Sarcina spre realizare:

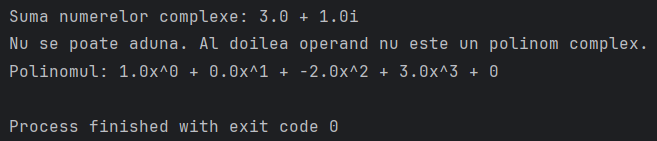
Realizarea polimorfismului pentru clasele date cu crearea metodelor conform specificului   
claselor

2. Complex <- ComplexPolynom

GitHub: https://github.com/Panfil-Ion/TAP.git

**Codul**

// Interfața pentru operațiile specifice numerelor complexe  
interface ComplexOperations {  
 Complex add(Complex other);  
 void display();  
}  
  
// Interfața pentru operațiile specifice polinoamelor de numere complexe  
interface ComplexPolynomOperations extends ComplexOperations {  
 void displayPolynom();  
}  
  
// Clasa pentru numere complexe  
class Complex implements ComplexOperations {  
 protected double real;  
 protected double imaginary;  
  
 // Constructor  
 public Complex(double real, double imaginary) {  
 this.real = real;  
 this.imaginary = imaginary;  
 }  
  
 // Adunarea a două numere complexe  
 public Complex add(Complex other) {  
 double newReal = this.real + other.real;  
 double newImaginary = this.imaginary + other.imaginary;  
 return new Complex(newReal, newImaginary);  
 }  
  
 // Metoda pentru afișarea numărului complex  
 public void display() {  
 System.*out*.println(this.real + " + " + this.imaginary + "i");  
 }  
}  
  
// Clasa pentru polinoame de numere complexe  
class ComplexPolynom extends Complex implements ComplexPolynomOperations {  
 private double[] coefficients;  
  
 public ComplexPolynom(double real, double imaginary, double[] coefficients) {  
 super(real, imaginary);  
 this.coefficients = coefficients;  
 }  
  
 @Override  
 public Complex add(Complex other) {  
 if (other instanceof ComplexPolynom) {  
 ComplexPolynom otherPolynom = (ComplexPolynom) other;  
 // Verificăm dacă ambele obiecte sunt polinoame  
 if (this.coefficients.length != otherPolynom.coefficients.length) {  
 System.*out*.println("Nu se poate aduna. Polinoamele nu au aceeasi lungime.");  
 return null;  
 }  
 double[] resultCoefficients = new double[this.coefficients.length];  
 for (int i = 0; i < this.coefficients.length; i++) {  
 resultCoefficients[i] = this.coefficients[i] + otherPolynom.coefficients[i];  
 }  
 return new ComplexPolynom(this.real, this.imaginary, resultCoefficients);  
 } else {  
 System.*out*.println("Nu se poate aduna. Al doilea operand nu este un polinom complex.");  
 return null;  
 }  
 }  
  
 public void displayPolynom() {  
 // Implementare pentru afișarea polinomului  
 System.*out*.print("Polinomul: ");  
 for (int i = 0; i < coefficients.length; i++) {  
 System.*out*.print(coefficients[i] + "x^" + i + " + ");  
 }  
 System.*out*.println("0");  
 }  
}  
  
  
public class Main {  
 public static void main(String[] args) {  
 // Exemplu de utilizare a claselor  
 Complex numarComplex1 = new Complex(2, 3);  
 Complex numarComplex2 = new Complex(1, -2);  
  
 Complex suma = numarComplex1.add(numarComplex2);  
 System.*out*.print("Suma numerelor complexe: ");  
 suma.display();  
  
 double[] coeficienti = {1, 0, -2, 3}; // exemplu de coeficienți pentru un polinom  
 ComplexPolynom polinomComplex = new ComplexPolynom(0, 0, coeficienti);  
  
 polinomComplex.add(numarComplex1); // exemplu de utilizare a metodei suprascrise din clasa derivată  
 polinomComplex.displayPolynom(); // exemplu de utilizare a metodei specifică clasei ComplexPolynom  
 }  
}

Rezultatul codului  


Concluzie

Acest cod demonstrează utilizarea conceptelor de interfețe, moștenire și polimorfism în Java pentru a manipula numere complexe și polinoame de numere complexe. Prin intermediul acestor concepte, putem organiza și extinde funcționalitatea într-o manieră modulară și flexibilă, permițându-ne să lucrăm cu aceste entități matematice într-un mod eficient și ușor de înțeles.