

Metode Avansate de Programare, 2021–2022, LABORATOR 1

DEADLINE: Săptămâna 2

- A. **3p.** Instalați un mediu de dezvoltare integrat pt Java (Java IDE) pe laptopurile proprii și încercați să vă familiarizați cu acesta.

La curs și seminar vom folosi: IntelliJ IDEA, Ultimate Edition (Pentru a putea activa licența, creați un JetBrains Account cu adresa de mail de la facultate).

[Download JDK]

<https://www.oracle.com/java/technologies/javase-jdk14-downloads.html>

[Download IntelliJ IDEA]

<https://account.jetbrains.com/login>

[IntelliJ IDEA] <https://www.jetbrains.com/idea/documentation/>

- B. **6p.** Se citesc, ca parametri în linia de comandă, separați prin spațiu, mai multe numere complexe de forma $a+bi$ și o operație, sub forma unui operator (+, -, *, /).

1. **2p.** Să se verifice dacă parametri citați în linia de comandă, separați prin spațiu, reprezintă o expresie aritmetică de forma:

$$n_1 \text{ op } n_2 \text{ op } \dots \text{ op } n_k,$$

unde $n_1, n_2 \dots n_k$ sunt numere complexe de forma $a + b * i$, iar op este operatorul dat.

Exemplu: $2 + 3 * i + 5 - 6 * i + -2 + i$

$\text{args}[0] = 2 + 3 * i$, $\text{args}[1] = +$, $\text{args}[2] = 5 - 6 * i$, $\text{args}[3] = +$, $\text{args}[4] = -2 + i$

2. **4p.** Dacă parametri citați în linia de comandă reprezintă o expresie aritmetică de forma descrisă la punctul 1, se cere să se afișeze rezultatul acestei expresii.

Exemplu: pentru expresia dată se va afișa: $5 - 2 * i$

Observatii:

1) **Sunteți încurajați să veniți cu propria proiectare pentru rezolvarea problemei dacă aceasta este corectă și respectă principiile de proiectare orientată obiect învățate până acum (Vezi SOLID design principles). O proiectare simplistă (totul într-un fișier, fără clase specializate) care doar rezolvă problema va fi punctată cu nota 8.**

2) Sugestii de proiectare (dacă nu va este teama să vă jucați cu clasele, puteți veni și cu propria abstractizare, chiar sunteți încurajați).

Se vor defini clasele:

- *NumarComplex*, având ca atribute *re*, *im* de tip real, iar ca metode operațiile definite pe mulțimea numerelor complexe: *adunare*, *scadere*, *inmultire*, *impairare* și *conjugatul*.

- *ComplexExpression*, clasa **abstractă**, ce are ca atribut *operation* de tipul *Operation* (enum având patru valori posibile: **ADDITION**, **SUBTRACTION**, **MULTIPLICATION**, **DIVISION**) și un vector (array) *args* de numere complexe, iar ca metode, o metodă *execute* ce returnează rezultatul expresiei aritmetice, apelând pentru aceasta metoda **abstractă** *executeOneOperation* [TemplateMethodDesignPattern]. Derivați din clasa *ComplexExpression* clasele necesare pentru executia celor patru tipuri de expresii, definite la punctul B.1., suprascriind, corespunzător tipului operației dat (+,-,/,*), metoda *executeOneOperation* - execută o operație din înșiruirea de operații ale expresiei.
- *ExpressionFactory* (singleton) [SingletonDesignPattern] [FactoryMethodDesignPattern] având metoda *createExpression* cu semnatura: **public** ComplexExpression createExpression(Operation operation, Complex[] args) și care creează o expresie în funcție de valoarea parametrului *operation* (vezi diagrama de clase).
- *ExpressionParser* care parsează expresia, verifică dacă este validă și construiește, folosind un obiect de tipul *ExpressionFactory*, expresia corespunzătoare operatorului dat.

Alte Referinte:

A se vedea și cursul și 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