Metode Avansate de Programare, LABORATOR 1

DEADLINE: Săptămâna 2

A. Instalați un mediu de dezvoltare intregrat pt Java (Java IDE) pe laptopurile proprii și încercați să vă familiarizați cu acesta.
La curs si seminar vom folosi: InteliJ IDEA, Ultimate Edition (Pentru a putea activa licenta, creati un JetBrains Account cu adresa de mail de la facultate).

[Download JDK]
https://www.oracle.com/java/technologies/downloads/

[Download InteliJ IDEA]
https://account.jetbrains.com/login
[InteliJ IDEA] https://www.jetbrains.com/idea/documentation/

B. Se citesc, ca parametri în linia de comanda, separati prin spatiu, mai multe numere complexe de forma a+b*i și o operație, sub forma unui operator (+, - , *, /).

1. Sa se verifice dacă parametri citiți în linia de comanda, separați prin spațiu, reprezintă o expresie aritmetica de forma:

$$n_1 \ op \ n_2 \ op \ \dots op \ n_k,$$

unde $n_1, n_2 \dots n_k$ sunt numere complexe de forma $a + b * i$, iar op este operatorul dat.
Exemplu: $2 + 3 * i + 5 - 6 * i + -2 + i$
args[0]= $2 + 3 * i$, args[1]= $+$, args[2]= $5 - 6 * i$, args[3]= $+$, args[4]= $-2 + i$

2. Dacă parametri citiți în linia de comandă reprezintă o expresie aritmetică de forma descrisă la punctul 1, se cere sa se afișeze rezultatul acestei expresii.
Exemplu: pentru expresia data se va afisa: $5 - 2 * i$

Observatii:

1) Sunteti incurajati să veniți cu propria proiectare pentru rezolvarea problemei și să aveți în vedere principiile de proiectare orientată obiect, învatate pana acum (Vezi SOLID design principles).
2) Sugestii de proiectare (daca nu va este teama sa va jucati cu clasele, puteti veni si cu propria abstractizare, chiar sunteti incurajati).
   Se vor defini clasele:
   - *NumarComplex*, având ca atribute *re, im* de tip real, iar ca metode operatiile definite pe multimea numerelor complexe: *adunare, scadere, inmultire, impartire si conjugatul.*
   - *ComplexExpression,* clasa abstracta, ce are ca atribute *operation* de tipul *Operation (enum* având patru valori posibile: `ADDITION`, `SUBSTRACTION`, `MULTIPLICATION`, `DIVISION`) si un vector (array) *args* de numere complexe, iar

ca metode, o metodă *execute* ce returneaza rezultatul expresiei aritmetice, apeland pentru aceasta metoda abstractă *executeOneOperation* [TemplateMethodDesignPattern]. Derivati din clasa *ComplexExpression* clasele necesare pentru executia celor patru tipuri de expresii, definite la punctul B.l., suprascriind, corespunzator tipului operatiei dat (+,-,/,*), metoda *executeOneOperation* - executa o operatie din insiruirea de operatii ale expresiei.

- *ExpressionFactory* (singleton) [SingletonDesignPattern] [FactoryMethodDesignPattern] având metoda *createExpression cu* signatura*:*
```
public ComplexExpression createExpression(Operation operation, Complex[] args)
```
si care creează o expresie în funcție de valoarea parametrului *operation* (vezi diagrama de clase).
- *ExpressionParser* care parsează expresia, verifică dacă este validă si construiește, folosind un obiect de tipul ExpressionFactory, expresia corespunzatoare operatorului dat.

Alte Referinte:

A se vedea si cursul si seminarul 1.

[FactoryMethodDesignPattern] https://www.tutorialspoint.com/design_pattern/factory_pattern.htm

[SingletonDesignPattern] https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm

[TemplateMethodDesignPattern] https://www.tutorialspoint.com/design_pattern/template_pattern.htm

3 oct. 2023

```
Pattern p = Pattern.compile("regex-string");
p.match(...) (true or false)
```

- declarăm 4 operații

addition → n_1, n_2

sub → n_1, n_2

multiplication → n_1, n_2

division → n_1, n_2

```
public interface ComplexOp{
    ComplexNr compute(ComplexNr value);
}
public Addition implement ComplexOp{
  @ override
  ComplexNr compute(ComplexNr v1, ComplexNr v2){
  ...
  return ...
  }
}

new Addition();
```

```
public class Addition implements ComplexOp{
        public ComplexNr compute(ComplexNr value){
        ....
        }
}
```

---

edit dim edit configuration la main

arguments: $n_0, \ldots, n_1, n_2, \ldots$

```
public static void main(String[] args)
{
...
}
```

Comenzi Git (în repo local)

- git status = ce e nou, ce a-a șters
- git add . = formez pachetul de schimbări
- git commit -m "mesaj" = ce am făt pe comit
- git push = trimitem spre repo
- git pull = aduce toate schimbările de pe server

branchuri : main, master, etc.