T3. Testare unitara in Java ` Lazar Cristian-Matei

Descriere generala

Proiectul contine implementarea unei clase ce are scopul de a clasifica IMC in categorii medicale uzuale („subponderal”, „normal”, „supraponderal”, „obez” si „obez morbid”) si detectarea valorilor invalide. Aceasta este gandita drept o clasa-utilitara si vine insotita de un set complet de teste JUnit care valideaza functionalitatea pe clase de echivalenta, valori de frontiera si cativa mutanti reprezentativi.

Hardware utilizat

- 8 GB RAM
- i7-8750H
- 1 TB stocare

Software utilizat

- Windows 11 Pro 64-bit
- IntelliJ IDEA Community Edition 2025.1.1.1
- JDK 17
- Apache Maven 3.9.9
- JUnit 5
- JaCoCo 0.8.10

Strategii de testare

1. Partitionarea in clase de echivalenta

CE1 – invalid (imc <= 0, imc > 100, imc = NaN, imc = infinit)

CE2 – subponderal (0 < imc < 18.5)

CE3 – normal (18.5 <= imc < 25)

CE4 – supraponderal (25 <= imc < 30)

CE5 – obez (30 <= imc < 40)

CE6 – obez morbid (imc >= 40)

2. Analiza valorilor de frontiera / extreme

```
// invalid
@Test void imcNegativ()
{
    assertEquals("invalid", ClasificareIMC.clasifica(-1));
}
@Test void imcZero()
{
    assertEquals("invalid", ClasificareIMC.clasifica(0));
}
@Test void imcPreaMare()
{
    assertEquals("invalid", ClasificareIMC.clasifica(100.1));
}
@Test void imcNaN()
{
    assertEquals("invalid", ClasificareIMC.clasifica(Double.NaN));
}
@Test void imcInfinitPozitiv()
{
    assertEquals("invalid", ClasificareIMC.clasifica(Double.POSITIVE_INFINITY));
}
```

```
// subponderal
@Test void subponderalMargine()
{
    assertEquals("subponderal", ClasificareIMC.clasifica(18.4));
}
@Test void subponderalMic()
{
    assertEquals("subponderal", ClasificareIMC.clasifica(10));
}
```

```
// normal
@Test void normalInferior()
{
    assertEquals("normal", ClasificareIMC.clasifica(18.5));
}
@Test void normalSuperior()
{
    assertEquals("normal", ClasificareIMC.clasifica(24.9));
}
```

```java
// supraponderal
@Test void supraponderalInferior()
{
    assertEquals("supraponderal", ClasificareIMC.clasifica(25.0));
}
@Test void supraponderalSuperior()
{
    assertEquals("supraponderal", ClasificareIMC.clasifica(29.9));
}
```

```java
// obez
@Test void obezInferior()
{
    assertEquals("obez", ClasificareIMC.clasifica(30.0));
}
@Test void obezSuperior()
{
    assertEquals("obez", ClasificareIMC.clasifica(39.9));
}
```

```java
// obez morbid
@Test void morbidInferior()
{
    assertEquals("obez morbid", ClasificareIMC.clasifica(40.0));
}
@Test void morbidCrescut()
{
    assertEquals("obez morbid", ClasificareIMC.clasifica(90));
}
```

3. Acoperire structurala (JaCoCo)

clasificare-imc

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
imc		100 %		100 %	0	9	0	11	0	1	0	1
Total	0 of 42	100 %	0 of 16	100 %	0	9	0	11	0	1	0	1

Se poate observa ca, la toate nivelurile, acoperirea este completa.

4. Complexitate ciclomatica si circuite independente

Complexitatea ciclomatica a metodei „clasifica” este 9. Acest lucru indica existenta a 9 drumuri logice independente, fiecare corespunzand unei conditii sau combinatii de conditii din cod. Testele unitare au fost dezvoltate astfel incat sa acopere toate aceste drumuri si asigura o acoperire structurala completa si un control riguros asupra tuturor posibilitatilor de executie.

5. Mutation testing

```
// -- Ucidere mutanti

// '< 25' |-> '<= 25'
@Test void ucideMutantMargineNormal()
{
    assertEquals("supraponderal", ClasificareIMC.clasifica(25.0));
}

// '< 40' |-> "<= 40'
@Test void ucideMutantMargineObez()
{
    assertEquals("obez morbid", ClasificareIMC.clasifica(40.0));
}

// returneaza "obez morbid" |-> returneaza "alt text"
@Test void ucideMutantReturnareMorbid()
{
    assertEquals("obez morbid", ClasificareIMC.clasifica(75));
}
```

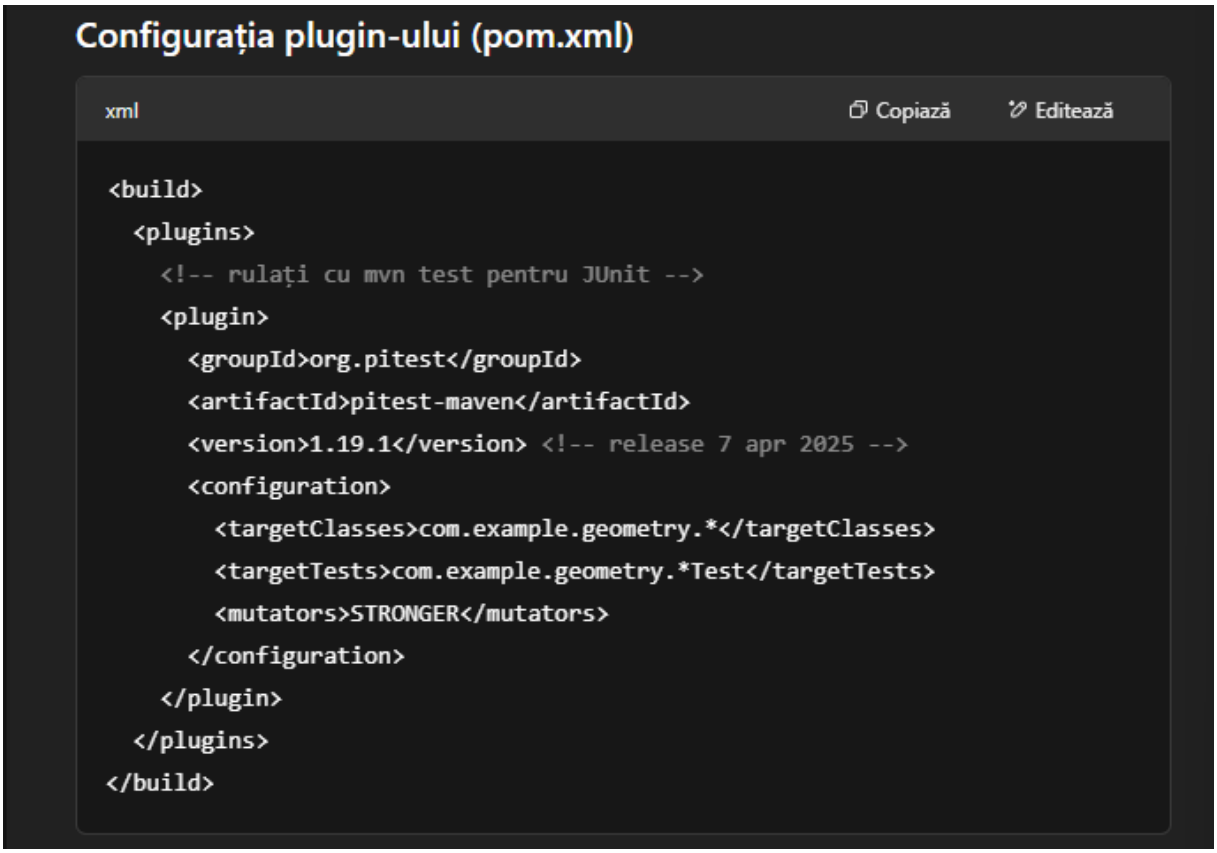
Am avut probleme cu configurarea Pitest, dar am scris teste pentru uciderea a 3 mutanti neechivalenti (2 mutanti de operator si unul de literal).

mvn clean test

```
[INFO] Results:
[INFO]
[INFO] Tests run: 18, Failures: 0, Errors: 0, Skipped: 0
```

Raport folosire AI

In cadrul realizarii proiectului, am folosit GPT-o3 pentru a-i cere indicatii in ceea ce priveste configurarea corecta a fisierului pom.xml:

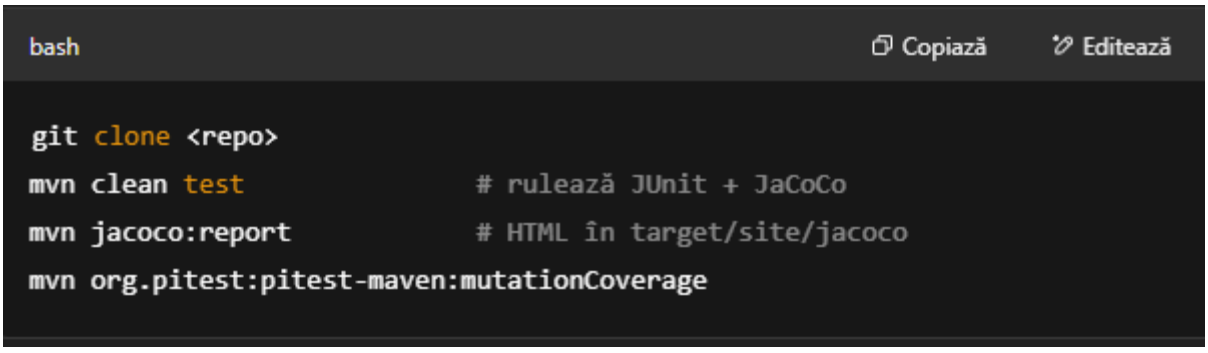


Configurația plugin-ului (pom.xml)

xml Copiază Editează

```xml
<build>
  <plugins>
    <!-- rulați cu mvn test pentru JUnit -->
    <plugin>
      <groupId>org.pitest</groupId>
      <artifactId>pitest-maven</artifactId>
      <version>1.19.1</version> <!-- release 7 apr 2025 -->
      <configuration>
        <targetClasses>com.example.geometry.*</targetClasses>
        <targetTests>com.example.geometry.*Test</targetTests>
        <mutators>STRONGER</mutators>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Si mi-a dat indicatii vizavi de workflow si de rapoartele care trebuie generate:



bash Copiază Editează

```bash
git clone <repo>
mvn clean test              # rulează JUnit + JaCoCo
mvn jacoco:report           # HTML în target/site/jacoco
mvn org.pitest:pitest-maven:mutationCoverage
```

De asemenea, i-am cerut exemple de teste unitare pentru valorile de frontiera si partitionarea in clase de echivalenta, iar el mi-a dat exemple de teste parametrizate pentru o clasa Triunghi:

```
@Nested @DisplayName("Echivalence Classes")
class EquivalencePartitioning {

    @ParameterizedTest(name = "{0}, {1}, {2} → {3}")
    @CsvSource({
        // valid
        "3,3,3,     EQUILATERAL",
        "3,3,4,     ISOSCELES",
        "3,4,5,     SCALENE",
        // invalid (regula triunghiului)
        "1,2,3,     INVALID"
    })
    void classifyByPartition(int a, int b, int c, Triangle.Type expected) {
        assertEquals(expected, Triangle.classify(a,b,c));
    }
```

```
@Nested @DisplayName("Boundary Values")
class BoundaryValues {

    @ParameterizedTest(name = "lower bound {0}")
    @CsvSource({
        "1,1,1,   EQUILATERAL",   // minim permis
        "1,2,2,   ISOSCELES",     // exact limita de existență
        "2,3,5,   INVALID"        // cu 1 peste/de-sub suma laturilor
    })
    void nearBoundaries(int a, int b, int c, Triangle.Type expected) {
        assertEquals(expected, Triangle.classify(a,b,c));
    }

    @ParameterizedTest
    @CsvSource(textBlock = """
            2147483646,2147483646,2147483646,EQUILATERAL
            2147483646,2147483646,2147483647,ISOSCELES
    """)
    void upperIntBoundaries(int a, int b, int c, Triangle.Type expected) {
        assertEquals(expected, Triangle.classify(a,b,c));
    }
}
```