Name: _____ **Date:** _____ **Class:** _____

The Bowling Scorer

You are designing software for an automatic scoring system in a bowling alley. This system consists of a sensor that determines how many pins a bowler has knocked down, a class that keeps track of a bowler's results, and a display system that shows bowlers' scores. Right now, you are working on the BowlingGame class, which records and stores a single player's scores through a bowling game.

Here are some things you might need to know about the game of bowling:

- A bowling game consists of 10 *frames*. Each frame contains one set of 10 pins, which the bowler tries to knock down.
- In each frame, a bowler gets to roll the ball (up to) twice, trying to knock down the pins.
- If the bowler knocks down all 10 pins with the first ball, this is a *strike*. The bowler does not roll a second ball that frame, since no pins remain standing.
- Otherwise, the bowler rolls a second ball, trying to knock down any pins not knocked down by the first ball. If the bowler knocks down all the remaining pins with the second ball, this is called a *spare*.
- If any pins remain standing after two balls, then the bowler moves to the next frame anyway.
- The tenth frame has special rules:
  - In some cases, the bowler is allowed to bowl three balls in the tenth frame.
  - If the bowler gets a spare in the tenth frame, then the pins are reset and the bowler gets to bowl one extra ball (still within the tenth frame) for extra points.
  - If the bowler gets a strike in the tenth frame, then the pins are reset and the bowler gets to bowl *two* extra balls at those pins. If the bowler gets *another* strike, the pins are reset again for the bowler's third (and final) ball. If the second ball is not a strike, though, the pins are not reset, and the bowler's third ball aims only at whatever pins were not knocked down by the second ball.
  - If the bowler does not get a strike or spare in the tenth frame, then the bowler gets no extra balls and the game is over.

Based on these rules, the BowlingGame class needs to include the following methods:

- A method called getCurrentFrame, which returns the frame that the player is in. This starts out at 1 and increases to 10. When the game is over, this method should return 11.
- A method called getCurrentBall, which returns the current ball the player is on within the current frame. This should normally return either 1 or 2, but it might return 3 during frame 10 (as noted above).
- A method called scoreBall, which tells the BowlingGame that the bowler rolled a ball, and takes in the number of pins knocked down by that ball as a parameter. This method does not return a value, but it stores the number of pins scored with that ball, and it updates the current frame and/or ball appropriately.
- A method called getBallScore. This method takes two parameters (a frame number and a ball number) and returns the number of pins scored in the given frame and ball.
- A method called isStrike, which takes a frame number as a parameter and returns true if a strike was scored in that frame or false otherwise.
- A method called isSpare, which works like isStrike, but instead returns true if a spare was scored in the given frame.

A skeleton version of the BowlingGame class is given below. Note that none of the class' method bodies have been written yet. You will fill in the methods for this class later, *after* creating some tests for the methods.

```
public class BowlingGame
{
    public BowlingGame()
    {
    }

    public int getCurrentFrame()
    {
        return 0;
    }

    public int getCurrentBall()
    {
        return 0;
    }

    public void scoreBall(int pins)
    {
    }

    public int getBallScore(int frame, int ball)
    {
        return 0;
    }

    public boolean isStrike(int frame)
    {
        return false;
    }

    public boolean isSpare(int frame)
    {
        return false;
    }
}
```

On the next page, you will find a JUnit test class that can be used to test the BowlingGame class. You will need to add more tests to the test class—right now, there is only one test, which is not enough to test all the functionality required for the class.

You can add JUnit test cases to this test class for the BowlingGame class. How many tests do you think you should add to be reasonably confident that the BowlingGame class is working correctly?

```
import org.junit.Test;
import static org.junit.Assert.*;

public class BowlingGameTest
{
     public BowlingGameTest()
     {
     }

     @Test
     public void testStartGame()
     {
          BowlingGame g = new BowlingGame();
          assertEquals(1, g.getCurrentFrame());
          assertEquals(1, g.getCurrentBall());
     }
}
```