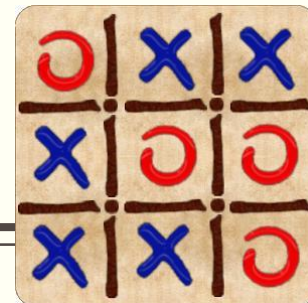




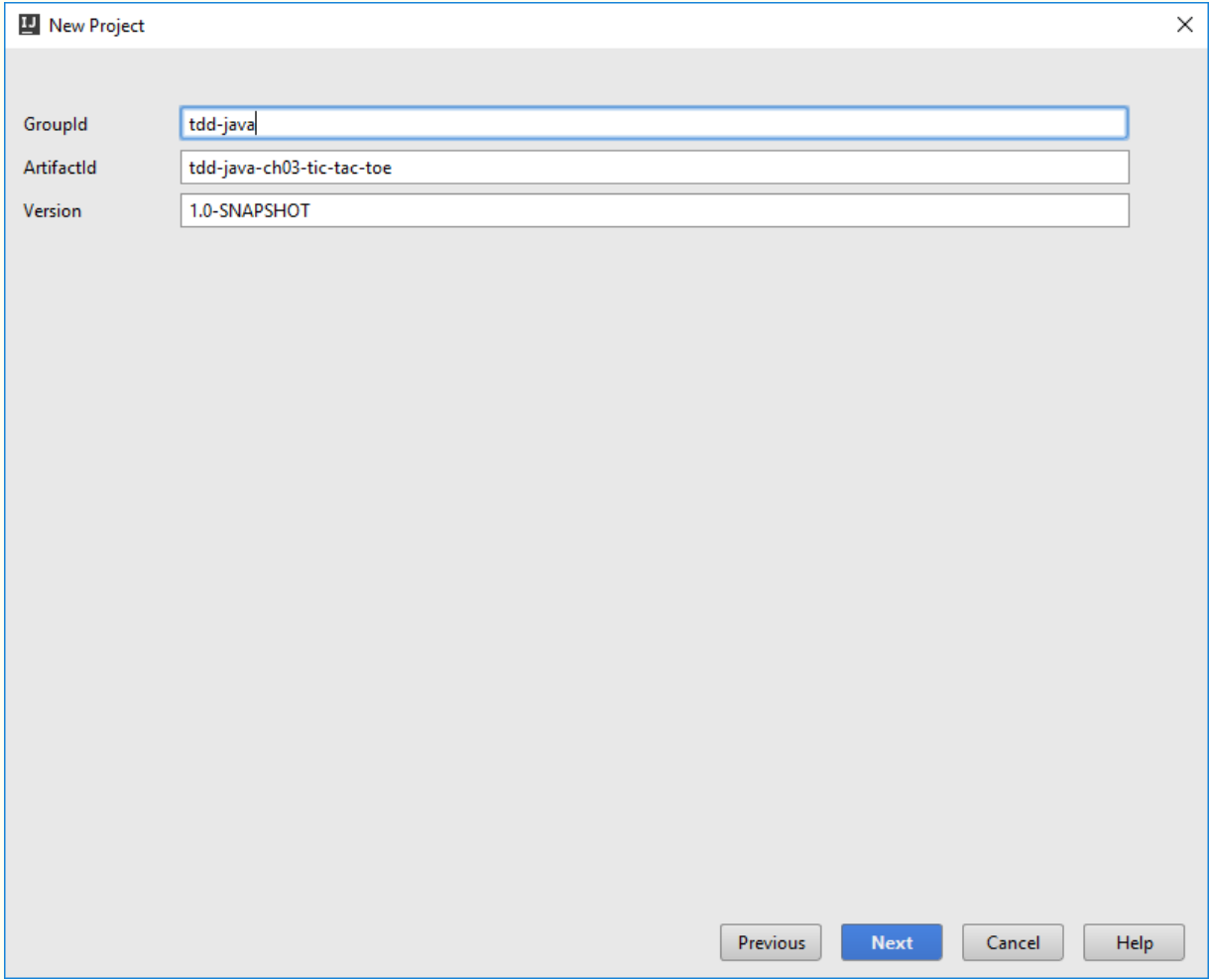
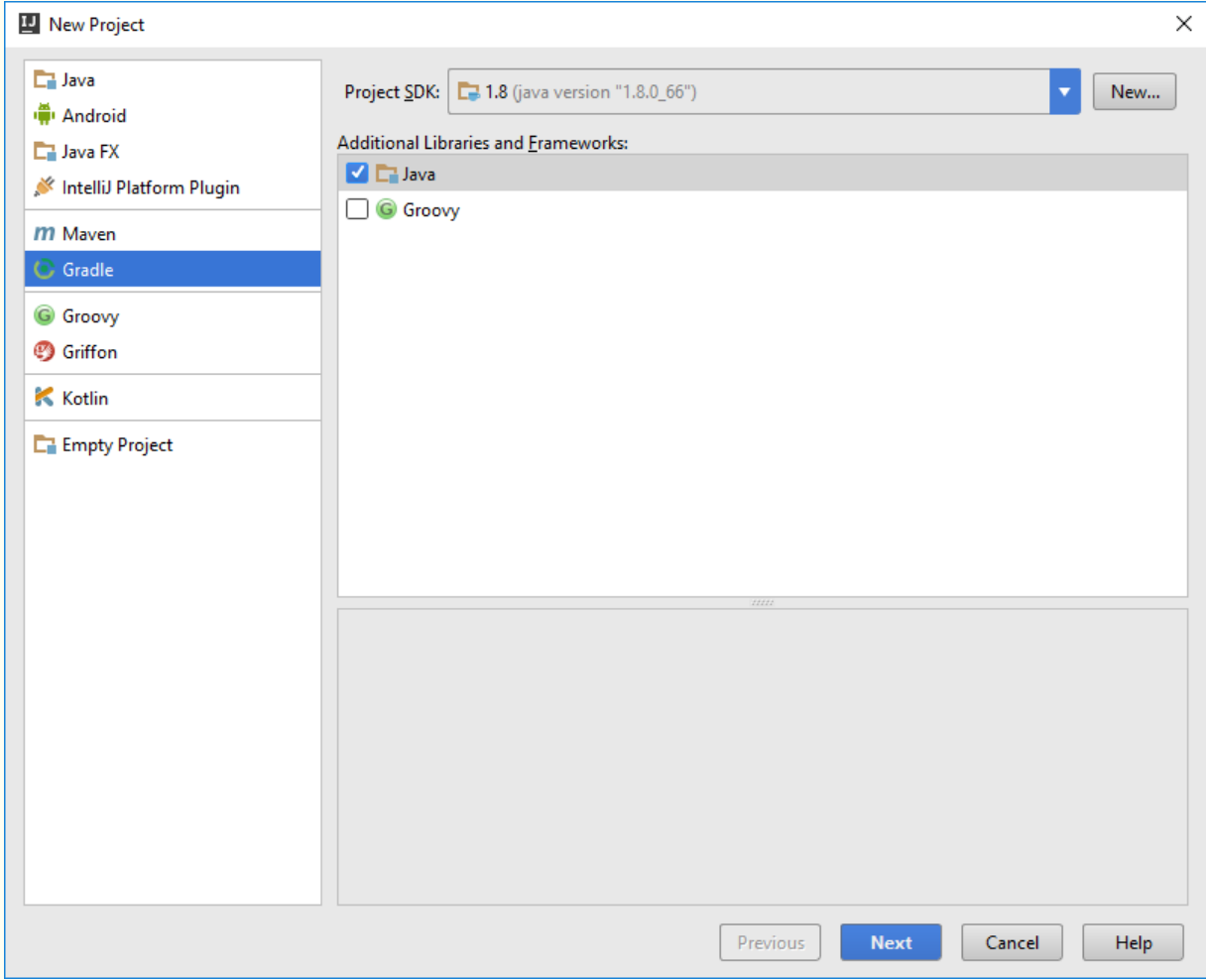
ПРИКЛАД РОЗРОБКИ ЧЕРЕЗ ТЕСТУВАННЯ

Питання 4.5

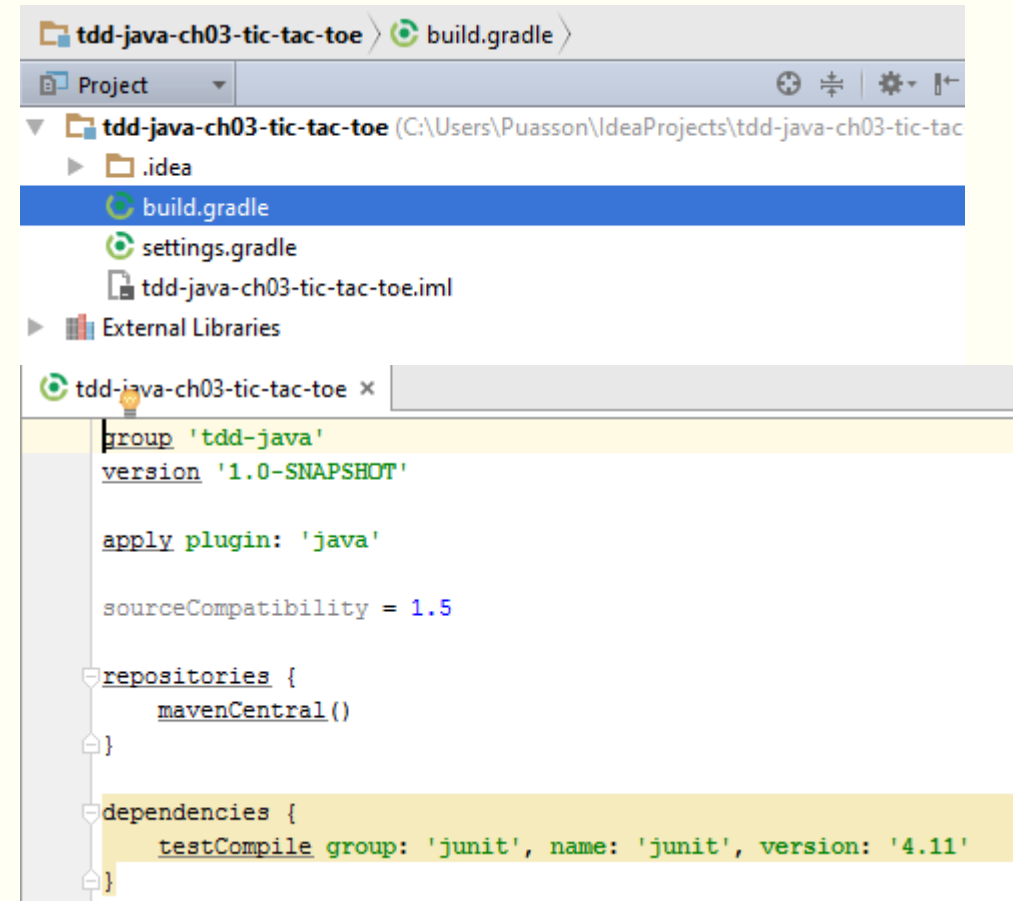
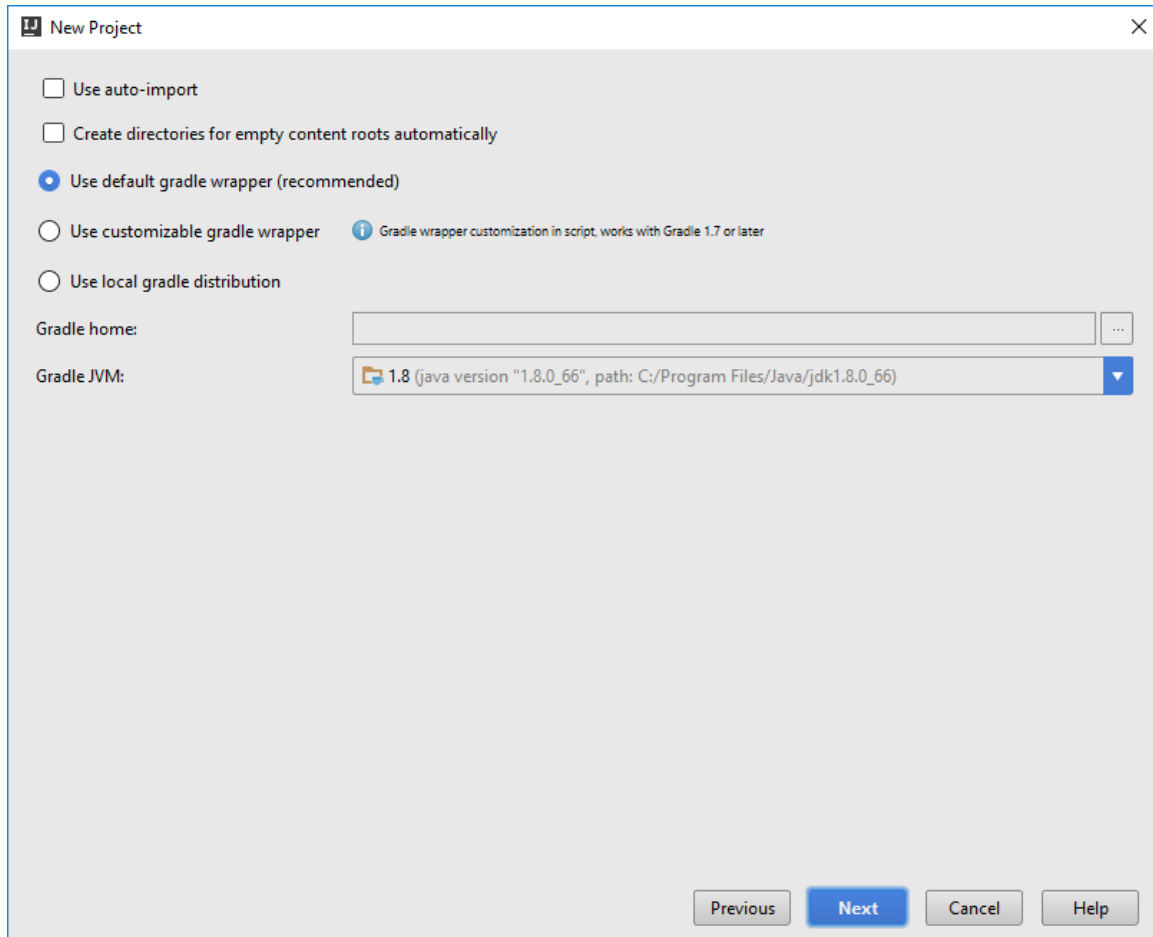


У наших планах

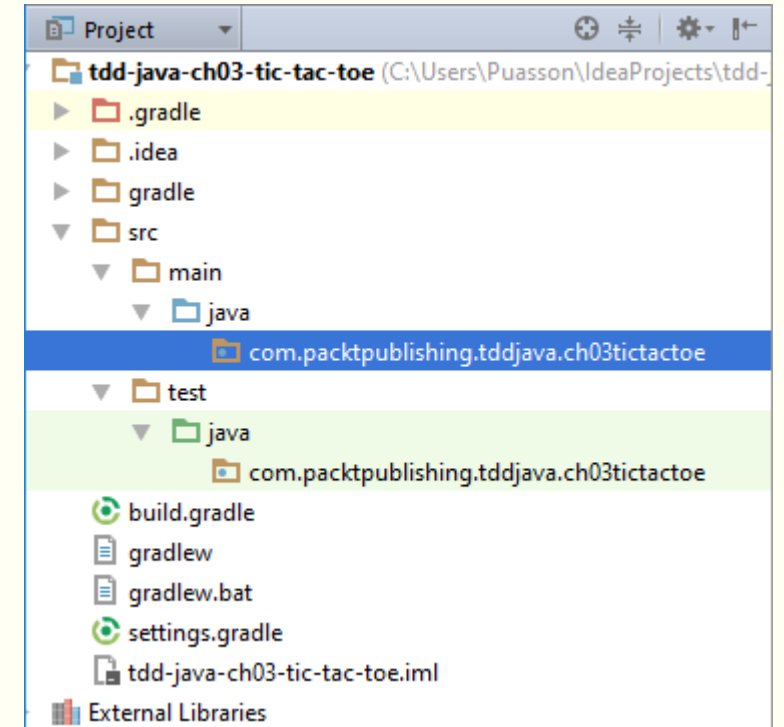
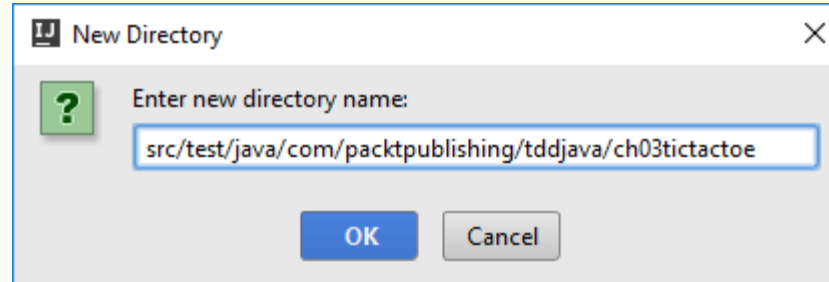
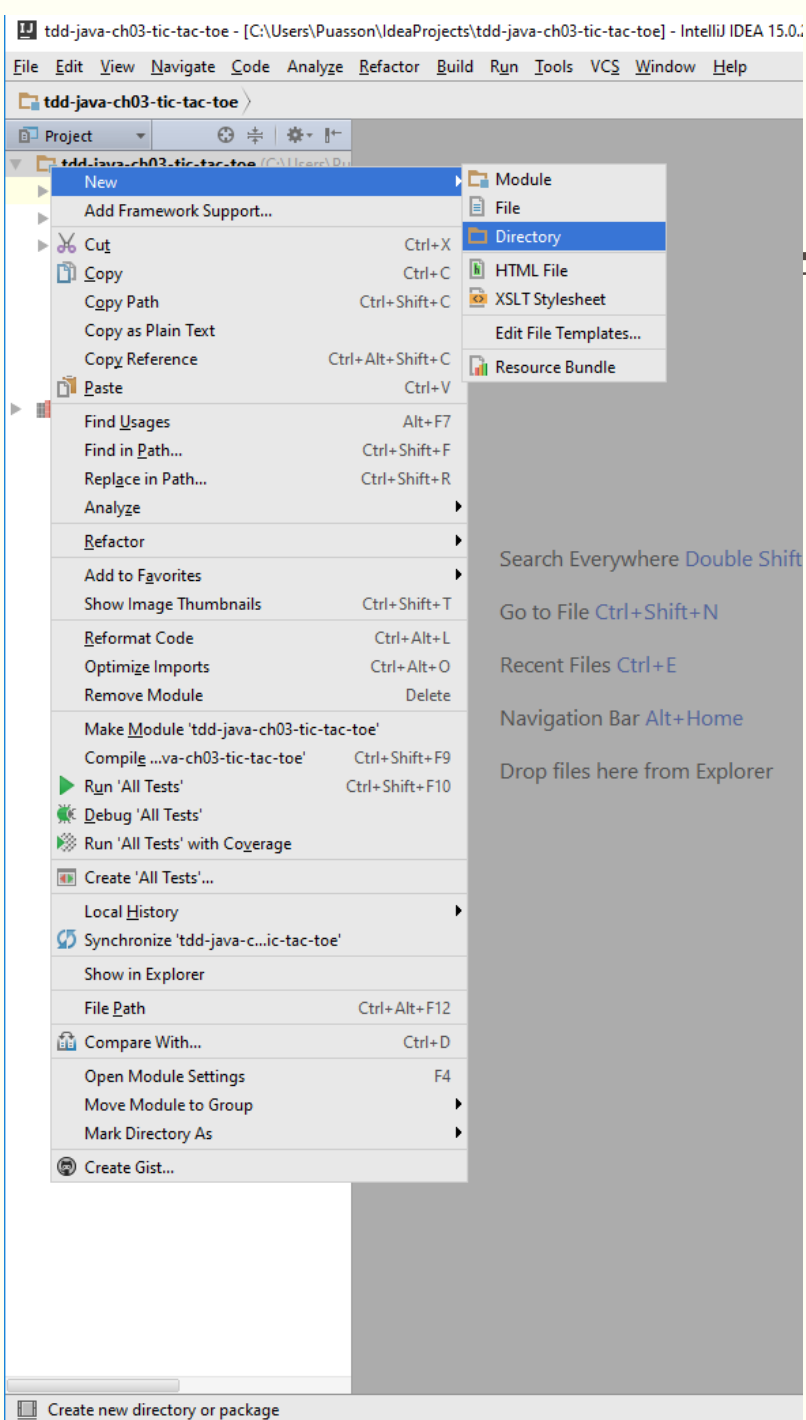
- Розробимо гру «Хрестики-нулики», проходячи по одній вимозі за раз.
 - Напишемо тест, щоб побачити, чи він провалюється.
 - Потім напишемо код, що реалізує цей тест, запустимо всі тести та доведемо їх до успішного проходження.
 - У кінці виконаємо рефакторинг та спробуємо покращити код.
 - Процес буде багато разів повторюватись, поки всі вимоги не будуть успішно імплементованими.
- Розпочнемо з налаштування середовища розробки для роботи з Gradle та Junit.
 - Потім заглибимось у процес red-green-refactor process.
 - Закінчивши налаштування та розбір теорії, пройдемося по високорівневих вимогах для додатку.
 - Після всіх налаштувань перейдемо до коду – одна вимога за один раз.
 - Як тільки все виконано, розглянемо покриття коду (code coverage) та вирішимо, чи воно прийнятне. Можливо, будуть потрібні нові тести.



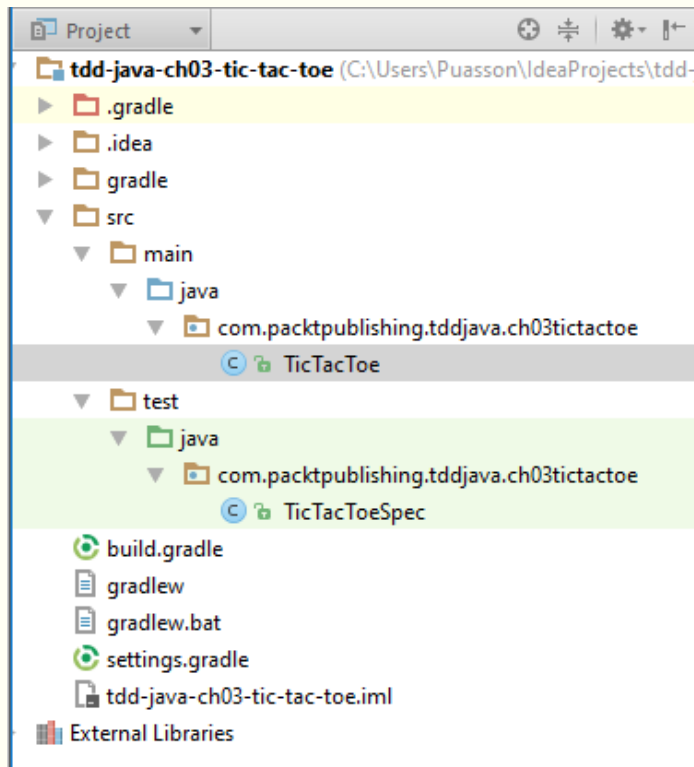
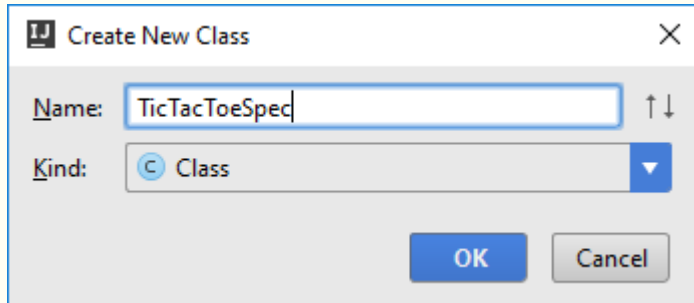
Налаштування середовища з Gradle та JUnit



Залишилось створити пакети для тестів та реалізації



Також потрібні класи для тестів та реалізації



- TicTacToeSpec – для тестування, TicTacToe – для реалізації
- Переваги відокремлення тестів від коду реалізації:
 - Усуває можливість випадкового пакування тестів разом з production binaries;
 - Багато інструментів збірки очікують на знаходження тестів у деякій source directory.
- Поширена практика – мати принаймні 2 source directories.
 - Код реалізації має знаходитись у src/main/java
 - Код тестів - у src/test/java.
 - У більших проектах кількість папок з кодом може зростати, проте відокремленість реалізації від тестів має залишатись.
- Інструменти збірки, зокрема Maven та Gradle, очікують відокремленість директорій з кодом, як передбачено naming conventions.

Розробка Tic-Tac-Toe

- **ВИМОГА 1:** треба розпочати із задання меж та того, що визначає invalid розміщення фігури.
 - Фігура (ріесе) може поміщатись в порожню комірку дошки 3×3.
- Розіб'ємо вимогу на 3 тести:
 - Коли фігура розміщена будь-де за межами осі X, викидається RuntimeException.
 - Коли фігура розміщена будь-де за межами осі Y, викидається RuntimeException.
 - Коли фігура розміщена на зайнятій області, викидається RuntimeException.
- Тести відносно першої вимоги стосуються валідації вхідного аргументу.
 - Нічого не сказано про те, що потрібно робити з фігурами.

```
package com.packtpublishing.tddjava.ch03tictactoe;

import org.junit.Before;
import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.ExpectedException;

public class TicTacToeSpec {

    @Rule
    public ExpectedException exception =
        ExpectedException.none();
    private TicTacToe ticTacToe;

    @Before
    public final void before() {
        ticTacToe = new TicTacToe();
    }

    @Test
    public void whenXOutsideBoardThenRuntimeException()
    {
        exception.expect(RuntimeException.class);
        ticTacToe.play(5, 2);
    }
}
```

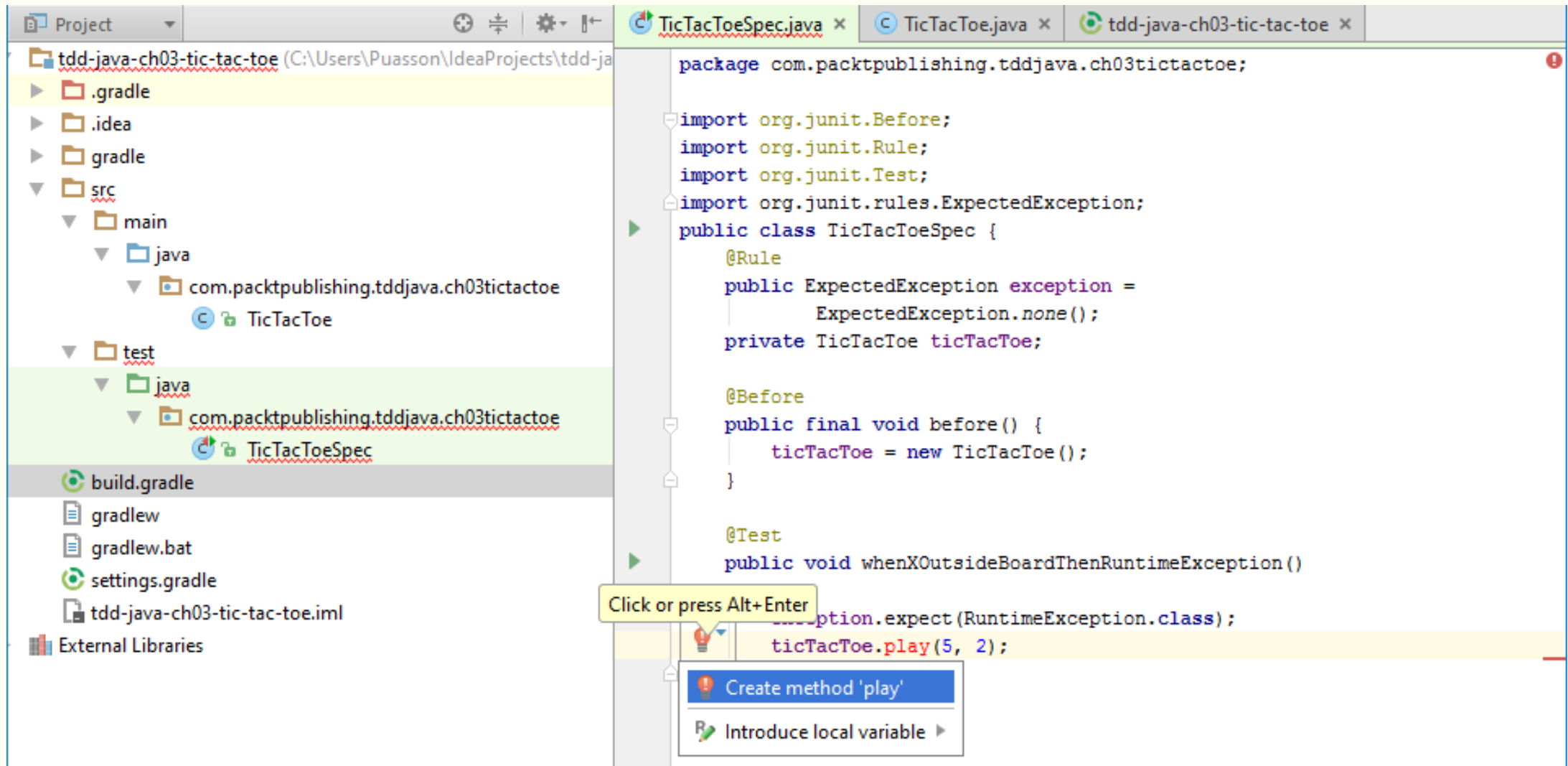
Тест

- Коли фігура поміщається за межі вісі X, викидається RuntimeException
 - У тесті визначаємо, що RuntimeException очікується, коли викликається метод ticTacToe.play(5, 2).
 - Короткий та простий тест.
- Потрібно створити метод play() та переконатись, що він викидає RuntimeException, коли X-аргумент менший за 1 або більше за 3.

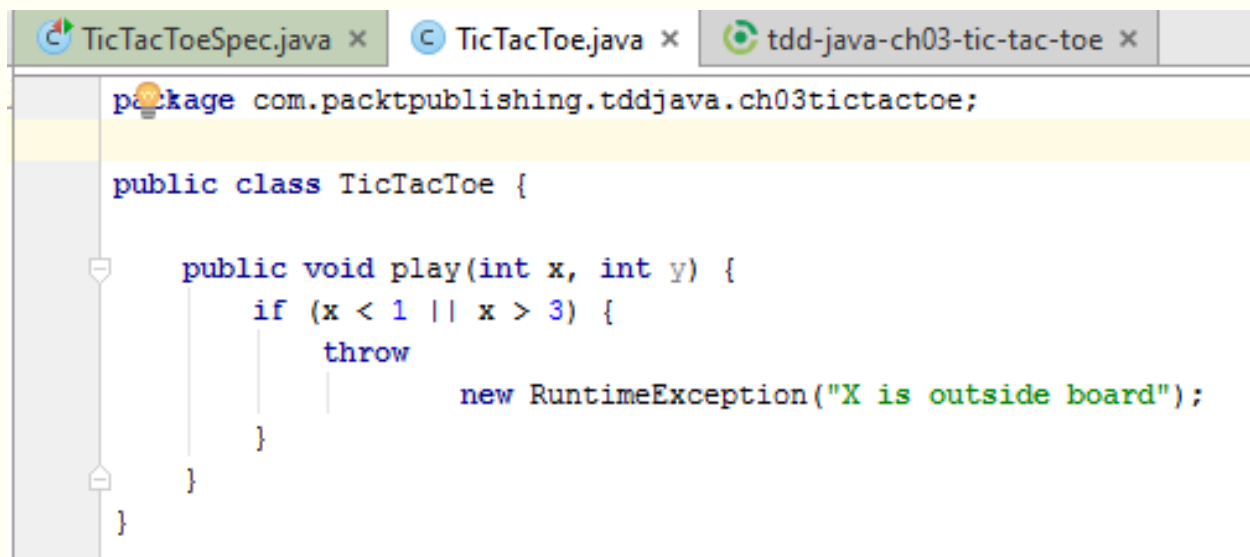
Слід запустити цей тест тричі:

- 1) fail, бо не існує методу play;
- 2) fail, оскільки RuntimeException не викидається
- 3) successful.

Перший крок тестування



Реалізація

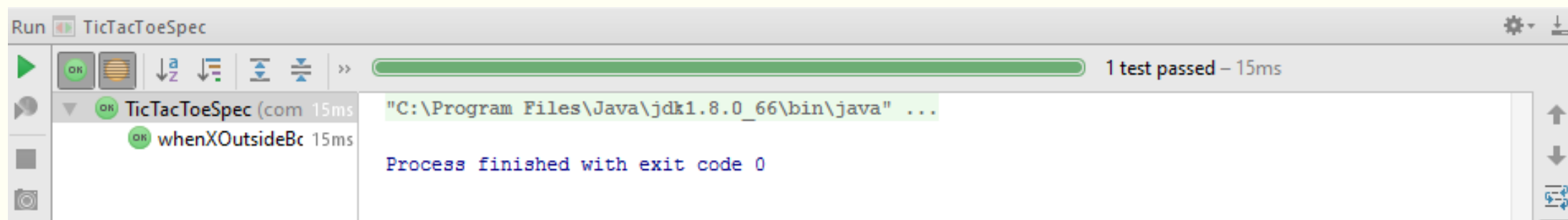


```
package com.packtpublishing.tddjava.ch03tictactoe;

public class TicTacToe {

    public void play(int x, int y) {
        if (x < 1 || x > 3) {
            throw
                new RuntimeException("X is outside board");
        }
    }
}
```

- Код містить самий необхідний мінімум для виконання вимоги.
 - Під *minimum* слід розуміти якомога менше *within reason*.
 - Ми не додаємо числа і нічого не повертаємо.
 - Орієнтуємось на невеличкі зміни дуже швидко



```
Run TicTacToeSpec
1 test passed - 15ms
TicTacToeSpec (com 15ms
  whenXOutsideBo 15ms
  "C:\Program Files\Java\jdk1.8.0_66\bin\java" ...
  Process finished with exit code 0
```

Test

- Цей тест такий же, як і попередній, проте потрібно тестувати вісь Y.

```
package com.packtpublishing.tddjava.ch03tictactoe;

import org.junit.Before;
import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.ExpectedException;

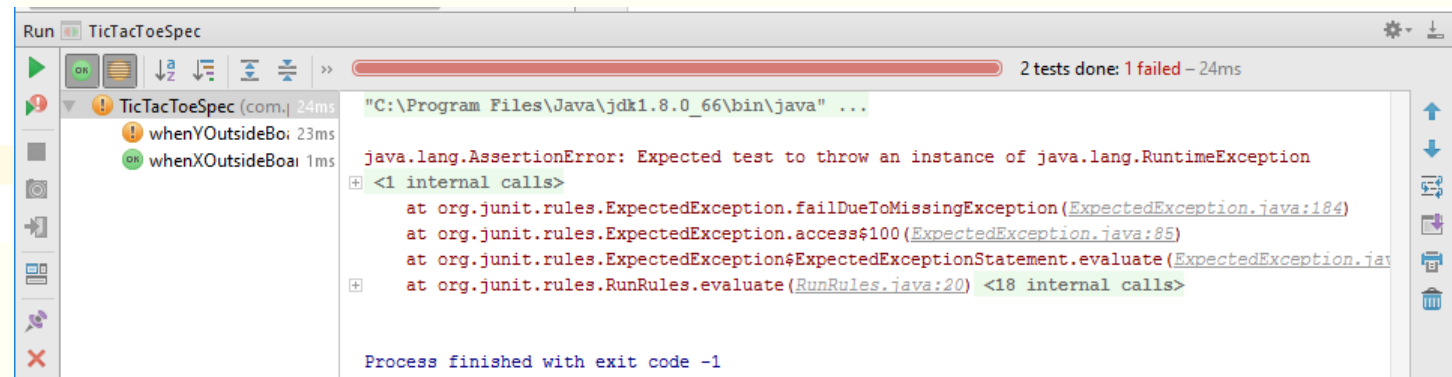
public class TicTacToeSpec {

    @Rule
    public ExpectedException exception =
        ExpectedException.none();
    private TicTacToe ticTacToe;

    @Before
    public final void before() {
        ticTacToe = new TicTacToe();
    }

    @Test
    public void whenXOutsideBoardThenRuntimeException()
    {
        exception.expect(RuntimeException.class);
        ticTacToe.play(5, 2);
    }

    @Test
    public void whenYOutsideBoardThenRuntimeException() {
        exception.expect(RuntimeException.class);
        ticTacToe.play(2, 5);
    }
}
```



Реалізація

```
TicTacToeSpec.java x  TicTacToe.java x  tdd-java-ch03-tic-tac-toe x
package com.packtpublishing.tddjava.ch03tictactoe;

public class TicTacToe {

    public void play(int x, int y) {
        if (x < 1 || x > 3) {
            throw
                new RuntimeException("X is outside board");
        } else if (y < 1 || y > 3) {
            throw
                new RuntimeException("X is outside board");
        }
    }
}
```

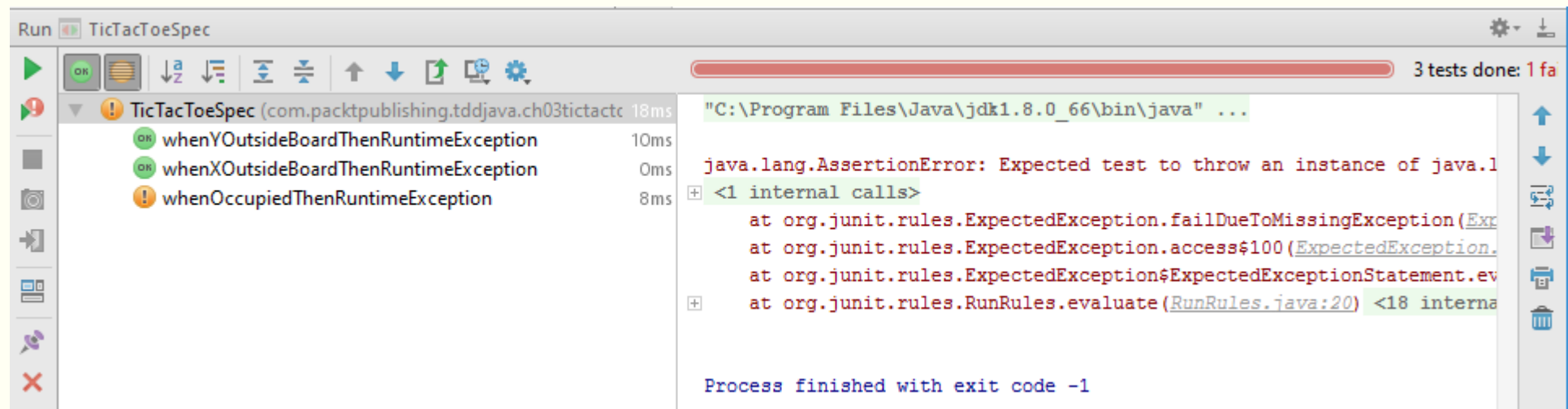
Run Rerun Failed Tests 1 test passed - 19ms

Tests	19ms
TicTacToeSpec	19ms
whenYOutsideBoardThenRuntimeException	19ms

Test

- Тепер ми знаємо, що фігури не будуть ставитись за межами дошки.
- Далі переконаємось, що вони будуть ставитись лише в порожні клітинки:

```
@Test
public void whenOccupiedThenRuntimeException() {
    ticTacToe.play(2, 1);
    exception.expect(RuntimeException.class);
    ticTacToe.play(2, 1);
}
```



Реалізація

```
private Character[][] board = {{'\0', '\0', '\0'},
                                {'\0', '\0', '\0'}, {'\0', '\0', '\0'}};

public void play(int x, int y) {
    if (x < 1 || x > 3) {
        throw
            new RuntimeException("X is outside board");
    } else if (y < 1 || y > 3) {
        throw
            new RuntimeException("Y is outside board");
    }
    if (board[x - 1][y - 1] != '\0') {
        throw
            new RuntimeException("Box is occupied");
    } else {
        board[x - 1][y - 1] = 'X';
    }
}
```

- Для реалізації останнього тесту збережемо розміщення поставлених фігур у масив.
 - При додаванні нової фігури буде потрібно перевіряти, щоб місце було незайняте. Інакше – викидати виключення:

TicTacToeSpec.java x TicTacToe.java x tdd-java-ch03-tic-tac-toe x

```
package com.packtpublishing.tddjava.ch03tictactoe;

public class TicTacToe {

    private Character[][] board = {{'\0', '\0', '\0'}, {'\0', '\0', '\p'}, {'\0', '\0', '\0'}};

    public void play(int x, int y) {
        if (x < 1 || x > 3) {
            throw
                new RuntimeException("X is outside board");
        } else if (y < 1 || y > 3) {
            throw
                new RuntimeException("Y is outside board");
        }
        if (board[x - 1][y - 1] != '\0') {
            throw
                new RuntimeException("Box is occupied");
        } else {
            board[x - 1][y - 1] = 'X';
        }
    }
}
```

Run Rerun Failed Tests

1 test passed - 15ms

Tests	Time	Output
OK Tests	15ms	"C:\Program Files\Java\jdk1.8.0_66\bin\java" ...
OK TicTacToeSpec	15ms	
OK whenOccupiedThenRuntimeException	15ms	Process finished with exit code 0

Рефакторинг

- Хоч тести виконуються, код дещо незрозумілий.
 - Слід виконати рефакторинг для відокремлення методів. Наприклад, так:

```
public void play(int x, int y) {
    checkAxis(x);
    checkAxis(y);
    setBox(x, y);
}

private void checkAxis(int axis) {
    if (axis < 1 || axis > 3) {
        throw
            new RuntimeException("X is outside board");
    }
}

private void setBox(int x, int y) {
    if (board[x - 1][y - 1] != '\0') {
        throw
            new RuntimeException("Box is occupied");
    } else {
        board[x - 1][y - 1] = 'X';
    }
}
```

Функціональність методу
play() не змінюється!


```
package com.packtpublishing.tddjava.ch03tictactoe;

public class TicTacToe {

    private Character[][] board = {{'\0', '\0', '\0'}, {'\0', '\0', '\0'}, {'\0', '\0', '\0'}};

    public void play(int x, int y) {
        checkAxis(x);
        checkAxis(y);
        setBox(x, y);
    }

    private void checkAxis(int axis) {
        if (axis < 1 || axis > 3) {
            throw
                new RuntimeException("X is outside board");
        }
    }

    private void setBox(int x, int y) {
        if (board[x - 1][y - 1] != '\0') {
            throw
                new RuntimeException("Box is occupied");
        } else {
            board[x - 1][y - 1] = 'X';
        }
    }
}
```

The screenshot shows the IntelliJ IDEA Run window for the `TicTacToeSpec` test class. The top bar indicates "Run" and "TicTacToeSpec". Below the toolbar, a green progress bar shows "All 3 tests passed". The test results table lists three tests, all marked as "OK":

Test Name	Duration
whenYOutsideBoardThenRuntimeException	11ms
whenXOutsideBoardThenRuntimeException	0ms
whenOccupiedThenRuntimeException	0ms

The console output shows the command used to run the tests: `"C:\Program Files\Java\jdk1.8.0_66\bin\java" ...` and the message: `Process finished with exit code 0`.

 All 3 tests passed

```
"C:\Program Files\Java\jdk1.8.0_66\bin\java" ...
```

```
Process finished with exit code 0
```

OK	whenYOutsideBoardThenRuntimeException	11ms
OK	whenXOutsideBoardThenRuntimeException	0ms
OK	whenOccupiedThenRuntimeException	0ms

Вимога 2

- Тепер будемо працювати над вибором ходу гравця
 - Повинен бути спосіб визначати, який гравець ходить наступним
- Розіб'ємо вимогу на 3 тести:
 - Перший хід має бути за гравцем, який грає хрестиком (X)
 - Якщо останній хід був за X, наступний має бути за нулик (O)
 - Якщо останній хід був за O, наступний має бути за X
- До цього моменту ми не використовували JUnit's asserts. Для їх підключення виконаємо статичний імпорт
 - `import static org.junit.Assert.*;`
- У нашому випадку потрібно порівнювати два символи: очікуваний та отриманий від методу `nextPlayer()`.

Test

- Першим має ходити X

```
@Test
public void givenFirstTurnWhenNextPlayerThenX() {
    assertEquals('X', ticTacToe.nextPlayer());
}
```

```
@Test
public void whenOccupiedThenRuntimeException() {
    ticTacToe.play(2, 1);
    exception.expect(RuntimeException.class);
    ticTacToe.play(2, 1);
}
```

press Alt+Enter

```
public void givenFirstTurnWhenNextPlayerThenX() {
    assertEquals('X', ticTacToe.nextPlayer());
}
```

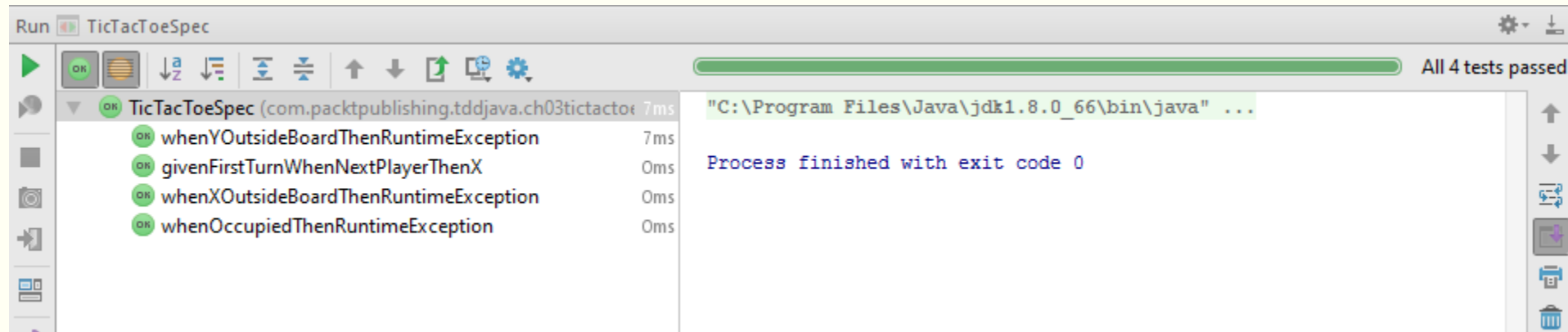


Create method 'nextPlayer'

Реалізація

- Поки що потреби перевіряти, чи ходить реальний гравець, немає

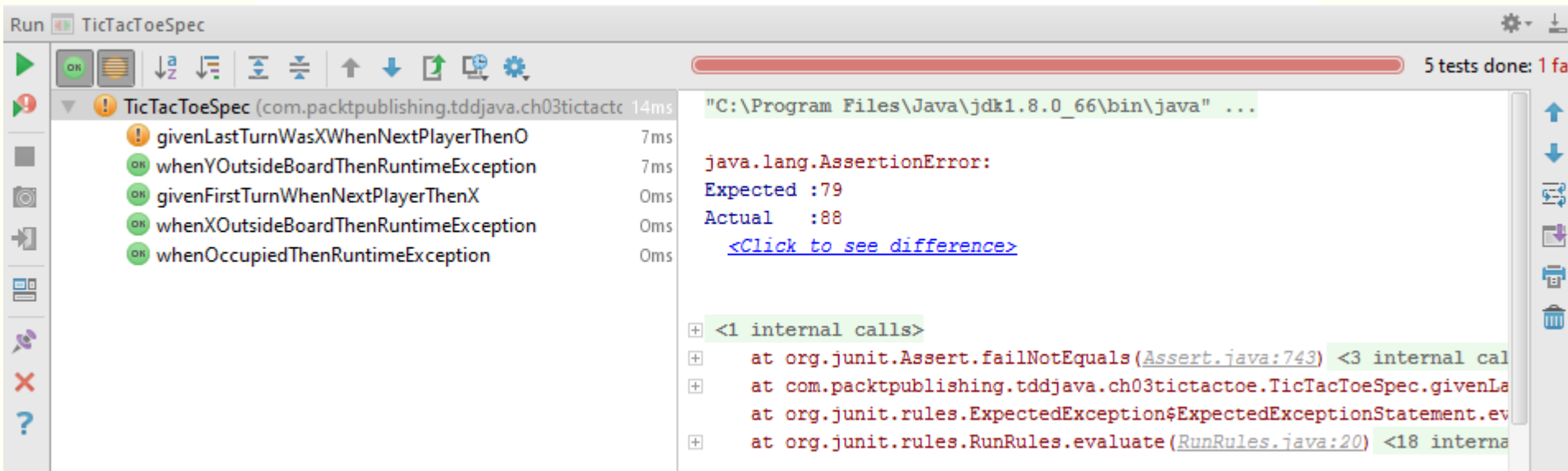
```
public char nextPlayer() {  
    return 'X';  
}
```



Test

- Тепер маємо переконатись, що гравці міняються. Після ходу X повинен бути хід O.

```
@Test
public void givenLastTurnWasXWhenNextPlayerThenO()
{
    ticTacToe.play(1, 1);
    assertEquals('O', ticTacToe.nextPlayer());
}
```



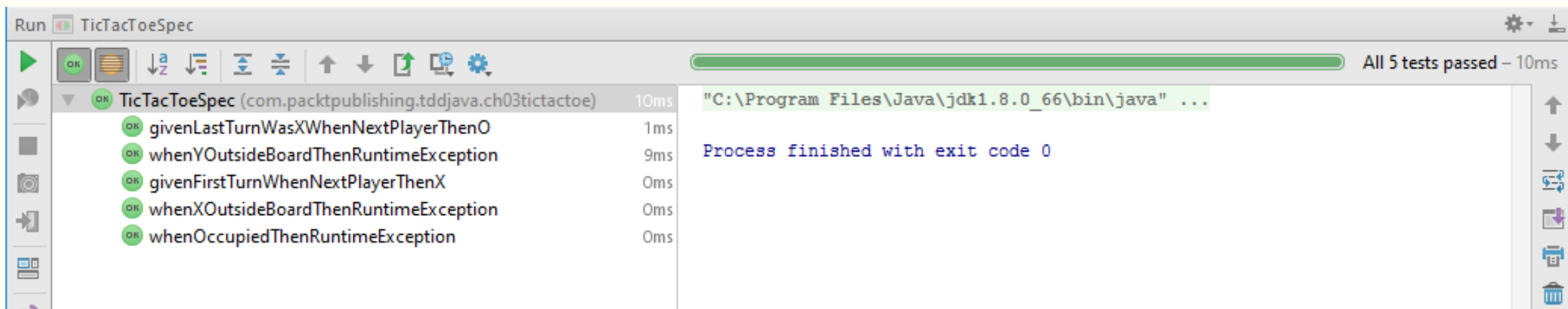
Реалізація

- Для того, щоб прослідкувати, хто має ходити, потрібно десь зберігати дані про те, хто ходив останній

```
private char lastPlayer = '\0';

public void play(int x, int y) {
    checkAxis(x);
    checkAxis(y);
    setBox(x, y);
    lastPlayer = nextPlayer();
}

public char nextPlayer() {
    if (lastPlayer == 'X') {
        return 'O';
    }
    return 'X';
}
```



Test

- Далі залишилось перевірити, чи йде за ходом О хід Х.
- Цей тест буде даремним, оскільки завжди буде хибно позитивним.
 - У нас **немає змін** реалізації

```
public char nextPlayer() {  
    if (lastPlayer == 'X') {  
        return 'O';  
    }  
    return 'X';  
}
```

- Якщо тест проходить успішно без внесення змін у код, його треба відкинути!

Вимога 3

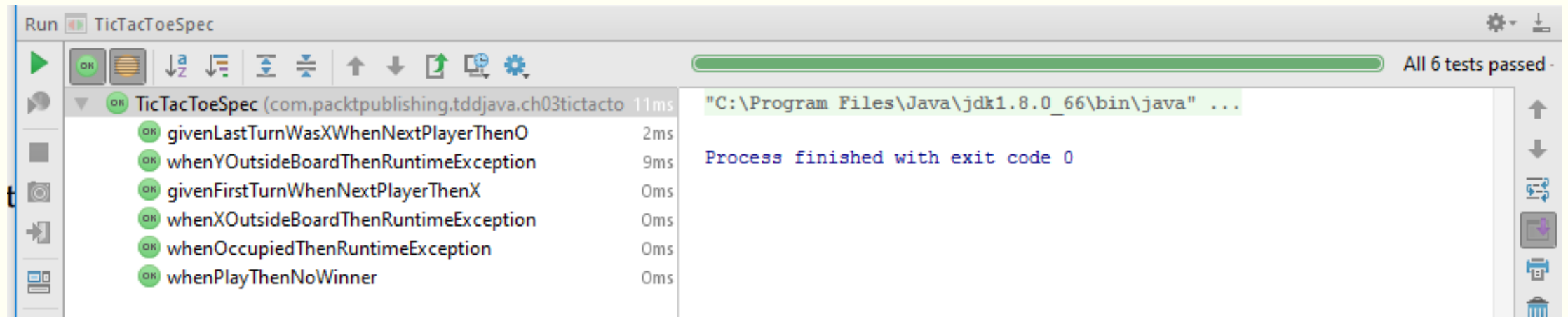
- Працюємо з правилами виграшу.
 - Ми повинні перевірити всі виграшні комбінації, і якщо одна з них виконується, оголосити переможця.
 - Для перевірки того, що отримано лінію однакових фігур, потрібно оглянути горизонтальні, вертикальні та діагональні лінії.
- Тест:
 - Визначимо відповідь методу play() за замовчуванням

```
@Test
public void whenPlayThenNoWinner()
{
    String actual = ticTacToe.play(1,1);
    assertEquals("No winner", actual);
}
```

- Якщо умова перемоги не виконується, переможця немає

Реалізація

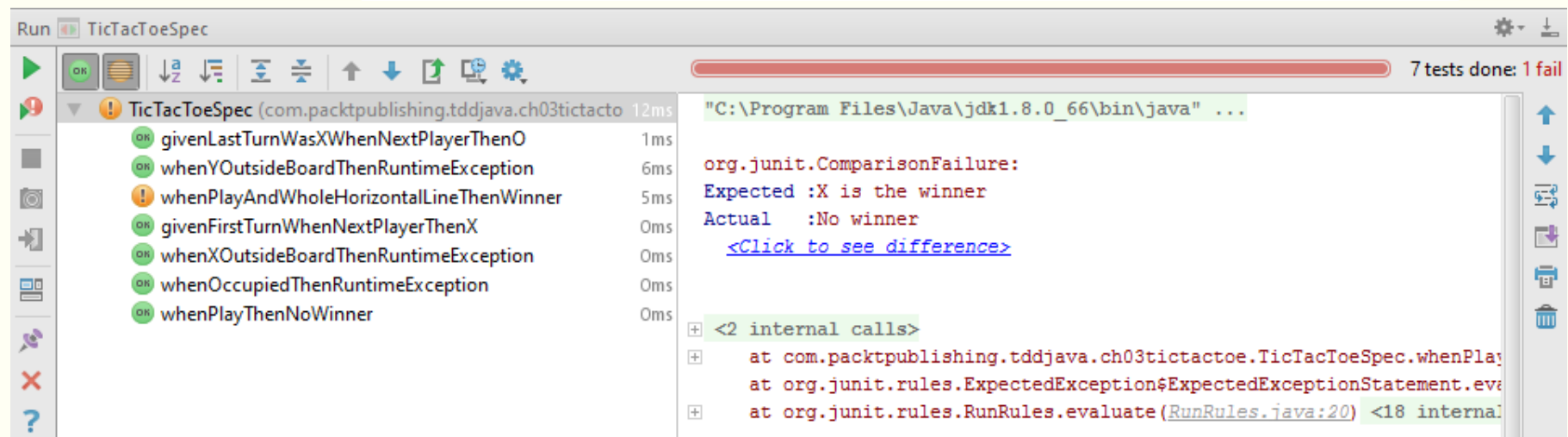
```
public String play(int x, int y) {  
    checkAxis(x);  
    checkAxis(y);  
    setBox(x, y);  
    lastPlayer = nextPlayer();  
    return "No winner";  
}
```



Тест

- Описавши відповідь за замовчуванням (переможця немає), слід почати роботу над умовами перемоги:

```
@Test
public void whenPlayAndWholeHorizontalLineThenWinner() {
    ticTacToe.play(1, 1); // X
    ticTacToe.play(1, 2); // O
    ticTacToe.play(2, 1); // X
    ticTacToe.play(2, 2); // O
    String actual = ticTacToe.play(3, 1); // X
    assertEquals("X is the winner", actual);
}
```

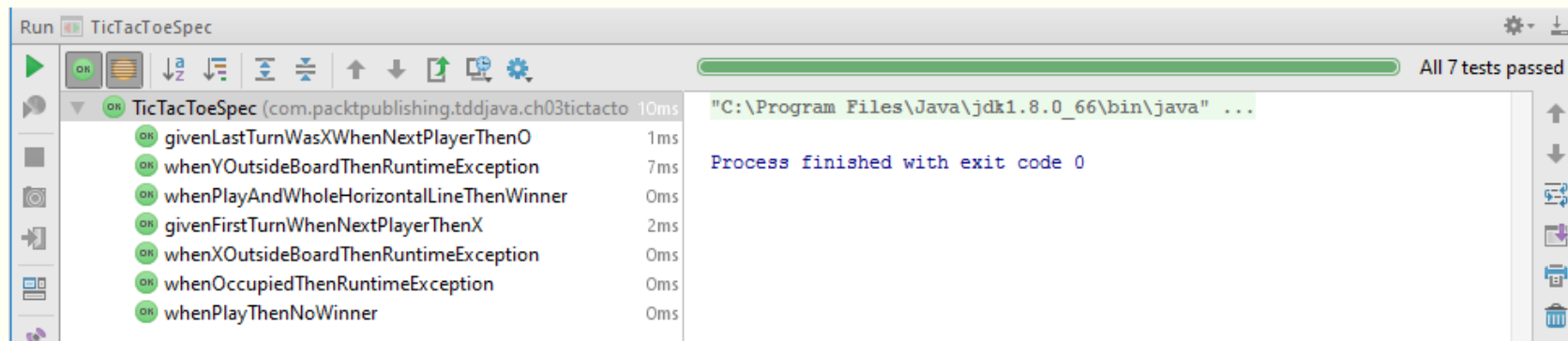


Реалізація

- Для виконання тесту потрібно перевірити, чи якась горизонтальна лінія заповнена однаковими позначками, такими ж, як від поточного гравця.
 - До цього моменту нас не цікавило, що клалось на the board array.
 - Тепер потрібно розуміти не лише, яка комірка порожня, але й ким з гравців вона заповнена.

```
public String play(int x, int y) {
    checkAxis(x);
    checkAxis(y);
    lastPlayer = nextPlayer();
    setBox(x, y, lastPlayer);
    for (int index = 0; index < 3; index++) {
        if (board[0][index] == lastPlayer &&
            board[1][index] == lastPlayer &&
            board[2][index] == lastPlayer) {
            return lastPlayer + " is the winner";
        }
    }
    return "No winner";
}

private void setBox(int x, int y, char lastPlayer)
{
    if (board[x - 1][y - 1] != '\0') {
        throw
            new RuntimeException("Box is occupied");
    } else {
        board[x - 1][y - 1] = lastPlayer;
    }
}
```



Рефакторинг

```
private static final int SIZE = 3;
public String play(int x, int y) {
    checkAxis(x);
    checkAxis(y);
    lastPlayer = nextPlayer();
    setBox(x, y, lastPlayer);
    if (isWin()) {
        return lastPlayer + " is the winner";
    }
    return "No winner";
}

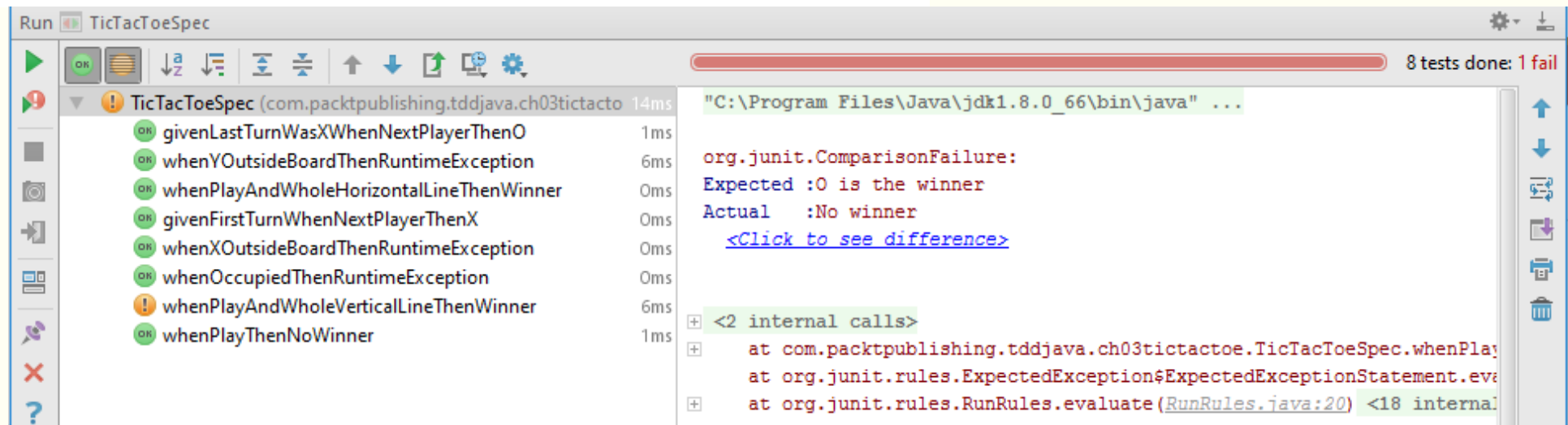
private boolean isWin() {
    for (int i = 0; i < SIZE; i++) {
        if (board[0][i] + board[1][i] + board[2][i]
            == (lastPlayer * SIZE)) {
            return true;
        }
    }
    return false;
}
```

- Отриманий код не обов'язково буде фінальною версією.
 - Він виконує свою мету – забезпечити якомога швидше покриття коду.
 - Нова версія вивела логіку виграшу в окремий метод

Тест

```
@Test
public void whenPlayAndWholeVerticalLineThenWinner() {
    ticTacToe.play(2, 1); // X
    ticTacToe.play(1, 1); // O
    ticTacToe.play(3, 1); // X
    ticTacToe.play(1, 2); // O
    ticTacToe.play(2, 2); // X
    String actual = ticTacToe.play(1, 3); // O
    assertEquals("O is the winner", actual);
}
```

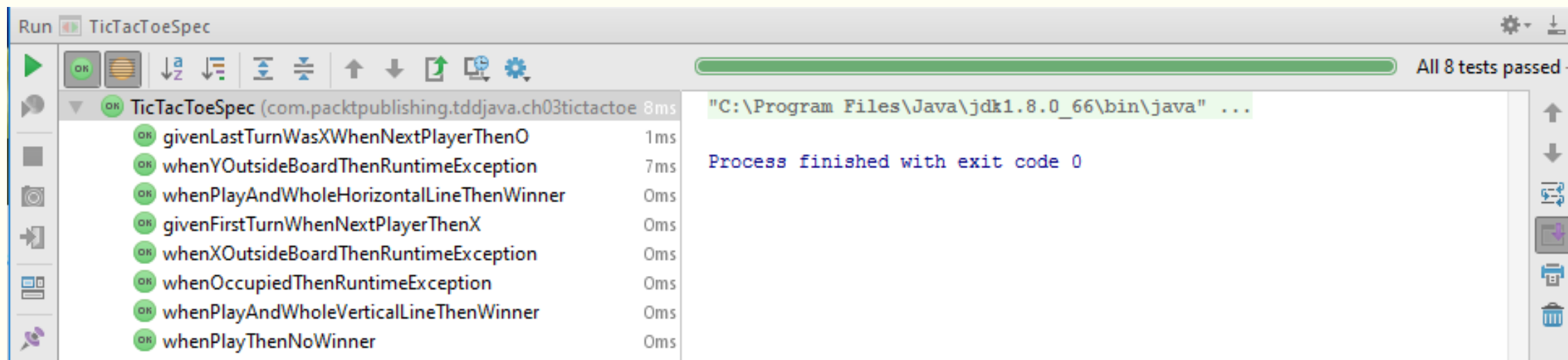
- Також потрібно перевірити виграшні вертикальні лінії



Реалізація

- Загалом повинна бути аналогічна попередній.

```
private boolean isWin() {  
    int playerTotal = lastPlayer * SIZE;  
    for (int i = 0; i < SIZE; i++) {  
        if ((board[0][i] + board[1][i] + board[2][i]) == playerTotal) {  
            return true;  
        } else if ((board[i][0] + board[i][1] + board[i][2]) == playerTotal) {  
            return true;  
        }  
    }  
  
    return false;  
}
```



The screenshot shows an IDE window titled "Run" with a sub-header "TicTacToeSpec". Below the header is a toolbar with various icons. The main area displays a list of test cases, each with a green "OK" status icon, the test name, and its execution time. To the right of the list, the command used to run the tests is shown: "C:\Program Files\Java\jdk1.8.0_66\bin\java" Below this, the output indicates "Process finished with exit code 0". On the far right, a green progress bar and the text "All 8 tests passed" are visible.

Test Case	Time
givenLastTurnWasXWhenNextPlayerThenO	1ms
whenYOutsideBoardThenRuntimeException	7ms
whenPlayAndWholeHorizontalLineThenWinner	0ms
givenFirstTurnWhenNextPlayerThenX	0ms
whenXOutsideBoardThenRuntimeException	0ms
whenOccupiedThenRuntimeException	0ms
whenPlayAndWholeVerticalLineThenWinner	0ms
whenPlayThenNoWinner	0ms

"C:\Program Files\Java\jdk1.8.0_66\bin\java" ...

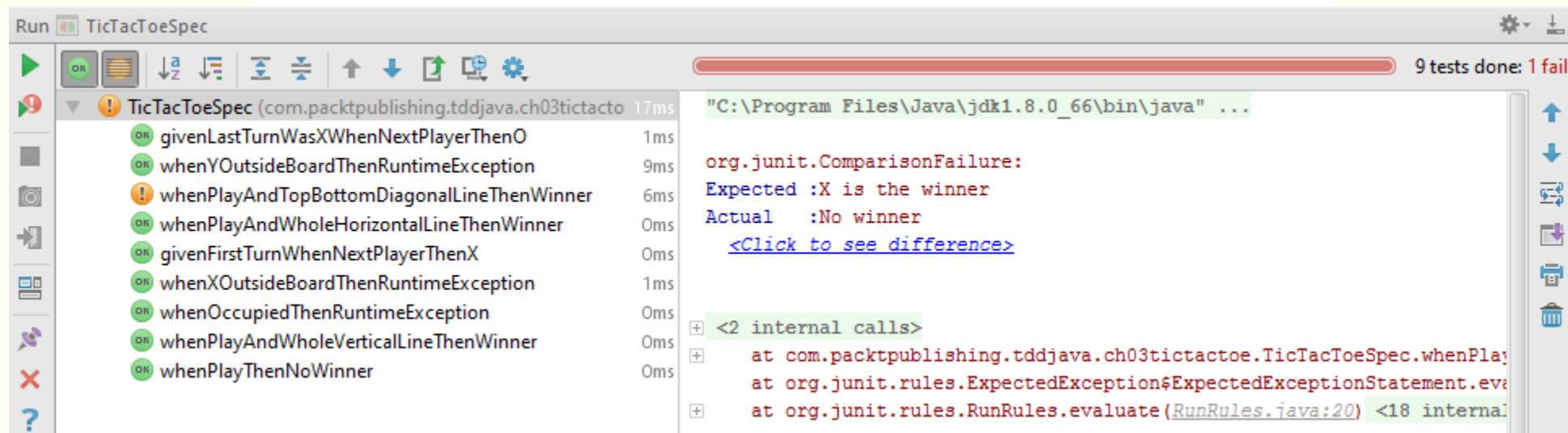
Process finished with exit code 0

All 8 tests passed

Test

- Далі звертаємо увагу на діагональні комбінації. Спершу головна діагональ

```
@Test
public void whenPlayAndTopBottomDiagonalLineThenWinner() {
    ticTacToe.play(1, 1); // X
    ticTacToe.play(1, 2); // O
    ticTacToe.play(2, 2); // X
    ticTacToe.play(1, 3); // O
    String actual = ticTacToe.play(3, 3); // O
    assertEquals("X is the winner", actual);
}
```



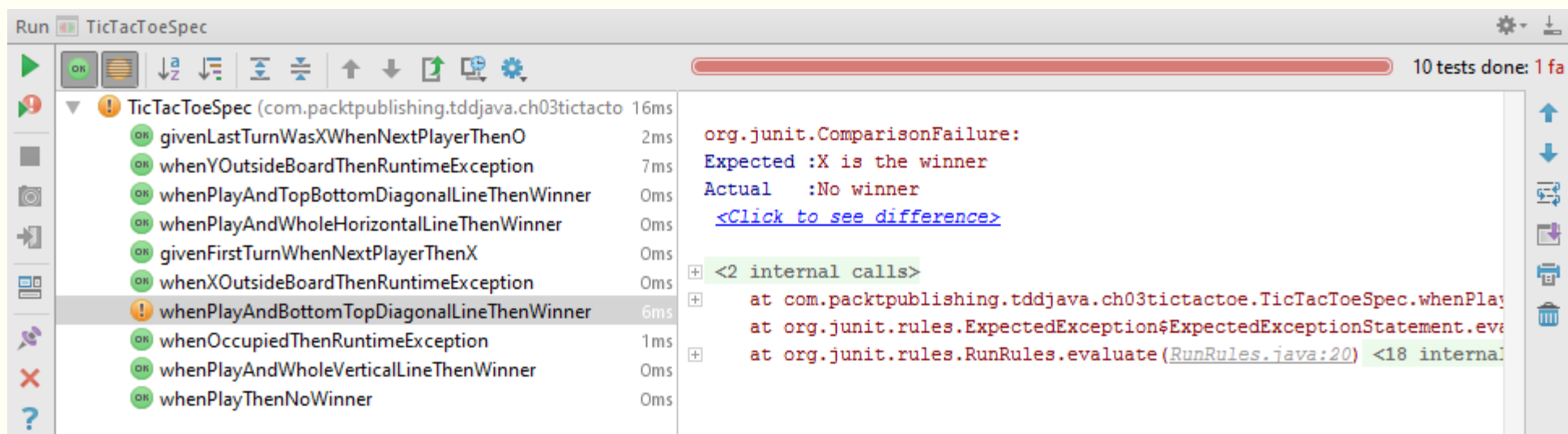
Реалізація

```
private boolean isWin() {  
    int playerTotal = lastPlayer * SIZE;  
  
    for (int i = 0; i < SIZE; i++) {  
        if ((board[0][i] + board[1][i] + board[2][i]) == playerTotal) {  
            return true;  
        } else if ((board[i][0] + board[i][1] + board[i][2]) == playerTotal) {  
            return true;  
        } else if ((board[0][0] + board[1][1] + board[2][2]) == playerTotal) {  
            return true;  
        }  
    }  
  
    return false;  
}
```


Test

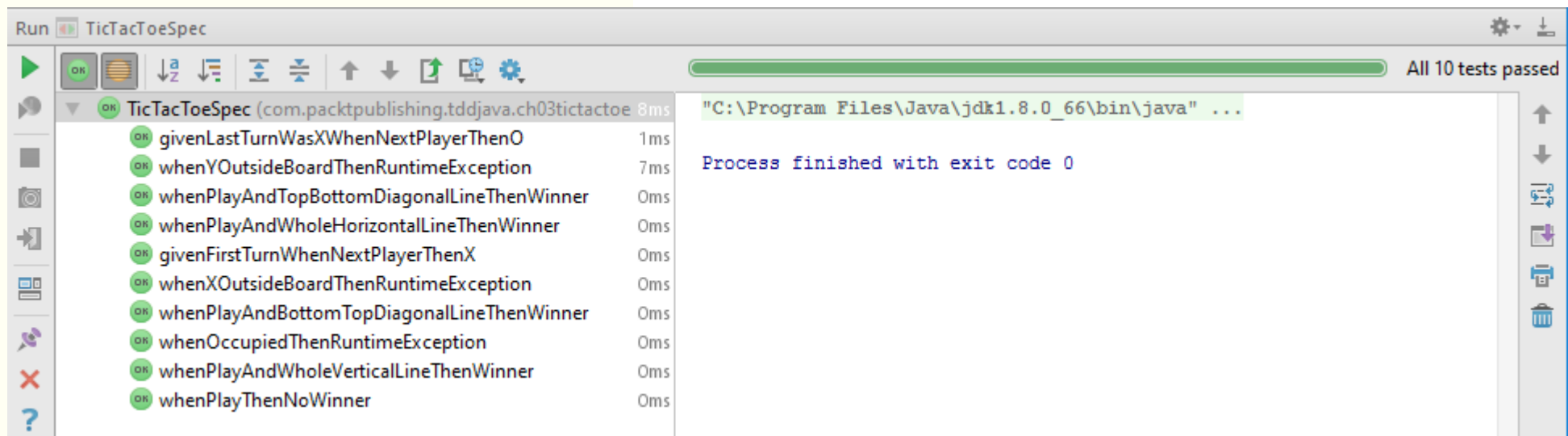
- Тест побічної діагоналі

```
@Test
public void whenPlayAndBottomTopDiagonalLineThenWinner() {
    ticTacToe.play(1, 3); // X
    ticTacToe.play(1, 1); // O
    ticTacToe.play(2, 2); // X
    ticTacToe.play(1, 2); // O
    String actual = ticTacToe.play(3, 1); // O
    assertEquals("X is the winner", actual);
}
```



Реалізація (аналогічна)

```
private boolean isWin() {  
    int playerTotal = lastPlayer * SIZE;  
  
    for (int i = 0; i < SIZE; i++) {  
        if ((board[0][i] + board[1][i] + board[2][i]) == playerTotal) {  
            return true;  
        } else if ((board[i][0] + board[i][1] + board[i][2]) == playerTotal) {  
            return true;  
        } else if ((board[0][0] + board[1][1] + board[2][2]) == playerTotal) {  
            return true;  
        }  
        else if ((board[2][0] + board[1][1] + board[0][2]) == playerTotal) {  
            return true;  
        }  
    }  
  
    return false;  
}
```



Рефакторинг

```
private boolean isWin() {
    int playerTotal = lastPlayer * SIZE;
    char diagonal1 = '\0';
    char diagonal2 = '\0';
    for (int i = 0; i < SIZE; i++) {
        diagonal1 += board[i][i];
        diagonal2 += board[i][SIZE - i - 1];
        if ((board[0][i] + board[1][i] + board[2][i]) == playerTotal) {
            return true;
        } else if ((board[i][0] + board[i][1] + board[i][2]) == playerTotal) {
            return true;
        }
    }
    if (diagonal1 == playerTotal || diagonal2 == playerTotal) {
        return true;
    }
    return false;
}
```

Вимога 4

- Обробляти ситуацію повного заповнення всіх комірок

```
@Test
public void whenAllBoxesAreFilledThenDraw() {
    ticTacToe.play(1, 1);
    ticTacToe.play(1, 2);
    ticTacToe.play(1, 3);
    ticTacToe.play(2, 1);
    ticTacToe.play(2, 3);
    ticTacToe.play(2, 2);
    ticTacToe.play(3, 1);
    ticTacToe.play(3, 3);
    String actual = ticTacToe.play(3, 2);
    assertEquals("The result is draw", actual);
}
```

The screenshot shows an IDE window titled "Run" for "TicTacToeSpec". The test results pane on the left lists 11 tests. The test "whenAllBoxesAreFilledThenDraw" is marked with a red exclamation mark and a duration of 11ms. The details pane on the right shows the failure message: "org.junit.ComparisonFailure: Expected :The result is draw Actual :No winner". The stack trace includes the test method and internal JUnit calls. The bottom status bar indicates "Process finished with exit code -1".

Test Name	Duration	Status
givenLastTurnWasXWhenNextPlayerThenO	1ms	OK
whenYOutsideBoardThenRuntimeException	7ms	OK
whenPlayAndTopBottomDiagonalLineThenWinner	0ms	OK
whenPlayAndWholeHorizontalLineThenWinner	0ms	OK
givenFirstTurnWhenNextPlayerThenX	0ms	OK
whenXOutsideBoardThenRuntimeException	0ms	OK
whenPlayAndBottomTopDiagonalLineThenWinner	0ms	OK
whenOccupiedThenRuntimeException	0ms	OK
whenAllBoxesAreFilledThenDraw	11ms	Failed
whenPlayAndWholeVerticalLineThenWinner	0ms	OK
whenPlayThenNoWinner	0ms	OK

11 tests done: 1 failed

org.junit.ComparisonFailure:
Expected :The result is draw
Actual :No winner
[Click to see difference](#)

<2 internal calls>
at com.packtpublishing.tddjava.ch03tictactoe.TicTacToeSpec.whenAllBoxesAreFilledThenDraw(TicTacToeSpec.java:20)
at org.junit.rules.ExpectedException\$ExpectedExceptionStatement.evaluate(ExpectedException\$ExpectedExceptionStatement.java:60)
at org.junit.rules.RunRules.evaluate(RunRules.java:20) <18 internal calls>

Process finished with exit code -1

```

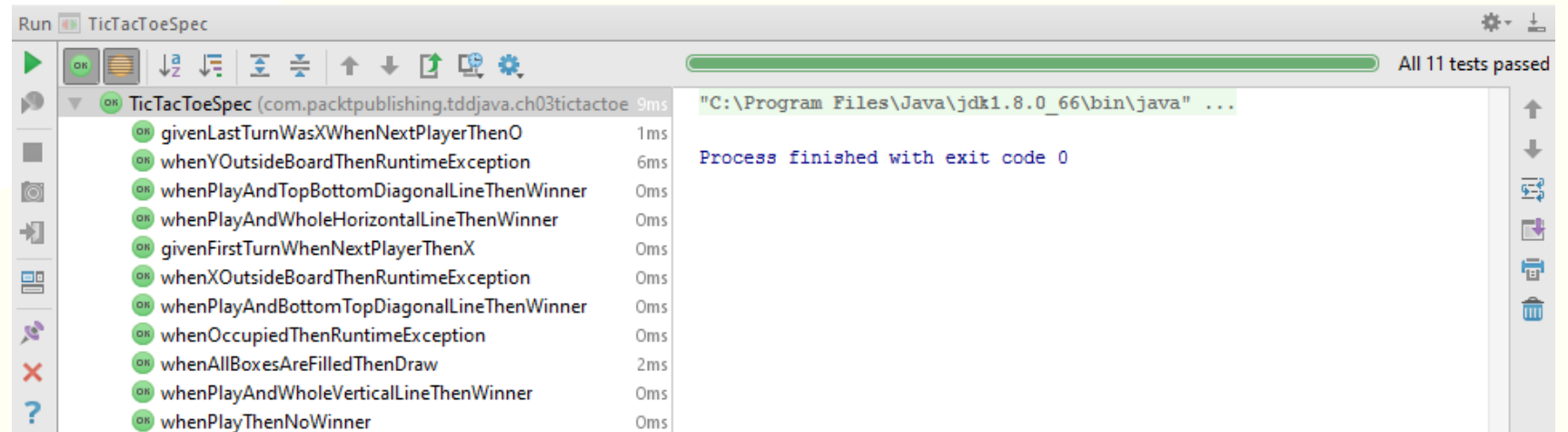
public String play(int x, int y) {
    checkAxis(x);
    checkAxis(y);
    lastPlayer = nextPlayer();
    setBox(x, y, lastPlayer);
    if (isWin()) {
        return lastPlayer + " is the winner";
    } else if (isDraw()) {
        return "The result is draw";
    } else {
        return "No winner";
    }
}

private boolean isDraw() {
    for (int x = 0; x < SIZE; x++) {
        for (int y = 0; y < SIZE; y++) {
            if (board[x][y] == '\0') {
                return false;
            }
        }
    }
    return true;
}

```

Реалізація

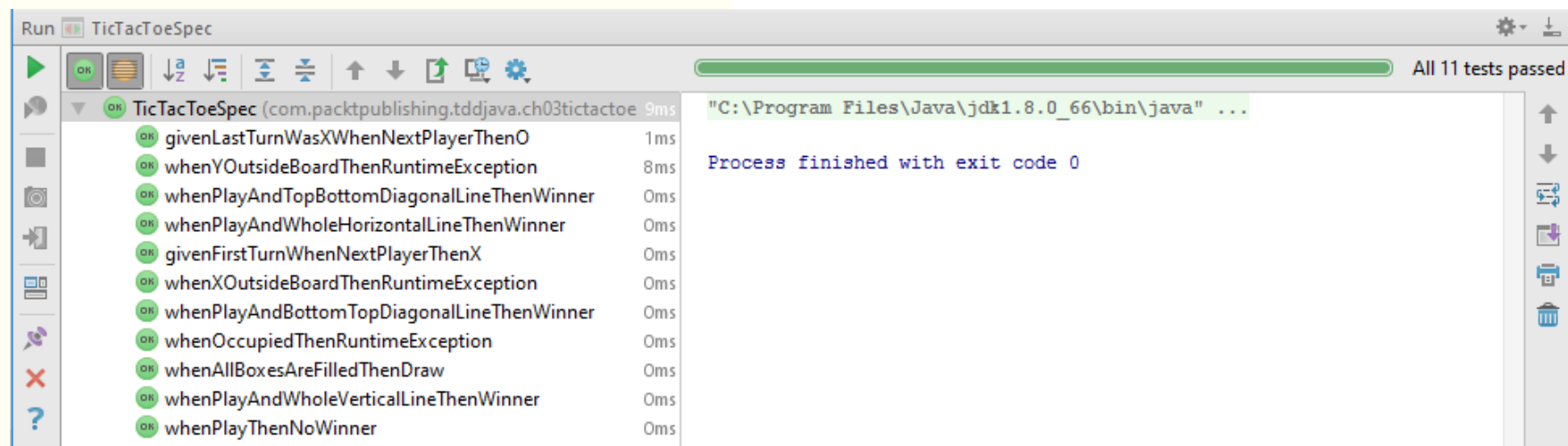
- Потрібно перевірити заповненість всіх комірок.



Рефакторинг

- Слід зауважити, що насправді не потрібно перевіряти всі комірки.
- Лише ту, що відноситься до позиції останньої фігури, що зіграла.

```
private boolean isWin(int x, int y) {  
    int playerTotal = lastPlayer * 3;  
    char horizontal, vertical, diagonal1, diagonal2;  
    horizontal = vertical = diagonal1 = diagonal2 = '\0';  
    for (int i = 0; i < SIZE; i++) {  
        horizontal += board[i][y - 1];  
        vertical += board[x - 1][i];  
        diagonal1 += board[i][i];  
        diagonal2 += board[i][SIZE - i - 1];  
    }  
    if (horizontal == playerTotal  
        || vertical == playerTotal  
        || diagonal1 == playerTotal  
        || diagonal2 == playerTotal) {  
        return true;  
    }  
    return false;  
}
```



Покриття коду (Code coverage)

- До цього фокусувались на red-green-refactor model.
- Питання: чи покривають ваші тести всі випадки?
 - Відповідь дають інструменти перевірки покриття коду, наприклад, JaCoCo
 - Такі інструменти спроектовано, в основному, для перевірки достатності тестів, що постачаються.
 - У TDD зворотний підхід.
- Для підключення JaCoCo записують у build.gradle:
apply plugin: 'jacoco'