Exception Handling in Java

Authored by : Sangeeta Joshi ***Presented by: Sangeeta Joshi***

Agenda

- **Exception**
- What are exceptions?
- Differerence between Exception and Errors.
- Exception Class Hierarchy.
- Checked and Unchecked Exceptions.
- **Exception Handling**
- Try
- Catch
- Finally
- Throw
- Throws
- **User Defined Exception**

What is Exception ?

- Exceptions are nothing but some anomalous conditions that occur during the execution of the program.

- Exceptions are the conditions or typically an event which may either cause a running program to terminate or change its normal flow of execution

Hierarchy of Exception Classes

- Throwable is the super class in Exception hierarchy
- It is in java.lang package.
- The Throwable class is further divided into two subclasses :-

Error class

Errors :

- These are the situations which can not be handled or can not be recovered from.
- If any such situation occurs, program will terminate.
- Those situations are rare & usually fatal

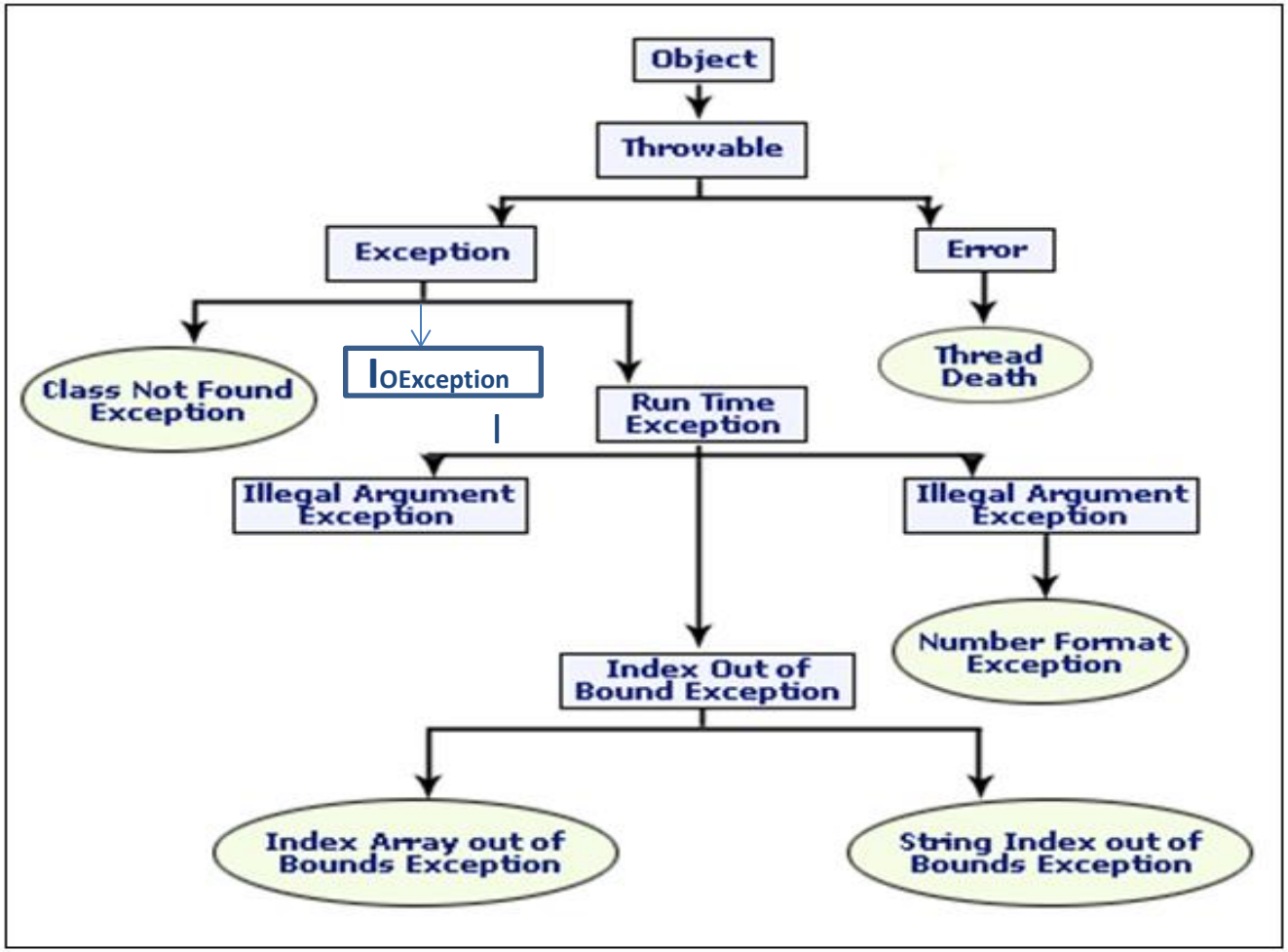
Example: *OutOfMemoryError*
some internal error in JVM

Exception class

Exceptions:

- These are the situations which can be handled.
- It is possible to recover from such situations & continue with program execution further.
- However normal flow of execution may change in such situations.

Exception Hierarchy



Exception Hierarchy

Checked Exception

- Checked Exceptions: Those are the exceptions where compiler will ensure that programmer has handled those situations.
- The code capable of generating such kind of exception has to be embedded in Try- catch block or compiler will complain (or method where this code resides should declare it in "throws" clause)
- Example : *IOException , SqlException*

Unchecked Exception

Unchecked Exceptions:

- Compiler will not check whether programmer has handled those situations or not.
- Example: all RunTimeExceptions

Exception Handling

- The various keywords for handling exceptions are below.
  - try
  - catch
  - finally
  - throw
  - throws

- The three exception handler components are used to catch and handle the exceptions. These are try, catch and finally clause.

Syntax for Exception handling

- Using try and catch:
- The syntax for the usage of try, catch and finally block is given below.

```
try {
        .....
    }
catch(<exceptionclass1> <obj1>)
    {
        .....
    }
finally{
        .....
    }
```

Try Block

- The code which is capable of generating some kind of exception (Checked Exception) must be written in Try block.
- Try block should be immediately followed by either a catch or finally block.
- A try block can have multiple catch blocks

Catch Block

The Catch Block is used as exceptional-handler. The Exception that arises in the try block is handled by the Catch -Block.

What is the problem with following code?

```
try {
        ------
        ------
        ------
    }
catch (RuntimeException re){  ---}
catch(NullPointerException  ne){---}
catch (Exception e) {-----}
catch (Throwable t){----}
```

Finally block

- Finally block gets executed in either of the cases :

*if Exception **occurs** or it **does not occur***

advantage : write clean up code in "finally"

How to Throw Exceptions in Java

Whenever an exceptional situation occurs, an object of that particular Exception is created & thrown.

We can also create an Exception object & explicitly throw it using keyword *"throw"*

Example:

```
try{  if(  ...some condition)
        throw new Exception();
         -----
      }
 catch(Exception e)
    {
          .......some handling code  here
     }
```

Important points

- If the method throwing a checked exception does not catch it then method must declare it as throws that exception
- Exceptions can cascade from a method to its caller & so on till main ()method throws that exception
- An overridden method in subclass can not throw more checked exception than the original method in super class

User Defined Exception

Sometimes you may need to handle certain situations which are specific to your application.

In such cases, you can create your own User Defined Exceptions.

Example:

```
class InsufficientBalanceException extends Exception
{
    ------------
    ------------
}
```

User Defined Exception

```
public void withdraw(int amt)
{
    try{
            if (balance < amt)
             throw new InsufficientBalanceException();
             ----
        }
    catch(InsufficientBalanceException e)
        {    handling code// some msg…}
}
```

Assertion

An *assertion* is a statement in the Java™ programming language that enables you to test your assumptions about your program.

Assertion facility is added in J2SE 1.4. In order to support this facility J2SE 1.4 added the keyword assert to the language, and AssertionError class. An assertion checks a boolean-typed expression that must be true during program runtime execution. The assertion facility can be enabled or disabled at runtime.

Assertion (contd)

Declaring Assertion

Assertion statements have two forms as given below

assert expression;

assert expression1 : expression2;

Assertion (contd)

Enable/Disable Assertions

By default assertion are disabled in Java runtime environment. The argument –eanbleassertion or –ea will enables assertion, while –disableassertion or –da will disable assertions at runtime.

The following command will run AssertionDemo with assertion enabled.

Java –ea AssertionDemo

or

Java –enableassertion AssertionDemo

Assignments

Any Questions?