CS 101 Midterm Exam

CS 101
Spring 2020
Midterm Exam
03/04/2020
Time Limit: 1:50 hours

- **Exam Structure:** This exam contains two parts. Part 01 is the theoretical exam. Part 02 is a practical exam. The theoretical exam part contains 10 questions (a mix of multiple-choice and descriptive questions) provided through an online portal. The practical exam part contains one or more programming requirements outlined in this document that require implementing code files to address all the requirements provided.
- **How to take this exam?** You may *not* use your books, notes, phone, and other resources on this exam. You are allowed to use a resource sheet (1 page can be 2 sided) handwritten or typed notes in order to help you to do well in this exam. You are allowed to take this exam using the lab pc or your personal laptop. You may *not* take this exam remotely unless there are prior arrangements made with the Professor. You are required to be physically present in Alden hall during the entire exam time.
- **Additional Details:** I have tried to proofread the exam carefully, but if you see something that looks like an error, please notify me immediately so that I can look at it and possibly correct it. I had tried my best to create the questions so that it is simple to understand and easy to read, but if you find the choice of words to be confusing or unclear, then you should take a break and stop by at my office as soon as possible so that I can clarify.

EXAM SHEET

1. (50 points) **PART 1:** Navigate to the online portal link below and follow the **Part 1 guidelines** provided in the portal.

   `https://forms.gle/oSte9SPWz3PpFnxg6`

2. (50 points) **PART 2:** Provide detailed implementation for the requirements outlined below:

   Accept the following GitHub invitation to create your repository for submission:

   `https://classroom.github.com/a/hD2kxigw`

   Hard Requirements: There will be full points awarded if you implement all the requirements thoroughly as indicated. Partial points will be awarded if there is any compromise done to implement any of the requirements listed below. If the code does not compile and/or throw any runtime errors, then there will be additional points taken. So be cautious to make necessary backups during the exam when you have working code and check the time regularly to make sure you finish the requirements within the allocated time period. Add necessary comments in your program files, so as to maximize your chances to receive partial credit.

   Make sure to add the following text as a comment in all your code files, including **Athletes.java** and **MovieRating.java** in the repository shared. Replace your name with **Steve Crammer**.

   This code is written in accordance with CS-101 honor code policy and exam policy outlined in the document above and all the code files submitted were completely implemented by me. - Steve Crammer

   I will not **grade** your work if you fail to add the statement above in all your code files. I will also not accept any late submission after 4:30 PM today. All your commits are logged in GitHub so please make sure to follow the protocols outlined above.

Athletes Class:

1. [**25 points**] The Athletes class contains a lot of starter code to make the implementation more interesting and enjoyable. The goal of this program is to automatically generate and fill out random values within an array. The array is a representation of the athlete's weights. The program prompts the user to type in the number of athletes. The method named `prepareData` is used to generate and fill out the array of athlete's weights. There is no need to modify the **main** method in the starter code. The program is fully functional in generating the array. However, the missing part is to find the lowest and highest weight in the array. Implement all the requirements outlined below:
   (a) Add the necessary logic inside the method named `findLow`. The method should return the lowest weight from the array of athlete's weights (available through the method input parameter `athletes`). Keep in mind that the method returns 0 as the lowest weight in the starter code. In your implementation, it is required to identify and store the lowest weight in the variable `res`. Also, it is important to make a note that the range of the weights in the athlete's array is from **150-300**. This means minimum value in the array is 150 and the maximum in value in the array is 300.
   (b) Add the necessary logic inside the method named `findHigh`. The method should return the highest weight from the array of athlete's weights (available through the method input parameter `athletes`). Keep in mind that the method returns 0 as the lowest weight in the starter code. In your implementation, it is required to identify and store the highest or largest weight in the variable `res`. Also, it is important to make a note that the range of the weights in the athlete's array is from **150-300**. This means minimum value in the array is 150 and the maximum in value in the array is 300.
2. The final output of the program should display all the values in the array and correctly display the lowest and highest weight. There is no **need** to make any changes in the starter code main method. Add in the required logic in `findLow` and `findHigh` will produce the expected output.

MovieRating Class:

1. [**25 points**] The MovieRating class contains a lot of starter code to make the implementation more interesting and enjoyable. The goal of this program is to automatically generate and fill out random values within an array, similar to the previous part. The array is a representation of the movie ratings. Unlike the previous part, here we manage a dynamic array. The dynamic array has the ability to grow big. To start with, the program initializes the array named `movie` to a capacity of 5. The method named `prepareData` is used to generate and fill out the array of movie ratings. There is no need to modify the **main** method in the starter code. The program is fully functional in generating the array. However, the missing part is to have the array `movie` grow based on the user prompt. If the user specifies "y" to add more ratings then, the method named `addMoreRatings` is called. Otherwise, the program will exit. Implement all the requirements outlined below:
   (a) Add the necessary logic inside the method named `addMoreRatings`. The method should make the `movie` array bigger (add 5 to the capacity). So for example, the array should grow in the sequence [5,10,15,20,...].

   Similar to our class discussions and Lab 04, it is required to use a temp array to make the `movie` array bigger. Also, it is really important to increment the **capacity** appropriately. At any point of time, the capacity defines the total capacity of the `movie` array.

   (b) Add the necessary logic inside the method named `countOfBad`. The method should return the total number of bad movies from the array of movie ratings. The movie ratings are available through the global array `movies`. Keep in mind that the method returns 0 as the bad movie count in the starter code. In your implementation, it is required to identify and store the total number of bad movies in the variable `res`. Also, it is important to make a note that the bad movies are those with a rating **less than 35**. The range of the ratings in the `movie` array are from **0-100**. This means minimum value in the array is 0 and the maximum in value in the array is 100.

   (c) Add the necessary logic inside the method named `countOfGood`. The method should return the total number of good movies from the array of movie ratings. The movie ratings are available through the global array `movies`. Keep in mind that the method returns 0 as the good movie count in the starter code. In your implementation, it is required to identify and store the total number of good movies in the variable `res`. Also, it is important to make a note that the good movies are those with a rating **greater than 65**. The range of the ratings in the `movie` array are from **0-100**. This means minimum value in the array is 0 and the maximum in value in the array is 100.

2. The final output of the program should grow the array (if the user specifies "y") and display all the values in the array named `movie`. Next the total number of bad and good movies should be displayed. There is no **need** to make any changes in the starter code main method. Add in the required logic in `countOfGood`, `countOfBad` and `addMovieRatings` will produce the expected output.

Self Reflection:

It is required that you do a self-reflection about your exam preparation, exam performance, and provide feedback about the exam and our CS-101 class. The reflection document can be added as a markdown file or as a PDF file named `reflection`. Add a self-reflection document with the following details:

1. What did you do from your side to do well in this exam? (Feel free to comment on anything by answering the questions below:).
   (a) did you attend classes regularly?
   (b) are you attentively listening to class discussions, following the lecture content, asking follow-up questions to clarify with the Professor either in class or during office hours, and making class notes from the lecture sessions?
   (c) did you complete the reading assignments?
   (d) how much time did you spend preparing for the exam?
   (e) how long did you spend on the lab assignments?
   (f) how did you do in the skill test, labs, and class activities?
2. What did you think of this test both Part 1 and 2? (Feel free to comment on anything — fairness, length, coverage, difficulty, or anything else.) Any constructive criticism is welcome.
3. Your feedback is more valuable to take any effort to enhance your learning experience. Share your feedback about CS-101 classes and labs so far this semester? For example, answer the following questions:
   (a) What are the most helpful aspects that you like to continue doing in our classes?
   (b) What would you like to start doing more so as to further enhance your learning in our classes?