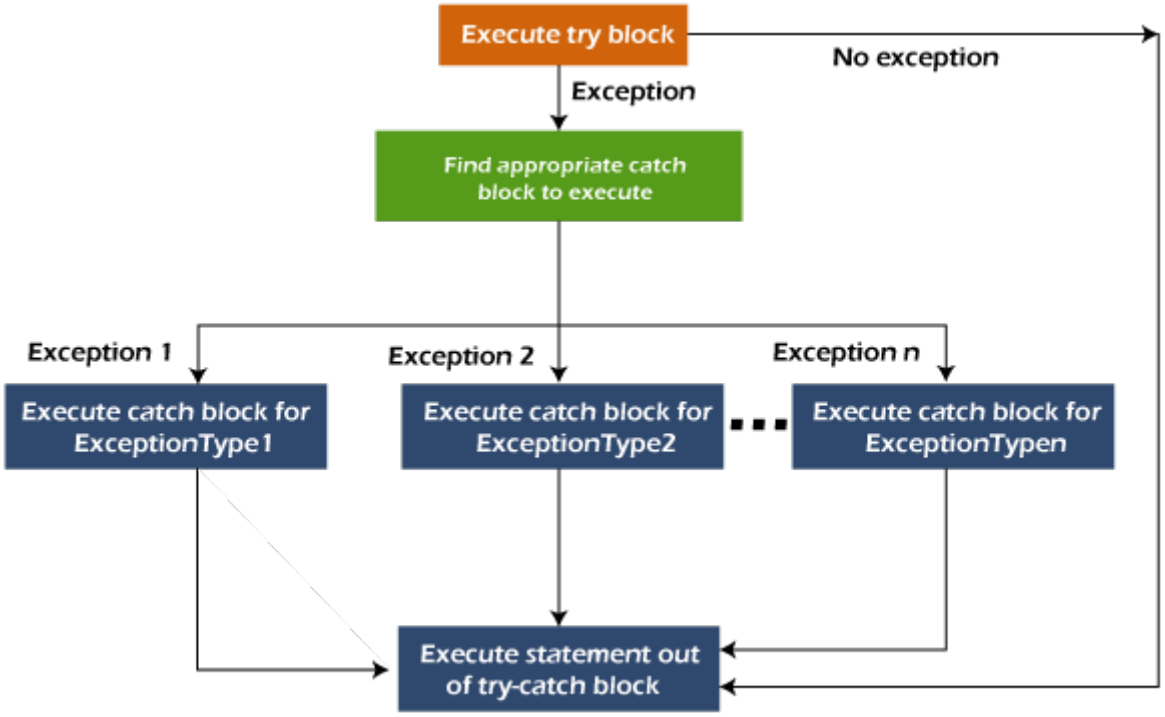
Day 27

Java Catch Multiple Exceptions
Java Multi-catch block

A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.
Points to remember

- At a time only one exception occurs and at a time only one catch block is executed.
- All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.

Flowchart of Multi-catch Block



Java Catch Multiple Exceptions
Example 1

Let's see a simple example of java multi-catch block.

```
public class MultipleCatchBlock1 {

    public static void main(String[] args) {

        try{
            int a[]=new int[5];
            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:

Arithmetic Exception occurs
rest of the code

Example 2

MultipleCatchBlock2.java

```
public class MultipleCatchBlock2 {

    public static void main(String[] args) {

        try{
            int a[]=new int[5];

            System.out.println(a[10]);
           }
           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

In this example, try block contains two exceptions. But at a time only one exception occurs and its corresponding catch block is executed.

MultipleCatchBlock3.java

```
public class MultipleCatchBlock3 {

    public static void main(String[] args) {

        try{
            int a[]=new int[5];
            a[5]=30/0;
            System.out.println(a[10]);
           }
           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:

Arithmetic Exception occurs
rest of the code

Example 4

In this example, we generate NullPointerException, but didn't provide the corresponding exception type. In such case, the catch block containing the parent exception class Exception will invoked.

MultipleCatchBlock4.java

```
public class MultipleCatchBlock4 {
```

```
    public static void main(String[] args) {

        try{
            String s=null;
            System.out.println(s.length());
           }
           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:

Parent Exception occurs
rest of the code

Example 5

Let's see an example, to handle the exception without maintaining the order of exceptions (i.e. from most specific to most general).

MultipleCatchBlock5.java

```
class MultipleCatchBlock5{
  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...");
 }
}
```

Test it Now

Output:

Java Nested try block

In Java, using a try block inside another try block is permitted. It is called as nested try block. Every statement that we enter a statement in try block, context of that exception is pushed onto the stack.

For example, the inner try block can be used to handle ArrayIndexOutOfBoundsException while the outer try block can handle the ArithemeticException (division by zero).
Why use nested try block

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.
Syntax:

....

```
//main try block
 try
 {
    statement 1;
    statement 2;
 //try catch block within another try block
    try
    {
       statement 3;
       statement 4;
 //try catch block within nested try block
       try
       {
          statement 5;
          statement 6;
     }
       catch(Exception e2)
       {
 //exception message
       }

    }
    catch(Exception e1)
    {
 //exception message
    }
 }
 //catch block of parent (outer) try block
 catch(Exception e3)
 {
 //exception message
 }
 ....
```

Java Nested try Example

Example 1

Let's see an example where we place a try block within another try block for two different exceptions.

NestedTryBlock.java

```
public class NestedTryBlock{
 public static void main(String args[]){
 //outer try block
  try{
  //inner try block 1
    try{
     System.out.println("going to divide by 0");
     int b =39/0;
    }
    //catch block of inner try block 1
    catch(ArithmeticException e)
    {
      System.out.println(e);
    }


    //inner try block 2
    try{
    int a[]=new int[5];

    //assigning the value out of array bounds
     a[5]=4;
     }

    //catch block of inner try block 2
    catch(ArrayIndexOutOfBoundsException e)
    {
       System.out.println(e);
    }


    System.out.println("other statement");
  }
  //catch block of outer try block
  catch(Exception e)
  {
    System.out.println("handled the exception (outer catch)");
  }

  System.out.println("normal flow..");
 }
}
```

Output:

Java Nested try block

When any try block does not have a catch block for a particular exception, then the catch block of the outer (parent) try block are checked for that exception, and if it matches, the catch block of outer try block is executed.

If none of the catch block specified in the code is unable to handle the exception, then the Java runtime system will handle the exception. Then it displays the system generated message for that exception.
Example 2

Let's consider the following example. Here the try block within nested try block (inner try block 2) do not handle the exception. The control is then transferred to its parent try block (inner try block 1). If it does not handle the exception, then the control is transferred to the main try block (outer try block) where the appropriate catch block handles the exception. It is termed as nesting.

NestedTryBlock.java

```
public class NestedTryBlock2 {

    public static void main(String args[])
    {
        // outer (main) try block
        try {

            //inner try block 1
            try {

                // inner try block 2
                try {
                    int arr[] = { 1, 2, 3, 4 };

                    //printing the array element out of its bounds
                    System.out.println(arr[10]);
                }

                // to handles ArithmeticException
                catch (ArithmeticException e) {
                    System.out.println("Arithmetic exception");
                    System.out.println(" inner try block 2");
                }
            }

            // to handle ArithmeticException
            catch (ArithmeticException e) {
                System.out.println("Arithmetic exception");
                System.out.println("inner try block 1");
            }
        }

        // to handle ArrayIndexOutOfBoundsException
        catch (ArrayIndexOutOfBoundsException e4) {
```

```
            System.out.print(e4);
            System.out.println(" outer (main) try block");
        }
        catch (Exception e5) {
            System.out.print("Exception");
            System.out.println(" handled in main try-block");
        }
    }
}
```

Output: