Object Oriented Programming

Exception Handling

Compiled By: Umair Yaqub

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.

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

Types of Exception: There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun Microsystems says there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

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.

3) Error: Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

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{}
```

Java catch block: Java catch block is used to handle the Exception. It must be used after the try block only.

You can use multiple catch block with a single try.

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...");
}
}
```

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

As displayed in the above example, rest of the code is not executed (in such case, rest of the code... statement is not printed). There can be 100 lines of code after exception. So all the code after exception will not be executed.

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...");
}
}
```

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

Example
The following is an array declared with 2 elements. Then the code tries to access the 3rd element of the array which throws an exception.

```
// File Name : ExcepTest.java
import java.io.*;

public class ExcepTest {

   public static void main(String args[]) {
      try {
         int a[] = new int[2];
         System.out.println("Access element three :" + a[3]);
      }catch(ArrayIndexOutOfBoundsException e) {
         System.out.println("Exception thrown  :" + e);
      }
      System.out.println("Out of the block");
   }
}
```

This will produce the following result −

Output

```
Exception thrown  :java.lang.ArrayIndexOutOfBoundsException: 3
Out of the block
```

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

Let's see a simple example of 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...");
 }
}
```

Output: task1 completed

```
       rest of the code...
```

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

Rule: 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...");
 }
}
```

Output:

```
Compile-time error
```