

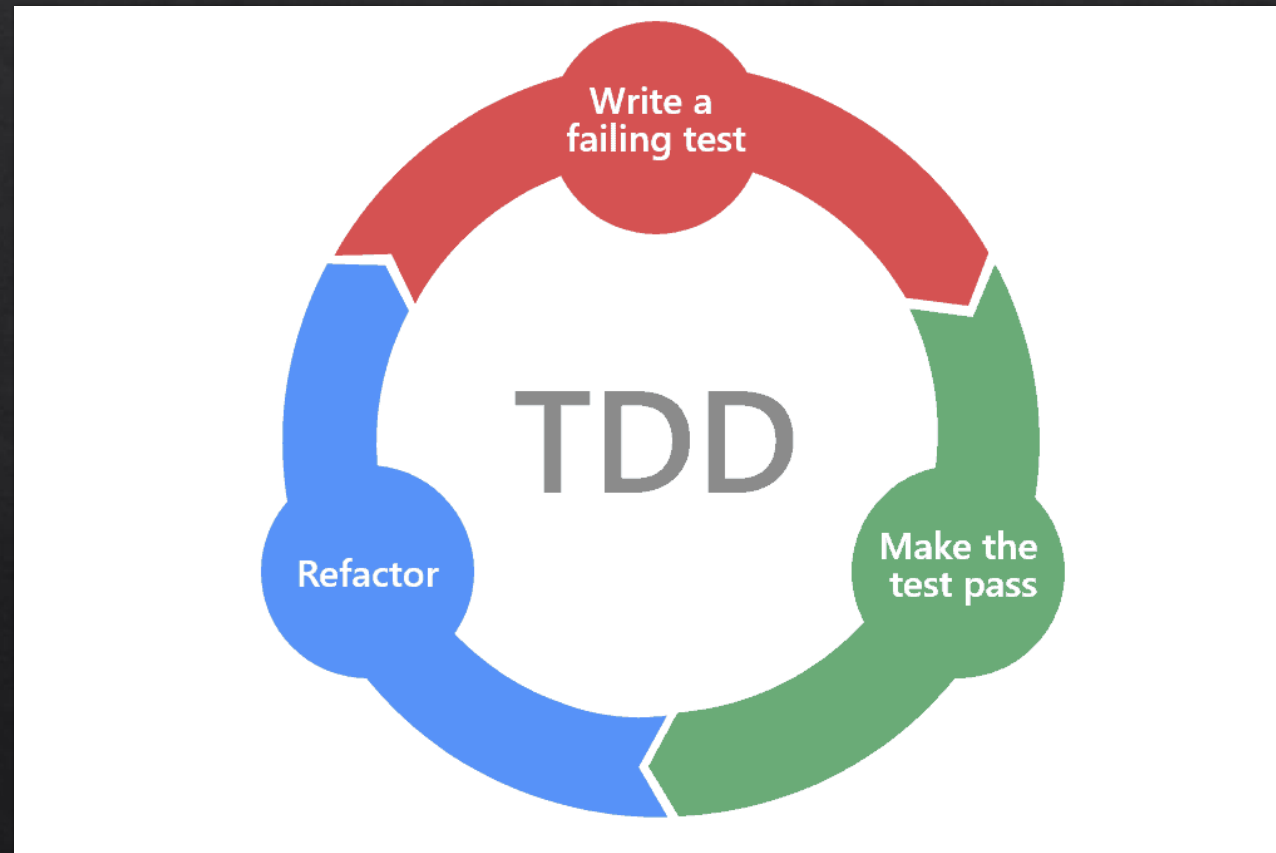
Test Driven Development

The hard way

Agenda

- ◆ Red / Green / Refactoring
- ◆ Implementation
 - ◆ Fake
 - ◆ Obvious
 - ◆ Triangulation
- ◆ Pitfall
- ◆ A good things

Red / Green / Refactoring



Fake it

```
1 import static org.junit.jupiter.api.Assertions.*;
2
3 import org.junit.jupiter.api.Test;
4
5 public class SquareTest {
6
7     @Test
8     public void testArea() {
9         Square s = new Square(3.0);
10        assertEquals(9.0, s.area());
11    }
12 }
13
```

```
1 public class Square {
2
3     public Square(double d) {}
4
5     public double area() {
6         return 9.0;
7     }
8 }
9
```

Obvious

```
1 import static org.junit.jupiter.api.Assertions.*;
2
3 import org.junit.jupiter.api.Test;
4
5 public class SquareTest {
6
7     @Test
8     public void testArea() {
9         Square s = new Square(3.0);
10        assertEquals(9.0, s.area());
11    }
12 }
13
```

```
1 public class Square {
2
3     private double d;
4
5     public Square(double d) {
6         this.d = d;
7     }
8
9     public double area() {
10        return d * d;
11    }
12 }
13
```

Triangulation



```
@Test
public void areaOfZero() {
    Square s = new Square(0.0);
    assertEquals(0.0, s.area());
}
```

```
6
7    public double area() {
8        return 0.0;
9    }
10 }
11
```

```
@Test
public void areaOfOne() {
    Square s = new Square(1.0);
    assertEquals(1.0, s.area());
}
}
```

```
3    private double d;
4
5    public Square(double d) {
6        this.d = d;
7    }
8
9    public double area() {
10        return d;
11    }
```

```
@Test
public void areaOfTwo() {
    Square s = new Square(2.0);
    assertEquals(4.0, s.area());
}
```

```
8
9    public double area() {
10        return d * d;
11    }
12 }
```

Pitfall

Fake Triangulation

```
game.play(); // 1
game.play(); // 2
game.play(); // 3
game.play(); // 4
game.play(); // 5
game.play(); // 6
game.play(); // 7
game.play(); // 8

assertEquals(false, game.getStatus());
```

```
2     ....
3     ....game.play(); // 1
4     ....game.play(); // 2
5     ....game.play(); // 3
6     ....game.play(); // 4
7     ....game.play(); // 5
8     ....game.play(); // 6
9     ....game.play(); // 7
10    ....game.play(); // 8
11    ....game.play(); // 9
12
13    ....assertEquals(true, game.getStatus());
```

A good thing

Prevents over-engineering

- ◊ Red test
- ◊ Coding...
- ◊ Coding...
- ◊ Coding...
- ◊ Too much code (smell?)
- ◊ git rebase
- ◊ Try again, simple way
- ◊ Coding....
- ◊ Green test

Tennis Game Kata

```
5 public class TennisGameTest {
6
7     @Test
8     public void shouldPlayerOneWin() {
9         TennisGame tennisGame = new TennisGame();
10
11         tennisGame.playerOneScores(); // 15:0
12         tennisGame.playerOneScores(); // 30:0
13         tennisGame.playerOneScores(); // 40:0
14         tennisGame.playerOneScores(); // Player One wins
15
16         assertEquals(ScoreResult.PLAYER_ONE_WIN, tennisGame.getScore());
17     }
18
19     @Test
20     public void shouldPlayerTwoWin() {
21         TennisGame tennisGame = new TennisGame();
22
23         tennisGame.playerTwoScores(); // 0:15
24         tennisGame.playerTwoScores(); // 0:30
25         tennisGame.playerTwoScores(); // 0:40
26         tennisGame.playerTwoScores(); // Player Two wins
27
28         assertEquals(ScoreResult.PLAYER_TWO_WIN, tennisGame.getScore());
29     }
30
31     @Test
32     public void shouldReturnPlayerOneAdvantage() {
33         TennisGame tennisGame = new TennisGame();
34
35         tennisGame.playerOneScores(); // 15:0
36         tennisGame.playerTwoScores(); // 15:15
37         tennisGame.playerOneScores(); // 30:15
38         tennisGame.playerTwoScores(); // 30:30
39         tennisGame.playerOneScores(); // Advantaged Player One
40
41         assertEquals(ScoreResult.PLAYER_ONE_ADVANTAGE, tennisGame.getScore());
42     }
43
44     @Test
```

```
44     @Test
45     public void shouldReturnPlayerTwoAdvantage() {
46         TennisGame tennisGame = new TennisGame();
47
48         tennisGame.playerOneScores(); // 15:0
49         tennisGame.playerTwoScores(); // 15:15
50         tennisGame.playerOneScores(); // 30:15
51         tennisGame.playerTwoScores(); // 30:30
52         tennisGame.playerTwoScores(); // Advantaged Player Two
53
54         assertEquals(ScoreResult.PLAYER_TWO_ADVANTAGE, tennisGame.getScore());
55     }
56
57     @Test
58     public void shouldReturnDeuce() {
59         TennisGame tennisGame = new TennisGame();
60
61         tennisGame.playerOneScores(); // 15:0
62         tennisGame.playerTwoScores(); // 15:15
63         tennisGame.playerOneScores(); // 30:15
64         tennisGame.playerTwoScores(); // 30:30
65         tennisGame.playerOneScores(); // Advantaged Player One
66         tennisGame.playerTwoScores(); // Deuce
67
68         assertEquals(ScoreResult.DEUCE, tennisGame.getScore());
69     }
70
71     @Test
72     public void shouldAdvantagePlayerWinsWhenScore() {
73         TennisGame tennisGame = new TennisGame();
74
75         tennisGame.playerOneScores(); // 15:0
76         tennisGame.playerTwoScores(); // 15:15
77         tennisGame.playerOneScores(); // 30:15
78         tennisGame.playerTwoScores(); // 30:30
79         tennisGame.playerOneScores(); // Advantaged Player One
80         tennisGame.playerOneScores(); // Player One Wins
81
82         assertEquals(ScoreResult.PLAYER_ONE_WIN, tennisGame.getScore());
83     }
```

Tennis Game Kata

- Class TennisGame
- Responsibility
 - Instance two players
 - Increment players score
 - Calculate game score
- Collaborations
 - Player
 - ScoreCalculator

```
1  public class TennisGame {
2
3      ..Player playerOne = new Player();
4      ..Player playerTwo = new Player();
5
6      ..private ScoreCalculator scoreCalculator = new ScoreCalculator();
7
8      ..public void playerOneScores() {
9          ....playerOne.score();
10     ..}
11
12     ..public void playerTwoScores() {
13         ....playerTwo.score();
14     ..}
15
16     ..public ScoreResult getScore() {
17         ....return scoreCalculator.calculate(playerOne, playerTwo);
18     ..}
19 }
```

Tennis Game Kata

- The ScoreCalculator refactoring and the missing mob programming
- `getScore() > 3`
- `Math.abs() > 1`
- `Equals()`
- `compareTo() > 0`
- Calculate method

```
1  ✓ public class ScoreCalculator {
2
3  ✓  ..public boolean playerWins(Player playerOne, Player playerTwo) {
4      .....int pointDiff = playerOne.getScore() - playerTwo.getScore();
5
6      .....return (
7          .....(playerOne.getScore() > 3 || playerTwo.getScore() > 3) &&
8              .....Math.abs(pointDiff) > 1
9          .....);
10     ..}
11
12  ✓  ..public boolean playerAdvantage(Player playerOne, Player playerTwo) {
13      .....return (
14          .....(playerOne.getScore() ≥ 3 || playerTwo.getScore() ≥ 3) &&
15              .....!playerOne.equals(playerTwo)
16          .....);
17     ..}
18
19  ✓  ..public boolean playerDuece(Player playerOne, Player playerTwo) {
20      .....return (
21          .....(playerOne.getScore() ≥ 3 || playerTwo.getScore() ≥ 3) &&
22              .....playerOne.equals(playerTwo)
23          .....);
24     ..}
25
26  ✓  ..ScoreResult calculate(Player playerOne, Player playerTwo) {
27  ✓  .....if (playerWins(playerOne, playerTwo)) {
28      .....return playerOne.compareTo(playerTwo) > 0
29      .....? ScoreResult.PLAYER_ONE_WIN
30      .....: ScoreResult.PLAYER_TWO_WIN;
31     ....}
32
33  ✓  .....if (playerAdvantage(playerOne, playerTwo)) {
34      .....return playerOne.compareTo(playerTwo) > 0
35      .....? ScoreResult.PLAYER_ONE_ADVANTAGE
36      .....: ScoreResult.PLAYER_TWO_ADVANTAGE;
37     ....}
38
39  ✓  .....if (playerDuece(playerOne, playerTwo)) {
40      .....return ScoreResult.DEUCE;
41     ....}
42
43     .....return ScoreResult.IN_PROGESS;
44     ..}
45 }
..
```

End