Test-Driven Development Fundamentals on Force.com: Step-by-Step Bowling Kata

Developer Track

Kyle Thornton, Mavens Consulting @knthornt
Sean Harrison, Mavens Consulting @GizmoForce

Safe Harbor

Safe harbor statement under the Private Securities Litigation Reform Act of 1995:

This presentation may contain forward-looking statements that involve risks, uncertainties, and assumptions. If any such uncertainties materialize or if any of the assumptions proves incorrect, the results of salesforce.com, inc. could differ materially from the results expressed or implied by the forward-looking statements we make. All statements other than statements of historical fact could be deemed forward-looking, including any projections of product or service availability, subscriber growth, earnings, revenues, or other financial items and any statements regarding strategies or plans of management for future operations, statements of belief, any statements concerning new, planned, or upgraded services or technology developments and customer contracts or use of our services.

The risks and uncertainties referred to above include – but are not limited to – risks associated with developing and delivering new functionality for our service, new products and services, our new business model, our past operating losses, possible fluctuations in our operating results and rate of growth, interruptions or delays in our Web hosting, breach of our security measures, the outcome of intellectual property and other litigation, risks associated with possible mergers and acquisitions, the immature market in which we operate, our relatively limited operating history, our ability to expand, retain, and motivate our employees and manage our growth, new releases of our service and successful customer deployment, our limited history reselling non-salesforce.com products, and utilization and selling to larger enterprise customers. Further information on potential factors that could affect the financial results of salesforce.com, inc. is included in our annual report on Form 10-Q for the most recent fiscal quarter ended July 31, 2012. This documents and others containing important disclosures are available on the SEC Filings section of the Investor Information section of our Web site.

Any unreleased services or features referenced in this or other presentations, press releases or public statements are not currently available and may not be delivered on time or at all. Customers who purchase our services should make the purchase decisions based upon features that are currently available. Salesforce.com, inc. assumes no obligation and does not intend to update these forward-looking statements.

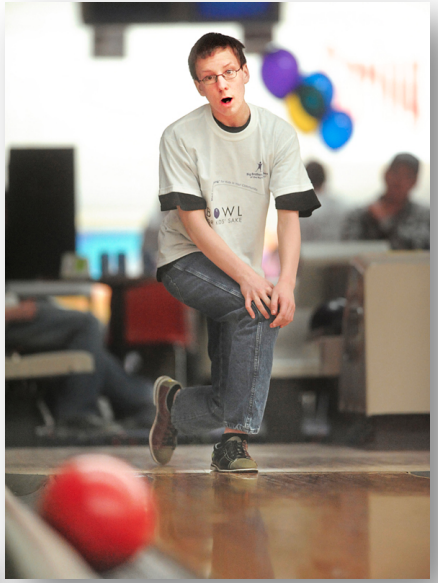
Photo by Metro-Goldwyn-Mayer (MGM) – © 1996 - MGM/UA

Test-Driven Development

Write your tests before your write your code

Learn by doing

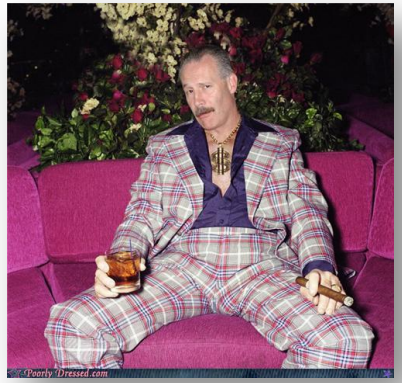
Kata = step-by-step example



Test-Driven Development

Classic Bowling Example

- 3 of the 5 tests today
- 4 and 5 online
- Provide code for you to finish



- Credit where credit is due
  - 2005 version of kata
  - ObjectMentor.com

This is not ‘Nam. This is Bowling. There are Rules.

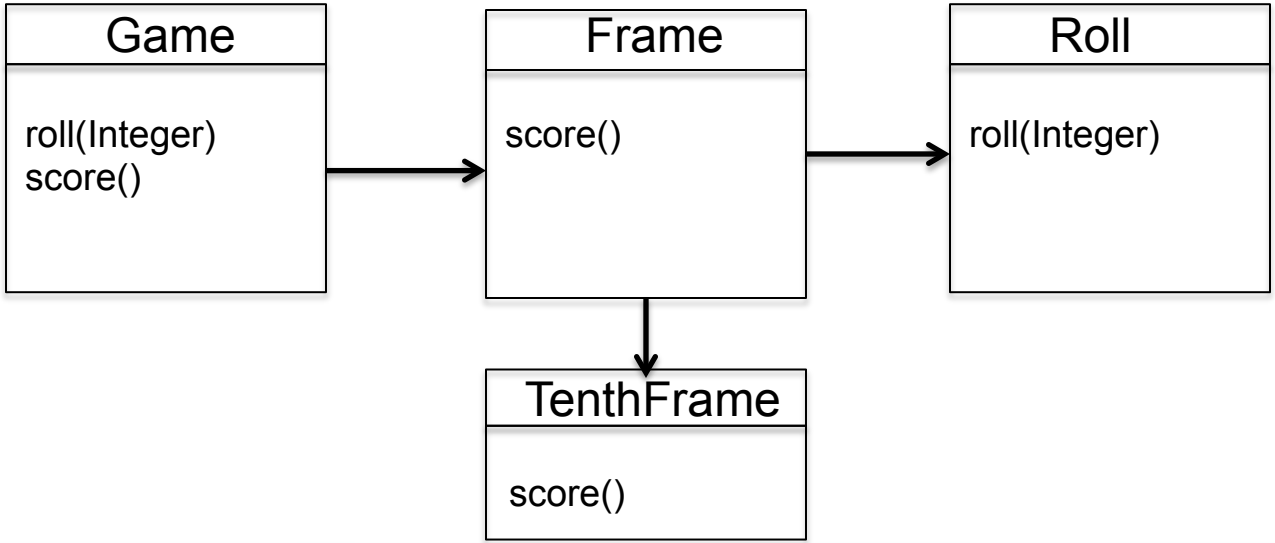
1		2		3		4		5		6		7		8		9		10	
3	2	8	/	3	2	X	-	3	2	5	3	6	/	7	1	2	4	7	/	6
5		18		23		38		43		51		68		76		82		98	

10 Frames / 2 rolls each
Each frame has a score
Frame score = pins down with rolls + bonuses

Bonuses:
- Spare = 2 Rolls + 1 bonus roll
- Strike = 1 Roll + 2 bonus rolls

10th Frame
- rolls as needed
- max rolls = 3

1		2		3		4		5		6		7		8		9		10		
3	2	8	2	3	2	10	X	3	2	5	3	6	/	7	1	2	4	7	/	6
5		20		25		40		45		53		70		78		84		100		



Assumptions

1. We are only calculating the score
2. The class we build will be for one bowler
3. We are not responsible for the input of the scores in this class
4. There would be a complimentary class to handle the score input

Let’s Define Our Use Cases

1. Score should be 0 if bowler rolls a complete gutter game
2. Score should be 20 if bowler knocks down 1 pin every roll
3. Score calculates properly if bowler rolls a spare
4. Score calculates properly if bowler rolls a strike
5. Score is 300 if bowler rolls a perfect game

The First Test

```
@isTest
private class BowlingGameTest {

    static testMethod void testGutterGame() {
        Game g = new Game();
    }
}
```

The First Test

```
@isTest
private class BowlingGameTest {

    static testMethod void testGutterGame() {
        Game g = new Game();
    }
}
```

```
public with sharing class Game {

}
```

The First Test

```
@isTest
private class BowlingGameTest {

    static testMethod void testGutterGame() {
        Game g = new Game();
    }
}
```

```
public with sharing class Game {

}
```

The First Test

```
@isTest
private class BowlingGameTest {

    static testMethod void testGutterGame() {
        Game g = new Game();
    }
}
```

```
public with sharing class Game {

}
```

The First Test

```
@isTest
private class BowlingGameTest {

    static testMethod void testGutterGame() {
        Game g = new Game();
        for (Integer i=0; i<20; i++ ) {
            g.roll(0);
        }
    }
}
```

```
public with sharing class Game {

}
```

The First Test

```
@isTest
private class BowlingGameTest {

    static testMethod void testGutterGame() {
        Game g = new Game();
        for (Integer i=0; i<20; i++ ) {
            g.roll(0);
        }
    }
}
```

```
public with sharing class Game {
    public void roll(Integer pins) {
    }
}
```

The First Test

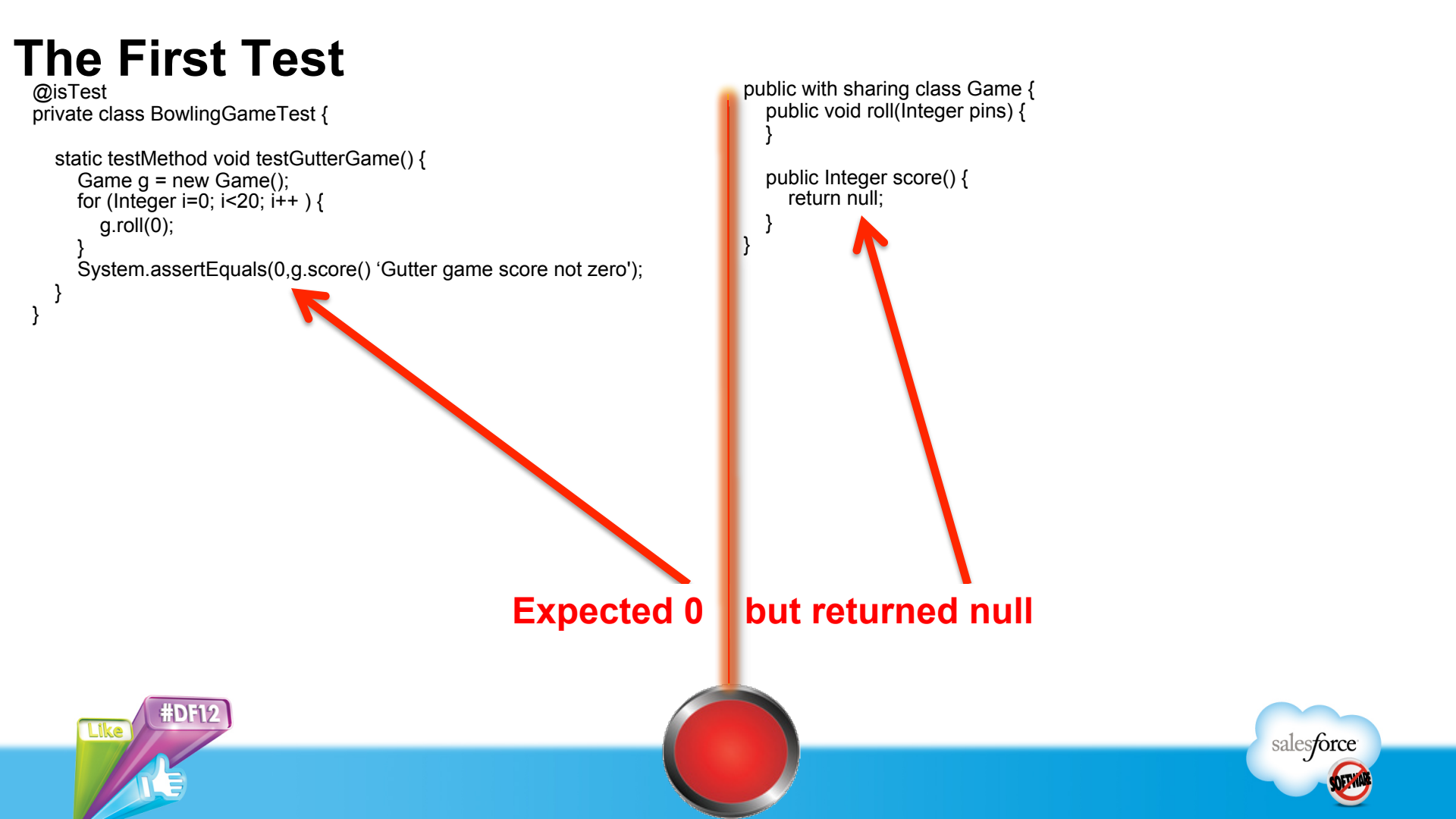
```
@isTest
private class BowlingGameTest {

    static testMethod void testGutterGame() {
        Game g = new Game();
        for (Integer i=0; i<20; i++ ) {
            g.roll(0);
        }
        System.assertEquals(0,g.score(), 'Gutter game score not zero');
    }
}
```

```
public with sharing class Game {
    public void roll(Integer pins) {
    }
}
```

The First Test

```
@isTest
private class BowlingGameTest {

    static testMethod void testGutterGame() {
        Game g = new Game();
        for (Integer i=0; i<20; i++ ) {
            g.roll(0);
        }
        System.assertEquals(0,g.score() ‘Gutter game score not zero');
    }
}
```

```
public with sharing class Game {
    public void roll(Integer pins) {
    }

    public Integer score() {
        return null;
    }
}
```

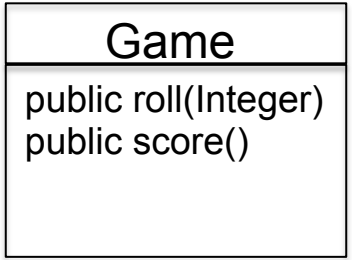


The First Test

```
@isTest
private class BowlingGameTest {

    static testMethod void testGutterGame() {
        Game g = new Game();
        for (Integer i=0; i<20; i++ ) {
            g.roll(0);
        }
        System.assertEquals(0,g.score(), ‘Gutter game score not zero');
    }
}
```

```
public with sharing class Game {
    public void roll(Integer pins) {
    }

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

Unit Test #1 - Score should be 0 if bowler rolls a complete gutter game



Next Use Case:

Score should be 20 if bowler knocks down 1 pin every roll

The Second Test

```
@isTest
private class BowlingGameTest {

    static testMethod void testGutterGame() {
        Game g = new Game();
        for (Integer i=0; i<20; i++ ) {
            g.roll(0);
        }
        System.assertEquals(0,g.score(), ‘Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        Game g = new Game();
        for (Integer i=0; i<20; i++ ) {
            g.roll(1);
        }
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }
}
```

```
public with sharing class Game {
    public void roll(Integer pins) {
    }

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

The Second Test

- Game creation is duplicated
- Roll loop is duplicated

```
@isTest
private class BowlingGameTest {

    static testMethod void testGutterGame() {
        Game g = new Game();
        for (Integer i=0; i<20; i++ ) {
            g.roll(0);
        }
        System.assertEquals(0,g.score(), ‘Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        Game g = new Game();
        for (Integer i=0; i<20; i++ ) {
            g.roll(1);
        }
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }
}
```

```
public with sharing class Game {
    public void roll(Integer pins) {
    }

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

The Second Test

- Game creation is duplicated
- Roll loop is duplicated

```
@isTest
private class BowlingGameTest {

    static testMethod void testGutterGame() {
        Game g = new Game();
        for (Integer i=0; i<20; i++ ) {
            g.roll(0);
        }
        System.assertEquals(0,g.score(), ‘Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        Game g = new Game();
        for (Integer i=0; i<20; i++ ) {
            g.roll(1);
        }
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }
}
```

```
public with sharing class Game {
    public void roll(Integer pins) {
    }

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

Expected 20 but returned 0

The Second Test

- Game creation is duplicated
- Roll loop is duplicated

```
@isTest
private class BowlingGameTest {

    static testMethod void testGutterGame() {
        Game g = new Game();
        for (Integer i=0; i<20; i++ ) {
            g.roll(0);
        }
        System.assertEquals(0,g.score(), ‘Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        Game g = new Game();
        for (Integer i=0; i<20; i++ ) {
            g.roll(1);
        }
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }
}
```

```
public with sharing class Game {
    private Integer score = 0;

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

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

Let's Do Some Clean-up

```
@isTest
private class BowlingGameTest {

    static testMethod void testGutterGame() {
        Game g = new Game();
        for (Integer i=0; i<20; i++ ) {
            g.roll(0);
        }
        System.assertEquals(0,g.score(), 'Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        Game g = new Game();
        for (Integer i=0; i<20; i++ ) {
            g.roll(1);
        }
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }
}
```

- Game creation is duplicated
- Roll loop is duplicated

```
public with sharing class Game {
    private Integer score = 0;

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

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

Let's Do Some Clean-up

- ~~Game creation is duplicated~~
- Roll loop is duplicated

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();

    static testMethod void testGutterGame() {
        for (Integer i=0; i<20; i++ ) {
            g.roll(0);
        }
        System.assertEquals(0,g.score(), 'Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        for (Integer i=0; i<20; i++ ) {
            g.roll(1);
        }
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }
}
```

```
public with sharing class Game {
    private Integer score = 0;

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

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

Let's Do Some Clean-up

- Roll loop is duplicated

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();

    static testMethod void testGutterGame() {
        Integer n = 20;
        Integer pins = 0;
        for (Integer i=0; i<n; i++ ) {
            g.roll(pins);
        }
        System.assertEquals(0,g.score(), 'Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        for (Integer i=0; i<20; i++ ) {
            g.roll(1);
        }
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }
}
```

```
public with sharing class Game {
    private Integer score = 0;

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

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

Let's Do Some Clean-up

- Roll loop is duplicated

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();

    private static void rollMany(Integer n, Integer pins) {
        for (Integer i=0; i<n; i++) {
            g.roll(pins);
        }
    }

    static testMethod void testGutterGame() {
        Integer n = 20;
        Integer pins = 0;
        rollMany(n, pins);
        System.assertEquals(0,g.score(), 'Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        for (Integer i=0; i<20; i++ ) {
            g.roll(1);
        }
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }
}
```

```
public with sharing class Game {
    private Integer score = 0;

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

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

Let's Do Some Clean-up

- Roll loop is duplicated

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();

    private static void rollMany(Integer n, Integer pins) {
        for (Integer i=0; i<n; i++) {
            g.roll(pins);
        }
    }

    static testMethod void testGutterGame() {
        rollMany(20, 0);
        System.assertEquals(0,g.score(), 'Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        for (Integer i=0; i<20; i++ ) {
            g.roll(1);
        }
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }
}
```

```
public with sharing class Game {
    private Integer score = 0;

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

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

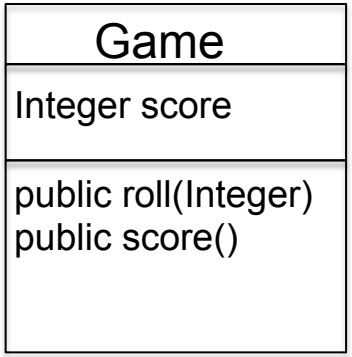
Let's Do Some Clean-up

- ~~Roll loop is duplicated~~

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();

    private static void rollMany(Integer n, Integer pins) {
        for (Integer i=0; i<n; i++) {
            g.roll(pins);
        }
    }

    static testMethod void testGutterGame() {
        rollMany(20, 0);
        System.assertEquals(0,g.score(), 'Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }
}
```

```
public with sharing class Game {
    private Integer score = 0;

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

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

Unit Test #2 - Score should be 20 if bowler knocks down 1 pin every roll



Next Use Case:

Score calculates properly if bowler rolls a spare

The Third Test

- Ugly comment in test

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();

    private static void rollMany(Integer n, Integer pins) {
        for (Integer i=0; i<n; i++) {
            g.roll(pins);
        }
    }

    static testMethod void testGutterGame() {
        rollMany(20, 0);
        System.assertEquals(0,g.score(), 'Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }

    static testMethod void testOneSpare() {
        g.roll(5);
        g.roll(5);  //spare
        g.roll(3);
        rollMany(17,0);
        System.assertEquals(16,g.score(), 'Spare not scored correctly');
    }
}
```

Ugly comment

```
public with sharing class Game {
    private Integer score = 0;

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

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

The Third Test

- Ugly comment in test

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();

    private static void rollMany(Integer n, Integer pins) {
        for (Integer i=0; i<n; i++) {
            g.roll(pins);
        }
    }

    static testMethod void testGutterGame() {
        rollMany(20, 0);
        System.assertEquals(0,g.score(), 'Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }

    static testMethod void testOneSpare() {
        g.roll(5);
        g.roll(5);  //spare
        g.roll(3);
        rollMany(17,0);
        System.assertEquals(16,g.score(), 'Spare not scored correctly');
    }
}
```

```
public with sharing class Game {
    private Integer score = 0;

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

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

Expected 16 but returned 13

The Third Test

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();

    private static void rollMany(Integer n, Integer pins) {
        for (Integer i=0; i<n; i++) {
            g.roll(pins);
        }
    }

    static testMethod void testGutterGame() {
        rollMany(20, 0);
        System.assertEquals(0,g.score(), 'Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }

    static testMethod void testOneSpare() {
        g.roll(5);
        g.roll(5);  //spare
        g.roll(3);
        rollMany(17,0);
        System.assertEquals(16,g.score(), 'Spare not scored correctly');
    }
}
```

- Ugly comment in test

```
public with sharing class Game {
    private Integer score = 0;

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

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

Tempted to use flag to Remember previous roll. Design must be wrong.

Roll() calculates score

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

Design is not going in the right direction.
-Methods doing the wrong job
-Going down a design path we would rather avoid

Let's comment out testOneSpare and do some rework

The Third Test

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();

    private static void rollMany(Integer n, Integer pins) {
        for (Integer i=0; i<n; i++) {
            g.roll(pins);
        }
    }

    static testMethod void testGutterGame() {
        rollMany(20, 0);
        System.assertEquals(0,g.score(), ‘Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }

    // static testMethod void testOneSpare() {
    //     g.roll(5);
    //     g.roll(5);  //spare
    //     g.roll(3);
    //     rollMany(17,0);
    //    System.assertEquals(16,g.score(), ‘Spare not scored correctly’);
    //}
}
```

```
public with sharing class Game {
    private Integer score = 0;

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

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

- Ugly comment in test

The Third Test

- Ugly comment in test

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();

    private static void rollMany(Integer n, Integer pins) {
        for (Integer i=0; i<n; i++) {
            g.roll(pins);
        }
    }

    static testMethod void testGutterGame() {
        rollMany(20, 0);
        System.assertEquals(0,g.score(), 'Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }

    // static testMethod void testOneSpare() {
    //     g.roll(5);
    //     g.roll(5);  //spare
    //     g.roll(3);
    //     rollMany(17,0);
    //    System.assertEquals(16,g.score(), 'Spare not scored correctly');
    //}
}
```

```
public with sharing class Game {
    private Integer score = 0;
    private list<Integer> rolls =
                        new  list<Integer>{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    private Integer currentRoll = 0;

    public void roll(Integer pins) {
        score += pins;
        rolls[curentRoll++] = pins;
    }

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

The Third Test

- Ugly comment in test

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();

    private static void rollMany(Integer n, Integer pins) {
        for (Integer i=0; i<n; i++) {
            g.roll(pins);
        }
    }

    static testMethod void testGutterGame() {
        rollMany(20, 0);
        System.assertEquals(0,g.score(), 'Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }

    // static testMethod void testOneSpare() {
    //     g.roll(5);
    //     g.roll(5);  //spare
    //     g.roll(3);
    //     rollMany(17,0);
    //    System.assertEquals(16,g.score(), 'Spare not scored correctly');
    //}
}
```

```
public with sharing class Game {
    private Integer score = 0;
    private list<Integer> rolls =
                        new  list<Integer>{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    private Integer currentRoll = 0;

    public void roll(Integer pins) {
        score += pins;
        rolls[curentRoll++] = pins;
    }

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

The Third Test

- Ugly comment in test

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();

    private static void rollMany(Integer n, Integer pins) {
        for (Integer i=0; i<n; i++) {
            g.roll(pins);
        }
    }

    static testMethod void testGutterGame() {
        rollMany(20, 0);
        System.assertEquals(0,g.score(), ‘Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }

    // static testMethod void testOneSpare() {
    //     g.roll(5);
    //     g.roll(5);  //spare
    //     g.roll(3);
    //     rollMany(17,0);
    //    System.assertEquals(16,g.score(), ‘Spare not scored correctly’);
    //}
}
```

```
public with sharing class Game {
    private list<Integer> rolls =
                        new  list<Integer>{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    private Integer currentRoll = 0;

    public void roll(Integer pins) {
        rolls[curentRoll++] = pins;
    }

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

The Third Test

- Ugly comment in test

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();

    private static void rollMany(Integer n, Integer pins) {
        for (Integer i=0; i<n; i++) {
            g.roll(pins);
        }
    }

    static testMethod void testGutterGame() {
        rollMany(20, 0);
        System.assertEquals(0,g.score(), 'Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }

    static testMethod void testOneSpare() {
        g.roll(5);
        g.roll(5);  //spare
        g.roll(3);
        rollMany(17,0);
        System.assertEquals(16,g.score(), 'Spare not scored correctly');
    }
}
```

```
public with sharing class Game {
    private list<Integer> rolls =
                        new  list<Integer>{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    private Integer currentRoll = 0;

    public void roll(Integer pins) {
        rolls[curentRoll++] = pins;
    }

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

Expected 16 but returned 13

The Third Test

- Ugly comment in test

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();

    private static void rollMany(Integer n, Integer pins) {
        for (Integer i=0; i<n; i++) {
            g.roll(pins);
        }
    }

    static testMethod void testGutterGame() {
        rollMany(20, 0);
        System.assertEquals(0,g.score(), ‘Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }

    static testMethod void testOneSpare() {
        g.roll(5);
        g.roll(5);  //spare
        g.roll(3);
        rollMany(17,0);
        System.assertEquals(16,g.score(), ‘Spare not scored correctly’);
    }
}
```

```
public with sharing class Game {
    private list<Integer> rolls =
                        new  list<Integer>{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    private Integer currentRoll = 0;

    public void roll(Integer pins) {
        rolls[curentRoll++] = pins;
    }

    public Integer score() {
        Integer score = 0;
        for (Integer 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 *one frame* (two balls) at a time.

The Third Test

- Ugly comment in test

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();

    private static void rollMany(Integer n, Integer pins) {
        for (Integer i=0; i<n; i++) {
            g.roll(pins);
        }
    }

    static testMethod void testGutterGame() {
        rollMany(20, 0);
        System.assertEquals(0,g.score(), ‘Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }

    // static testMethod void testOneSpare() {
    //     g.roll(5);
    //     g.roll(5);  //spare
    //     g.roll(3);
    //     rollMany(17,0);
    //    System.assertEquals(16,g.score(), ‘Spare not scored correctly’);
    //}
}
```

```
public with sharing class Game {
    private list<Integer> rolls =
                        new  list<Integer>{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    private Integer currentRoll = 0;

    public void roll(Integer pins) {
        rolls[curentRoll++] = pins;
    }

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

The Third Test

- Ugly comment in test

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();

    private static void rollMany(Integer n, Integer pins) {
        for (Integer i=0; i<n; i++) {
            g.roll(pins);
        }
    }

    static testMethod void testGutterGame() {
        rollMany(20, 0);
        System.assertEquals(0,g.score(), ‘Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }

    static testMethod void testOneSpare() {
        g.roll(5);
        g.roll(5);  //spare
        g.roll(3);
        rollMany(17,0);
        System.assertEquals(16,g.score(), ‘Spare not scored correctly’);
    }
}
```

```
public with sharing class Game {
    private list<Integer> rolls =
                        new  list<Integer>{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    private Integer currentRoll = 0;

    public void roll(Integer pins) {
        rolls[curentRoll++] = pins;
    }

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

Expected 16 but returned 13

The Third Test

- Ugly comment in test

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();

    private static void rollMany(Integer n, Integer pins) {
        for (Integer i=0; i<n; i++) {
            g.roll(pins);
        }
    }

    static testMethod void testGutterGame() {
        rollMany(20, 0);
        System.assertEquals(0,g.score(), ‘Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }

    static testMethod void testOneSpare() {
        g.roll(5);
        g.roll(5);  //spare
        g.roll(3);
        rollMany(17,0);
        System.assertEquals(16,g.score(), ‘Spare not scored correctly’);
    }
}
```

```
public with sharing class Game {
    private list<Integer> rolls =
                        new  list<Integer>{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    private Integer currentRoll = 0;

    public void roll(Integer pins) {
        rolls[curentRoll++] = pins;
    }

    public Integer score() {
        Integer score = 0;
        Integer i = 0;
        for (Integer 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;
    }
}
```

Some More Clean-up

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();

    private static void rollMany(Integer n, Integer pins) {
        for (Integer i=0; i<n; i++) {
            g.roll(pins);
        }
    }

    static testMethod void testGutterGame() {
        rollMany(20, 0);
        System.assertEquals(0,g.score(), ‘Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }

    static testMethod void testOneSpare() {
        g.roll(5);
        g.roll(5);  //spare
        g.roll(3);
        rollMany(17,0);
        System.assertEquals(16,g.score(), ‘Spare not scored correctly’);
    }
}
```

- Ugly comment in test
- Ugly comment in conditional
- i is a bad variable name

```
public with sharing class Game {
    private list<Integer> rolls =
                        new  list<Integer>{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    private Integer currentRoll = 0;

    public void roll(Integer pins) {
        rolls[curentRoll++] = pins;
    }

    public Integer score() {
        Integer score = 0:
        Integer i = 0;
        for (Integer 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;
    }
}
```

Bad variable name

Ugly Comment

Some More Clean-up

- Ugly comment in test
- Ugly comment in conditional

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();

    private static void rollMany(Integer n, Integer pins) {
        for (Integer i=0; i<n; i++) {
            g.roll(pins);
        }
    }

    static testMethod void testGutterGame() {
        rollMany(20, 0);
        System.assertEquals(0,g.score(), ‘Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }

    static testMethod void testOneSpare() {
        g.roll(5);
        g.roll(5);  //spare
        g.roll(3);
        rollMany(17,0);
        System.assertEquals(16,g.score(), ‘Spare not scored correctly’);
    }
}
```

```
public with sharing class Game {
    private list<Integer> rolls =
                        new  list<Integer>{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    private Integer currentRoll = 0;

    public void roll(Integer pins) {
        rolls[curentRoll++] = pins;
    }

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

Some More Clean-up

- Ugly comment in test

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();

    private static void rollMany(Integer n, Integer pins) {
        for (Integer i=0; i<n; i++) {
            g.roll(pins);
        }
    }

    static testMethod void testGutterGame() {
        rollMany(20, 0);
        System.assertEquals(0,g.score(), ‘Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(), 'An all ones game score not 20');
    }

    static testMethod void testOneSpare() {
        g.roll(5);
        g.roll(5);  //spare
        g.roll(3);
        rollMany(17,0);
        System.assertEquals(16,g.score(), ‘Spare not scored correctly’);
    }
}
```

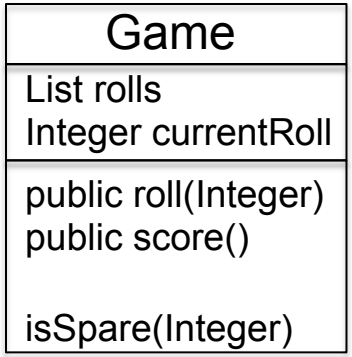
```
public with sharing class Game {
    private list<Integer> rolls =
                        new  list<Integer>{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    private Integer currentRoll = 0;

    public void roll(Integer pins) {
        rolls[curentRoll++] = pins;
    }

    public Integer score() {
        Integer score = 0;
        Integer rollsIndex = 0;
        for (Integer frame=0, frame<10; frame++) {
            if ( isSpare(rollsIndex)) {
                score += 10 +  rolls[i+2];
                rollsIndex += 2;
            } else {
                score += rolls[rollsIndex] + rolls[rollsIndex+1];
                rollsIndex+=2;
            }
        }
        return score;
    }

    private Boolean isSpare(Integer rollsIndex) {
        return rolls[rollsIndex] + rolls[rollsIndex+1] == 10;
    }
```

Some More Clean-up

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();
    ……..

    static testMethod void testGutterGame() {
        rollMany(20, 0);
        System.assertEquals(0,g.score(), ‘Gutter game score not zero');
    }

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(),  'An all ones game score not 20');
    }

    static testMethod void testOneSpare() {
        rollSpare();
        g.roll(3);
        rollMany(17,0);
       System.assertEquals(16,g.score(), ‘Spare not scored correctly’);
    }

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

```
public with sharing class Game {
    private list<Integer> rolls =
                        new  list<Integer>{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    private Integer currentRoll = 0;

    public void roll(Integer pins) {
        rolls[curentRoll++] = pins;
    }

    public Integer score() {
        Integer score = 0;
        Integer rollsIndex = 0;
        for (Integer frame=0, frame<10; frame++) {
            if ( isSpare(rollsIndex)) {
                score += 10 +  rolls[i+2];
                rollsIndex += 2;
            } else {
                score += rolls[rollsIndex] + rolls[rollsIndex+1];
                rollsIndex+=2;
            }
        }
        return score;
    }

    private Boolean isSpare(Integer rollsIndex) {
        return rolls[rollsIndex] + rolls[rollsIndex+1] == 10;
    }
```

Unit Test #3 – Score calculates properly if bowler rolls a spare



Next Use Case:

Score calculates properly if bowler rolls a strike

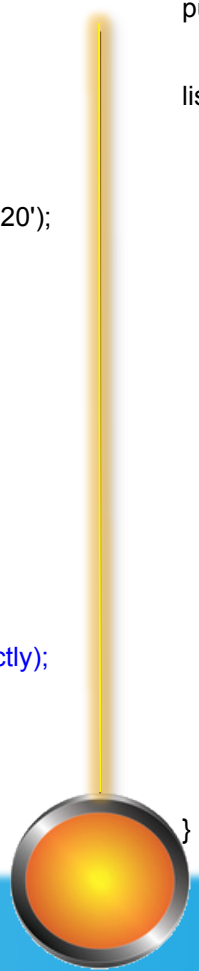
The Fourth Test

- Ugly comment in test

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();
    ……..

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(),  'An all ones game score not 20');
    }

    static testMethod void testOneSpare() {
        rollSpare();
        g.roll(3);
        rollMany(17,0);
       System.assertEquals(16,g.score(), 'Spare not scored correctly');
    }

    public testMethod void testOneStrike() {
        g.roll(10); //strike
        g.roll(3);
        g.roll(4);
        rollMany(16, 0);
        System.assertEquals(24, g.score(), 'One strike not scored correctly);
    }

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

Ugly Comment

```
public with sharing class Game {
    private list<Integer> rolls =
                        new
list<Integer>{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    private Integer currentRoll = 0;

    public void roll(Integer pins) {
        rolls[curentRoll++] = pins;
    }

    public Integer score() {
        Integer score = 0;
        Integer rollsIndex = 0;
        for (Integer frame=0, frame<10; frame++) {
            if ( isSpare(rollsIndex)) {
                score += 10 +  rolls[i+2];
                rollsIndex += 2;
            } else {
                score += rolls[rollsIndex] + rolls[rollsIndex+1];
                rollsIndex+=2;
            }
        }
        return score;
    }

    private Boolean isSpare(Integer rollsIndex) {
        return rolls[rollsIndex] + rolls[rollsIndex+1] == 10;
    }
}
```



The Fourth Test

- Ugly comment in test
- Ugly comment in conditional
- Ugly expressions

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();
    ……..

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(),  'An all ones game score not 20');
    }

    static testMethod void testOneSpare() {
        rollSpare();
        g.roll(3);
        rollMany(17,0);
       System.assertEquals(16,g.score(), 'Spare not scored correctly');
    }

    public testMethod void testOneStrike() {
        g.roll(10); //strike
        g.roll(3);
        g.roll(4);
        rollMany(16, 0);
        System.assertEquals(24, g.score(), 'One strike not scored correctly);
    }

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

```
public with sharing class Game {
    private list<Integer> rolls =
                        new  list<Integer>{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    private Integer currentRoll = 0;
    ……
    public Integer score() {
        Integer score = 0;
        Integer rollsIndex = 0;
        for (Integer frame=0, frame<10; frame++) {
            if (rolls[rollsIndex] == 10) { //strike
                score += 10 + rolls[rollsIndex+1] + rolls[rollsIndex+2];
                rollsIndex++;
            } else if ( isSpare(rollsIndex)) {
                score += 10 +  rolls[i+2];
                rollsIndex += 2;
            } else {
                score += rolls[rollsIndex] + rolls[rollsIndex+1];
                rollsIndex+=2;
            }
        }
        return score;
    }

    private Boolean isSpare(Integer rollsIndex) {
        return rolls[rollsIndex] + rolls[rollsIndex+1] == 10;
    }
}
```

Ugly Comment

Ugly Expressions

Ugly Expression

The Fourth Test

- Ugly comment in test
- Ugly comment in conditional
- Ugly expressions

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();
    ……..

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(),  'An all ones game score not 20');
    }

    static testMethod void testOneSpare() {
        rollSpare();
        g.roll(3);
        rollMany(17,0);
       System.assertEquals(16,g.score(), 'Spare not scored correctly');
    }

    public testMethod void testOneStrike() {
        g.roll(10); //strike
        g.roll(3);
        g.roll(4);
        rollMany(16, 0);
        System.assertEquals(24, g.score(), 'One strike not scored correctly);
    }

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

```
public with sharing class Game {
    private list<Integer> rolls =
                        new  list<Integer>{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    private Integer currentRoll = 0;
    ……
    public Integer score() {
        Integer score = 0;
        Integer rollsIndex = 0;
        for (Integer frame=0, frame<10; frame++) {
            if (rolls[rollsIndex] == 10) { //strike
                score += 10 + rolls[rollsIndex+1] + rolls[rollsIndex+2];
                rollsIndex++;
            } else if ( isSpare(rollsIndex)) {
                score += 10 +  rolls[i+2];
                rollsIndex += 2;
            } else {
                score += rolls[rollsIndex] + rolls[rollsIndex+1];
                rollsIndex+=2;
            }
        }
        return score;
    }

    private Boolean isSpare(Integer rollsIndex) {
        return rolls[rollsIndex] + rolls[rollsIndex+1] == 10;
    }
}
```

The Fourth Test

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();
    ……..

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(),  'An all ones game score not 20');
    }

    static testMethod void testOneSpare() {
        rollSpare();
        g.roll(3);
        rollMany(17,0);
       System.assertEquals(16,g.score(), 'Spare not scored correctly');
    }

    public testMethod void testOneStrike() {
        g.roll(10); //strike
        g.roll(3);
        g.roll(4);
        rollMany(16, 0);
        System.assertEquals(24, g.score(), 'One strike not scored correctly);
    }

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


```
public with sharing class Game {
……..
    public Integer score() {
        Integer score = 0;
        Integer rollsIndex = 0;
        for (Integer frame=0, frame<10; frame++) {
            if (rolls[rollsIndex] == 10) { //strike
                score += 10 + strikeBonus(rollsIndex);
                rollsIndex++;
            } else if ( isSpare(rollsIndex)) {
                score += 10 +  spareBonus(rollsIndex);
                rollsIndex += 2;
            } else {
                score += sumOfBallsInFrame(rollsIndex);
                rollsIndex+=2;
            }
        }
        return score;
    }
    private Integer sumOfBallsInFrame(Integer rollsIndex) {
        return rolls[rollsIndex] + rolls[rollsIndex+1];
    }
    private Integer spareBounus(Integer rollsIndex) {
        return rolls[rollsIndex+2];
    }
    private Integer strikeBounus(Integer rollsIndex) {
        return rolls[rollsIndex+1] + rolls[rollsIndex+2];
    }
    private Boolean isSpare(Integer rollsIndex) {
        return rolls[rollsIndex] + rolls[rollsIndex] == 10;
    }
```

- Ugly comment in test
- Ugly comment in conditional

The Fourth Test

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();
    ……..

    static testMethod void testAllOnes() {
        rollMany(20, 1);
        System.assertEquals(20,g.score(),  'An all ones game score not 20');
    }

    static testMethod void testOneSpare() {
        rollSpare();
        g.roll(3);
        rollMany(17,0);
       System.assertEquals(16,g.score(), 'Spare not scored correctly');
    }

    public testMethod void testOneStrike() {
        g.roll(10); //strike
        g.roll(3);
        g.roll(4);
        rollMany(16, 0);
        System.assertEquals(24, g.score(), 'One strike not scored correctly);
    }

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


```
public with sharing class Game {
  ……..
    public Integer score() {
        Integer score = 0;
        Integer rollsIndex = 0;
        for (Integer frame=0, frame<10; frame++) {
            if (isStrike(rollsIndex)) {
                score += 10 + strikeBonus(rollsIndex);
                rollsIndex++;
            } else if ( isSpare(rollsIndex)) {
                score += 10 +  spareBonus(rollsIndex);
                rollsIndex += 2;
            } else {
                score += sumOfBallsInFrame(rollsIndex);
                rollsIndex+=2;
            }
        }
        return score;
    }

    private Boolean isStrike(Integer rollsIndex) {
        return rolls[rollsIndex] == 10;
    }
    private Integer sumOfBallsInFrame(Integer rollsIndex) {
        return rolls[rollsIndex] + rolls[rollsIndex+1];
    }
    private Integer spareBounus(Integer rollsIndex) {
        return rolls[rollsIndex+2];
    }
    private Integer strikeBounus(Integer rollsIndex) {
        return rolls[rollsIndex+1] + rolls[rollsIndex+2];
    }
    private Boolean isSpare(Integer rollsIndex) {
        return rolls[rollsIndex] + rolls[rollsIndex+1] == 10;
    }
}
```

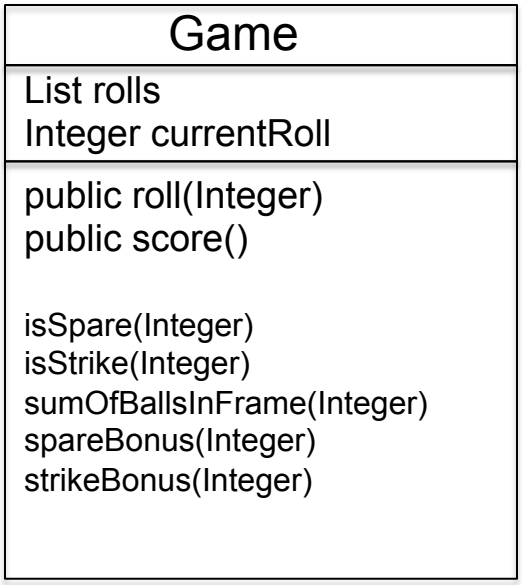
- Ugly comment in test

The Fourth Test

```
@isTest
private class BowlingGameTest {
    private static Game g = new Game();
    ……..
    static testMethod void testOneSpare() {
        rollSpare();
        g.roll(3);
        rollMany(17,0);
       System.assertEquals(16,g.score(), 'Spare not scored correctly');
    }

    public testMethod void testOneStrike() {
        rollStrike(10);
        g.roll(3);
        g.roll(4);
        rollMany(16, 0);
        System.assertEquals(24, g.score(), 'One strike not scored correctly);
    }

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

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

```
public with sharing class Game {
    private list<Integer> rolls =
                    new  list<Integer>{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    private Integer currentRoll = 0;

    public void roll(Integer pins) {
        rolls[curentRoll++] = pins;
    }

    public Integer score() {
        Integer score = 0;
        Integer rollsIndex = 0;
        for (Integer frame=0, frame<10; frame++) {
            if (isStrike(rollsIndex)) {
                score += 10 + strikeBonus(rollsIndex);
                rollsIndex++;
            } else if ( isSpare(rollsIndex)) {
                score += 10 +  spareBonus(rollsIndex);
                rollsIndex += 2;
            } else {
                score += sumOfBallsInFrame(rollsIndex);
                rollsIndex+=2;
            }
        }
        return score;
    }

    private Boolean isStrike(Integer rollsIndex) {
        return rolls[rollsIndex] == 10;
    }
    private Integer sumOfBallsInFrame(Integer rollsIndex) {
        return rolls[rollsIndex] + rolls[rollsIndex+1];
    }
    private Integer spareBounus(Integer rollsIndex) {
        return rolls[rollsIndex+2];
    }
    private Integer strikeBounus(Integer rollsIndex) {
        return rolls[rollsIndex+1] + rolls[rollsIndex+2];
    }
    private Boolean isSpare(Integer rollsIndex) {
        return rolls[rollsIndex] + rolls[rollsIndex+1] == 10;
    }
}
```

Unit Test #4 – Score calculates properly if bowler rolls a strike

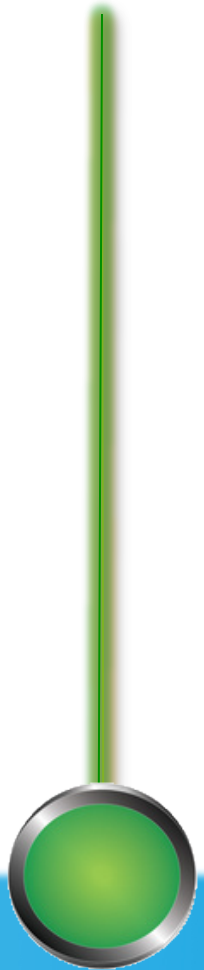


Next Use Case:

Score is 300 if bowler rolls a perfect game

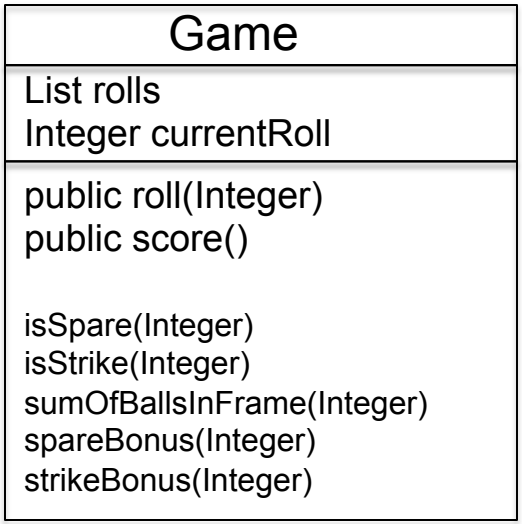
The Fifth Test

```
    ……..
    static testMethod void testOneSpare() {
        rollSpare();
        g.roll(3);
        rollMany(17,0);
       System.assertEquals(16,g.score(), 'Spare not scored correctly');
    }

    public testMethod void testOneStrike() {
        rollStrike(10);
        g.roll(3);
        g.roll(4);
        rollMany(16, 0);
        System.assertEquals(24, g.score(), 'One strike not scored correctly);
    }

    private testMethod void testPerfectGame() {
        rollMany(12,10);
        System.assertEquals(300, g.score(), 'Perfect Game not scored correctly);
    }

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

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



```
public with sharing class Game {
    ……..
    public Integer score() {
        Integer score = 0;
        Integer rollsIndex = 0;
        for (Integer frame=0, frame<10; frame++) {
            if (isStrike(rollsIndex)) {
                score += 10 + strikeBonus(rollsIndex);
                rollsIndex++;
            } else if ( isSpare(rollsIndex)) {
                score += 10 +  spareBonus(rollsIndex);
                rollsIndex += 2;
            } else {
                score += sumOfBallsInFrame(rollsIndex);
                rollsIndex+=2;
            }
        }
        return score;
    }

    private Boolean isStrike(Integer rollsIndex) {
        return rolls[rollsIndex] == 10;
    }
    private Integer sumOfBallsInFrame(Integer rollsIndex) {
        return rolls[rollsIndex] + rolls[rollsIndex+1];
    }
    private Integer spareBounus(Integer rollsIndex) {
        return rolls[rollsIndex+2];
    }
    private Integer strikeBounus(Integer rollsIndex) {
        return rolls[rollsIndex+1] + rolls[rollsIndex+2];
    }
    private Boolean isSpare(Integer rollsIndex) {
        return rolls[rollsIndex] + rolls[rollsIndex+1] == 10;
    }
}
```

Unit Test #5 – Score is 300 if bowler rolls a perfect game



What Have We Seen?

A disciplined test-driven approach means…

…Clean, more readable code

…Refining our design as we go

…Quicker feedback

…Regression testing becomes trivial

Follow-up

You finish the app with the following:

- https://github.com/knthornt/Dreamforce-2012---TDD-Session
- Complete Slide Deck (with tests 4 & 5)
- Tests 1 – 3 package
- Fully baked version

Completed kata video

- http://bit.ly/U9dKa8

Thank You!

Kyle Thornton, Mavens Consulting @knthornt

Sean Harrison, Mavens Consulting @GizmoForce

Mavens

MavensConsulting.com info@mavens.io 312.725.8528