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 Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc

Advantage of Exception Handling

The core advantage of exception handling is to maintain the normal flow of the application. An 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 are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.

Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

- Checked Exception
- Unchecked Exception

1) Checked Exception

The classes which directly inherit 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 which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

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 of try block, the rest of the block code will not execute. So, it is recommended not to keeping 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 block

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

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.

Syntax of catch block

```
catch(Exception_class_Name ref)
{
  //error handling code
```

}

Syntax of try-catch block

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

Problem without exception handling

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

Example 1

```
class Ex
{
    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, the rest of the code is not executed (in such case, the 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 the above problem by a java try-catch block.

Example 2

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

Output:

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

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

Common Java exception

- **ArithmeticException**
  If we divide any number by zero, there occurs an ArithmeticException

  Example

  ```
  int n=50/0;
  ```

- **ArrayIndexOutOfBoundsException**

If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below.

Example

```
int n[]=new int[5];
a[10]=50;   //ArrayIndexOutOfBoundsException
```

Example

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

```
class Ex
{
   public static void main(String[] args)
   {
      try
      {
      int arr[]= {1,3,5,7};
      System.out.println(arr[10]); //may throw exception
      }
         // handling the array exception
      catch(ArrayIndexOutOfBoundsException e)
      {
         System.out.println(“Array Index out of bounds");
      }
      System.out.println("rest of the code");
   }
}
```

Output:

Array Index out of bounds

rest of the code

- **NullPointerException**

If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.

Example

```
String s=null;

System.out.println(s.length());  //NullPointerException
```

- **NumberFormatException:** The wrong formatting of any value may occur NumberFormatException. Suppose I have a string variable that has characters, converting this variable into digit will occur NumberFormatException.

Example

```
String s="abc";

int i=Integer.parseInt(s);  //NumberFormatException
```

- **FileNotFoundException**
  t is used to attempt to access a nonexistent file.
- **IOException**
  It is used to I/O failures, such as inability to read from a file.
- **OutOfMemoryException**
  It is used to when there's not enough memory to allocate a new object.