OOP3200 – Object Oriented Programming II

Java Lab 1
Logic, Arrays and Strings

Due: Week 9 (Sunday November 15, 2020) @ midnight

Value 6%

Logic, Arrays and Strings **Maximum Mark: 100**

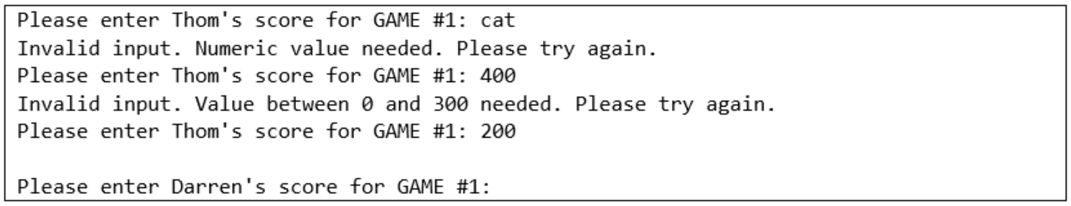
Overview In this lab you will create a **console Java program** that will have the user enter the bowling scores for **two games** played by a team of **three players**. The program will then display each of the player's **individual scores** and **average**. This program requires the use of a **two-dimensional array**.

Instructions:
Section 1: Requirements for StandardDeck Class
(70 Marks: Functionality)

1. **Getting User Input: Prompt the user.** for each of the player's scores, game by game, as shown in the **example output below**. Note that the players are listed by name rather than by number. **Hint**: solve this using an **array of strings** that holds the player names. The player's names are **your name**, **your partner's name**, and your **instructor's name**. Get all the scores for the **first game**, then all the scores for the **second game**. (20 Marks: Functionality)

2. **Getting User Input: Input and Validation.** Include domain (range) validation in your solution. The **lowest score** a bowler can have is **zero**. The **highest** is **three hundred**. If the user enters a value outside of that range, the program should display an **error message** and **re-prompt**. Validation for **non-numeric input** is also required. Bowling scores are always whole numbers. (20 Marks: Functionality).

Validation Example

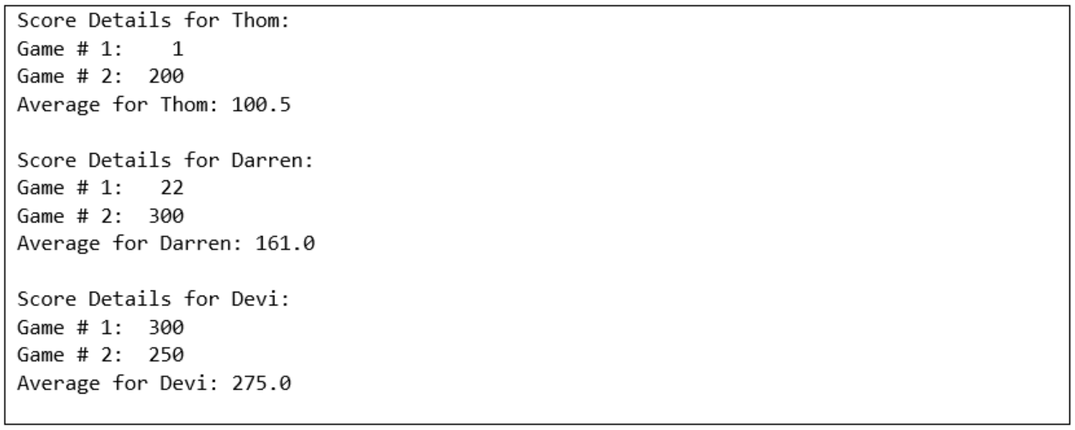


```
Please enter Thom's score for GAME #1: cat
Invalid input. Numeric value needed. Please try again.
Please enter Thom's score for GAME #1: 400
Invalid input. Value between 0 and 300 needed. Please try again.
Please enter Thom's score for GAME #1: 200

Please enter Darren's score for GAME #1:
```

3. **Processing and Output:** For each player, display his/her **individual scores** and his/her **average score**. A typical convention for calculating bowling scores is to **round the average down**, however in this project display the average to **one decimal place**. Note that you must use **nested loops** to produce this output. Match the sample output provided. Note that it shows the scores player by player. (30 Marks: Functionality).

Output Example



```
Score Details for Thom:
Game # 1:    1
Game # 2:  200
Average for Thom: 100.5

Score Details for Darren:
Game # 1:   22
Game # 2:  300
Average for Darren: 161.0

Score Details for Devi:
Game # 1:  300
Game # 2:  250
Average for Devi: 275.0
```

Section 2: Program Requirements

(10 Marks: Functionality)

1. Your `main()` function should demonstrate each of the features to provide the appropriate input, data validation and output as described above. How you do this is up to you (10 Marks: Functionality).

Section 3: General Requirements

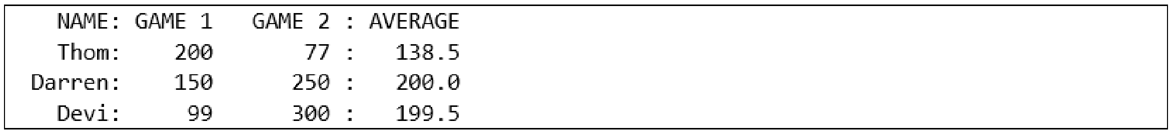
(5 Marks: Internal Documentation, 5 Marks: Version Control, 10 Marks: Demo Video)

1. Include **Internal Documentation** for your site **(5 Marks: Internal Documentation):**
    a. Ensure you include a **comment header** for your **C++ Files** that indicate: the **File name**, **Student's Name**, **StudentID, and Date,** (2 Marks: Internal Documentation).
    b. Ensure you include **method and function headers** for all of your **class methods,** and any **utility functions** (1 Marks: Internal Documentation)
    c. Ensure all your code uses **contextual variable names** that help make the files human-readable and follow the C++ style guide (1 Marks: Internal Documentation).
    d. Ensure you include **inline comments** that describe your program's functionality where appropriate. **Note:** Please avoid "over-commenting" (1 Marks: Internal Documentation)
2. Share your files on **GitHub** to demonstrate Version Control Best Practices **(5 Marks: Version Control).**
    a. Create an appropriately named GitHub Repository that you and your partner will use for this lab. Only one repository is required (1 Mark: Version Control)
    b. Your repository must include **your code** and be well structured (2 Marks: Version Control).
    c. Your repository must include **commits** that demonstrate the project being updated at different stages of development – each time a major change is implemented (2 Marks: Version Control).

3. Create a Short Video presentation on **YouTube** or another streaming provider. You must include a short **PowerPoint** (or Google Slides) Slide Deck that includes a **single slide** to start your video (10 Marks: Video)
    a. The **first** (and only) **Slide** of your Slide Deck must include **current images** of you and your partner (no avatars allowed) that are displayed appropriately on the page. You must also include your **Full Names**, **Student IDs**, the **Course Code**, **Course Name,** and your **Assignment** information. (2 Marks: video)
    b. You or your partner will **demonstrate** your program's functionality. You must show your program working properly in the console (2 Marks: Video)
    c. You or your partner will **describe** the code in your files that drives the functionality of your program (2 Marks Video).
    d. Sound for your Video must at an appropriate level so that your voices may be **clearly heard,** and your screen resolution should be set so that your program's code and console details are **clearly visible** (2 Marks: Video).
    e. Your Short Video should run no more than 5 minutes (2 Marks: Video).

Optional Features and Things to Explore

- Use **nested loops** to produce the required output in a **tabular format**, as shown below. Note that a user should be able to increase the number of games and players by a **reasonable amount** and still expect your logic to work:



```
   NAME: GAME 1   GAME 2 : AVERAGE
   Thom:    200       77 :   138.5
 Darren:    150      250 :   200.0
   Devi:     99      300 :   199.5
```

- Create one or more methods to handle different tasks (e.g. `ReadScore()`, and/or `DisplayReport()`). Avoid using class level variables by passing data to the method(s) or returning data from the method(s) as needed.

SUBMITTING YOUR WORK

Your submission should include:

1. A zip archive of your program's Project files uploaded to DCConnect.
2. A working link to your appropriately named GitHub repository
3. A working link to your demo video posted on YouTube or another streaming provider

Evaluation Criteria

Feature	Description	Marks
Functionality	Program deliverables are me and site functions are met. No errors, including submission of user inputs.	80
Internal Documentation	File headers present, including file & student names & description. Functions and classes include headers describing functionality & scope. Inline comments and descriptive variable names included and follow style guide.	5
Version Control	GitHub repo created with commit history demonstrating regular updates. 2 marks for initial commit. 2 marks for working with GitHub throughout the development process	5
Video Presentation	Your short video must demonstrate your site and describe your code. Your audio must be at an appropriate level and your screen must be clearly seen.	10
Total		**100**

This assignment is weighted **6%** of your total mark for this course.

Late submissions:

- 20% deducted for each day late.

External code (e.g. from the internet or other sources) can be used for student submissions within the following parameters:

1. The code source (i.e. where you got the code and who wrote it) must be cited in your internal documentation.
2. It encompasses a maximum of 10% of your code (any more will be considered cheating).
3. You must understand any code you use and include documentation (comments) around the code that explains its function.
4. You must get written approval from me via email.