Java throw and throws

In this tutorial, we will learn to use throw and throws keyword for exception handling with the help of examples.

In Java, exceptions can be categorized into two types:

- **Unchecked Exceptions:** They are not checked at compile-time but at run-time.For example: `ArithmeticException`, `NullPointerException`, `ArrayIndexOutOfBoundsException`, exceptions under `Error` class, etc.
- **Checked Exceptions:** They are checked at compile-time. For example, `IOException`, `InterruptedException`, etc.

Usually, we don't need to handle unchecked exceptions. It's because unchecked exceptions occur due to programming errors. And, it is a good practice to correct them instead of handling them.

This tutorial will now focus on how to handle checked exceptions using `throw` and `throws`.

Java throws keyword

We use the `throws` keyword in the method declaration to declare the type of exceptions that might occur within it.

Its syntax is:

```
accessModifier returnType methodName() throws ExceptionType1, ExceptionType2 … {
  // code
}
```

As you can see from the above syntax, we can use `throws` to declare multiple exceptions.

Example 1: Java throws Keyword

```
import java.io.*;
class Main {
  public static void findFile() throws IOException {
    // code that may produce IOException
    File newFile=new File("test.txt");
    FileInputStream stream=new FileInputStream(newFile);
  }

  public static void main(String[] args) {
    try{
      findFile();
    } catch(IOException e){
      System.out.println(e);
    }
  }
}
```

Output

```
java.io.FileNotFoundException: test.txt (No such file or directory)
```

When we run this program, if the file `test.txt` does not exist, `FileInputStream` throws a `FileNotFoundException` which extends the `IOException` class.

If a method does not handle exceptions, the type of exceptions that may occur within it must be specified in the `throws` clause so that methods further up in the call stack can handle them or specify them using `throws` keyword themselves.

The `findFile()` method specifies that an `IOException` can be thrown. The `main()` method calls this method and handles the exception if it is thrown.

Throwing multiple exceptions

Here's how we can throw multiple exceptions using the `throws` keyword.

```
import java.io.*;
class Main {
  public static void findFile() throws NullPointerException, IOException, InvalidClassException {

    // code that may produce NullPointerException
    … … …

    // code that may produce IOException
    … … …

    // code that may produce InvalidClassException
    … … …
  }

  public static void main(String[] args) {
    try{
      findFile();
    } catch(IOException e1){
      System.out.println(e1.getMessage());
    } catch(InvalidClassException e2){
      System.out.println(e2.getMessage());
    }
  }
}
```

Here, the `findFile()` method specifies that it can throw `NullPointerException`, `IOException`, and `InvalidClassException` in its `throws` clause.

Note that we have not handled the `NullPointerException`. This is because it is an unchecked exception. It is not necessary to specify it in the `throws` clause and handle it.

---

throws keyword Vs. try...catch...finally

There might be several methods that can cause exceptions. Writing `try...catch` for each method will be tedious and code becomes long and less-readable.

`throws` is also useful when you have checked exception (an exception that must be handled) that you don't want to catch in your current method.

Java throw keyword

The `throw` keyword is used to explicitly throw a single exception.

When an exception is thrown, the flow of program execution transfers from the `try` block to the `catch` block. We use the `throw` keyword within a method.

Its syntax is:

```
throw throwableObject;
```

A throwable object is an instance of class `Throwable` or subclass of the `Throwable` class.

Example 2: Java throw keyword

```
class Main {
  public static void divideByZero() {
    throw new ArithmeticException("Trying to divide by 0");
  }

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

Output

```
Exception in thread "main" java.lang.ArithmeticException: Trying to divide by 0
    at Main.divideByZero(Main.java:3)
    at Main.main(Main.java:7)
exit status 1
```

In this example, we are explicitly throwing an `ArithmeticException.`

Note: `ArithmeticException` is an unchecked exception. It's usually not necessary to handle unchecked exceptions.

Example 3: Throwing checked exception

```
import java.io.*;
class Main {
  public static void findFile() throws IOException {
    throw new IOException("File not found");
  }

  public static void main(String[] args) {
    try {
      findFile();
      System.out.println("Rest of code in try block");
    } catch (IOException e) {
      System.out.println(e.getMessage());
    }
  }
}
```

Output

```
File not found
```

The `findFile()` method throws an `IOException` with the message we passed to its constructor.

Note that since it is a checked exception, we must specify it in the `throws` clause.

The methods that call this `findFile()` method need to either handle this exception or specify it using `throws` keyword themselves.

We have handled this exception in the `main` `()` method. The flow of program execution transfers from the `try` block to `catch` block when an exception is thrown. So, the rest of the code in the `try` block is skipped and statements in the `catch` block are executed.