

# Introduction to Computer Programming (Java A)

## Lab 13

### [Objective]

- Learn exception handling.

### [Exercises]

Suppose that you want to use a [java.io.BufferedReader](#) to read the text from a disk file. The program did not handle the exception declared, which resulted in compilation error.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

class ReadTextFile
{
    public static void main ( String[] args )
    {
        String fileName = "sample.txt" ;
        String line;

        // Create a BufferedReader and Attach a file
        BufferedReader in = new BufferedReader( new FileReader( fileName ) );

        // while not end of file
        while ((line=in.readLine())!=null)
            System.out.println(line);

        // close the file
        in.close();
    }
}
```

### Run result:

```
G:\2020Spring\CS102A\LAB14\LAB14_CODE\src>javac ReadTextFile.java
ReadTextFile.java:15: 错误：未报告的异常错误FileNotFoundException；必须对其进行捕获或声明以便抛出
    BufferedReader in = new BufferedReader( new FileReader( fileName ) );
                                   ^
ReadTextFile.java:18: 错误：未报告的异常错误IOException；必须对其进行捕获或声明以便抛出
    while ((line=in.readLine())!=null)
                        ^
ReadTextFile.java:23: 错误：未报告的异常错误IOException；必须对其进行捕获或声明以便抛出
        in.close();
        ^
3 个错误
```

### Why?

Because the FileReader's constructor, the readLine() , and the close () declare exceptions.

If a method declares an exception in its signature, you cannot use this method without handling the exception - you can't compile the program.

Fortunately, there are two ways to solve this problem.

### Method 1

Catch the exception via a "try-catch" (or "try-catch-finally") construct.

```
try {  
    // Main logic here  
    open file;  
    process file;  
    .....  
} catch (FileNotFoundException ex) { // Exception handlers below  
    // Exception handler for "file not found"  
} catch (IOException ex) {  
    // Exception handler for "IO errors"  
} finally {  
    close file; // always try to close the file  
}
```

try-catch-finally construct

Rewrite the previous code according to this structure to add exception handling.

```
import java.io.BufferedReader;  
import java.io.FileNotFoundException;  
import java.io.FileReader;  
import java.io.IOException;  
  
class ReadTextFileWithCatch  
{  
    public static void main ( String[] args )  
    {  
        String fileName = "sample.txt";  
        String line;  
        BufferedReader in = null;  
        try  
        {  
            // Create a BufferedReader and Attach a file  
            in = new BufferedReader( new FileReader( fileName ) );  
  
            // while not end of file  
            while ((line=in.readLine())!=null)  
                System.out.println(line);  
  
        }  
        catch (FileNotFoundException ex)  
        {  
            System.out.println("There is no this file!");  
        }  
        catch (IOException ex){  
            System.out.println("Read fileexception!");  
        }  
        // close the file  
        finally {  
            System.out.println("close the file");  
            if (in != null)  
            {  
                try{  
                    in.close();  
                }  
            }  
        }  
    }  
}
```

```

        catch (IOException ex){
            System.out.println("file close IOException");
        }
    }
}
}
}

```

Take note that the main logic in the try-block is separated from the error handling codes in the catch-block.

## Method2

You decided not to handle the exception in the current method, but throw the exception up the call stack for the next higher-level method to handle.

```

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

class ReadTextFileWithThrow
{
    public static void main ( String[] args ) throws IOException
    {
        String fileName = "sample.txt" ;
        String line;
        // Create a BufferedReader and Attach a file
        BufferedReader in = new BufferedReader( new FileReader( fileName ) );
        // while not end of file
        while ((line=in.readLine())!=null){
            System.out.println(line);
        }
        // close the file
        in.close();
    }
}

```

In this case, the next higher-level method of `main()` is the JVM .

## Call Stack for exception

Run the following code to see call stack of the exception.

```

public class MethodCallStackDemo {
    public static void main(String[] args){
        System.out.println("Enter main()");
        methodA();
        System.out.println("Exit main()");
    }

    public static void methodA() {
        System.out.println("Enter methodA()");
        try {

```

```

        methodB();
    } catch (ArithmeticException ex) {
        System.out.println(ex.toString());
    }

    System.out.println("Exit methodA()");
}
public static void methodB() throws ArithmeticException {
    System.out.println("Enter methodB()");
    methodC();
    System.out.println("Exit methodB()");
}
public static void methodC() throws ArithmeticException {
    System.out.println("Enter methodC()");
    methodD();
    System.out.println("Exit methodC()");
}

public static void methodD() throws ArithmeticException {
    System.out.println("Enter methodD()");
    // divide-by-0 triggers an ArithmeticException
    System.out.println(1 / 0);
    System.out.println("Exit methodD()");
}
}

```

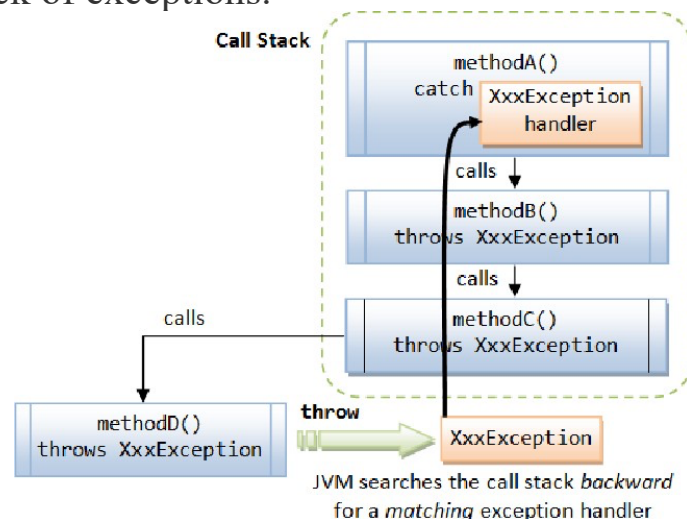
Run result:

```

Enter main()
Enter methodA()
Enter methodB()
Enter methodC()
Enter methodD()
java.lang.ArithmeticException: / by zero
Exit methodA()
Exit main()

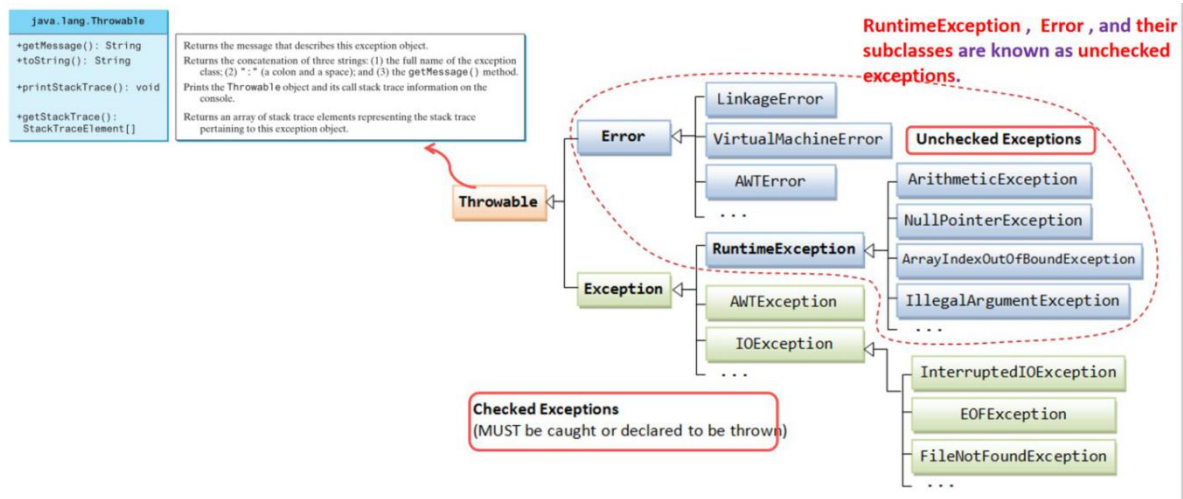
```

The following picture is a good explanation of the procedure for calling the stack of exceptions.



# Exception Classes - Throwable, Error, Exception & RuntimeException

The figure below shows the hierarchy of the Exception classes. The base class for all Exception objects is `java.lang.Throwable`, together with its two subclasses `java.lang.Exception` and `java.lang.Error`.



- The `Error` class describes internal system errors.
- The `Exception` class describes the error caused by your program.
- `RuntimeException`, `Error`, and their subclasses are known as unchecked exceptions. All other exceptions are known as checked exceptions, meaning the compiler forces the programmer to check and deal with them in a try-catch block or declare it in the method header

Five keywords are used in exception handling: **try**, **catch**, **finally**, **throws** and **throw** (take note that there is a difference between throw and throws).

**Java's exception handling consists of three operations:**

1. Declaring exceptions;
2. Throwing an exception; and
3. Catching an exception.

**The exception info is helpful to debug, it tells:**

1) Exception type

- ✓ Arithmetic
- ✓ ArrayIndexOutOfBoundsException
- ✓ NegativeArraySizeException
- ✓ NullPointerException
- ✓ NumberFormatException

2) Exception reason

- ✓ Divided by zero

✓ 3 is out of array Index bounds

✓ ...

### 3) Exception place

To further familiarize you with common exceptions, we define common exceptions as enumerations and write a program that selectively trigger exception.

```
public class CommonExceptionDemo {

    public static void main(String[] args) {

        ExceptionEnum exceptionIndex = ExceptionEnum.CLASSCAST;
        switch(exceptionIndex)
        {
            case ARITHMETIC:
            {
                System.out.println(1/0);
            }
            break;

            case INDEXOUTOFBOUNDS:
            {
                int[] anArray = new int[3];
                System.out.println(anArray[3]);
            }

            break;

            case NEGATIVEARRAYSIZE:
            {
                int[] anArray = new int[-1];
            }
            break;

            case NULLPOINTER:
            {
                String[] strs = new String[3];
                System.out.println(strs[0].length());
            }

            break;
        }
    }
}
```

```

        case NUMBERFORMAT:
        {
            Integer.parseInt("abc");
        }
        break;
        case CLASSCAST:
        {
            Object o = new Object();
            Integer i = (Integer)o;
        }

        break;
    }
}

enum ExceptionEnum {
    ARITHMETIC,
    INDEXOUTOFBOUNDS,
    NEGATIVEARRAYSIZE,
    NULLPOINTER,
    NUMBERFORMAT,
    CLASSCAST
;
}

```

You can change the value of [exceptionIndex](#) to learn about the various common exceptions.


## Lab exercise

Modify the program **CommonExceptionDemo.java** to accomplish the following tasks:

1. Display the info(name and ordinal value) of every element in a enum "ExceptionEnum".
2. Ask user to input a integer.
3. According to the value of user's input, trigger the Exception and show its information.
4. While get the input value use `try` and `catch` to check:
  - 1) If the input is not a number trigger `InputMismatchException`, Catch it and print the Exception message.
  - 2) If the input is in a number but its value is not Between 0 and 5,Throw an `IllegalArgumentException`,Catch it and print the exception message.

```

enum ExceptionEnum {
    ARITHMETIC,
    INDEXOUTOFBOUNDS,
    NEGATIVEARRAYSIZE,
    NULLPOINTER,
    NUMBERFORMAT,
    CLASSCAST
};

//complete code here to return the element whose ordinal value is same with ordn.
//and throw a IllegalArgumentException to alert the value of ordn is not in the range of 0~5,
public static ExceptionEnum getExceptionByOrdinal(int ord) throws IllegalArgumentException {
    
}
}

```


The sample inputs and outputs are as follows:



Exception:

ARITHMETIC(0)  
INDEXOUTOFBOUNDS(1)  
NEGATIVEARRAYSIZE(2)  
NULLPOINTER(3)  
NUMBERFORMAT(4)  
CLASSCAST(5)

Please INPUT an integer to select the TYPE OF EXCEPTION(0~5):1

Here is End 

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3  
at CommonExceptionDemo.main(CommonExceptionDemo.java:29)

Exception:

ARITHMETIC(0)  
INDEXOUTOFBOUNDS(1)  
NEGATIVEARRAYSIZE(2)  
NULLPOINTER(3)  
NUMBERFORMAT(4)  
CLASSCAST(5)

Please INPUT an integer to select the TYPE OF EXCEPTION(0~5):6

java.lang.IllegalArgumentException

Here is End

Exception:

ARITHMETIC(0)  
INDEXOUTOFBOUNDS(1)  
NEGATIVEARRAYSIZE(2)  
NULLPOINTER(3)  
NUMBERFORMAT(4)  
CLASSCAST(5)

Please INPUT an integer to select the TYPE OF EXCEPTION(0~5):c

java.util.InputMismatchException

Here is End