**BSK - Bowling Score Keeper**

# **Goal**

BSK (Bowling Score Keeper) is an API to calculate the score of a bowling game.

Each turn of a bowling game is called a frame. A (bowling) game consists of 10 frames. A game can have up to two bonus throws.

So far, you do not need to know more. Further details will be given in the User Stories section.

# **Instructions**

**READ** the instructions carefully.

**FORK** the repository and make sure your forked repository is **PUBLIC**. Then, **CLONE** the repository and **IMPORT** the project into Eclipse.

You are asked to **DEVELOP** BSK by following **TDD**. In particular, you are asked to implement, according to the provided user stories, several methods of the classes **Frame** and **Game** (see the Eclipse project).

You **CANNOT:**

* change the signature (i.e., method name, parameter types, and return type) of the provided methods;
* move the provided methods to other classes;
* change the name of the provided classes.

You **CAN** add fields, methods (e.g., methods used by tests to set up the fixture or methods used by the provided methods), or even classes (including other test classes), as long as you comply with the provided API (see the API Usage section and the Eclipse project).

You **DO NOT** need to develop a GUI.

While developing BSK, you **CAN** use the internet to consult Java APIs (e.g., the API of the String class), Java tutorials (e.g., how Java regex works), and sites to translate from English into Italian. However, you **CANNOT** use the internet to communicate with others.

The BSK requirements are divided into a set of **USER STORIES**, which serve as a to-do list.

You should be able to incrementally develop BSK without an upfront comprehension of all the BSK requirements. **DO NOT** read ahead, and handle the requirements (i.e., specified in the user stories) one at a time in the provided order. Develop BSK by starting from the first story’s requirement. When a story is **IMPLEMENTED**, move on to the **NEXT** one. A story is implemented when you are confident that your program correctly implements all the functionality stipulated by the story’s requirement. This implies that all your test cases for that story and all the test cases for the previous stories pass. You may need to review your program as you progress towards more advanced requirements.

Each time you see the JUnit green bar (i.e., end of the **GREEN** phase and end of the **REFACTOR** one), **COMMIT**.

If you need to handle error situations (including situations unspecified by the user stories), throw a **BowlingException**. If a method does not have a throw declaration, you can add it.

At the end of the task, **PUSH** your repository on Bitbucket.

# **API Usage**

Take some minutes to understand, in broad terms, how the API works (see also the JavaDoc comments of the provided methods in the Eclipse project). If you do not fully understand the API, do not worry because further details will be given later in the User Stories section.

A typical API usage follows.

**// It initializes an empty bowling game.**

**Game game = new Game();**

**// It adds 10 frames to this bowling game.**

**game.addFrame(new Frame(1, 5));**

**game.addFrame(new Frame(2, 5));**

**game.addFrame(new Frame(1, 1));**

**game.addFrame(new Frame(4, 2));**

**game.addFrame(new Frame(8, 0));**

**game.addFrame(new Frame(2, 3));**

**game.addFrame(new Frame(1, 3));**

**game.addFrame(new Frame(1, 6));**

**game.addFrame(new Frame(2, 0));**

**game.addFrame(new Frame(10, 0));**

**// It sets the first bonus throw.**

**game.setFirstBonusThrow(4);**

**// It sets the second bonus throw.**

**game.setSecondBonusThrow(5);**

**// It computes the score of this game.**

**int score = game.calculateScore();**

# **User Stories**

Remember to read and implement the user story once at a time (in the provided order).

Therefore, do not read the next user story, if the current one is not implemented yet.

Each time you see the JUnit green bar (i.e., end of the **GREEN** phase and end of the **REFACTOR** one), **COMMIT**.

## **1 - Frame**

Each turn of a bowling game is called a frame. 10 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 throws to knock down the pins. The value of a throw is given by the number of pins knocked down in that throw.

**Requirement:**

* Implement **Frame.Frame(int firstThrow, int secondThrow)** to define a frame given the pins knocked down in the first and second throws, respectively.
* Implement **Frame.getFirstThrow()** and **Frame.getSecondThrow()** to get the pins knocked down in the first and second throws, respectively.

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

## **2 - Frame score**

The score of an ordinary frame is the sum of the pins knocked down in its two throws.

**Requirement:** Implement **Frame.getScore()** to calculate the score of an ordinary frame.

**Example:** The score of the frame [2, 6] is 8.

## **3 - Game**

A single game consists of 10 frames.

**Requirement:**

* Implement **Game.Game()** and **Game.addFrame(Frame frame)** to define a game consisting of 10 frames.
* Implement **Game.getFrameAt(int index)** to get the i-th frame of the game. Note that the index ranges from 0 to 9. This implies that 0 allows returning the first frame, 1 allows returning the second frame, and so on.

**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. [1, 5] is the first frame, [3, 6] is the second frame, and so on. Note that you can reuse this game from now on to represent different scenarios by 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:** Implement **Game.calculateScore()** to calculate 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 - Spare**

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

**Requirement:**

* Implement **Frame.setBonus(int bonus)** and **Frame.getBonus()** to set and get the bonus of the frame, respectively.
* Implement **Frame.isSpare()** to determine whether, or not, a frame is spare.
* Implement **Frame.getScore()** to calculate the score of a spare frame (including the bonus).
* Implement **Game.calculateScore()** to calculate the score of a game containing a spare frame.

**Example:** Suppose that [1, 9] and [3, 6] are consecutive frames; then the first frame is spare and its score is equal to 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] (containing a spare frame) has a score of 88.

## **6 - Strike**

A frame is called a strike if all 10 pins are knocked down in the first throw of the frame. In this case, there is no second throw (i.e., its value is 0). The score of a strike frame is 10 plus a bonus equal to the sum of the next two throws of the subsequent frame.

**Requirement:**

* Implement **Frame.isStrike()** to determine whether, or not, a frame is strike.
* Implement **Frame.getScore()** to calculate the score of a strike frame.
* Implement **Game.calculateScore()** to calculate the score of a game containing a strike frame.

**Example:** Suppose that [10, 0] and [3, 6] are consecutive frames; then the first frame is strike and its score is equal to 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] (containing a strike frame) has a score of 94.

## **7 - Strike and spare**

A strike frame can be followed by a spare frame. When this happens, the score of the strike frame is not affected by the score of the spare frame.

**Requirement:** Implement **Game.calculateScore()** to calculate the score of a game containing a strike frame followed by a spare frame.

**Example:** Suppose that [10, 0] [4, 6] [7, 2] are consecutive frames; then the score of the strike frame is 10+4+6=20 while the score of the spare frame 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**

A strike frame can be followed by another strike frame. When this happens, the computation of the score of the first strike frame requires the values of the throws from the two subsequent frames.

**Requirement:** Implement **Game.calculateScore()** to calculate the score of a game containing a strike frame followed by another strike frame.

**Example:** Suppose that [10, 0] [10, 0] [7, 2] are consecutive frames; then the score of the first strike frame is 10+10+7=27 while the score of the second strike frame 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**

A spare frame can be followed by another spare frame. When this happens, the score of the first spare frame is not affected by the score of the second spare frame.

**Requirement:** Implement **Game.calculateScore()** to calculate the score of a game containing a spare frame followed by another spare frame.

**Example:** Suppose that [8, 2] [5, 5] [7, 2] are consecutive frames; then the score of the first spare frame is 8+2+5=15 while the score of the second spare frame is 5+5+7=17. 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 the last frame of a game is spare, the player will be given a bonus throw. However, this bonus throw does not belong to a regular frame. That is, the bonus throw is used to calculate the score of the last spare frame.

**Requirement:**

* Implement **Game.setFirstBonusThrow(int firstBonusThrow)** and **Game.getFirstBonusThrow()** to set and get a bonus throw, respectively.
* Implement **Game.calculateScore()** to calculate the score of a game when the last frame is 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 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 the last frame of a game is strike, the player will be given two bonus throws. However, these bonus throws do not belong to a regular frame. That is, the bonus throws are used to calculate the score of the last strike frame.

**Requirement:**

* Implement **Game.setSecondBonusThrow(int secondBonusThrow)** and **Game.getSecondBonusThrow()** to set and get the second bonus throw, respectively.
* Implement **Game.calculateScore()** to calculate the score of a game when the last frame is 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 strike. If the bonus throws are [7, 2], the last frame has a score of 10+7+2=19. The game has a score of 92.

## **12 - Best score**

A perfect game consists of all strikes (i.e., a total of 12 strikes, one for each regular frame plus two strikes as the bonus throws) and has a score of 300. Note that the computation of the score of the ninth strike frame requires the values of the first throw of the tenth frame and the first bonus throw.

**Requirement:** Implement **Game.calculateScore()** to calculate the score of a perfect game.

**Example:** The game [10, 0] [10, 0] [10, 0] [10, 0] [10, 0] [10, 0] [10, 0] [10, 0] [10, 0] [10, 0] with [10, 10] as the bonus throws has a score of 300.