Exception Handling

By Rahul Barve

Objectives

- Introduction to Exception Handling
- Need for Exception Handling
- A Simple Example
- Understand Exception Hierarchy
- Exception Types
- Exception Handling Keywords
- User Defined Exceptions

Exception Handling

By Rahul Barve

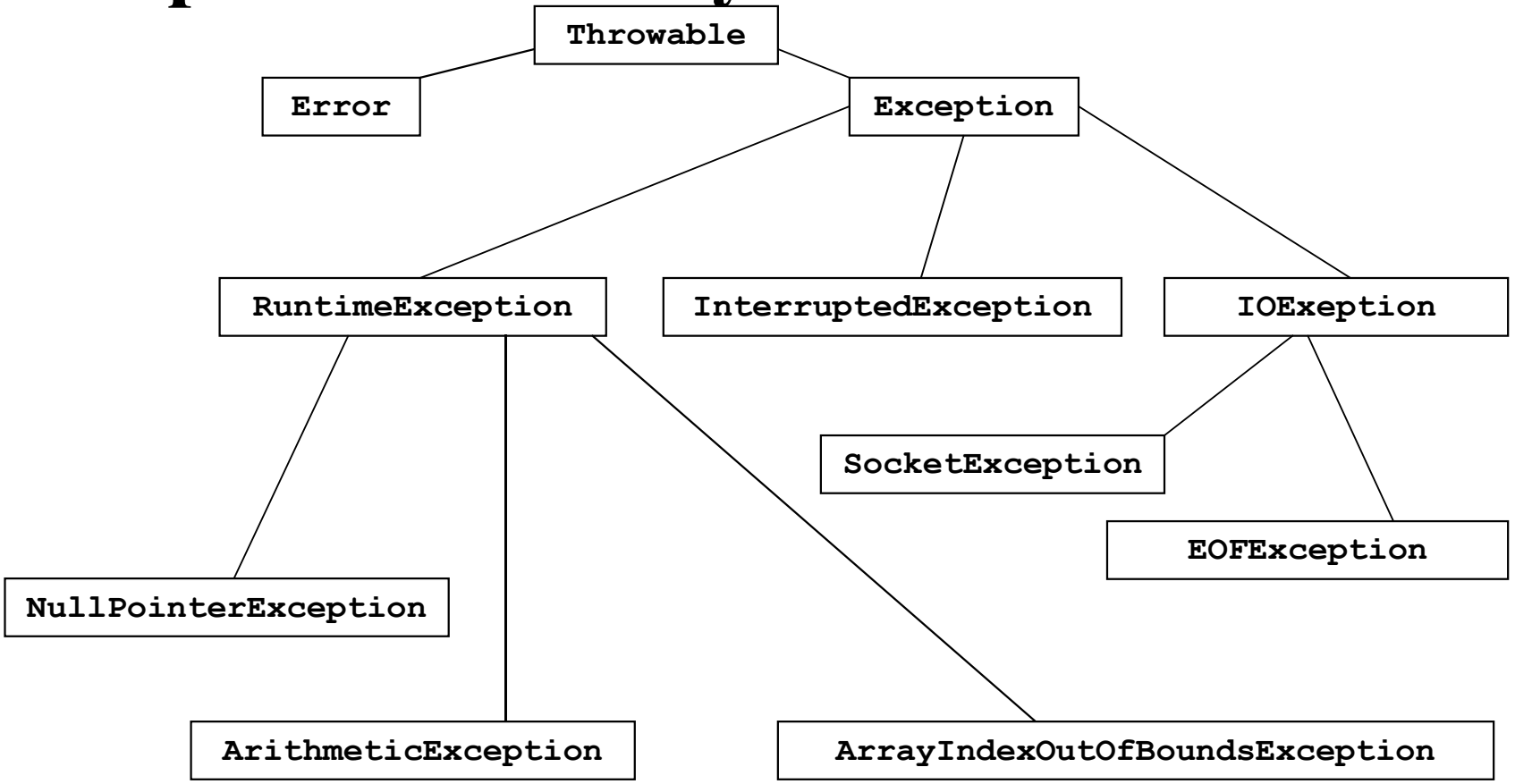
Exception Handling

- Exception Handling is an object oriented way of handling errors which occur during program's execution.
- Problem solving code is decoupled from error handling code and hence the program is less complex.

Exception Handling

- Exceptions in Java are actual objects.
- `Exception` objects encapsulate the error information.
- Exceptions are created when an abnormal situation occurs in a Java program.

Exception Hierarchy



By Rahul Barve

Exception Hierarchy

- The topmost class in the exception hierarchy is `Throwable`.
- It belongs to `java.lang` package.
- It includes all types of runtime errors and hence it is derived by `Error` and `Exception`.

Error

By Rahul Barve

Error

- `Error` indicates a runtime error which is not under the control of a developer.
- It describes resource exhaustion in JVM.

Error

- Rare and usually fatal.
- E.g.
  - `StackOverflowError`
  - `OutOfMemoryError`

Exception

By Rahul Barve

Exception

- `Exception` indicates a runtime error which is under the control of a developer.
- Frequent but not fatal.

Exception Types

Exception Types

- Exceptions are further divided into 2 types:
  - Unchecked Exceptions
  - Checked Exceptions

Unchecked Exceptions

By Rahul Barve

Unchecked Exceptions

- Unchecked Exceptions occur due to programming mistakes i.e. a non robust code.
- They are also called as Runtime Exceptions and hence expressed using a class `RuntimeException`.
- All classes descended from `RuntimeException` are runtime exceptions.

Unchecked Exceptions

- Include problems such as:
  - Bad cast
  - Out of bounds array access
  - A null pointer access

Unchecked Exceptions

- `NullPointerException`
- `ArrayIndexOutOfBoundsException`
- `ArithmeticException`

Checked Exceptions

By Rahul Barve

Checked Exceptions

- Occur due to problems in the environment settings.
- These exceptions are enforced by a compiler to be handled.
- These exceptions are expressed with the help of classes which are not descendants of `RuntimeException`.

Checked Exceptions

- Problems include such as:
  - Opening a file that does not exist.
  - Unable to load a class.

Checked Exceptions

- `FileNotFoundException`
- `ClassNotFoundException`

Handling Exceptions

By Rahul Barve

Handling Exceptions

- To handle the exceptions, it is necessary to enclose the statements, which are probable to fire an exception, within a `try` block.

Handling Exceptions

- E.g.

```
try {
    //Statement 1
    //Statement 2
}
```

Handling Exceptions

- If an exception is raised, it needs to be handled using an exception handler.
- This is done using a `catch` block.

Handling Exceptions

- E.g.

```
catch(<Exception Type> <ref-name){
    //Statements
}
```

Handling Multiple Exceptions

By Rahul Barve

Handling Multiple Exceptions

- If a block of code is capable for firing multiple exceptions, it is possible to handle them by providing multiple `catch` blocks.

Handling Multiple Exceptions

- E.g.

```
try {
  //Statements
}
catch(<exception type> <ref-name>) {
  //Statements
}
catch(<exception type> <ref-name>) {
  //Statements
}
```

Handling Multiple Exceptions

- When using multiple `catch` blocks, if the exception types represent parent-child relationship, then the `catch` block of sub type must appear before the `catch` block of super type.

Handling Multiple Exceptions

- It is also possible to handle multiple exceptions using a single `catch` block.
- This feature has been introduced by Java version 1.7.

Handling Multiple Exceptions

- E.g.

```
try {
  //Statements
}
catch(<ex 1> | <ex 2> <ref-name>) {
  //Statements
}
```

Handling Multiple Exceptions

- Since a single `catch` block is handling multiple exceptions, it is necessary to identify the type of the exceptions, so that different types of actions can be taken based upon the exception type.
- This is done by using `instanceof` operator.

`try` / `catch` Limitation

By Rahul Barve

`try` / `catch` Limitation

- Although `try` and `catch` blocks are useful to handle the exceptions, they have a common limitation.
- None of these give guarantee about the execution of the statements.

`try` / `catch` Limitation

- Sometimes, it becomes mandatory to execute the statements irrespective of whether the exception is fired or not.
- This is accomplished by using a `finally` block.

`finally`

- Statements enclosed within a `finally` block always execute.
- This is generally useful to perform clean-up operations.

finally

- E.g.

```
finally {
    //Statements
}
```

`finally`

- The `finally` block especially creates an impact for the methods of which the return type is other than `void`.

try-catch-finally Rules

By Rahul Barve

`try-catch-finally` Rules

- Every `try` block must be used in conjunction with either `catch`, `finally` or both.
- The blocks must appear one after the other without any statements in between.

`try-catch-finally` Rules

- `catch` block cannot appear without `try` block.
- `finally` block cannot appear without `try` block.

Using `throws`

Using `throws`

- If several methods of a class are probable to fire an exception, it becomes difficult to manage writing `try-catch` constructs in each method.
- This can be simplified by using `throws`.

Using `throws`

- Used by method and constructor definitions which may fire exceptions but not willing to handle.
- Instructs the compiler to enforce the calling program to handle the exception (Checked Exceptions only).

Using throws

```
public void readFile(String fileName) throws
FileNotFoundException {
     //Statements
}
public void openFile(String fileName) {
     readFile(fileName); //ERROR
}
```

Using `throws`

- If a called program method uses `throws` clause for a checked exception, then the calling program method, when using `throws` clause, must use a type which is equal or a super type.

Using `throws`

```
public class MyClass{
     public void testOne()
     throws Exception {…}
}
```

Using throws

```
public class YourClass {
        //ERROR
        public void testTwo() throws IOException{
                new MyClass().testOne();
        }
        //OK
        public void testThree() throws Exception{
                new MyClass().testOne();
        }
        //OK
        public void testFour() throws Throwable{
                new MyClass().testOne();
        }
}
```

Using `throws`

- In method overriding, using `throws` clause, an overridden method can widen the scope but cannot narrow it.

Using throws

```
public class Base {
      public void test1() throws Exception {…}
      public void test2() throws Exception {…}
      public void test3() throws Exception {…}
}
public class Derived extends Base {
      //OK
      public void test1() throws IOException{…}
      //OK
      public void test2() {…}
      //ERROR
      public void test3() throws Throwable{…}
}
```

Using `throw`

Using `throw`

- In most cases, `JRE` is responsible for firing an exception; but sometimes it might be necessary to fire an exception forcefully.
- This can be accomplished by using `throw` clause.

Using `throw`

- Syntax: `throw <Throwable>`
- E.g.

```
if(<condition>){
    Exception ex = new Exception();
    throw ex;
}
```

Using `throw`

- Sometimes, it becomes necessary to create a domain specific exception and throw it explicitly.
- Such exceptions are known as User Defined exceptions.

Using `throw`

- User defined exceptions are generally customized by creating a class that inherits either `Exception` or `RuntimeException`.
- E,g,

```
public class LowBalanceException extends Exception {…}
```

Using `throw`

- Once, a user defined exception class is created it can be used to raise an exception forcefully depending upon the condition.

Using throw

```
public  void  withdraw(float  amount)
throws LowBalanceException {
    if(balance < amount){
        String msg = "Low Balance!!";
        LowBalanceException lx =
        new LowBalanceException(msg);
        throw lx;
    }
}
```

Lets Summarize

- Introduction to Exception Handling
- Need for Exception Handling
- Exception Hierarchy
- Types of Exceptions
- Exception Handling Keywords
- User Defined Exceptions