



Разработка через тестирование Введение, Bowling Game Kata

Test Driven Development

Ivan Dyachenko <IDyachenko@luxoft.com>

Для кого этот тренинг?

1

Beginner

Хорошая точка входа

2

Intermediate

Поможет лучше всё структурировать в голове и
объяснять коллегам

3

Advanced

Можно использовать для обучения и проверки
других

Содержание

1

Введение в TDD

2

Extreme Programming Practices

3

TDD "Red-Green-Refactor"

4

Обзор JUnit

5

Bowling Game Kata

6

Workshop

7

Стратегии запуска тестов

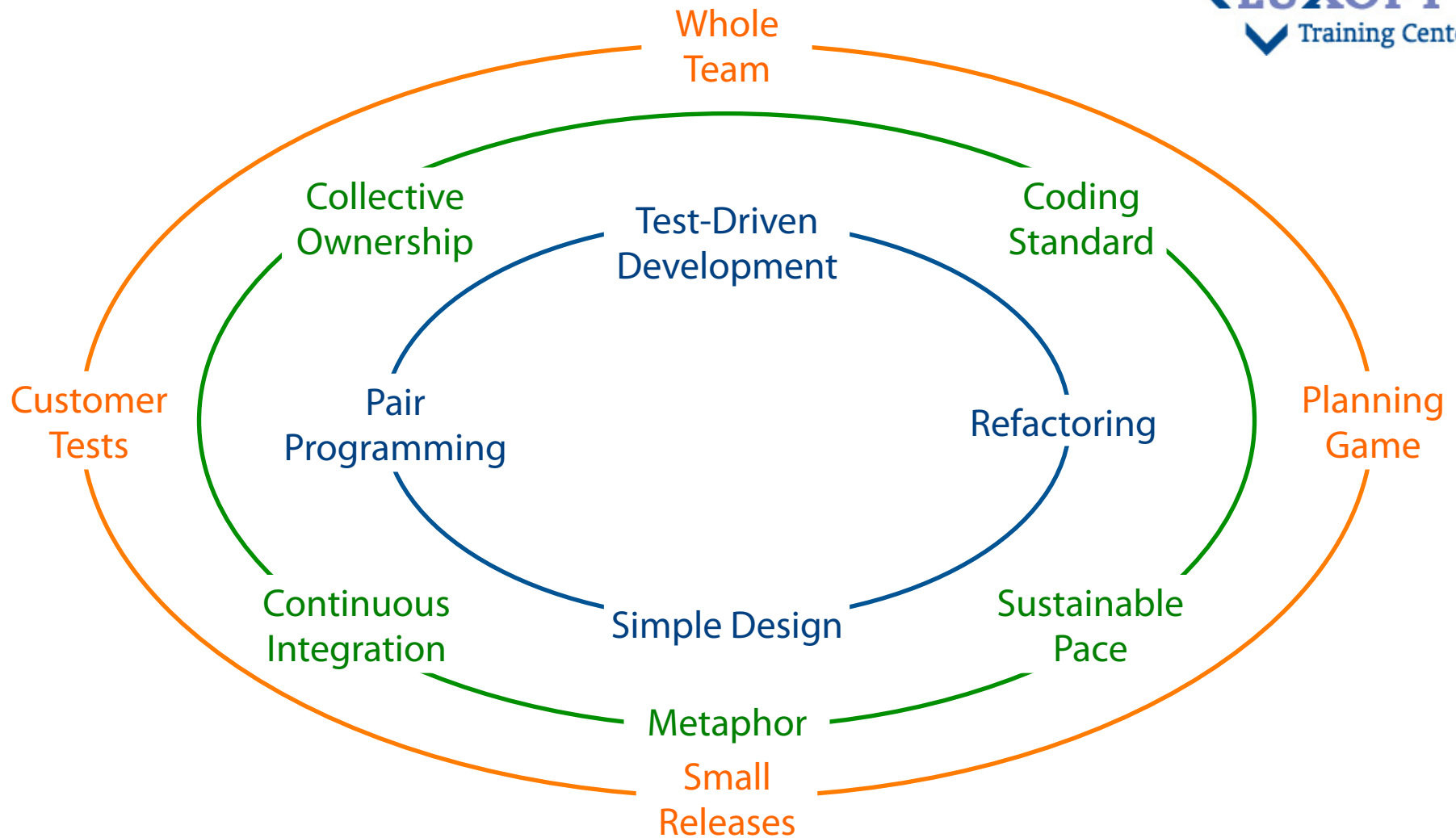
Кент Бек



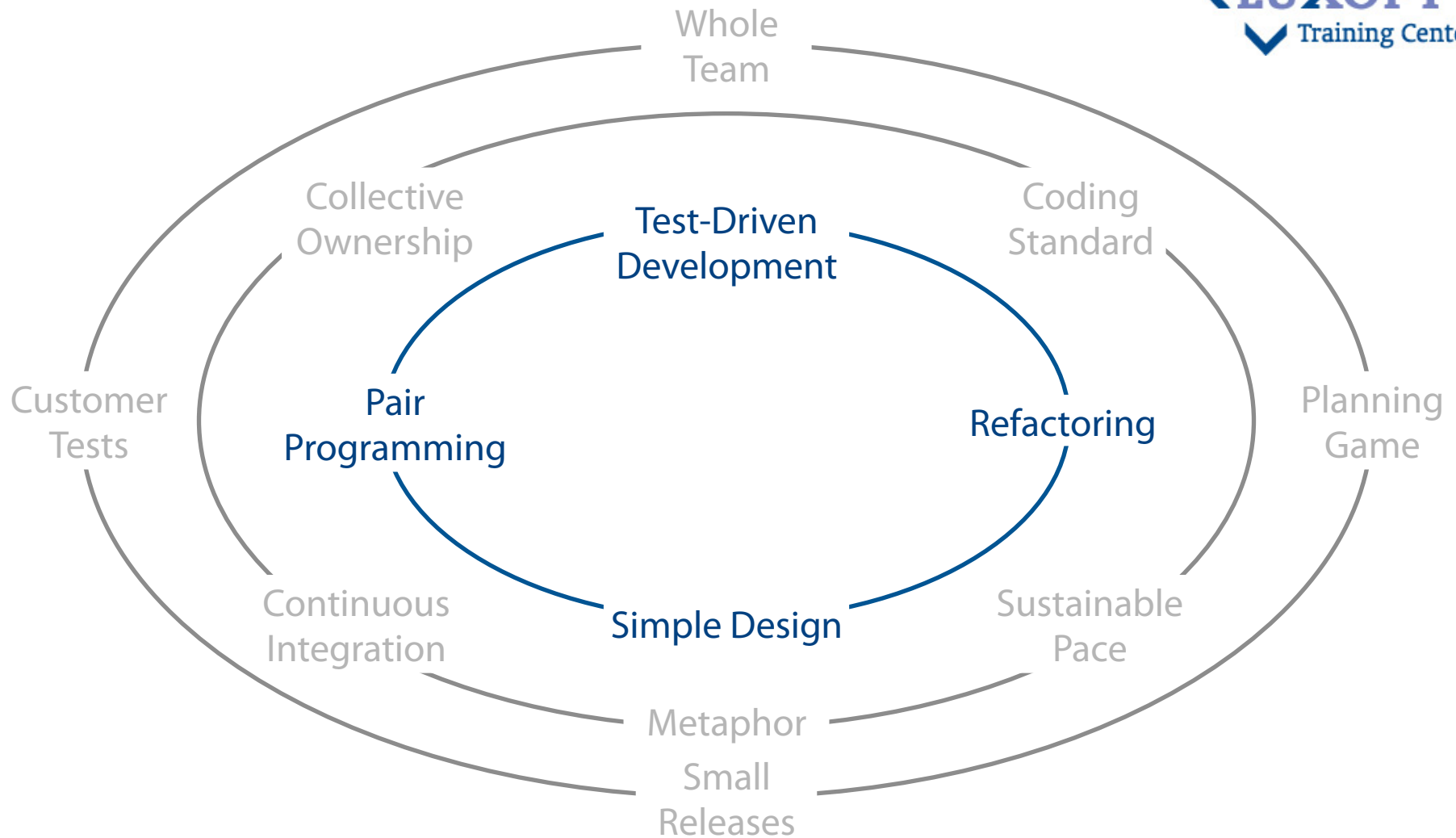
Экстремальное
программирование (XP)

Разработка через
тестирование (TDD)

Agile Manifesto в 2001
году.



Extreme Programming Practices



Extreme Programming Practices

«Чистый код, который работает»

Мифы TDD

Unit tests == TDD

TDD == 100% coverage

TDD == время * 2

TDD == серебряная пуля

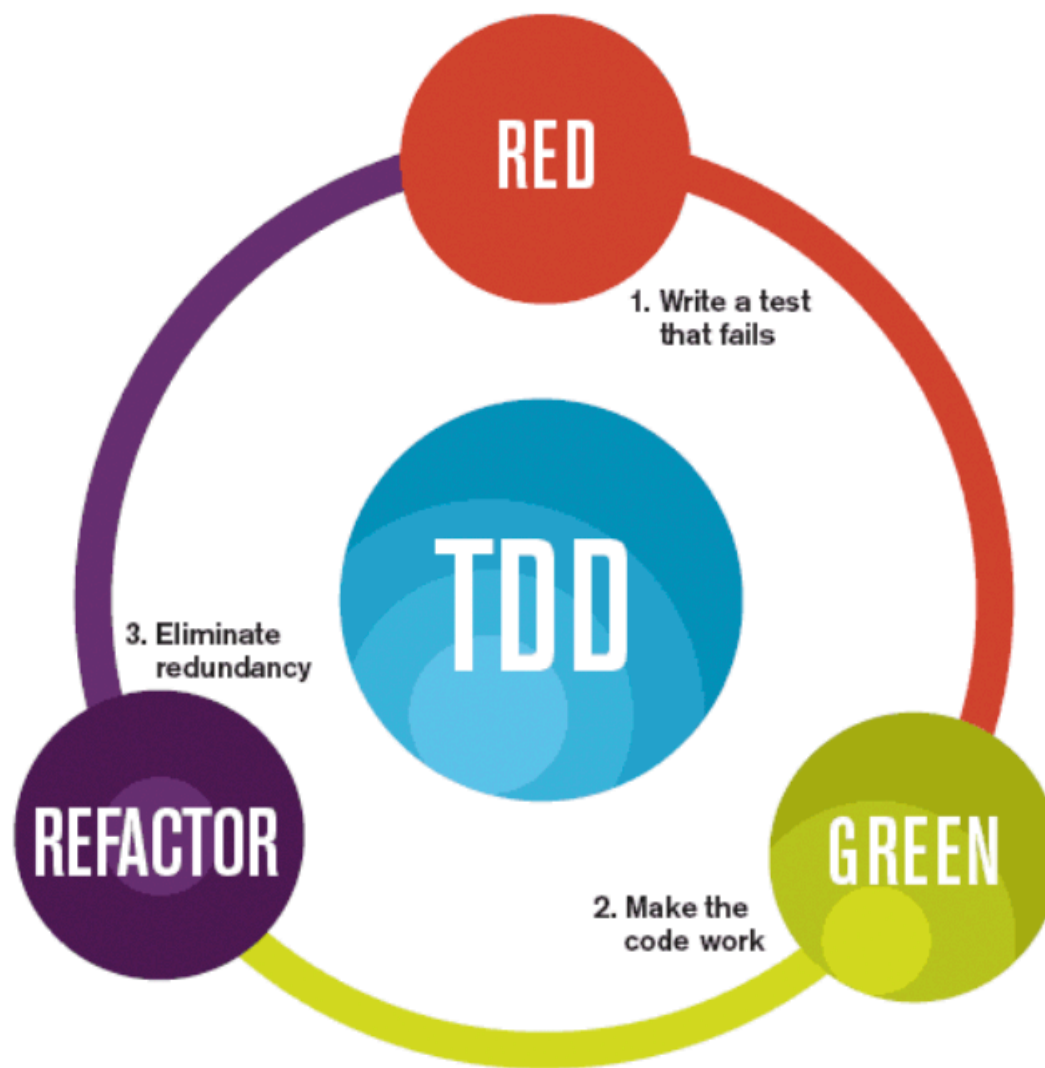
Два простых правила TDD

1

Пишем новый код только тогда, когда автоматический код не сработал

2

Удаляем дублирование



Мантра TDD – “red, green, refactor”

RED / GREEN / REFACTOR



0

Подумай

RED / GREEN / REFACTOR

0

Подумай

1

Напиши тест

RED / GREEN / REFACTOR

0

Подумай

1

Напиши тест

2

Скомпилируй

RED / GREEN / REFACTOR

0

Подумай

1

Напиши тест

2

Скомпилируй

3

Исправь ошибки

RED / GREEN / REFACTOR

0

Подумай

1

Напиши тест

2

Скомпилируй

3

Исправь ошибки

4

Запусти и убедись что тесты упали

RED / GREEN / REFACTOR

0

Подумай

1

Напиши тест

2

Скомпилируй

3

Исправь ошибки

4

Запусти и убедись что тесты упали

5

Напиши код

RED / GREEN / REFACTOR

0

Подумай

1

Напиши тест

2

Скомпилируй

3

Исправь ошибки

4

Запусти и убедись что тесты упали

5

Напиши код

6

Запусти и убедись что тесты прошли

RED / GREEN / REFACTOR

0

Подумай

1

Напиши тест

2

Скомпилируй

3

Исправь ошибки

4

Запусти и убедись что тесты упали

5

Напиши код

6

Запусти и убедись что тесты прошли

7

Рефакторинг

RED / GREEN / REFACTOR

0

Подумай

1

Напиши тест

2

Скомпилируй

3

Исправь ошибки

4

Запусти и убедись что тесты упали

5

Напиши код

6

Запусти и убедись что тесты прошли

7

Рефакторинг

Повтори



Ошибки TDD

Писать тест после кода

Писать тест который сразу проходит

Писать сразу много тестов

Asserts

- assertEquals
- assertFalse
- assertNotNull
- assertNull
- assertNotSame
- assertEquals
- assertTrue

TestCase

- run
- setUp
- tearDown

Annotations

- @Test
- @Before
- @After

JUnit Annotations

- `@Test`
- `@Before`
- `@After`
- `@BeforeClass`
- `@AfterClass`
- `@Ignore`
- `@Test (expected = Exception.class)`
- `@Test(timeout=100)`

Пример



Bowling Game Kata - подсчет очков игры в боулинг

www.objectmentor.com

wiki.agiledev.ru

www.slideshare.net

AMF FRONTIER LANES































7300 EAST THOMAS RD - SCOTTSDALE

(480) 946.5308

1/30/2008

Score

7:15:20PM

Team Team 3		Lane 3				Game 1				1/30/2008		
Player		1	2	3	4	5	6	7	8	9	10	Total
BRIAN		- 8	⑦ -	- 8	7 1	- -	8 -	5 2	- 6	7 -	⑧ -	
Hdcp	0	8	15	23	31	31	39	46	52	59	67	67
												
JOHN		⑧ 1	6 -	9 -	- -	6 2	1 4	8 1	9 -	4 4	8 / 6	
Hdcp	0	9	15	24	24	32	37	46	55	63	79	79
												
TOM		X	X	7 /	9 /	9 -	X	7 /	X	9 /	X X 7	
Hdcp	0	27	47	66	85	94	114	134	154	174	201	201
												

Правила игры в боулинг

Раунды

- 10 раундов (frame-ов)
- В одном frame – 2 броска
- Цель – выбить кегли (pins)
- Выигрывает набравший больше всех очков

Подсчет очков

- 10 сразу (strike). Заканчивается досрочно.
10 очков + pins + pins
- 10 очков со второго броска (spare)
10 очков + pins
- Max 300 очков (12 strikes)

Последний frame

- 1-й strike – дает возможность бросить еще два раза
- 2-й spare – дает возможность бросить еще один раз

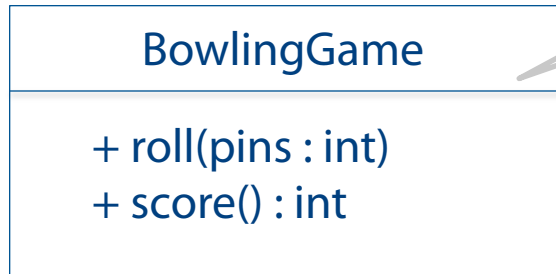
Надо написать

BowlingGame
+ roll(pins : int) + score() : int

Написать класс BowlingGame, который имеет два метода

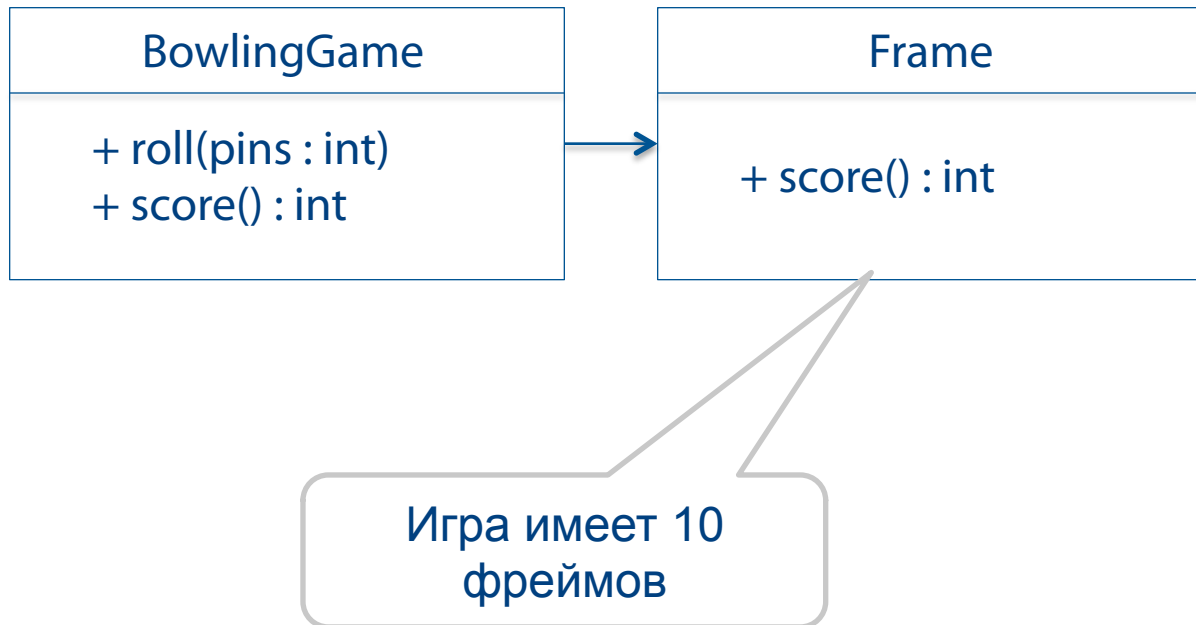
- roll (pins : int) – вызывается каждый раз когда игрок бросает шар. Pins – количество выбитых кеглей в этом броске
- score() : int – вызывается в конце игры. Показывает результат игры

A quick design session

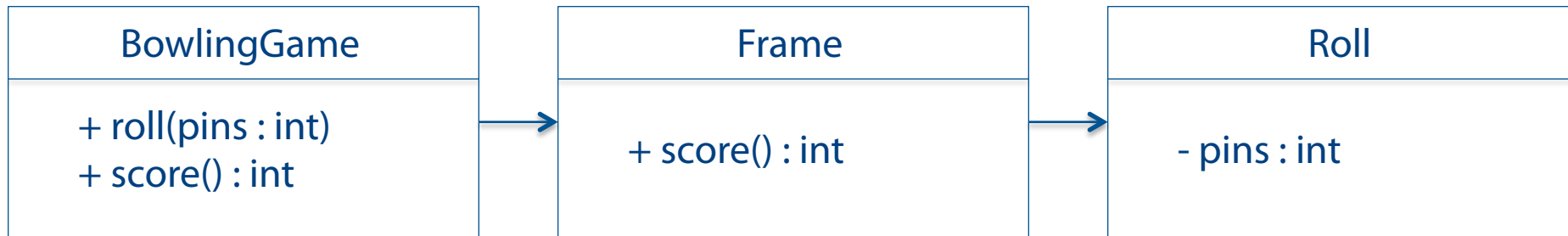


Нам нужен
BowlingGame класс

A quick design session

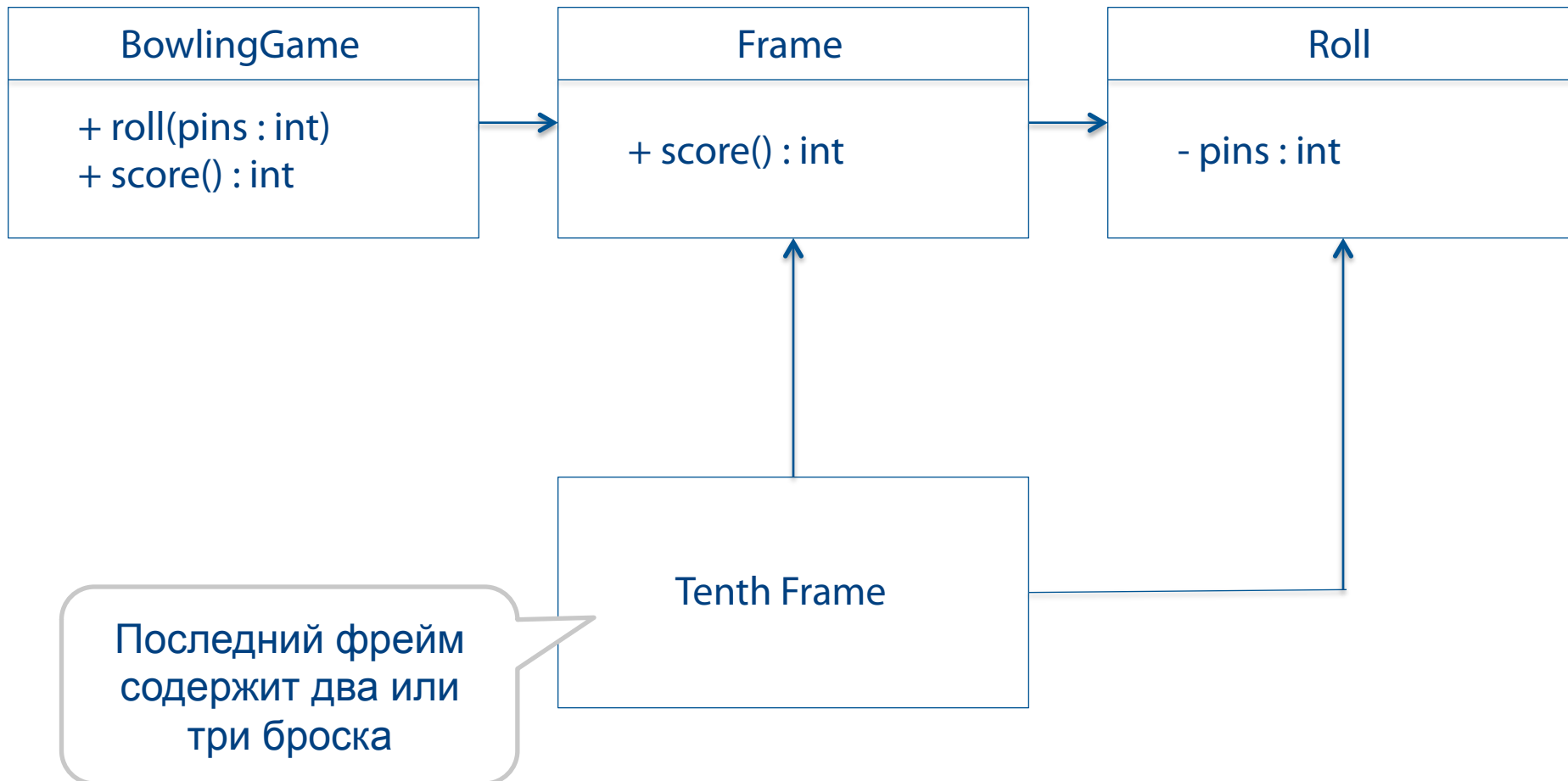


A quick design session

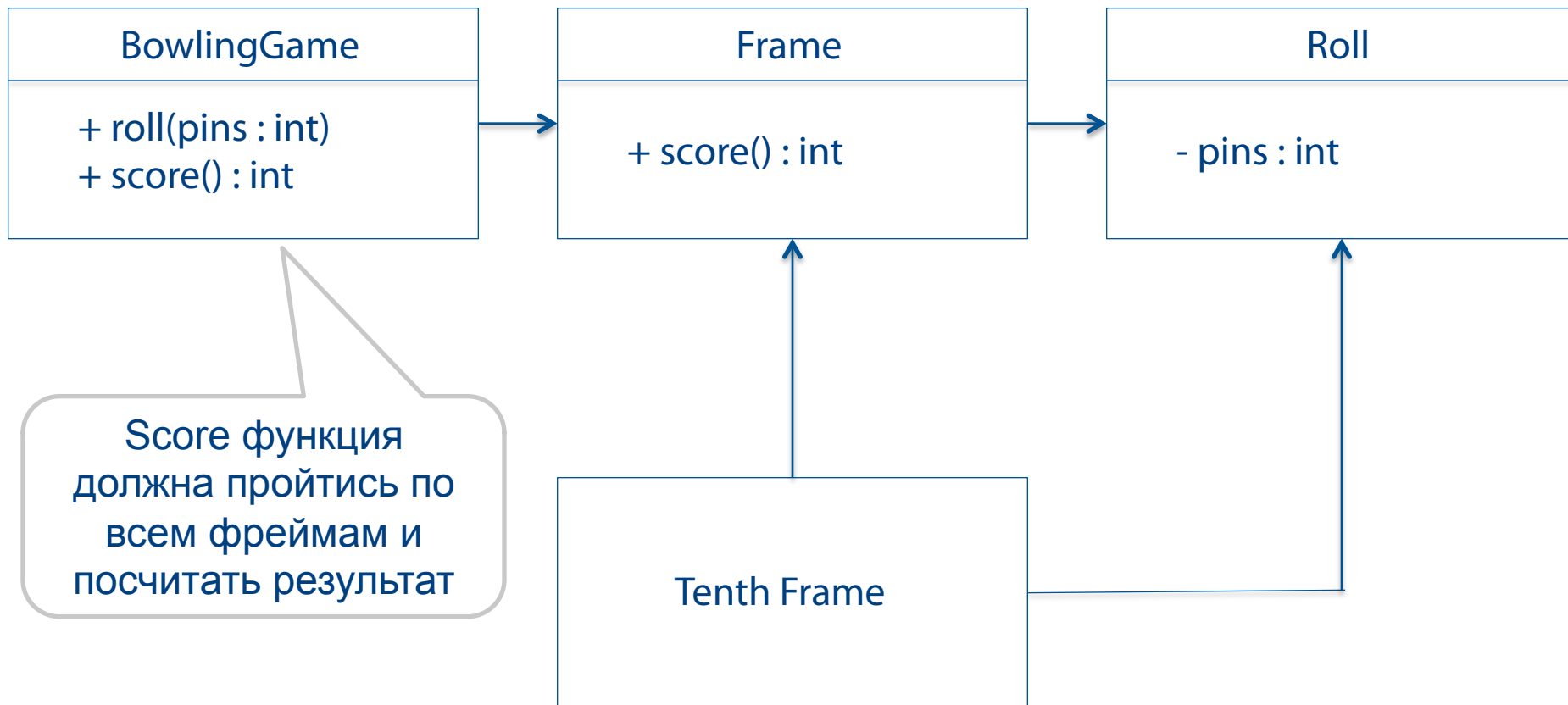


Каждый фрейм
состоит из одного
или двух бросков

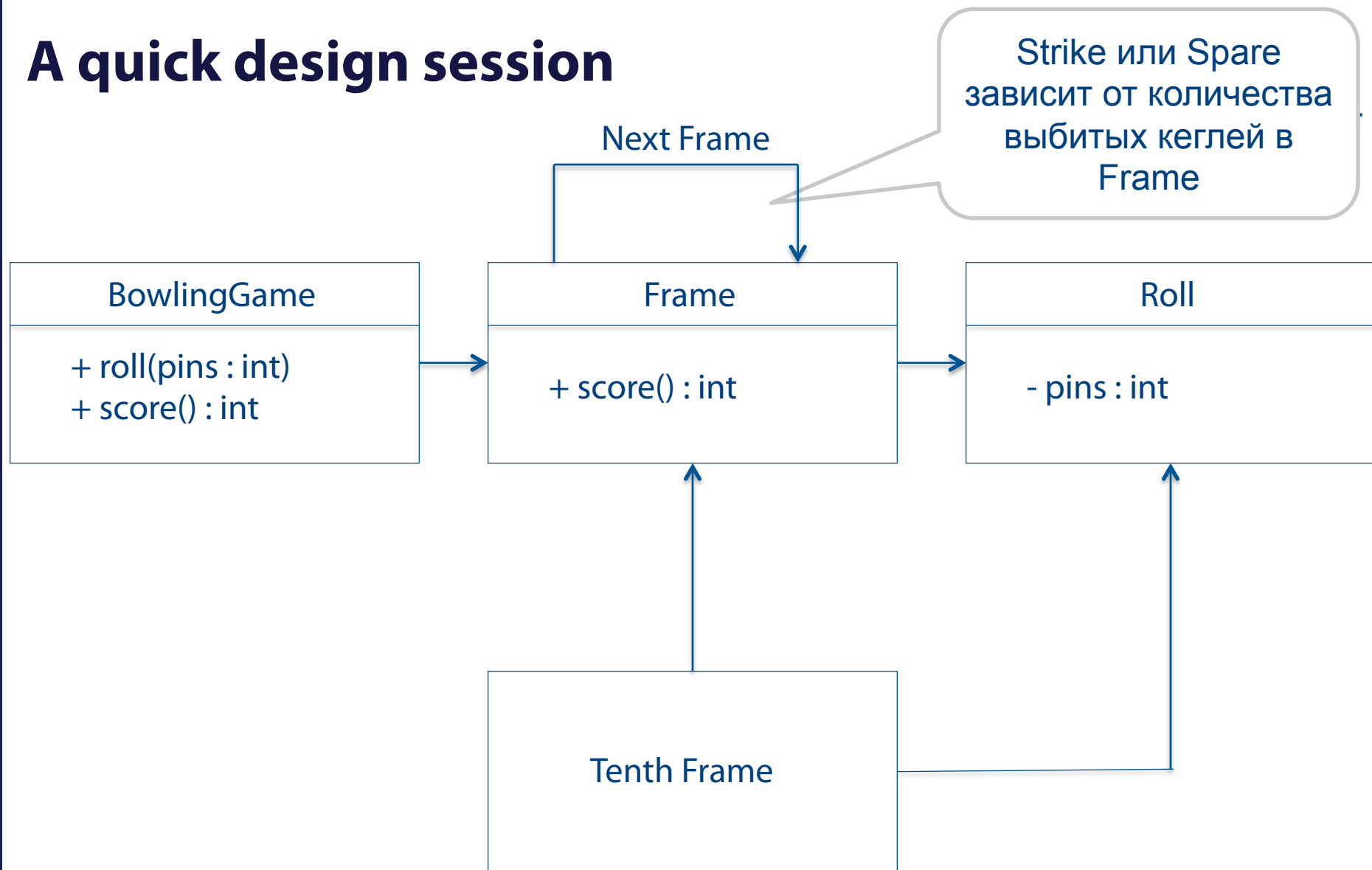
A quick design session



A quick design session



A quick design session



Готовим среду для тестирования



Еще раз повторим правила

Раунды

- 10 раундов (frame-ов)
- В одном frame – 2 броска
- Цель – выбить кегли (pins)
- Выигрывает набравший больше всех очков

Подсчет очков

- 10 сразу (strike). Заканчивается досрочно.
10 очков + pins + pins
- 10 очков со второго броска (spare)
10 очков + pins
- Max 300 очков (12 strikes)

Последний frame

- 1-й strike – дает возможность бросить еще два раза
- 2-й spare – дает возможность бросить еще один раз

Test List

```
public class BowlingGameTest {
```

Подумай

```
/**
```

```
 * should score zero on gutter game  
 * should score twenty when all one  
 * should score spare correctly  
 * should score all spares correctly  
 * should score strike correctly  
 * should score super game  
 */
```

```
}
```

git checkout bowling-0.1




Assert First

```
@Test
public void shouldScoreZeroOnGutterGame() {
    assertEquals(0, game.score());
}
```




Test First

```
@Test
public void shouldScoreZeroOnGutterGame() {
    for (int i = 0; i < 20; i++) {
        game.roll(0);
    }
    assertEquals(0, game.score());
}
```



Пишем тест
игнорируя
ошибки
компилятора



Исправляем ошибки, пишем код

```
@Test
public void shouldScoreZeroOnGutterGame() {
    for (int i = 0; i < 20; i++) {
        game.roll(0);
    }
    assertEquals(0, game.score());
}
```

```
public class BowlingGame {

    public void roll(int pins) {

    }

    public int score() {
        return 0;
    }
}
```

Пишем минимум
кода! KISS



Следующий тест

```
@Test
public void shouldScoreTwentyWhenAllOne() {
    for (int i = 0; i < 20; i++) {
        game.roll(1);
    }
    assertEquals(20, game.score());
}
```

Тест падает



Пишем код

```
@Test
public void shouldScoreTwentyWhenAllOne() {
    for (int i = 0; i < 20; i++) {
        game.roll(1);
    }
    assertEquals(20, game.score());
}
```

Тест проходит

```
.....

public class BowlingGame {

    private int score;

    public void roll(int pins) {
        score += pins;
    }

    public int score() {
        return score;
    }
}
```

[git checkout bowling-0.2](#)



Убираем дублирование

```
@Test
public void shouldScoreZeroOnGutterGame() {
    int count = 20;
    int pins = 0;
    for (int i = 0; i < count; i++) {
        game.roll(pins);
    }
    assertEquals(0, game.score());
}
```



Рефакторинг

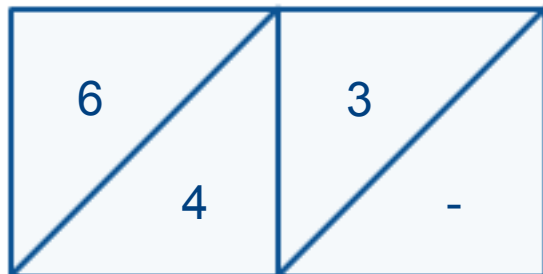
```
@Test
public void shouldScoreZeroOnGutterGame() {
    rollMany(20, 0);
    assertEquals(0, game.score());
}
```

```
@Test
public void shouldScoreTwentyWhenAllOne() {
    rollMany(20, 1);
    assertEquals(20, game.score());
}
```

```
private void rollMany(int count, int pins) {
    for (int i = 0; i < count; i++) {
        game.roll(pins);
    }
}
```



Проверка на Spare



10 + 3

3

Тест падает

@Test

```
public void shouldScoreSpareCorrectly() {  
    game.roll(6);  
    game.roll(4);  
    game.roll(3);  
    rollMany(17, 0);  
    assertEquals(16, game.score());  
}
```



Design review

```
public class BowlingGame {  
    private int score;  
    public void roll(int pins) {  
        score += pins;  
    }  
    public int score() {  
        return score;  
    }  
}
```

roll – занимается
подсчетом очков, но
имя метода не
указывает на это

score – не занимается подсчетом
очков, хотя имя указывает, что
должна



Redesign

Добавляем метод
в Ignore

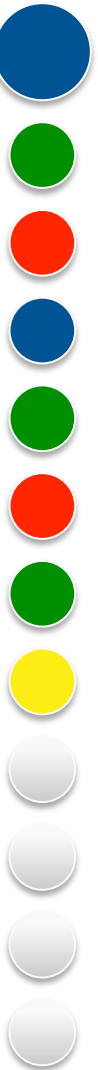


```
@Ignore("until we get design right")
@Test
public void shouldScoreSpareCorrectly() {
    game.roll(6);
    game.roll(4);
    game.roll(3);
    rollMany(17, 0);
    assertEquals(16, game.score());
}
```



Redesign

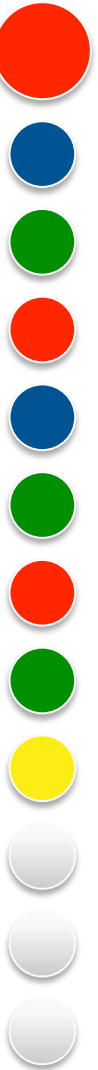
```
public class BowlingGame {  
  
    private final int[] rolls = new int[21];  
    private int currentRoll = 0;  
  
    public void roll(int pins) {  
        rolls[currentRoll++] = pins;  
    }  
  
    public int score() {  
        int score = 0;  
        for (int i = 0; i < rolls.length; i++) {  
            score += rolls[i];  
        }  
        return score;  
    }  
}
```



Убираем Ignore

```
@Test
public void shouldScoreSpareCorrectly() {
    game.roll(6);
    game.roll(4);
    game.roll(3);
    rollMany(17, 0);
    assertEquals(16, game.score());
}
```

Тест падает



Пишем код

```
public int score() {  
    int score = 0;  
    for (int i = 0; i < rolls.length; i++) {  
        if (rolls[i] + rolls[i+1] == 10) // spare  
            score += 10 + rolls[i+2];  
        score += rolls[i];  
    }  
    return score;  
}
```

Хорошая идея! За одним
исключением – это не
работает

У нас все еще проблемы с дизайном.
Очевидно, надо считать очки по фреймам



Очередная сессия Redesign

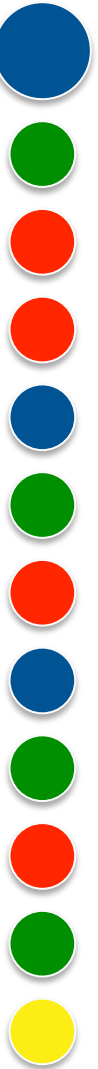
```
@Ignore("until we get design right")  
@Test  
public void shouldScoreSpareCorrectly() {  
    game.roll(6);  
    game.roll(4);  
    game.roll(3);  
    rollMany(17, 0);  
    assertEquals(16, game.score());  
}
```

Добавляем метод
в Ignore



Refactoring

```
public int score() {  
    int score = 0;  
    int i = 0;  
    for (int frames = 0; frames < 10; frames++) {  
        score += rolls[i] + rolls[i+1];  
        i += 2;  
    }  
    return score;  
}
```



Убираем Ignore

@Test

```
public void shouldScoreSpareCorrectly() {  
    game.roll(6);  
    game.roll(4);  
    game.roll(3);  
    rollMany(17, 0);  
    assertEquals(16, game.score());  
}
```

Тест падает



Пишем код

```
public int score() {  
    int score = 0;  
    int i = 0;  
    for (int frames = 0; frames < 10; frames++) {  
        if (rolls[i] + rolls[i + 1] == 10) { // spare  
            score += 10 + rolls[i + 2];  
            i += 2;  
        } else {  
            score += rolls[i] + rolls[i + 1];  
            i += 2;  
        }  
    }  
    return score;  
}
```



Design review

Плохое имя переменной

```
public int score() {  
    int score = 0;  
    int i = 0;  
    for (int frames = 0; frames < 10; frames++) {  
        if (rolls[i] + rolls[i + 1] == 10) { // spare  
            score += 10 + rolls[i + 2];  
            i += 2;  
        } else {  
            score += rolls[i] + rolls[i + 1];  
            i += 2;  
        }  
    }  
    return score;  
}
```

Плохо иметь комментарий в
условии



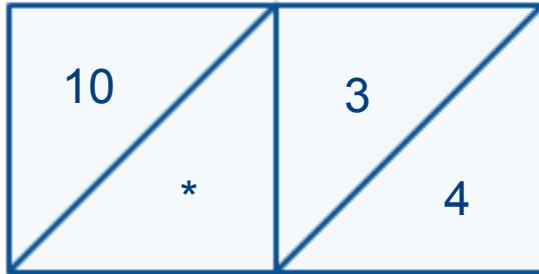
Refactoring

```
public int score() {  
    int score = 0;  
    int frameIndex = 0;  
    for (int frames = 0; frames < 10; frames++) {  
        if (isSpare(frameIndex)) {  
            score += 10 + rolls[frameIndex + 2];  
            frameIndex += 2;  
        } else {  
            score += rolls[frameIndex] + rolls[frameIndex + 1];  
            frameIndex += 2;  
        }  
    }  
    return score;  
}
```

```
private boolean isSpare(int frameIndex) {  
    return rolls[frameIndex] + rolls[frameIndex + 1] == 10;  
}
```



Проверка на Strike



$10 + 3 + 4$

$3 + 4$

@Test

```
public void shouldScoreStrikeCorrectly()
{
```

```
    game.roll(10); // strike
```

```
    game.roll(3);
```

```
    game.roll(4);
```

```
    rollMany(16, 0);
```

```
    assertEquals(24, game.score());
```

```
}
```

Комментарий в тесте



Пишем код

```
public int score() {  
    int score = 0;  
    int frameIndex = 0;  
    for (int frames = 0; frames < 10; frames++) {  
        if (rolls[frameIndex] == 10) { // strike  
            score += 10 +  
                rolls[frameIndex + 1] +  
                rolls[frameIndex + 2];  
            frameIndex++;  
        } else if (isSpare(frameIndex)) {  
            score += 10 +  
                rolls[frameIndex + 2];  
            frameIndex += 2;  
        } else {  
            score += rolls[frameIndex] +  
                rolls[frameIndex + 1];  
            frameIndex += 2;  
        }  
    }  
    return score;  
}
```



Design review

```
public int score() {  
    int score = 0;  
    int frameIndex = 0;  
    for (int frames = 0; frames < 10; frames++) {  
        if (rolls[frameIndex] == 10) { // strike  
            score += 10 +  
                rolls[frameIndex + 1] +  
                rolls[frameIndex + 2];  
            frameIndex++;  
        } else if (isSpare(frameIndex)) {  
            score += 10 +  
                rolls[frameIndex + 2];  
            frameIndex += 2;  
        } else {  
            score += rolls[frameIndex] +  
                rolls[frameIndex + 1];  
            frameIndex += 2;  
        }  
    }  
    return score;  
}
```

Комментарий для условия

Непонятное
выражение



Refactoring

```
public int score() {  
    int score = 0;  
    int frameIndex = 0;  
    for (int frames = 0; frames < 10; frames++) {  
        if (isStrike(frameIndex)) {  
            score += strikeBonus(frameIndex);  
            frameIndex++;  
        } else if (isSpare(frameIndex)) {  
            score += spireBonus(frameIndex);  
            frameIndex += 2;  
        } else {  
            score += sumOfBallsInFrame(frameIndex);  
            frameIndex += 2;  
        }  
    }  
    return score;  
}
```



Refactoring

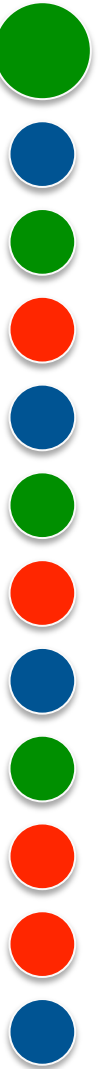
```
@Test
public void shouldScoreStrikeCorrectly() {
    rollStrike();
    game.roll(3);
    game.roll(4);
    rollMany(16, 0);
    assertEquals(24, game.score());
}
```

```
private void rollStrike() {
    game.roll(10);
}
```



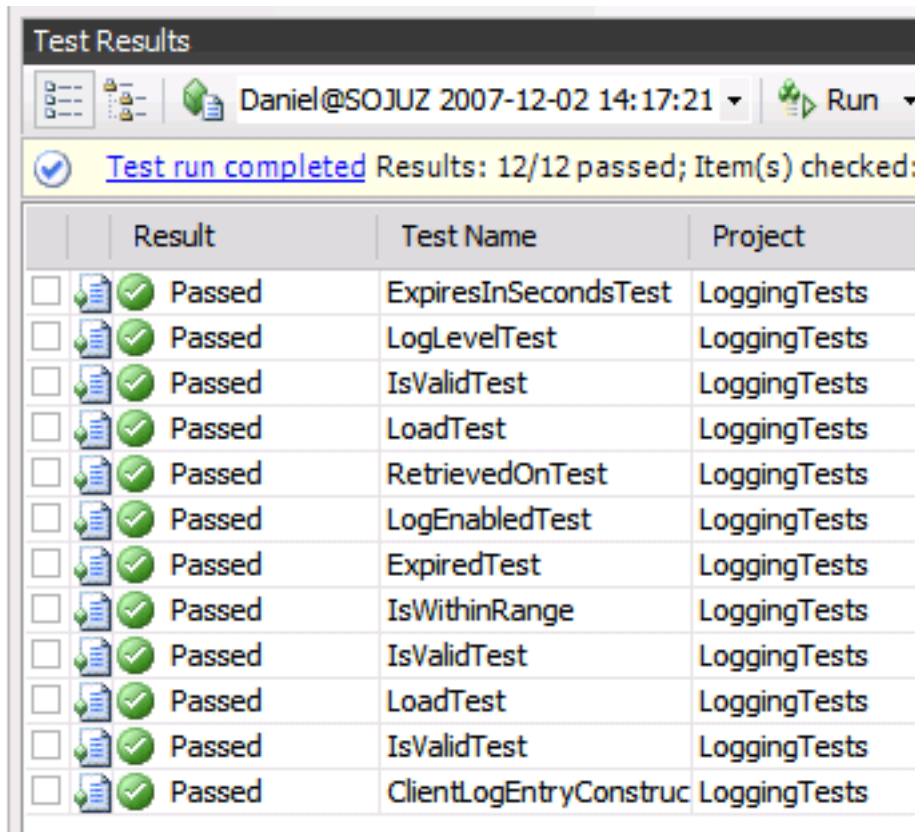
Контрольный пример

```
@Test
public void shouldScorePerfectGame() {
    rollMany(12, 10);
    assertEquals(300, game.score());
}
```



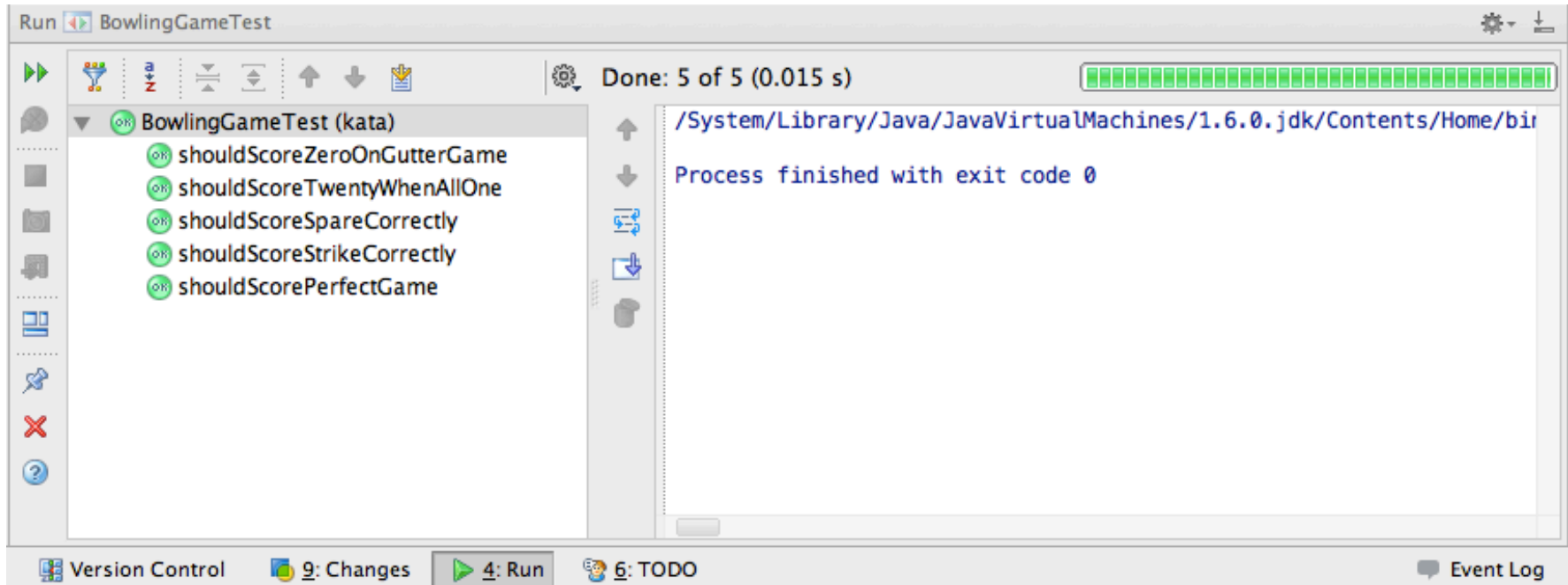
Стратегии запуска тестов

- IDE
- Console
- Ant
- Maven
- Test Runner
- CI
- File watcher



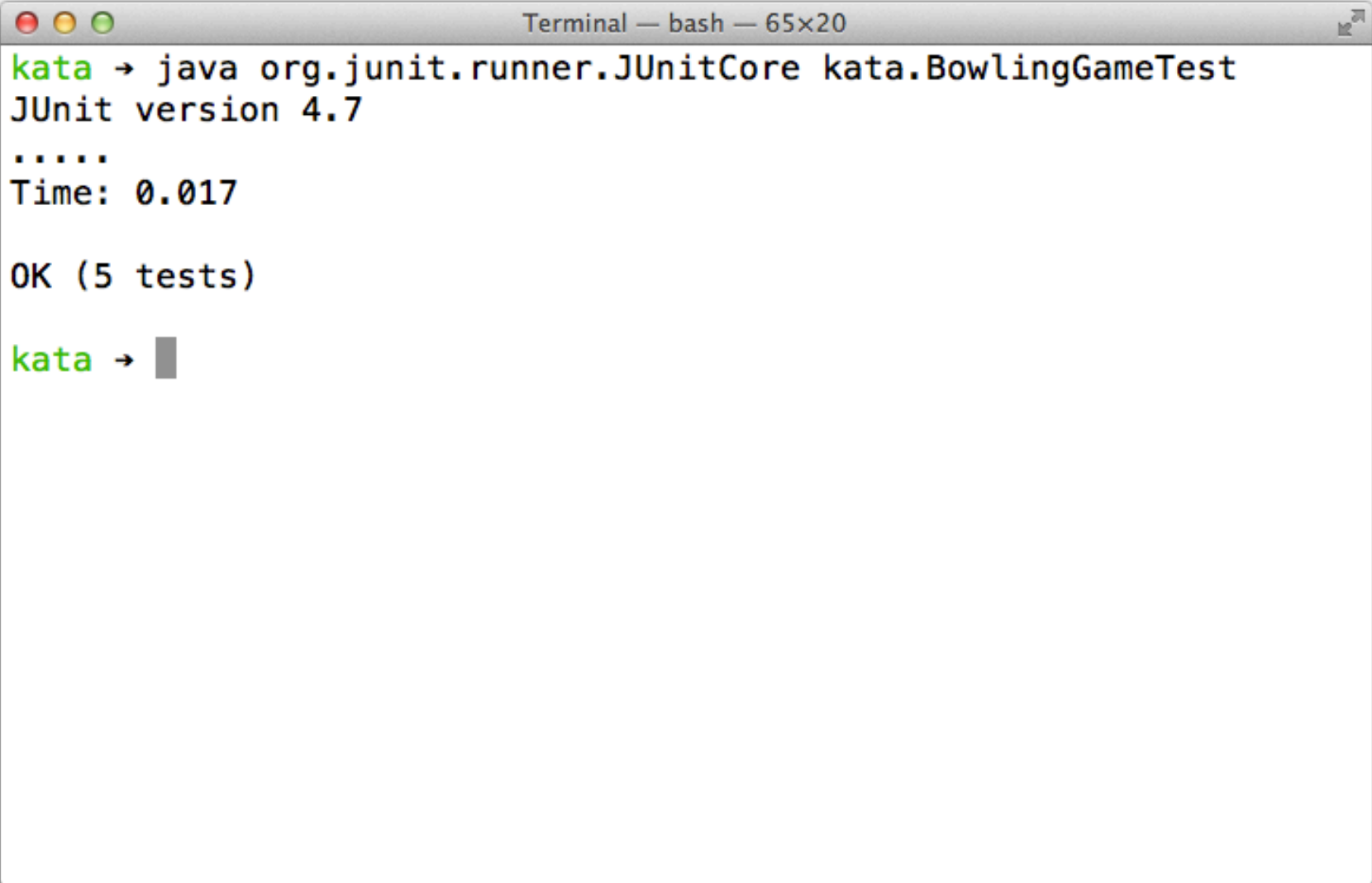
The screenshot shows a 'Test Results' window with a toolbar containing icons for a list, a folder, a document, a user profile, a date/time string 'Daniel@SOJUZ 2007-12-02 14:17:21', and a 'Run' button. Below the toolbar, a status bar indicates 'Test run completed' with a blue checkmark icon, followed by the text 'Results: 12/12 passed; Item(s) checked:'. The main area is a table with four columns: 'Result', 'Test Name', and 'Project'. Each row represents a test that has passed, indicated by a green checkmark icon in the 'Result' column. The 'Test Name' and 'Project' columns list the specific tests and the project they belong to.

	Result	Test Name	Project
<input type="checkbox"/>	Passed	ExpiresInSecondsTest	LoggingTests
<input type="checkbox"/>	Passed	LogLevelTest	LoggingTests
<input type="checkbox"/>	Passed	IsValidTest	LoggingTests
<input type="checkbox"/>	Passed	LoadTest	LoggingTests
<input type="checkbox"/>	Passed	RetrievedOnTest	LoggingTests
<input type="checkbox"/>	Passed	LogEnabledTest	LoggingTests
<input type="checkbox"/>	Passed	ExpiredTest	LoggingTests
<input type="checkbox"/>	Passed	IsWithinRange	LoggingTests
<input type="checkbox"/>	Passed	IsValidTest	LoggingTests
<input type="checkbox"/>	Passed	LoadTest	LoggingTests
<input type="checkbox"/>	Passed	IsValidTest	LoggingTests
<input type="checkbox"/>	Passed	ClientLogEntryConstruc	LoggingTests



- Сделать шаблон для быстрого создания тестов
- Добавить Short-Cut для быстрого запуска тестов

Console

A screenshot of a macOS Terminal window. The title bar reads "Terminal — bash — 65x20". The terminal content shows a command to run JUnit tests, followed by the JUnit version, a separator of five dots, the execution time, the result of the tests, and a new prompt line.

```
Terminal — bash — 65x20
kata → java org.junit.runner.JUnitCore kata.BowlingGameTest
JUnit version 4.7
.....
Time: 0.017

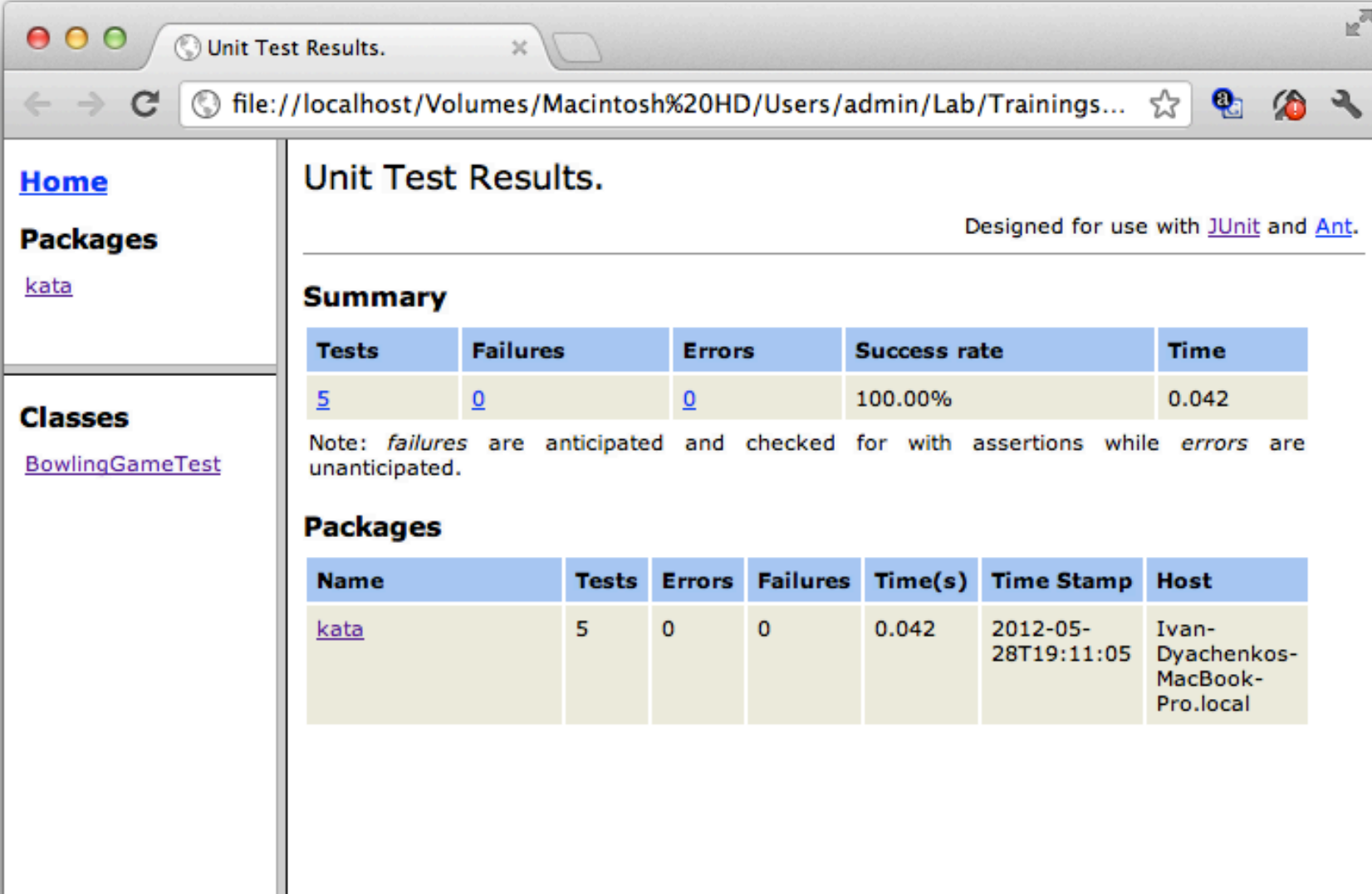
OK (5 tests)

kata →
```

Ant



- ant
- ant coverage



The screenshot shows a web browser window titled "Unit Test Results." with a file path in the address bar: `file:///localhost/Volumes/Macintosh%20HD/Users/admin/Lab/Trainings...`. The browser interface includes standard navigation buttons and a sidebar on the left with links for "Home", "Packages" (containing a link to "kata"), and "Classes" (containing a link to "BowlingGameTest").

The main content area displays the "Unit Test Results." page, which is designed for use with JUnit and Ant. It features a "Summary" section with a table showing the overall test results:

Tests	Failures	Errors	Success rate	Time
5	0	0	100.00%	0.042

Below the summary table, a note states: "Note: *failures* are anticipated and checked for with assertions while *errors* are unanticipated."

The "Packages" section contains a table detailing the results for the "kata" package:

Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
kata	5	0	0	0.042	2012-05-28T19:11:05	Ivan-Dyachenkos-MacBook-Pro.local

maven

- mvn clean install
- mvn clean install -Pcoverage

© Luxoft Training 2012

67

Continuous integration



Hudson



Team City

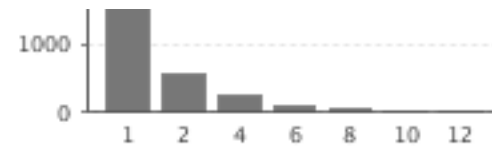


Continuous integration



.PI
22 lines
2 blocks
2 files

10.7 / class
5,033 cmpx ▲
8,640 statements ▲



ations

6 ▲

locker 0

ritical 0

lajor 353 ▲ 

linor 2

fo 231 ▲ 

Code coverage

66.8% ▲

919 tests ▲

+2 skipped ▼

2:28 min ▼

Test success

99.9%

0 failures

1 errors

⚠ Alerts : Coverage < 70.

Tags



File watcher

Guard build status passing

Преимущества от TDD

Меньше ошибок

Проще рефакторить

Документация

Лучший дизайн кода

Быстрее процесс разработки

Вопросы



Цикл TDD ?

Вопросы

Цикл TDD ?

Какие преимущества ?

Цикл TDD ?

Какие преимущества ?

Ошибки TDD ?



Вопросы ?



Разработка через тестирование

IDyachenko@luxoft.com

```
git clone git://github.com/ivan-dyachenko/Trainings.git
```

```
https://github.com/ivan-dyachenko/Trainings
```