Practical - 7 exception Handling

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.

here are mainly two types of exceptions:

1. Checked Exception : checked at compile time e.g IOException, SQLException
2. Unchecked Exception ; Unchecked exceptions are not checked at compile-time, but they are checked at runtime. E.g ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException

Java Exception Keywords

try	The "try" keyword is used to specify a block where exception code are placed. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

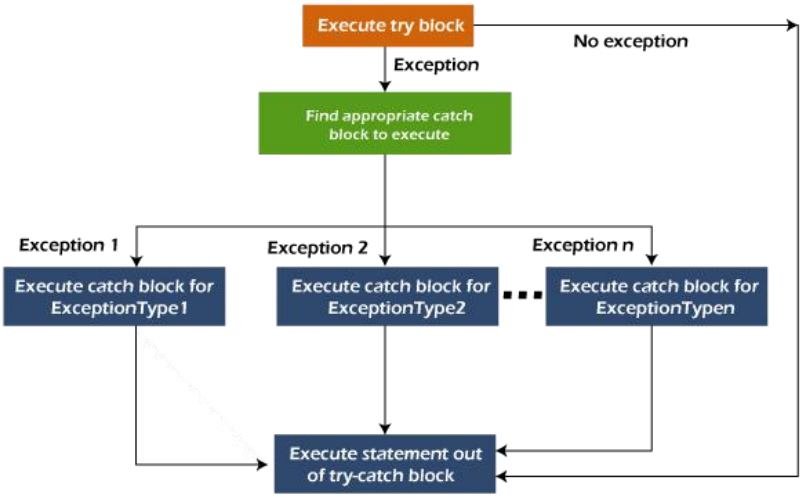
E.g 1

```
public class JavaExceptionExample{
  public static void main(String args[]){
   try{
      int data=100/0;
```

```
  }catch(ArithmeticException e){System.out.println(e);}
  System.out.println("rest of the code...");
 }
}
```

Out put : java.lang.ArithmeticException: / by zero
rest of the code…

- **Multiple catch**
- A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler.
- at a time only one exception occurs and at a time only one catch block is executed.
- In the case of multiple catch . Exception class catch(Exception e) must be the last exception otherwise it will generate compile time error.




```
public class MultipleCatchBlock1 {

   public static void main(String[] args) {

       try{
          int a[]=new int[5];
          a[6] =12;
           a[5]=30/0;
         }
         catch(ArithmeticException e)     {
            System.out.println("Arithmetic Exception occurs");
           }
         catch(ArrayIndexOutOfBoundsException e)
           {    System.out.println("ArrayIndexOutOfBounds Exception occurs");
           }
```

```
        catch(Exception e)
          {   System.out.println("Parent Exception occurs");
          }
        System.out.println("rest of the code");
    }
}
```

Output : ArrayIndexOutOfBounds Exception occurs
rest of the code

- **Finally Block**

  finally block in Java can be used to put "**cleanup**" code such as closing a file, closing connection.
  Finally block will execute even after the exception does not thrown.
  If program generate the exception which is not handled by the program even then after finally block is executed after the try block and then the program terminates abnormally.

e.g

```
class TestFinallyBlock {
  public static void main(String args[]){
  try{
   int data=25/5;
   System.out.println(data);
  }     catch(Exception e){
System.out.println(e);
}
 finally {
System.out.println("finally block is always executed");
}

System.out.println("rest of phe code...");
  }
}
```

Output :

5
Finally block is always execute
rest of code

E.g

```
public 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...");
    }
  }
```

Output :

finally block is always executed

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

at TestFinallyBlock1.main(TestFinallyBlock1.java:9)

- **Java throw Exception**

  The Java throw keyword is used to throw an exception explicitly. We can throw either checked or unchecked exceptions in Java by throw keyword. It is mainly used to throw a custom exception

  We can also define our own set of conditions and throw an exception explicitly using throw keyword.

  syntax

```
throw new exception_class("error message");
```

Java throws keyword is used in the method signature to declare an exception which might be thrown by the function while the execution of the code.

e.g

```
public class TestThrow1 {
  //function to check if person is eligible to vote or not
  public static void validate(int age) {
    if(age<18) {
      //throw Arithmetic exception if not eligible to vote
      throw new ArithmeticException("Person is not eligible to vote");
    }
    else {
      System.out.println("Person is eligible to vote!!");
    }
  }
  //main method
  public static void main(String args[]){
    //calling the function
    validate(13);
    System.out.println("rest of the code...");
  }
}
```

Output:

```
Exception in thread "main" java.lang.ArithmeticException: Person is not eligible to vote
        at TestThrow1.validate(TestThrow1.java:6)
        at TestThrow1.main(TestThrow1.java:15)
```

```
public class TestThrow1 {
  //function to check if person is eligible to vote or not
  public static void validate(int age) throws ArithmeticException {
    if(age<18) {
      //throw Arithmetic exception if not eligible to vote
      throw new ArithmeticException("Person is not eligible to vote");
    }
    else {
      System.out.println("Person is eligible to vote!!");
    }
  }
  //main method
  public static void main(String args[]){
    //calling the function
```

```
        validate(13);
        System.out.println("rest of the code...");
    }
   }
```

- **Userdefined Exception**

```
class UserDefinedException extends Exception
{
    public UserDefinedException(String str)
    {
        // Calling constructor of parent Exception
        super(str);
    }
}
// Class that uses above MyException
public class TestThrow2
{ static String str;
    public static void main(String args[])
    {
        try
        {
            if(str==null)
            // throw an object of user defined exception
            throw new UserDefinedException("This is user-defined exception");
        }
        catch (UserDefinedException ude)
        {
            System.out.println("Caught the exception");
            // Print the message from MyException object
            System.out.println(ude.getMessage());
            // getMessage() is exception class method
        }
    }
}
```

Output:

Caught the exception

This is user-defined exception

Exercise

1	Create a java program which takes the value(integer) from the command line arguments and store it into the variables a. perform b = 10 /a and generate ArithmeticException. Take the array of int . and write a code which will throw the

	exception ArrayIndexOutOfBoundsException.
2	Create userdefine exception called NumberException . which is will be handled by the when user input the negative number. (hint store the user input into variable , if variable value is negative then it should throw the user defined exception.
3	Write a Java program to create a class Bank which have property balance and deposit() and withdraw() methods. withdraw() throws an InsufficientFundsException exception if the amount to be withdrawn is more than the available balance. (Use throws keyword) create user defined InsufficientFundsException.
4	Create a user defined exception called “NoMatchException” that is thrown when a string is not equal to “Object Oriented Programing with JAVA”. Write a program that uses this exception. (Use throw keyword)