

Programming Assignment 10



Qualifying

Overview

As you know, *perspecTV* is a company that designs, creates and markets products that provide a new perspective on television. Their products make television both more interactive and more informative.

For this assignment, you must create a main class named `Qualifying` that makes use of the `ScoreCalculator` class (you developed earlier) to calculate and display the results of the qualifying round of a diving competition for a single diver at a particular event. This main class will serve as the “proof of concept” for the full `forScore` product, which will work with multiple divers and multiple events.

Specifications

`Qualifying` is the main class for the product (i.e., it has a `main()` method that is the entry point for the product). This class must construct and setup a `ScoreBoard` object (see the discussion of the existing components below), iteratively prompt the user to enter information about the diver’s performance and display that information on the `ScoreBoard`, and then shutdown. It must work for the specific qualifying dives of the JMU Invitational.

The class must comply with the following specifications:

- 1 It must have (and use appropriately) the following “class constants”:

```
public static final String[] GROUPS = {"1", "4", "1", "4", "1"};
public static final int[] HALF_SOMERSAULTS = {3, 5, 1, 7, 1};
public static final int[] HALF_TWISTS = {0, 0, 2, 0, 0};
public static final String[] POSITIONS = {"B", "B", "A", "B", "C"};
public static final boolean[] FLYINGS = {false, false, false, false, false};
```

These are conformal arrays that contain information about the five dives (see the discussion of the `Dive` object below) that are required during the qualifying round of the JMU Invitational. For example, element 0 of the four arrays contain information about dive 0.

- 2 During the “setup phase” the product must:
 - 2.1 Construct a `ScoreBoard` (see the discussion of the existing components below) with a title of "JMU Invitational" and 7 judges.
 - 2.2 Set the ad to display on the left side of the `ScoreBoard` to "DukeDashAd.png".
 - 2.3 Set the ad to display on the right side of the `ScoreBoard` to "WeatherBitsAd.png".
 - 2.4 Start the `ScoreBoard` named (using either of the `startup()` methods).
 - 2.5 Construct a `Scanner` object to use when reading responses from the user.
- 3 After the “startup phase” the product must enter the “iterations phase”. There must be exactly as many iterations as there are elements in the conformal arrays discussed above. For each iteration, the product must:
 - 3.1 Construct a `Dive` object (see the discussion of the existing components below) using the information from the conformal arrays.
 - 3.2 Prompt the user (on `System.out`) to enter the following information (in order) and then read the user’s response.
 - 3.2.1 (Prompt) "Name and Number:"
(Response) The driver’s family name (e.g., `Flintstone`) followed by a space followed by the sequence number (e.g., 1, 2, 3, 4, or 5).
 - 3.2.2 (Prompt) "Scores:"
(Response) A space-delimited list of judges scores (which will be `String` representations of `double` values, with or without decimal points).

You may assume that the user will always respond appropriately (on `System.in`) to any given prompt. In other words, you never need to re-prompt or validate the user’s responses.

 - 3.3 Call the `totalScore()` method in the `ScoreCalculator` class, passing it the scores (obtained from the user) and the difficulty of the dive (obtained from the `Dive` object’s `getDifficulty()` method).
 - 3.4 Modify the `ScoreBoard` object by calling its `showScore()` method.
- 4 After the “iteration phase” this product must enter the “shutdown” phase.
 - 4.1 Prompt the user to press the enter/return key when done.

- 4.2 Read the blank/empty line that results from the user pressing the enter/return key.
- 4.3 Call the `shutdown()` method of the `ScoreBoard` object.
- 4.4 Close the `Scanner` object.

Existing Components

The Dive Class

Another programmer at *perspecTV* has already designed, implemented, and tested a `Dive` class that encapsulates information about an individual dive. It contains the following “class constants”:

```
public static final String FORWARD = "1";
public static final String BACKWARD = "2";
public static final String REVERSE = "3";
public static final String INWARD = "4";

public static final String STRAIGHT = "A";
public static final String PIKE = "B";
public static final String TUCK = "C";
public static final String FREE = "D";
```

It also has the following explicit value constructor:

```
Dive(String group, int halfSomersaults, int halfTwists,
     String position, boolean flying)
```

where the `group` can be `Dive.FORWARD`, `Dive.BACKWARD`, `Dive.REVERSE`, or `Dive.INWARD`; `halfSomersaults` is the number of half-somersaults (e.g., 7 for a 3½); `halfTwists` is the number of half-twists (e.g., 2 for 1 full twist); `position` can be `Dive.STRAIGHT`, `Dive.PIKE`, `Dive.TUCK`, or `Dive.FREE`; and `flying` indicates whether the dive is started in the flying position or not.

It also has the following method that you will need to use:

```
public double getDifficulty()
Gets the FINA degree of difficulty associated with this Dive.
```

Finally, it has the following method that you probably won't need to use (except, perhaps, when debugging), but that is used by the `ScoreBoard` class:

```
public String getCode()
Gets the 4 or 5 character code associated with this Dive. Non-twisting dives have 4-character codes and twisting dives have 5-character codes. The code for a twisting dive consists of a 5 followed by the group, number of half-somersaults, number of half-twists, and position. The code for a non-twisting dive consists of the group, followed by a 1 or 0 (for flying or non-flying), followed by the number of half-somersaults, and position.
```

The ScoreBoard Class

Another programmer at *perspecTV* has already designed, implemented and tested the `ScoreBoard` class. This class has the following “constants”:

```
public static final int LEFT;  
public static final int RIGHT;
```

It also has the following explicit value constructor:

```
ScoreBoard(String title, int numberOfJudges)
```

It has the following methods that you probably will need to use:

```
public void setAd(String name, int side)
```

Sets the add to display on the given side of the `ScoreBoard` (where `side` can be either `ScoreBoard.LEFT` or `ScoreBoard.RIGHT`).

```
public void showScore(String familyName, String sequenceNumber,  
                      Dive dive, String[] scores, double total)
```

Displays the judges scores and total score for a particular dive performed by a particular diver.

```
public void shutdown()
```

Shut the `ScoreBoard` down.

```
public void startup()
```

Start the `ScoreBoard` at the default location on screen.

```
public void startup(int x, int y)
```

Start the `ScoreBoard` at the given location on screen (where 0,0 is the upper-left corner of the screen).

Finally, it has the following method that you probably won't need to use (except, perhaps, when debugging) but that is used by the `ScoreBoard` class:

```
public String getAd(int side)
```

Gets the name of the advertisement being displayed on the given side (`ScoreBoard.LEFT` or `ScoreBoard.RIGHT`) of the `ScoreBoard`.

An Important Change

You **must not** use the `JMUConsole`, class the `Text` class, or the `length()` method in the `Array` class for this assignment (or, indeed, any longer in this course).

Recommended Process

1. Read and understand the entire assignment.
2. Create a directory/folder for this assignment.
3. Copy `ScoreCalculator.java` into the directory you created. You may use either your implementation or the solution that was provided to you.
4. Download the [.zip file](#) containing `Dive.class`, `ScoreBoard.class` and the two advertisements (`DukeDashAd.png` and `WeatherBitsAd.png`), and **unzip them into the directory you created for this assignment.**
5. Think about the best way to implement the `Qualifying` class.
6. Implement the `Qualifying` class.
7. Test the `Qualifying` class (and debug it, if necessary).
8. Submit `ScoreCalculator.java`, and `Qualifying.java` in a file named `pa10.zip` using Autolab. Do not include any other files in the `.zip` file.

Hints

Because of the way the user's responses will be formatted, it is going to be easiest to read each line as a `String`, and then split it into parts. So, for example, assuming that a `Scanner` named `keyboard` has already been declared and constructed, and that a `String` named `line` has been declared, the easiest way to prompt for and read the diver's name information is as follows:

```
String[] nameInfo;  
System.out.println("Name and Number:"); // Prompt the user  
line = keyboard.nextLine();           // Read the response as a single String  
nameInfo = line.split(" ");           // Split the String at the space
```

Also, remember that the `totalScore()` method in the `ScoreCalculator` class is passed a `String[]` (not a `double[]`). So, the same kind of thing can be done to prompt for and read the line containing the scores.

Help with Testing

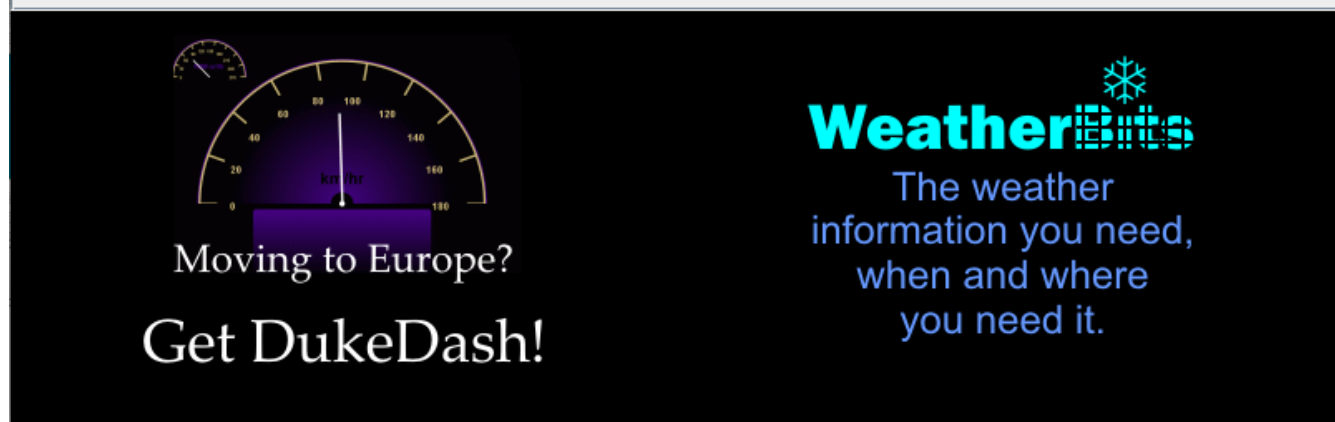
The team at *perspecTV* has created a system test that you should use to test your implementation. The user input for the test is as follows:

```
Flintstone 1
5.5 5.0 6.0 6.5 6.0 6.5 5.0
Flintstone 2
4.5 4.5 4.5 5.0 5.5 5.0 5.0
Flintstone 3
8.5 8.0 8.0 8.5 8.0 8.5 9.0
Flintstone 4
4.5 4.5 4.5 4.5 4.5 4.5 4.5
Flintstone 5
8.0 8.0 8.0 8.0 8.0 8.5 8.5
```

where the last [Enter] is omitted so that the program does not terminate.

The correct output for this test is:

JMU Invitational										
Name	Dive	Difficulty	Judge 0	Judge 1	Judge 2	Judge 3	Judge 4	Judge 5	Judge 6	Score
Flintstone, 1	103B	1.7	5.5	5.0	6.0	6.5	6.0	6.5	5.0	9.86
Flintstone, 2	405B	3.4	4.5	4.5	4.5	5.0	5.5	5.0	5.0	16.32
Flintstone, 3	5112A	2.0	8.5	8.0	8.0	8.5	8.0	8.5	9.0	16.60
Flintstone, 4	407B	3.7	4.5	4.5	4.5	4.5	4.5	4.5	4.5	16.65



The advertisement features a speedometer on the left with a needle pointing to 100 km/hr. Below it, the text reads "Moving to Europe? Get DukeDash!". On the right, there is a logo for "WeatherBites" with a snowflake icon above the word "Bites". Below the logo, the text reads "The weather information you need, when and where you need it."

Grading

Your code will first be graded by Autolab and then by the Professor. The grade you receive from Autolab is the maximum grade that you can receive on the assignment.

Autolab Grading

Your code must compile (in Autolab, this will be indicated in the section on “Does your code compile?”), and all class names and method signatures comply with the specifications (in Autolab, this will be indicated in the section on “Do your class names, method signatures, etc. comply with the specifications?”) for you to receive any points on this assignment.

Autolab will then grade your submission as follows:

Conformance to the Course Style Guide: **20 points** (All or Nothing)

Correctness: **80 points** (All or Nothing)

Manual Grading

After the due date, the Professor may manually review your code. At this time, points may be deducted for inelegant code, inappropriate variable names, bad comments, etc.