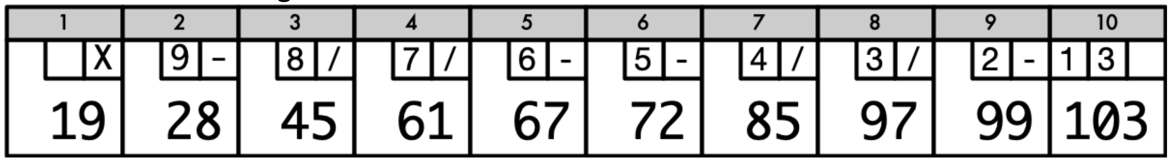
Bowling Scoreboard Challenge
COSC 3327 Algorithms and Data Structures
Due: See Canvas

Bowling

Bowling is a game in which players take turns rolling a ball down an alley, trying to knock down pins. A complete game consists of ten frames per player. Scoring a game correctly involves, among other things, being able to identify strikes, spares, and the tenth frame. Here's what a bowling scoreboard looks like:



1	2	3	4	5	6	7	8	9	10
X	9 -	8 /	7 /	6 -	5 -	4 /	3 /	2 -	1 3
19	28	45	61	67	72	85	97	99	103

Notice that a strike is symbolized by an 'X', while a spare is symbolized by a '/'. Also, a roll that knocks down zero pins is denoted by the symbol '-', not a '0'.

Representation

The representation is based on the observation that a bowler's score can be completely determined by observing how many pins the bowler knocks down for each roll. The pinsKnockedDownArray (see SinglePlayerBowlingScoreboardImpl) is used to record the results of each of the bowler's rolls. Since the maximum number of rolls per game is 21 for a single player, if a bowler bowled a perfect game (12 strikes) then pinsKnockedDownArray would contain [10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0] (i.e. twelve 10's followed by nine 0's) and rollCount would equal 12. Not all roll sequences are legal, for example, [10, 9, 4, 5, 8, 10, 0, 3, 9, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] does not correspond to a properly played bowling game.

Assignment

Your assignment is to implement the SinglePlayerBowlingScoreboard interface (see below) and the AssignmentMetaData interface, make your preconditions executable (where possible), and create JUnit4 test cases of your own. You should make your preconditions executable and create your test cases **early** in your development process! Your code must use the representation as described in the last section. Note that the interfaces SinglePlayerBowlingScoreboard and AssignmentMetaData as well as the test cases are in the SinglePlayerBowlingScoreboard_NoSparesStrikes_NoTenth_STARTER_KIT.zip on Canvas.

Deliverables

Post a .zip file to the Canvas Assignment containing:

- SinglePlayerBowlingScoreboardImpl_*LastName*.java (in package ‘bowling’)
- Any other supporting .java files (use _*LastName*.java suffix and put supporting files in package ‘bowling’)

Rules

- Fix getScoreboardDisplay() if needed in the Skeleton if you want to use it.
- Fix the executable preconditions for getMark() in the Skeleton if necessary.
- Ensure that I will be able to compile and run your code.
- Your implementation must handle partial games as well as complete games.
- You cannot change the interface, preconditions, or postconditions.
- You cannot add any additional instance variables to your SinglePlayerBowlingScoreboardImpl. Use only those shown in the partial implementation given above.
- Make sure you put strikes in the correct box!
- Make sure you use the correct marks for strikes, spares, and -’s.
- Implement a `toString()` in SinglePlayerBowlingScoreboardImpl that displays the player’s game in traditional bowling format. (I will not test your toString()).
- Write some of your own test cases. What do the distributed test cases not test?
- Do not hand in your first attempt!!

Hints

- If a method is multiple pages long, you are doing something wrong.
- Be careful, ball number and box index don't always match up
- Be careful, `getCurrentFrame()` is not always the same as (rollCount/2)
- There is a difference between a 0 and an empty in box 3 of the tenth frame

Steps To Help Manage Complexity

1. Write your own test cases
    - Helps uncover/define/refine fundamental concepts and behaviors
    - Ask customer (i.e., Dr. Kart) for any clarification
2. Develop/evolve helper functions
3. Have an implementation plan: What helper methods will you invent and use?
    - `getArrayIndexOfFirstBall(int frameNumber)`
    - `getPinCount(int frameNumber)`
    - `isStrike(int frameNumber)`
    - `isSpare(int frameNumber)`
    - `getBallNumber(int arrayIndex)`
4. Pick good names
5. As you begin implementing, focus on getting the easy parts of your program working. Ask me about:
    - `assert frameNumber < 10: “frameNumber = ” + frameNumber + “>= 10!”;`

SinglePlayerBowlingScoreboard.java

```
package bowling;

public interface SinglePlayerBowlingScoreboard
{
	public String getPlayerName();

	public boolean isGameOver();

	public int getScore(int frameNumber);

	public Mark getMark(int frameNumber, int boxIndex);

	//part of pre: !isGameOver()
	//part of post: 1 <= rv <= 10
	public int getCurrentFrame();

	//part of pre: !isGameOver()
	//part of post: 1 <= rv <= 3
	//part of post: frameNumber < 10 ==> rv <= 2
	public int getCurrentBall();

	//part of pre: 0 <= pinsKnockedDown <= 10
	//much more here...
	public void recordRoll(int pinsKnockedDown);

	//Ex: 9-  44  9/  61  9-  1-  --   X  7/
	//Stan 9  17  33  40  49  50  50  70
	public String toString();
}
```

SinglePlayerBowlingScoreboard (Partial) Implementation

```
package bowling;

public class SinglePlayerBowlingScoreboardImpl implements SinglePlayerBowlingScoreboard
{
   private static final int MAXIMUM_ROLLS = 21;      //Maximum rolls in a one player game
   private String playerName;
   private int[] pinsKnockedDownArray = new int[MAXIMUM_ROLLS];
   private int rollCount = 0;

   public SinglePlayerBowlingScoreboardImpl(String playerName)
   {
        assert playerName != null : "playerName is null!";
        this.playerName = playerName;
   }
   .
   .
   .
}
```

Mark.java

```
   package bowling;
   import java.util.HashMap;
   import java.util.Map;

   public enum Mark {
        ZERO("-"),ONE("1"),TWO("2"),THREE("3"),FOUR("4"),FIVE("5"),SIX("6"),SEVEN("7"),EIGHT("8"),
        NINE("9"),STRIKE("X"),SPARE("/"),EMPTY(" ");

        private final String stringRepresentation;
        private Mark(String stringRepresentation) {
                        this.stringRepresentation = stringRepresentation;
        }

        private static Map<Integer, Mark> integerToMarkMap;
        static
        {
                integerToMarkMap = new HashMap<Integer, Mark>();
                integerToMarkMap.put(0, ZERO);
                integerToMarkMap.put(1, ONE);
                integerToMarkMap.put(2, TWO);
                integerToMarkMap.put(3, THREE);
                integerToMarkMap.put(4, FOUR);
                integerToMarkMap.put(5, FIVE);
                integerToMarkMap.put(6, SIX);
                integerToMarkMap.put(7, SEVEN);
                integerToMarkMap.put(8, EIGHT);
                integerToMarkMap.put(9, NINE);
        }

       //part of pre: 0 <= pinCount <= 9
       //part of post: rv != null
       public static Mark translate(int pinCount)
       {
        assert integerToMarkMap.containsKey(pinCount) : "pinCount not in the set integerToMarkMap.keySet() = "
   + integerToMarkMap.keySet() + "! : pinCount = " + pinCount;
        return integerToMarkMap.get(pinCount);
       }

       public String toString()
       {
        return stringRepresentation;
       }
   }
```