OOP Project 1

Command line calculator in Java

What it does ?

- Evaluate standard math operators

```
Enter expression to evaluate:
> 5+6^2 - 3/6
 = 40.5
```

- Evaluate common math functions and constants

```
Enter expression to evaluate:
> sin( ln (e^2))
 = 0.9092974268256817
```

What it does ?

- remember the last result

```
Enter expression to evaluate:
> exp ( 3 )
 = 20.085536923187668

Enter expression to evaluate:
> ans * 2
 = 40.171073846375336
```

What it does ?

- Display helper a helper menu and history

```
Enter expression to evaluate:
> !help
Supported Operators:
    +, -, *, / , ^
Supported Single Argument Functions:
    sin, cos, ln, exp,
Function use syntax:
    function('argument')
    or  function 'argument'
    eg: sin(ln(13)) <=> sin ln 23
Supported Constants:
    e, pi
```

```
Enter expression to evaluate:
> !his
5+6^2 - 3/6
ans + sin(ln(67))
sin(ln(67))
sin( ln (e^2))
```

How it does it

Application.java

```java
import pl.poznan.put.calculator.SYCalculator;
import pl.poznan.put.calculator.CalculatorMenu;

import java.util.Scanner;

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

        Scanner reader = new Scanner(System.in);
        SYCalculator calculator = new SYCalculator();
        CalculatorMenu menu = new CalculatorMenu(calculator, reader);

        menu.startMenu();

        menu.runLoop();

        reader.close();

    }
}
```

How it does it

Menu.java

```java
package pl.poznan.put.calculator;

public interface Menu {
    1 usage  1 implementation
    void startMenu();
    1 usage  1 implementation
    void runLoop();
}
```

Calculator.java

```java
package pl.poznan.put.calculator;

import java.util.ArrayList;

1 usage  1 implementation
public interface Calculator {
    1 usage  1 implementation
    double expressionValue(String inputString) throws Exception;
    1 usage  1 implementation
    ArrayList<String> expressionHistory();
}
```

How it does it

Lexer

```java
package pl.poznan.put.calculator;

import java.util.List;

1 usage   1 implementation
public interface LexerInterface {
    1 usage   1 implementation
    List<Token> scanTokens();
}
```

Token

```java
package pl.poznan.put.calculator;

public record Token(TokenType type, String value) {

    @Override
    public String toString() {
        return type.toString() + " " + value;
    }
}
```

How it does it

Expression - the heart of the Calculator

```java
package pl.poznan.put.calculator;

public interface Expression {
    1 usage  1 implementation
    double evaluate() throws Exception;
    1 usage  1 implementation
    void parse() throws Exception;
}
```

parse : infix form -> postfix form

evaluate: postfix form -> value

Problems I encountered

Imperative / functional style of thinking

- First writing the logic, then creating objects around it. This created problems with encapsulation and shared state

No first class methods in Java

- There is no way to pass around ‘function objects’, like one can in python or C++. Both calculator and expression need a copy of the standard functions supported by the calculator.

What I learned

1. Not naming objects with “-er” actually makes sense
    a. “Parser” -> “Expression”
2. Exceptions are better than Null, NaN etc.
3. Plan before you begin writing
4. OOP projects are hard to change after the planning stage
    a. Adding new, unplanned functionality is difficult.

What could be improved

SYExpression (implementation of Expression Interface) class could be immutable: two types of expressions => more complexity

Calculator menu could take an out stream as argument