDemo.Demo13.test3(Demo13.java:20)

at com.qsp.ExceptionDemo.Demo13.test2(Demo13.java:16)

at com.qsp.ExceptionDemo.Demo13.test1(Demo13.java:11)

at com.qsp.ExceptionDemo.Demo13.main(Demo13.java:8)



Test3()

Test4()

Test2()



Test1()

Main d1

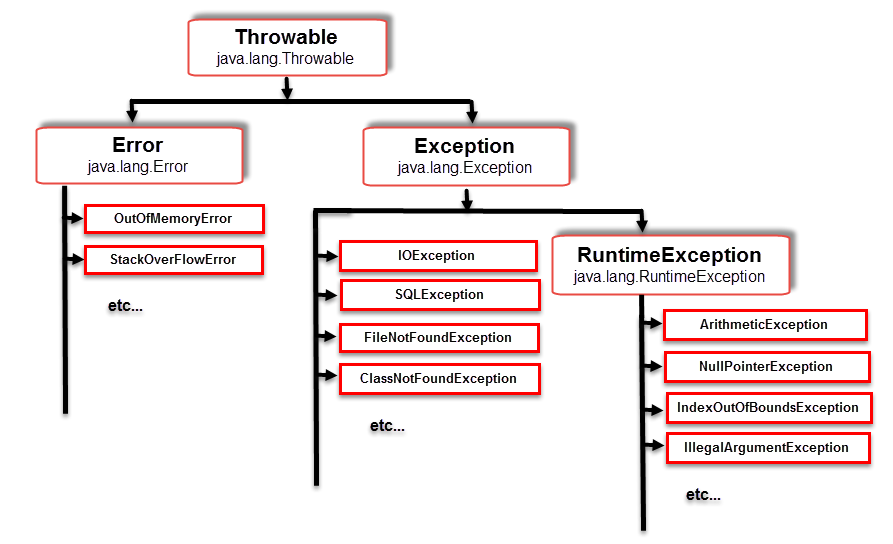
Stack Memory Heap Memory

Q. When does Stack Unwinding Happens?

A. While Executing any program the method of the classes will be loaded into the stack for the execution purpose.Stack will be loaded with methods in the orders the way its called. If the last entered method generates any exception then JVM looks for the handler in the same method, if no handler is found in the method then the exception propogrates to the called method , In the called method if the exception is not handled then the exception will propagates to its called method, finally the exception reaches main method.if the exception is not handled , then JVM forcibily removes all the method from the stack and terminates program execution .this is known as Stack Unwinding

When an exception is thrown it can be caught by reference variable of same class or its super class or its super class….in the catch block

BLOCK DIAGRAM:



**THROWABLE**: it is the **super most class of all exception classes** and its super class is **Object class**

**THROW**: throw keyword is used to throws an exception(existing exception class) in the program.

**THROWS**: its used to handle the exception

The **throws** declaration should be done in the **method signature**, the **throws** keyword should be used only for **checked exceptions**.

Syntax:

Thows Exceptionname

Exception in Java are classified into 2 types

1.Checked Exception

2.Unchecked Exception

3.Error- error is considered as the unchecked exception. Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

**Checked Exception:**

An Exception where the compiler can check at the time of compilation is known as checked Exception

• Checked Exception should be handled in order to compile the program successfully.

• Checked Exception can be handled in 2 ways

1)Surrounding try-catch block

2)By using throws declaration statement

The **throws** declaration should be done in the method signature, the throws keyword should be used only for checked exceptions.

**Unchecked Exception**: Exceptions which are not been able to be identified by the compiler at the time of compilation is known as unchecked exceptions.

1 and 2 are Unchecked exception

3 are checked exception

**throws** keyword cannot be used for **unchecked exception** and it should be handled only through **try,catch** block.

• **throw** keyword is used **to generate** or throws an exception(**existing exception class or user defined exception**) in the program.

• We can develop our own exception class by inheriting one of Java exception classes.

**Do You Know?**

|  |
| --- |
| * What is the difference between checked and unchecked exceptions? * What happens behind the code int data=50/0;? * Why use multiple catch block? * Is there any possibility when the finally block is not executed? * What is exception propagation? * What is the difference between the throw and throws keyword? * What are the 4 rules for using exception handling with method overriding?  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](https://www.javatpoint.com/java-variables), 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](https://www.javatpoint.com/java-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 it's 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 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. no.** | **Key** | **final** | **finally** | **finalize** |
| 1. | Definition | final is the keyword and access modifier which is used to apply restrictions on a class, method or variable. | finally is the block in Java Exception Handling to execute the important code whether the exception occurs or not. | finalize is the method in Java which is used to perform clean up processing just before object is garbage collected. |
| 2. | Applicable to | Final keyword is used with the classes, methods and variables. | Finally block is always related to the try and catch block in exception handling. | finalize() method is used with the objects. |
| 3. | Functionality | (1) Once declared, final variable becomes constant and cannot be modified. (2) final method cannot be overridden by sub class. (3) final class cannot be inherited. | (1) finally block runs the important code even if exception occurs or not. (2) finally block cleans up all the resources used in try block | finalize method performs the cleaning activities with respect to the object before its destruction. |
| 4. | Execution | Final method is executed only when we call it. | Finally block is executed as soon as the try-catch block is executed.  It's execution is not dependant on the exception. | finalize method is executed just before the object is destroyed. |

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

   {

       FinalizeExample obj = **new** FinalizeExample();

        // printing the hashcode

        System.out.println("Hashcode is: " + obj.hashCode());

        obj = **null**;

        // calling the garbage collector using gc()

**System.gc();**

        System.out.println("End of the garbage collection");

    }

   // defining the finalize method

**protected** **void** finalize()

   {             System.out.println("Called the finalize() method");

    }

}