Lecture – 13 (Exception Handling)

Exception Handling in Java

The **exception handling in java** is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.

In this page, we will learn about java exception, its type and the difference between checked and unchecked exceptions.

What is exception handling

Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.

Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**. Exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;//exception occurs
statement 6;
statement 7;
statement 8;
statement 9;
statement 10;
```

Suppose there is 10 statements in your program and there occurs an exception at statement 5, rest of the code will not be executed i.e. statement 6 to 10 will not run. If we perform exception handling, rest of the statement will be executed. That is why we use exception handling in java.

Java try block

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.

Syntax of java try-catch

```
try{
//code that may throw exception
}catch(Exception_class_Name ref){
}
```

Syntax of try-finally block

```
try{
//code that may throw exception
}finally{
}
```

Problem without exception handling

Let's try to understand the problem if we don't use try-catch block.

```
public class Testtrycatch1{
  public static void main(String args[]){
     int data=50/0;//may throw exception
     System.out.println("rest of the code...");
}
}
```

Solution by exception handling

Let's see the solution of above problem by java try-catch 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 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.

Why use java finally

- Finally block in java can be used to put "cleanup" code such as closing a file, closing connection etc.

Let's see the java finally example where **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.

We can throw either checked or uncheked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

The syntax of java throw keyword is given below.

```
throw exception;
```

Let's see the example of throw IOException.

```
throw new IOException("sorry device error);
```

java throw keyword example

In this example, we have created the validate method that takes integer value as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote.

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

Java throws keyword

The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing checkup before the code being used.

Syntax of java throws

```
1. return_type method_name() throws exception_class_name{
2. //method code
3. }
```

Which exception should be declared

Ans) checked exception only, because:

- **unchecked Exception:** under your control so correct your code.
- **error:** beyond your control e.g. you are unable to do anything if there occurs VirtualMachineError or StackOverflowError.

Advantage of Java throws keyword

Now Checked Exception can be propagated (forwarded in call stack).

It provides information to the caller of the method about the exception.

Java throws example

Let's see the example of java throws clause which describes that checked exceptions can be propagated by throws keyword.

```
import java.io.IOException;
class Testthrows1{
```

```
  void m()throws IOException{
    throw new IOException("device error");//checked exception
  }
  void n()throws IOException{
    m();
  }
  void p(){
   try{
    n();
   }catch(Exception e){System.out.println("exception handled");}
  }
  public static void main(String args[]){
   Testthrows1 obj=new Testthrows1();
   obj.p();
   System.out.println("normal flow...");
  }
}
```