

Back-End Development

Lab Exercises

User API



G. Jongen, J. Pieck, E. Steegmans, B. Van Impe



## INHOUD

---

<b>1 TOOLS</b>	<b>5</b>
1.1 VISUAL STUDIO CODE	5
1.2 GITHUB REPOSITORY	6
<b>2 DOMAIN CLASS USER</b>	<b>7</b>
2.1 DOWNLOAD DEMO CODE	7
2.2 UNDERSTANDING DEMO CODE	7
2.3 CLASS DIAGRAM USER	8
2.4 RUN THE TESTS AND IMPLEMENT THE USER CLASS UNTIL THEY ALL PASS	9
2.5 PUSH YOUR CODE IN YOUR GITHUB REPOSITORY	12












# 1 TOOLS

---

## 1.1 Visual Studio Code

We will use Visual Studio Code as IDE (Integrated Development Environment). If you don't have installed Visual Studio Code, install it now.

You also need to install the following plugins:

-  *Test Runner for Java*
-  *Project Manager for Java*
-  *Maven for Java*
-  *Language Support for Java(TM) by Red Hat*
-  *Extension pack for Java*
-  *Debugger for Java*
-  *Spring Boot Tools*
-  *Spring Boot Dashboard*
-  *GitLens – Git supercharged*
-  *GitHub Pull Requests and Issues*
-  *Thunder Client*

## 1.2 Install Java

To be able to write and execute Java code you need to install a JDK (Java Development Kit). We need minimum version 17. Depending on your operating system you can use the following tutorials:

- Mac: <https://java.tutorials24x7.com/blog/how-to-install-java-17-on-mac>
- Windows: <https://java.tutorials24x7.com/blog/how-to-install-java-17-on-windows>

## 1.3 GitHub repository

The solution of your User lab should be saved in a personal **private** GitHub repository.

Create a **private** GitHub repository called UserBackEnd\_<name>\_<firstname>

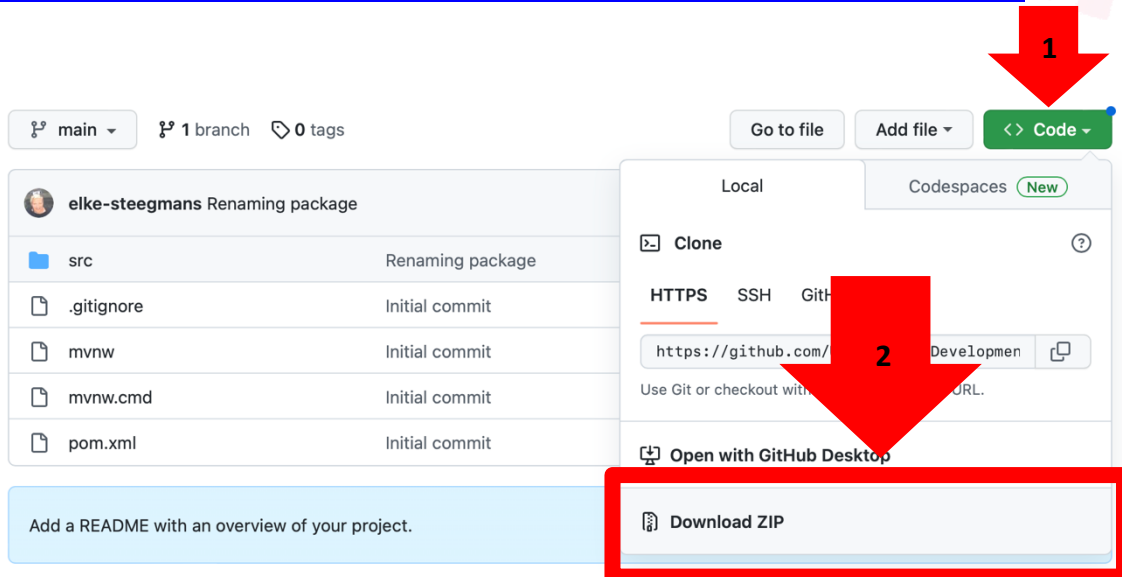
- where you replace <name> and <firstname> with your name and firstname
  - example: UserBackEnd\_Steegmans\_Elke

## 2 Domain class User

### 2.1 Download start code

Download the start code as a ZIP (DO **NOT** CLONE THIS REPO) from GitHub repository

[https://github.com/UCLLBackEndDevelopment/01\\_Labo\\_BackEndUsers\\_Model](https://github.com/UCLLBackEndDevelopment/01_Labo_BackEndUsers_Model)



### 2.2 Understanding start code

Have a look at the start code (User and DemoApplication) and answer the following questions.

1. Which methods in the class User are constructors?
2. What does it mean that the instance variables name and age are set private in the User class?
3. In which classes are objects created?
4. Look at the class DemoApplication: what will be printed in the Terminal?

## 2.3 Class diagram User

Given the class diagram for the User class, look at it and try to understand it as good as you can. Keep the class diagram open while you will do paragraph 2.4.

User
<pre>-name: String -age: int -membershipYears: List&lt;Integer&gt; -email: String -password: String</pre>
<pre>+User(name: String, age: int, email: String, password: String) +getName(): String +getAge(): int +getEmail(): String +getPassword(): String +countYearsOfMembership(): int +addMembershipYear(year: int) +countMembershipYearsAfter1999(): int +getFirstMembershipYear(): int +getNumberOfMembershipYearsIn2000(): int +isPasswordCorrect(password: String): boolean +toString(): String</pre>

We use the Java classes String and ArrayList, open the Java API of these classes, you can use them in paragraph 2.4:

- <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html>
- <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/ArrayList.html>




## 2.4 Run the tests and implement the User class until all the tests pass



**DO NOT CHANGE ANYTHING IN THE USERTEST CLASS**


Now we will start implementing the methods of the class diagram repeating following steps until the test class UserTest colors green for all test methods:

1. Go to a test method and uncomment it (select the whole part of the test method and do shift+/- for Windows or command+/- for Mac



```
class UserTest {  
    //given  
    private String validNameElke = "Elke";  
    private int validAgeElke = 44;  
    private String validEmailElke = "elke.steegmans@ucll.be";  
    private String validPasswordElke = "ikgahetnietvertellenhoor";  
  
    //...//constructor  
    //...//happy case  
    //...//@Test  
    //...//void givenValidValues_whenCreatingUser_thenUserIsCreatedWithThoseValues() {  
    //...//...//when  
    //...//...User elke = new User(validNameElke, validAgeElke, validEmailElke, validPasswordElke);  
    //...//...//then  
    //...//...assertNotNull(elke);  
    //...//...assertEquals(validNameElke, elke.getName());  
    //...//...assertEquals(validAgeElke, elke.getAge());  
    //...//...assertEquals(0, elke.countYearsOfMembership());  
    //...//...assertEquals(validEmailElke, elke.getEmail());  
    //...//...assertEquals("@$-"+validPasswordElke+"&#%", elke.getPassword());  
    //...//...}  
}
```

You, 19 minutes ago • Uncommitted changes



```

10 class UserTest {
11
12     //given
13     private String validNameElke = "Elke";
14     private int validAgeElke = 44;
15     private String validEmailElke = "elke.steegmans@ucll.be";
16     private String validPasswordElke = "ikgahetnietvertellenhoor";
17
18     //constructor
19     //happy case
20     @Test
21     void givenValidValues_whenCreatingUser_thenUserIsCreatedWithThoseValues() {
22         //when
23         User elke = new User(validNameElke, validAgeElke, validEmailElke, validPasswordElke);
24
25         //then
26         assertNotNull(elke);
27         assertEquals(validNameElke, elke.getName());
28         assertEquals(validAgeElke, elke.getAge());
29         assertEquals(expected: 0, elke.countYearsOfMembership());
30         assertEquals(validEmailElke, elke.getEmail());
31         assertEquals("@$-"+validPasswordElke+"&#%", elke.getPassword());
32     }

```

2. If you get compilation errors (red curly lines under parts of code), hover over them, read them and fix them.

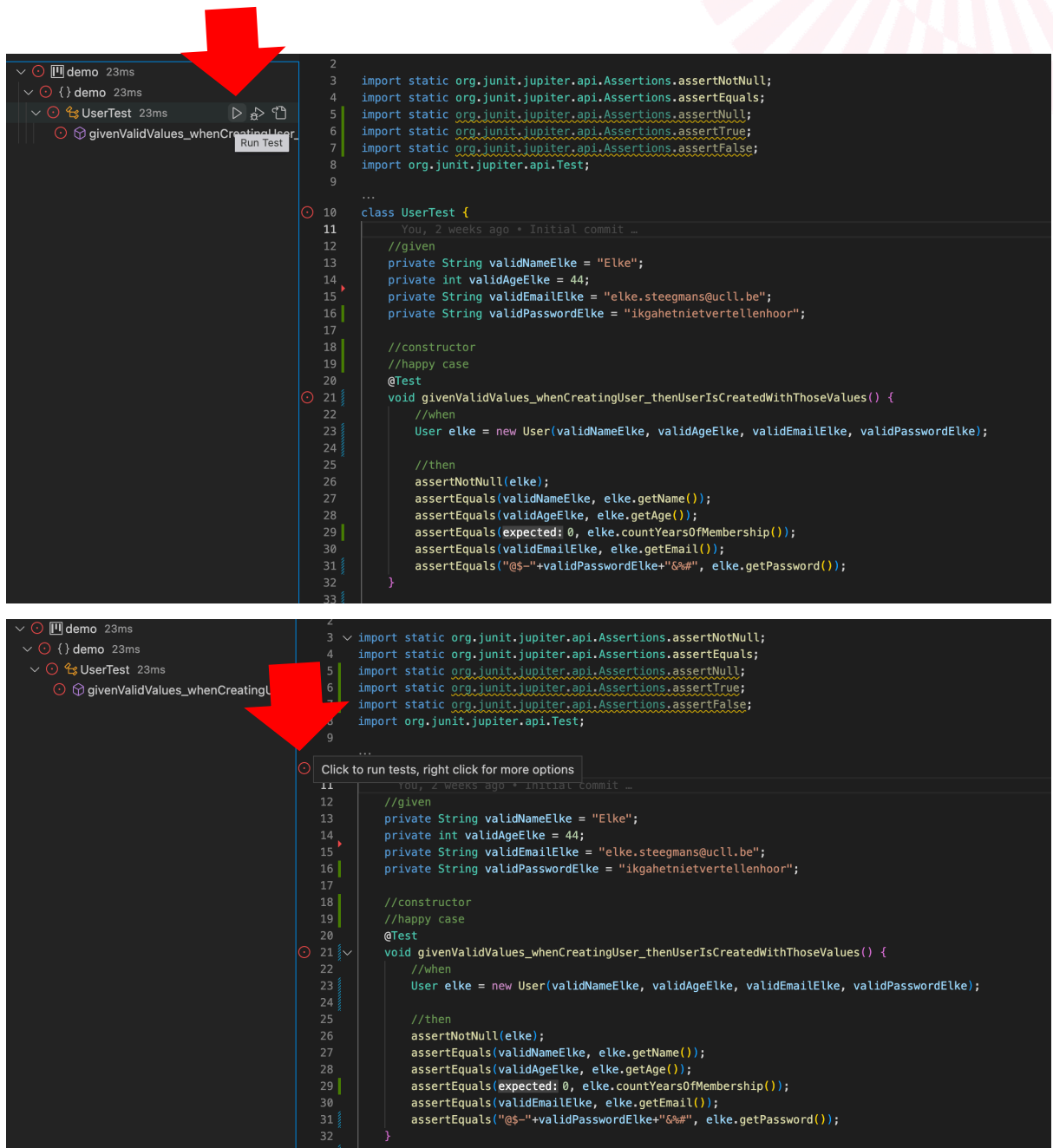


```

10 class UserTest {
11     //given
12     private String validNameElke = "Elke";
13     private int validAgeElke = 44;
14     private String validEmailElke = "elke.steegmans@ucll.be";
15     private String validPasswordElke = "ikgahetnietvertellenhoor";
16
17     //constructor
18     //happy case
19     @Test
20     void givenValidValues_whenCreatingUser_thenUserIsCreatedWithThoseValues() {
21         //when
22         User elke = new User(validNameElke, validAgeElke, validEmailElke, validPasswordElke);
23
24         //then
25         assertNotNull(elke);
26         assertEquals(validNameElke, elke.getName());
27         assertEquals(validAgeElke, elke.getAge());
28         assertEquals(expected: 0, elke.countYearsOfMembership());
29         assertEquals(validEmailElke, elke.getEmail());
30         assertEquals("@$-"+validPasswordElke+"&#%", elke.getPassword());
31     }
32 }

```

3. Run the test class, it colors red.



```
2
3 import static org.junit.jupiter.api.Assertions.assertNotNull;
4 import static org.junit.jupiter.api.Assertions.assertEquals;
5 import static org.junit.jupiter.api.Assertions.assertNull;
6 import static org.junit.jupiter.api.Assertions.assertTrue;
7 import static org.junit.jupiter.api.Assertions.assertFalse;
8 import org.junit.jupiter.api.Test;
9
10 ...
11 class UserTest {
12     //given
13     private String validNameElke = "Elke";
14     private int validAgeElke = 44;
15     private String validEmailElke = "elke.steegmans@ucll.be";
16     private String validPasswordElke = "ikgahetnietvertellenhoor";
17
18     //constructor
19     //happy case
20     @Test
21     void givenValidValues_whenCreatingUser_thenUserIsCreatedWithThoseValues() {
22         //when
23         User elke = new User(validNameElke, validAgeElke, validEmailElke, validPasswordElke);
24
25         //then
26         assertNotNull(elke);
27         assertEquals(validNameElke, elke.getName());
28         assertEquals(validAgeElke, elke.getAge());
29         assertEquals(expected: 0, elke.countYearsOfMembership());
30         assertEquals(validEmailElke, elke.getEmail());
31         assertEquals("@$-" + validPasswordElke + "&#%", elke.getPassword());
32     }
33 }
```

4. Read the name of the test method and try to understand the code written in the test method.

```
21 | void givenValidValues_whenCreatingUser_thenUserIsCreatedWithThoseValues() {
22 |     //when
23 |     User elke = new User(validNameElke, validAgeElke, validEmailElke, validPasswordElke);
24 |
25 |     //then
26 |     assertNotNull(elke);
27 |     assertEquals(validNameElke, elke.getName());
28 |     assertEquals(validAgeElke, elke.getAge());
29 |     assertEquals(expected: 0, elke.countYearsOfMembership());
30 |     assertEquals(validEmailElke, elke.getEmail()); org.opentest4j.AssertionFailedError: expected: [6#101;6#108;6#x6b]
```

Expected	Actual
elke.steegmans@ucll.be	Elke

Test run at 2/19/2023, 6:35:09 AM  
Test run at 2/19/2023, 6:27:10 AM  
Test run at 2/18/2023, 3:43:57 PM  
Test run at 2/18/2023, 3:23:49 PM  
Test run at 2/18/2023, 2:42:07 PM  
Test run at 2/18/2023, 2:41:43 PM  
Test run at 2/18/2023, 2:35:55 PM

These steps are explained in detail in the slides of Test Driven Development which you can find on Toledo!

5. Implement the method in the User class which is tested here for this concrete case.
6. When you think you are ready with your implementation, rerun the test class again
  - a. If all test methods are green, you are ready and you can start with the next test method
  - b. If there are still one or more test methods red, read the error, try to understand the error and go back to step 3 rewriting the code.

Repeat this until all test methods of the test class are colored green!

## 2.5 Run DemoApplication

Run the DemoApplication class and see what is printed in the Terminal. What is the difference with your answer on question 2.2.4? Explain why it is different now! If you can't explain it ask your coach!

## 2.6 Push your code to your GitHub repository

Only when all test methods are colored green, commit and push your code to your repo. Don't forget to give a descriptive comment each time you commit and push!