**Exception:**

Exceptions are event that arise due to the occurrence of unexpected behaviour in certain statements, disrupting the normal execution of a program.

Exception can arise due to a number of situations. For example,

1. Trying to access the 11th element of an array when the array contains only 10 elements(ArrayIndexOutOfBoundsException)
2. Division by zero(ArithmeticException)
3. Accessing a file which is not present(FileNotFoundException)

There are predefined classes for all exception types representing each such situation. The topmost class in the hierarchy is java.lang.Throwable. This class has two siblings: Error and Exception. All the classes representing exceptional conditions are subclass of the Exception class. Whenever an exception occurs in a method, the runtime environment identifies the type of Exception and throw the object of it. If the method does not employ any exception handling mechanism, then the exception is passed to the caller method, and so on.

**Example**

class Ex1

{

public static void main(String args[])

{

method1();

}

static void method1()

{

System.out.println("In method 1,calling method 2");

method2();

System.out.println("Returned from method 2");

}

static void method2()

{

System.out.println("In method 2,calling method 3");

method3();

System.out.println("Returned from method 3");

}

static void method3()

{

System.out.println("In method 3");

int a=20,b=0,c;

c=a/b;

System.out.println("method 3 exits");

}

}

**OUTPUT**

In method 1,calling method 2

In method 2,calling method 3

In method 3

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

at Ex1.method3(Ex1.java:23)

at Ex1.method2(Ex1.java:16)

at Ex1.method1(Ex1.java:10)

at Ex1.main(Ex1.java:5)

**Exception Hierarchy**

All exception and errors types are sub classes of class **Throwable**, which is base class of hierarchy.One branch is headed by **Exception**. This class is used for exceptional conditions that user programs should catch. NullPointerException is an example of such an exception.Another branch,**Error** are used by the Java run-time system([JVM](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/)) to indicate errors having to do with the run-time environment itself(JRE). StackOverflowError is an example of such an error.

[](https://media.geeksforgeeks.org/wp-content/uploads/Exception-in-java1.png)

**Exception Handling Techniques**

Java provides five keywords for Exception handeling: try,catch,throw,throws,finally.

**try-catch**

The try/catch block can be placed within any method that you can feel throw exception. All the statements to be tried for exception are put in a try block and immediately following the try is the catch block. catch block is used to catch any exception raised from try block. if exception occurs in any statement in try block, the following statements are not executed and control immediately passes to the corresponding catch block.

class Ex2

{

public static void main(String args[])

{

method1();

}

static void method1()

{

System.out.println("In method 1,calling method 2");

method2();

System.out.println("Returned from method 2");

}

static void method2()

{

System.out.println("In method 2,calling method 3");

try{

method3();

}

catch(Exception e)

{

System.out.println("Exception Handeled");

}

System.out.println("Returned from method 3");

}

static void method3()

{

System.out.println("In method 3");

int a=20,b=0,c;

c=a/b;

System.out.println("method 3 exits");

}

}

Output:

In method 1,calling method 2

In method 2,calling method 3

In method 3

Exception Handeled

Returned from method 3

Returned from method 2

**throw Keyword**

The throw keyword is used to explicitly throw an exception.

Whether implicit or explicit, objects of exception need to be created before they are thrown. throw is more useful when we want to throw a user-defined exception.

class Ex3

{

public static void main(String args[])

{

method1();

}

static void method1()

{

System.out.println("In method 1,calling method 2");

method2();

System.out.println("Returned from method 2");

}

static void method2()

{

System.out.println("In method 2,calling method 3");

try{

method3();

}

catch(Exception e)

{

System.out.println("Exception Handeled"+e);

}

System.out.println("Returned from method 3");

}

static void method3()

{

System.out.println("In method 3");

throw new ArithmeticException("Testing Throw");

}

}

Output:

In method 1,calling method 2

In method 2,calling method 3

In method 3

Exception Handeledjava.lang.ArithmeticException: Testing Throw

Returned from method 3

Returned from method 2

**throws**

The throws is added to the method signature to let the caller know about what exceptions the called method can throw. It is the responsibility of the caller to either handel the exception or it can also pass the exception. If all the methods in a program pass the exception to their callers, then ultimately the exception passes to the default exception handler. A method can throw more than one Exception, the exception list are separated by comma.

class Ex4

{

public static void main(String args[])

{

method1();

}

static void method1()

{

System.out.println("In method 1,calling method 2");

method2();

System.out.println("Returned from method 2");

}

static void method2()

{

System.out.println("In method 2,calling method 3");

try{

method3(4,0);

}

catch(Exception e)

{

System.out.println("Exception Handeled"+e);

}

System.out.println("Returned from method 3");

}

static void method3() throws Exception

{

System.out.println("In method 3");

if(b==0)

throw new ArithmeticException("Testing Throw");

}

}

Output:

In method 1,calling method 2

In method 2,calling method 3

In method 3

Exception Handeledjava.lang.ArithmeticException: Testing Throw

Returned from method 3

Returned from method 2

**finally block**

The finally block is always executed in try-catch-finally statements irrespective of whether an exception thrown from within the try/catch block or not.

**User defined exception in java**

In java we can create our own exception class and throw that exception using throw keyword. These exceptions are known as **user-defined** or **custom** exceptions.

import java.util.\*;

class Myexcep extends Exception

{

public String toString()

{

return " Uesr defined exception--odd no not allowed";

}

}

class Ex5

{

void msg(int a)throws Myexcep

{

if(a%2==0)

{

System.out.println("even");

}

else

{

throw new Myexcep();

}

}

public static void main(String args[])

{

int num;

Scanner sc=new Scanner(System.in);

System.out.println("enter any value");

num=sc.nextInt();

Ex5 e1=new Ex5();

try

{

e1.msg(num);

}

catch(Exception e)

{

System.out.println("exception handeled:"+e);

}

}

}

OUTPUT

enter any value

55

exception handeled: Uesr defined exception --odd no not allowed