https://gpcoder.com/2430-xu-ly-ngoai-le-trong-java-exception-handling/

EXCEPTION HANDLING

Instructor:

Table of contents

Section 1

JAVA EXCEPTIONS

Introduction Exception

- During the execution of a program, the computer will face the some of situations:
  - syntax error
  - logic algorithm error
  - runtime error
- An exception is an abnormal condition that arises[xuất hiện] in a code sequence at run time.
  - an exception is a run-time error
- Java's exception handling avoids the problems in the run-time.

Introduction Exception

- 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.
- **Exception** is an abnormal condition.
- 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.
  - ✓ **Example**:

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;//exception occurs
statement 6;
statement 7;
statement 8;
statement 9;
statement 10;
```

Hierarchy of Java Exception classes

- The **java.lang.Throwable** class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error.
- A hierarchy of Java Exception classes are given below:

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:
  - ✓ Checked Exception
  - ✓ Unchecked Exception
  - ✓ Error
- **Exception Keywords**

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.

Section 2

EXCEPTION HANDLING

Exception handling

- When an exceptional condition arises, an object representing that exception is created and thrown in the method that caused the error.
- Java exception handling is managed via five keywords: **try**, **catch**, **throw**, **throws**, and **finally**.
  - Program statements that you want to monitor for exceptions are contained within a **try** block.
  - Your code can catch this exception (using **catch**) and handle it in some rational manner.
  - To manually throw an exception, use the keyword **throw**. Any exception that is thrown out of a method must be specified as such by a **throws** clause. Any code that absolutely must be executed before a method returns is put in a **finally** block.

Exception handling

- Syntax of Java try-catch

```
try{
//code that may throw an exception

}catch(Exception_Class_Name ref){}
```

- Syntax of try-finally block

```
try{
//code that may throw an exception

} finally {}
```

- Syntax of try-catch-finally block

```
try{
//code that may throw an exception

} catch(Exception_Class_Name ref){

} finally {}
```

Multi-catch block

- A try block can be followed by one or more catch blocks.
- Each catch block must contain a different exception handler.
- You have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.

```
try {
    // block of code to monitor for errors
}
catch (ExceptionType1 exOb) {
    // exception handler for ExceptionType1
}
catch (ExceptionType2 exOb) {
    // exception handler for ExceptionType2
}
// ...
```

- **Points to remember:**
  - ✓ At a time only one exception occurs 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 ArithmeticException must come before catch for Exception.

Exception handling

- **Solution ex2:**

```
} catch (ArithmeticException ex1) {
  System.out.println(ex1.getStackTrace());
}
catch(ArrayIndexOutOfBoundsException ex2)
{
  System.out.println(ex2.getStackTrace());
}
catch(FileNotFoundException ex3)
{
  System.out.println(ex3.getStackTrace());
}
```

Multi-catch block

- In this example, try block contains two exceptions.
- But at a time only one exception occurs and its corresponding catch block is invoked.

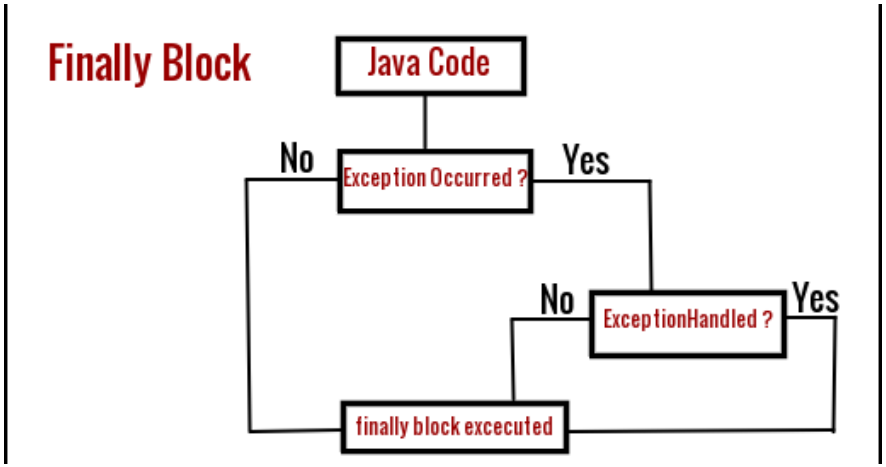
```
public class MultipleCatchBlock {
  public static void main(String[] args) {

    try {
      int a[] = new int[5];
      a[5] = 30 / 0;
      System.out.println(a[10]);

    } catch (ArithmeticException e) {
      System.out.println("Arithmetic Exception occurs");
    } catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("ArrayIndexOutOfBounds Exception occurs");
    } catch (Exception e) {
      System.out.println("Parent Exception occurs");
    }
    System.out.println("rest of the code");
  }
}
```

Output:

```
Arithmetic Exception occurs
rest of the code
```

finally block

- **Java finally block** is a block that is used *to execute important code* such as closing connection, stream etc.
- Java finally block is always executed whether exception is handled or not.
- Java finally block follows try or catch block.



Example:

```
Connection conn= null;
try {
    conn= get the db conn;
    //do some DML/DDL
} catch(SQLException ex) {
} finally {
    conn.close();
}
```

Difference between final, finally and finalize

final	finally	finalize
• Final is used to apply restrictions on class, method and variable • Final class can't be inherited • Final method can't be overridden • Final variable value can't be changed.	• Finally is used to place important code, it will be executed whether exception is handled or not.	• Finalize is used to perform clean up processing just before object is garbage collected.
Final is a keyword.	Finally is a block.	Finalize is a method.

Section 3

CHECKED AND UNCHECKED EXCEPTION

Checked vs Unchecked exceptions

Checked exceptions

- All exceptions other than Runtime Exceptions are known as Checked exceptions as the **compiler checks them during compilation** to see whether the programmer has handled them or not.
- If these exceptions are not handled/declared in the program**, it will give compilation error**.
- **Examples of Checked Exceptions:**
  - `ClassNotFoundException`
  - `IllegalAccessException`
  - `NoSuchFieldException`
  - `EOFException, etc.`

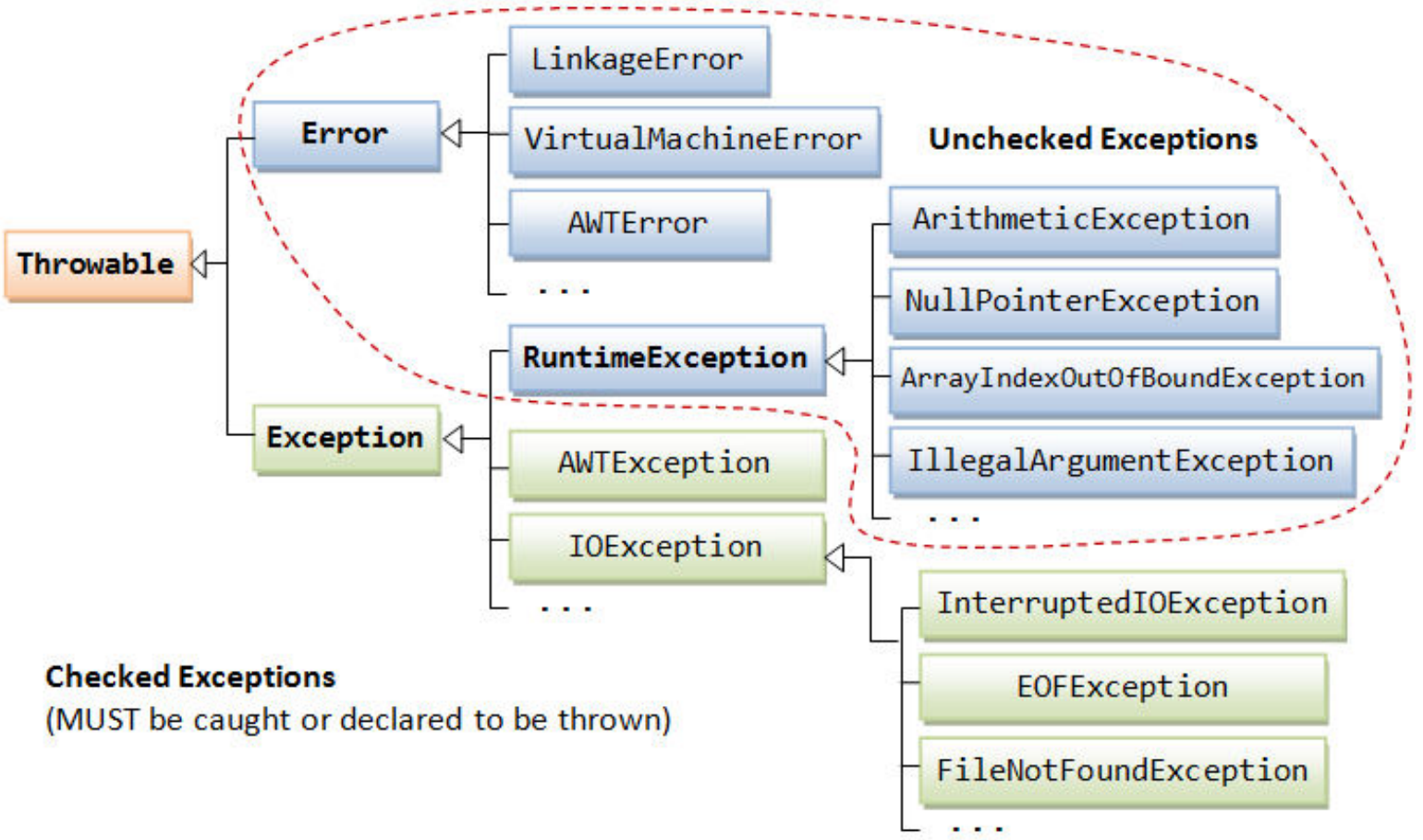
Checked vs Unchecked exceptions (cont.)

Unchecked Exceptions

- Runtime Exceptions are also known as Unchecked Exceptions as the **compiler do not check whether** the programmer has handled them or not but it's the duty of the programmer to handle these exceptions and provide a safe exit.
- If these exceptions are not handled/declared in the program, **it will not give compilation error**.
- **Examples of UnChecked Exceptions:**
  - `ArithmeticException`
  - `ArrayIndexOutOfBoundsException`
  - `NullPointerException`
  - `NegativeArraySizeException, etc.`

Exception classes

- The figure below shows the hierarchy of the Exception classes.
  - The base class for all Exception objects is: `java.lang.Throwable`, `java.lang.Exception` and `java.lang.Error`.



Section 4

THROW AND THROWS

throw keyword

- The Java throw keyword is used to explicitly throw an exception.
- We can throw either **checked** or **uncheked** exception in java by throw keyword.
- The throw keyword is mainly used to throw custom exception.
- **Syntax**:

```
throw exception;
```

```
throw new IOException("sorry device error);
```

- **Example**:

```
public static void isAge(int age) {
   if (age < 18) {
     throw new ArithmeticException("Not valid");
   } else {
     System.out.println("welcome to vote");
   }
 }
```

throw keyword

```
public class Main {

  public static void main(String[] args) {
    try {
      Validator.isAge(15);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

}
```

Output:

```
java.lang.ArithmeticException: Not valid
at fa.training.utils.Validator.isAge(Validator.java:20)
at fa.training.jpe2.Main.main(Main.java:9)
```

throws keyword

- The **Java throws keyword** is used to declare an exception.
  - ✓ It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.
- Exception Handling is mainly used to handle the checked exceptions.
- *Because*:
  - ✓ unchecked Exception: under your control so correct your code. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used
  - ✓ error: beyond your control e.g. you are unable to do anything if there occurs VirtualMachineError or StackOverflowError.
- **Syntax**:

```
return_type method_name() throws Exception_Class_Name {
//method code
}
```

Difference between throw and throws

throw	throws
Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
Throw is followed by an instance.	Throws is followed by class.
Throw is used within the method.	Throws is used with the method signature.
You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException,SQLException.

Section 5

COMMON SCENARIOS OF JAVA EXCEPTIONS

Common Scenarios of Java Exceptions

1. A scenario where ArithmeticException occurs

   If we divide any number by zero, there occurs an ArithmeticException.

```
int a=50/0;//ArithmeticException
```

2. A scenario where NullPointerException occurs

   If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.

```
String s = null;
System.out.println(s.length());//NullPointerException
```

Common Scenarios of Java Exceptions

3. A scenario where NumberFormatException occurs

The wrong formatting of any value may occur NumberFormatException. Suppose I have a string variable that has characters, converting this variable into digit will occur NumberFormatException..

```
String s = "abc";
int i = Integer.parseInt(s);//NumberFormatException
```

4. A scenario where ArrayIndexOutOfBoundsException occurs

If you are inserting any value in the wrong index, it would result in ArrayIndexOutOfBoundsException as shown below:.

```
int a[]=new int[5];
a[10]=50; //ArrayIndexOutOfBoundsException
```

Summary

- Java Exception
- Exception Handling
- Checked And Unchecked Exception
- Throw and throws keywords
- Common Scenarios of Java Exceptions

Thank you