**ДНІПРОВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ   
ІМЕНІ ОЛЕСЯ ГОНЧАРА**

Факультет прикладної математики

Кафедра обчислювальної математики та математичної кібернетики

# ЗВІТ

до контрольно - модульної роботи

з дисципліни «Сучасні середовища програмування»

Виконала:

Студент групи ПА-17-1

Нікітченко М. І.

Дніпро

2020

**Завдання 2**

**Програмування лінійних алгоритмів**

У звіті повинні бути представлені:

1.Постановка завдання

•Застосувати патерн Компонувальник (Composite) для задачі обчис-лення похідної довільної складеної функції.

•Скласти звіт за формою:

**1)Постановка завдання**

**Варіант 2**

m = 2; c = -1; t = 1.2; b = 0.7

**2)Опис логічної структури Java-програми. Опис патернів, перелік кла-сів та параметрів конструкторів**

Середовище програмування обрано IntelliJ IDEA. Для виконання роботи було створено декілька класів, такі як:

Constant.java

Cos.java

Difference.java

Division.java

Function.java

Linear.java

Main.java

Menu.java

Multiplication.java

PowerFunction.java

Sin.java

Sum.java

Tan.java

Кожен клас відповідає за конкретні пункти завдання.

Клас, де знаходиться головна функція - Main.java – потрібен для виводу на консоль дій, які повинен зробити користувач.

Клас Menu.java – де за допомогою кейсів, ми просимо обрати пункт меню: 0 – вихід, 1 – похідні.

Клас Constant.java – використано для створення константи.

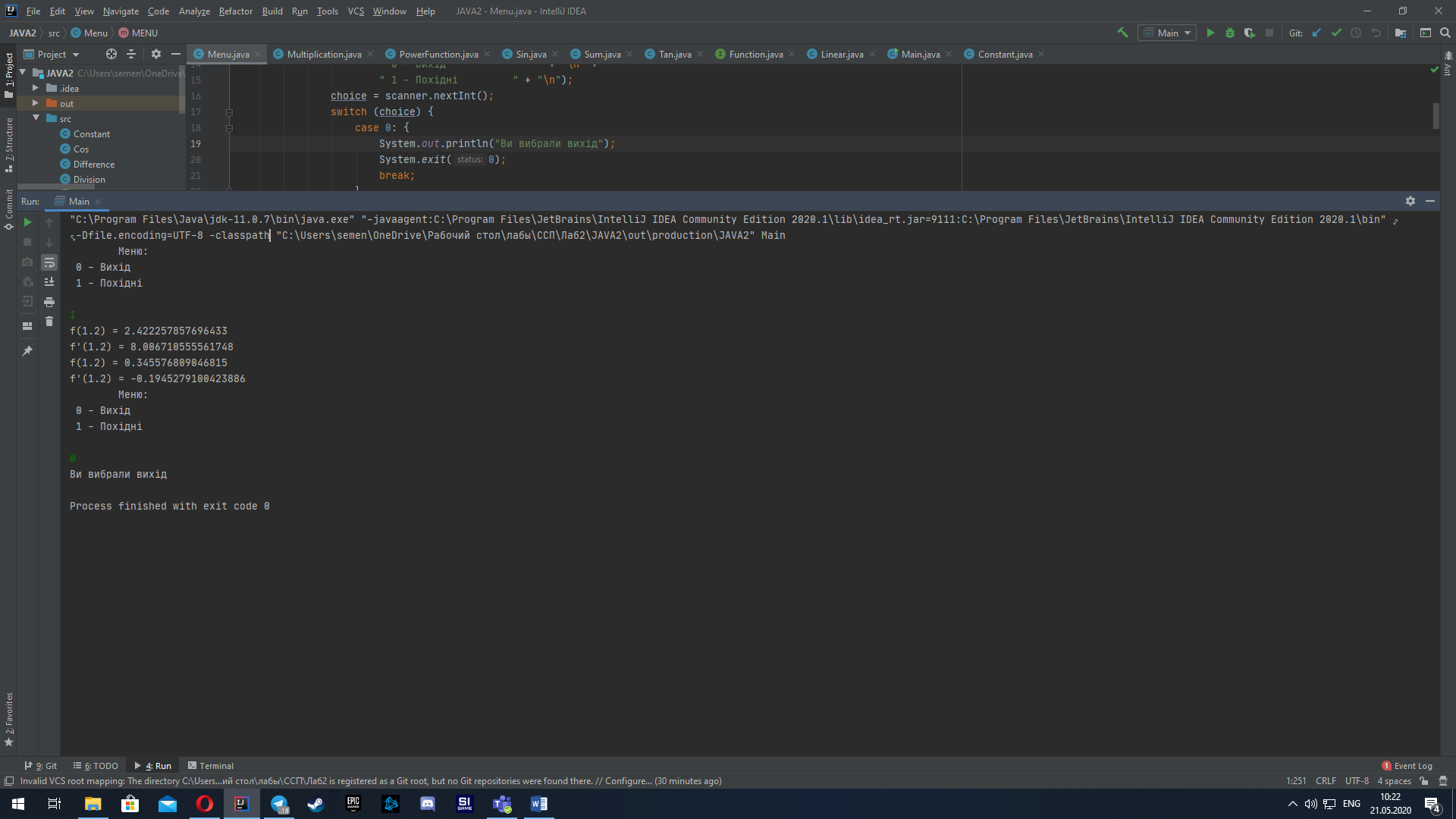
Клас Division.java – використано для ділення.

Клас Multiplication.java – використано для множення.

Клас PowerFunction.java – використано для зведення до ступеня.

Клас Sum.java – використано для суми.

1. **Результати виконання програми:**



1. **Висновки**

В ході виконання цієї лабораторної роботи було вивчено як правильно формувати програму, тобто для окремого завдання – свій клас з потрібними функціями.

Також використовував бібліотеку для відловлювання помилок, розробив інтерфейс для класів.

1. **Додатки (код програми)**

***Код програми:***

Constant.java:

public class Constant implements Function{  
 private final double value;  
  
 public Constant(double value) {  
 this.value = value;  
 }  
  
 @Override  
 public double calculate(double x) {  
 return value;  
 }  
  
 @Override  
 public Function derivative() {  
 return *ZERO*;  
 }  
  
 public static final Constant *ZERO* = new Constant(0);  
}

Cos.java:

import static java.lang.Math.*cos*;  
  
public class Cos implements Function {  
 public final Function inner\_func;  
  
 public Cos(Function inner\_func) {  
 this.inner\_func = inner\_func;  
 }  
  
 @Override  
 public double calculate(double x) {  
 return *cos*(inner\_func.calculate(x));  
 }  
  
 @Override  
 public Function derivative() {  
 return new Multiplication(new Difference(new Constant(0), new Sin(inner\_func)), inner\_func.derivative());  
 }  
}

Difference.java:

public class Difference implements Function {  
 private final Function a, b;  
  
 public Difference(Function a, Function b) {  
 this.a = a;  
 this.b = b;  
 }  
  
 @Override  
 public double calculate(double x) {  
 return a.calculate(x) - b.calculate(x);  
 }  
  
 @Override  
 public Function derivative() {  
 return new Difference(a.derivative(), b.derivative());  
 }  
}

Division.java:

public class Division implements Function {  
 public final Function a, b;  
  
 public Division(Function a, Function b) {  
 this.a = a;  
 this.b = b;  
 }  
  
 @Override  
 public double calculate(double x) {  
 return a.calculate(x) / b.calculate(x);  
 }  
  
 @Override  
 public Function derivative() {  
 return (new Multiplication(a, new PowerFunction(-1, b))).derivative();  
 }  
}

Function.java:

public interface Function {  
 double calculate(double x);  
 Function derivative ();  
}

Linear.java:

public class Linear implements Function {  
 public final double coefficient;  
  
 public Linear(double coefficient) {  
 this.coefficient = coefficient;  
 }  
  
 @Override  
 public double calculate(double x) {  
 return coefficient \* x;  
 }  
  
 @Override  
 public Function derivative() {  
 return new Constant(coefficient);  
 }  
}

Main.java:

import java.util.InputMismatchException;  
import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 Runnable runnable = () -> {  
 *RunProg*();  
 };  
 new Thread(runnable).start();  
 }  
  
 static void RunProg() {  
 Menu menu = new Menu();  
 menu.MENU();  
 }  
  
}

Menu.java:

import java.util.InputMismatchException;  
import java.util.Scanner;  
  
public class Menu {  
 public void MENU(){  
 int choice = 100;  
 Scanner scanner = new Scanner(System.*in*);  
 do {  
 try {  
  
  
 System.*out*.println(  
 " Меню: " + "\n" +  
 " 0 - Вихід " + "\n" +  
 " 1 - Похідні " + "\n");  
 choice = scanner.nextInt();  
 switch (choice) {  
 case 0: {  
 System.*out*.println("Ви вибрали вихід");  
 System.*exit*(0);  
 break;  
 }  
 case 1: {  
 double b = 0.7;  
 double c = -1;  
 double t = 1.2;  
 double m = 2;  
 final Function expression = new PowerFunction(1.0 / 3,  
 new Sum(new Multiplication(new Constant(m), new Tan(new Linear(t))),  
 new Multiplication(new Constant(c), new Sin(new Linear(t)))));  
 System.*out*.println("f("+t+") = " + expression.calculate(t));  
 System.*out*.println("f'("+t+") = " + expression.derivative().calculate(t));  
 final Function expression1 = new Sum(new Multiplication(new Constant(m),  
 new Cos(new Multiplication(new Multiplication(new Constant(b), new Constant(t)),  
 new Sin(new Linear(t))))), new Constant(c));  
 System.*out*.println("f("+t+") = " + expression1.calculate(t));  
 System.*out*.println("f'("+t+") = " + expression1.derivative().calculate(t));  
 }  
  
 }  
 } catch (InputMismatchException e) {  
 System.*out*.println("Невірне значення");  
 scanner.nextLine();  
 }  
 } while (choice != 0);  
 }  
}

Multiplication.java:

public class Multiplication implements Function{  
 public final Function a, b;  
  
 public Multiplication(Function a, Function b) {  
 this.a = a;  
 this.b = b;  
 }  
  
 @Override  
 public double calculate(double x) {  
 return a.calculate(x) \* b.calculate(x);  
 }  
  
 @Override  
 public Function derivative() {  
 return new Sum(new Multiplication(a, b.derivative()), new Multiplication(a.derivative(), b));  
 }  
}

PowerFunction.java:

import static java.lang.Math.*pow*;  
  
public class PowerFunction implements Function {  
 public final double degree;  
 public final Function func;  
  
 public PowerFunction(double degree, Function func) {  
 this.degree = degree;  
 this.func = func;  
 }  
  
 @Override  
 public double calculate(double x) {  
 return *pow*(func.calculate(x), degree);  
 }  
  
 @Override  
 public Function derivative() {  
 return new Multiplication(new Constant(degree),  
 new Multiplication(func.derivative(), new PowerFunction(degree - 1, func)));  
 }  
}

Sin.java:

import static java.lang.Math.*sin*;  
  
public class Sin implements Function{  
 public final Function function;  
  
 public Sin(Function function) {  
 this.function = function;  
 }  
  
 @Override  
 public double calculate(double x) {  
 return *sin*(function.calculate(x));  
 }  
  
 @Override  
 public Function derivative() {  
 return new Multiplication(new Cos(function), function.derivative());  
 }  
}

Sum.java:

public class Sum implements Function {  
 private final Function a, b;  
  
 public Sum(Function a, Function b) {  
 this.a = a;  
 this.b = b;  
 }  
  
 @Override  
 public double calculate(double x) {  
 return a.calculate(x) + b.calculate(x);  
 }  
  
 @Override  
 public Function derivative() {  
 return new Sum(a.derivative(), b.derivative());  
 }  
}

Tan.java:

import static java.lang.Math.*tan*;  
  
public class Tan implements Function {  
 public final Function function;  
  
 public Tan(Function function) {  
 this.function = function;  
 }  
 @Override  
 public double calculate(double x) {  
 return *tan*(function.calculate(x));  
 }  
  
 @Override  
 public Function derivative() {  
 return new Multiplication(new Division(  
 new Constant(1), new PowerFunction(2, new Cos(function))),  
 function.derivative());  
 }  
}