Exceptions in Java

Java try-catch block

Java try block

Java try block is used to enclose the code that might throw an exception. It must be used within the method.

If an exception occurs at the particular statement in the try block, the rest of the block code will not execute. So, it is recommended not to keep the code in try block that will not throw an exception.

Java try block must be followed by either catch or finally block.

Syntax of Java try-catch

```
1. try{
2. //code that may throw an exception
3. }catch(Exception_class_Name ref){}
```

Syntax of try-finally block

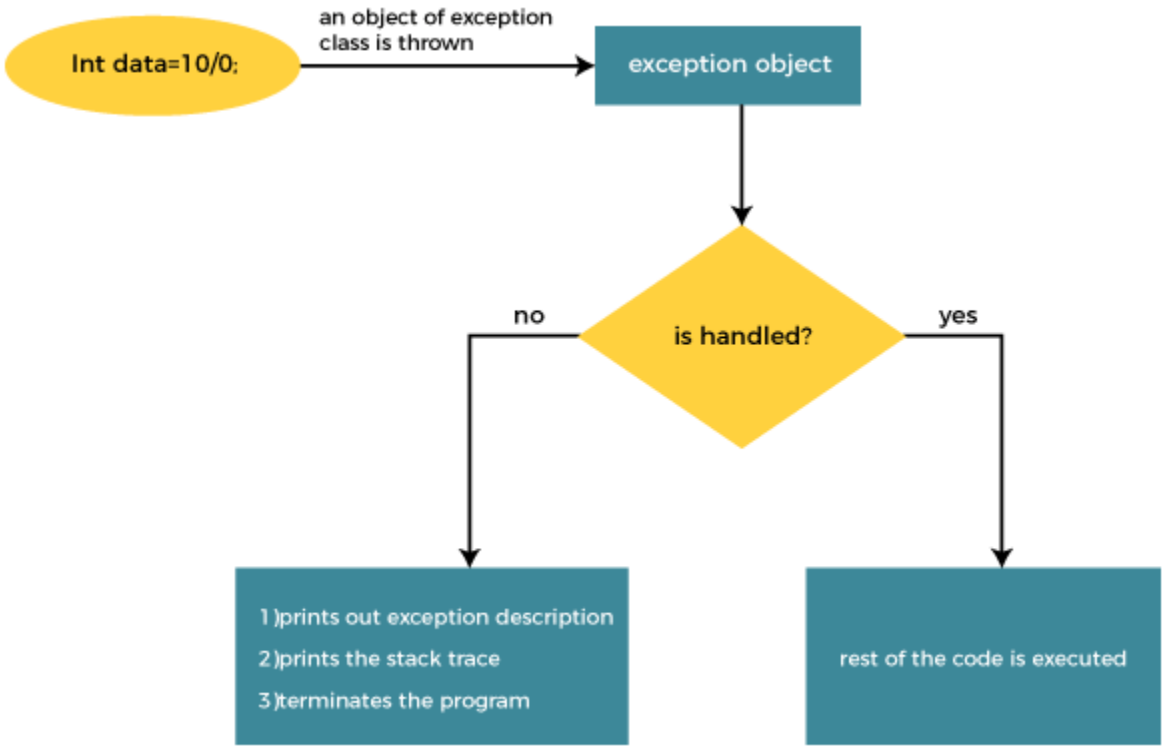
```
1. try{
2. //code that may throw an exception
3. }finally{}
```

Java catch block

Java catch block is used to handle the Exception by declaring the type of exception within the parameter. The declared exception must be the parent class exception (i.e., Exception) or the generated exception type. However, the good approach is to declare the generated type of exception.

The catch block must be used after the try block only. You can use multiple catch block with a single try block.

Internal Working of Java try-catch block



The JVM firstly checks whether the exception is handled or not. If exception is not handled, JVM provides a default exception handler that performs the following tasks:

- Prints out exception description.
- Prints the stack trace (Hierarchy of methods where the exception occurred).
- Causes the program to terminate.

But if the application programmer handles the exception, the normal flow of the application is maintained, i.e., rest of the code is executed.

Problem without exception handling

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

Example 1

TryCatchExample1.java

```
1. public class TryCatchExample1 {
2.
3.     public static void main(String[] args) {
4.
5.         int data=50/0; //may throw exception
6.
7.         System.out.println("rest of the code");
8.
9.     }
10.
11.}
```

Test it Now

Output:

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

As displayed in the above example, the **rest of the code** is not executed (in such case, the **rest of the code** statement is not printed).

There might be 100 lines of code after the exception. If the exception is not handled, all the code below the exception won't be executed.

Solution by exception handling

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

Example 2

TryCatchExample2.java

```
1. public class TryCatchExample2 {
2.
3.     public static void main(String[] args) {
4.         try
5.         {
6.         int data=50/0; //may throw exception
7.         }
```

```
8.            //handling the exception
9.        catch(ArithmeticException e)
10.       {
11.           System.out.println(e);
12.       }
13.       System.out.println("rest of the code");
14.   }
15.
16.}
```

Test it Now

Output:

```
java.lang.ArithmeticException: / by zero
rest of the code
```

As displayed in the above example, the **rest of the code** is executed, i.e., the **rest of the code** statement is printed.

Example 3

In this example, we also kept the code in a try block that will not throw an exception.

TryCatchExample3.java

```
1.  public class TryCatchExample3 {
2.
3.     public static void main(String[] args) {
4.        try
5.        {
6.        int data=50/0; //may throw exception
7.                    // if exception occurs, the remaining statement will not exceute
8.        System.out.println("rest of the code");
9.        }
10.           // handling the exception
11.       catch(ArithmeticException e)
12.       {
```

```
13.             System.out.println(e);
14.         }
15.
16.     }
17.
18.}
```

Test it Now

Output:

```
java.lang.ArithmeticException: / by zero
```

Here, we can see that if an exception occurs in the try block, the rest of the block code will not execute.

Example 4

Here, we handle the exception using the parent class exception.

TryCatchExample4.java

```
1.  public class TryCatchExample4 {
2.
3.      public static void main(String[] args) {
4.          try
5.          {
6.          int data=50/0; //may throw exception
7.          }
8.              // handling the exception by using Exception class
9.          catch(Exception e)
10.         {
11.             System.out.println(e);
12.         }
13.         System.out.println("rest of the code");
14.     }
15.
16.}
```

Test it Now

Output:

```
java.lang.ArithmeticException: / by zero
rest of the code
```

Example 5

Let's see an example to print a custom message on exception.

TryCatchExample5.java

```
1. public class TryCatchExample5 {
2.
3.     public static void main(String[] args) {
4.         try
5.         {
6.         int data=50/0; //may throw exception
7.         }
8.              // handling the exception
9.         catch(Exception e)
10.        {
11.                 // displaying the custom message
12.            System.out.println("Can't divided by zero");
13.        }
14.    }
15.
16.}
```

Test it Now

Output:

```
Can't divided by zero
```

Example 6

Let's see an example to resolve the exception in a catch block.

TryCatchExample6.java

```
1. public class TryCatchExample6 {
2.
3.     public static void main(String[] args) {
4.         int i=50;
5.         int j=0;
6.         int data;
7.         try
8.         {
9.         data=i/j; //may throw exception
10.        }
11.            // handling the exception
12.        catch(Exception e)
13.        {
14.             // resolving the exception in catch block
15.            System.out.println(i/(j+2));
16.        }
17.    }
18.}
```

Test it Now

Output:

```
25
```

Example 7

In this example, along with try block, we also enclose exception code in a catch block.

TryCatchExample7.java

```
1. public class TryCatchExample7 {
2.
3.     public static void main(String[] args) {
4.
```

```
5.        try
6.        {
7.        int data1=50/0; //may throw exception
8.
9.        }
10.            // handling the exception
11.       catch(Exception e)
12.       {
13.          // generating the exception in catch block
14.       int data2=50/0; //may throw exception
15.
16.       }
17.   System.out.println("rest of the code");
18.   }
19.}
```

Test it Now

Output:

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

Here, we can see that the catch block didn't contain the exception code. So, enclose exception code within a try block and use catch block only to handle the exceptions.

Example 8

In this example, we handle the generated exception (Arithmetic Exception) with a different type of exception class (ArrayIndexOutOfBoundsException).

TryCatchExample8.java

```
1.  public class TryCatchExample8 {
2.
3.     public static void main(String[] args) {
4.        try
5.        {
6.        int data=50/0; //may throw exception
```

```
7.
8.         }
9.             // try to handle the ArithmeticException using ArrayIndexOutOfBoundsExc
    eption
10.        catch(ArrayIndexOutOfBoundsException e)
11.        {
12.            System.out.println(e);
13.        }
14.        System.out.println("rest of the code");
15.    }
16.
17.}
```

Test it Now

Output:

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

Example 9

Let's see an example to handle another unchecked exception.

TryCatchExample9.java

```
1.  public class TryCatchExample9 {
2.
3.      public static void main(String[] args) {
4.          try
5.          {
6.          int arr[]= {1,3,5,7};
7.          System.out.println(arr[10]); //may throw exception
8.          }
9.              // handling the array exception
10.         catch(ArrayIndexOutOfBoundsException e)
11.         {
12.             System.out.println(e);
13.         }
```

```
14.         System.out.println("rest of the code");
15.     }
16.}
```

Test it Now

Output:

```
java.lang.ArrayIndexOutOfBoundsException: 10
rest of the code
```

Example 10

Let's see an example to handle checked exception.

TryCatchExample10.java

```
1.  import java.io.FileNotFoundException;
2.  import java.io.PrintWriter;
3.  public class TryCatchExample10 {
4.      public static void main(String[] args) {
5.          PrintWriter pw;
6.          try {
7.              pw = new PrintWriter("jtp.txt"); //may throw exception
8.              pw.println("saved");
9.          }
10.// providing the checked exception handler
11. catch (FileNotFoundException e) {
12.
13.             System.out.println(e);
14.         }
15.     System.out.println("File saved successfully");
16.     }
17.}
```

Test it Now

Output:

```
File saved successfully
```