# Unit Testing and TDD Additional Material SFWR ENG 3S03: Software Testing

Alicia Marinache

Department of Computing and Software, McMaster University
Canada L8S 4L7, Hamilton, Ontario

# UT and TDD Additional Material

## ➡Individual Reading

Take a few minutes to read the following articles (estimated 10-15 minutes per article)

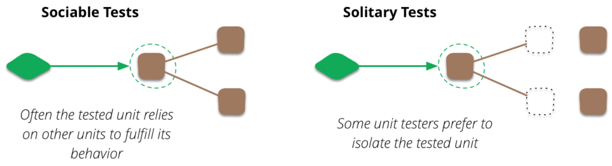
- `https://martinfowler.com/bliki/UnitTest.html`
- `https://martinfowler.com/bliki/IntegrationTest.html`

What are **your** key takeaways?

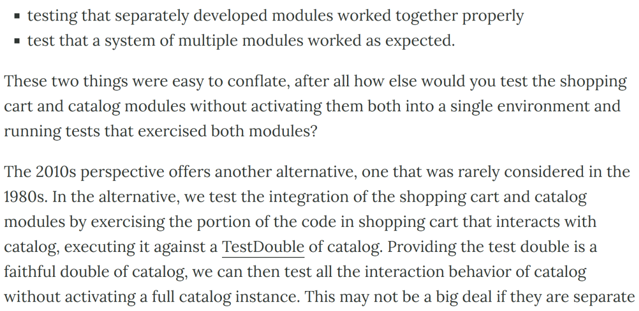
## Martin Fowler: Unit Test (May 5, 2014)

### Solitary or Sociable?

A more important distinction is whether the unit you're testing should be sociable or solitary [3]. Imagine you're testing an order class's price method. The price method needs to invoke some functions on the product and customer classes. If you like your unit tests to be solitary, you don't want to use the real product or customer classes here, because a fault in the customer class would cause the order class's tests to fail. Instead you use TestDoubles for the collaborators.


Sociable Tests
Often the tested unit relies on other units to fulfill its behavior
Solitary Tests
Some unit testers prefer to isolate the tested unit

## Martin Fowler: Integration Testing (Jan 16, 2018)

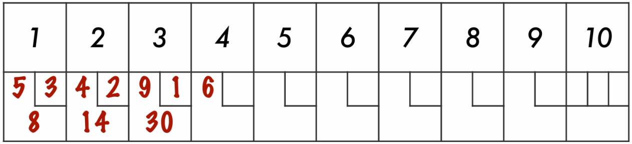


- testing that separately developed modules worked together properly
- test that a system of multiple modules worked as expected.

These two things were easy to conflate, after all how else would you test the shopping cart and catalog modules without activating them both into a single environment and running tests that exercised both modules?

The 2010s perspective offers another alternative, one that was rarely considered in the 1980s. In the alternative, we test the integration of the shopping cart and catalog modules by exercising the portion of the code in shopping cart that interacts with catalog, executing it against a TestDouble of catalog. Providing the test double is a faithful double of catalog, we can then test all the interaction behavior of catalog without activating a full catalog instance. This may not be a big deal if they are separate

## TDD Example - Bowling

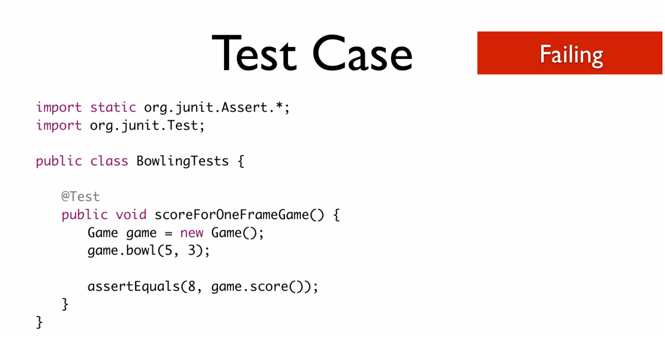


- Ten frames in each game.
- Two balls thrown in (most) frames.
- Bonus points for spares and strikes.

Example based on Agile Principles, Patterns and Practices In C#, Martin & Martin, 2006.

# UT and TDD Additional Material

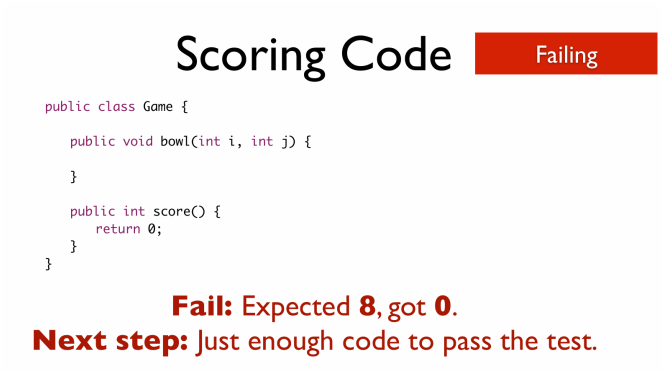
## ➡UT and TDD Example



Test Case

Failing

```
import static org.junit.Assert.*;
import org.junit.Test;

public class BowlingTests {

    @Test
    public void scoreForOneFrameGame() {
        Game game = new Game();
        game.bowl(5, 3);

        assertEquals(8, game.score());
    }
}
```



## Scoring Code

Failing

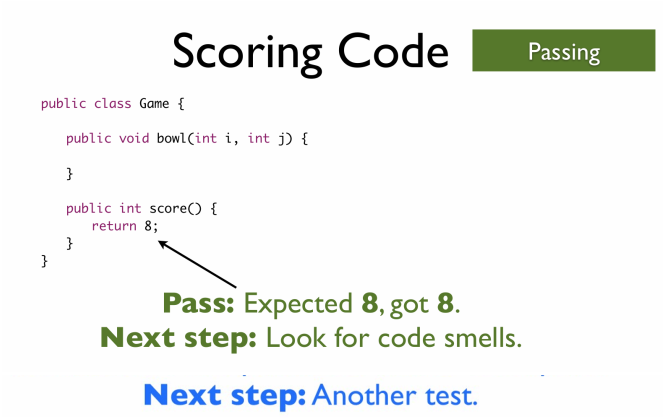
```
public class Game {

    public void bowl(int i, int j) {

    }

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

**Fail:** Expected **8**, got **0**.

**Next step:** Just enough code to pass the test.

# UT and TDD Additional Material

## ➥UT and TDD Example



### Scoring Code

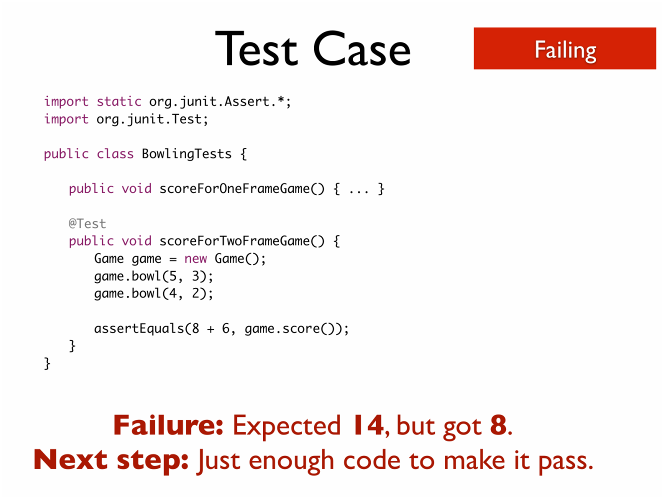
Passing

```
public class Game {

    public void bowl(int i, int j) {

    }

    public int score() {
        return 8;
    }
}
```

**Pass:** Expected **8**, got **8**.

**Next step:** Look for code smells.

**Next step:** Another test.

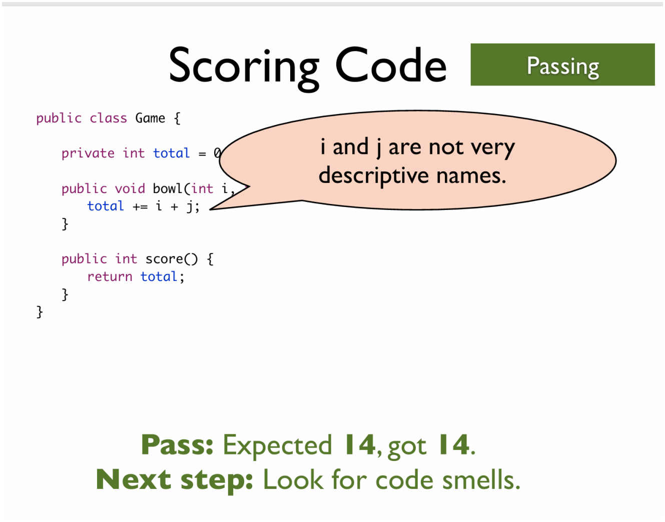


## Test Case

Failing

```
import static org.junit.Assert.*;
import org.junit.Test;

public class BowlingTests {

    public void scoreForOneFrameGame() { ... }

    @Test
    public void scoreForTwoFrameGame() {
        Game game = new Game();
        game.bowl(5, 3);
        game.bowl(4, 2);

        assertEquals(8 + 6, game.score());
    }
}
```

**Failure:** Expected **14**, but got **8**.
**Next step:** Just enough code to make it pass.



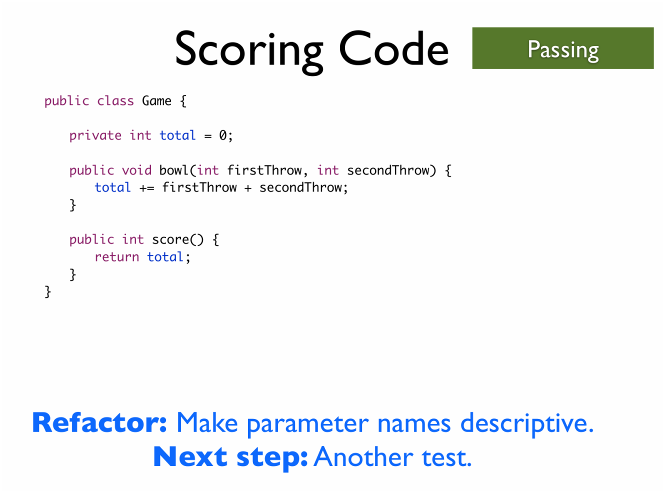
## Scoring Code

Passing

```
public class Game {

    private int total = 0

    public void bowl(int i,
        total += i + j;
    }

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

i and j are not very descriptive names.

**Pass:** Expected **14**, got **14**.
**Next step:** Look for code smells.



## Scoring Code

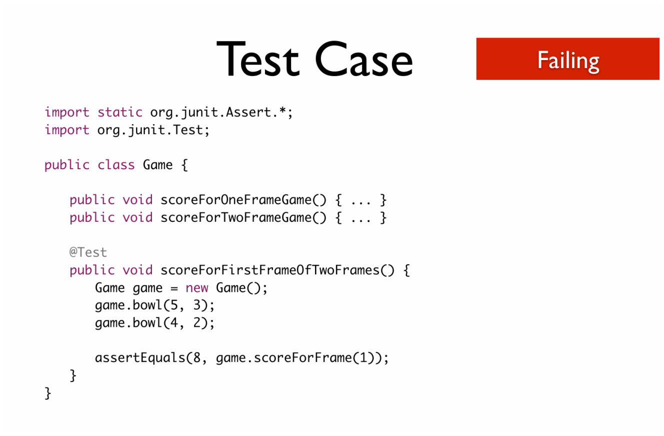
Passing

```
public class Game {

    private int total = 0;

    public void bowl(int firstThrow, int secondThrow) {
        total += firstThrow + secondThrow;
    }

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

**Refactor:** Make parameter names descriptive.
**Next step:** Another test.

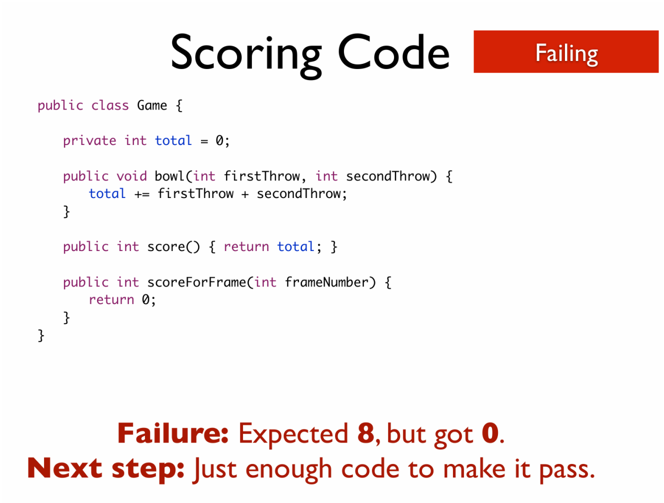
## UT and TDD Additional Material

### UT and TDD Example



**Test Case** Failing

```
import static org.junit.Assert.*;
import org.junit.Test;

public class Game {

    public void scoreForOneFrameGame() { ... }
    public void scoreForTwoFrameGame() { ... }

    @Test
    public void scoreForFirstFrameOfTwoFrames() {
        Game game = new Game();
        game.bowl(5, 3);
        game.bowl(4, 2);

        assertEquals(8, game.scoreForFrame(1));
    }
}
```



## Scoring Code

Failing

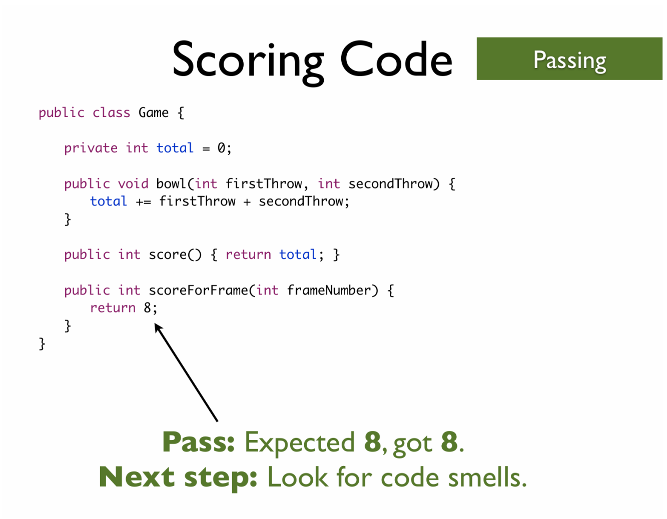
```
public class Game {

    private int total = 0;

    public void bowl(int firstThrow, int secondThrow) {
        total += firstThrow + secondThrow;
    }

    public int score() { return total; }

    public int scoreForFrame(int frameNumber) {
        return 0;
    }
}
```

**Failure:** Expected **8**, but got **0**.

**Next step:** Just enough code to make it pass.

# UT and TDD Additional Material

## ➥UT and TDD Example



### Scoring Code

Passing

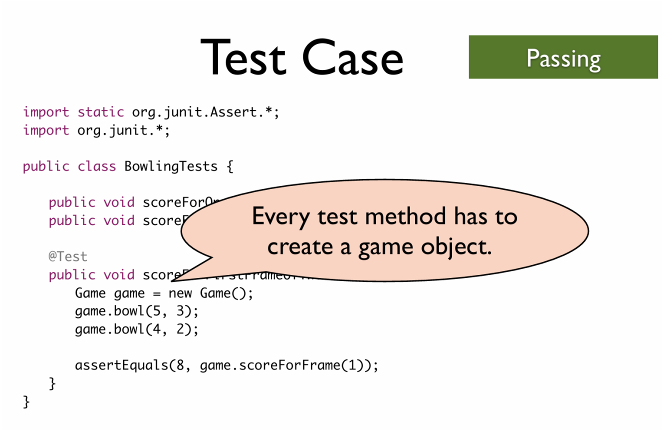
```
public class Game {

    private int total = 0;

    public void bowl(int firstThrow, int secondThrow) {
        total += firstThrow + secondThrow;
    }

    public int score() { return total; }

    public int scoreForFrame(int frameNumber) {
        return 8;
    }
}
```

**Pass:** Expected **8**, got **8**.

**Next step:** Look for code smells.

# UT and TDD Additional Material

## UT and TDD Example



Test Case

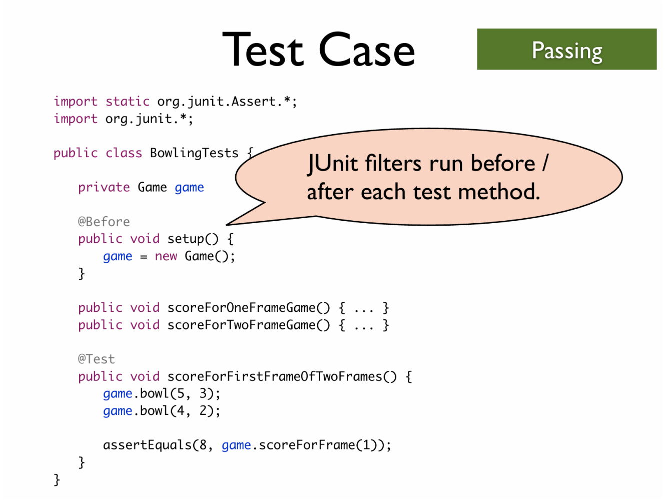
Passing

```
import static org.junit.Assert.*;
import org.junit.*;

public class BowlingTests {

    public void scoreForO
    public void scoreF

    @Test
    public void scoreF       irstFrameOf...
        Game game = new Game();
        game.bowl(5, 3);
        game.bowl(4, 2);

        assertEquals(8, game.scoreForFrame(1));
    }
}
```

Every test method has to create a game object.

# UT and TDD Additional Material

## UT and TDD Example

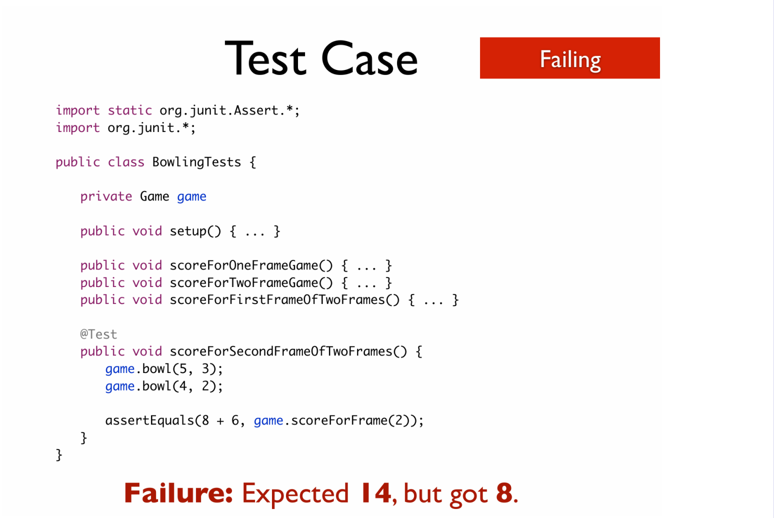


Test Case

Passing

```
import static org.junit.Assert.*;
import org.junit.*;

public class BowlingTests {

    private Game game

    @Before
    public void setup() {
        game = new Game();
    }

    public void scoreForOneFrameGame() { ... }
    public void scoreForTwoFrameGame() { ... }

    @Test
    public void scoreForFirstFrameOfTwoFrames() {
        game.bowl(5, 3);
        game.bowl(4, 2);

        assertEquals(8, game.scoreForFrame(1));
    }
}
```

JUnit filters run before / after each test method.



## Test Case

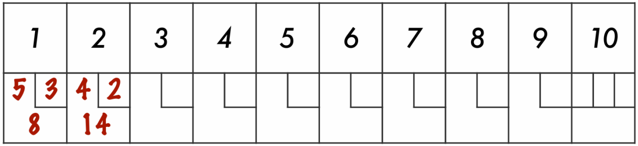
Failing

```
import static org.junit.Assert.*;
import org.junit.*;

public class BowlingTests {

    private Game game

    public void setup() { ... }

    public void scoreForOneFrameGame() { ... }
    public void scoreForTwoFrameGame() { ... }
    public void scoreForFirstFrameOfTwoFrames() { ... }

    @Test
    public void scoreForSecondFrameOfTwoFrames() {
        game.bowl(5, 3);
        game.bowl(4, 2);

        assertEquals(8 + 6, game.scoreForFrame(2));
    }
}
```

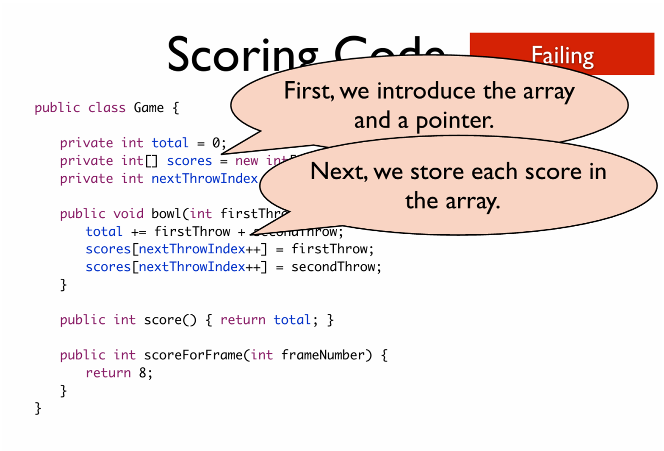
**Failure:** Expected **14**, but got **8**.

## Scoring Code

- Storing the total isn't enough any more.
- We need a quick design session.



- How about an array to store the scores?



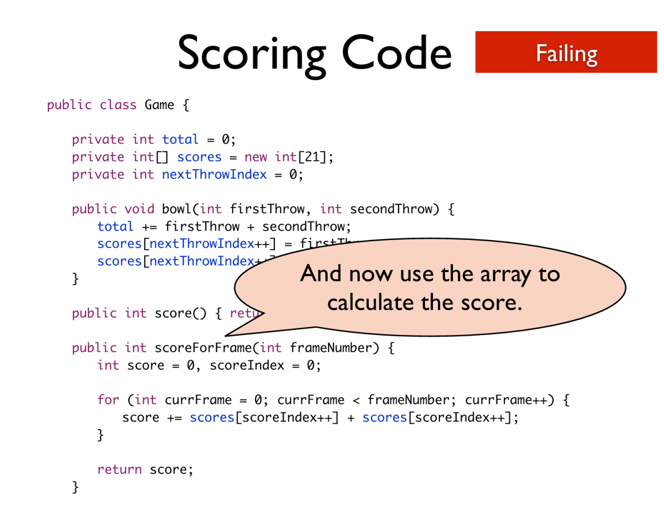
## Scoring Code

Failing

First, we introduce the array and a pointer.

Next, we store each score in the array.

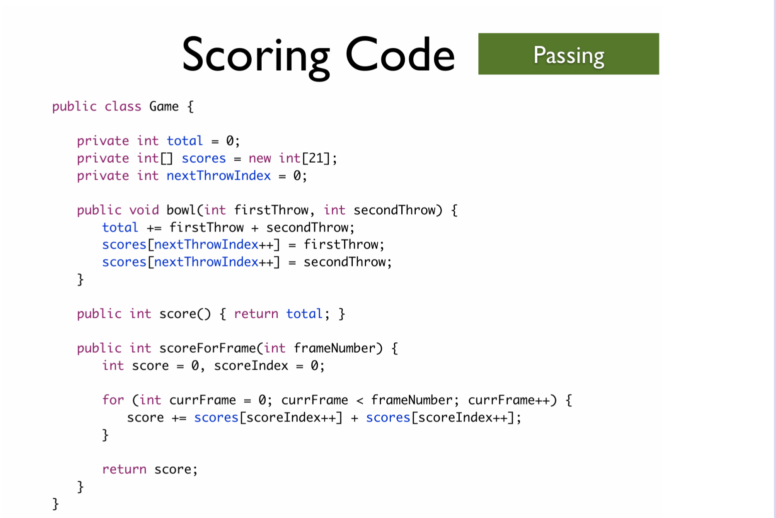
```
public class Game {

    private int total = 0;
    private int[] scores = new int
    private int nextThrowIndex

    public void bowl(int firstThr
        total += firstThrow + secondThrow;
        scores[nextThrowIndex++] = firstThrow;
        scores[nextThrowIndex++] = secondThrow;
    }

    public int score() { return total; }

    public int scoreForFrame(int frameNumber) {
        return 8;
    }
}
```



## Scoring Code

Failing

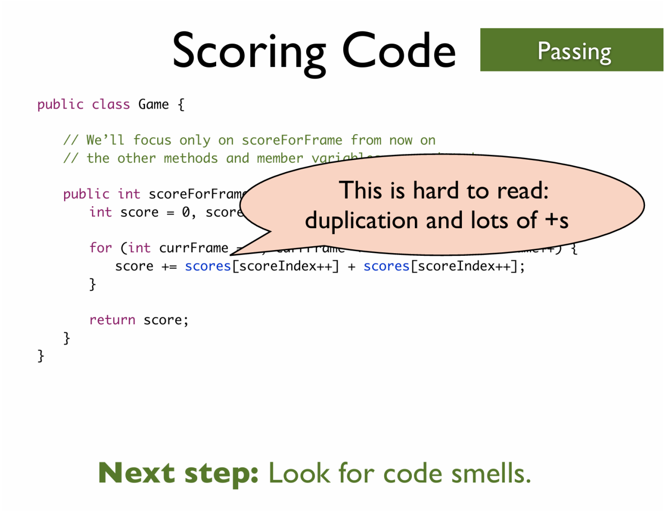
```
public class Game {

    private int total = 0;
    private int[] scores = new int[21];
    private int nextThrowIndex = 0;

    public void bowl(int firstThrow, int secondThrow) {
        total += firstThrow + secondThrow;
        scores[nextThrowIndex++] = firstTh
        scores[nextThrowIndex++
    }

    public int score() { retu

    public int scoreForFrame(int frameNumber) {
        int score = 0, scoreIndex = 0;

        for (int currFrame = 0; currFrame < frameNumber; currFrame++) {
            score += scores[scoreIndex++] + scores[scoreIndex++];
        }

        return score;
    }
```



## Scoring Code

Passing

```
public class Game {

    private int total = 0;
    private int[] scores = new int[21];
    private int nextThrowIndex = 0;

    public void bowl(int firstThrow, int secondThrow) {
        total += firstThrow + secondThrow;
        scores[nextThrowIndex++] = firstThrow;
        scores[nextThrowIndex++] = secondThrow;
    }

    public int score() { return total; }

    public int scoreForFrame(int frameNumber) {
        int score = 0, scoreIndex = 0;

        for (int currFrame = 0; currFrame < frameNumber; currFrame++) {
            score += scores[scoreIndex++] + scores[scoreIndex++];
        }

        return score;
    }
}
```



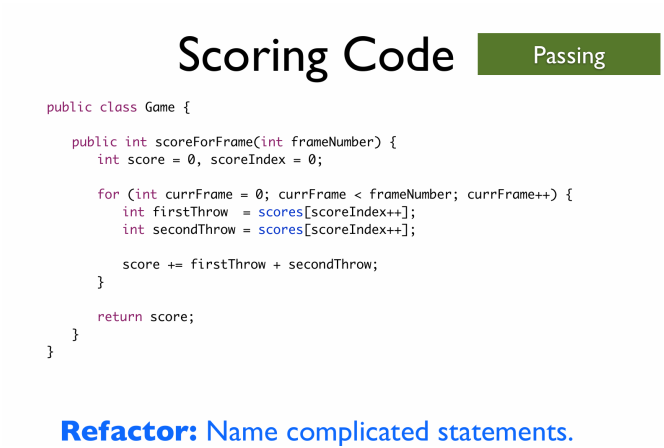
## Scoring Code

Passing

```
public class Game {

    // We'll focus only on scoreForFrame from now on
    // the other methods and member variabl

    public int scoreForFram
        int score = 0, score

        for (int currFrame                                          ame++) {
            score += scores[scoreIndex++] + scores[scoreIndex++];
        }

        return score;
    }
}
```

This is hard to read: duplication and lots of +s

**Next step:** Look for code smells.



## Scoring Code

Passing

```
public class Game {

    public int scoreForFrame(int frameNumber) {
        int score = 0, scoreIndex = 0;

        for (int currFrame = 0; currFrame < frameNumber; currFrame++) {
            int firstThrow  = scores[scoreIndex++];
            int secondThrow = scores[scoreIndex++];

            score += firstThrow + secondThrow;
        }

        return score;
    }
}
```

**Refactor:** Name complicated statements.