TutorialKart

Java Exception Handling

Java Exception Handling

Typically, a Java application depends on multiple resources. Some of them are memory, file system, internet, etc. We would write code assuming that these resources are available all the time and in abundance. But, what if there is no enough memory, what if there is no file you are trying to read, what if the internet speed is so slow that it makes a timeout, etc. And these are not even the tip of iceberg. So, how do you handle these exceptional scenarios in your program? The answer is Java Exception Handling

Exception Handling in Java deals with how you can make your application ready to accept these exceptions, and act accordingly.

Java provides statements, tools and some checked exceptions, to handle exceptions.

Try Catch statement helps to catch any exceptions thrown by code, and make necessary decisions based on these exceptions. This helps prevent the abrupt termination of the Java program that is running.

Stack Trace is the information recorded by Java about which caused this exception.

Checked Exceptions are those that we can expect when we are using some methods, functions or operators.

We shall learn about all of these in detail, in this tutorial.

Java Try Catch

Consider that the method or operator you are using can make the runtime environment throw an exception. Now, we need to catch this exception and handle the due course of the program, so that the program may not end being stopped without executing completely.

We have the following program, where we are trying to access an element from array whose index is out of the range of array.

Java Program

```
public class Example {
    public static void main(String[] args) {
        int[] numbers = {2, 8, 9};
```

```
        int c = numbers[6];
        System.out.print(c);
    }
}
```

Output

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 6
    at Example.main(Example.java:4)
```

Java runtime has thrown an exception that the array index is out of bounds. And the execution is stopped at where the exception occurred. Rest of the program is not executed, which is a good thing, because we do not know what variable **c** has and what it could do to the subsequent course of program execution. But, we have to be ready to expect that such thing could occur when we are accessing an array element using index.

Let us put, try-catch statement into action. Surround that block of code which you would like execute, when no exception occurs, with **try**. Then follow up with **catch** block that catches this exception. In catch block, you have access to the exception object, and also you can decide what to happen when this exception occurs.

Java Program

```
public class Example {
    public static void main(String[] args) {
        int[] numbers = {2, 8, 9};
        int c = 0;
        try {
            c = numbers[6];
            System.out.print(c);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("You tried to access an element that is out of bounds for
            System.out.println("Continuing with c value of 1.");
            c = 1;
        }
        System.out.print(c);
    }
}
```

Output

```
You tried to access an element that is out of bounds for the array.
Continuing with c value of 1.
1
```

The program has not stopped when an exception occurred, but instead, it came to know what that exception is, and how to logically proceed further when this exception occurred.

More about Java Try Catch.

Java Stack Trace

When you catch an exception in catch block, you get access to exception object. This object contains lot of data and stack track is one of it.

Stack Trace contains information about which line of code triggered this exception, and the stack of calls made.

You may print this stack trace to console or a log file, and help fix the bugs during development or production.

Java Program

```
public class Example {
    public static void main(String[] args) {
        int[] numbers = {2, 8, 9};
        int c = 0;
        try {
            c = numbers[6];
            System.out.print(c);
        } catch (ArrayIndexOutOfBoundsException e) {
            e.printStackTrace();
        }
        System.out.print(c);
    }
}
```

Output

```
java.lang.ArrayIndexOutOfBoundsException: 6
    at Example.main(Example.java:6)
0
```

Java Exceptions And How to Handle these Exceptions

- Handle Java ArithmeticException.
- Handle Java ArrayIndexOutOfBoundsException.
- Handle Java ArrayStoreException.
- Handle Java NullPointerException.
- Handle Java NumberFormatException.
- Handle Java StringIndexOutOfBoundsException.