Programming with Java

Exception Handling in Java

Dictionary Meaning: Exception is an abnormal condition.

In java, an exception (or exceptional event) is a problem that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.

Common scenarios where exceptions may occur

– Scenario where ArithmeticException occurs: If we divide any number by zero, there occurs an ArithmeticException.

int a=50/0;

– Scenario where NullPointerException occurs: If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

String s=null; System.out.println(s.length());

– Scenario where ArrayIndexOutOfBoundsException occurs: If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:

int a[]=new int[5]; a[10]=50;

exceptions can also occur when:

- A user has entered an invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications or the JVM has run out of memory.

As can be seen from the examples above, some exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

There are three categories of exceptions in Java

1. **Checked exceptions** − A checked exception is an exception that is checked (notified) by the compiler at compilation-time, these are also called as compile time exceptions. These exceptions cannot simply be ignored, the programmer should take care of (handle) these exceptions.
2. **Unchecked exceptions** − An unchecked exception is an exception that occurs at the time of execution. These are also called as Runtime Exceptions. These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation.
3. **Errors** − These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

Java Exception Handling

A method in Java catches an exception using a combination of the **try** and **catch** keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as *protected code*, and the syntax for using try/catch looks like the following −

Syntax

```
try {
  // Protected code
} catch (ExceptionName e1) {
  // Catch block
}
```

The code which is prone to exceptions is placed in the try block. When an exception occurs, that exception occurred is handled by catch block associated with it. Every try block should be immediately followed either by a catch block or finally block.

A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in protected code, the catch block (or blocks) that follows the try is checked. If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block much as an argument is passed into a method parameter.

Example:

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

Output:

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

Example:

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

Multiple Catch Blocks

A try block can be followed by multiple catch blocks. The syntax for multiple catch blocks looks like the following –

Syntax

```
try {
   // Protected code
} catch (ExceptionType1 e1) {
   // Catch block
} catch (ExceptionType2 e2) {
   // Catch block
} catch (ExceptionType3 e3) {
   // Catch block
}
```

The previous statements demonstrate three catch blocks, but you can have any number of them after a single try. If an exception occurs in the protected code, the exception is thrown to the first catch block in the list. If the data type of the exception thrown matches ExceptionType1, it gets caught there. If not, the exception passes down to the second catch statement. This continues until the exception either is caught or falls through all catches, in which case the current method stops execution and the exception is thrown down to the previous method on the call stack.

The Throws/Throw Keywords

If a method does not handle a checked exception, the method must declare it using the throws keyword. The throws keyword appears at the end of a method's signature.

You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the throw keyword.

Try to understand the difference between throws and throw keywords, throws is used to postpone the handling of a checked exception and throw is used to invoke an exception explicitly.

The following method declares that it throws a RemoteException −

Example

```java
public class className {
   public void deposit(double amount) throws RemoteException {
      // Method implementation
      throw new RemoteException();
   }
   // Remainder of class definition
}
```

A method can declare that it throws more than one exception, in which case the exceptions are declared in a list separated by commas. For example, the following method declares that it throws a RemoteException and an InsufficientFundsException −

Example

```
public class className {
   public void withdraw(double amount) throws RemoteException,
      InsufficientFundsException {
      // Method implementation
   }
   // Remainder of class definition
}
```

The Finally Block

The **finally** block follows a try block or a catch block. A **finally** block of code always executes, irrespective of occurrence of an Exception.

Using a **finally** block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.

A finally block appears at the end of the catch blocks and has the following syntax:

try { // Protected code }

catch(ExceptionType1 e1) { // Catch block }

catch(ExceptionType2 e2) { // Catch block }

catch(ExceptionType3 e3) { // Catch block }

finally { // The finally block always executes. }

Example

```
public class ExcepTest {
public static void main(String args[]) {
      int a[] = new int[2];
      try {
            System.out.println("Access element three :" + a[3]);
      }catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("Exception thrown :" + e);
      }finally {
            a[0] = 6;
            System.out.println("First element value: " + a[0]);
            System.out.println("The finally statement is executed");
        }
 }
}
```

Output Exception thrown: java.lang.ArrayIndexOutOfBoundsException: 3 First element value: 6

The finally statement is executed