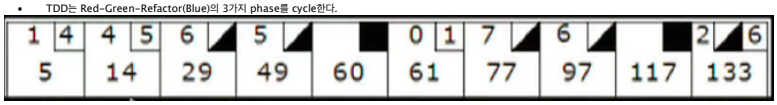
Bowling Game Demo

- TDD는 Red-Green-Refactor(Blue)의 3가지 phase를 cycle한다.



규칙

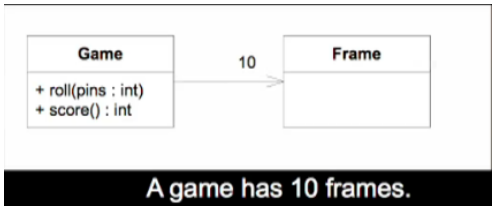
- 볼링 게임은 10개의 프레임으로 구성된다.
- 각 프레임은 대개 2 롤을 갖는다(10개의 핀을 쓰러 뜨리기 위해 2번의 기회를 갖는다).
- Spare: 10 + next first roll에서 쓰러 뜨린 핀수.
- Strike: 10 + next two rolls에서 쓰러 뜨린 핀수.
- 10th 프레임은 특별. spare 처리하면 3번 던질 수 있음.

이 예제의 목적

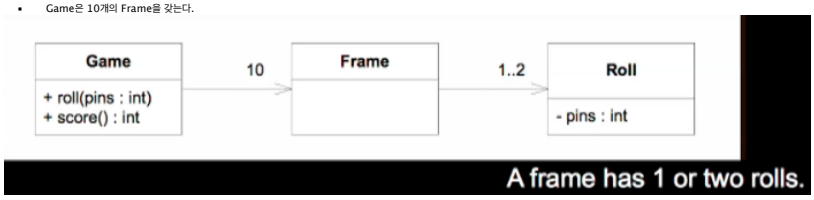
- 이 예제의 목적은 Game이라는 클래스를 생성하는 것이다.

- roll과 score라는 2개의 메소드를 갖는다.
- roll 메소드는 ball을 roll할 때마다 호출된다. 인자로는 쓰러뜨린 핀수를 갖는다.
- score 메소드는 게임이 끝난 후에만 호출되어 게임의 점수를 반환한다.

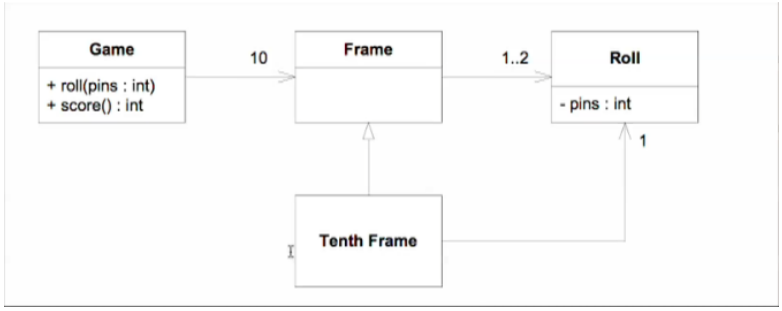
간단한 설계 단계를 거친다.



- Game은 roll, score 함수를 갖는다.
- Game은 10개의 Frame을 갖는다.

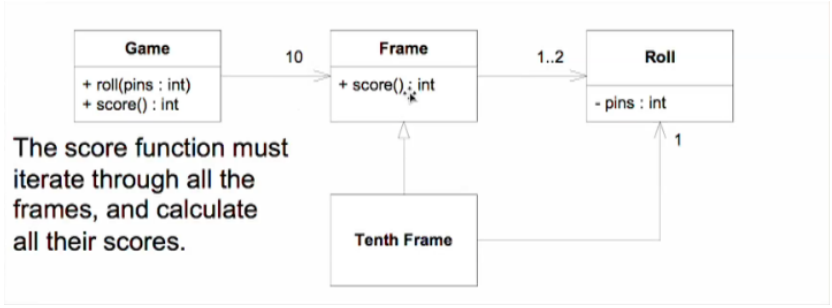


- Frame은 1..2개의 Roll을 갖는다.
- 10번 프레임은 예외를 갖는다(1..2 roll을 갖는 것이 아니라 2..3 roll을 갖는다).
- 객체지향에서 이런 예외를 어떻게 표현하나 ???

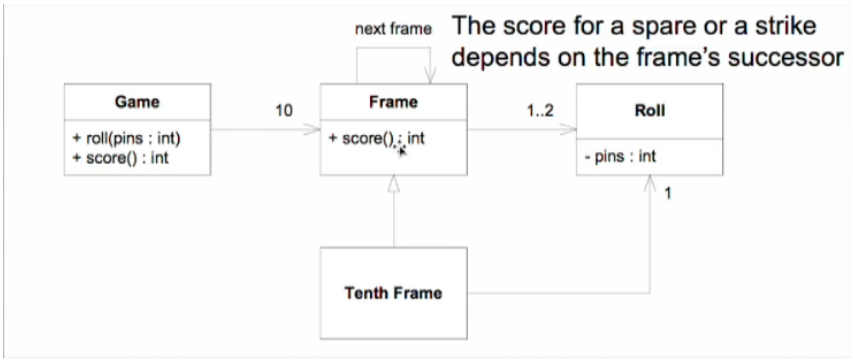


이게 score 함수의 알고리즘에 대해 생각해보자.

- frame수만큼 loop를 돌면서 각 frame의 점수를 합산할 것이다.



- frame은 roll수 만큼 loop를 돌면서 점수를 계산할 것이다. strike, spare를 위해서 look ahead roll을 해야 한다.



자 이제 설계가 있다. 이제 TDD를 할 차례이다.

- TDD에서 이상한 일을 한다. 설계를 무시하는 것이다. Well not quite. We don't ignore it. But we don't follow it. We use it kind of a guide line.

```
package bowlinggame;

import org.junit.Test;

public class BowlingTest {
    @Test
    public void nothing() throws Exception {
    }
}
```

- **아무것도 없는 nothing이라는 테스트로 시작.** 이것도 실행해 본다. 밥 아저씨는 항상 이렇게 뭔가 실행되는 것으로 시작한다고 한다. 그래서 심지어 코드가 없더라도 실행되는 뭔가를 가지고 있다.
- 그리곤 지운다. 아마 테스트 작성을 위한 설정이 제대로 되었는지 확인하는가보다. 나중에 알았지만 항상 동작하는 코드로 작업하기 위해서이다. 이게 개발자들에게는 편안함을 준다.
- **무슨 테스트를 작성해야 하나 ?**
  - failing unit test가 있기 전에는 production code를 작성하면 안된다.
  - 이미 개발자는 어떤 production code를 작성해야 하는지 안다.
  - **public class Game**을 작성해야 한다는 것을 이미 안다.
  - But I'm not allowed to. I have to write unit test first.
  - 이제부터 이런 이상한 게임을 나 자신과 해야한다.
- **어떤 테스트를 작성해야 내가 원하는 코드를 작성하게 될까 ?**
  - 어떤 failing test를 작성해야 Game.java에 public class Game을 선언하게 될까 ?
- 이제부터 red-green-refactor cycle로 들어가자.
  - red phase
    - next most interesting case but still really simple
  - green phase
    - make it pass
  - blue phase
    - refactor
- **가장 쉽고(간단하고) 흥미로운(easy/simple and interesting) 테스트부터 작성**

1st Cycle

red phase

- I must write failing unit test. public class Game을 작성하기 위해 필요한 failing unit test는 뭔가 ? canCreateGame 테스트 메소드이다.

```
package bowlinggame;

import org.junit.Test;

public class BowlingTest {
    @Test
    public void nothing() throws Exception {
    }
}
```

```
package bowlinggame;

import org.junit.Test;

public class BowlingTest {
    @Test
    public void canCreateGame() throws Exception {
        Game g = new Game();
    }
}
```

- 위 테스트를 작성하면 fail(compile error)이 발생한다.

green phase

- hot fix를 이용해서 public class Game을 만든다.

```
package bowlinggame;

public class Game {
}
```

- 이제 테스트를 수행하면 테스트가 성공한다.

blue phase

- blue phase에서는 refactor를 한다. 이 코드에서는 refactor할 것이 없다.

2nd Cycle

red phase

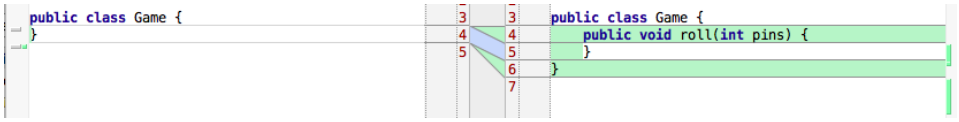
- 다음 테스트는 roll을 할 수 있는가 확인하는 메소드(canRoll)이다.

```
public class BowlingTest {
    @Test
    public void canCreateGame() throws Exception {
        Game g = new Game();
    }
}
```

```
public class BowlingTest {
    @Test
    public void canCreateGame() throws Exception {
        Game g = new Game();
    }

    @Test
    public void canRoll() {
        Game g = new Game();
        g.roll(0);
    }
}
```

green phase

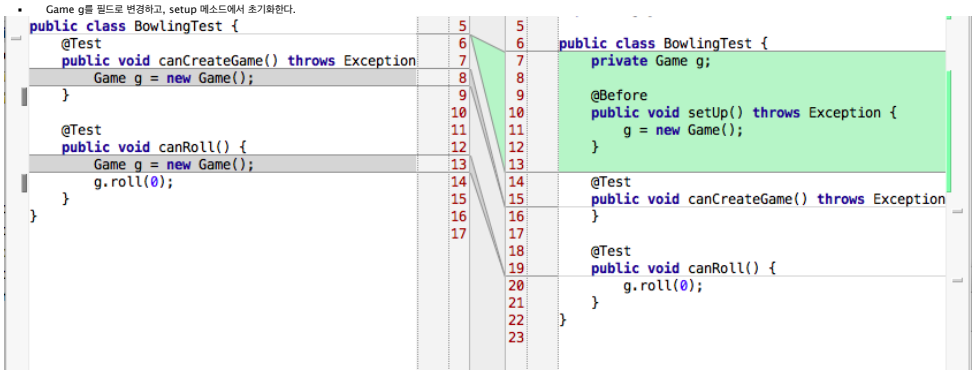
- hot fix로 roll 메소드 추가



- 테스트 수행

blue phase

- 테스트 클래스에 코드 중복이 있다.
- Game g를 필드로 변경하고, setup 메소드에서 초기화한다.



- 테스트 수행. 패스.
- empty test(canCreateGame)을 제거.
  - **후에 지울 테스트를 작성하는 것도 일반적인 일이다.**
- 테스트 수행. 패스.

3rd Cycle

red phase

- score 함수를 추가하고 싶지만, score 함수는 게임이 종료된 후에만 호출 가능하다.
- 만일 실제 코드를 작성해야 한다면 가장 단순한 실제 코드를 작성하라. 완전한 게임이 되도록 roll 메소드를 호출하는 것이다. roll할 수 있는 가장 단순한 게임은 뭔가 ? Gutter Game이다.

- compile error가 발생한다.

green phase

- compile이 되기 위해 요구되는 메소드를 추가하고 -1을 반환하여 여전히 테스트가 실패하도록 한다.

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

- 테스트. 실패한다.
- 0을 반환해서 테스트가 성공하도록 한다. 진짜 멍청한 짓이지만, 반면에 쉬운 해결책이다.
- 내 테스트가 **실패하는 경우와 성공하는 경우** 모두를 보았다. 이로 인해 **내 테스트가 정상적으로 동작함을 알았다.**

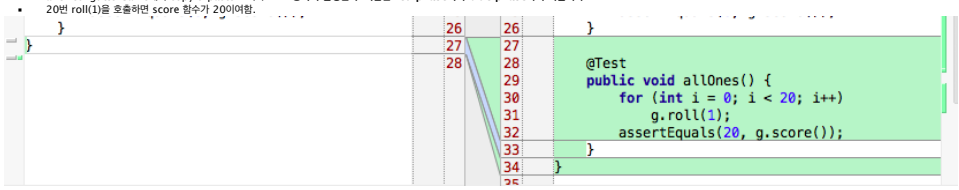
blue phase

- nothing.

4th Cycle

red phase

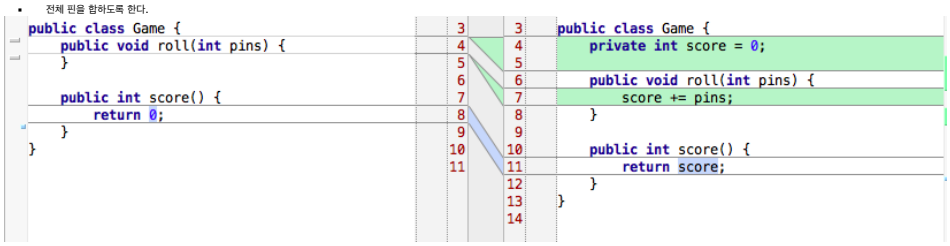
- allOnes. gutterGame에서 copy & paste로 시작. **코드 중복이 발생한다. 지금은 red phase이니 blue phase까지 기둘려라**...
- 20번 roll(1)을 호출하면 score 함수가 20이어함.



- 실패해야 함.

green phase

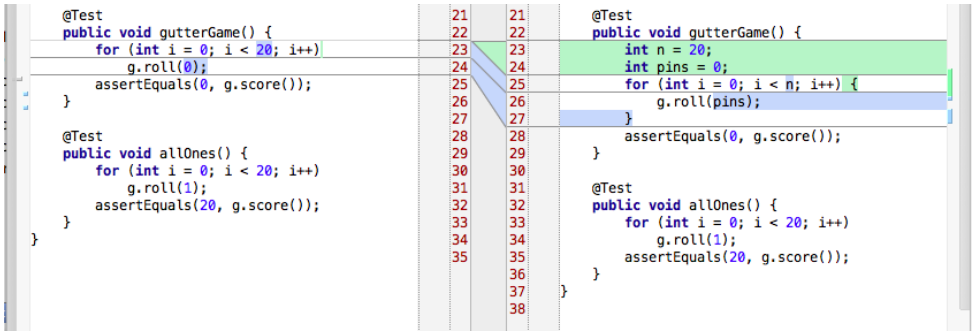
- score 함수에서 20일 리턴. 이렇게 하면 두번째 테스트는 성공하지만 첫번째 테스트가 실패한다.
- 전체 핀을 합하도록 한다.



- 동작하는 프로그램이 있는 것은 프로그래머라는 느낌을 갖게 한다.

blue phase

- 테스트에 코드 중복이 있다. 다른 점을 파라미터로 처리하기 위해 로컬 변수로 추출



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

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

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

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

- extract method(rollMany)
- intelli)에선 이렇게 하면 나머지 중복도 찾아줌.
- extract한 로컬 변수들을 인라인. canRoll 처럼 의미 없어진 테스트를 제거.

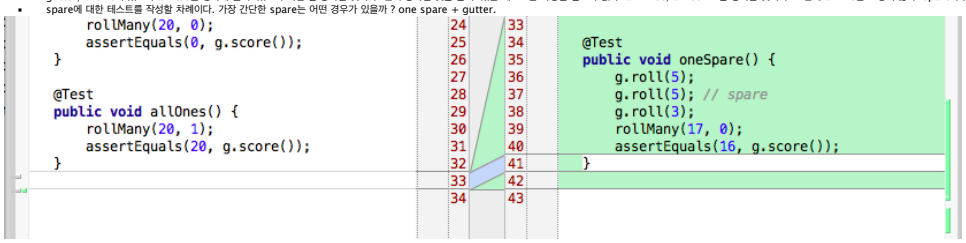
```
    @Test
    public void gutterGame() {
        int n = 20;
        int pins = 0;
        rollMany(n, pins);
        assertEquals(0, g.score());
    }
```

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

5th Cycle

red phase

- gutter, allOne이 있으니 allTwo를 생각해 볼 수 있으나 이건 잘 동작할 것이다. 뻔히 동작할 것을 알 수 있는 테스트는 작성할 필요가 없다. allThree, allFour도 잘 동작할 것이다. 그런데 allFive는 그렇지 않다. spare가 있기 때문에.
- spare에 대한 테스트를 작성할 차례이다. 가장 간단한 spare는 어떤 경우가 있을까 ? one spare + gutter.



```
        rollMany(20, 0);
        assertEquals(0, g.score());
    }

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

```

    @Test
    public void oneSpare() {
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(17, 0);
        assertEquals(16, g.score());
    }

```

- ugly comment(/* spare)가 있다. 나중에 refactor하자. 지금은 아니다.

green phase

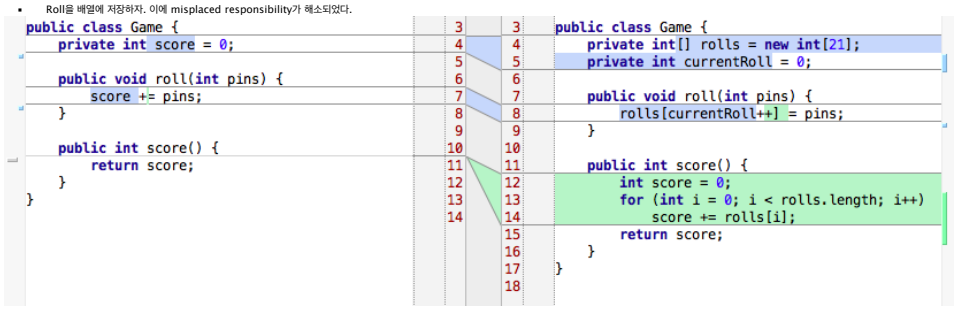
- 근데 어떻게 해야 할지 모르겠다.

```
public void roll(int pins) {
  if(pins + lastPins == 10)
    ...
}
```

- 위와 같이 하려다 보니 이상하다. 플래그 변수, 정적 변수를 사용해야 하고...
- 이처럼 끔찍한 일을 해야 하는 경우가 생길때마 잠시 멀러나야 한다.
- 뭔가 디자인이 잘 못된 것이다. 뭔 디자인 ? 단지 코딩만 했는데.
- 맞다 하지만 디자인이 잘못된 것이 분명하다. 잘못된 디자인이 있다.
- 디자인 원칙이 위배된 것이 있다.
  - 첫번째 원칙: 스코어를 계산하는 것을 의미하는 이름을 갖는 함수가 무엇인가 ?
  - score 함수이다. 근데 실제로 score를 계산하는 함수는 roll 함수이다.
  - **잘못된 책임 할당(misplaced responsibility)**이 디자인 원칙, 잘못된 디자인 냄새이다.
  - roll에선 각 roll을 저장하고, score에서 계산을 해야 한다.
  - **어쩌지. refactoring. 근데 failing test가 있다. @Ignore 처리**...

blue phase

- 이제 테스트가 수행되니 리팩토링하자.
- Roll을 배열에 저장하자. 이에 misplaced responsibility가 해소되었다.



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

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

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

```
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;
    }
}
```

6th Cycle

red phase

- oneSpare 메소드에서 @Ignore를 제거. 어전히 이 테스트는 실패.
- score함수의 for 루프에 아래와 같이 추가해 볼 수 있다.

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

- 근데 리팩토링하는 것이 더 좋아 보인다.

blue phase

- 다시 @Ignore 추가
- 테스트들이 모두 성공한다.
- 루프가 매 롤이 아니라 매 프레임별로 수행되도록 하기 위해 리팩토링하고자 한다.

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

```
    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;
    }
}
```

red phase

- @Ignore를 제거

green phase

```
    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;
    }
}
```

```
    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 comment가 있으나 패스

blue phase

- i를 firstInFrame으로 rename. 여러차례의 시도로 적절한 이름을 찾음.
- ugly comment(/* spare)를 extract method로 제거
- 테스트 코드에서도 ugly comment제거. rollSpare로 extract.

```
    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;
    }
}
```

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

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

```
    public int score() {
        int score = 0;
        int firtInFrame = 0;
        for (int frame = 0; frame < 10; frame++) {
            if(isSpare(firtInFrame)) {
                score += 10 + rolls[firtInFrame + 2];
                firtInFrame += 2;
            }
            else {
                score += rolls[firtInFrame] + rolls[firtInFra
                firtInFrame += 2;
            }
        }
        return score;
    }

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

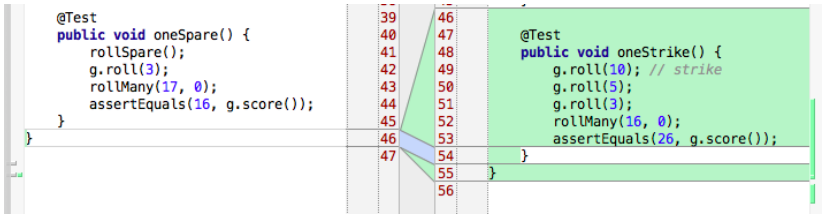
```
    @Test
    public void oneSpare() {
        g.roll(5);
        g.roll(5); // spare
        g.roll(3);
        rollMany(17, 0);
        assertEquals(16, g.score());
    }
}
```

```
    @Test
    public void oneSpare() {
        rollSpare();
        g.roll(3);
        rollMany(17, 0);
        assertEquals(16, g.score());
    }

    private void rollSpare() {
        g.roll(5);
        g.roll(5);
    }
}
```

7th Cycle

red phase

- oneSpare를 해 봤으니 oneStrike를 해보자.

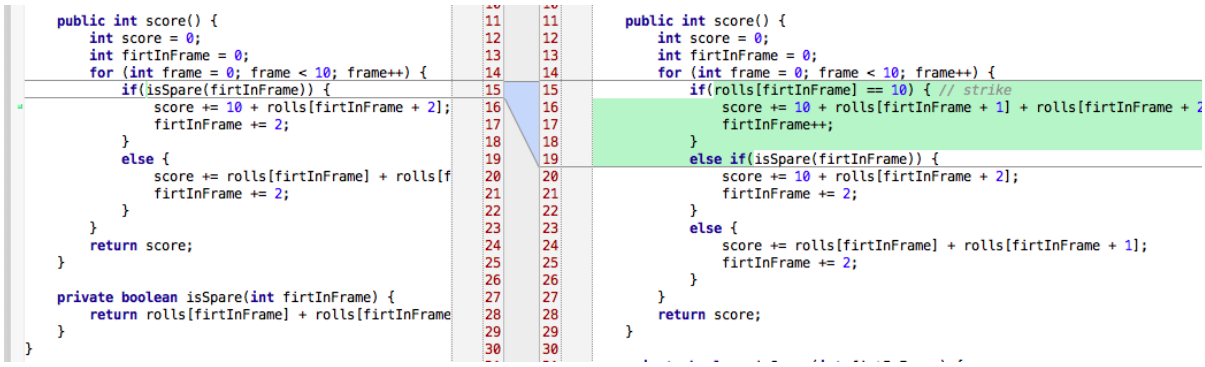


```
@Test
public void oneSpare() {
    rollSpare();
    g.roll(3);
    rollMany(17, 0);
    assertEquals(16, g.score());
}
```

```
@Test
public void oneStrike() {
    g.roll(10); // strike
    g.roll(5);
    g.roll(3);
    rollMany(16, 0);
    assertEquals(26, g.score());
}
```

- 또 ugly comment(/* strike)

green phase

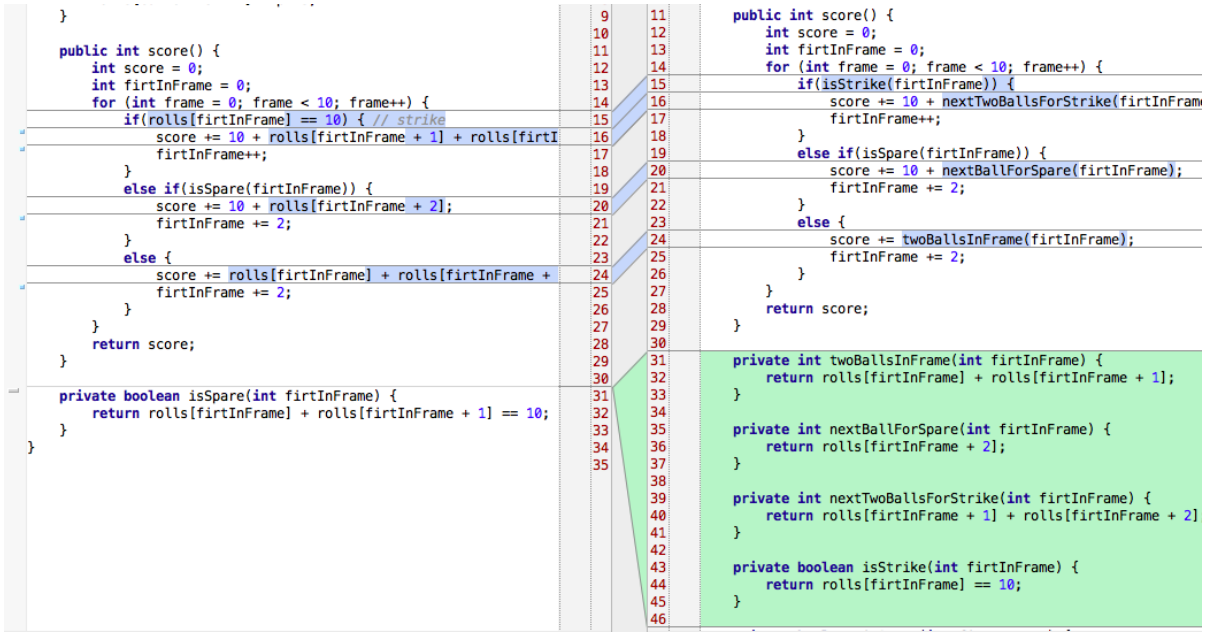


```
public int score() {
    int score = 0;
    int firtInFrame = 0;
    for (int frame = 0; frame < 10; frame++) {
        if(isSpare(firtInFrame)) {
            score += 10 + rolls[firtInFrame + 2];
            firtInFrame += 2;
        }
        else {
            score += rolls[firtInFrame] + rolls[f
            firtInFrame += 2;
        }
    }
    return score;
}

private boolean isSpare(int firtInFrame) {
    return rolls[firtInFrame] + rolls[firtInFrame
}
```

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

- ugly comment(/* strike)

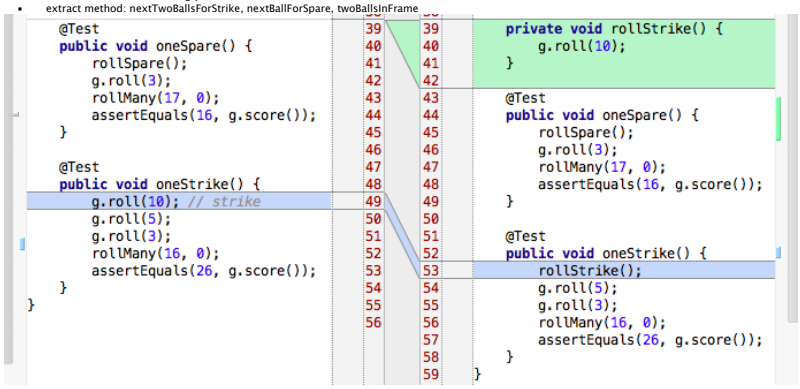
blue phase



```
}

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

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

```
public int score() {
    int score = 0;
    int firtInFrame = 0;
    for (int frame = 0; frame < 10; frame++) {
        if(isStrike(firtInFrame)) {
            score += 10 + nextTwoBallsForStrike(firtInFram
            firtInFrame++;
        }
        else if(isSpare(firtInFrame)) {
            score += 10 + nextBallForSpare(firtInFrame);
            firtInFrame += 2;
        }
        else {
            score += twoBallsInFrame(firtInFrame);
            firtInFrame += 2;
        }
    }
    return score;
}

private int twoBallsInFrame(int firtInFrame) {
    return rolls[firtInFrame] + rolls[firtInFrame + 1];
}

private int nextBallForSpare(int firtInFrame) {
    return rolls[firtInFrame + 2];
}

private int nextTwoBallsForStrike(int firtInFrame) {
    return rolls[firtInFrame + 1] + rolls[firtInFrame + 2]
}

private boolean isStrike(int firtInFrame) {
    return rolls[firtInFrame] == 10;
}
```

- extract method: isStrike로 ugly comment 제거
- extract method: nextTwoBallsForStrike, nextBallForSpare, twoBallsInFrame



```
@Test
public void oneSpare() {
    rollSpare();
    g.roll(3);
    rollMany(17, 0);
    assertEquals(16, g.score());
}

@Test
public void oneStrike() {
    g.roll(10); // strike
    g.roll(5);
    g.roll(3);
    rollMany(16, 0);
    assertEquals(26, g.score());
}
}
```

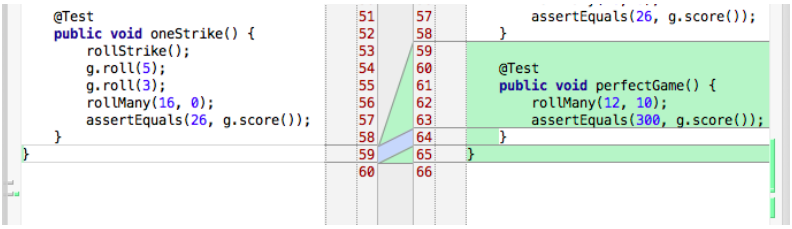
```
private void rollStrike() {
    g.roll(10);
}

@Test
public void oneSpare() {
    rollSpare();
    g.roll(3);
    rollMany(17, 0);
    assertEquals(16, g.score());
}

@Test
public void oneStrike() {
    rollStrike();
    g.roll(5);
    g.roll(3);
    rollMany(16, 0);
    assertEquals(26, g.score());
}
}
```

- extract method: rollStrike로 test에 있는 ugly comment 제거(/* comment)

8th Cycle

red phase

- perfectGame



```
    @Test
    public void oneStrike() {
        rollStrike();
        g.roll(5);
        g.roll(3);
        rollMany(16, 0);
        assertEquals(26, g.score());
    }
}
```

```
        assertEquals(26, g.score());
    }

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

- red phase인데 테스트가 성공한다. 왜 그럴까 ?

```
for (int frame = 0; frame < 10; frame++) {
    if(isStrike(firtInFrame)) {
        score += 10 + nextTwoBallsForStrike(firtInFrame);
        firtInFrame++;
    }
    else if(isSpare(firtInFrame)) {
        score += 10 + nextBallForSpare(firtInFrame);
        firtInFrame += 2;
    }
    else {
        score += twoBallsInFrame(firtInFrame);
        firtInFrame += 2;
    }
}
```

- production code를 살펴보니 3개의 경우 수가 있는데, 실제 볼링 게임에도 그 3가지 경우 수만 존재한다.
- 이런게 알고리즘이다. 알고리즘은 for 루프와 3개의 if-else 문장으로 구성된다.
- 앞서 UML로 표현된 클래스 다이어그램을 보라. 이렇게 해야 할까 ? 당신은 Game 클래스를, 또 당신은 Frame 클래스를, … 그리고 다음주 화요일에 만나서 통합해 보자. 한 400라인은 될 코드로. 고작 14줄이면 될 것을.
- 늘 이런 것은 아니다. 하지만 자주 발생한다. 설계하고 TDD로 진행하다보면 다를 수 있다.
- 설계를 따라 개발을 시도한 적이 있다.
    - Roll 클래스에 대해서 테스트를 작성하고자 했다. 하지만 테스트가 존재하지 않는 것을 발견했다. 테스트할 행위가 없었다.
    - 그래서 Frame 클래스에 대해서 테스트를 작성하고자 했다. Frame 클래스에도 테스트할 행위가 없었다.
    - 다시 물러서서 Game 클래스를 살피는 작성할 테스트가 있었다. 이 클래스에 대해서 하나씩 테스트를 작성하기 시작했다. 점진적으로 2개의 if 문장과 for 루프 문장이 드러나기 시작했다.
    - 그러다 보니 이 알고리즘이 볼링 규칙처럼 보였다.
    - **냅킨에 그렸던 설계는 달성되지 않았지만 보다 훨씬 간단한 알고리즘이 나타났다.**