OOP
Lab 13.

Objectives

- Create and throw exceptions: `throw, throws`
- Exception handling: `try-catch-finally`
- User-defined exceptions
- File streams: `FileInputStream, FileOutputStream`

Create a Project/Module with 4 packages: `lab13_1, lab13_2, lab13_3, lab13_4`

Exercise 1.

Write a program (`main` method) that reads a line from the standard input and prints the sum of numbers in the line. Your program should be robust and work with lines containing non-numeric data. For example:

Input Line: `1   3.2   apple 4.7   pear  5`
Output: 13.9

Exercise 2.

Write a static method that reads student data from a text file and writes the data to another file. The input file (`students.csv`) contains in each line data of one student in the following format:

```
NeptunId, firstname, lastname, credits, birthyear, birthmonth, birthday
```

The input file may contain the following errors:

- Incomplete lines (lines containing less than 7 items)
- Some of the numeric data may contain non numeric characters or real numbers instead of integers (e.g. credits) resulting in `NumberFormatException`
- The year, month, day values do not form a valid date
- You should also check the existence of the input file
- ...

Create a `warning.csv` file that contains the data of students having less than 30 credits. For each incomplete line print an error message on the standard output. Lines containing formatting exceptions should result in an error message. You should write different error messages for different types of errors. For example:

Input line: `ABCXYZ, John, Black`
Error message: I`NCOMPLETE LINE: ABCXYZ, John, Black`

Input line: `A1B1C1, John, White, 3.5, 2000, 4, 23`
Error message: NUMBER FORMAT EXCEPTION: `A1B1C1, John, White, 3.5, 2000, 4, 23`

Input line: `A1B1C1, Rebecca, White, 55, 2000, 2, 30`
Error message: INVALID DATE: `A1B1C1, Rebecca, White, 55, 2000, 2, 30`

Exercise 3.

Create a `FileUtil` **interface** (>= Java 8) with two static methods:

- A method which encodes the bytes of a file simply by incrementing each byte of the file (0→ 1, 1→ 2, …. 255 → 0).
- Another static method for decoding the file.

```
public interface FileUtil {
   public static void encode(String inputFileName, String outputFileName)
throws IOException {
    try (InputStream in = new FileInputStream(inputFileName)) {
        try (OutputStream out = new FileOutputStream(outputFileName)) {
            int ch;
            while ((ch = in.read()) != -1) {
                out.write((ch + 1) % 256);
            }
        }
    }
}

   public static void decode(String inputFileName, String outputFileName)
throws IOException{
...
   }
}
```

Usage example:

```
try {
   FileUtil.encode("labor11/src/lab11_3/FileUtil.java", "temp1.java");
   FileUtil.decode("temp1.java", "temp2.java");
} catch (IOException e) {
   e.printStackTrace();
}
```

Exercise 4

Postfix expression evaluation

In this exercise you will write a flexible program to evaluate a postfix expression (See Fig. 1). The postfix expression may contain real numbers (decimal point) and four binary operators: `+`, `-`, `*`, `/`.
First of all, you need a stack for storing the partial results of the evaluation.

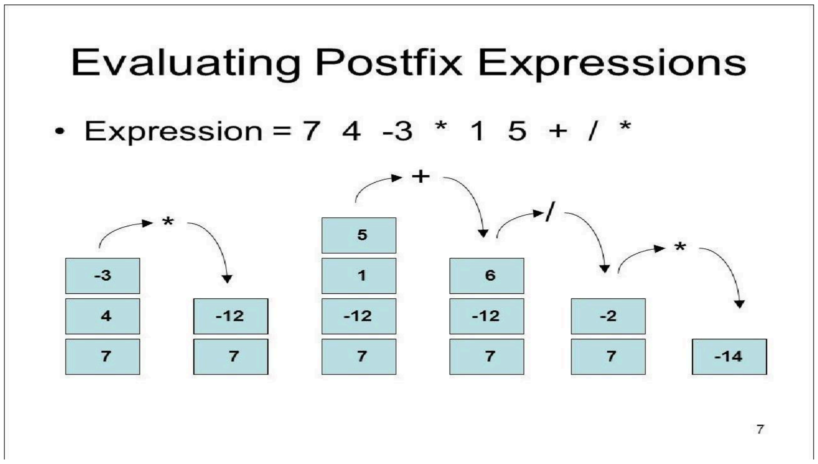
Evaluating Postfix Expressions
• Expression = 7 4 -3 * 1 5 + / *
*
-3
4
7
-12
7
+
5
1
-12
7
6
-12
7
/
-2
7
*
-14
7

Fig. 1

Use the StackAggregation from lab7_2.
Steps (see class diagram on Fig. 2):

- Copy the file `StackAggregation.java` (lab7_2).

- Rename the class to Stack (Refactoring!).
- Create your own **checked exception** class for the `Stack`: `StackException`.
- Modify the `push`, `top` and `pop` methods in order to throw an exception when the stack is full or empty.
- Create an `IExpression` interface with two static methods
  - `isOperator` checks whether the argument is an operator

```
public static  boolean isOperator(String op)
```

  - `evaluate` method evaluates the argument and returns its numeric value

```
public static double evaluate(String postfixExpression) throws
ExpressionException
```

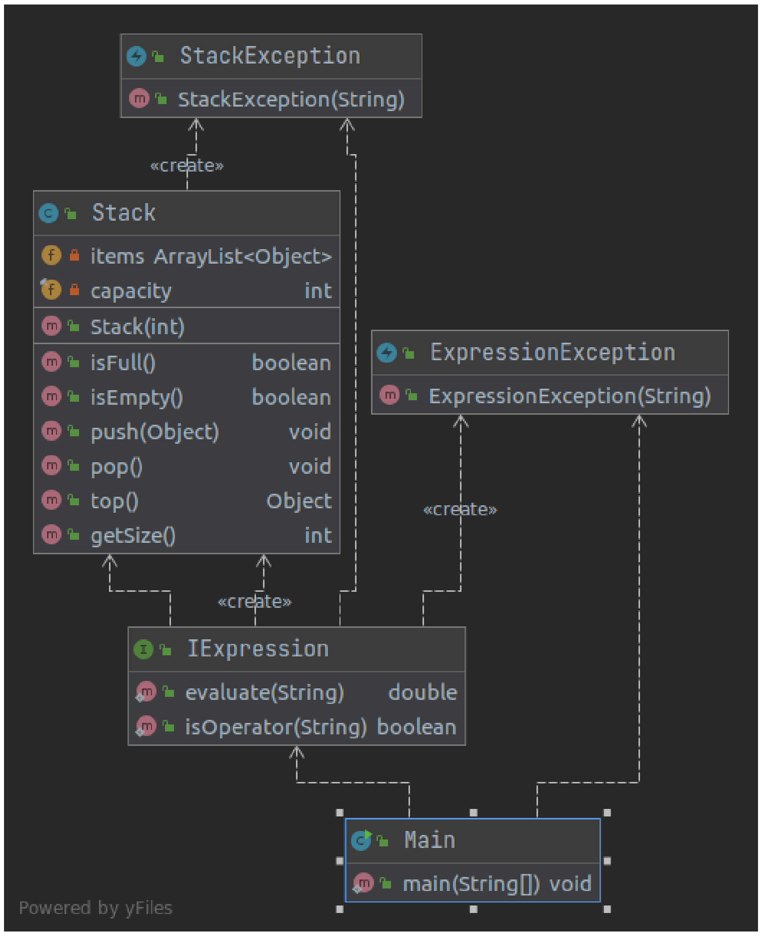

Fig. 2 . Class diagram for expression evaluation

Usage:

```
public static void main(String[] args) {
        String expressions[] = {
               "1 2 + 3 2 + * ",
               "1 2 + +",
               "1 a +",
               "1 2,3 *",
               "1 3 /" };

        for( String expr: expressions ) {
            try {
                System.out.println("Eval(" + expr + "): " +
                                       IExpression.evaluate(expr));
            } catch (ExpressionException e) {
                System.out.println("Wrong expression: " + expr);
                System.out.println("\t" + e.getMessage());
            }
        }

    }
```

Outputs for different expressions:

-
  ```
  Eval(1 2 + 3 2 + *): 15.0
  ```

-
  ```
  Wrong expression: 1 2 + +
          Wrong postfix expression
  ```

-
  ```
  Wrong expression: 1 a +
           Wrong operand: a
  ```

-
  ```
  Wrong expression: 1,3 a +
          Wrong operand: 1,3
  ```

-
  ```
  Eval(-1 2 3 * +): 5.0
  ```