**Exception in java**

In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

**The exception handling** in java is a task to maintain normal flow of the program. It is an alternate way way to continue rest of the program normally.Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.

Types of Exception

Checked Exception

Unchecked Exception

**1) Checked Exception**

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

**2) Unchecked Exception**

The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

================================================================================================

**Problem without exception handling**

public class Testtrycatch1{

public static void main(String args[]){

int data=10/0;//may throw exception

String s=null;

System.out.println(s.length());//NullPointerException

String s="abc";

int i=Integer.parseInt(s);//NumberFormatException

int a[]=new int[5];

a[10]=50; //ArrayIndexOutOfBoundsException

System.out.println("rest of the code...");

}

}

-----------------------------------------------------------------------------------------------

**Java try-catch**

Java try block is used to enclose the code that might throw an exception. It must be used within the method.Java try block must be followed by either catch or finally block.

public class Testtrycatch2{

public static void main(String args[]){

try{

int data=50/0;

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

System.out.println("rest of the code...");

}

}

=============================================================================================

**Java Multi catch block**

If you have to perform different tasks at the occurrence of different Exceptions, use java multi catch block.

public class TestMultipleCatchBlock{

public static void main(String args[]){

try{

int a[]=new int[5];

a[5]=30/0;

}

catch(ArithmeticException e)

{

System.out.println("task1 is completed");

}

catch(ArrayIndexOutOfBoundsException e)

{

System.out.println("task 2 completed");

}

catch(Exception e)

{

System.out.println("common task completed");

}

System.out.println("rest of the code...");

}

}

==============================================================================================

**Rule1:** At a time only one Exception is occured and at a time only one catch block is executed.

**Rule2:** All catch blocks must be ordered from most specific to most general i.e. catch for ArithmeticException must come before catch for Exception .

==============================================================================================

class TestMultipleCatchBlock1{

public static void main(String args[]){

try{

int a[]=new int[5];

a[5]=30/0;

}

catch(Exception e){System.out.println("common task completed");}

catch(ArithmeticException e){System.out.println("task1 is completed");}

catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}

System.out.println("rest of the code...");

}

} //compile time error

================================================================================================

**Java finally block**

Java finally block is a block that is used to execute important code such as closing connection, stream etc.Java finally block is always executed whether exception is handled or not.

Java finally block follows try or catch block.

================================================================================================

**Case 1:** **where exception doesn't occur**

class TestFinallyBlock{

public static void main(String args[]){

try{

int data=25/5;

System.out.println(data);

}

catch(NullPointerException e)

{

System.out.println(e);

}

finally{

System.out.println("finally block is always executed");

}

System.out.println("rest of the code...");

}

}

================================================================================================

**Case 2 exception occurs and not handled.**

class TestFinallyBlock1{

public static void main(String args[]){

try{

int data=25/0;

System.out.println(data);

}

catch(NullPointerException e){

System.out.println(e);

}

finally{

System.out.println("finally block is always executed");

}

System.out.println("rest of the code...");

}

}

===============================================================================================

**Case 3 exception occurs and handled.**

public class TestFinallyBlock2{

public static void main(String args[]){

try{

int data=25/0;

System.out.println(data);

}

catch(ArithmeticException e){

System.out.println(e);

}

finally{

System.out.println("finally block is always executed");

}

System.out.println("rest of the code...");

}

}

=====================================================================================

**Java throw keyword**

The Java throw keyword is used to explicitly throw an exception.

public class TestThrow1{

static void validate(int age){

if(age<18)

throw new ArithmeticException("not valid");

else

System.out.println("welcome to vote");

}

public static void main(String args[]){

validate(13);

System.out.println("rest of the code...");

}

}

==============================================================================================