JAVA

Exception Handling

By :

Subodh Sharma

Exception handling

"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".

"An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e at run time, that disrupts the normal flow of the program's instructions"

Error & Exception :

Error : An Error indicates serious problem that a reasonable application should not try to catch.

Exception: Exception indicates conditions that a reasonable application might try to catch

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.

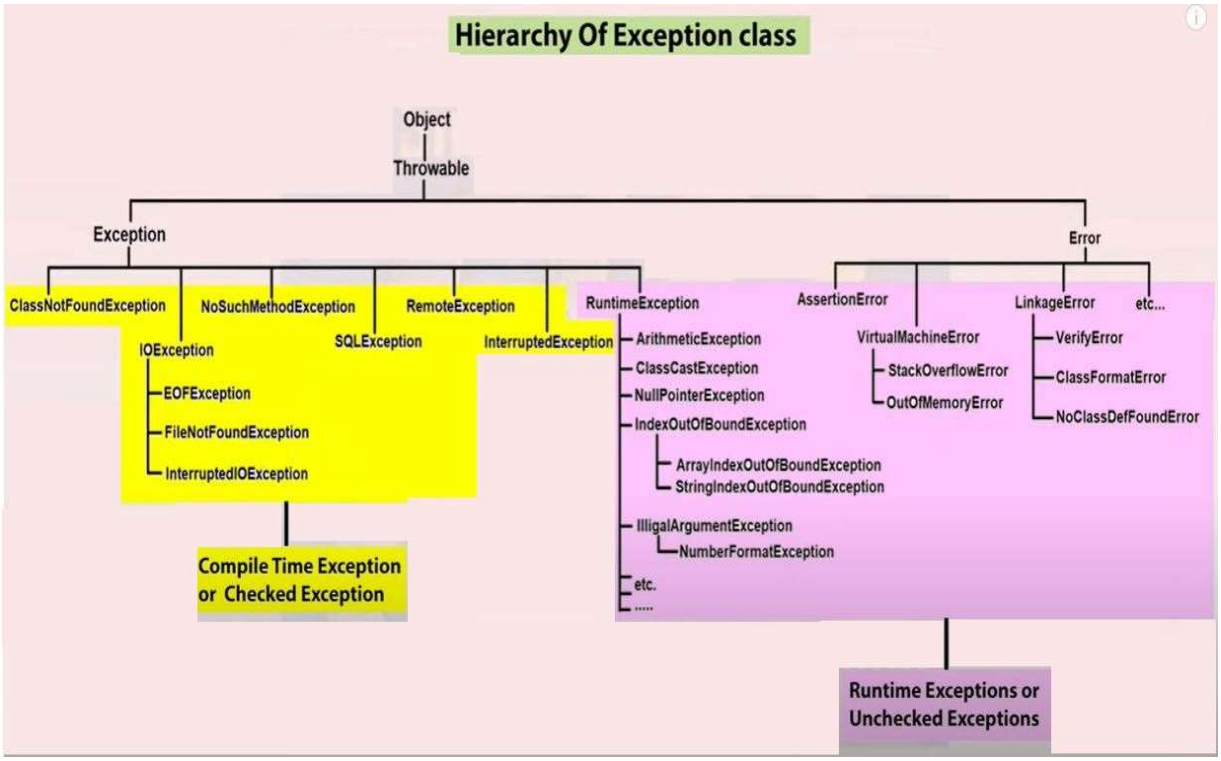
Exception Hierarchy

All exception and errors types are sub classes of class **Throwable**, which is base class of hierarchy.

One branch is headed by **Exception**. This class is used for exceptional conditions that user programs should catch. NullPointerException is an example of such an exception.

Another branch, **Error** are used by the Java run-time system(JVM) to indicate errors having to do with the run-time environment itself(JRE). StackOverflowError is an example of such an error.

Figure

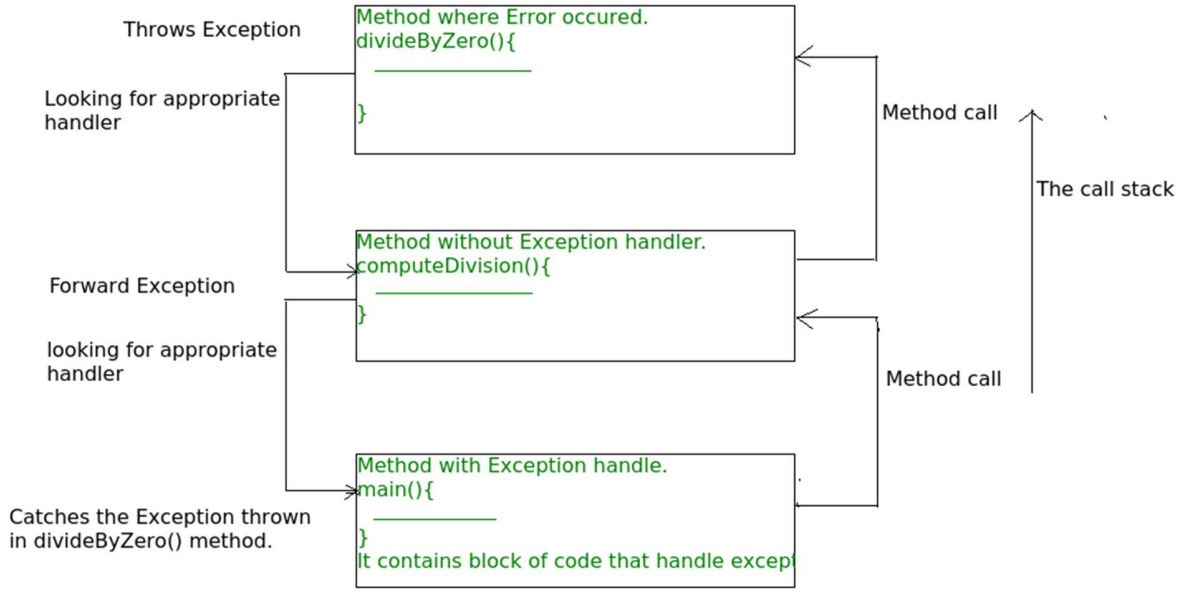


How JVM handle an Exception?

Default Exception Handling : Whenever inside a method, if an exception has occurred, the method creates an Object known as Exception Object and hands it off to the run-time system(JVM). The exception object contains name and description of the exception, and current state of the program where exception has occurred. Creating the Exception Object and handling it to the run-time system is called throwing an Exception. There might be the list of the methods that had been called to get to the method where exception was occurred. This ordered list of the methods is called **Call Stack**. Now the following procedure will happen.

- The run-time system searches the call stack to find the method that contains block of code that can handle the occurred exception. The block of the code is called **Exception handler**.
- The run-time system starts searching from the method in which exception occurred, proceeds through call stack in the reverse order in which methods were called.
- If it finds appropriate handler then it passes the occurred exception to it. Appropriate handler means the type of the exception object thrown matches the type of the exception object it can handle.
- If run-time system searches all the methods on call stack and couldn't have found the appropriate handler then run-time system handover the Exception Object to **default exception handler** , which is part of run-time system. This handler prints the exception information in the following format and terminates program **abnormally**.

figure



The call stack and searching the call stack for exception handler.

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:

1. Checked Exception
2. Unchecked Exception
3. Error

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.

Checked: are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using *throws* keyword.

For example, consider the following Java program that opens file at location "C:\test\a.txt" and prints the first three lines of it. The program doesn't compile, because the function main() uses FileReader() and FileReader() throws a checked exception *FileNotFoundException*. It also uses readLine() and close() methods, and these methods also throw checked exception *IOException*

```
import java.io.*;
class Main {
    public static void main(String[] args) {
        FileReader file = new FileReader("C:\\test\\a.txt");
        BufferedReader fileInput = new BufferedReader(file);

        // Print first 3 lines of file "C:\test\a.txt"
        for (int counter = 0; counter < 3; counter++)
            System.out.println(fileInput.readLine());

        fileInput.close();
    }
}
```

Output:

```
Exception in thread "main" java.lang.RuntimeException: Uncompilable source code -
unreported exception java.io.FileNotFoundException; must be caught or declared to be
thrown
    at Main.main(Main.java:5)
```

To fix the above program, we either need to specify list of exceptions using throws, or we need to use try-catch block. We have used throws in the below program. Since *FileNotFoundException* is a subclass of *IOException*, we can just specify *IOException* in the throws list and make the above program compiler-error-free.

```
import java.io.*;
class Main {
    public static void main(String[] args) throws IOException {
        FileReader file = new FileReader("C:\\test\\a.txt");
        BufferedReader fileInput = new BufferedReader(file);

        // Print first 3 lines of file "C:\test\a.txt"
        for (int counter = 0; counter < 3; counter++)
            System.out.println(fileInput.readLine());
        fileInput.close();
} }
```

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.

Unchecked are the exceptions that are not checked at compiled time.

3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

In Java exceptions under *Error* and *RuntimeException* classes are unchecked exceptions, everything else under throwable is checked.

Java Exception Keywords

There are 5 keywords which are used in handling exceptions in Java.

Keyword	Description
Try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

Java Exception Handling Example

Let's see an example of Java Exception Handling where we using a try-catch statement to handle the exception.

Note : The following example is for try, catch, finally, multiple catch :

Example :

```
class Ex1
{    public static void main(String s[])
   {
    try {
         int a= Integer.parseInt(s[1]);
         int b= Integer.parseInt(s[2]);
        int ans = a/b;
            System.out.println(s[0]+" of "+ a+"/"+b+" = "+ans);
       }
    catch(ArithmeticException e)
       {      System.out.println("Arithmetic Exce");      }
    catch(ArrayIndexOutOfBoundsException e)
      {        System.out.println("Array Index");          }
    catch(Exception e)
     {   System.out.println("Something else:");
          System.out.println(e);       }
  finally {
   System.out.println("closed properly");
         }
 }}
```

throw keyward :

In the above program, we have only been catching exceptions that are thrown by the Java Runtime system. However, it is possible for us to throw an exception explicitly using throw keyword. The general form is :

throw ThrowableInstance;

Here throwableInstance must be an object of type Throwable or a subclass of Throwable. (Note that int, char, double, String … cannot be used as exception.)

Example:

```
class Ex5
{
 public static void main(String s[])
 {
 try {
      System.out.println("To see demo:");
      System.out.println("Write on command line :");
      System.out.println("java Ex5 show\n");

      if (s[0].equals("show"))
        {        System.out.println("Creating and throwing object");
                 System.out.println("of ArithmeticException Class\n");
                 throw new ArithmeticException();
        }
    }
catch(ArithmeticException e)
       {
    System.out.println("object is received");
    System.out.println("in catch block\n");
       }

finally {     System.out.println("finally block ");
         }
}
```

throws keyward :

if a method is capable of causing an exception that it does not handle, it must specify this behavior so that callers of the method can guard themselves against that exception. We do this by including a throws clause in the method's declaration. A throws clause lists the types of exceptions that a method might throw. This is necessary for all exceptions, **except** of Error or RuntimeException or any of their sub classes..

```
type method_name(parameter list) throws exception-list
        {
                //body of method.
        }
```

Example :

Checked: are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then **the method must either handle the exception or it must specify the exception using *throws* keyword.**

```
import java.io.*;
class Main {
    public static void main(String[] args) throws IOException {
        FileReader file = new FileReader("C:\\test\\a.txt");
        BufferedReader fileInput = new BufferedReader(file);

        // Print first 3 lines of file "C:\test\a.txt"
        for (int counter = 0; counter < 3; counter++)
            System.out.println(fileInput.readLine());
        fileInput.close();
} }
```

Creating your Own Exception Subclasses :

Although Java built-in exceptions handle most common errors, but we can create our own exception types to handle situations specific to our applications. Exception class does not define any method of its own, actually, it inherit all methods provided by Throwable. We may also wish to override one or more of these methods.

Example : Here we create MyAge Exception class (sub class of Exception), in which we override the method toString().

```
class MyAge extends Exception
{
  private int age;
   MyAge(int a)
    {
      age=a;
    }
  public String toString()
   {
    return "UnderAge {"+age+"}";
   }
}
```

```
class Election
{

 static void vote(int ag) throws MyAge
  {
    if (ag<18)
      throw new MyAge(ag);
     else
    System.out.println("..Voted successful.. ");
  }

 public static void main(String s[])
 {
    String name=s[0];
    int myage= Integer.parseInt(s[1]);
 try {
     System.out.println(name+":"+" Age="+myage);
     vote(myage);
    }
    catch(MyAge e)
    {
     e.printStackTrace();
    }
 }
}
```

Java's Built-in Exception :

Inside the standard package java.lang. Java defines several exception classes. The most general of these exceptions are sub classes of the standard type RuntimeException. Since Java.lang is implicityly imported into all java program. So most exceptions derived from RunTimeException are automatically available. (These are called unchecked exceptions). The list of these are :

List of Unchecked RuntimeExceptions Subclasses :

Arithmetic Exception	Arithmetic error, such as divide by zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bound.
ArrayStoredException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid casting
NullPointerException	Invalid use of a null reference.
NumberFormtException	Invalid conversion of a string to a numeric format

List of Checked Exceptions Defined in java.lang :

ClassNotFoundException
NoSuchFieldException
NoSuchMethodException

Java Exceptions Index

1. Java Try-Catch Block

2. Java Multiple Catch Block
3. Java Nested Try
4. Java Finally Block
5. Java Throw Keyword
6. Java Exception Propagation
7. Java Throws Keyword
8. Java Throw vs Throws
9. Java Final vs Finally vs Finalize
10. Java Exception Handling with Method Overriding
11. Java Custom Exceptions