Activity 5-1: TDD Practice 1 - Bowling Game Scorer

Practicing test-driven development (TDD)

This is a paired-programming exercise.

The goal of this activity is to practice the red-green-refactor cycle:

- RED: You must add a new test or some new code to an ongoing test, which causes the test to fail.
- GREEN: You must add the code needed to make the failing test pass.
- REFACTOR: You must clean up the code, including test code, to make them more readable; must do before moving on to writing the next test.

Use the red-green-refactor prop as you work through this activity to help you internalize the process.

General Rules to follow (TDD Rules):

1. You are not allowed to write any production code unless it is to make a failing unit test pass.
2. You are not allowed to write any more of a unit test than is sufficient to fail; and not compiling is failing.
3. You are not allowed to write any more production code than is sufficient to pass the failing test.

Bowling game scoring rules

1. A bowling game consists of 10 **frames**.
2. On each frame the player makes up to two **throws**.
3. Each throw/roll knocks down a number of **pins**. Each pin knocked down scores a point.
4. There is a total of 10 pins that can be knocked down on each frame.
5. Knocking all 10 pins down with the two rolls of the frame is called a **spare**.
6. The pins in the next roll after the spare are added to the spare frame's score.
7. Knocking all 10 pins down with the first of the frame rolls (and subsequently NOT doing a second roll) is called a **strike**.
8. The pins in the next two rolls after a strike are added to the strike frame's score.
9. More rolls may occur after all 10 frames are completed, to account for the extra points allotted to spares/strikes on the last frame.

10. Your score is the sum of your scores in all the frames.

Frame	Roll 1	Roll 2	Special	Extra Points	Frame Score	Game Score
1	1	4		0	5	5
2	4	5		0	9	14
3	6	4	Spare	**5**	15	29
4	**5**	5	Spare	**10**	20	49
5	**10**		Strike	**1**	11	60
6	**0**	**1**		0	1	61
7	7	3	Spare	**6**	16	77
8	**6**	4	Spare	**10**	20	97
9	**10**		Strike	**10**	20	117
10	**2**	**8**	Spare	**6**	16	133
	6					133

Example: Typical game

Frame	Roll 1	Roll 2	Special	Extra Points	Frame Score	Game Score
1	10		Strike	**20**	30	30
2	**10**		Strike	**20**	30	60
3	**10**		Strike	**20**	30	90
4	**10**		Strike	**20**	30	120
5	**10**		Strike	**20**	30	150
6	**10**		Strike	**20**	30	180
7	**10**		Strike	**20**	30	210
8	**10**		Strike	**20**	30	240
9	**10**		Strike	**20**	30	270
10	**10**		Strike	**20**	30	300
	10	**10**				300

Example: Perfect game

Development Steps

These are the steps (as represented by test cases) that you will be going through to develop the bowling game scorer.

1. Getting empty test to compile
2. Can create game

3. Can roll
4. Score a gutter game
5. Score all ones game
6. Score game with one spare and rest gutter balls
7. Score game with one strike and rest gutter balls
8. Score a perfect game

Step 1: Getting empty test to compile

RED
- Start a new IntelliJ project.
- Create a test directory, and mark it as "Test Source Directory"
- Create a BowlingTest.java class in the test directory.
- Create a nothing() test with the @Test annotation.

GREEN
- Use intention on the red-marked words to add JUnit 4 to the classpath if needed, and to import org.junit.test.
- Make sure you can run your test.

REFACTOR
- Nothing to do.

Step 2: Can create game

RED
- Change the nothing test to canCreateGame.
- Add a Game g = new Game(); line to the test.

GREEN
- Use intention on the red-marked words to create a new class Game.
- Check that tests pass.

REFACTOR
- Nothing to do.

Step 3: Can roll

RED
- Change the canCreateGame test to canRoll.
- Add a g.roll(0); to the test.

GREEN
- Use intention on the red-marked word to create the method roll in Game.
- Make sure the parameter to that method is pins.
- Check that tests pass.

REFACTOR
- Nothing to do.

Step 4: Score a gutter game

RED
- Make a new gutterGame test.
- In the body create a new game g, then roll twenty zeroes (for loop).
- Finish up with 'assertEquals(0, g.score());

GREEN
- Use intention on assertEquals to import the static method from org.junit.
- Use intention on score to create a score method in Game.
- Make the method return -1 and run the test to see it fail.
- Make the method return 0 and run the test to see it pass.

REFACTOR
- Extract Field from the Game g variable, and have it initialized in the setUp method.
- Remove the local g variable from any of the tests, so that the field variable is used.
- Eliminate the no-longer-relevant canRoll test.
- Use intention on Exception to remove the unneeded throws Exception.

Step 5: Score all ones game

RED
- Add a allOnes test.
- In its body roll twenty ones in a loop.
- Finish up with assertEquals(20, g.score());.
- Watch the test fail.

GREEN
- In Game, create a field score, initialize it to 0, and return it in the score method.
- In the roll method, increment the score by the pins.
- Watch the test pass.

REFACTOR
- In gutterGame test, perform Extract Variable on the 0 within the roll, to variable pins. Move it above the loop.
- Perform Extract Variable on the 20 in the loop, to variable n.
- Perform Extract Method on the loop, to method rollMany, place parameters in order n, pins, and replace duplicates.
- In gutterGame test, perform Inline on the local variables n and pins.
- Perform a Move refactoring on the rollMany method, to the Game class.
- Confirm tests still pass.

Step 6: Score game with one spare and rest gutter balls

PRE-REFACTOR
- Create field rolls in Game, to store array of ints. Initialize it to an array of length 21.
- Create field currentRoll in Game, initialized to 0. Delete the score field.
- In roll method, add the pins to the currentRoll spot in the rolls array, and increment currentRoll. Remove the score increment.
- In score method, loop over the rolls array up to the currentRoll, and accumulate the values in a local variable score, then return that variable.
- Confirm that your tests still pass.
- Change for loop in score so that it loops over 10 frames. Create a local firstInFrame variable starting at 0. For each frame add to the score the value of rolls at firstInFrame and firstInFrame + 1, then increment firstInFrame by 2.
- Confirm that your tests still pass.

RED
- Add a oneSpare test, on which you roll 5, 5 and 3, then roll 17 zeroes (using rollMany).
- Assert that the score should equal 16.
- Watch the test fail.

GREEN
- At the start of the loop, add an if test for whether rolls[firstInFrame] + rolls[firstInFrame + 1] equals 10. If it is, increment the score by 10 + rolls[firstInFrame + 2] then increment firstInFrame by 2.
- Place the remaining body of the loop into an else clause.
- Watch the tests pass.

REFACTOR
- Extract the test in the if into a method isSpare.
- Extract the two 5 rolls in the test method into a rollSpare method.

Step 7: Score game with one strike and rest gutter balls

RED
- Add a oneStrike test, on which you roll 10, then 3 and 4, then 17 zeroes.
- Assert that the score should equal 24.
- Watch the test fail.

GREEN
- Add a new if to the beginning of the for loop, testing if rolls[firstInFrame] equals 10. Place an else before the next if.
- In the body of the new if, increment the score by 10 plus rolls[firstInFrame + 1] + rolls[firstInFrame + 2] then increment firstInFrame by 1.
- Watch the test pass.

REFACTOR
- Extract the conditional test for a strike into an isStrike method (don't fold parameters).
- Extract the rolls[firstInFrame + 1] + rolls[firstInFrame + 2] part of the strike conditional branch into a nextTwoBallsForStrike method.

- Extract the `rolls[firstInFrame + 2]` part of the spare conditional branch into a `nextBallForSpare` method (don't process duplicates).
- Extract the `rolls[firstInFrame] + rolls[firstInFrame + 1];` part of the normal frame part into a `twoBallsInFrame` method.

Step 8: Score a perfect game

RED
- Add a `perfectGame` test, on which you roll twelve 10s.
- Assert that the score equals 300.
- Watch the test actually pass.

GREEN
- We're done!

REFACTOR
- Nothing to do.