@drpicox 2020

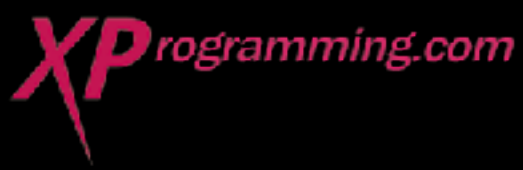
Java

Bowling Game Kata

Object Mentor, Inc.

www.objectmentor.com
blog.objectmentor.com

fitnesse.org



www.junit.org

Origin

» Robert C. Martin created this Kata in 2005

» You can see how he solves it at:

https://cleancoders.com/video-details/clean-code-episode-6-p2

» The Objective is create «*Muscle Memory*»

Repeat frequently until be natural following the steps in all your developments

Plan

1. Analysing bowling score algotithm.
   (Robert C Martin original slides)

2. What is TDD?

3. The Kata.

Scoring Bowling.

1 4	4 5	6 ◢	5 ◢	■	0 1	7 ◢	6 ◢	■	2 ◢ 6
5	14	29	49	60	61	77	97	117	133

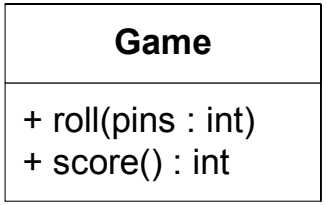
The game consists of 10 frames as shown above. In each frame the player has two opportunities to knock down 10 pins. The score for the frame is the total number of pins knocked down, plus bonuses for strikes and spares.

A spare is when the player knocks down all 10 pins in two tries. The bonus for that frame is the number of pins knocked down by the next roll. So in frame 3 above, the score is 10 (the total number knocked down) plus a bonus of 5 (the number of pins knocked down on the next roll.)

A strike is when the player knocks down all 10 pins on his first try. The bonus for that frame is the value of the next two balls rolled.

In the tenth frame a player who rolls a spare or strike is allowed to roll the extra balls to complete the frame. However no more than three balls can be rolled in tenth frame.

The Requirements.



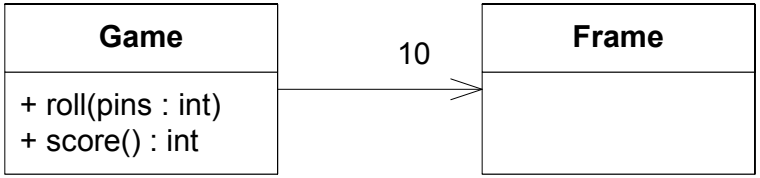
- Write a class named “Game” that has two methods
  - roll(pins : int) is called each time the player rolls a ball. The argument is the number of pins knocked down.
  - score() : int is called only at the very end of the game. It returns the total score for that game.

A quick design session

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

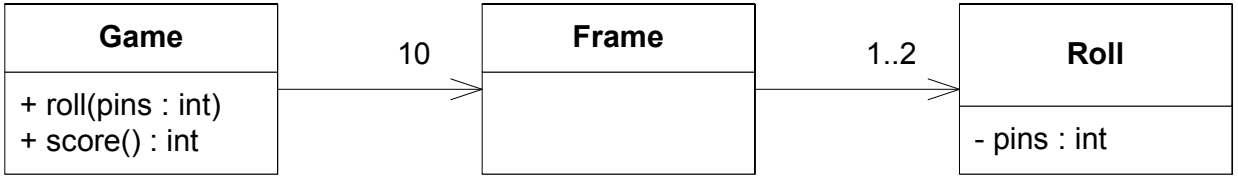
Clearly we need the Game class.

A quick design session



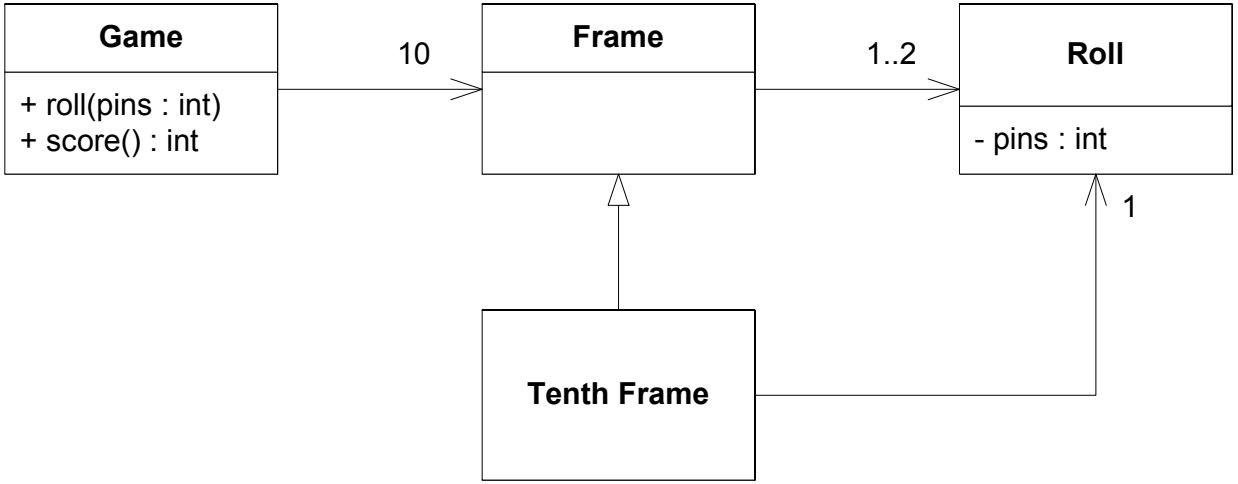
A game has 10 frames.

A quick design session



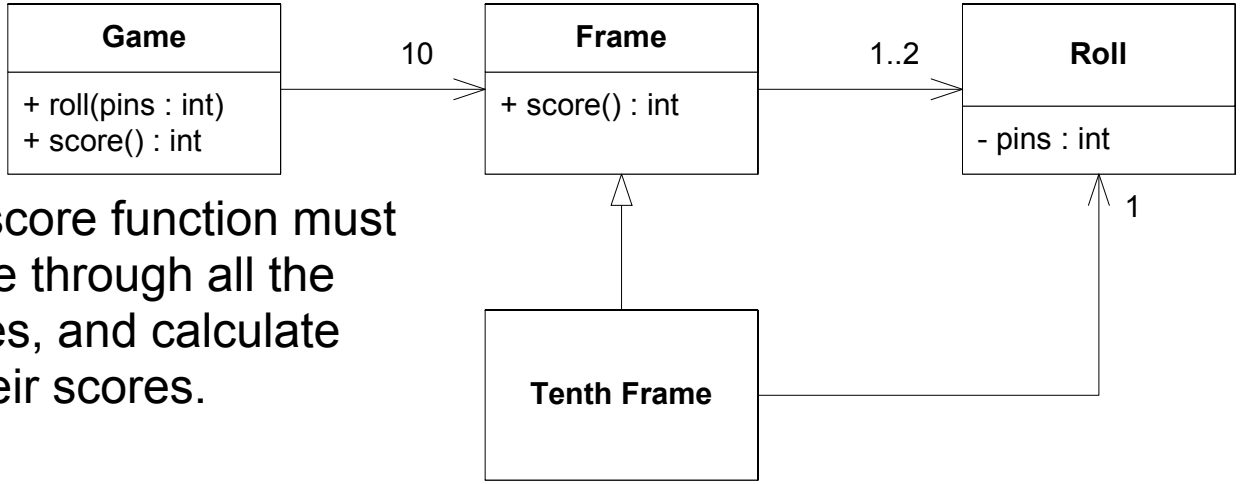
A frame has 1 or two rolls.

A quick design session



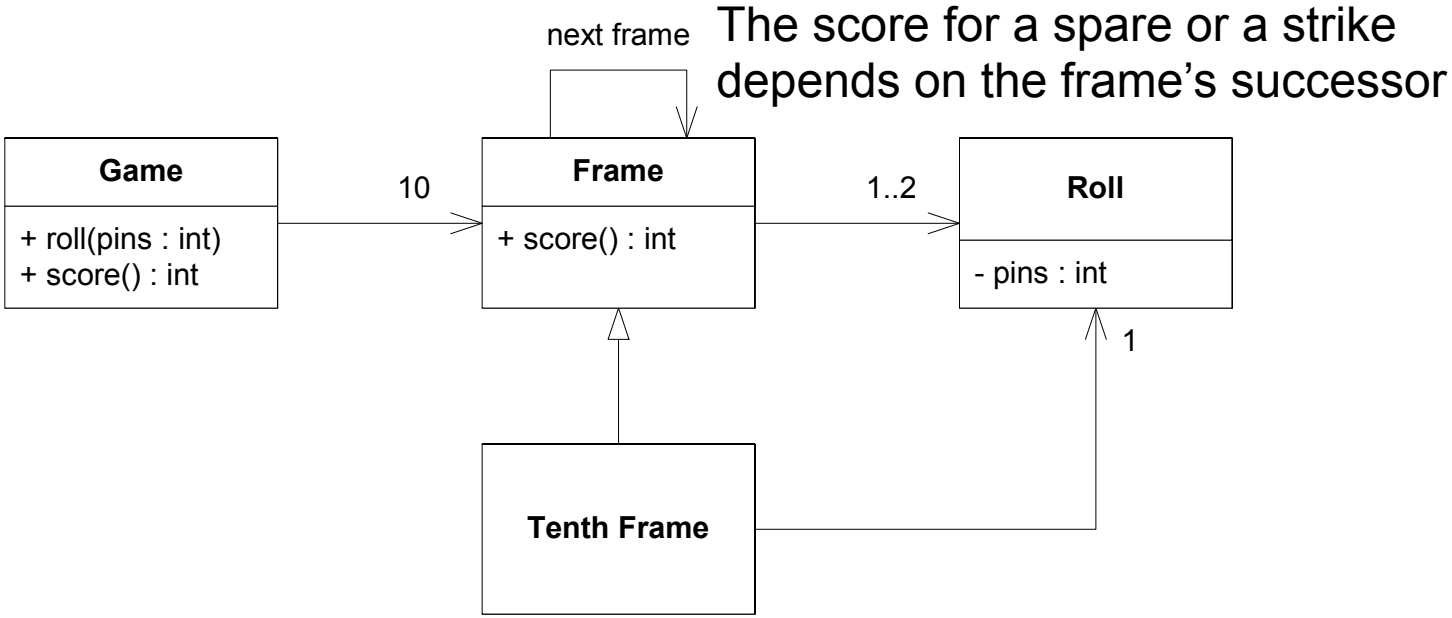
The tenth frame has two or three rolls.
It is different from all the other frames.

A quick design session



The score function must iterate through all the frames, and calculate all their scores.

A quick design session



What is TDD?

» Three Rules (it is a discipline)

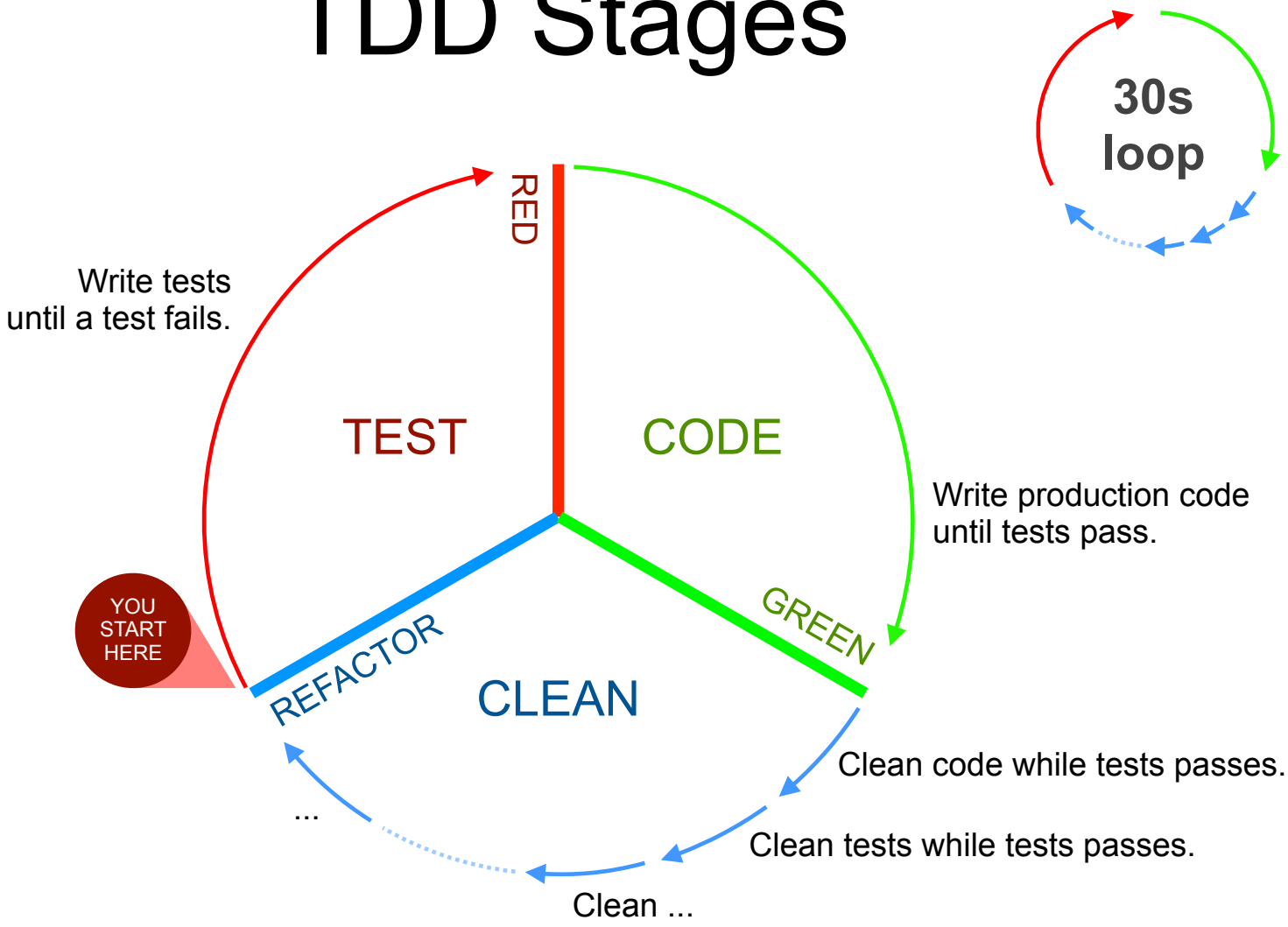
1. You are not allowed to write any production code unless it is to make a failing unit test pass.

2. You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.

3. You are not allowed to write any more production code than is sufficient to pass the one failing unit test.

http://www.butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd

TDD Stages



Test-Driven Development By Example — Kent Beck

Begin.

- Create the BowlingGame project
- Create a test file `BowlingTest.java`

```
class BowlingTest {

}
```

Begin.

- Create the BowlingGame project
- Create a test file `BowlingTest.java`

```
public class GameTest {
}
```

Try to execute the test and verify that you get the following message:

```
Warning: GameTest isn't test class
```

The first test.

```
import org.junit.jupiter.api.Test;

public class BowlingGameTest {
    @Test
    public void create_game() {
        var g = new Game();
    }
}
```

```
Cannot resolve symbol 'Game'
```

The first test.

```
import org.junit.jupiter.api.Test;

public class BowlingGameTest {
    @Test
    public void create_game() {
        var g = new Game();
    }
}
```

right clicked, autofixed

```
public class Game {
}
```

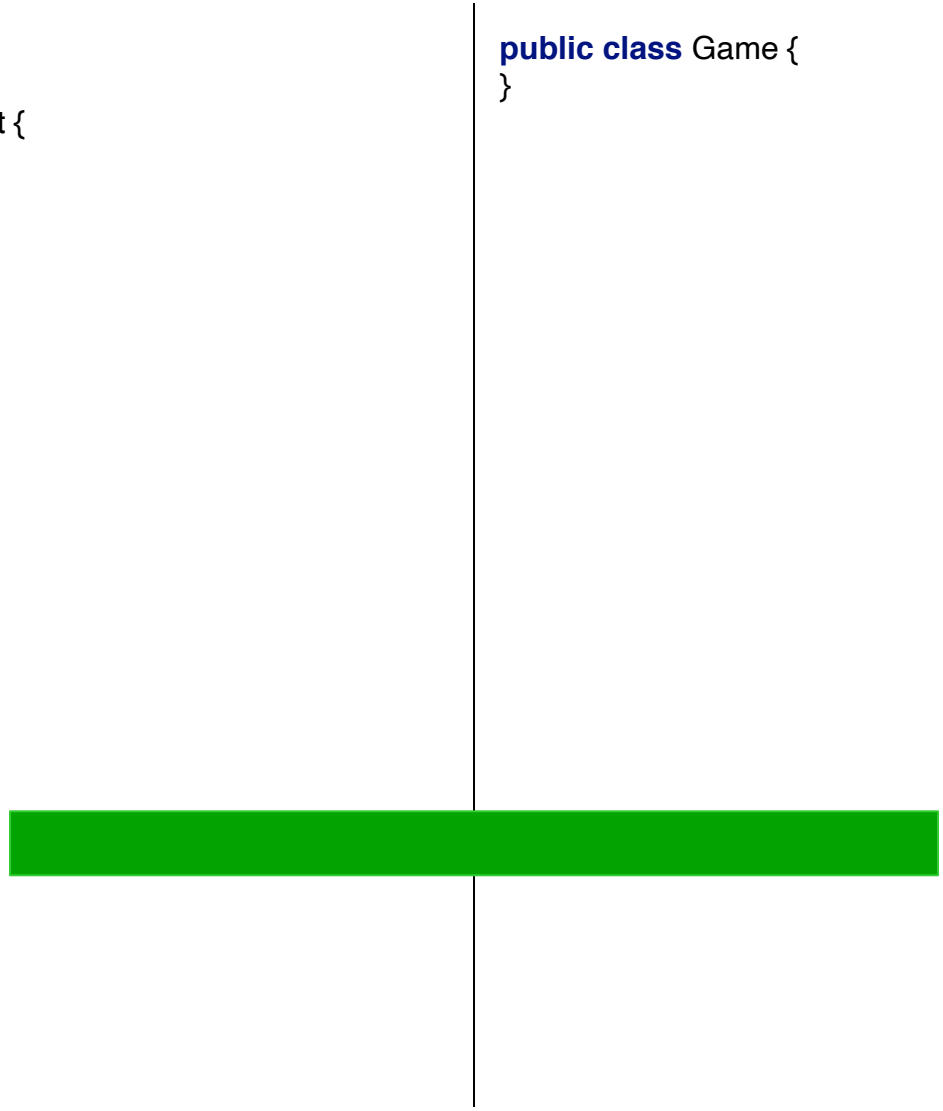
commit 2

The first test.

```java
import org.junit.jupiter.api.Test;

public class BowlingGameTest {
    @Test
    public void create_game() {
        var g = new Game();
    }
}
```

```java
public class Game {
}
```



The first test.

```java
import org.junit.jupiter.api.Test;

public class BowlingGameTest {
    @Test
    public void create_game() {
        var g = new Game();
    }
    @Test
    public void roll_a_ball() {
        var g = new Game();
        g.roll(0);
    }
}
```

```java
public class Game {
}
```

```
Cannot resolve method 'roll' in 'Game'
```

commit 4

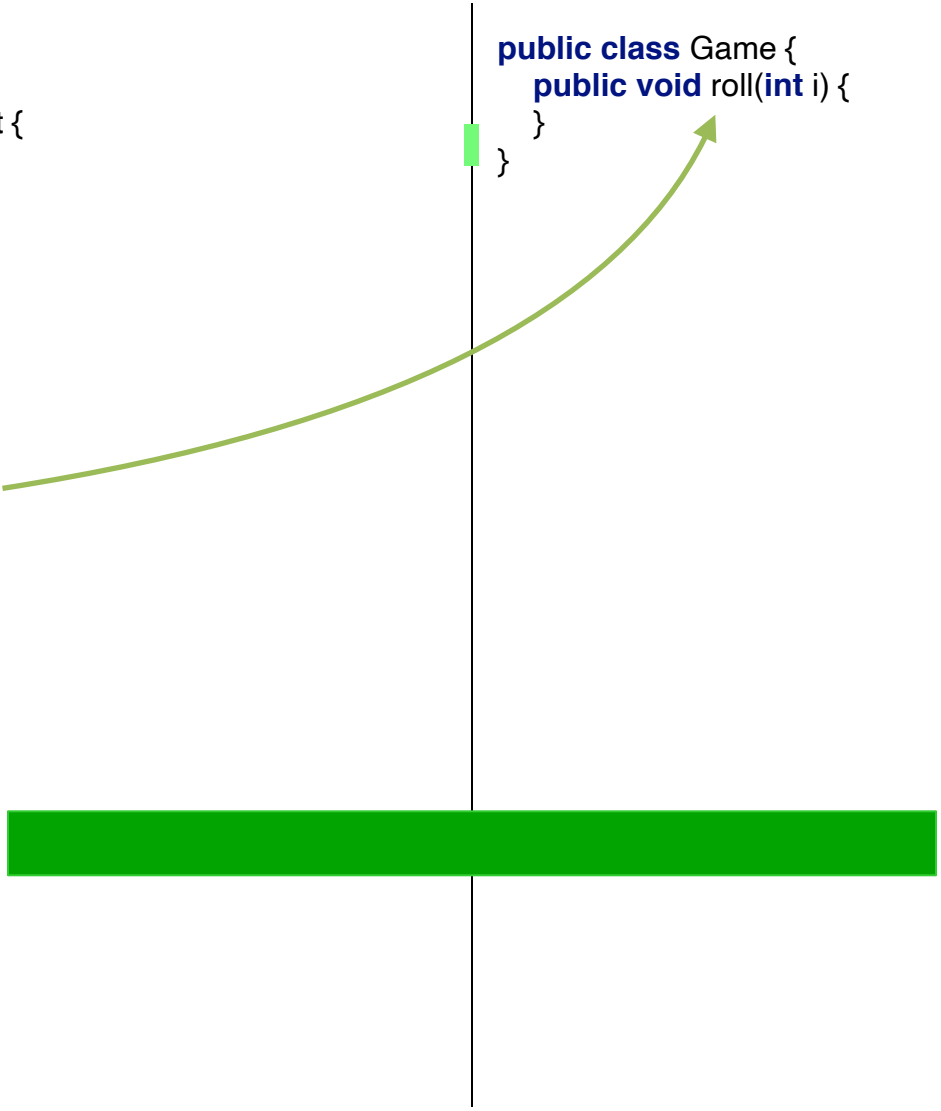
The first test.

```
import org.junit.jupiter.api.Test;

public class BowlingGameTest {
    @Test
    public void create_game() {
        var g = new Game();
    }
    @Test
    public void roll_a_ball() {
        var g = new Game();
        g.roll(0);
    }
}
```

right clicked, autofixed

```
public class Game {
    public void roll(int i) {
    }
}
```



- Game creation is duplicated

The first test.

```
import org.junit.jupiter.api.Test;

public class BowlingGameTest {
    @Test
    public void create_game() {
        var g = new Game();
    }
    @Test
    public void roll_a_ball() {
        var g = new Game();
        g.roll(0);
    }
}
```

```
public class Game {
    public void roll(int i) {
    }
}
```

- Game creation is duplicated

commit 5

The first test.

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class BowlingGameTest {
    private Game g;
    @BeforeEach
    public void setUp() {
        g = new Game();
    }
    @Test
X   public void create_game() {
    }
    @Test
    public void roll_a_ball() {
        var g = new Game();
        g.roll(0);
    }
}
```

```
public class Game {
    public void roll(int i) {
    }
}
```

- ~~Game creation is duplicated~~

commit 6

The first test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class BowlingGameTest {
    private Game g;
    @BeforeEach
    public void setUp() {
        g = new Game();
    }
    @Test
    public void create_game() {
    }
    @Test
    public void roll_a_ball() {
        g.roll(0);
    }
}
```

X

```java
public class Game {
    public void roll(int i) {
    }
}
```

- No longer needed stairstep-test

The first test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class BowlingGameTest {
    private Game g;
    @BeforeEach
    public void setUp() {
        g = new Game();
    }
    @Test
    public void create_game() {
    }
    @Test
    public void roll_a_ball() {
        g.roll(0);
    }
}
```

```java
public class Game {
    public void roll(int i) {
    }
}
```

- ~~No longer needed stairstep-test~~

commit 7

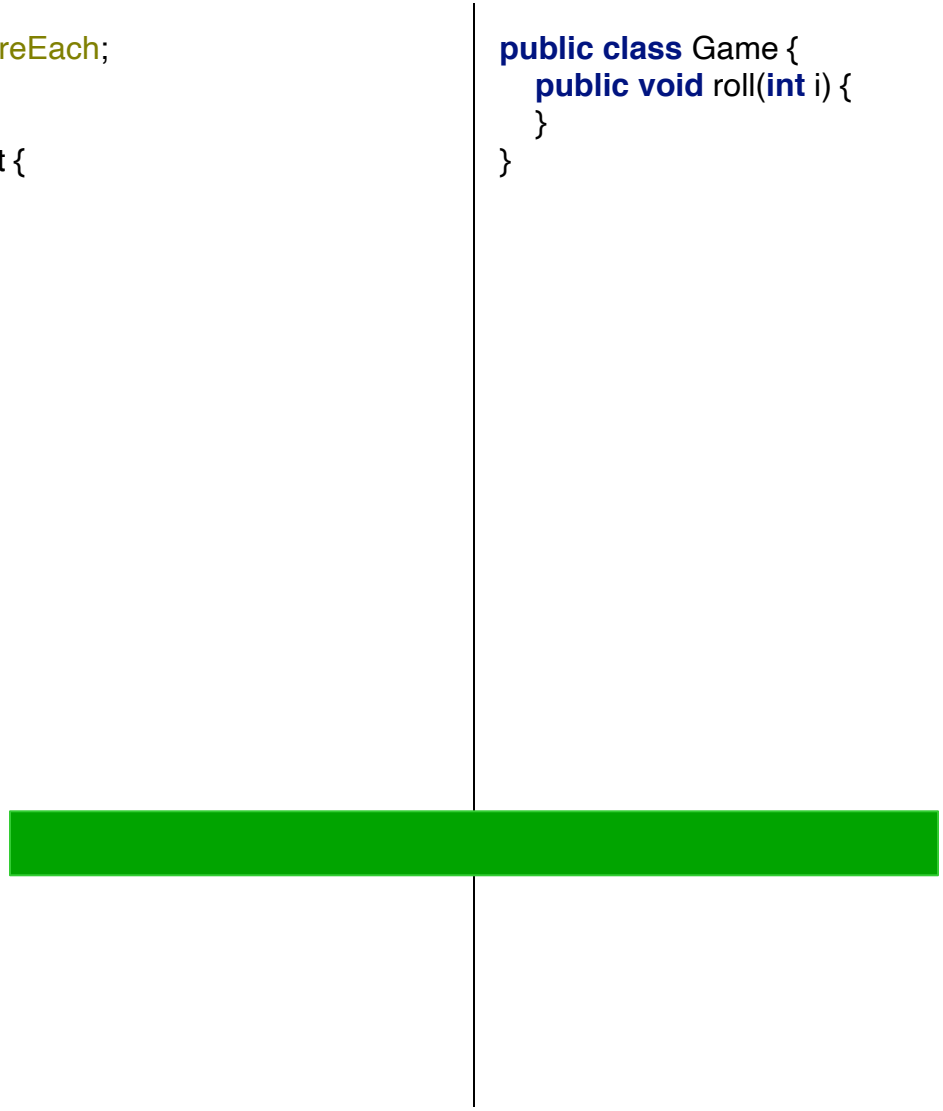
The first test.

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class BowlingGameTest {
    private Game g;
    @BeforeEach
    public void setUp() {
        g = new Game();
    }
    @Test
    public void roll_a_ball() {
        g.roll(0);
    }
}
```

X

```
public class Game {
    public void roll(int i) {
    }
}
```



The first test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach
    public void setUp() {
        g = new Game();
    }
    @Test
    public void roll_a_ball() {
        g.roll(0);
    }
    @Test
    public void gutter_game() {
        for (var i = 0; i < 20; i += 1)
            g.roll(0);

        assertThat(g.score()).isEqualTo(0);
    }
}
```

```java
public class Game {
    public void roll(int i) {
    }
}
```

```
Cannot resolve method 'score' in 'Game'
```

The first test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach
    public void setUp() {
        g = new Game();
    }
    @Test
    public void roll_a_ball() {
        g.roll(0);
    }
    @Test
    public void gutter_game() {
        for (var i = 0; i < 20; i += 1)
            g.roll(0);

        assertThat(g.score()).isEqualTo(0);
    }
}
```

right clicked, autofixed

```java
public class Game {
    public void roll(int i) {
    }

    public int score() {
        return -1;
    }
}
```

* NOTE: If your IDE autogenerates a return 0, change it for -1 so it fails

```
Expected: 0. But was: -1.
```

The first test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach
    public void setUp() {
        g = new Game();
    }
    @Test
    public void roll_a_ball() {
        g.roll(0);
    }
    @Test
    public void gutter_game() {
        for (var i = 0; i < 20; i += 1)
            g.roll(0);

        assertThat(g.score()).isEqualTo(0);
    }
}
```

```java
public class Game {
    public void roll(int i) {
    }

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

- No longer needed stairstep-test

The first test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach
    public void setUp() {
        g = new Game();
    }
    @Test
    public void roll_a_ball() {
        g.roll(0);
    }
    @Test
    public void gutter_game() {
        for (var i = 0; i < 20; i += 1)
            g.roll(0);

        assertThat(g.score()).isEqualTo(0);
    }
}
```

```java
public class Game {
    public void roll(int i) {
    }

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

commit 11

~~- No longer needed stairstep-test~~

The first test.

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach
    public void setUp() {
        g = new Game();
X   }
    @Test
    public void gutter_game() {
        for (var i = 0; i < 20; i += 1)
            g.roll(0);

        assertThat(g.score()).isEqualTo(0);
    }
}
```

```
public class Game {
    public void roll(int i) {
    }

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

The Second test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach
    public void setUp() {
        g = new Game();
    }
    @Test
    public void gutter_game() {
        for (var i = 0; i < 20; i += 1)
            g.roll(0);

        assertThat(g.score()).isEqualTo(0);
    }
    @Test
    public void all_ones() {
        for (var i = 0; i < 20; i += 1)
            g.roll(1);

        assertThat(g.score()).isEqualTo(20);
    }
}
```

```java
public class Game {
    public void roll(int i) {
    }

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

- Roll loop is duplicated

The Second test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach
    public void setUp() {
        g = new Game();
    }
    @Test
    public void gutter_game() {
        for (var i = 0; i < 20; i += 1)
            g.roll(0);

        assertThat(g.score()).isEqualTo(0);
    }
    @Test
    public void all_ones() {
        for (var i = 0; i < 20; i += 1)
            g.roll(1);

        assertThat(g.score()).isEqualTo(20);
    }
}
```

```java
public class Game {
    public void roll(int i) {
    }

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

- Roll loop is duplicated

commit 12

The Second test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach
    public void setUp() {
        g = new Game();
    }
    @Test
    public void gutter_game() {
        for (var i = 0; i < 20; i += 1)
            g.roll(0);

        assertThat(g.score()).isEqualTo(0);
    }
    @Test
    public void all_ones() {
        for (var i = 0; i < 20; i += 1)
            g.roll(1);

        assertThat(g.score()).isEqualTo(20);
    }
}
```

```java
public class Game {
    public void roll(int i) {
    }

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

```
Expected: 20. But was: 0.
```

- Roll loop is duplicated

commit 13

The Second test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach
    public void setUp() {
        g = new Game();
    }
    @Test
    public void gutter_game() {
        for (var i = 0; i < 20; i += 1)
            g.roll(0);

        assertThat(g.score()).isEqualTo(0);
    }
    @Test
    public void all_ones() {
        for (var i = 0; i < 20; i += 1)
            g.roll(1);

        assertThat(g.score()).isEqualTo(20);
    }
}
```

```java
public class Game {
    private int score = 0;

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

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

- Roll loop is duplicated

commit 14

The Second test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach
    public void setUp() {
        g = new Game();
    }
    @Test
    public void gutter_game() {
        var rolls = 20;
        var pins = 0;
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);

        assertThat(g.score()).isEqualTo(0);
    }
    @Test
    public void all_ones() {
        for (var i = 0; i < 20; i += 1)
            g.roll(1);

        assertThat(g.score()).isEqualTo(20);
    }
}
```

```java
public class Game {
    private int score = 0;

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

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

- Roll loop is duplicated

commit 15

The Second test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach
    public void setUp() {
        g = new Game();
    }
    @Test
    public void gutter_game() {
        var rolls = 20;
        var pins = 0;
        rollMany(rolls, pins);

        assertThat(g.score()).isEqualTo(0);
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test
    public void all_ones() {
        for (var i = 0; i < 20; i += 1)
            g.roll(1);

        assertThat(g.score()).isEqualTo(20);
    }
}
```

selected and right clicked, extract method

```java
public class Game {
    private int score = 0;

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

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

- Roll loop is duplicated

commit 16

The Second test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach
    public void setUp() {
        g = new Game();
    }
    @Test
    public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test
    public void all_ones() {
        for (var i = 0; i < 20; i += 1)
            g.roll(1);

        assertThat(g.score()).isEqualTo(20);
    }
}
```

```java
public class Game {
    private int score = 0;

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

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

- Roll loop is duplicated

commit 17

The Second test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach
    public void setUp() {
        g = new Game();
    }
    @Test
    public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test
    public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
}
```

```java
public class Game {
    private int score = 0;

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

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

- ~~Roll loop is duplicated~~

commit 18

The Second test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach
    public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test
    public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test
    public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
}
```

```java
public class Game {
    private int score = 0;

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

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

The Third test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
}
```

```java
public class Game {
    private int score = 0;

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

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

- ugly comment in test.

The Third test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
}
```

```java
public class Game {
    private int score = 0;

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

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

- ugly comment in test.

commit 19

The Third test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
}
```

```java
public class Game {
    private int score = 0;

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

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

```
Expected: 16. But was: 13.
```

- ugly comment in test.

The Third test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
}
```

```java
public class Game {
    private int score = 0;

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

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

tempted to use flag to remember previous roll. So design must be wrong.

- ugly comment in test.

The Third test.

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
}
```

```
public class Game {
    private int score = 0;

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

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

roll() calculates score, but name does not imply that.

score() does not calculate score, but name implies that it does.

Design is wrong. Responsibilities are misplaced.

- ugly comment in test.

commit 20

The Third test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
//    @Test public void one_spare() {
//        g.roll(5);
//        g.roll(5); // spare
//        g.roll(3);
//        rollMany(17, 0);
//        assertThat(g.score()).isEqualTo(16);
//    }
}
```

```java
public class Game {
    private int score = 0;

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

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

- ugly comment in test.

commit 21

The Third test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
//    @Test public void one_spare() {
//        g.roll(5);
//        g.roll(5); // spare
//        g.roll(3);
//        rollMany(17, 0);
//        assertThat(g.score()).isEqualTo(16);
//    }
}
```

```java
public class Game {
    private int score = 0;
    private int rolls[] = new int[21];
    private int currentRoll = 0;

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

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

- ugly comment in test.

commit 22

The Third test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
//    @Test public void one_spare() {
//        g.roll(5);
//        g.roll(5); // spare
//        g.roll(3);
//        rollMany(17, 0);
//        assertThat(g.score()).isEqualTo(16);
//    }
}
```

```java
public class Game {
    private int score = 0;
    private int rolls[] = new int[21];
    private int currentRoll = 0;

    public void roll(int pins) {
        score += pins;
        rolls[currentRoll++] = pins;
    }

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

- ugly comment in test.

commit 23

The Third test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
//    @Test public void one_spare() {
//        g.roll(5);
//        g.roll(5); // spare
//        g.roll(3);
//        rollMany(17, 0);
//        assertThat(g.score()).isEqualTo(16);
//    }
}
```

```java
public class Game {
    private int score = 0;
    private int rolls[] = new int[21];
    private int currentRoll = 0;

    public void roll(int pins) {
        score += pins;
        rolls[currentRoll++] = pins;
    }

    public int score() {
        int score = 0;
        for (int i = 0; i < rolls.length; i++)
            score += rolls[i];
        return score;
    }
}
```

- ugly comment in test.

commit 24

The Third test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
//    @Test public void one_spare() {
//        g.roll(5);
//        g.roll(5); // spare
//        g.roll(3);
//        rollMany(17, 0);
//        assertThat(g.score()).isEqualTo(16);
//    }
}
```

```java
public class Game {
    private int score = 0;
    private 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;
    }
}
```

- ugly comment in test.

commit 25

The Third test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
//    @Test public void one_spare() {
//        g.roll(5);
//        g.roll(5); // spare
//        g.roll(3);
//        rollMany(17, 0);
//        assertThat(g.score()).isEqualTo(16);
//    }
}
```

```java
public class Game {
    private 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;
    }
}
```

- ugly comment in test.

commit 26

The Third test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
}
```

```java
public class Game {
    private 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;
    }
}
```

```
Expected: 16. But was: 13.
```

- ugly comment in test.

The Third test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
}
```

```java
public class Game {
    private 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++) {
            if (rolls[i] + rolls[i + 1] == 10) // spare
                score += ...
            score += rolls[i];
        }
        return score;
    }
}
```

This isn't going to work because i might not refer to the first ball of the frame.

Design is still wrong.

Need to walk through array two balls (one frame) at a time.

- ugly comment in test.

commit 27

The Third test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
//    @Test public void one_spare() {
//        g.roll(5);
//        g.roll(5); // spare
//        g.roll(3);
//        rollMany(17, 0);
//        assertThat(g.score()).isEqualTo(16);
//    }
}
```

```java
public class Game {
    private 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;
    }
}
```

- ugly comment in test.

commit 28

The Third test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
//    @Test public void one_spare() {
//        g.roll(5);
//        g.roll(5); // spare
//        g.roll(3);
//        rollMany(17, 0);
//        assertThat(g.score()).isEqualTo(16);
//    }
}
```

```java
public class Game {
    private int rolls[] = new int[21];
    private int currentRoll = 0;

    public void roll(int pins) {
        rolls[currentRoll++] = pins;
    }

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

- ugly comment in test.

commit 29

The Third test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
}
```

```java
public class Game {
    private int rolls[] = new int[21];
    private int currentRoll = 0;

    public void roll(int pins) {
        rolls[currentRoll++] = pins;
    }

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

```
Expected: 16. But was: 13.
```

- ugly comment in test.

commit 30

The Third test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
}
```

```java
public class Game {
    private int rolls[] = new int[21];
    private int currentRoll = 0;

    public void roll(int pins) {
        rolls[currentRoll++] = pins;
    }

    public int score() {
        int score = 0;
        int i = 0;
        for (int frame = 0; frame < 10; frame++) {
            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;
    }
}
```

-ugly comment in test.
-ugly comment in conditional.
-i is a bad name for this variable

The Third test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
}
```

```java
public class Game {
    private int rolls[] = new int[21];
    private int currentRoll = 0;

    public void roll(int pins) {
        rolls[currentRoll++] = pins;
    }

    public int score() {
        int score = 0;
        int i = 0;
        for (int frame = 0; frame < 10; frame++) {
            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;
    }
}
```

commit 31

-ugly comment in test.
-ugly comment in conditional.

The Third test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
}
```

```java
public class Game {
    private int rolls[] = new int[21];
    private int currentRoll = 0;

    public void roll(int pins) {
        rolls[currentRoll++] = pins;
    }

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

right clicked, IDE refactor, rename variable

-ugly comment in test.

commit 32

The Third test.

```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static com.google.common.truth.Truth.assertThat;

public class BowlingGameTest {
    private Game g;
    @BeforeEach public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
}
```

```java
public class Game {
    private int rolls[] = new int[21];
    private int currentRoll = 0;

    public void roll(int pins) {
        rolls[currentRoll++] = pins;
    }

    public int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame = 0; frame < 10; frame++) {
            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;
    }
}
```

selected and right clicked, extract method

commit 33

The Third test.

```java
public class BowlingGameTest {
    private Game g;
    @BeforeEach public void setUp() {
        g = new Game();
    }
    private void rollMany(int rolls, int pins) {
        for (var i = 0; i < rolls; i += 1)
            g.roll(pins);
    }
    private void rollSpare() {
        g.roll(5);
        g.roll(5);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        rollSpare();
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
}
```

selected and right clicked, extract method

```java
public class Game {
    private int rolls[] = new int[21];
    private int currentRoll = 0;

    public void roll(int pins) {
        rolls[currentRoll++] = pins;
    }

    public int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame = 0; frame < 10; frame++) {
            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;
    }
}
```

- ugly comment in test.

commit 34

The Fourth test.

```java
public class BowlingGameTest {
    ...
    private void rollSpare() {
        g.roll(5);
        g.roll(5);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        rollSpare();
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
    @Test public void one_strike() {
        g.roll(10); // strike
        g.roll(3);
        g.roll(4);
        rollMany(16, 0);
        assertThat(g.score()).isEqualTo(24);
    }
}
```

```java
public class Game {
    private int rolls[] = new int[21];
    private int currentRoll = 0;

    public void roll(int pins) {
        rolls[currentRoll++] = pins;
    }

    public int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame = 0; frame < 10; frame++) {
            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;
    }
}
```

```
Expected: 24. But was: 17.
```

- ugly comment in test

commit 35

The Fourth test.

```java
public class BowlingGameTest {
    ...
    private void rollSpare() {
        g.roll(5);
        g.roll(5);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        rollSpare();
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
    @Test public void one_strike() {
        g.roll(10); // strike
        g.roll(3);
        g.roll(4);
        rollMany(16, 0);
        assertThat(g.score()).isEqualTo(24);
    }
}
```

```java
public class Game {
    private int rolls[] = new int[21];
    private int currentRoll = 0;
    public void roll(int pins) {
        rolls[currentRoll++] = pins;
    }
    public int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame = 0; frame < 10; frame++) {
            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;
    }
    private boolean isSpare(int frameIndex) {
        return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
    }
}
```

- ugly comment in test.
- ugly comment in conditional.
- ugly expressions.

The Fourth test.

```java
public class BowlingGameTest {
    ...
    private void rollSpare() {
        g.roll(5);
        g.roll(5);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        rollSpare();
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
    @Test public void one_strike() {
        g.roll(10); // strike
        g.roll(3);
        g.roll(4);
        rollMany(16, 0);
        assertThat(g.score()).isEqualTo(24);
    }
}
```

```java
public class Game {
    private int rolls[] = new int[21];
    private int currentRoll = 0;
    public void roll(int pins) {
        rolls[currentRoll++] = pins;
    }
    public int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame = 0; frame < 10; frame++) {
            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;
    }
    private boolean isSpare(int frameIndex) {
        return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
    }
}
```

commit 36

- ugly comment in test.
- ugly comment in conditional.
- ugly expressions.

The Fourth test.

```java
public class BowlingGameTest {
    ...
    private void rollSpare() {
        g.roll(5);
        g.roll(5);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        rollSpare();
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
    @Test public void one_strike() {
        g.roll(10); // strike
        g.roll(3);
        g.roll(4);
        rollMany(16, 0);
        assertThat(g.score()).isEqualTo(24);
    }
}
```

```java
public class Game {
    private int rolls[] = new int[21];
    private int currentRoll = 0;
    public void roll(int pins) {
        rolls[currentRoll++] = pins;
    }
    public int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame = 0; frame < 10; frame++) {
            if (rolls[frameIndex] == 10) { // strike
                score += 10 + strikeBonus(frameIndex);
                frameIndex++;
            } else if (isSpare(frameIndex)) {
                score += 10 + rolls[frameIndex + 2];
                frameIndex += 2;
            } else {
                score += rolls[frameIndex] + rolls[frameIndex + 1];
                frameIndex += 2;
            }
        }
        return score;
    }
    private int strikeBonus(int frameIndex) {
        return rolls[frameIndex + 1] + rolls[frameIndex + 2];
    }
    private boolean isSpare(int frameIndex) {
        return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
    }
}
```

commit 37

- ugly comment in test.
- ugly comment in conditional.
- ugly expressions.

The Fourth test.

```java
public class BowlingGameTest {
    ...
    private void rollSpare() {
        g.roll(5);
        g.roll(5);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        rollSpare();
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
    @Test public void one_strike() {
        g.roll(10); // strike
        g.roll(3);
        g.roll(4);
        rollMany(16, 0);
        assertThat(g.score()).isEqualTo(24);
    }
}
```

```java
public class Game {
    private int rolls[] = new int[21];
    private int currentRoll = 0;
    public void roll(int pins) {
        rolls[currentRoll++] = pins;
    }
    public int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame = 0; frame < 10; frame++) {
            if (rolls[frameIndex] == 10) { // strike
                score += 10 + strikeBonus(frameIndex);
                frameIndex++;
            } else if (isSpare(frameIndex)) {
                score += 10 + spareBonus(frameIndex);
                frameIndex += 2;
            } else {
                score += rolls[frameIndex] + rolls[frameIndex + 1];
                frameIndex += 2;
            }
        }
        return score;
    }
    private int spareBonus(int frameIndex) {
        return rolls[frameIndex + 2];
    }
    private int strikeBonus(int frameIndex) {
        return rolls[frameIndex + 1] + rolls[frameIndex + 2];
    }
    private boolean isSpare(int frameIndex) {
        return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
    }
}
```

commit 38

- ugly comment in test.
- ugly comment in conditional.
- ~~ugly expressions.~~

The Fourth test.

```
public class BowlingGameTest {
    ...
    private void rollSpare() {
        g.roll(5);
        g.roll(5);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        rollSpare();
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
    @Test public void one_strike() {
        g.roll(10); // strike
        g.roll(3);
        g.roll(4);
        rollMany(16, 0);
        assertThat(g.score()).isEqualTo(24);
    }
}
```

```
public class Game {
    ...
    public int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame = 0; frame < 10; frame++) {
            if (rolls[frameIndex] == 10) { // strike
                score += 10 + strikeBonus(frameIndex);
                frameIndex++;
            } else if (isSpare(frameIndex)) {
                score += 10 + spareBonus(frameIndex);
                frameIndex += 2;
            } else {
                score += sumOfBallsInFrame(frameIndex);
                frameIndex += 2;
            }
        }
        return score;
    }
    private int sumOfBallsInFrame(int frameIndex) {
        return rolls[frameIndex] + rolls[frameIndex + 1];
    }
    private int spareBonus(int frameIndex) {
        return rolls[frameIndex + 2];
    }
    private int strikeBonus(int frameIndex) {
        return rolls[frameIndex + 1] + rolls[frameIndex + 2];
    }
    private boolean isSpare(int frameIndex) {
        return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
    }
}
```

commit 39

-ugly comment in test
~~-ugly comment in conditional.~~
~~-ugly expressions.~~

The Fourth test.

```java
public class BowlingGameTest {
    ...
    private void rollSpare() {
        g.roll(5);
        g.roll(5);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        rollSpare();
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
    @Test public void one_strike() {
        g.roll(10); // strike
        g.roll(3);
        g.roll(4);
        rollMany(16, 0);
        assertThat(g.score()).isEqualTo(24);
    }
}
```

```java
public class Game {
    ...
    public int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame = 0; frame < 10; frame++) {
            if (isStrike(frameIndex)) {
                score += 10 + strikeBonus(frameIndex);
                frameIndex++;
            } else if (isSpare(frameIndex)) {
                score += 10 + spareBonus(frameIndex);
                frameIndex += 2;
            } else {
                score += sumOfBallsInFrame(frameIndex);
                frameIndex += 2;
            }
        }
        return score;
    }
    private boolean isStrike(int frameIndex) {
        return rolls[frameIndex] == 10;
    }
    private int sumOfBallsInFrame(int frameIndex) {
        return rolls[frameIndex] + rolls[frameIndex + 1];
    }
    private int spareBonus(int frameIndex) {
        return rolls[frameIndex + 2];
    }
    private int strikeBonus(int frameIndex) {
        return rolls[frameIndex + 1] + rolls[frameIndex + 2];
    }
    private boolean isSpare(int frameIndex) {
        return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
    }
}
```

commit 40

- ~~ugly comment in test~~
- ~~ugly comment in conditional.~~
- ~~ugly expressions.~~

The Fourth test.

```java
public class BowlingGameTest {
    ...
    private void rollSpare() {
        g.roll(5);
        g.roll(5);
    }
    private void rollStrike() {
        g.roll(10);
    }
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        rollSpare();
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
    @Test public void one_strike() {
        rollStrike();
        g.roll(3);
        g.roll(4);
        rollMany(16, 0);
        assertThat(g.score()).isEqualTo(24);
    }
}
```

```java
public class Game {
    ...
    public int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame = 0; frame < 10; frame++) {
            if (isStrike(frameIndex)) {
                score += 10 + strikeBonus(frameIndex);
                frameIndex++;
            } else if (isSpare(frameIndex)) {
                score += 10 + spareBonus(frameIndex);
                frameIndex += 2;
            } else {
                score += sumOfBallsInFrame(frameIndex);
                frameIndex += 2;
            }
        }
        return score;
    }
    private boolean isStrike(int frameIndex) {
        return rolls[frameIndex] == 10;
    }
    private int sumOfBallsInFrame(int frameIndex) {
        return rolls[frameIndex] + rolls[frameIndex + 1];
    }
    private int spareBonus(int frameIndex) {
        return rolls[frameIndex + 2];
    }
    private int strikeBonus(int frameIndex) {
        return rolls[frameIndex + 1] + rolls[frameIndex + 2];
    }
    private boolean isSpare(int frameIndex) {
        return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
    }
}
```

The Fifth test.

```java
public class BowlingGameTest {
    ...
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        rollSpare();
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
    @Test public void one_strike() {
        rollStrike();
        g.roll(3);
        g.roll(4);
        rollMany(16, 0);
        assertThat(g.score()).isEqualTo(24);
    }
    @Test public void perfect_game() {
        rollMany(12, 10);
        assertThat(g.score()).isEqualTo(300);
    }
}
```

```java
public class Game {
    ...
    public int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame = 0; frame < 10; frame++) {
            if (isStrike(frameIndex)) {
                score += 10 + strikeBonus(frameIndex);
                frameIndex++;
            } else if (isSpare(frameIndex)) {
                score += 10 + spareBonus(frameIndex);
                frameIndex += 2;
            } else {
                score += sumOfBallsInFrame(frameIndex);
                frameIndex += 2;
            }
        }
        return score;
    }
    private boolean isStrike(int frameIndex) {
        return rolls[frameIndex] == 10;
    }
    private int sumOfBallsInFrame(int frameIndex) {
        return rolls[frameIndex] + rolls[frameIndex + 1];
    }
    private int spareBonus(int frameIndex) {
        return rolls[frameIndex + 2];
    }
    private int strikeBonus(int frameIndex) {
        return rolls[frameIndex + 1] + rolls[frameIndex + 2];
    }
    private boolean isSpare(int frameIndex) {
        return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
    }
}
```

The Fifth test.

```java
public class BowlingGameTest {
    ...
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        rollSpare();
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
    @Test public void one_strike() {
        rollStrike();
        g.roll(3);
        g.roll(4);
        rollMany(16, 0);
        assertThat(g.score()).isEqualTo(24);
    }
    @Test public void perfect_game() {
        rollMany(12, 10);
        assertThat(g.score()).isEqualTo("fail");
    }
}
```

```java
public class Game {
    ...
    public int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame = 0; frame < 10; frame++) {
            if (isStrike(frameIndex)) {
                score += 10 + strikeBonus(frameIndex);
                frameIndex++;
            } else if (isSpare(frameIndex)) {
                score += 10 + spareBonus(frameIndex);
                frameIndex += 2;
            } else {
                score += sumOfBallsInFrame(frameIndex);
                frameIndex += 2;
            }
        }
        return score;
    }
    private boolean isStrike(int frameIndex) {
        return rolls[frameIndex] == 10;
    }
    private int sumOfBallsInFrame(int frameIndex) {
        return rolls[frameIndex] + rolls[frameIndex + 1];
    }
    private int spareBonus(int frameIndex) {
        return rolls[frameIndex + 2];
    }
    private int strikeBonus(int frameIndex) {
        return rolls[frameIndex + 1] + rolls[frameIndex + 2];
    }
    private boolean isSpare(int frameIndex) {
        return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
    }
}
```

```
Expected: "fail". But was: 300.
```

The Fifth test.

```java
public class BowlingGameTest {
    ...
    @Test public void gutter_game() {
        rollMany(20, 0);
        assertThat(g.score()).isEqualTo(0);
    }
    @Test public void all_ones() {
        rollMany(20, 1);
        assertThat(g.score()).isEqualTo(20);
    }
    @Test public void one_spare() {
        rollSpare();
        g.roll(3);
        rollMany(17, 0);
        assertThat(g.score()).isEqualTo(16);
    }
    @Test public void one_strike() {
        rollStrike();
        g.roll(3);
        g.roll(4);
        rollMany(16, 0);
        assertThat(g.score()).isEqualTo(24);
    }
    @Test public void perfect_game() {
        rollMany(12, 10);
        assertThat(g.score()).isEqualTo(300);
    }
}
```

```java
public class Game {
    ...
    public int score() {
        int score = 0;
        int frameIndex = 0;
        for (int frame = 0; frame < 10; frame++) {
            if (isStrike(frameIndex)) {
                score += 10 + strikeBonus(frameIndex);
                frameIndex++;
            } else if (isSpare(frameIndex)) {
                score += 10 + spareBonus(frameIndex);
                frameIndex += 2;
            } else {
                score += sumOfBallsInFrame(frameIndex);
                frameIndex += 2;
            }
        }
        return score;
    }
    private boolean isStrike(int frameIndex) {
        return rolls[frameIndex] == 10;
    }
    private int sumOfBallsInFrame(int frameIndex) {
        return rolls[frameIndex] + rolls[frameIndex + 1];
    }
    private int spareBonus(int frameIndex) {
        return rolls[frameIndex + 2];
    }
    private int strikeBonus(int frameIndex) {
        return rolls[frameIndex + 1] + rolls[frameIndex + 2];
    }
    private boolean isSpare(int frameIndex) {
        return rolls[frameIndex] + rolls[frameIndex + 1] == 10;
    }
}
```

Test Results	73 ms
GameTest	73 ms
one_strike()	67 ms
perfect_game()	1 ms
all_ones()	3 ms
one_spare()	1 ms
gutter_game()	1 ms

End.