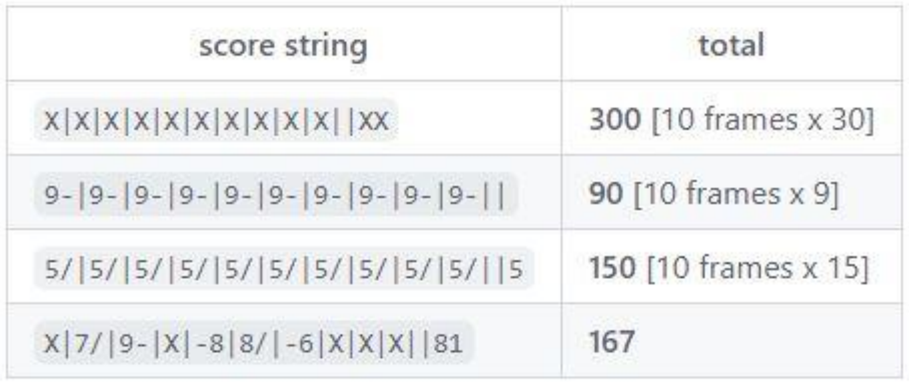
PRACTICING KATAS

2ND TRY ON « BOWLING KATA » WITH A LITTLE BIT DESIGN AND LOTS OF REFACTORING

21.10.2021

Remember the 3 parts of the Code:

- **Unfold:** get the data -> parse the input String
- **Map:** operation on data -> calculate the Score
- **Fold:** take and aggregate the result -> parse the output



score string	total
X|X|X|X|X|X|X|X|X|X||XX	**300** [10 frames x 30]
9-|9-|9-|9-|9-|9-|9-|9-|9-|9-||	**90** [10 frames x 9]
5/|5/|5/|5/|5/|5/|5/|5/|5/|5/||5	**150** [10 frames x 15]
X|7/|9-|X|-8|8/|-6|X|X|X||81	**167**

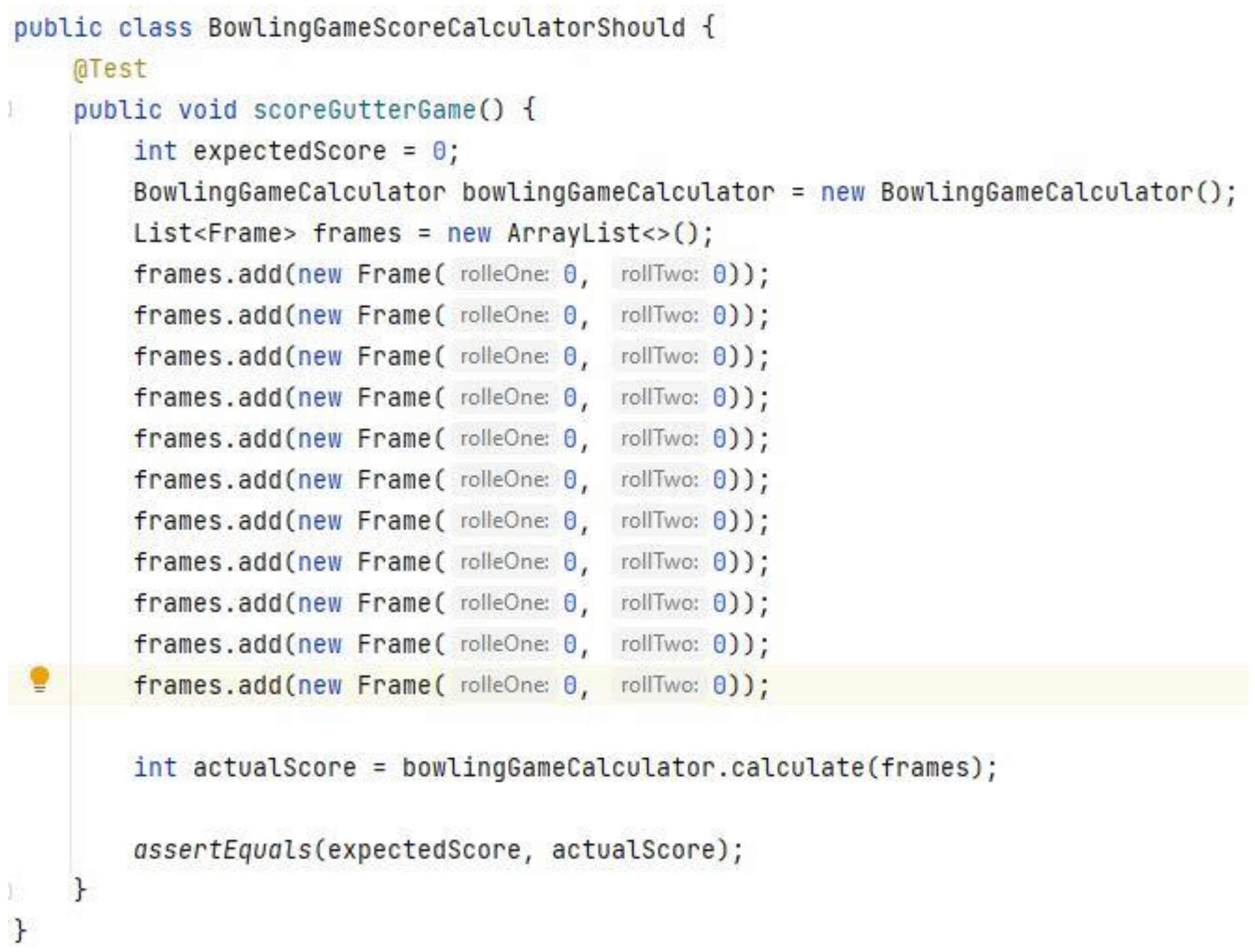
Decided to focus on the calculation

➢ Directly changed the input to array of integers for single rolls:

E.g.: 3,4, 1,2, 8,1, 3,6, 1,5, 0,2, 0,0, 7,2, 6,1, 5,2

- X is 10
- - is 0
- / is the remaining rolled pins to get a spare

FIRST STEPS



```java
public class BowlingGameScoreCalculatorShould {
    @Test
    public void scoreGutterGame() {
        int expectedScore = 0;
        BowlingGameCalculator bowlingGameCalculator = new BowlingGameCalculator();
        List<Frame> frames = new ArrayList<>();
        frames.add(new Frame( rolleOne: 0,  rollTwo: 0));
        frames.add(new Frame( rolleOne: 0,  rollTwo: 0));
        frames.add(new Frame( rolleOne: 0,  rollTwo: 0));
        frames.add(new Frame( rolleOne: 0,  rollTwo: 0));
        frames.add(new Frame( rolleOne: 0,  rollTwo: 0));
        frames.add(new Frame( rolleOne: 0,  rollTwo: 0));
        frames.add(new Frame( rolleOne: 0,  rollTwo: 0));
        frames.add(new Frame( rolleOne: 0,  rollTwo: 0));
        frames.add(new Frame( rolleOne: 0,  rollTwo: 0));
        frames.add(new Frame( rolleOne: 0,  rollTwo: 0));
        frames.add(new Frame( rolleOne: 0,  rollTwo: 0));

        int actualScore = bowlingGameCalculator.calculate(frames);

        assertEquals(expectedScore, actualScore);
    }
}
```

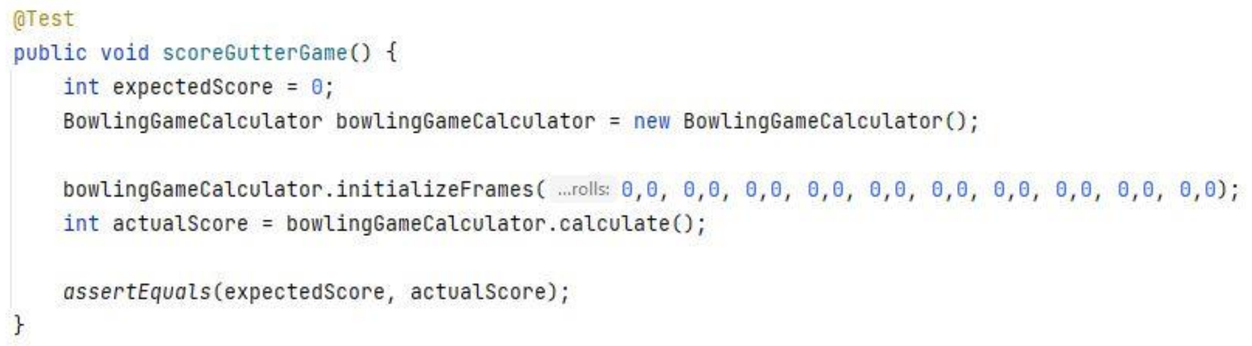
What shall the first test test?

- Try to think of “simplest” input: gutter game

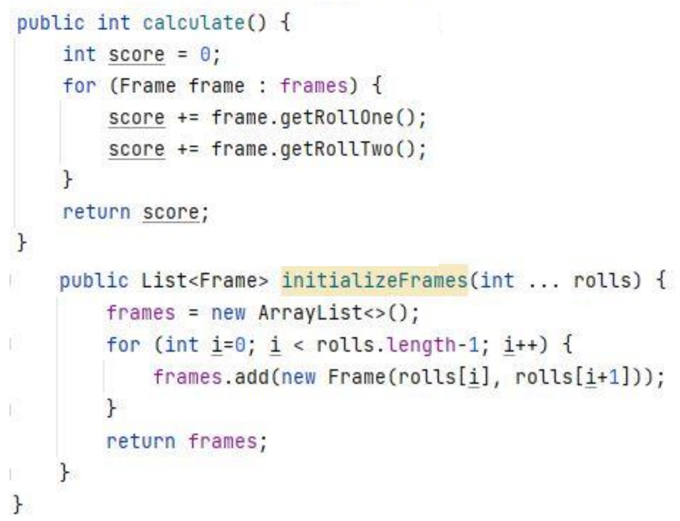
Any design ideas upfront?

- A game consists of frames
  - Decided to use Frame-class with two rolls for each frame
  - BowlingGameCalculator consumes the frame-list to calculate the game score

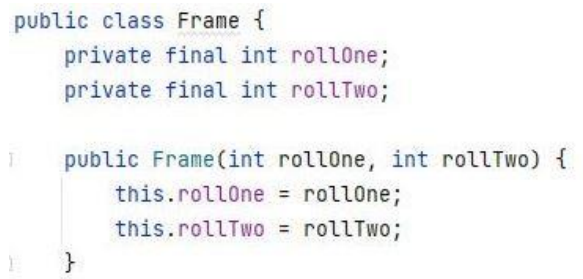
FIRST STEPS



```java
@Test
public void scoreGutterGame() {
    int expectedScore = 0;
    BowlingGameCalculator bowlingGameCalculator = new BowlingGameCalculator();

    bowlingGameCalculator.initializeFrames( ...rolls: 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0);
    int actualScore = bowlingGameCalculator.calculate();

    assertEquals(expectedScore, actualScore);
}
```



```java
public int calculate() {
    int score = 0;
    for (Frame frame : frames) {
        score += frame.getRollOne();
        score += frame.getRollTwo();
    }
    return score;
}
    public List<Frame> initializeFrames(int ... rolls) {
        frames = new ArrayList<>();
        for (int i=0; i < rolls.length-1; i++) {
            frames.add(new Frame(rolls[i], rolls[i+1]));
        }
        return frames;
    }
}
```

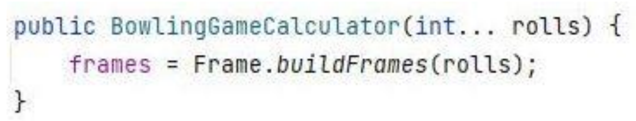


```java
public class Frame {
    private final int rollOne;
    private final int rollTwo;

    public Frame(int rollOne, int rollTwo) {
        this.rollOne = rollOne;
        this.rollTwo = rollTwo;
    }
```

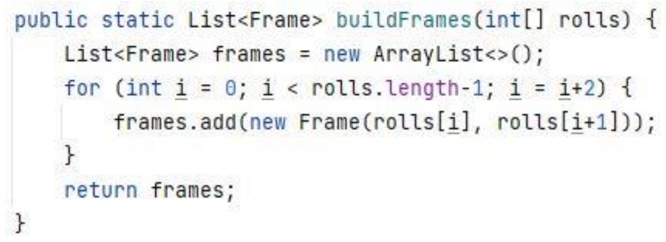
Make it green (with most obvious code)

- BowlingGameCalculator initializes Frames from input
- Every Frame consists of 2 rolls
- Calculate() sums up the two rolls of every frame

FIRST REFACTORINGS



```java
public BowlingGameCalculator(int... rolls) {
    frames = Frame.buildFrames(rolls);
}
```

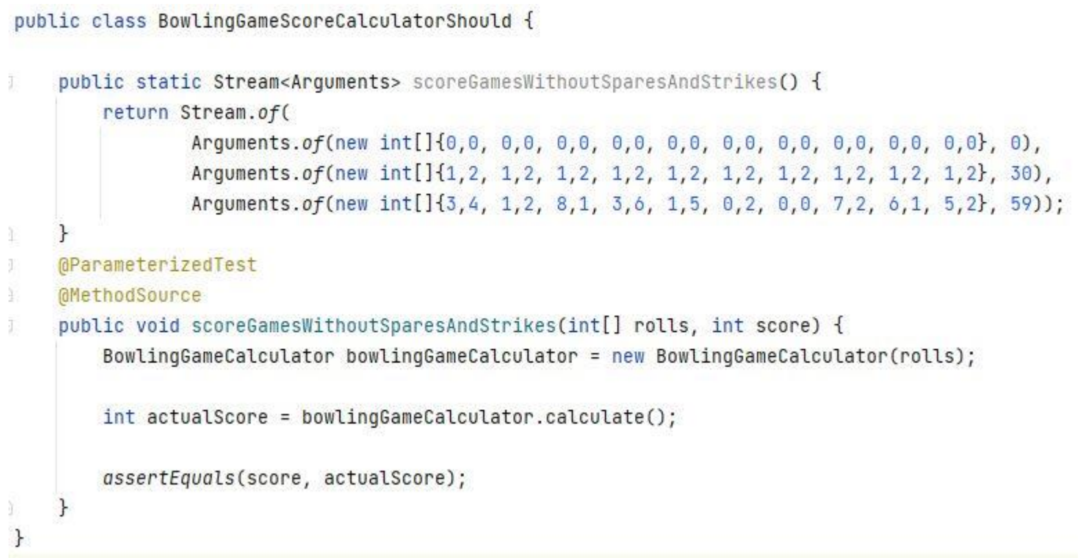


```java
public static List<Frame> buildFrames(int[] rolls) {
    List<Frame> frames = new ArrayList<>();
    for (int i = 0; i < rolls.length-1; i = i+2) {
        frames.add(new Frame(rolls[i], rolls[i+1]));
    }
    return frames;
}
```

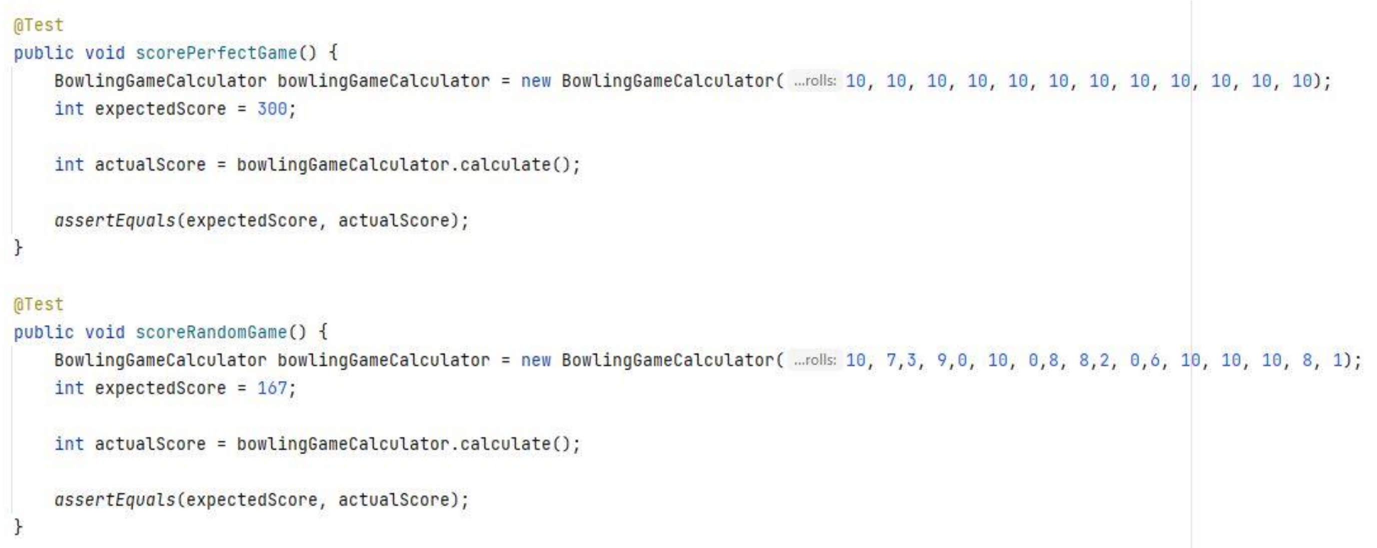
What can we already refactor?

- ➢ Move the initializing of the Frame-list into the Frame-class (**Feature Envy**)

AFTER A WHILE… ADDING MORE FEATURES



```java
public class BowlingGameScoreCalculatorShould {

    public static Stream<Arguments> scoreGamesWithoutSparesAndStrikes() {
        return Stream.of(
                Arguments.of(new int[]{0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0}, 0),
                Arguments.of(new int[]{1,2, 1,2, 1,2, 1,2, 1,2, 1,2, 1,2, 1,2, 1,2, 1,2}, 30),
                Arguments.of(new int[]{3,4, 1,2, 8,1, 3,6, 1,5, 0,2, 0,0, 7,2, 6,1, 5,2}, 59));
    }
    @ParameterizedTest
    @MethodSource
    public void scoreGamesWithoutSparesAndStrikes(int[] rolls, int score) {
        BowlingGameCalculator bowlingGameCalculator = new BowlingGameCalculator(rolls);

        int actualScore = bowlingGameCalculator.calculate();

        assertEquals(score, actualScore);
    }
}
```



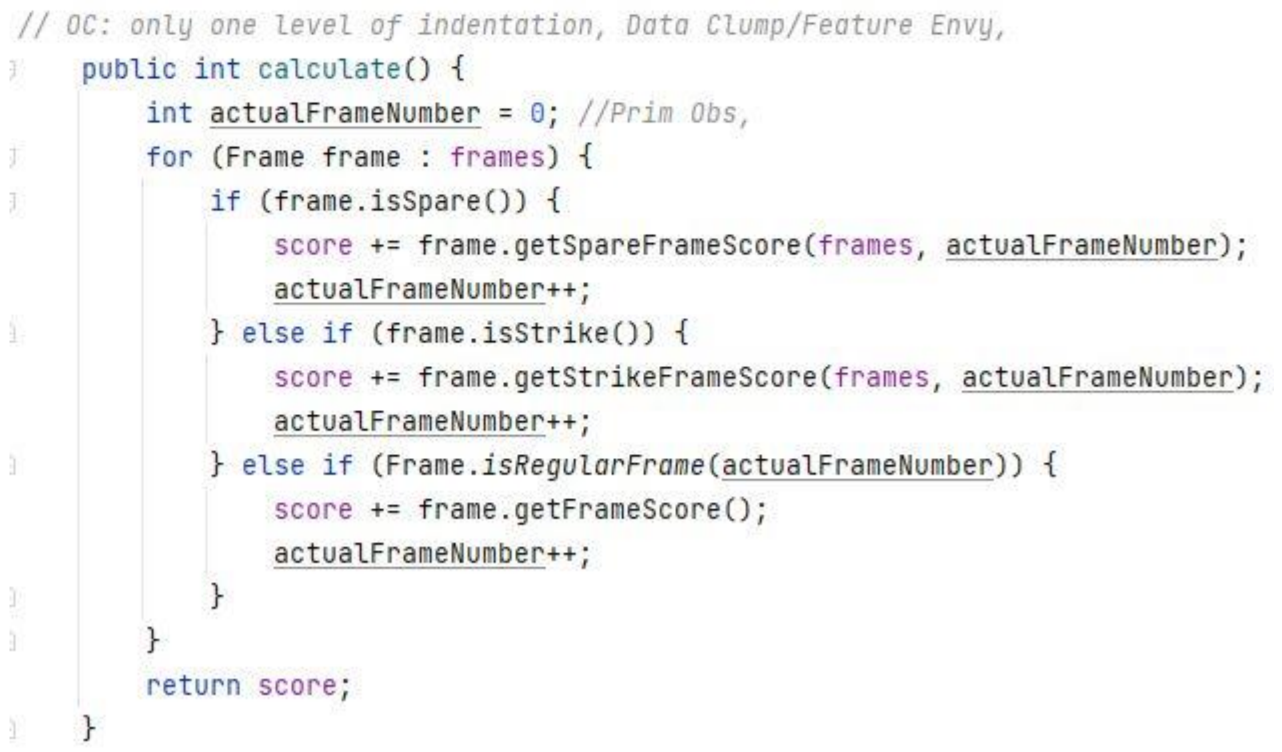
```java
@Test
public void scorePerfectGame() {
    BowlingGameCalculator bowlingGameCalculator = new BowlingGameCalculator( ...rolls: 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10);
    int expectedScore = 300;

    int actualScore = bowlingGameCalculator.calculate();

    assertEquals(expectedScore, actualScore);
}

@Test
public void scoreRandomGame() {
    BowlingGameCalculator bowlingGameCalculator = new BowlingGameCalculator( ...rolls: 10, 7,3, 9,0, 10, 0,8, 8,2, 0,6, 10, 10, 10, 8, 1);
    int expectedScore = 167;

    int actualScore = bowlingGameCalculator.calculate();

    assertEquals(expectedScore, actualScore);
}
```

TESTS

Summed up the tests for games without spares or strikes into a parameterized test

New tests for the new features of dealing with strikes and spares

AFTER A WHILE… ADDING MORE FEATURES



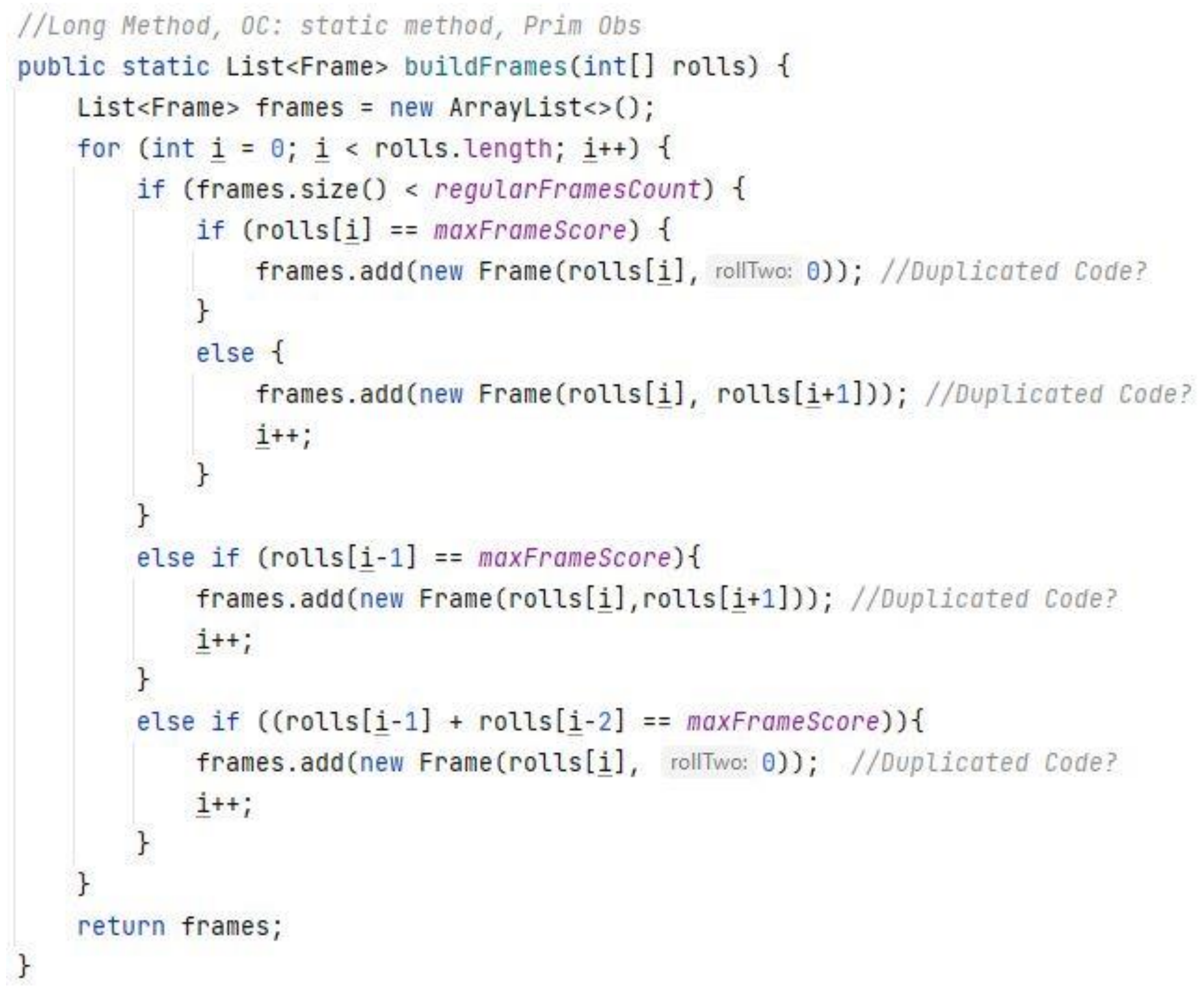
```
// OC: only one level of indentation, Data Clump/Feature Envy,
    public int calculate() {
        int actualFrameNumber = 0; //Prim Obs,
        for (Frame frame : frames) {
            if (frame.isSpare()) {
                score += frame.getSpareFrameScore(frames, actualFrameNumber);
                actualFrameNumber++;
            } else if (frame.isStrike()) {
                score += frame.getStrikeFrameScore(frames, actualFrameNumber);
                actualFrameNumber++;
            } else if (Frame.isRegularFrame(actualFrameNumber)) {
                score += frame.getFrameScore();
                actualFrameNumber++;
            }
        }
        return score;
    }
```

PRODUCTION CODE

BowlingGameCalculator class

- Object Calisthenics:
  - Only one level of indentation
    - Several if/else statements in the for-loop
- Data Clump/Feature Envy
  - For the calculation the usage of Frame class methods is excessive
- Primitive Obsession
  - For the calculation we need an Integer counter for the frame we are operating with
- Others
  - The score is mutated
  - “get”-methods are calculating and returning at once

AFTER A WHILE… ADDING MORE FEATURES



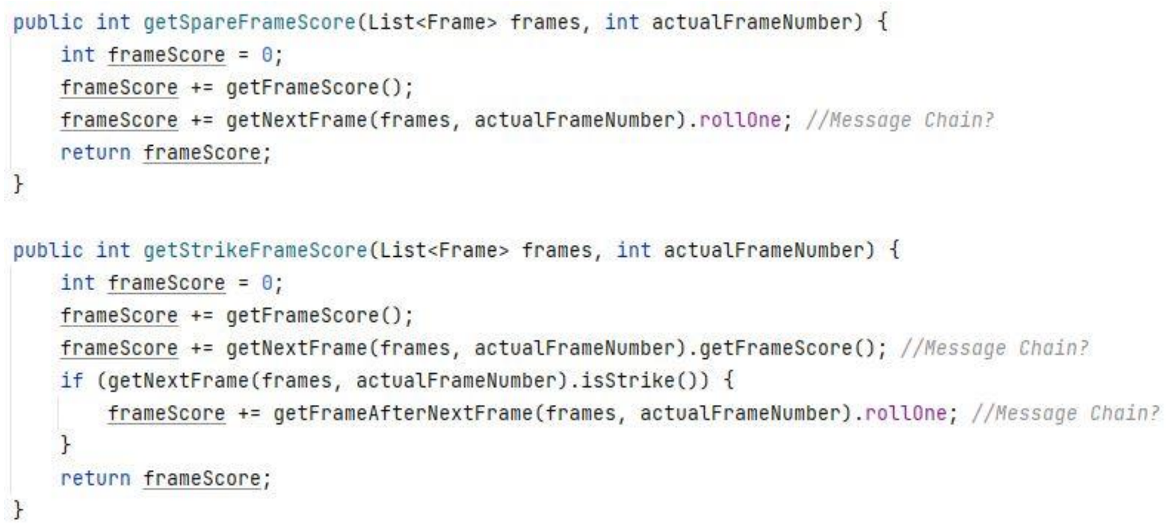
```
//Long Method, OC: static method, Prim Obs
public static List<Frame> buildFrames(int[] rolls) {
    List<Frame> frames = new ArrayList<>();
    for (int i = 0; i < rolls.length; i++) {
        if (frames.size() < regularFramesCount) {
            if (rolls[i] == maxFrameScore) {
                frames.add(new Frame(rolls[i], rollTwo: 0)); //Duplicated Code?
            }
            else {
                frames.add(new Frame(rolls[i], rolls[i+1])); //Duplicated Code?
                i++;
            }
        }
        else if (rolls[i-1] == maxFrameScore){
            frames.add(new Frame(rolls[i],rolls[i+1])); //Duplicated Code?
            i++;
        }
        else if ((rolls[i-1] + rolls[i-2] == maxFrameScore)){
            frames.add(new Frame(rolls[i], rollTwo: 0));  //Duplicated Code?
            i++;
        }
    }
    return frames;
}
```

PRODUCTION CODE

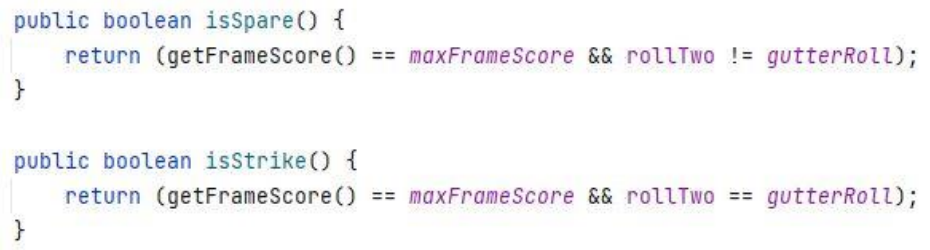
Frame class (buildFrames)

- Long Method
  - The method is longer than 15 lines
- Object Calisthenics:
  - Static method, object should have a state
  - But buildFrames() just parses the input into a list of frames
  - Maybe this method belongs to another class like “FrameParser”
- Primitive Obsession
  - Iterating over “i” (no good naming) and initializing the frames with mathmatical operations with “i”
- Duplicated Code?
  - Initializing “normal” frames (this and next roll) and initializing strike games with this roll and “0”
  - Idea: Frame as interface for “RegularFrame”, “SpareFrame” and “StrikeFrame”?

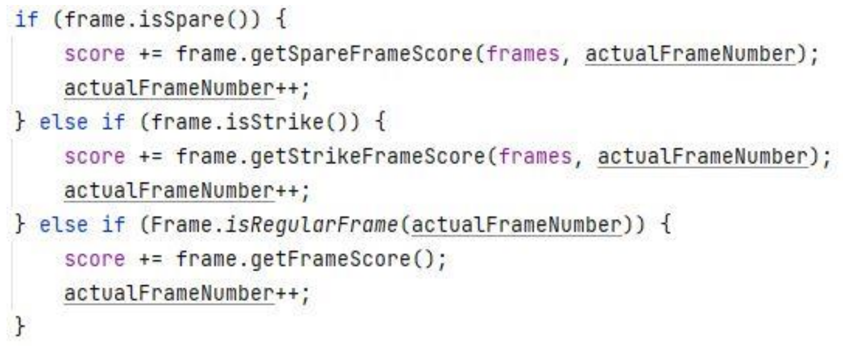
STRATEGY PATTERN?



```java
public int getSpareFrameScore(List<Frame> frames, int actualFrameNumber) {
    int frameScore = 0;
    frameScore += getFrameScore();
    frameScore += getNextFrame(frames, actualFrameNumber).rollOne; //Message Chain?
    return frameScore;
}

public int getStrikeFrameScore(List<Frame> frames, int actualFrameNumber) {
    int frameScore = 0;
    frameScore += getFrameScore();
    frameScore += getNextFrame(frames, actualFrameNumber).getFrameScore(); //Message Chain?
    if (getNextFrame(frames, actualFrameNumber).isStrike()) {
        frameScore += getFrameAfterNextFrame(frames, actualFrameNumber).rollOne; //Message Chain?
    }
    return frameScore;
}
```



```java
public boolean isSpare() {
    return (getFrameScore() == maxFrameScore && rollTwo != gutterRoll);
}

public boolean isStrike() {
    return (getFrameScore() == maxFrameScore && rollTwo == gutterRoll);
}
```



```java
if (frame.isSpare()) {
    score += frame.getSpareFrameScore(frames, actualFrameNumber);
    actualFrameNumber++;
} else if (frame.isStrike()) {
    score += frame.getStrikeFrameScore(frames, actualFrameNumber);
    actualFrameNumber++;
} else if (Frame.isRegularFrame(actualFrameNumber)) {
    score += frame.getFrameScore();
    actualFrameNumber++;
}
```

Calculating Score based on kind of frame

- Interface Frame just needs method “getFrameScore”
- “getFrameScore” is implemented differently by RegularFrame, SpareFrame and StrikeFrame

No longer need of methods “isSpare” or “isStrike”, because the Frames are initialized as “SpareFrame” or “StrikeFrame”

We will get rid of the “if” and “else if” statements and can just call “frame.getFrameScore” for every frame when it is initialized correctly

BUT AT LEAST… IT WORKED :D

sauce: https://lh3.googleusercontent.com/proxy/jrcVNzLGGGMSPfEbG3ApX2GnkDBnlN0WvUCwMOYGPyZd_KIbkiNVYJkmqWzSRaHbgxmIfYpwVXy1vJGFM16oUQ

Don’t celebrate ugly code!!!

sauce: https://www.voicy.network/Content/Clips/Images/44a1257f-5b56-4399-8de5-ffe79252a47b.jpg

EXPERIENCES AND SUCCESSES

The power of the mob

- Would not have tried so many „wrong“ ideas thanks to discussions in the mob
- Won‘t have been stuck as often and for that time thanks to having access to the ideas of the other mob members
  - In the mob we would have found a working solution much faster and it would have been much „prettier“ in that time

Thanks to the course I managed to get at least to a working score calculation!

- Did just take a piece of the time it took me last time without having a fully working solution
- At least the code is not AS UGLY as last time :D (but still is not obvious…)

MERCI

Looking forward to next module
And
Have a great weekend!

LinkedIn: Dominique Latza

dominique_latza@live.de