МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Український державний університет науки і технологій**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №1**

**з дисципліни «Проектний практикум»**

**на тему:** «***Дослідження предметної області. Виявлення функціональних вимог»***

Виконав: студент гр. ПЗ21140 (940П)

Ахленко Д. А.

Талибов Е. Т.

Перевірив:   
Гришечкіна Т. С.

Дніпро, 2023

**Лабораторна робота №1**

**Тема: *Дослідження предметної області. Виявлення функціональних вимог***.

**Мета:** Отримати навики виявлення та опис функціональних вимог до продукту.

1. **Постановка завдання**

Створення текстового редактору для виконання рефакторингу вихідних текстів програм для мови С++. Функції редактору:

* Вбудування методу
* Перейменування методу

1. **Концептуальна модель системи (узагальнена діаграма класів).**

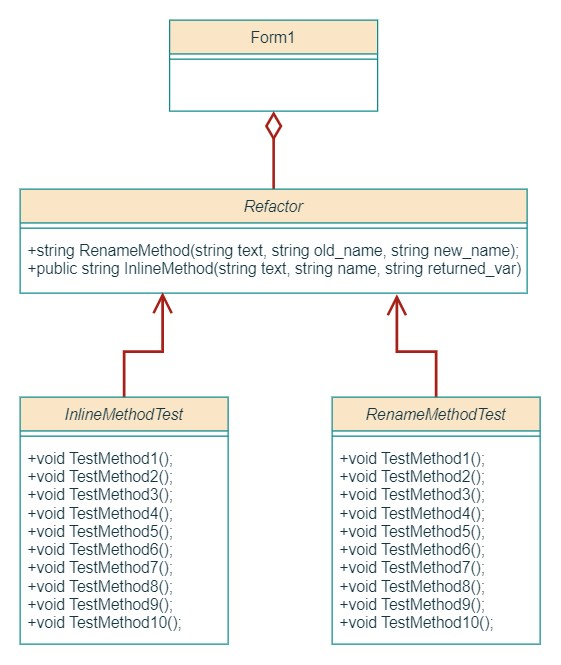


Рис. 1. – Узагальнена діаграма класів

1. **Модульні тести, детально охоплюючі функціональні особливості**

InlineMethod:

[TestMethod] //--------------------------------------------------------------------  
public void TestMethod1() {

Refactor refactor = new Refactor();

string method\_call = "math.plus(6, 4)";

string text =

"#include <iostream>\r\n" +

"class MathOperation { public:\r\n" +

" int plus(int first, int second) {\r\n" +

" return first + second;}\r\n" +

"};using namespace std;\r\n" +

"int main() {\r\n" +

" MathOperation math;\r\n" +

" int n = math.plus(6, 4);\r\n" +

" cout << n;\r\n" +

" return 0;\r\n" +

"}";

string returned\_var = "t";

string res =

"#include <iostream>\r\n" +

"class MathOperation { public:\r\n" +

" int plus(int first, int second) {\r\n" +

" return first + second;}\r\n" +

"};using namespace std;\r\n" +

"int main() {\r\n" +

" MathOperation math;\r\n" +

" //MInline: math.plus(6, 4)\r\n" +

" int t = 6 + 4;\r\n" +

" int n = t;\r\n" +

" cout << n;\r\n" +

" return 0;\r\n" +

"}";

Assert.AreEqual(res, refactor.InlineMethod(method\_call, text, returned\_var));

}

[TestMethod] //--------------------------------------------------------------------

public void TestMethod2() {

Refactor refactor = new Refactor();

string method\_call = "math.mult(8, 2)";

string text =

"#include <iostream>\r\n" +

"class MathOperation { public:\r\n" +

" int plus(int first, int second) {\r\n" +

" return first + second;}\r\n" +

" int minus(int first, int second) {\r\n" +

" return first - second;}\r\n" +

" int mult(int first, int second) {\r\n" +

" return first \* second;}\r\n" +

" int div(int first, int second) {\r\n" +

" return first / second;}\r\n" +

"};using namespace std;\r\n" +

"int main() {\r\n" +

" MathOperation math;\r\n" +

" int n = math.mult(8, 2);\r\n" +

" cout << n;\r\n" +

" return 0;\r\n" +

"}";

string returned\_var = "mult\_res";

string res =

"#include <iostream>\r\n" +

"class MathOperation { public:\r\n" +

" int plus(int first, int second) {\r\n" +

" return first + second;}\r\n" +

" int minus(int first, int second) {\r\n" +

" return first - second;}\r\n" +

" int mult(int first, int second) {\r\n" +

" return first \* second;}\r\n" +

" int div(int first, int second) {\r\n" +

" return first / second;}\r\n" +

"};using namespace std;\r\n" +

"int main() {\r\n" +

" MathOperation math;\r\n" +

" //MInline: math.mult(8, 2)\r\n" +

" int mult\_res = 8 \* 2;\r\n" +

" int n = mult\_res;\r\n" +

" cout << n;\r\n" +

" return 0;\r\n" +

"}";

Assert.AreEqual(res, refactor.InlineMethod(method\_call, text, returned\_var));

}

[TestMethod] //--------------------------------------------------------------------

public void TestMethod3() {

Refactor refactor = new Refactor();

string method\_call = "math.div(3, 1)";

string text =

"#include <iostream>\r\n" +

"class MathOperation { public:\r\n" +

" int plus(int first, int second) {\r\n" +

" return first + second;}\r\n" +

" int minus(int first, int second) {\r\n" +

" return first - second;}\r\n" +

" int mult(int first, int second) {\r\n" +

" return first \* second;}\r\n" +

" int div(int first, int second) {\r\n" +

" return first / second;}\r\n" +

"};using namespace std;\r\n" +

"int main() {\r\n" +

" MathOperation math;\r\n" +

" int n = math.mult(8, 2) + math.div(3, 1);\r\n" +

" cout << n;\r\n" +

" return 0;\r\n" +

"}";

string returned\_var = "res\_div";

string res =

"#include <iostream>\r\n" +

"class MathOperation { public:\r\n" +

" int plus(int first, int second) {\r\n" +

" return first + second;}\r\n" +

" int minus(int first, int second) {\r\n" +

" return first - second;}\r\n" +

" int mult(int first, int second) {\r\n" +

" return first \* second;}\r\n" +

" int div(int first, int second) {\r\n" +

" return first / second;}\r\n" +

"};using namespace std;\r\n" +

"int main() {\r\n" +

" MathOperation math;\r\n" +

" //MInline: math.div(3, 1)\r\n" +

" int res\_div = 3 / 1;\r\n" +

" int n = math.mult(8, 2) + res\_div;\r\n" +

" cout << n;\r\n" +

" return 0;\r\n" +

"}";

Assert.AreEqual(res, refactor.InlineMethod(method\_call, text, returned\_var));

}

[TestMethod] //--------------------------------------------------------------------

public void TestMethod4() {

Refactor refactor = new Refactor();

string method\_call = "user.displayInfo()";

string text =

"#include <iostream>\r\n" +

"class User { // Клас \"Користувач\"\r\n" +

"public:\r\n" +

" User(std::string name, int age) : name(name), age(age) {}\r\n" +

" void displayInfo() {\r\n" +

" std::cout << \"Ім'я: \" << name << std::endl;\r\n" +

" std::cout << \"Вік: \" << age << \" років\" << std::endl;}\r\n" +

"private:\r\n" +

" std::string name;\r\n" +

" int age;\r\n" +

"};\r\n" +

"int main() {\r\n" +

" User user(\"Іван\", 30);\r\n" +

" std::cout << \"Інформація про користувача:\" << std::endl;\r\n" +

" user.displayInfo();\r\n" +

" return 0;\r\n" +

"}";

string returned\_var = "\_";

string res =

"Worning! Not allowed to inline method.\n Thet methot use private class variable";

Assert.AreEqual(res, refactor.InlineMethod(method\_call, text, returned\_var));

}

[TestMethod] //--------------------------------------------------------------------

public void TestMethod5() {

Refactor refactor = new Refactor();

string method\_call = "user(\"Іван\", 30)";

string text =

"#include <iostream>\r\n" +

"class User { // Клас \"Користувач\"\r\n" +

"public:\r\n" +

" User(std::string name, int age) : name(name), age(age) {}\r\n" +

" void displayInfo() {\r\n" +

" std::cout << \"Ім'я: \" << name << std::endl;\r\n" +

" std::cout << \"Вік: \" << age << \" років\" << std::endl;}\r\n" +

"private:\r\n" +

" std::string name;\r\n" +

" int age;\r\n" +

"};\r\n" +

"int main() {\r\n" +

" User user(\"Іван\", 30);\r\n" +

" std::cout << \"Інформація про користувача:\" << std::endl;\r\n" +

" user.displayInfo();\r\n" +

" return 0;\r\n" +

"}";

string returned\_var = "\_";

string res =

"Worning! Not allowed to inline method.\n Thet methot is class constructor";

Assert.AreEqual(res, refactor.InlineMethod(method\_call, text, returned\_var));

}

[TestMethod] //--------------------------------------------------------------------

public void TestMethod6() {

Refactor refactor = new Refactor();

string method\_call = "car.displayInfo()";

string text =

"#include <iostream>\r\n" +

"class Car { // Клас Автомобіль\r\n" +

"public:\r\n" +

" Car(std::string make, std::string model) : make(make), model(model) {}\r\n" +

" void displayInfo() {\r\n" +

" std::cout << \"Марка: \" << make << std::endl;\r\n" +

" std::cout << \"Модель: \" << model << std::endl;}\r\n" +

" std::string make;\r\n" +

" std::string model;\r\n" +

"};\r\n" +

"int main() {\r\n" +

" Car car(\"Toyota\", \"Camry\");\r\n" +

" std::cout << \"\\nІнформація про автомобіль:\" << std::endl;\r\n" +

" car.displayInfo();\r\n" +

" return 0;\r\n" +

"}";

string returned\_var = "\_";

string res =

"#include <iostream>\r\n" +

"class Car { // Клас Автомобіль\r\n" +

"public:\r\n" +

" Car(std::string make, std::string model) : make(make), model(model) {}\r\n" +

" void displayInfo() {\r\n" +

" std::cout << \"Марка: \" << make << std::endl;\r\n" +

" std::cout << \"Модель: \" << model << std::endl;}\r\n" +

" std::string make;\r\n" +

" std::string model;\r\n" +

"};\r\n" +

"int main() {\r\n" +

" Car car(\"Toyota\", \"Camry\");\r\n" +

" std::cout << \"\\nІнформація про автомобіль:\" << std::endl;\r\n" +

" //MInline: car.displayInfo();\r\n" +

" std::cout << \"Марка: \" << car.make << std::endl;\r\n" +

" std::cout << \"Модель: \" << car.model << std::endl;\r\n" +

" return 0;\r\n" +

"}";

Assert.AreEqual(res, refactor.InlineMethod(method\_call, text, returned\_var));

}

[TestMethod] //--------------------------------------------------------------------

public void TestMethod7() {

Refactor refactor = new Refactor();

string method\_call = "c1.Subtract(p, q)";

string text =

"class Class1 { public:\r\n" +

" int Add(int a, int b) {\r\n" +

" return a + b;}\r\n" +

" int Subtract(int a, int b) {\r\n" +

" return a - b;}\r\n" +

"};\r\n" +

"class Class2 { public:\r\n" +

" Class1 c1;\r\n" +

" int Calculate(int x, int y) {\r\n" +

" return x + c1.Add(x, y);}\r\n" +

" int Compute(int p, int q) {\r\n" +

" return p - c1.Subtract(p, q);}\r\n" +

"};\r\n" +

"int main() {\r\n" +

" Class2 c2;\r\n" +

" int result1 = c2.Calculate(10, 5);\r\n" +

" int result2 = c2.Compute(20, 8);\r\n" +

" std::cout << \"Result 1: \" << result1 << std::endl;\r\n" +

" std::cout << \"Result 2: \" << result2 << std::endl;\r\n" +

" return 0;\r\n" +

"}";

string returned\_var = "res";

string res =

"class Class1 { public:\r\n" +

" int Add(int a, int b) {\r\n" +

" return a + b;}\r\n" +

" int Subtract(int a, int b) {\r\n" +

" return a - b;}\r\n" +

"};\r\n" +

"class Class2 { public:\r\n" +

" Class1 c1;\r\n" +

" int Calculate(int x, int y) {\r\n" +

" return x + c1.Add(x, y);}\r\n" +

" int Compute(int p, int q) {\r\n" +

" //MInline: c1.Subtract(p, q);\r\n" +

" int res = p - q;\r\n" +

" return p - res;}\r\n" +

"};\r\n" +

"int main() {\r\n" +

" Class2 c2;\r\n" +

" int result1 = c2.Calculate(10, 5);\r\n" +

" int result2 = c2.Compute(20, 8);\r\n" +

" std::cout << \"Result 1: \" << result1 << std::endl;\r\n" +

" std::cout << \"Result 2: \" << result2 << std::endl;\r\n" +

" return 0;\r\n" +

"}";

Assert.AreEqual(res, refactor.InlineMethod(method\_call, text, returned\_var));

}

[TestMethod] //--------------------------------------------------------------------

public void TestMethod8() {

Refactor refactor = new Refactor();

string method\_call = "c2.Calculate(10, 5)";

string text =

"class Class1 { public:\r\n" +

" int Add(int a, int b) {\r\n" +

" return a + b;}\r\n" +

" int Subtract(int a, int b) {\r\n" +

" return a - b;}\r\n" +

"};\r\n" +

"class Class2 { public:\r\n" +

" Class1 c1;\r\n" +

" int Calculate(int x, int y) {\r\n" +

" return x + c1.Add(x, y);}\r\n" +

" int Compute(int p, int q) {\r\n" +

" return p - c1.Subtract(p, q);}\r\n" +

"};\r\n" +

"int main() {\r\n" +

" Class2 c2;\r\n" +

" int result1 = c2.Calculate(10, 5);\r\n" +

" int result2 = c2.Compute(20, 8);\r\n" +

" std::cout << \"Result 1: \" << result1 << std::endl;\r\n" +

" std::cout << \"Result 2: \" << result2 << std::endl;\r\n" +

" return 0;\r\n" +

"}";

string returned\_var = "res";

string res =

"class Class1 {public:\r\n" +

" int Add(int a, int b) {\r\n" +

" return a + b;}\r\n" +

" int Subtract(int a, int b) {\r\n" +

" return a - b;}\r\n" +

"};\r\n" +

"class Class2 { public:\r\n" +

" Class1 c1;\r\n" +

" int Calculate(int x, int y) {\r\n" +

" return x + c1.Add(x, y);}\r\n" +

" int Compute(int p, int q) {\r\n" +

" return p - c1.Subtract(p, q);}\r\n" +

"};\r\n" +

"int main() {\r\n" +

" Class2 c2;\r\n" +

" //MInline: c2.Calculate(10, 5);\r\n" +

" int res = 10 + c2.c1.Add(10, 5);\r\n" +

" int result1 = res;\r\n" +

" int result2 = c2.Compute(20, 8);\r\n" +

" std::cout << \"Result 1: \" << result1 << std::endl;\r\n" +

" std::cout << \"Result 2: \" << result2 << std::endl;\r\n" +

" return 0;\r\n" +

"}";

Assert.AreEqual(res, refactor.InlineMethod(method\_call, text, returned\_var));

}

[TestMethod] //--------------------------------------------------------------------

public void TestMethod9() {

Refactor refactor = new Refactor();

string method\_call = "IsPrime(i)";

string text =

"class DataProcessor { public:\r\n" +

" int ProcessData(int data) {\r\n" +

" int result = 0;\r\n" +

" if (data > 0) for (int i = 1; i <= data; ++i) {\r\n" +

" if (IsPrime(i)) result += i;}\r\n" +

" else std::cerr << \"Error: Invalid input data.\" << std::endl;\r\n" +

" return result;}\r\n" +

" bool IsPrime(int number) {\r\n" +

" if (number <= 1) return false; \r\n" +

" for (int i = 2; i \* i <= number; ++i) \r\n" +

" if (number % i == 0) return false;\r\n" +

" return true;}\r\n" +

"};\r\n" +

"int main() {\r\n" +

" DataProcessor processor;\r\n" +

" int inputData = 10;\r\n" +

" int result = processor.ProcessData(inputData);\r\n" +

" std::cout << \"Result: \" << result << std::endl;\r\n" +

" return 0;\r\n" +

"}";

string returned\_var = "boolRes";

string res =

"class DataProcessor { public:\r\n" +

" int ProcessData(int data) {\r\n" +

" int result = 0;\r\n" +

" if (data > 0) for (int i = 1; i <= data; ++i) {\r\n" +

" //MInline: IsPrime(i);\r\n" +

" bool boolRes; {\r\n" +

" if (i <= 1) boolRes = false;\r\n" +

" for (int i\_ = 2; i \* i\_ <= i; ++i\_)\r\n" +

" if (i % i\_ == 0) boolRes = false;\r\n" +

" boolRes = true;}\r\n" +

" if (boolRes) result += i;}\r\n" +

" else std::cerr << \"Error: Invalid input data.\" << std::endl;\r\n" +

" return result;}\r\n" +

" bool IsPrime(int number) {\r\n" +

" if (number <= 1) return false; \r\n" +

" for (int i = 2; i \* i <= number; ++i) \r\n" +

" if (number % i == 0) return false;\r\n" +

" return true;}\r\n" +

"};\r\n" +

"int main() {\r\n" +

" DataProcessor processor;\r\n" +

" int inputData = 10;\r\n" +

" int result = processor.ProcessData(inputData);\r\n" +

" std::cout << \"Result: \" << result << std::endl;\r\n" +

" return 0;\r\n" +

"}";

Assert.AreEqual(res, refactor.InlineMethod(method\_call, text, returned\_var));

}

[TestMethod] //--------------------------------------------------------------------

public void TestMethod10() {

Refactor refactor = new Refactor();

string method\_call = "processor.ProcessData(inputData)";

string text =

"class DataProcessor { public:\r\n" +

" int ProcessData(int data) {\r\n" +

" int result = 0;\r\n" +

" if (data > 0) for (int i = 1; i <= data; ++i) {\r\n" +

" if (IsPrime(i)) result += i;}\r\n" +

" else std::cerr << \"Error: Invalid input data.\" << std::endl;\r\n" +

" return result;}\r\n" +

" bool IsPrime(int number) {\r\n" +

" if (number <= 1) return false; \r\n" +

" for (int i = 2; i \* i <= number; ++i) \r\n" +

" if (number % i == 0) return false;\r\n" +

" return true;}\r\n" +

"};\r\n" +

"int main() {\r\n" +

" DataProcessor processor;\r\n" +

" int inputData = 10;\r\n" +

" int result = processor.ProcessData(inputData);\r\n" +

" std::cout << \"Result: \" << result << std::endl;\r\n" +

" return 0;\r\n" +

"}";

string returned\_var = "res";

string res =

"class DataProcessor { public:\r\n" +

" int ProcessData(int data) {\r\n" +

" int result = 0;\r\n" +

" if (data > 0) for (int i = 1; i <= data; ++i) {\r\n" +

" if (IsPrime(i)) result += i;}\r\n" +

" else std::cerr << \"Error: Invalid input data.\" << std::endl;\r\n" +

" return result;}\r\n" +

" bool IsPrime(int number) {\r\n" +

" if (number <= 1) return false; \r\n" +

" for (int i = 2; i \* i <= number; ++i) \r\n" +

" if (number % i == 0) return false;\r\n" +

" return true;}\r\n" +

"};\r\n" +

"int main() {\r\n" +

" DataProcessor processor;\r\n" +

" int inputData = 10;\r\n" +

" //MInline: processor.ProcessData(inputData);\r\n" +

" int res; {int result\_ = 0;\r\n" +

" if (inputData > 0) for (int i = 1; i <= inputData; ++i) \r\n" +

" if (processor.IsPrime(i)) result\_ += i;\r\n" +

" else std::cerr << \"Error: Invalid input data.\" << std::endl;\r\n" +

" res = result\_;}\r\n" +

" int result = res;\r\n" +

" std::cout << \"Result: \" << result << std::endl;\r\n" +

" return 0;\r\n" +

"}";

Assert.AreEqual(res, refactor.InlineMethod(method\_call, text, returned\_var));

}

RenameMethod:

[TestMethod] //--------------------------------------------------------------------

public void TestMethod1()

{

Refactor refactor = new Refactor();

string old\_name = "bubbleSort";

string new\_name = "ballSort";

string text =

"#include <iostream>\r\n" +

"void bubbleSort(int arr[], int n) {\r\n" +

" for (int i = 0; i < n - 1; i++) {\r\n" +

" for (int j = 0; j < n - i - 1; j++) {\r\n" +

" if (arr[j] > arr[j + 1]) {\r\n" +

" int temp = arr[j];\r\n" +

" arr[j] = arr[j + 1];\r\n" +

" arr[j + 1] = temp;\r\n" +

" }\r\n" +

" }\r\n" +

" }\r\n" +

"}";

string res =

"#include <iostream>\r\n" +

"void ballSort(int arr[], int n) {\r\n" +

" for (int i = 0; i < n - 1; i++) {\r\n" +

" for (int j = 0; j < n - i - 1; j++) {\r\n" +

" if (arr[j] > arr[j + 1]) {\r\n" +

" int temp = arr[j];\r\n" +

" arr[j] = arr[j + 1];\r\n" +

" arr[j + 1] = temp;\r\n" +

" }\r\n" +

" }\r\n" +

" }\r\n" +

"}";

Assert.AreEqual(res, refactor.RenameMethod(text, old\_name, new\_name));

}

[TestMethod] //--------------------------------------------------------------------

public void TestMethod2()

{

Refactor refactor = new Refactor();

string old\_name = "Add";

string new\_name = "Plus";

string text =

"#include <iostream>\r\n" +

"class MathUtils {\r\n" +

"public:\r\n" +

" static int Add(int a, int b) {\r\n" +

" return a + b;\r\n" +

" }\r\n" +

"int main() {\r\n" +

" int sum = MathUtils::Add(5, 3);\r\n" +

" std::cout << \"5 + 3 = \" << sum << std::endl;\r\n" +

" return 0;\r\n" +

"}";

string res =

"#include <iostream>\r\n" +

"class MathUtils {\r\n" +

"public:\r\n" +

" static int Plus(int a, int b) {\r\n" +

" return a + b;\r\n" +

" }\r\n" +

"int main() {\r\n" +

" int sum = MathUtils::Plus(5, 3);\r\n" +

" std::cout << \"5 + 3 = \" << sum << std::endl;\r\n" +

" return 0;\r\n" +

"}";

Assert.AreEqual(res, refactor.RenameMethod(text, old\_name, new\_name));

}

[TestMethod] //--------------------------------------------------------------------

public void TestMethod3()

{

Refactor refactor = new Refactor();

string old\_name = "popUpSort";

string new\_name = "bubbleSort";

string text =

"#include <iostream>\r\n" +

" popUpSort(mass, 100);\r\n" +

" for (int i = 0; i < n; i++) {\r\n" +

" std::cout << arr[i] << \" \";\r\n" +

" }\r\n" +

" return 0;\r\n" +

"}";

string res =

"#include <iostream>\r\n" +

" bubbleSort(arr, n);\r\n" +

" for (int i = 0; i < n; i++) {\r\n" +

" std::cout << arr[i] << \" \";\r\n" +

" }\r\n" +

" return 0;\r\n" +

"}";

Assert.AreEqual(res, refactor.RenameMethod(text, old\_name, new\_name));

}

[TestMethod] //--------------------------------------------------------------------

public void TestMethod4()

{

Refactor refactor = new Refactor();

string old\_name = "Square";

string new\_name = "Rectangle";

string text =

"#include <iostream>\r\n" +

"class MathUtils {\r\n" +

"public:\r\n" +

" static double Square(double x) {\r\n" +

" return x \* x;\r\n" +

" }\r\n" +

"};\r\n" +

"int main() {\r\n" +

" double square = MathUtils::Square(4.0);\r\n" +

" std::cout << \"Square() of 4.0 is \" << square << std::endl;\r\n" +

" return 0;\r\n" +

"}";

string res =

"#include <iostream>\r\n" +

"class MathUtils {\r\n" +

"public:\r\n" +

" static double Rectangle(double x) {\r\n" +

" return x \* x;\r\n" +

" }\r\n" +

"};\r\n" +

"int main() {\r\n" +

" double square = MathUtils::Rectangle(4.0);\r\n" +

" std::cout << \"Rectangle() of 4.0 is \" << square << std::endl;\r\n" +

" return 0;\r\n" +

"}";

Assert.AreEqual(res, refactor.RenameMethod(text, old\_name, new\_name));

}

[TestMethod] //--------------------------------------------------------------------

public void TestMethod5()

{

Refactor refactor = new Refactor();

string old\_name = "RemoveItem";

string new\_name = "DeleteItem";

string text =

"#include <iostream>\r\n" +

"#include <vector>\r\n" +

"class ShoppingCart {\r\n" +

"public:\r\n" +

" void AddItem(const std::string& item) {\r\n" +

" items\_.push\_back(item);\r\n" +

" }\r\n" +

" void RemoveItem(const std::string& item) {\r\n" +

" for (auto it = items\_.begin(); it != items\_.end(); ++it) {\r\n" +

" if (\*it == item) {\r\n" +

" items\_.erase(it);\r\n" +

" break;\r\n" +

" }\r\n" +

" }\r\n" +

" }\r\n" +

" void DisplayCart() {\r\n" +

" std::cout << \"Shopping Cart Contents:\" << std::endl;\r\n" +

" for (const std::string& item : items\_) {\r\n" +

" std::cout << \"- \" << item << std::endl;\r\n" +

" }\r\n" +

" }\r\n" +

"private:\r\n" +

" std::vector<std::string> items\_;\r\n" +

"};\r\n" +

"int main() {\r\n" +

" ShoppingCart myCart;\r\n" +

" myCart.AddItem(\"Item 1\");\r\n" +

" myCart.AddItem(\"Item 2\");\r\n" +

" myCart.AddItem(\"Item 3\");\r\n" +

" myCart.DisplayCart();\r\n" +

" myCart.RemoveItem(\"Item 2\");\r\n" +

" std::cout << \"After removing Item 2:\" << std::endl;\r\n" +

" myCart.DisplayCart();\r\n" +

" return 0;\r\n" +

"}";

string res =

"#include <iostream>\r\n" +

"#include <vector>\r\n" +

"class ShoppingCart {\r\n" +

"public:\r\n" +

" void AddItem(const std::string& item) {\r\n" +

" items\_.push\_back(item);\r\n" +

" }\r\n" +

" void DeleteItem(const std::string& item) {\r\n" +

" for (auto it = items\_.begin(); it != items\_.end(); ++it) {\r\n" +

" if (\*it == item) {\r\n" +

" items\_.erase(it);\r\n" +

" break;\r\n" +

" }\r\n" +

" }\r\n" +

" }\r\n" +

" void DisplayCart() {\r\n" +

" std::cout << \"Shopping Cart Contents:\" << std::endl;\r\n" +

" for (const std::string& item : items\_) {\r\n" +

" std::cout << \"- \" << item << std::endl;\r\n" +

" }\r\n" +

" }\r\n" +

"private:\r\n" +

" std::vector<std::string> items\_;\r\n" +

"};\r\n" +

"int main() {\r\n" +

" ShoppingCart myCart;\r\n" +

" myCart.AddItem(\"Item 1\");\r\n" +

" myCart.AddItem(\"Item 2\");\r\n" +

" myCart.AddItem(\"Item 3\");\r\n" +

" myCart.DisplayCart();\r\n" +

" myCart.DeleteItem(\"Item 2\");\r\n" +

" std::cout << \"After removing Item 2:\" << std::endl;\r\n" +

" myCart.DisplayCart();\r\n" +

" return 0;\r\n" +

"}";

Assert.AreEqual(res, refactor.RenameMethod(text, old\_name, new\_name));

}

[TestMethod] //--------------------------------------------------------------------

public void TestMethod6()

{

Refactor refactor = new Refactor();

string old\_name = "insertionSort";

string new\_name = "injectionSort";

string text =

"#include <iostream>\r\n" +

"void insertionSort(int arr[], int n) {\r\n" +

" for (int i = 1; i < n; i++) {\r\n" +

" int key = arr[i];\r\n" +

" int j = i - 1;\r\n" +

" while (j >= 0 && arr[j] > key) {\r\n" +

" arr[j + 1] = arr[j];\r\n" +

" j--;\r\n" +

" }\r\n" +

" arr[j + 1] = key;\r\n" +

" }\r\n" +

"}";

string res =

"#include <iostream>\r\n" +

"void injectionSort(int arr[], int n) {\r\n" +

" for (int i = 1; i < n; i++) {\r\n" +

" int key = arr[i];\r\n" +

" int j = i - 1;\r\n" +

" while (j >= 0 && arr[j] > key) {\r\n" +

" arr[j + 1] = arr[j];\r\n" +

" j--;\r\n" +

" }\r\n" +

" arr[j + 1] = key;\r\n" +

" }\r\n" +

"}";

Assert.AreEqual(res, refactor.RenameMethod(text, old\_name, new\_name));

}

[TestMethod] //--------------------------------------------------------------------

public void TestMethod7()

{

Refactor refactor = new Refactor();

string old\_name = "displayInfo";

string new\_name = "showInfo";

string text =

"#include <iostream>\r\n" +

"#include <string>\r\n" +

"class Student {\r\n" +

"public:\r\n" +

" Student(std::string firstName, std::string lastName, int age, double gpa)\r\n" +

" : firstName\_(firstName), lastName\_(lastName), age\_(age), gpa\_(gpa) {\r\n" +

" }\r\n" +

" void displayInfo() {\r\n" +

" std::cout << \"Ім'я: \" << firstName\_ << \" \" << lastName\_ << std::endl;\r\n" +

" std::cout << \"Вік: \" << age\_ << \" років\" << std::endl;\r\n" +

" std::cout << \"Середній бал: \" << gpa\_ << std::endl;\r\n" +

" }\r\n" +

"private:\r\n" +

" std::string firstName\_;\r\n" +

" std::string lastName\_;\r\n" +

" int age\_;\r\n" +

" double gpa\_;\r\n" +

"};";

string res =

"#include <iostream>\r\n" +

"#include <string>\r\n" +

"class Student {\r\n" +

"public:\r\n" +

" Student(std::string firstName, std::string lastName, int age, double gpa)\r\n" +

" : firstName\_(firstName), lastName\_(lastName), age\_(age), gpa\_(gpa) {\r\n" +

" }\r\n" +

" void showInfo() {\r\n" +

" std::cout << \"Ім'я: \" << firstName\_ << \" \" << lastName\_ << std::endl;\r\n" +

" std::cout << \"Вік: \" << age\_ << \" років\" << std::endl;\r\n" +

" std::cout << \"Середній бал: \" << gpa\_ << std::endl;\r\n" +

" }\r\n" +

"private:\r\n" +

" std::string firstName\_;\r\n" +

" std::string lastName\_;\r\n" +

" int age\_;\r\n" +

" double gpa\_;\r\n" +

"};";

Assert.AreEqual(res, refactor.RenameMethod(text, old\_name, new\_name));

}

[TestMethod] //--------------------------------------------------------------------

public void TestMethod8()

{

Refactor refactor = new Refactor();

string old\_name = "add";

string new\_name = "append";

string text =

"#include <iostream>\r\n" +

"double add(double a, double b) {\r\n" +

" return a + b;\r\n" +

"}\r\n" +

"int main() {\r\n" +

" double num1, num2;\r\n" +

" std::cout << \"Введіть перше число: \";\r\n" +

" std::cin >> num1;\r\n" +

" std::cout << \"Введіть друге число: \";\r\n" +

" std::cin >> num2;\r\n" +

" double sum = add(num1, num2);\r\n" +

" std::cout << \"Сума чисел \" << num1 << \" і \" << num2 << \" дорівнює \" << sum << std::endl;\r\n" +

" return 0;\r\n" +

"}";

string res =

"#include <iostream>\r\n" +

"double append(double a, double b) {\r\n" +

" return a + b;\r\n" +

"}\r\n" +

"int main() {\r\n" +

" double num1, num2;\r\n" +

" std::cout << \"Введіть перше число: \";\r\n" +

" std::cin >> num1;\r\n" +

" std::cout << \"Введіть друге число: \";\r\n" +

" std::cin >> num2;\r\n" +

" double sum = append(num1, num2);\r\n" +

" std::cout << \"Сума чисел \" << num1 << \" і \" << num2 << \" дорівнює \" << sum << std::endl;\r\n" +

" return 0;\r\n" +

"}";

Assert.AreEqual(res, refactor.RenameMethod(text, old\_name, new\_name));

}

[TestMethod] //--------------------------------------------------------------------

public void TestMethod9()

{

Refactor refactor = new Refactor();

string old\_name = "raiseSalary";

string new\_name = "increaseSalary";

string text =

"#include <iostream>\r\n" +

"#include <string>\r\n" +

"class Employee {\r\n" +

"public:\r\n" +

" Employee(std::string name, int id, double salary)\r\n" +

" : name\_(name), id\_(id), salary\_(salary) {\r\n" +

" }\r\n" +

" void displayInfo() {\r\n" +

" std::cout << \"Працівник: \" << name\_ << std::endl;\r\n" +

" std::cout << \"ID: \" << id\_ << std::endl;\r\n" +

" std::cout << \"Зарплата: \" << salary\_ << \" грн\" << std::endl;\r\n" +

" }\r\n" +

" void raiseSalary(double percent) {\r\n" +

" salary\_ \*= (1 + percent / 100);\r\n" +

" }\r\n" +

"private:\r\n" +

" std::string name\_;\r\n" +

" int id\_;\r\n" +

" double salary\_;\r\n" +

"};\r\n" +

"int main() {\r\n" +

" Employee employee(\"Іван Петров\", 101, 5000.0);\r\n" +

" std::cout << \"Інформація про працівника:\" << std::endl;\r\n" +

" employee.displayInfo();\r\n" +

" employee.raiseSalary(10);\r\n" +

" std::cout << \"\\nОновлена інформація про працівника:\" << std::endl;\r\n" +

" employee.displayInfo();\r\n" +

" return 0;\r\n" +

"}";

string res =

"#include <iostream>\r\n" +

"#include <string>\r\n" +

"class Employee {\r\n" +

"public:\r\n" +

" Employee(std::string name, int id, double salary)\r\n" +

" : name\_(name), id\_(id), salary\_(salary) {\r\n" +

" }\r\n" +

" void displayInfo() {\r\n" +

" std::cout << \"Працівник: \" << name\_ << std::endl;\r\n" +

" std::cout << \"ID: \" << id\_ << std::endl;\r\n" +

" std::cout << \"Зарплата: \" << salary\_ << \" грн\" << std::endl;\r\n" +

" }\r\n" +

" void increaseSalary(double percent) {\r\n" +

" salary\_ \*= (1 + percent / 100);\r\n" +

" }\r\n" +

"private:\r\n" +

" std::string name\_;\r\n" +

" int id\_;\r\n" +

" double salary\_;\r\n" +

"};\r\n" +

"int main() {\r\n" +

" Employee employee(\"Іван Петров\", 101, 5000.0);\r\n" +

" std::cout << \"Інформація про працівника:\" << std::endl;\r\n" +

" employee.displayInfo();\r\n" +

" employee.increaseSalary(10);\r\n" +

" std::cout << \"\\nОновлена інформація про працівника:\" << std::endl;\r\n" +

" employee.displayInfo();\r\n" +

" return 0;\r\n" +

"}";

Assert.AreEqual(res, refactor.RenameMethod(text, old\_name, new\_name));

}

[TestMethod] //--------------------------------------------------------------------

public void TestMethod10()

{

Refactor refactor = new Refactor();

string old\_name = "calculateFactorial";

string new\_name = "computeFactorial";

string text =

"#include <iostream>\r\n" +

"class FactorialCalculator {\r\n" +

"public:\r\n" +

" int calculateFactorial(int n) {\r\n" +

" if (n <= 0) {\r\n" +

" return 1;\r\n" +

" } else {\r\n" +

" int result = 1;\r\n" +

" for (int i = 1; i <= n; ++i) {\r\n" +

" result \*= i;\r\n" +

" }\r\n" +

" return result;\r\n" +

" }\r\n" +

" }\r\n" +

"};\r\n" +

"int main() {\r\n" +

" FactorialCalculator calculator;\r\n" +

" int num;\r\n" +

" std::cout << \"Введіть число: \";\r\n" +

" std::cin >> num;\r\n" +

" int factorial = calculator.calculateFactorial(num);\r\n" +

" std::cout << \"Факторіал числа \" << num << \" дорівнює \" << factorial << std::endl;\r\n" +

" return 0;\r\n" +

"}";

string res =

"#include <iostream>\r\n" +

"class FactorialCalculator {\r\n" +

"public:\r\n" +

" int computeFactorial(int n) {\r\n" +

" if (n <= 0) {\r\n" +

" return 1;\r\n" +

" } else {\r\n" +

" int result = 1;\r\n" +

" for (int i = 1; i <= n; ++i) {\r\n" +

" result \*= i;\r\n" +

" }\r\n" +

" return result;\r\n" +

" }\r\n" +

" }\r\n" +

"};\r\n" +

"int main() {\r\n" +

" FactorialCalculator calculator;\r\n" +

" int num;\r\n" +

" std::cout << \"Введіть число: \";\r\n" +

" std::cin >> num;\r\n" +

" int factorial = calculator.computeFactorial(num);\r\n" +

" std::cout << \"Факторіал числа \" << num << \" дорівнює \" << factorial << std::endl;\r\n" +

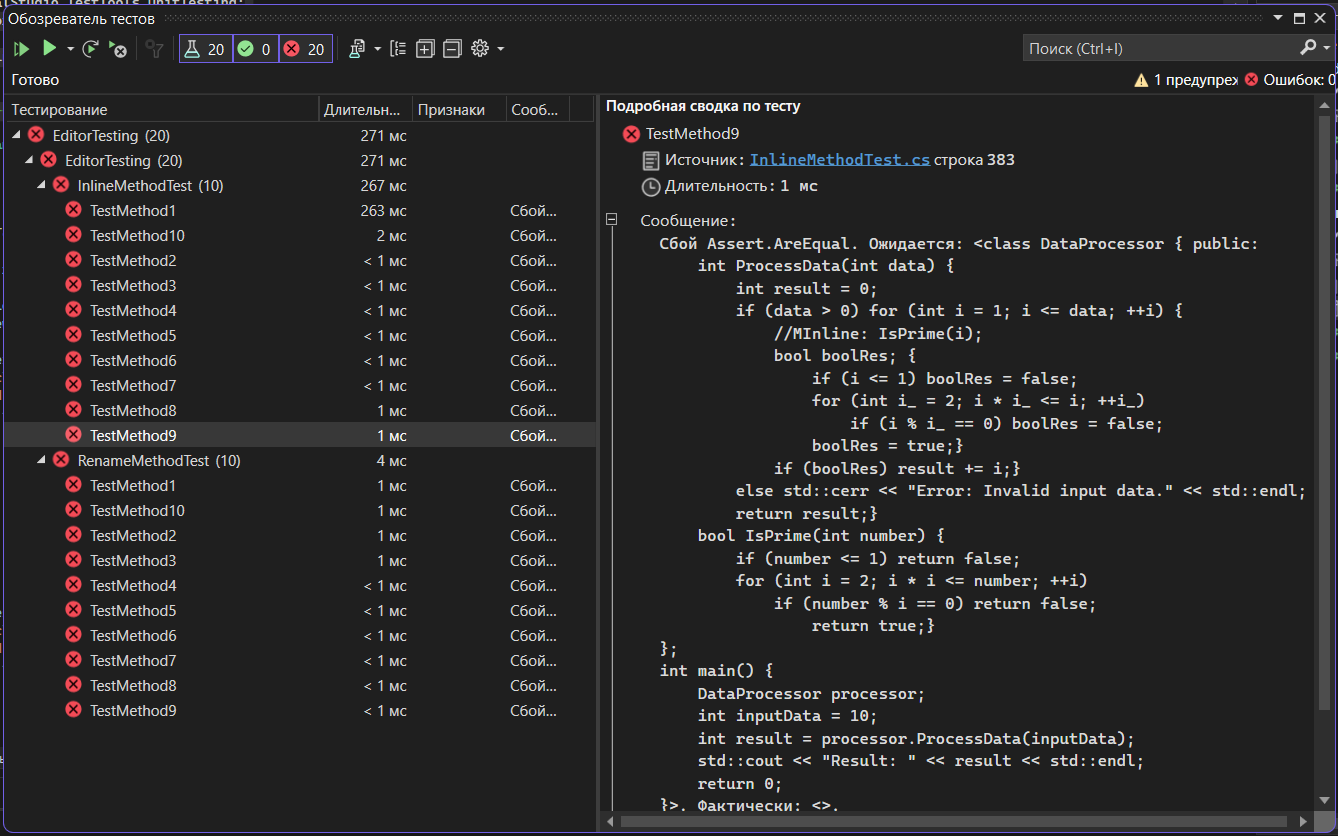
" return 0;\r\n" +

"}";

Assert.AreEqual(res, refactor.RenameMethod(text, old\_name, new\_name));

}

1. **Проходження складених тестів**

****

Жоден з тестів не пройшов вдало через відсутність реалізації функцій.

1. **Аналіз результатів. Висновки.**

Під час проведення лабораторної роботи було складено модульні тести, які надають можливість перевірити на правильність роботу функцій «Вбудування методів» та «Перейменування методів» програми текстового редактору. Складені тести були невдало пройдено через відсутність реалізації функцій, що перевіряються цими тестами. Також було складено концептуальну модель системи (узагальнену діаграму класів).

Використання тестових прикладів дозволяє виявити помилки та недоліки на ранніх етапах розробки, сприяє збереженню часу та ресурсів, а також поліпшує якість програмного продукту. Тестування також сприяє підтримці та рефакторингу коду, оскільки воно допомагає відслідковувати зміни та їх вплив на існуючий функціонал.

У висновку, використання тестових прикладів є важливим етапом у розробці програмного забезпечення. Вони дозволяють забезпечити надійність та стабільність програми, а також сприяють підвищенню продуктивності розробки.