Sumar

Lista de Figuri

Lista de Tabele

Lista de Snippets

Tutorialul pentru utilizarea platformei de testare JUnit 5 in IntelliJ IDEA poate conține anumiți pași care pot fi omiși.

1. Configurarea JUnit 5.x pentru proiectul Maven în IntelliJ IDEA

În proiectul Maven, în fişierul ***pom.xml*** se verifică includerea dependenţei pentru `junit-jupiter-engine` pentru **JUnit 5.x** (vezi Figure 1 şi Snippet 1).

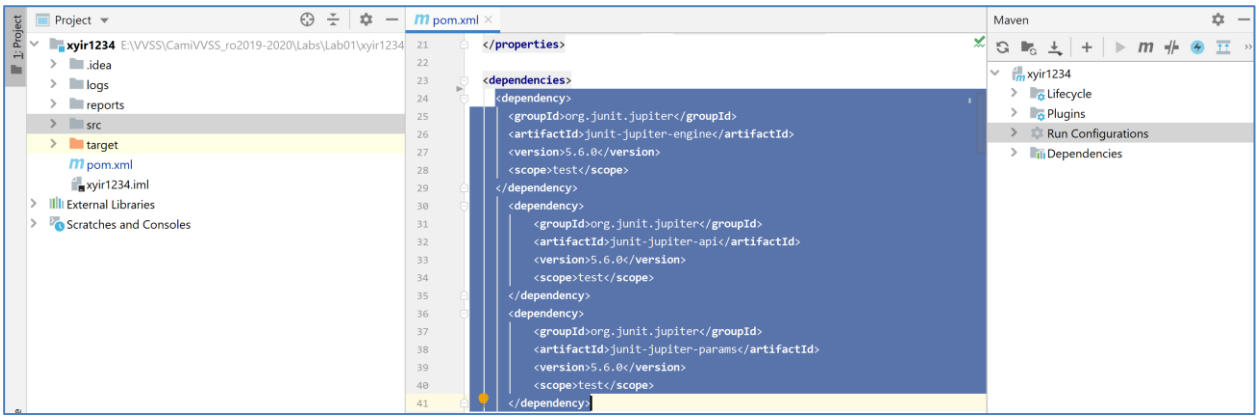

Figure 1. Dependenţele incluse în fişierul *pom.xml*, necesare utilizării JUnit 5.x în cadrul unui proiect Maven

```xml
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <version>5.6.0</version>
  <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.6.0</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-params</artifactId>
    <version>5.6.0</version>
    <scope>test</scope>
</dependency>
```

Snippet 1 Dependenţa inclusă în fişierul *pom.xml* pentru utilizarea JUnit 5.x într-un proiect Maven

2. Crearea unei clase de testare folosind JUnit

1. în **IntelliJ IDEA**, se poziţionează cursorul pe numele clasei pentru care se creează testul;
2. se foloseşte combinaţia de taste **Alt+Enter**;
3. din meniul pop-up care apare se selectează opţiunea **Create Test** (vezi Figure 2);
4. se va deschide o fereastră care permite (vezi Figure 3):
   - alegerea platformei de testare:
     - în cadrul activităţilor de laborator se va utiliza **JUnit 5.x**;
     - Table 1 prezintă o parte dintre diferenţele de utilizare la nivelul adnotărilor între **JUnit 4.x** şi **JUnit 5.x**;
   - stabilirea numelui clasei care va conţine teste (**Class name**): `TaskTest`;
   - (opţional) se poate bifa utilizarea metodelor:
     - `setUp()` – pentru iniţializarea stării *înainte* de execuţia fiecărui test;

  - `tearDown()` – pentru finalizarea testului (e.g., revenirea la starea anterioară execuţiei testului), apelată *după* execuţia fiecărui test;
  - **cele două metode se apelează implicit înainte şi după fiecare test rulat;**
- (opţional) se pot alege metodele din clasă care vor fi testate şi pentru care se vor crea teste stub (i.e., metode care vor fi descrise ulterior);
- la utilizarea **JUnit 4.x/5.x**, testele *stub* generate vor fi adnotate cu **@Test**;
- click **OK** pentru crearea clasei cu teste.

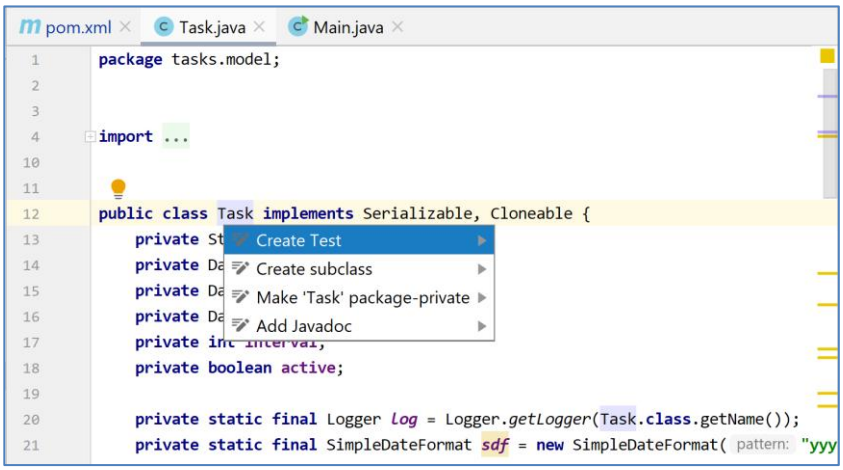

Figure 2. Crearea unei clase pentru testarea entităţii Task

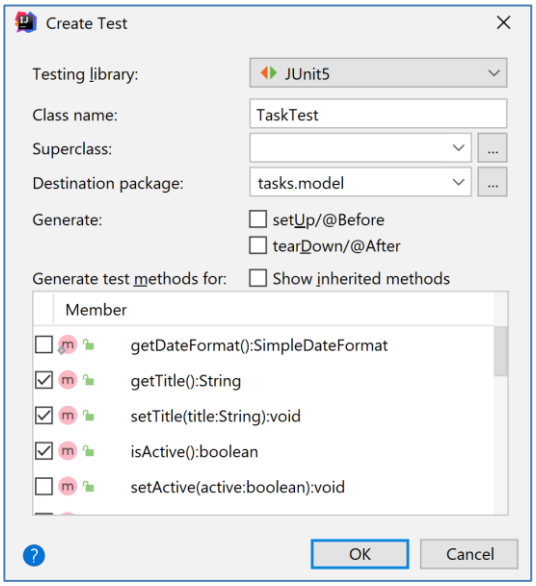

Figure 3. Alegerea framework-ului de testare şi configurarea clasei de testare

3. Scrierea unui test folosind JUnit

Exemplu 1: testarea metodei `getTitle()`:

- se declară referinţe `private` ale tipurilor utilizate în cadrul testului, i.e., `Task`, `SimpleDateFormat`, `Date` în clasa `TaskTest`:

```
private Task task;
private Date start, end;
private SimpleDateFormat sdf;
```

- referinţele se iniţializează în metoda `setUp()`:

```
sdf= Task.getDateFormat();
try {
    start=sdf.parse("2025-02-27 12:00");
    end=sdf.parse("2025-02-27 10:00");
} catch (ParseException e) {
    fail(e.getMessage());
}
task = new Task("seminar", start, end, 1);
```

- ***înainte de execuţia fiecărui test*** se execută metoda `setUp()` care va instanţia obiectele `sdf`, `start`, `end` şi `task`;
- în testul `getTitle()` se adaugă codul de testare pentru metoda `getTitle()` din clasa `Task`:

```
assertEquals("seminar", task.getTitle(), "Task title name should be \'seminar\'");
```

- dacă în cadrul unui test o construcţie `assert` eşuează, execuţia testului se încheie imediat, iar JUnit setează statusul acestuia ca failed;

Exemplu 2: testarea constructorului:

- în testul `createTask()` se adaugă codul de testare pentru testarea construirii unui obiect de tip `Task`:

```
@Test
public void createTask(){
    Task task1 = new Task("lab", start, end,1);

    //assertNotEquals(task1, null);
    assert task1 != null;
}
```

4. Execuţia testelor folosind JUnit

1. în **IntelliJ IDEA**, având clasa `TaskTest` ca şi clasă curentă:
   - din meniul **Run** ---> **Run 'TaskTest' sau**
   - click dreapta pe numele clasei `TaskTest` în **Project Explorer** şi se alege opţiunea **Run 'TaskTest';**
2. *Java Perspective* se modifică prin apariţia tab-ului **Run** sub **Project Explorer** (vezi Figure 4);

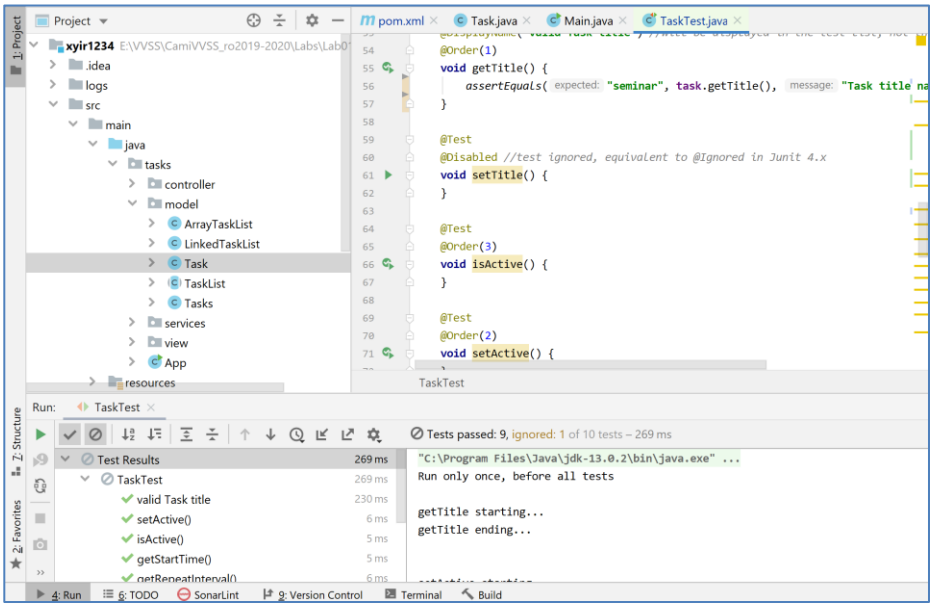

Figure 4. Afişarea rezultatului execuţiei testelor

3. la selectarea unui test din lista de teste executate, în frame-ul din partea dreaptă sunt oferite detalii cu privire la execuţie (i.e., valoare aşteptată, valoare obţinută, excepţii aruncate, etc.).

5. Localizarea bug-urilor. *Test Case* vs. *Tested Method*

1. execuţia eşuată a unui test este determinată de bug-uri care pot apărea în:
   - codul sursă testat (e.g., `getTite()`) – **[tested_method#bug]**;
   - testul propriu-zis (e.g., `testGetTitle()`) – **[test_case#bug]**;
2. se verifică dacă datele de intrare din etapa de proiectare a testului au fost preluate corect în implementare;
3. după verificare şi re-execuţia testului:
   - **success** ---> pentru datele de intrare furnizate, metoda obţine rezultatele aşteptate **[test_case#bug: fixed]**;
   - **failed** ---> există un bug în metoda testată **[tested_method#bug: needs debugging]**.

6. *JUnit 4* vs. *JUnit 5*

Îmbunătățirile disponibile în **JUnit 5** fac referire la diferite aspecte:

- modalitatea de execuție a testelor prin folosirea unor adnotări care stabilesc ordinea de execuție a testelor (e.g., ordine alfabetică, pre-stabilită, aleator), folosind adnotări ca: `@TestMethodOrder(<tip de ordonare>.class)`, `@Order(<număr>)`.
- introducerea de adnotări noi sau adaptarea celor existente, ca în Table 1.

JUnit 4	JUnit 5
`@BeforeClass`	`@BeforeAll`
`@AfterClass`	`@AfterAll`
`@Before`	`@BeforeEach`
`@After`	`@AfterEach`
`@Ignore`	`@Disabled`

`@Test` <cu diverși parametri>
 	
e.g.:
`@Test (timeout = <<value>>)`
 	
`@Test (expected = <<ClassException>>.class)`
	`@Test` [fără parametri]
	Pentru simularea aceluiași comportament din **JUnit 4** se introduc adnotări sau metode `assert` particulare
	
	e.g.: `@Timeout assertTimeout(ofMinutes(2), () -> {/*…*/});`
	
	`assertThrows(<<ClassException>>.class, () ->/*…*/});`

`@Category`	`@Tag`

Table 1. JUnit 4 vs. JUnit 5

- descrierea facilă a testelor parametrizate prin utilizarea unor adnotări specifice (e.g.: `@ParameterizedTest`, `@ValueSource`) și includerea dependenței către `junit-jupiter-params` (similar cu includerea dependenței pentru `junit-jupiter-engine`).
- execuția repetată a testelor prin utilizarea adnotării `@RepeatedTest` cu diferite signaturi;
- alte facilități documentate în https://junit.org/junit5/docs/current/user-guide/#writing-tests.