**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ В. Н. КАРАЗІНА**

**ФАКУЛЬТЕТ КОМП’ЮТЕРНИХ НАУК**

**КАФЕДРА БЕЗПЕКИ ІНФОРМАЦІЙНИХ СИСТЕМ І ТЕХНОЛОГІЙ**

**Лабораторна робота №3**

***з навчальної дисципліни***

**«Математичні методи та технології тестування та верифікації програмного забезпечення»**

Виконала:

студентка групи КС-23 **Рузудженк С. Р.**

Перевірив:

доцент

**Нарєжній О. П.**

Харків – 2019

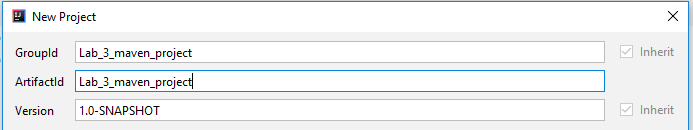
**Лабораторна робота №3**

з навчальної дисципліни

«Математичні методи та технології тестування та верифікації програмного забезпечення»

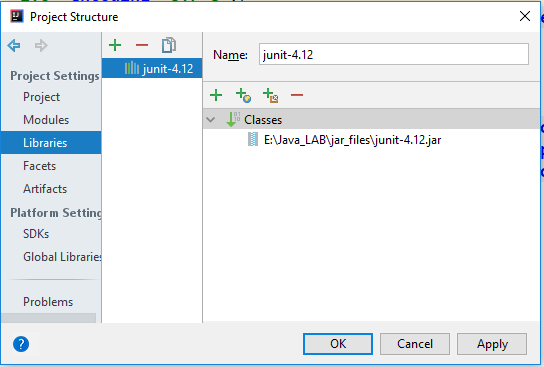
**Тема роботи:** Вивчення модульного тестування (unit testing) на прикладі фреймворку JUnit

**Мета роботи:** Вивчення фрейморку для модульного тестування JUnit, збірників проектів, написання автоматизованих тестів для програмного забезпечення.

**Хід роботи**

Запускаємо IntelliJ IDEA, створюємо новий проект.

Далі необхідно підключити бібліотеку JUnit.



Додаємо залежності до файлу pom.xml:

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

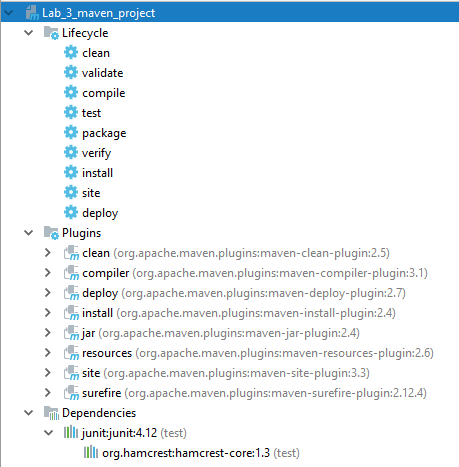
<scope>test</scope>

</dependency>

</dependencies>

Тепер необхідно підтягнути наші залежності. Для цього виконуємо таку послідовність дій:

*View→Tool Windows→Maven Project→клавіша* 



Далі створюємо консольну програму «Калькулятор»:

**public class** **Calculator** {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Calculator calculator = new Calculator ();

String operation;

System.out.println ("Input number A");

int A = scanner.nextInt ();

System.out.println ("Input number B");

int B = scanner.nextInt ();

do {

System.out.println("Choose operation \n+\t-\t\*\t\nenter 'e' for exit");

operation = scanner.next();

switch (operation) {

case "+":

System.out.println (calculator.Sum (A, B));

break;

case "-":

System.out.println (calculator.Difference (A, B));

break;

case "\*":

System.out.println (calculator.Multiple (A, B));

break;

case "/":

System.out.println (calculator.division (A, B));

break;

case "e":

System.exit (0);

break;

default:

System.out.println("Input error! Try again!");

}

}while(operation !="e");

}

public int Sum(int A, int B){

System.out.println ("Result: "+ (A+B));

return (A+B);

}

public double division(int A, int B){

System.out.println ("Result: "+ (double)(A/B));

return (A/B);

}

public int Multiple(int A, int B){

System.out.println ("Result: "+ (A\*B));

return (A\*B);

}

public int Difference(int A, int B){

System.out.println ("Result: "+ (A-B));

return (A-B);

}

}

Для перевірки роботи нашого калькулятора необхідно створити декілька тестів. Для цього використовуємо *Assert.assertEquals ("Failure", expectedResult, result, 0);* для того, щоб порівняти результат обчислення – дійсного та очікуваного. Також у разі невдачі ми забезпечуємо вивід повідомлення "Failure".

import org.junit.Assert;

import org.junit.Test;

**public class CalculatorTest** {

Calculator calculator = new Calculator ();

**@Test**

public void testDiv(){

double result = calculator.division (2,2);

double expectedResult = 1;

Assert.assertEquals ("Failure", expectedResult, result, 0);

}

**@Test**

public void testMult(){

int result = calculator.Multiple (2,2);

int expectedResult = 4;

Assert.assertEquals ("Failure", expectedResult, result, 0);

}

**@Test**

public void testSum(){

int result = calculator.Sum (2,2);

int expectedResult = 4;

Assert.assertEquals ("Failure", expectedResult, result, 0);

}

**@Test**

public void testDif(){

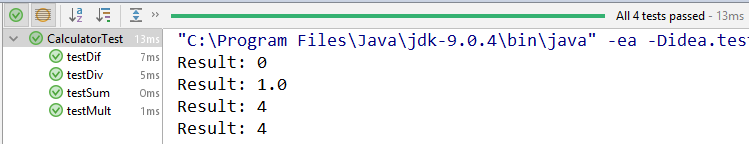
int result = calculator.Difference (2,2);

int expectedResult = 0;

Assert.assertEquals ("Failure", expectedResult, result, 0);

}

}

Бачимо, що результатом запуску класу *CalculatorTest* буде успішне виконання усіх чотирьох тестів.

Якщо ж ми, наприклад, змінимо у тесті *testDiv()* очікуваний результат з 1 на 3, то з’явиться наше повідомлення "Failure", що говорить про помилку.

**@Test**

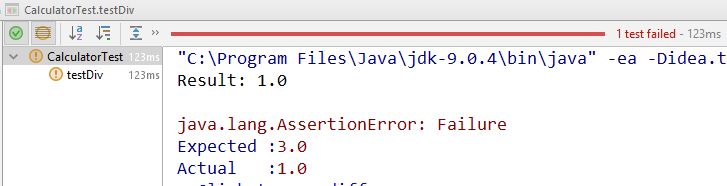
public void testDiv(){

double result = calculator.division (2,2);

double expectedResult = 3;

Assert.assertEquals ("Failure", expectedResult, result, 0);

}



**Висновки**

Отже, модульне тестування дозволяє проводити тести всієї програми або її окремих компонентів (units). Такий рівень тестування передбачає написання автоматизованих тестових сценаріїв, які виявляють дефекти на стадії розробки програмного застосунку і дозволяють розробнику виправити помилки відразу ж без їх занесення у базу дефектів. Бібліотека JUnit використовується для написання модульних тестів, що повторюються. Вона забезпечує перевантаження методів для всіх примітивних типів, об’єктів та масивів. Використовувані параметри: очікуване значення (expectedResult) та актуальне значення (result). Можливим також є вивід повідомлення у разі невдачі тесту.

Таким чином, у ході виконання лабораторної роботи було вивчено фрейморк для модульного тестування JUnit, збірників проектів, написання автоматизованих тестів для програмного забезпечення, а також написана програма «Калькулятор». У результаті роботи, отримано чотири тести, що успішно пройшли.