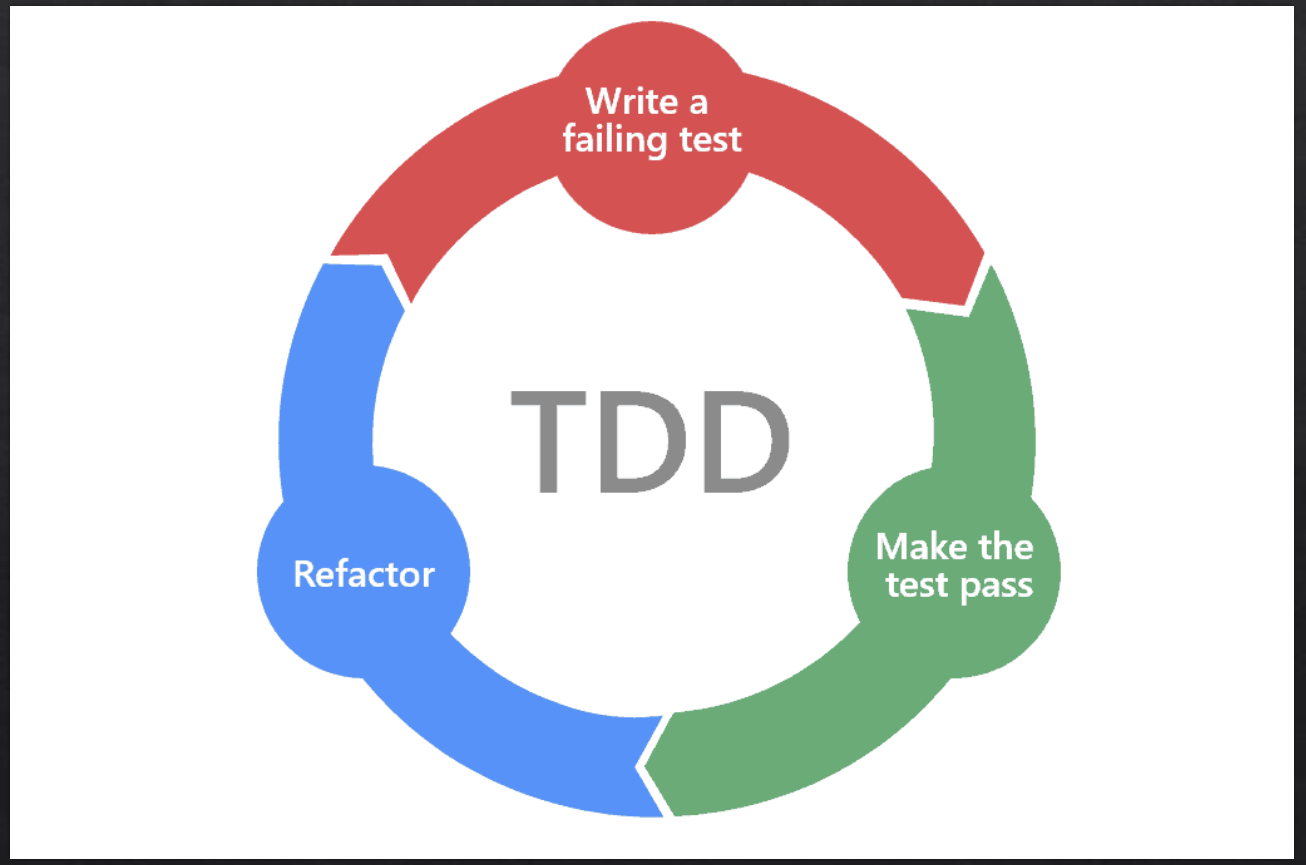
Test Driven Development

The hard way

Agenda

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

Red / Green / Refactoring

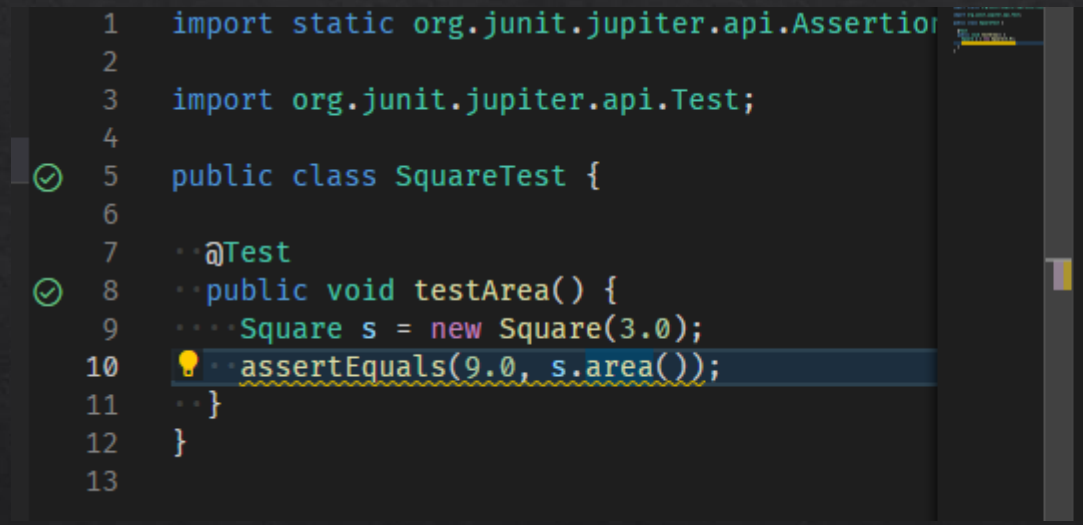


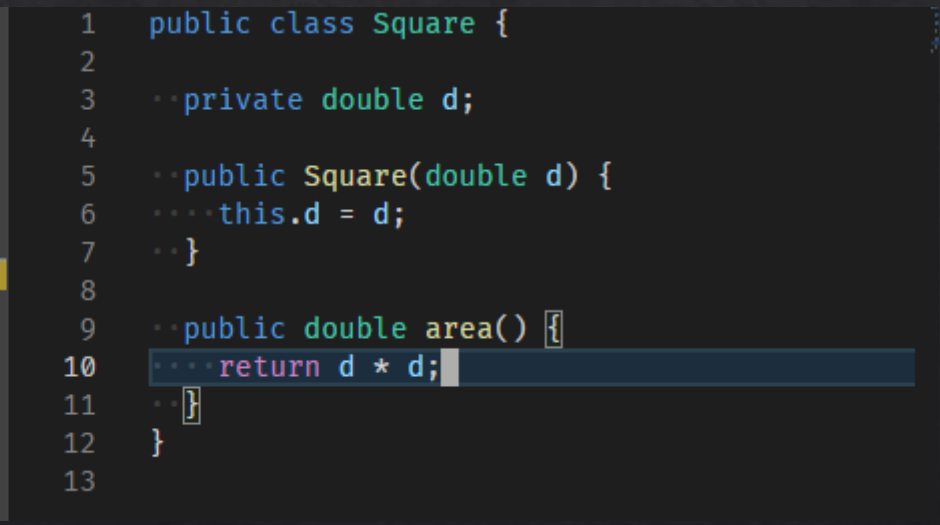
Fake it

```java
import static org.junit.jupiter.api.Assertio

import org.junit.jupiter.api.Test;

public class SquareTest {

    @Test
    public void testArea() {
        Square s = new Square(3.0);
        assertEquals(9.0, s.area());
    }
}
```

```java
public class Square {

    public Square(double d) {}

    public double area() {
        return 9.0;
    }
}
```

Obvious


1 import static org.junit.jupiter.api.Assertio
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

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

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

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

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

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

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

Pitfall

Fake Triangolation

```
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());
```

```
    game.play(); // 1
    game.play(); // 2
    game.play(); // 3
    game.play(); // 4
    game.play(); // 5
    game.play(); // 6
    game.play(); // 7
    game.play(); // 8
    game.play(); // 9

    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

```java
public class TennisGameTest {

    @Test
    public void shouldPlayerOneWin() {
        TennisGame tennisGame = new TennisGame();

        tennisGame.playerOneScores(); // 15:0
        tennisGame.playerOneScores(); // 30:0
        tennisGame.playerOneScores(); // 40:0
        tennisGame.playerOneScores(); // Player One wins

        assertEquals(ScoreResult.PLAYER_ONE_WIN, tennisGame.getScore());
    }

    @Test
    public void shouldPlayerTwoWin() {
        TennisGame tennisGame = new TennisGame();

        tennisGame.playerTwoScores(); // 0:15
        tennisGame.playerTwoScores(); // 0:30
        tennisGame.playerTwoScores(); // 0:40
        tennisGame.playerTwoScores(); // Player Two wins

        assertEquals(ScoreResult.PLAYER_TWO_WIN, tennisGame.getScore());
    }

    @Test
    public void shouldReturnPlayerOneAdvantage() {
        TennisGame tennisGame = new TennisGame();

        tennisGame.playerOneScores(); // 15:0
        tennisGame.playerTwoScores(); // 15:15
        tennisGame.playerOneScores(); // 30:15
        tennisGame.playerTwoScores(); // 30:30
        tennisGame.playerOneScores(); // Advantaged Player One

        assertEquals(ScoreResult.PLAYER_ONE_ADVANTAGE, tennisGame.getScore());
    }

    @Test
    public void shouldReturnPlayerTwoAdvantage() {
        TennisGame tennisGame = new TennisGame();

        tennisGame.playerOneScores(); // 15:0
        tennisGame.playerTwoScores(); // 15:15
        tennisGame.playerOneScores(); // 30:15
        tennisGame.playerTwoScores(); // 30:30
        tennisGame.playerTwoScores(); // Advantaged Player Two

        assertEquals(ScoreResult.PLAYER_TWO_ADVANTAGE, tennisGame.getScore());
    }

    @Test
    public void shouldReturnDeuce() {
        TennisGame tennisGame = new TennisGame();

        tennisGame.playerOneScores(); // 15:0
        tennisGame.playerTwoScores(); // 15:15
        tennisGame.playerOneScores(); // 30:15
        tennisGame.playerTwoScores(); // 30:30
        tennisGame.playerOneScores(); // Advantaged Player One
        tennisGame.playerTwoScores(); // Deuce

        assertEquals(ScoreResult.DEUCE, tennisGame.getScore());
    }

    @Test
    public void shouldAdvantagePlayerWinsWhenScore() {
        TennisGame tennisGame = new TennisGame();

        tennisGame.playerOneScores(); // 15:0
        tennisGame.playerTwoScores(); // 15:15
        tennisGame.playerOneScores(); // 30:15
        tennisGame.playerTwoScores(); // 30:30
        tennisGame.playerOneScores(); // Advantaged Player One
        tennisGame.playerOneScores(); // Player One Wins

        assertEquals(ScoreResult.PLAYER_ONE_WIN, tennisGame.getScore());
    }
```

Tennis Game Kata

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

```java
public class TennisGame {

  Player playerOne = new Player();
  Player playerTwo = new Player();

  private ScoreCalculator scoreCalculator = new ScoreCalculator();

  public void playerOneScores() {
    playerOne.score();
  }

  public void playerTwoScores() {
    playerTwo.score();
  }

  public ScoreResult getScore() {
    return scoreCalculator.calculate(playerOne, playerTwo);
  }
}
```

Tennis Game Kata

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

```java
public class ScoreCalculator {

    public boolean playerWins(Player playerOne, Player playerTwo) {
        int pointDiff = playerOne.getScore() - playerTwo.getScore();

        return (
            (playerOne.getScore() > 3 || playerTwo.getScore() > 3) &&
            Math.abs(pointDiff) > 1
        );
    }

    public boolean playerAdvantage(Player playerOne, Player playerTwo) {
        return (
            (playerOne.getScore() >= 3 || playerTwo.getScore() >= 3) &&
            !playerOne.equals(playerTwo)
        );
    }

    public boolean playerDuece(Player playerOne, Player playerTwo) {
        return (
            (playerOne.getScore() >= 3 || playerTwo.getScore() >= 3) &&
            playerOne.equals(playerTwo)
        );
    }

    ScoreResult calculate(Player playerOne, Player playerTwo) {
        if (playerWins(playerOne, playerTwo)) {
            return playerOne.compareTo(playerTwo) > 0
                ? ScoreResult.PLAYER_ONE_WIN
                : ScoreResult.PLAYER_TWO_WIN;
        }

        if (playerAdvantage(playerOne, playerTwo)) {
            return playerOne.compareTo(playerTwo) > 0
                ? ScoreResult.PLAYER_ONE_ADVANTAGE
                : ScoreResult.PLAYER_TWO_ADVANTAGE;
        }

        if (playerDuece(playerOne, playerTwo)) {
            return ScoreResult.DEUCE;
        }

        return ScoreResult.IN_PROGESS;
    }
}
```

End