Noname manuscript No.
(will be inserted by the editor)

Increasing Validity Through Replication: An Illustrative TDD Case

Adrian Santos · Sira Vegas · Fernando Uyaguari · Oscar Dieste · Burak Turhan · Natalia Juristo

Received: 01/03/18 / Accepted: -

Appendix A: BSK

The objective is to develop an application that can calculate the score of a single bowling game using TDD. There is no graphical user interface. You will only work with objects and JUnit test cases in this assignment. You will not need a main method. The application's requirements are divided into a set of user stories, which serve as your to-do list. You should be able to incrementally develop a complete solution without an upfront comprehension of all the game's rules. Do not read ahead, and handle the requirements one at a time in the order provided. Solve the problem using TDD, starting with the first story's requirement. Remember to always lead with a test case, taking hints from the examples provided. Only when you are done with a story, move on to the next one. A story is done when you are confident that your program correctly implements all the functionality stipulated by the story's requirement, meaning all of your test cases for that story and all of the test cases for the previous stories pass. You may need to tweak your solution as you progress towards more advanced requirements.

1. Frame Each turn of a bowling game is called a frame. Ten pins are arranged in each frame. The goal of the player is to knock down as many pins as possible in each frame. The player has two chances, or throws, to do so. The value of a throw is given by the number of pins knocked down in that throw.

Requirement: Define a frame as composed of two throws. The first and second throws should be distinguishable.

Example: [2, 4] is a frame with two throws, in which two pins were knocked down in the first throw and four pins were knocked down in the second.

2. Frame Score An ordinary frame's score is the sum of its throws.

Requirement: Compute the score of an ordinary frame.

Examples: The score of the frame [2, 6] is 8. The score of the frame [0, 9] is 9.

3. Game A single game consists of 10 frames.

Adrian Santos
M3S-ITEE University of Oulu, Finland
E-mail: adrian.santos.parrilla@oulu.fi

Sira Vegas, Oscar Dieste, Natalia Juristo
Escuela Técnica Superior de Ingenieros Informáticos, Universidad Politécnica de Madrid, Spain
E-mail: svegas/odieste/natalia@fi.upm.es

Fernando Uyaguari
ETAPA, Ecuador
E-mail: fuyaguari01@gmail.com

Burak Turhan
Department of Computer Science, Brunel University London, England
E-mail: burak.turhan@brunel.ac.uk

Requirement: Define a game, which consists of 10 frames.

Example: The sequence of frames [1, 5] [3, 6] [7, 2] [3, 6] [4, 4] [5, 3] [3, 3] [4, 5] [8, 1] [2, 6] represents a game. You will reuse this game from now on to represent different scenarios, modifying only a few frames each time.

4. Game Score The score of a bowling game is the sum of the individual scores of its frames.

Requirement: Compute the score of a game.

Example: The score of the game [1, 5] [3, 6] [7, 2] [3, 6] [4, 4] [5, 3] [3, 3] [4, 5] [8, 1] [2, 6] is 81.

5. Strike A frame is called a strike if all 10 pins are knocked down in the first throw. In this case, there is no second throw. A strike frame can be written as [10, 0]. The score of a strike equals 10 plus the sum of the next two throws of the subsequent frame.

Requirement: Recognize a strike frame. Compute the score of a strike. Compute the score of a game containing a strike.

Examples: Suppose [10, 0] and [3, 6] are consecutive frames. Then the first frame is a strike and its score equals $10 + 3 + 6 = 19$. The game [10, 0] [3, 6] [7, 2] [3, 6] [4, 4] [5, 3] [3, 3] [4, 5] [8, 1] [2, 6] has a score of 94.

6. Spare A frame is called a spare when all 10 pins are knocked down in two throws. The score of a spare frame is 10 plus the value of the first throw from the subsequent frame.

Requirement: Recognize a spare frame. Compute the score of a spare. Compute the score of a game containing a spare frame.

Examples: [1, 9], [4, 6], [7, 3] are all spares. If you have two frames [1, 9] and [3, 6] in a row, the spare frame's score is $10 + 3 = 13$. The game [1, 9] [3, 6] [7, 2] [3, 6] [4, 4] [5, 3] [3, 3] [4, 5] [8, 1] [2, 6] has a score of 88.

7. Strike and Spare A strike can be followed by a spare. The strike's score is not affected when this happens.

Requirement: Compute the score of a strike when it is followed by a spare. Compute the score of a game with a spare following a strike.

Examples: In the sequence [10, 0] [4, 6] [7, 2], a strike is followed by a spare. In this case, the score of the strike is $10 + 4 + 6 = 20$, and the score of the spare is $4 + 6 + 7 = 17$. The game [10, 0] [4, 6] [7, 2] [3, 6] [4, 4] [5, 3] [3, 3] [4, 5] [8, 1] [2, 6] has a score of 103.

8. Multiple Strikes Two strikes in a row are possible. You must take care when this happens for the computation of the first strike's score requires the values of throws from two subsequent frames.

Requirement: Compute the score of a strike that is followed by another strike. Compute the score of a game with two strikes in a row.

Examples: In the sequence [10, 0] [10, 0] [7, 2], the score of the first strike is $10 + 10 + 7 = 27$. The score of the second strike is $10 + 7 + 2 = 19$. The game [10, 0] [10, 0] [7, 2] [3, 6] [4, 4] [5, 3] [3, 3] [4, 5] [8, 1] [2, 6] has a score of 112.

9. Multiple Spares Two spares in a row are possible. The first spare's score is not affected when this happens.

Requirement: Compute the score of a game with two spares in a row.

Example: The game [8, 2] [5, 5] [7, 2] [3, 6] [4, 4] [5, 3] [3, 3] [4, 5] [8, 1] [2, 6] has a score of 98.

10. Spare as the Last Frame When a game's last frame is a spare, the player will be given a bonus throw. However, this bonus throw does not belong to a regular frame. It is only used to calculate the score of the last spare.

Requirement: Compute the score of a spare when it is the last frame of a game. Compute the score of a game when its last frame is a spare.

Example: The last frame in the game [1, 5] [3, 6] [7, 2] [3, 6] [4, 4] [5, 3] [3, 3] [4, 5] [8, 1] [2, 8] is a spare. If the bonus throw is [7], the last frame has a score of $2 + 8 + 7 = 17$. The game has a score of 90.

11. Strike as the Last Frame When a game's last frame is a strike, the player will be given two bonus throws. However, these two bonus throws do not belong to a regular frame. They are only used to calculate the score of the last strike frame.

Requirement: Compute the score of a spare when it is the last frame of a game. Compute the score of a game when the last frame is a strike.

Example: The last frame in the game [1, 5] [3, 6] [7, 2] [3, 6] [4, 4] [5, 3] [3, 3] [4, 5] [8, 1] [10, 0] is a strike. If the bonus throws are [7, 2], the last frame's score is $10 + 7 + 2 = 19$. The game's score is 92.

12. Bonus is a Strike Further bonus throws are not granted when a game's last frame is a spare and the bonus throw is a strike.

Requirement: Compute the score of a game in which the last frame is a spare and the bonus throw is a strike.

Example: In the game [1, 5] [3, 6] [7, 2] [3, 6] [4, 4] [5, 3] [3, 3] [4, 5] [8, 1] [2, 8], the last frame is a spare. If the bonus throw is [10], the game's score is 93.

13. Best Score A perfect game consists of all strikes (a total of 12, including the bonus throws), and has a score of 300.

Requirement: Check that the score of a perfect game is 300.

Example: A perfect game looks like [10, 0] [10, 0] [10, 0] [10, 0] [10, 0] [10, 0] [10, 0] [10, 0] [10,0] [10, 0] with bonus throws [10, 10]. It's score is 300.

14. Real Game

Requirement: Check that the score of the game [6, 3] [7, 1] [8, 2] [7, 2] [10, 0] [6, 2] [7, 3] [10, 0] [8, 0] [7, 3] [10] is 135.

Congratulations, you are done!

Appendix B: Mars Rover API

- Develop an API that moves a rover around on a grid.
- You are given the initial starting point (x,y) of a rover and the direction (N,S,E,W) in which it is facing.
- The rover receives a character array of commands
- Implement commands that move the rover forward/backward (f,b)
- Implement commands that turn the rover left/right (l,r)
- Implement wrapping from one edge of the grid to another (planets are spheres after all)
- Implement obstacle detection before each move to a new square. If a given sequence of commands encounters an obstacle, the rover moves up to the last possible point and reports the obstacle.
- Example: The rover is on a 100x100 grid at location (0, 0) and facing NORTH. The rover is given the commands ffrff and should end up at (2,2).

Appendix C: Sudoku

Sudoku is a game with a few simple rules, where the goal is to place nine sets of positive digits (19) into the cells of a fixed grid structure. A valid Sudoku solution should conform to the following rules:

- A cell in a Sudoku game can only store positive digits, i.e. 1...9.
- A "sub-grid" is a 3x3 arrangement of cells.
- All digits appear only once in a sub-grid, i.e., they cannot be repeated.
- The Sudoku board (or global grid) consists of a 3x3 arrangement of sub-grids, yielding a 9x9 arrangement of cells.
- A digit can appear only once in the rows of the global grid.
- A digit can appear only once in the columns of the global grid.

Your task is to check the validity of a given solution for a Sudoku game:

1. You should read the candidate solution from a string variable, which would have been displayed as below, when printed on the screen:

```
1 2 3 4 5 6 7 8 9
9 1 2 3 4 5 6 7 8
8 9 1 2 3 4 5 6 7
7 8 9 1 2 3 4 5 6
6 7 8 9 1 2 3 4 5
5 6 7 8 9 1 2 3 4
4 5 6 7 8 9 1 2 3
3 4 5 6 7 8 9 1 2
2 3 4 5 6 7 8 9 1
```

2. Check whether the provided string follows the correct format (i.e., 9 lines with 9 entries in each line).
3. Check the validity of the candidate solution against the rules listed above.
4. Throw "CustomSudokuException" for all error cases including but not limited to: wrong format for a solution string.
5. Implement the functionality in your program to return a string message on the validity of the solution:
   - If it is valid, the following message shall be displayed: "This is a valid solution", followed by the solution itself (as above).
   - If the solution is not valid, you shall return a failure message, indicating the reason why it is not valid, e.g., "9 appears more than once in row 5" (Assume that the lower left corner of the grid has the coordinates (1, 1)). Again, you shall display the solution after this error message.

Appendix D: Music Fone

MusicPhone is an application that runs on a GPS-enabled, MP3-capable mobile phone. MusicPhone will make recommendations for artists that the user may like and find upcoming concert events for artists using data gathered from the Last.FM website. The goal of the following tasks is to implement the necessary logic.

Task A: Ramp-up

A1. Run the project (Select the project, right click and select Run As -> Run Configurations. From the configurations, select MusicPhone from C/C++ Application) and see three UI windows appear. The Player and GPS UIs are complete. The Recommender window is just a skeleton. You will implement most of the required functionality in the Recommender class.

A2. Run the project with Unit Test configuration (Select the project, right click and Run As->Run Configurations. From the configurations, select MusicPhone UnitTest from C/C++ Unit) and view the green bar. Check the structure of the sample test in SmokeTest.cpp to get familiar with the application.

A3. Take 5-10 minutes to read through the information in the provided documentation.

Task B: Compute distance to a concert

The user will see how far away upcoming concert events are for a particular artist based on the user's current GPS position and the position of the concert venue. Implement the Distance class in `commons.dataClasses` and a public method with the signature `public static double computeDistance (GeoPoint, GeoPoint, String)` in this class to calculate the great-circle distance between two geo-coded positions. A geo-coded position is represented by a `GeoPoint` object that specifies a latitude and longitude in degrees. The method computes the great-circle distance between two points in either kilometres ("km") or miles ("mi") as specified by the third parameter. This parameter can be lower case, upper case or a combination of both. Valid latitude values are between -90 and 90 degrees; valid longitude values are between -180 and 180 degrees. Sometimes a `GeoPoint` has an invalid latitude or longitude. In these cases, the distance returned should be -1.

Geopoint$_1$		Geopoint$_2$		*units*	**Great-circle distance** (in *units*)
LatR_1 LatD_1	LonR_1 LonD_1	LatR_2 LatD_2	LonR_2 LonD_2		
0 0	*0* 0	*0* 0	*0* 0	km	0
0 0	*0* 0	*1.047* 60	*0* 0	km	6671.70
0 0	*0* 0	*6.283* 360	*6.283* 360	mi	-1
0 0	*0* 0	*1.047* 60	*0* 0	mi	4145.60
0.630 36.12	*-1.513* -86.67	*0.592* 33.94	*-2.066* -118.40	mi	1793.55

Numbers in roman type are in degrees.
Numbers in italics are in radians

The formula for the great-circle distance is:

lonD= longitude in degrees
latD= latitude in degrees
$lonR$= longitude in radians
$latR$= latitude in radians
radians=(degrees x π)/100

$$a = \sqrt{\sin^2(\frac{\Delta LatR}{2}) + \cos(LatR_1)x\cos(LatR_2)x\sin^2(\frac{\Delta LonR}{2})}$$

Great circle distance $= 2x\arcsin(min(1.0, a))xRadius$
where $Radius$ is the Earth's radius in kilometres (6371.01) or in miles (3958.76):

Task C: Find concerts for an artist

Implement the `getDestinationsForArtist` method of the `Recommender` class. When the user clicks on an artist's name in the list of recommended artists, the UI calls the `getDestinationsForArtist` method to obtain a list of destinations, where each destination contains the information on an upcoming concert together with the distance to the concert's location from the user's current position. The user's current position is provided by the `GPS` class. If the concert's location has an invalid `GeoPoint`, then the distance should be marked as -1. You will need to use `IConnector's GetConcertForArtist` method to implement this task.

Task D: Recommend artists

Implement the `getRecommendations` method of the `Recommender` class. MusicPhone recommends artists based on the favourite artists of Last.FM users who are the top fans of the artist currently playing on the user's player. MusicPhone recommends the 20 most-frequently-appearing artists

among the fans' top artists (as in "people who like this artist also like these other artists..."). `getRecommendations` should return a list of `Recommendation` objects. Each `Recommendation` object consists of an artist's name (artist) and the frequency of occurrence (`fanCount`) of that artist. You will need to use `IConnector's getTopFansForArtist` and `getTopArtistsByFan` methods to implement this method. *Example:* Suppose the currently playing artist is *Cher*. Suppose *John*, *Sally*, and *Tom* are the only top fans of Cher. If *Tom* and *John* both like *U2* as well, then `getRecommendations` should return a list that includes a recommendation containing *U2* with a `FanCount` of 2.

Task E: Compute an itinerary for concerts

E0. Implement the `buildItineraryForArtists` method of the `Recommender` class. In the UI, the user can click on buttons that will add (Add button) or remove (Remove button) a selected artist to a list of artists who the user would like to see in concert. Whenever this list is changed, the application recalculates an itinerary of upcoming concerts by calling this method. An itinerary is a list of `Destination` objects, where each object contains the concert information and the distance to the concert's location from the previous destination. The first destination's distance in the list is the distance from the user's current position. The itinerary must be chronologically ordered according to the start date of the concerts that it includes. Address and satisfy the constraints *one by one* in any order:

E1. The itinerary starts with the concert having the earliest start date for any of the selected artists.

E2. An artist has at most one concert in the itinerary (some artists may have no concerts, however).

E3. No two concerts in the itinerary may take place on the same day.

E4. If multiple artists have concerts on the same day, the itinerary includes only the concert that is closest in distance to the previous destination on the list.

E5. If the same artist has multiple concerts on the same day, the itinerary includes only the concert that is closest in distance to the previous destination on the list.

MusicPhone Project layout

app - Package containing the UI resources and classes that provide data from Last.FM.

commons - Package containing the data models, interfaces and logic modules used by the MusicPhone UI and tests. *You will implement the missing logic inside this project.* You may **not** change any of the interfaces or the `DeviceManager`. You may extend the behaviour of the `Recommender`, but without changing its existing behaviour or its interface. You may add one or more classes here if required.

commons.dataClasses - Package containing the basic classes representing the entitites present in MusicPhone.

commons.interfaces - Package containing the interfaces that are implemented by the different parts of this system. You should not change any of these.

commons.xmlData - Package containing the dump of Last.FM response for the API necssary to solve the task in XML format. These are useful for testing purposes.

commons.dataConnectors - Package containing the implementation of IConnector (LastFmXmlConnector) methods to parse the Last FM XML files. You may want to instantiate this class for testing purposes.

gps, **player**, and **recommender** - Packages containing the UIs for the application main components. The UI is bounded to a class (e.g., GpsUI.java and Gps.java) which is the implementation of the interfaces present in commons.interfaces.

What you should know before you start

Initialization of components

The application project creates specific instances of `IPlayer`, `IGps`, and `IRecommender` objects. These instances persist when the application is running. The application will set the `Connector` property of the `Recommender` object to an instance of `LastFmXmlConnector`, which implements the `IConnector` interface.

The IConnector interface

Defines access to XML data from Last.FM. The `IConnector` class you need to test your implementation with preloaded XML data is `LastFmXmlConnector`. This class has a 0-argument constructor. To access the XML data from `Commons`, use the `Connector` property of the `Recommender class`.

XML data for testing

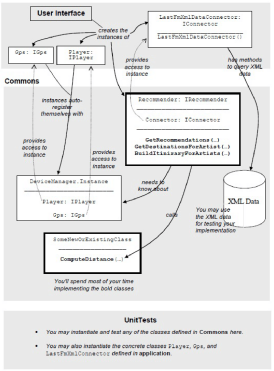
Located in the XmlData folder of the **Commons** project. The `LastFmXmlConnector` class accesses the files in this folder.

DeviceManager.Instance

Provides singleton access to instances of the `IPlayer` and `IGps` objects. When these objects are instantiated by the application, they register with the `DeviceManager`.

Architecture

Review the block diagram on the next page.



Appendix E: Residuals LMM for QLTY