Bootcamp Training

Java Spring AWS Frontend Training

SummitWorks™
GLOBAL SOLUTION ARCHITECTS

Authorized & published by Summitworks Technologies Inc

Agenda: Day -3

- Exceptions
  - Exception Hierarchy
    - Built-in Exceptions
    - Exceptions Methods
  - Try block and Catching Exceptions
  - Multiple Catch Blocks
  - Catching Multiple Type of Exceptions
  - The Throws/Throw Keywords
  - The Finally Block
  - The try-with-resources
  - User-defined Exceptions and Common Exceptions

Exception

- are usually used to denote something unusual that does not conform to the standard rules.
- In programming, exceptions are events that arise due to the occurrence of unexpected behaviour in certain statements, disrupting the normal execution of a program.

Causes of Exception

- Exceptions can arise due to a number of situations. For example,
  - Trying to access the 11th element of an array when the array contains of only 10 element (*ArrayIndexOutOfBoundsException*)
  - Division by zero (*ArithmeticException*)
  - Accessing a file which is not present (*FileNotFoundException*)
  - Failure of I/O operations (*IOException*)
  - Illegal usage of null. (*NullPointerException*)

Types of Exception

- **Checked exceptions**: which are checked by the compiler at compile time is called checked exception for execution of the app

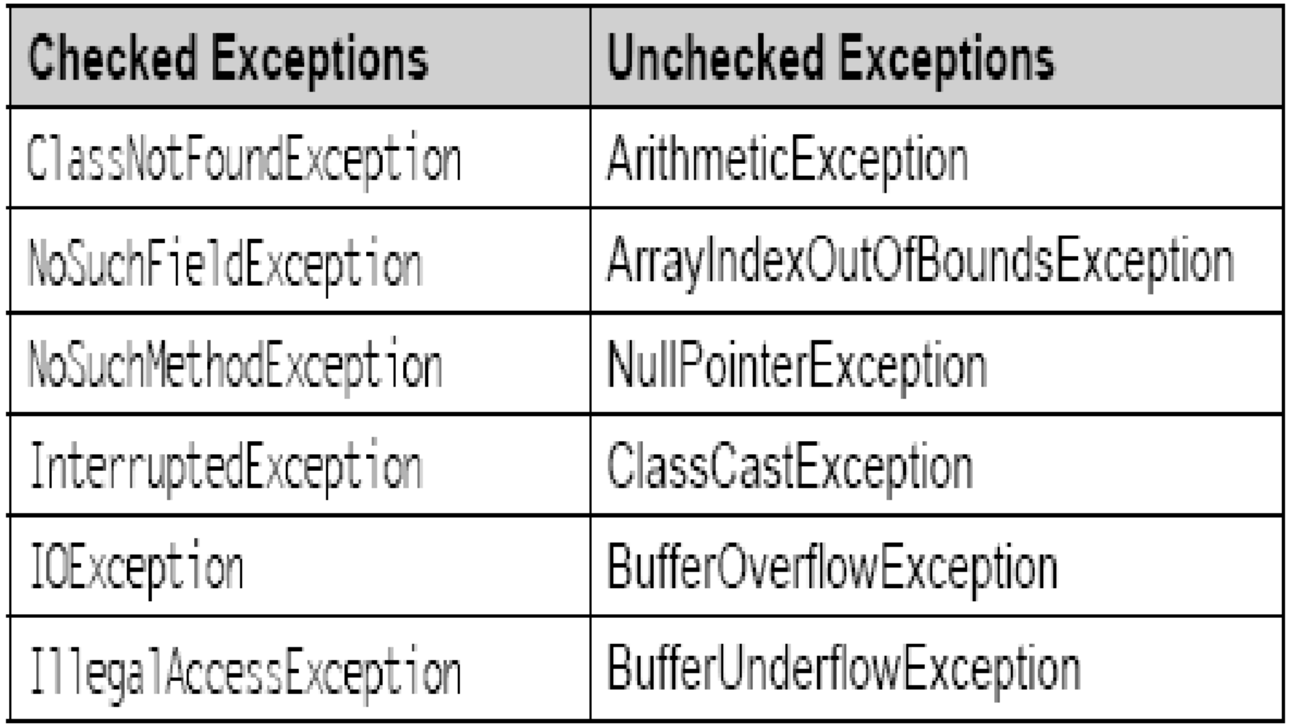
  Ex: IOException, SQLException
- **Unchecked exceptions:** checked by the JVM at run time

  Ex: ArrayIndexOutOfBoundsException

  NullPointerException
- **Error**:Error is irrecoverable

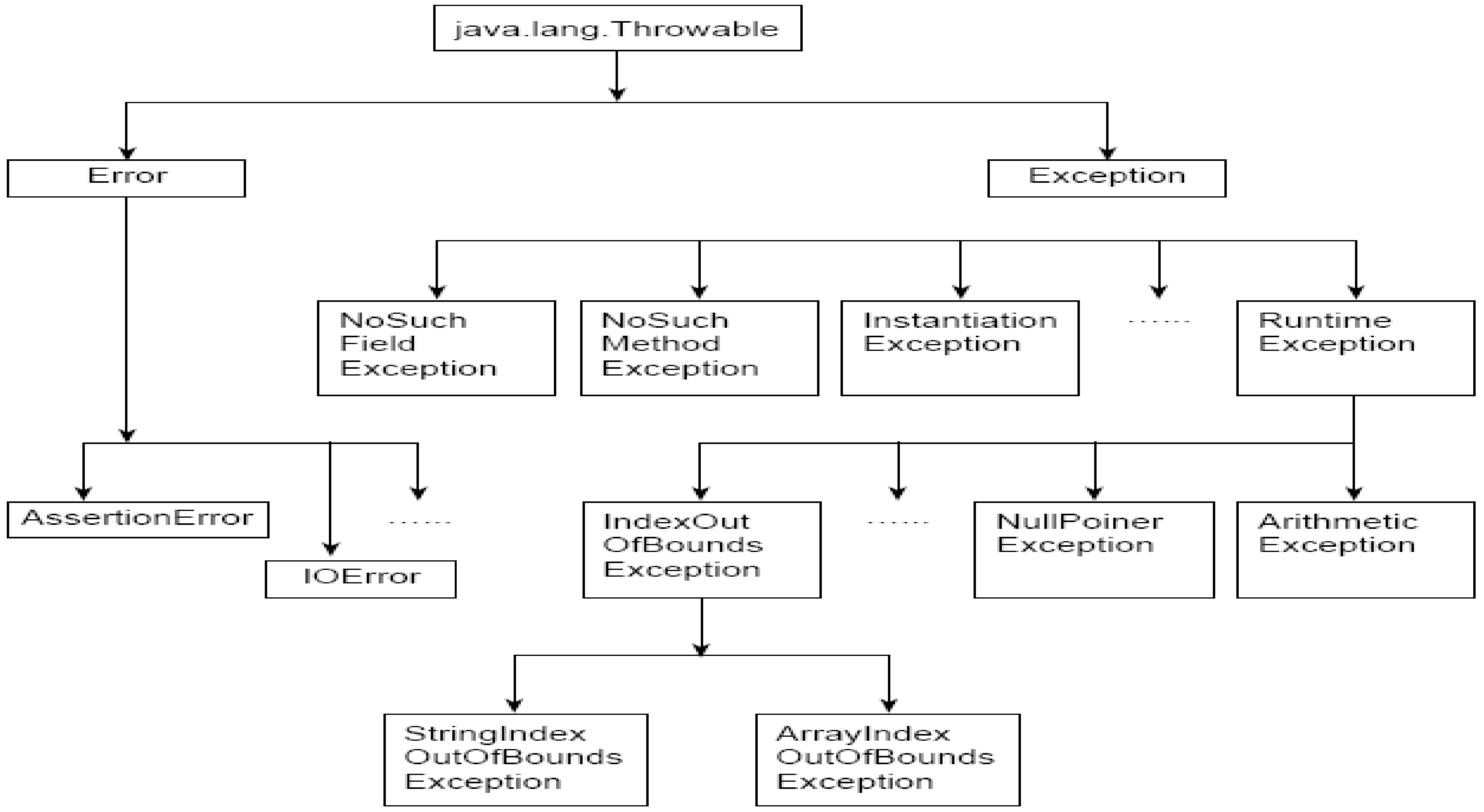
  Ex: OutOfMemoryError, VirtualMachineError, AssertionError etc.

Types of Exceptions



Checked Exceptions	Unchecked Exceptions
ClassNotFoundException	ArithmeticException
NoSuchFieldException	ArrayIndexOutOfBoundsException
NoSuchMethodException	NullPointerException
InterruptedException	ClassCastException
IOException	BufferOverflowException
IllegalAccessException	BufferUnderflowException

Exception Hierarchy



Exception Handling Techniques

- try..catch
- throw
- throws
- finally

try…catch

- try/catch block can be placed within any method that you feel can throw exceptions.
- All the statements to be tried for exceptions are put in a try block
- catch block is used to catch any exception raised from the try block.
- If exception occurs in any statement in the try block control immediately passes to the corresponding catch block.

Example

```
static void method2()
{
        System.out.println(“IN Method 2, Calling Method 3”);
        try{
                method3();
        }
        catch(Exception e)
        {
                System.out.println(“Exception Handled”);
        }
System.out.println(“Returned from method 3”);
}
```

Multiple catch clauses

```
static void method2()
{
        System.out.println(“IN Method 2, Calling Method 3”);
        try{
                method3(); }
        catch(ArithmeticException ae)
        {
                System.out.println (“Arithmetic Exception Handled: “ +ae);
        }
        catch(Exception e)
        {
                System.out.println(“Exception Handled”);
        }
System.out.println(“Returned from method 3”);
}
```

Note: catch having super class types should be defined later than the catch clauses with subclass types. The order is important.

Nested try..catch

- try{ …..//statements
- try{ …..//statements
- }
- catch(ArithmeticException ae){ . . . .}
- …// statements
- try{…//statements}
- catch(ArrayIndexOutOfBoundsException ie){}
- }
- catch(Exception e){…..}

throw keyword

- used to explicitly throw an exception.
- useful when we want to throw a user-defined exception.
- The syntax for *throw* keyword is as follows:
  - throw new ThrowableInstance;

  For example
  - throw new NullPointerException();

throws keyword

- is added to the method signature to let the caller know about what exception the called method can throw.
- responsibility of the caller to either handle the exception (using try…catch mechanism) or it can also pass the exception (by specifying throws clause in its method declaration).
- If all the methods in a program pass the exception to their callers (including main( )), then ultimately the exception passes to the default exception handler.

finally

- **finally** block is executed in all circumstances
  - if the exception occurs or
  - it is normal return (using return keyword) from methods.
- mandatory to execute statements like related to release of resources, etc. can be put in a **finally** block.
- The syntax of the **finally** keyword is as follows:
  - try {……}
  - catch(Throwable e){……}
  - finally {……..}

The finally block will not be executed if program exits(either by calling System.exit() , even if you use return, finally block will be executed

If exception occurs and did not handled

```
class TestFinallyBlock1{
  public static void main(String args[])
{
  try{
   int data=25/0;
   System.out.println(data);
  }
  catch(NullPointerException e)
{
System.out.println(e);
}
  finally
{
System.out.println("finally block is always executed");
}
  System.out.println("rest of the code...");
  } }
```

What is the output??

Output:

```
Output: finally block is always executed
        Exception in thread main java.lang.ArithmeticException: / by zero
```

Rules in Exception handling

- For each try block there can be zero or more catch blocks, but only one finally block.
- At a time only one Exception is occurred and at a time only one catch block is executed.
- All catch blocks must be ordered from most specific to most general i.e. catch for Arithmetic Exception must come before catch for Exception .
- If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.
- If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but can declare unchecked exception.

Multi catch

- Java 7 introduced the multi catch statement to catch multiple exceptions using a single catch

```
try {
// statements
}
catch (Exception1 | Exception2 | Exception3 e)
{ // statements }
```

- Exception1, Exception2, and Exception3 , belonging to different hierarchies are handled in a single catch block. For e.g.

Error Vs Exception In Java :

- 1) Recovering from Error is not possible. The only solution to errors is to terminate the execution. Where as you can recover from Exception by using either try-catch blocks or throwing exception back to caller.

- 2) You will not be able to handle the Errors using try-catch blocks. Even if you handle them using try-catch blocks, your application will not recover if they happen. On the other hand, Exceptions can be handled using try-catch blocks and can make program flow normal if they happen.

- 3) Exceptions in java are divided into two categories – checked and unchecked. Where as all Errors belongs to only one category i.e unchecked.

Customized Exception

- Create a class , which is sub class of Exception or RuntimException
- Provide string arg constructor
- Use throw keyword to throw exception when ever you want to raise the exception.

You can make this exception as checked or unchecked , depends on the application/coding standards.

Extend Exception class if you want to create checked exception

Extend RuntimeException class if you want to create unchecked exception

Queries?