Exception Handling

Exception:-In our program if any unexpected/unwanted event that interrupt the normal flow of the program is called exception.

Example:-

IOException
InterruptedException
InsufficientBalanceException

Exception Handling:-In our program there may be a chance of exception occurs highly recommended to handle the exception for normal or graceful termination of the program.

Exception Handling means we provide an alternative way to normal termination of the program it doesn't mean we repair the exception.

Reason of the Exception:-

A) provide wrong input
B) Access non-existence file
C) Access out-of-range values
D) Network connection problem.

Example:-

```
Class Test
{
Statement-1
Statement-2
Statement-3
Statement-4
Statement-5
.
.
.
Statement-20
Statement-21
Statement-22
Statement-23
}
```

Suppose an exception occurs in statement-5, from that statement onward statement will be not executed and the program will be terminated abnormally so highly recommended handle exceptions for graceful/normal termination of the program, and from statement-5 onward all statements will be executed.

Throwable class:- It is base class of exception hierarchy (it like Object class Object class is root/base class of all java class) its contains two child class Exception and Error.

- java.lang.Throwable
  - A. java.lang.Exception
  - B. java.lang.Error

Class hierarchy of Exception classes:- In generally exception is occurs by program and these are recoverable.

- Object
- Throwable
- Exception
  1. RuntimeException
     - A. IndexOutOfBoundsException
       - a) ArrayIndexOutOfBoundException
       - b) StringIndexOutOfBoundException
     - B. IllegalArgumentException
     - C. ArithmeticException
     - D. ClassCasteException
     - E. IllegalArgumentException
  2. IOException
     - a) EOFException
     - b) FileNotFoundException
     - c) InterruptionIOException
  3. SQLException
  4. ServletException

Class hierarchy of exception Error:- Most of the time error is generate due to lack of system resources not by program.

- Object
- Throwable
- Error
  1. java.lang.VirtualMachineError
     - a) OutOfMemoryError
     - b) StackOverflowError
  2. AssertionError
  3. LinkageError
  4. NoClassDefFoundError

Checked exception:-

The exception which is checked by the java compiler is called checked Exception.

Example:-
IOException
FileNotFoundException
SQLException
InterruptedException
ClassNotFoundException.....etc.

A CheckedException is child class of java.lang.Exception but not child class of java.lang.RuntimeException.

In our program there may be a chance of generating a Checked Exception we should compulsorily Handle it otherwise we will get compile time error.

We can handle checked exception by using try-catch and throws keywords.

UncheckedException:-

The exception which is not checked by the compiler is called UncheckedException

RuntimeException and its child classes, Error, and its child classes are considered CheckedException

CheckedException is not determined at the time of the method call it will determine at the method execution.

It represented a programming error. That will be generated from the inappropriate use of a piece of code.

Example:-ArithmeticException, NullPointerException, ArrayIndexOutOfBoundException, VirtualMachineErrro.

Note:-Every Exception either Checked or Unchecked it be generated at Runtime not compile time.

FullyCheckedException and PartialCheckedException:-

FullyCheckedException:-

The exception which are considered as Fully checked if and only if Base Exception class and its child are checked Exception class.

Example:-IOException,SQLException,InterruptedException.

PartialCheckedException:-

If base Exception class contain both types of exception class checked and unchecked such type of exception class called partialCheckedException.
Example:- Throwable, Exception,.....etc.

Error:-

The error is considered to be serious exceptional condition and it can't be directly controlled by our code.

An Error is a subclass of java.lang.Throwable.

An Error can be caught by an exception handler, but should not be handled.

Most of the time errors are not generated by programmer mistakes these are due to lack of system resources and these are non-recoverable.

Example:-VirtualMachineError,StackOverflowError,AssertionError
.....etc

Exception Handling Key word in java:-

- try
- catch
- finally
- throw
- throws

Exception handling by using try-catch block:-

Syntax of try-catch block:-

```
try
{
```

```
//risky code (exceptional code)
}
catch(Exception_name reference_variable)
{
//Exception handling code this code will be executed if exception is
occurs in try block.
}
```

Relation between try-catch block:-

Case1:- In this Example we do not handle the exception.

Example:-

```
class Test
{
public static void main(String[]args)
{
System.out.println("Hello");
System.out.println(10/0);//risky code
System.out.println("hi");
}
}
```

Result:- Hello

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
at Test.main(Test.java:6)
```

Case2:-

In This Example we handle the exception by using try-catch block.

Example:-

```
class Test
{
public static void main(String[]args)
{
System.out.println("Hello");
try
{
System.out.println(10/0);//risky code
}
catch(ArithmeticException e)
{
System.out.println(20/5);//exception handling code
}
System.out.println("hi");
}
```

```
}
Result:-Hello
4
Hi
```

Case4:-

Independent try block is not allowed we declare try block always with following combination. If we declare try block independent then we will get compile time error.

- try-catch
- try-finally
- try-catch-finally

Example:-

```
class Test
{
public static void main(String[]args)
{
try
{
System.out.println(10/2);
}
}
}
Result:- Test.java:5: error: 'try' without 'catch', 'finally' or
resource declarations
try
```

Exception Handling try with multiple catch block:-

It is not recommended to handle all types of exception with same catch block because it reduces the readability of the code.

Example:-

```
try
{
//Risky code.
}
catch(Exception e)
{
//Handling code
}
```

To overcome this problem highly recommended declaring try with multiple catch blocks. Handle different type of exception with different catch blocks.

Example:-

```
try
{
//Risky Code
}
catch(ArithmeticException ae)
{}
catch(NullPointerException ne)
{}
catch(ClassCasteException cce)
{}
```

Case1:-

When we declare multiple catch block with try then the order of catch block is very important we declare multiple catch child to parent otherwise we will get compile time error.

Example:-1 Declare multiple catch block and order of catch block is child to parent

```
class Test
{
public static void main(String[]args)
{
try
{
System.out.println(10/0);
}
catch(ArithmeticException ae)
{
System.out.println(ae+ "catch block will be executed");
}
catch(Exception e)
{
System.out.println(e+"catch block will be executed");
}
}
}
```

Result:- java.lang.ArithmeticException: / by zero catch block will be executed

Case2:-

We can't declare two catch block same type otherwise we will get compile time error.

Example:-

```
class Test
{
public static void main(String[]args)
{
try
{
System.out.println(10/0);
}
catch(ArithmeticException ae)
{
System.out.println(ae+"catch block will be executed");
}
catch(ArithmeticException e)
{
System.out.println(e+"catch block will be executed");
}
}
}
```

Result:- Test.java:13: error: exception ArithmeticException has already been caught catch(ArithmeticException e)

finally block:-

This block is always associate with try-catch independent finally perform not allowed.

we can use finally block to performed clean-up activity the resources which are open in try block.

finally block is always execute either exception raises or not in try block.

Syntax of finally block

```
try
{
//risky code
}
```

```
catch(Exception e)
{
//Exception handling code
}
finally
{
//CleanUp Code
}
```

Exmple:-

```
import java.util.*;
class Test
{
public static void main(String[]args)
{
Scanner sc=null;
try
{
sc=new Scanner(System.in);
System.out.println("Enter your Name");
String name=sc.next();
System.out.println("your name is= "+name);
}
catch(InputMismatchException me)
{
System.out.println("Plz Enter valid Name");
}
finally
{
sc.close();
System.out.println("try block open resources is close");
}
}
}
```

Result:-

```
Enter your Name
vikas
your name is= vikas
try block open resources is close
```

There are three methods to print Exception information:-

Throwable class define the following method to print exception information.

printStackTrace():-Name of exception: Description stack trace

toString():-Name of exception :Description

getMessage():-Description

Example:-

```
class Test
{
public static void main(String[]args)
{
try
{
System.out.println(10/0);
}
catch(ArithmeticException ae)
{
ae.printStackTrace();
System.out.println(ae.toString());
System.out.println(ae.getMessage());
}
}
}
```

Result:-

```
java.lang.ArithmeticException: / by zero
at Test.main(Test.java:7)
java.lang.ArithmeticException: / by zero
/ by zero
```

throw keyword:-In case default Exception handling exception object will by the method in which exception is raised and after creation exception object method handover that exception object to the JVM implicitly.

But sometimes based on our requirement we can create an exception object explicitly and we can handover to the JVM manually we perform this operation by using the throw keyword.

The main purpose of the throw keyword is to throw a user-defined exception or predefined exceptions explicitly.

Syntax of throw keyword:-

```
throw new Exception class(passed some argument);
```

Example:-In this program main method is create Exception object and handover to the jvm automatically.

```
class Test
{
public static void main(String[]args)
{
int []x=new int[3];
System.out.println(x[5]);
}
}
```

Result:- Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 5
at Test.main(Test.java:6)

Example:-In this program we create exception explicitly and hand over to the jvm manually.

```
class Test
{
public static void main(String[]args)
{
int []x=new int[3];
throw new ArrayIndexOutOfBoundsException(":5");
}
}
```

Result:- Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: :5
at Test.main(Test.java:6)

Case3:-

We can apply throw keyword only for throwable we can't throw normal java object otherwise we will get compile time error.

Example:-In this program throw keyword throw ArithmeticException object and that object is throwable type.

```
class Test
{
public static void main(String[]args)
{
throw new ArithmeticException();
}
}
```

Result:- Exception in thread "main" java.lang.ArithmeticException
at Test.main(Test.java:5)

Example:-In this program throw keyword throw Test object and that is not throwable type so we will get compile time error.

```
class Test
{
public static void main(String[]args)
{
throw new Test();
}
}
```

Result:- Test.java:5: error: incompatible types: Test cannot be converted to Throwable
throw new Test();

Example:-In this program throw keyword throw Test object and that is throwable type because Test class extends Throwable class.

```
class Test extends RuntimeException
{
public static void main(String[]args)
{
throw new Test();
}
}
```

Result:- Exception in thread "main" Test
at Test.main(Test.java:5)

User Defined/Customized Exception:-Based on our requirement sometimes we defined our own exception such types of exception are called user defined exception.

Example:-
InSufficientBalance
YouAreNotValidPerson
YorAreNotEligibleForVoting etc.

Example:-

```
class InsufficientBalance extends RuntimeException
{
InsufficientBalance(String s)
```

```
{
super(s);
}
}
class SufficientBalance extends RuntimeException
{
SufficientBalance(String s)
{
super(s);
}
}

class WithDraw
{
public void transation(int balance)
{
if(balance>1000)
{
throw new SufficientBalance("you can withdraw");
}
else
{
throw new InsufficientBalance("you can't withdraw");
}
}
public static void main(String[]args)
{
WithDraw wd=new WithDraw();
int balance =Integer.parseInt(args[0]);
wd.transation(balance);
}
}
```

Result:-

```
C:\Users\sony\Desktop>java WithDraw 1200
Exception in thread "main" SufficientBalance: you can withdraw
at WithDraw.transation(Test.java:22)
at WithDraw.main(Test.java:33)
```

throws Keyword:-

We used throws keyword to handle checked exception only there no any impact on unchecked exception.

In our program, there may be a chance of raises checked exception we should compulsorily handle it. By using try-catch or throws keyword otherwise we will get compile time error.

throws keyword we used only for convincing the compiler, By using of throws keyword we can't prevent abnormal termination of the program. We can use the throws keyword to delegate the responsibility of exception handling to the caller it may be method or JVM.

Example:-In this program delegate the responsibility of exception handling to JVM.

```
class Test
{
public static void main(String[]args)throws InterruptedException
{
Thread.sleep(20000);
}
}
```

Result:-compile and run fine

Example:-In this program m2() method delegates the responsibility of exception handling to caller method m1() and m1() method delegates the responsibility of exception handling to the main method and main() method delegate responsibility to JVM.

```
class Test
{
public void m2()throws InterruptedException
{
Thread.sleep(2000);
}
public void m1()throws InterruptedException
{
m2();
}
public static void main(String[]args)throws InterruptedException
{
Test t=new Test();
t.m1();
}
}
```

Result:-compile and run fine.

Case1:-

A method can throws more than one Exception simultaneously.

Example:-

```
import java.io.*;
class Test
{
public void m1()throws IOException,InterruptedException
{
FileInputStream fis=new FileInputStream("xyz.txt");
Thread.sleep(2000);
System.out.println("all are checked Exception");
}
public void m2()
{
try
{
m1();
}
catch(IOException ioe)
{
ioe.printStackTrace();
}
catch(InterruptedException ie)
{
ie.printStackTrace();
}
}
public static void main(String[]args)
{
Test t=new Test();
t.m2();
}
}
```

Result:-all are checked Exception

Case2:-

We can use throws keyword only on the method and the constructor but not on the class.

Case3:-In our program there may be a chance of raised check exception we should compulsorily handle it in this program throw the keyword throw Exception object and that is checked exception so should compulsorily handle otherwise we will get compile time error.

Example:-

```
class Test
{
public static void main(String[]args)
{
throw new Exception();
}
}
```

Result:- Test.java:5: error: unreported exception Exception; must be caught or declared to be thrown throw new Exception();

Case4:-

In this program throw keyword throw Error object and that is unchecked exception so we have not need to compulsory handle and code compile fine.

Example:-

```
class Test
{
public static void main(String[]args)
{
throw new Error();
}
}
```

Result:-we get runtime exception

```
Exception in thread "main" java.lang.Error
at Test.main(Test.java:5)
```

Common Exception and Error in Exception Handling:-

IndexOutOfBoundException:-When we try to access an invalid ArrayList position Then we will get IndexOutOfBoundException.

Example:-

```
import java.util.*;
class Test
{
public static void main(String[]args)
{
ArrayList<String> al=new ArrayList<String>();
al.add("vikas");
al.add("bishal");
System.out.println(al);
System.out.println(al.get(-1));
```

```
}
}
```
Result:- [vikas, bishal]
```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
-1
```
ArrayIndexOutOfBoundException:-It is child class of IndexOutOfBoundException when we try to access an invalid array position we get ArrayIndexOutOfBoundException .

Example:-
```
class Test
{
public static void main(String[]args)
{
int [] x={10,20,30,40,50};
System.out.println(x[6]);
}
}
```
Result:- Exception in thread "main"
```
java.lang.ArrayIndexOutOfBoundsException: 6
at Test.main(Test.java:6)
```

StringIndexOutOfBoundException:-It is child exception of IndexOutOfBoundException when we try to access an invalid String position we get StringIndexOutOfBoundException.

Example:-
```
class Test
{
public static void main(String[]args)
{
System.out.println("vikas".charAt(9));
}
}
```
Result:- Exception in thread "main"
```
java.lang.StringIndexOutOfBoundsException: String index out of
range: 9
at java.lang.String.charAt(Unknown Source)
at Test.main(Test.java:5)
```

NegativeArraySizeException:-When we assigning negative values size of the array we will get NegativeArraySizeException it unchecked Exception.

Example:-

```
class Test
{
public static void main(String[]args)
{
int [] x=new int[-9];
}
}
```

Result:- Exception in thread "main"
java.lang.NegativeArraySizeException
at Test.main(Test.java:5)

NullPointerException:-It is child class of RuntimeException.when we want performed any operation on null we get NullPointerException.

Example:-

```
class Test
{
public static void main(String[]args)
{
String s=null;
System.out.println(s.length());
}
}
```

Result:- Exception in thread "main" java.lang.NullPointerException
at Test.main(Test.java:6)

ArithmeticException:-In Integer arithmetic there is no chance of holding infinite and undefined value so if we got infinite or undefined result we get ArithmeticException

But floating Arithmetic can be hold infinite and undefined result. Infinite result we can represent by using +INFINITY (POSITIVE) and –INFINITY (NEGATIVE) constant and undefined result we can represent NaN constant only.

Exapmle:-

```
class Test
{
public static void main(String[]args)
{
System.out.println(10.10/0);
System.out.println(-10.10/0);
System.out.println(0.0/0);
System.out.println(-0.0/0);
```

```
System.out.println(10/0);
System.out.println(0/0);
}
}
```

Result:-

```
-Infinity
NaN
NaN
Exception in thread "main" java.lang.ArithmeticException: / by zero
at Test.main(Test.java:9)
```

IllegalArgument Exception:- when we call a method with illegal or inappropriate argument we get IllegalArgument Exception.

Example:-

```
class ThreadDemo
{
public static void main(String[]args)
{
Thread.currentThread().setPriority(12);
}
}
```

Result:- Exception in thread "main"

```
java.lang.IllegalArgumentException
at java.lang.Thread.setPriority(Unknown Source)
at ThreadDemo.main(Test.java:5)
```

IllegalThreadStateException:- whenever we are try to restart a thread we get IllegalThreadStateException.

Example:-

```
class MyThread extends Thread
{}
class ThreadDemo
{
public static void main(String[]args)
{
MyThread mt=new MyThread();
mt.start();
mt.start();
}
}
```

Result:- Exception in thread "main"

```
java.lang.IllegalThreadStateException
```

```
at java.lang.Thread.start(Unknown Source)
at ThreadDemo.main(Test.java:9)
```

NumberFormatException:- It is child class of IllegalArgumentException .It is unchecked exception raised by the programmer to indicate that we trying to convert string to number and String is not properly format.

Example:-

```
class Test
{
public static void main(String[]args)
{
int i=Integer.parseInt(args[0]);
System.out.println(i);
}
}
```

Result:-

```
java  Test ten
Exception in thread "main" java.lang.NumberFormatException: For input string: "ten".
```

ClassCasteException:- A Java ClassCasteException is an unchecked Exception that can be occurs when you try to improperly convert a class from one type to another.

Example:-

```
class Test
{
public static void main(String[]args)
{
Object obj=new Integer(100);
System.out.println((String)obj);
}
}
```

Result:-

```
Exception in thread "main" java.lang.ClassCastException:
java.lang.Integer cannot be cast to java.lang.String
at Test.main(Test.java:7)
```

InputMisMatchException:- It child class of RuntimeException when we provide input from keyboard is not defined type then we get InputMisMatchException.

Example:-

```
import java.util.*;
class Test
```

```
{
public static void main(String[]args)
{
Scanner sc=new Scanner(System.in);
System.out.println("Enter your Id");
int id=sc.nextInt();
System.out.println("your id is= "+id);
}
}
```

Result:- Enter your Id
vikas
Exception in thread "main" java.util.InputMismatchException

StackOverFlowError:-It child class of Error and hence it is unchecked raises automatically by jvm whenever we trying to perform recursive method call.

Example:-

```
class Test
{
public static void m1()
{
m2();
}
public static void m2()
{
m1();
}
public static void main(String[]args)
{
m1();
}
}
```

Result:-we got stackOverFlowError.

NoClassDefFounError:-It is child class of Error. It is unchecked exception raises automatically if jvm unable to find required .class file.

Example:-java Test ....press Enter If Test.class not present in specific directory we will get NoClassDefFoundError.

ExceptionInInitializerError:-It is child class of error hence it is unchecked exception raised automatically by jvm if any exception occur while executing of static variable assignment and static block.

Example:-

```
class Test
{
static int x=10/0;
public static void main(String[]args)
{
}
}
```

Result:- Exception in thread "main"
java.lang.ExceptionInInitializerError
Caused by: java.lang.ArithmeticException: / by zero

Assertion Error:-It is child class of error and hence it is unchecked raised explicitly by programmer to indicate that assert statement is fails.

Example:-

```
class Test
{
public static void main(String[]args)
{
int x=10;
assert(x>10);
System.out.println("Hello");
}
}
```

Result:- Exception in thread "main" java.lang.AssertionError
at Test.main(Test.java:6)