

Ministerul Educației, Culturii și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Departamentul Ingineria Software și Automatică



RAPORT

Lucrare de laborator Nr.2

Disciplina: Testarea Software

Tema: Testarea Unitara

A efectuat:

st.gr.TI-201FR Dascal Dumitru

A verificat :

conf. univ., dr. Prisăcaru Andrian

Chișinău 2024

Scopul: Realizarea sarcinilor de la Profesor, realizarea unei sarcini individuale.

Notiuni teoretice:

Testul unitar este o practică în dezvoltarea de software prin care se testează individual unități componente ale codului sursă. O unitate poate fi o funcție, o metodă sau chiar o clasă întreagă. Scopul testelor unitare este de a verifica că fiecare unitate funcționează așa cum este așteptat în mod izolat, adică fără a depinde de alte componente sau de mediul în care rulează. Aceste teste sunt de obicei automate și pot fi integrate într-un proces de construire continuă pentru a asigura calitatea și integritatea codului.

JUnit este un cadru (framework) de testare unitară pentru limbajul de programare Java. A fost creat pentru a simplifica procesul de scriere și rulare a testelor unitare în aplicațiile Java. JUnit furnizează un set de clase și metode predefinite care permit dezvoltatorilor să scrie și să execute teste pentru clasele și metodele lor Java.

Cu JUnit, dezvoltatorii pot defini teste pentru diferite funcționalități ale codului lor, precum și pentru a verifica comportamentul acestuia în diverse condiții și scenarii. Acest cadru facilitează automatizarea procesului de testare și furnizează rezultate clare și ușor de interpretat despre starea de sănătate a codului.

JUnit are o sintaxă simplă și intuitivă, permițând dezvoltatorilor să se concentreze mai mult pe scrierea testelor în sine decât pe detalii tehnice. De asemenea, este compatibil cu diferite unelte și medi de dezvoltare, ceea ce îl face un instrument popular în comunitatea dezvoltatorilor Java.

Testarea unitara fără utilizarea bibliotecilor:

```
build.gradle.kts (untitled) × mathclass.java CustomMath.java × C
1 public class CustomMath {
2
3     public static int sum(int x, int y){ 1 usage
4         return x+y; //возвращает результат сложения двух чисел
5     }
6     public static int division(int x, int y){ 1 usage
7         if (y==0) {
8             throw new IllegalArgumentException("dividen is 0");
9         } //бросается исключение
10        return(x/y); //возвращает результат деления
11    }
12
13
14    public static void main(String[] args) {
15        // реализация
16    }
17 }
18
✓ Tests passed: 1 of 1 test - 11 ms
> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test
```

Acest cod Java definește o clasă numită CustomMath. Această clasă conține două metode statice:

- `sum(int x, int y)` - metodă primește două argumente de tip întreg și returnează suma lor.
- `division(int x, int y)` - metodă primește două argumente de tip întreg și returnează rezultatul împărțirii primului argument la al doilea. Dacă al doilea argument este zero, metoda aruncă o excepție de tip `IllegalArgumentException` pentru a indica că nu se poate face împărțirea prin zero.

În metoda `main`, momentan nu este implementată nicio funcționalitate, dar este locul unde pot fi adăugate apeluri la metodele clasei sau alte acțiuni necesare pentru a testa funcționalitatea clasei `CustomMath`.

Sarcina 1

Creați un proiect cu clasa `CustomMath` de mai sus.

Omiteți din metoda main a clasei CustomMath verificarea funcției sum.

Omiteți din metoda testSum apelul metodei fail.

Asigurați-vă că testarea funcției sum trece pentru datele de intrare curente.

Adăugați în raport codul funcției testSum și rezultatul testării (rezultatele testelor PrtSc al ferestrei).

```
1 import org.junit.jupiter.api.Test;
2
3 import static org.junit.jupiter.api.Assertions.assertEquals;
4 import static org.junit.jupiter.api.Assertions.*;
5
6 public class CustomMathTest {
7
8     public CustomMathTest() {
9
10    }
11
12    /**
13     * Test of sum method, of class CustomMath.
14     */
15    @Test
16    public void testSum() {
17        System.out.println("sum");
18    }
19 }
```

Tests passed: 1 of 1 test – 11 ms

```
> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.5/userguide/command_line_interface.html#sec:command_line_warnings in the
BUILD SUCCESSFUL in 696ms
3 actionable tasks: 2 executed, 1 up-to-date
19:19:33: Execution finished ':test --tests "mathclass"'.

```

Sarcina 2

Omiteți verificarea metodei division din metoda main a clasei CustomMath.

Porniți testarea pentru $y=0$ și $y!=0$ (manual, schimbând secvențial valorile inițiale a lui y).

Plasați în raport codul testării și rezultatele testului testDivisionByZero pentru diferite valori ale lui y

```
build.gradle.kts (untitledtesting)  mathclass.java  CustomMath.java  CustomMathTest.java  testDivisionByZero.java  Main.j
```

```
1 import org.junit.jupiter.api.Test;
2
3 import static org.junit.jupiter.api.Assertions.assertEquals;
4 import static org.junit.jupiter.api.Assertions.*;
5
6 public class testDivisionByZero {
7     @Test
8     public void testDivisionByZero() {
9         int x = 0;
10        int y = 0;
11        int expResult = 0;
12        try {
13            int result=CustomMath.division(x, y);
14            assertEquals(expResult, result);
15            if(y==0) fail("Деление на ноли не создает исключительной ситуации");
16        }
17        catch(IllegalArgumentException e){
```

```
✓ Tests passed: 1 of 1 test – 11 ms

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.5/userguide/command\_line\_interface.html#sec:command\_line\_warnings in the
BUILD SUCCESSFUL in 676ms
3 actionable tasks: 2 executed, 1 up-to-date
19:25:21: Execution finished ':test --tests "mathclass"'.

```

Codul dat conține două fișiere, care conțin clase de testare pentru metoda division:

Fișierul testDivisionByZero.java: Acest fișier conține o clasă de testare cu același nume, care conține o metodă de test numită testDivisionByZero. În această metodă de test, se încearcă împărțirea a două numere întregi (x și y), unde y este setat inițial la zero. Se definește și un rezultat așteptat (expResult), care este tot zero în acest caz. Se testează și condiția pentru generarea excepției IllegalArgumentException în cazul în care y este zero, iar aceasta nu ar trebui să apară.

CustomMath:

```

1  public class CustomMath {
2
3      public static int sum(int x, int y){ no usages
4          return x+y; //возвращает результат сложения двух чисел
5      }
6
7      public static void main(String[] args) {
8          // реализация
9      }
10 }
11

```

Sarcina 3

Modificați metoda de testare testDivisionByZero (), astfel încât funcția să verifice împărțirea la zero, și de asemenea să furnizeze date de intrare corecte.

Includeți în raport varianta finală a claselor CustomMath și CustomMathTest, precum și PrtSc a rezultatului testării.

```

1  import org.junit.jupiter.params.ParameterizedTest;
2  import org.junit.jupiter.params.provider.ValueSource;
3  import static org.junit.jupiter.api.Assertions.*;
4
5  public class testDivisionByZero {
6
7      @ParameterizedTest
8      @ValueSource(ints = {0, 2, 5}) // Указываем различные значения для параметра y
9      public void testDivision(int y) {
10         int x = 10;
11         int expResult;
12
13         if (y != 0) {
14             expResult = x / y;
15         } else {
16             // Если y равно нулю, ожидаем результат также равный нулю
17             expResult = 0;
18         }
19
20         try {

```

```

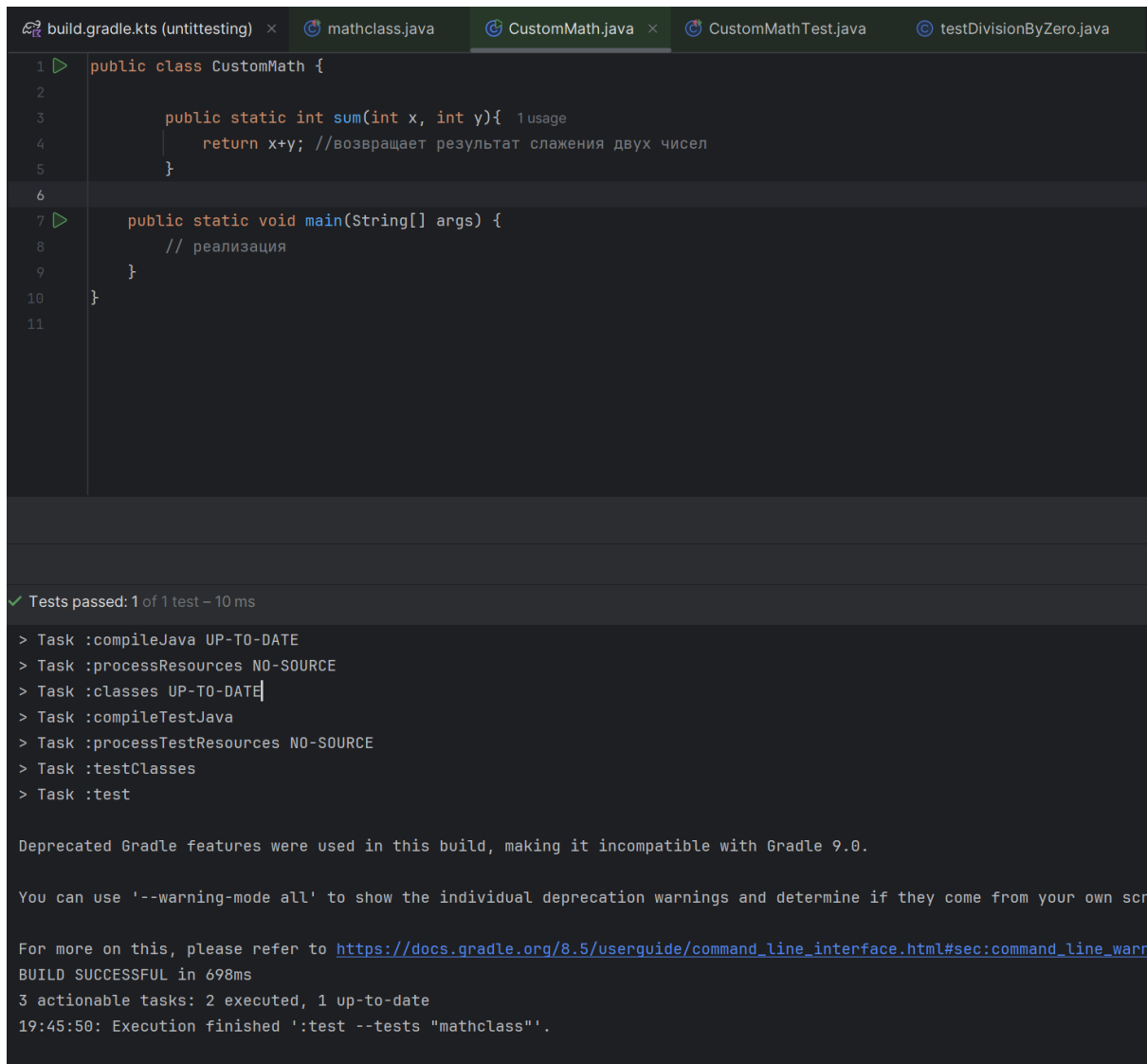
✓ Tests passed: 1 of 1 test - 12 ms

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.
You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.
For more on this, please refer to https://docs.gradle.org/8.5/userguide/command_line_interface.html#sec:command_line_warnings in the
BUILD SUCCESSFUL in 675ms
3 actionable tasks: 1 executed, 2 up-to-date
19:43:17: Execution finished ':test --tests "mathclass"'.

```

CustomMath:



The screenshot shows an IDE with several tabs: build.gradle.kts (untitled), mathclass.java, CustomMath.java, CustomMathTest.java, and testDivisionByZero.java. The CustomMath.java tab is active, displaying the following code:

```
1 public class CustomMath {
2
3     public static int sum(int x, int y){ 1 usage
4         return x+y; //возвращает результат сложения двух чисел
5     }
6
7     public static void main(String[] args) {
8         // реализация
9     }
10 }
11
```

Below the code editor, the test results are shown:

```
✓ Tests passed: 1 of 1 test – 10 ms

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own script or from Gradle.

For more on this, please refer to https://docs.gradle.org/8.5/userguide/command\_line\_interface.html#sec:command\_line\_warnings.

BUILD SUCCESSFUL in 698ms
3 actionable tasks: 2 executed, 1 up-to-date
19:45:50: Execution finished ':test --tests "mathclass"'.

```

Sarcina 4

Extindeți clasa de testare, astfel încât să utilizeze metoda `assertTrue` și / sau `assertFalse`.

Adăugarea sau modificarea metodelor clasei testate includeți în raport.

```
build.gradle.kts (untitled) x mathclass.java CustomMath.java CustomMathTest.java x testDivisionByZero.java Main.J v
1 import org.junit.jupiter.api.Test;
2 import static org.junit.jupiter.api.Assertions.*;
3
4 public class CustomMathTest {
5
6     @Test
7     public void testIsPositive() {
8         assertTrue(CustomMath.isPositive( num: 5), message: "Положительное число должно быть положительным");
9         assertFalse(CustomMath.isPositive( num: -5), message: "Отрицательное число не должно быть положительным");
10        assertFalse(CustomMath.isPositive( num: 0), message: "Ноль не является положительным числом");
11    }
12 }

Tests passed: 1 of 1 test - 11 ms

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

For more on this, please refer to https://docs.gradle.org/8.5/userguide/command\_line\_interface.html#sec:command\_line\_warnings in the
BUILD SUCCESSFUL in 712ms
3 actionable tasks: 2 executed, 1 up-to-date
19:48:43: Execution finished ':test --tests "mathclass"'.

```

CustomMath:

```
build.gradle.kts (untitled) x mathclass.java CustomMath.java x CustomMathTest.java testDivisionByZero.java Main.J v
1 public class CustomMath { 2 usages
2
3     public static int sum(int x, int y) { 1 usage
4         return x + y;
5     }
6
7     public static int division(int x, int y) { no usages
8         if (y == 0) {
9             throw new IllegalArgumentException("Деление на ноль невозможно");
10        }
11        return x / y;
12    }
13
14    public static boolean isPositive(int num) { no usages
15        return num > 0;
16    }
17 }

Tests passed: 1 of 1 test - 11 ms

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test

Deprecated Gradle features were used in this build, making it incompatible with Gradle 9.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

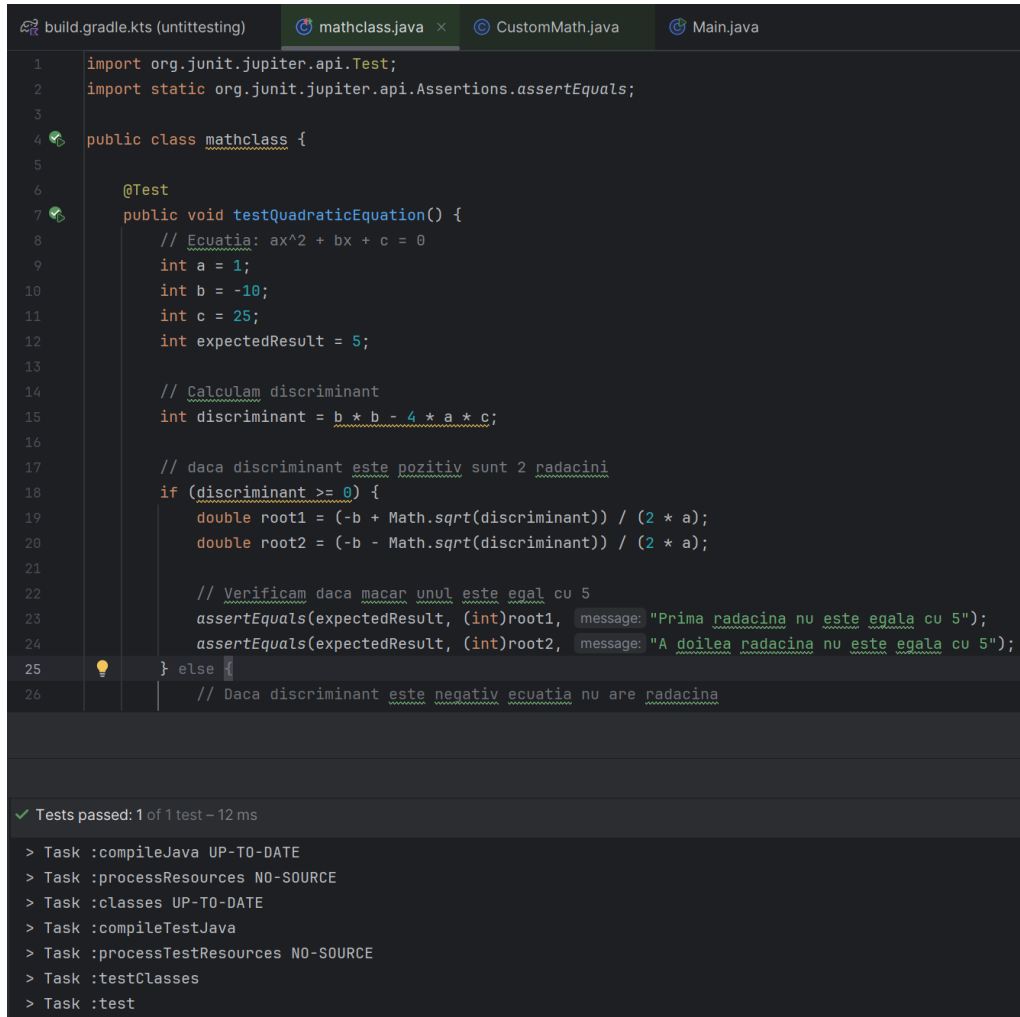
For more on this, please refer to https://docs.gradle.org/8.5/userguide/command\_line\_interface.html#sec:command\_line\_warnings in the
BUILD SUCCESSFUL in 673ms
3 actionable tasks: 1 executed, 2 up-to-date
19:49:26: Execution finished ':test --tests "mathclass"'.

```


Sarcina Individuala:

Verificarea corectitudinii a unei ecuatii patrata, anume daca radacina este egala cu 5.

Este = 5



```
1 import org.junit.jupiter.api.Test;
2 import static org.junit.jupiter.api.Assertions.assertEquals;
3
4 public class mathclass {
5
6     @Test
7     public void testQuadraticEquation() {
8         // Ecuatia: ax^2 + bx + c = 0
9         int a = 1;
10        int b = -10;
11        int c = 25;
12        int expectedResult = 5;
13
14        // Calculam discriminant
15        int discriminant = b * b - 4 * a * c;
16
17        // daca discriminant este pozitiv sunt 2 radacini
18        if (discriminant >= 0) {
19            double root1 = (-b + Math.sqrt(discriminant)) / (2 * a);
20            double root2 = (-b - Math.sqrt(discriminant)) / (2 * a);
21
22            // Verificam daca macar unul este egal cu 5
23            assertEquals(expectedResult, (int)root1, message: "Prima radacina nu este egala cu 5");
24            assertEquals(expectedResult, (int)root2, message: "A doilea radacina nu este egala cu 5");
25        } else {
26            // Daca discriminant este negativ ecuatia nu are radacina
27        }
28    }
29 }
```

✓ Tests passed: 1 of 1 test – 12 ms

```
> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
> Task :test
```

Nu este = 5

```
4 public class mathclass {
5
6     @Test
7     public void testQuadraticEquation() {
8         // Ecuatia: ax^2 + bx + c = 0
9         int a = 1;
10        int b = 10;
11        int c = -25;
12        int expectedResult = 5;
13
14        // Calculam discriminant
15        int discriminant = b * b - 4 * a * c;
16
17        // daca discriminant este pozitiv sunt 2 radacini
18        if (discriminant >= 0) {
19            double root1 = (-b + Math.sqrt(discriminant)) / (2 * a);
20            double root2 = (-b - Math.sqrt(discriminant)) / (2 * a);
21
22            // Verificam daca macar unul este egal cu 5
23            assertEquals(expectedResult, (int)root1, message: "Prima r
24            assertEquals(expectedResult, (int)root2, message: "A doile
25        } else {
26            // Daca discriminant este negativ ecuatie nu are radacina
```

✖ Tests failed: 1 of 1 test – 16 ms

Prima radacina nu este egala cu 5
Expected :5
Actual :2
[<Click to see difference>](#)

Expected	Actual
5	2

Concluzii:

Testarea unitară, în special folosind un cadru precum JUnit, aduce multiple beneficii în dezvoltarea software detectează erori timpurii, precum permite identificarea erorilor și a bug-urilor într-un stadiu incipient al dezvoltării, ceea ce face ca remediarea acestora să fie mai ușoară și mai puțin costisitoare. Am atras atenție că îmbunătățește calitatea codului, dezvoltatorii sunt încurajați să scrie cod modular, cu funcții și metode bine definite și cu o cuprindere clară a funcționalității acestora. Acest lucru duce la un cod mai clar, mai ușor de înțeles și de întreținut. În ansamblu, testarea unitară cu JUnit și alte cadre similare este esențială pentru asigurarea calității și robusteții aplicațiilor software, contribuind la dezvoltarea unor produse fiabile și eficiente.