



Test Smell Game



Co-funded by
the European Union

A Game for Learning how to Detect and Fix Test Smells



Test Smell Game

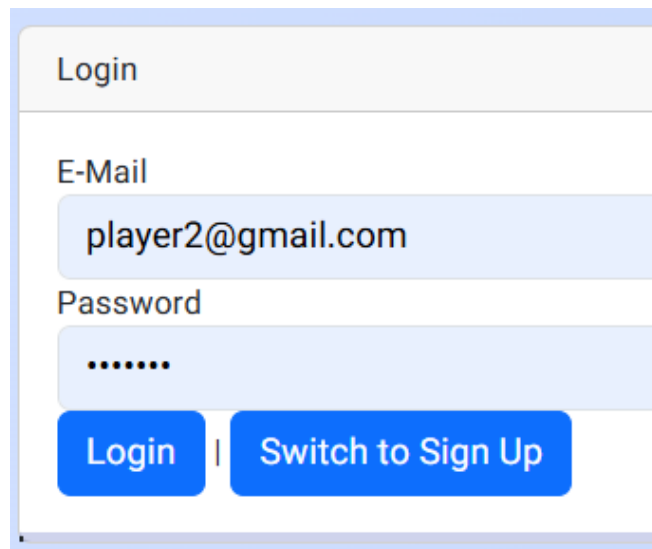


- The presence of test smells related to low-quality test cases is a known factor contributing to problems in maintaining both test suites and production code.
- The **TSGame (Test Smell Game)** capsule provides a serious game where students can familiarize with test smells by practicing with their detection and removal from JUnit test code.
- TSGame has been implemented as a Web-based application that allows a teacher to assign students test smell detection and refactoring tasks that they have to accomplish in game sessions. Upon completion of the tasks they have the possibility to gain rewards.

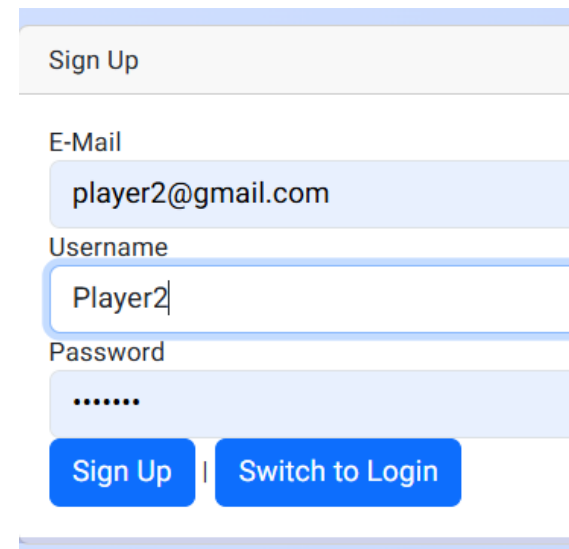
How To Play

Registration and Login

- The player must register by indicating email, username and password.
- After registration, the player must log in with the same credentials



Mockup of a Login form. The form has a title 'Login' at the top. Below it are three input fields: 'E-Mail' containing 'player2@gmail.com', and 'Password' containing masked characters '.....'. At the bottom are two blue buttons: 'Login' and 'Switch to Sign Up'.



Mockup of a Sign Up form. The form has a title 'Sign Up' at the top. Below it are three input fields: 'E-Mail' containing 'player2@gmail.com', 'Username' containing 'Player2', and 'Password' containing masked characters '.....'. At the bottom are two blue buttons: 'Sign Up' and 'Switch to Login'.

Game modes

- There are three game modes:
 - **Refactoring** : given a test code and a list of smells, the player has to refactor the test class by removing the test smells (trying to avoid code coverage losses)
 - **Check Game** : given a test code, the player has to guess which test smells occur in the test code
 - **Missions** : sequences of activities to complete, including small lectures about test smells, refactoring exercises and check games
- Each game mode can provide experience points and rewards

Refactoring Game Mode

- In this game mode the GUI show the code of a class under test and the corresponding code of a test class
- Clicking on **Compile** the test class will be executed and a list of the existing smells will be shown

The screenshot displays the Refactoring Game Mode interface. It features two code editors at the top: **Calculator.java** and **CalculatorTest.java**. Below the editors are **Compile** and **Finish** buttons. A **Shell** window shows the output of the test execution, indicating that all tests passed. At the bottom, a **Smells** section lists the detected code smells: **Assertion Roulette**, **Redundant Assertion**, and **Magic Number Test**, each with a count of 1. The interface also shows a **Score** of 3 and a **Refactoring result** of true.

```
Calculator.java
1 package com.darwinintore.test;
2
3 /**
4  * Calculator class:
5  * Basic Mathematical functions like
6  * Add, Subtract, Multiply, Divide.
7  */
8
9 public class Calculator {
10     //no-arg constructor
11     public Calculator() {
12     }
13     /**
14      * Sum method.
15      */
16 }
```

```
CalculatorTest.java
1 package com.darwinintore.test;
2
3 import org.junit.*;
4 import static org.junit.Assert.*;
5
6
7
8 //Arrange-Act-Assert pattern
9
10 public class CalculatorTest {
11
12     private Calculator objCalculatorTest;
13
14     @Before
15     public void setUp() {
```

Compile Finish

Shell

```
1 [INFO] T E S T S
2 [INFO] -----
3 [INFO] Running com.darwinintore.test.CalculatorTest
4 [INFO] Tests run: 4, Failures: 0, Errors: 0,
```

Compilation Successful

Score : 3

Refactoring result : true
Original coverage: 85
Refactored coverage: 85

Smells :

- Assertion Roulette : 1
- Redundant Assertion : 1
- Magic Number Test : 1

Refactoring Game Mode

- With the help of the tool that notifies which methods contains test smells, the player has to refactor the test code for removing the smell's causes
- After the player has modified the code by clicking **Compile** it is possible to know if the smells have been removed
- **Note:** smells could be removed simply by deleting code. This solution is considered valid by the system only if there is not a sensible reduction in **code coverage**

```
11 public Calculator() {
12 }
13 /**
14  * Sum method.
15  */
16 public int add(int a, int b) {
17     return a + b;
18 }
19 /**
20  * Subtract method.
21  */
22 public int subtract(int a, int b) {
23     return a - b;
24 }
25
```

```
16
17 @Test
18 public void testAdd() {
19     int a = 15;
20     int b = 20;
21     int expectedResult = 35;
22     // Act
23     long result = objCalcUnderTest.add(a, b);
24     // Assert
25     assertEquals(expectedResult, result);
26     assertEquals(result, result);
27 }
28
29 @Test
30 public void testSubtract() {
```

Class under Test

Compile Button

Compile

Save in Solution Repository

Shell

```
1 [INFO] T E S T S
2 [INFO] -----
3 [INFO] Running com.dariotintore.tesi.CalculatorTest
4 [INFO] Tests run: 5, Failures: 0, Errors: 0,
```

Test Compile Results

Test class with smells

Results

Exercise Compilation Summary

Coverage Loss Acceptable: true

Original Coverage: 100

Refactored Coverage: 100

Smells Allowed: 3

Your refactored code has 4 smells remaining — too many to pass! The limit is 3



Remaining Smells

Assertion Roulette : 1

testAdd

Redundant Assertion : 1

Magic Number Test : 1

Lazy Test : 1

Remaining Smells

Example

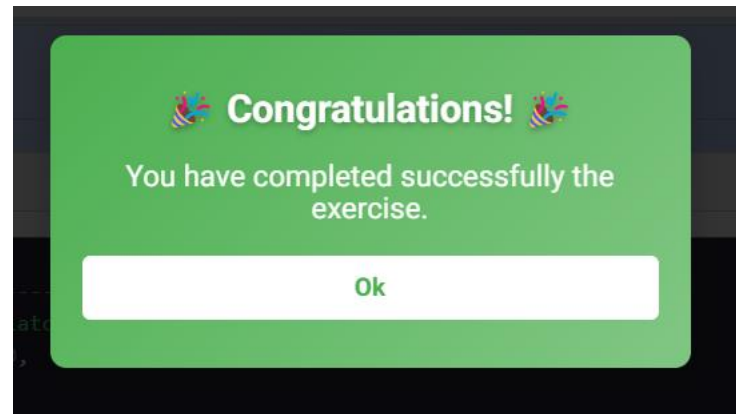
- In this example the **Redundant Assertion** has been refactored by removing it and without losses in code coverage

```
@Test
public void testAdd() {
    int a = 15;
    int b = 20;
    int expectedResult = 35;
    // Act
    long result = objCalcUnderTest.add(a, b);
    // Assert
    assertEquals(expectedResult, result);
    assertEquals(result, result);
}
```

```
@Test
public void testAdd() {
    int a = 15;
    int b = 20;
    int expectedResult = 35;
    // Act
    long result = objCalcUnderTest.add(a, b);
    // Assert
    assertEquals(expectedResult, result);
    //assertEquals(result, result);
}
```

Refactoring Game Mode

- The game can be terminated in any moment by clicking **Finish**
 - Conventionally, the system considered valid an exercise with few remaining smells (the number of allowed smells is different for each exercise)
- The system will download four files (Class code, Test Code, Shell Code and Results)
- The system will return to the main screen, but other solutions to the same exercise can be submitted further: new solutions update the previous solution



Collaborative Mode

- In some refactoring games, it is possible to share our solutions with other players, in order to understand other possible solutions to the same problem
- Code in the **solution repository** can be read by players and also commented

Solution Repository Class Refactored by the Player

Original Class under Test

player2

Score : 1

Monday, May 12, 2025 at 2:20:52 PM GMT+02:00

Fullscreen

```
16
17 @Test
18 public void testAdd() {
19     int a = 15;
20     int b = 20;
21     int expectedResult = 35;
22     // Act
23     long result = objCalcUnderTest.add(a, b);
24     // Assert
25     assertEquals(expectedResult, result);
26     assertEquals(result, result);
27 }
28
29 @Test
30 public void testSubtract() {
```

```
16
17 @Test
18 public void testAdd() {
19     int a = 15;
20     int b = 20;
21     int expectedResult = 35;
22     // Act
23     long result = objCalcUnderTest.add(a, b);
24     // Assert
25     assertEquals(expectedResult, result);
26     //assertEquals(result, result);
27 }
28
29 @Test
30 public void testSubtract() {
```

Refactoring result : true

Smells :

Magic Number Test

testDivide

Lazy Test

divide

0 0

Comments

Write a comment

Results Summary

Comments

Leaderboard

- In competitive games, a leaderboard of the better solutions is maintained

REFACTORING EXERCISE PODIUM

Calculator

POSITION 1 (SCORE 3)

player1

POSITION 2 (SCORE 0)

player2 , player3 , player4

JunitExample

POSITION 1 (SCORE 2)

player1 , player3

POSITION 2 (SCORE 1)

player2

POSITION 3 (SCORE 0)

player4

Check Game Mode

- In this game mode, a sequence of **questions** is posed to the player
- In each question a test class is shown : the player has to recognize which types of test smells occurs in that class
- A list of possible test smells type is reported
- Each class may contain zero, one or more than one type of test smells
- The player has to recognize all the test smells in order to win the game

Calculator.java

Which is the smell present in the following method?

```
1  @Test
2  public void testAdd() {
3      int a = 15; int b = 20;
4      int expectedResult = 35;
5      //Act
6      long result = objCalcUnderTest.add(a, b);
7      //Assert
8      assertEquals(expectedResult, result);
9  }
10
```

☐ Assertion roulette

☐ Magic number

Next

Check Game Example

Which is the smell present in the following method?

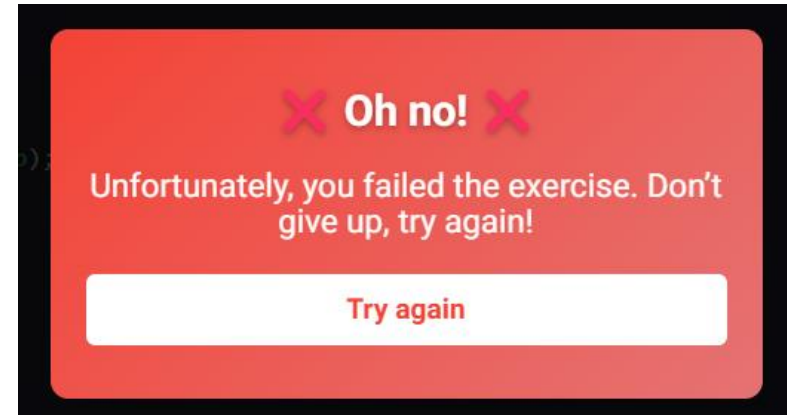
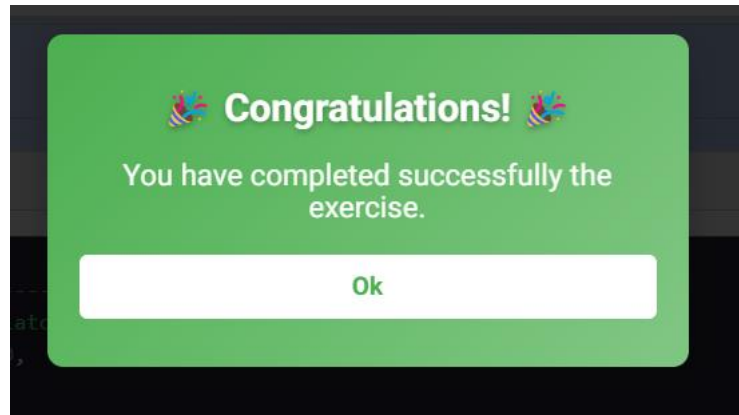
```
1 @Test
2 public void testAdd() {
3     int a = 15; int b = 20;
4     int expectedResult = 35;
5     //Act
6     long result = objCalcUnderTest.add(a, b);
7     //Assert
8     assertEquals(expectedResult, result);
9 }
10
```



- ☐ Empty Test
- ☐ Redundant Assertion
- ☐ No smell present

Check Game Results


- Check Games can be repeated until all the correct answers are provided



Mission Game Mode

- In the Mission Game Mode, the player has to accomplish a sequence of activities, including:
 - **Learning** about specific test smells;
 - **Refactoring** specific test smells from test code;
 - **Recognizing** test smells in test code
- The accomplishment of each new mission gives **experience points** and/or **rewards**

Mission Game Mode

TSGame Home Refactoring Game Check Game Assignments Missions Settings Logout 

Available Missions

Tutorial

Test Smells (pt. 1)

Achievement: Tutorial test smells 1

100%

Completed

Test Smells (pt. 2)

Achievement: Tutorial test smells 2

0%

Go to mission

Test Smells (pt. 3)

Achievement: Tutorial test smells 3

0%


Go to mission

Unlock after completing:

- Test Smells (pt. 1)
- Test Smells (pt. 2)

 **Missione Completata!** 

Congratulations! You have complete the mission and unlocked the badge:
Tutorial test smells 1.






Welcome to your first mission!

In this mission, you will learn what test smells are and how to identify some of the most common ones. Through a series of educational screens, you will discover why some tests can be fragile or difficult to maintain.

At the end of the theoretical section, you will take a short quiz to test your knowledge. Don't worry if you don't pass on the first try—you can retake it as many times as you like!



And remember, you can leave the mission at any time and resume from the last completed step.



 **Congratulations!** 

You have completed successfully the mission step.

Ok

 **Holy cow, you have completed the mission** 

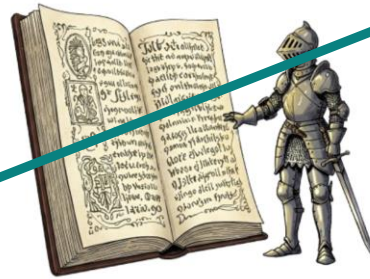
You can see the new badge in your account page

Awesome!

Mission Game Mode

What is a test smell?

Test smells are defined as bad programming practices in unit test code (such as how test cases are organized, implemented and interact with each other) that indicate potential design problems in the test source code.



Learning

Refactoring

Check Game

```
Calculator.java
1 package com.dariotintore.tesi;
2
3 /**
4  * Calculator class:
5  * Basic Mathematical functions like
6  * Add, Subtract, Multiply, Divide.
7  *
8  */
9 public class Calculator {
10     //no-arg constructor
11     public Calculator() {
12     }
13     /**
14      * Sum method.
15      */
16 }

CalculatorTest.java
6
7
8 //Arrange-Act-Assert pattern
9
10 public class CalculatorTest {
11
12     private Calculator objCalcUnderTest;
13
14     @Before
15     public void setUp() {
16         //Arrange
17         objCalcUnderTest = new Calculator();
18     }
19
20     @Test
21 }
```

Step 8

Which is the smell present in the following method?

```
1 @Test
2 @Ignore
3 public void testSubtract() {
4     int a = 25; int b = 20;
5     int expectedResult = 5;
6     long result = objCalcUnderTest.subtract(a, b);
7     assertEquals(expectedResult, result);
8 }
9
```

☐ Ignored Test

☐ Redundant Print

☐ Empty Test

Back Next Submit Next



Exercise Compilation Summary

Coverage Loss Acceptable: true

Original Coverage: 85
Refactored Coverage: 85

Smells Allowed: 3

Your refactored code has 3 smells remaining, staying within the allowed limit of 3.

Score: 0

Experience points and Rewards

player2

player2@gmail.com

Level : ★★☆☆


Experience points : 9


Progress to next badge


You need **1** more points to achieve the **Bronze badge** badge.


Progress to next level


You have reached the maximum level, awesome!

 **User Score**


 Refactoring Score: **1**

 Check Smell Score: **8**

 Missions Score: **0**

 **GAME MODE RANKINGS**

check-smell	1 st position
missions	3 rd position
refactoring	3 rd position

 **REFACTORING EXERCISE RANKINGS**

Calculator	2 nd position
JunitExample	2 nd position

Assignment

- In addition, there is an Assignment mode that can be used by teachers that want to assign specific exercises to specific players

Refactoring type assignments
Refactoring Assignment 1 Assignment type: refactoring The assignment is unavailable
Check smell type assignments
Check Smell Assignment 1 Assignment type: check-smell XMLParser Available from 2024-12-10 at 00:00 To be delivered by 2024-12-10 at 17:45