

Lab 02 Specification – A getting Started Exercise on Object-Oriented Programming 50 points

Lab Goals

- Learn to develop reusable code.
- Do a simple workout on Arrays to understand indexing.
- Practice fundamental techniques in object-oriented programming.
- A quick recap on the use of access modifiers in Java.

Suggestions for Success

- Take a look at the suggestions for successfully completing the lab assignment, which is available at:
<https://www.cs.allegheeny.edu/sites/amohan/resources/suggestions.pdf>

Learning Assignment

If you have not done so already, please read all of the relevant "GitHub Guides", available at the following website:
<https://guides.github.com/>

that explains how to use many of the features that GitHub provides. This reading will help you to understand how to use both GitHub and GitHub Classroom. To do well on this assignment, you should also read

- **GT chapter 02, 2.1 and 2.2**

Assignment Details

Now that we have discussed some basics of object-oriented programming together in the last few lectures, it is your turn. In this lab, you will practice a variety of programs to retain the knowledge of object-oriented programming that had been covered so far. This includes modifying one or more code files to implement a series of functionalities.

The lab has two parts. The first part is required to be completed by the end of the lab session. The second part is required to be completed within a one week time period. Please make sure to push your code file(s) and commit by the deadline provided in each of the following section(s).

It is required for all students to follow the honor code. Some important points from the class honor code are outlined below for your reference:

1. Students are not allowed to share code files and/or other implementation details. It is acceptable to have a healthy discussion with your peers. However, this discussion should be limited to sharing ideas only.
2. Submitting a copy of the other's program(s) is strictly not allowed. Please note that all work done during lab sessions will be an opportunity for students to learn, practice, and master the materials taught in this course. By doing the work individually, students maximize the learning and increase the chances to do well in other assessments such as skill test, exams, etc . . .

At any duration during and/or after the lab, students are recommended to team up with the Professor and the TL's to clarify if there is any confusion related to the lab and/or class materials.

Section 1: In-Lab Exercise (25 points)

Attendance is in general a required component in computer science class(es) and laboratory session(s). By attending these sessions regularly, students get an opportunity to interact with the Professor and the Technical Leaders(s). This interaction helps maximize the student's learning experience. Attendance is taken in the form of Mastery Quizze(s) and by checking the commit log of the student's In-Lab exercise submission. No student is allowed to take the Mastery Quizze(s) outside the laboratory room unless prior arrangements are done with the Professor. It is against the class honor code to violate the policy outlined above.

Part 01 - Attendance Mastery Quiz (5 points)

The Mastery Quiz is graded based on a Credit/No-credit basis. Students are required to complete the Mastery Quiz only in the laboratory room, which is in **Alden 101** before 3:30 PM. Submissions after 3:30 PM is not acceptable. Students who had completed the Mastery Quiz according to the policy outlined above will automatically receive (5) points towards their lab grade. The link to the Mastery Quiz is provided below:

<https://forms.gle/DrvN4zKZsG7Yqtb7>

Part 02 - A Simple Exercise on Object Oriented Programming (15 points)



Students are required to complete this part only in the laboratory room, which is in **Alden 101**. It is required to complete the final commit by the end of the laboratory session, which is before **4:20 PM** on the same day that the lab is given to the students. There is late submission accepted for this part, according to the late policy outlined in the course syllabus.

Write a Java program that implements the `Patient Database` program using a series of requirements outlined below.

1. **ADD** the following statement to all your code files including the file named `Patient.java` and `PatientInfo.java`.

This work is mine unless otherwise cited - Student Name

2. Let us suppose that a hospital wants to create a list of their inpatients. The information to store include:
 - (a) Patient Full Name
 - (b) Admission Date
 - (c) Disease
 - (d) Discharge Date
3. For simplicity, let us make the `Patient Database` program work for ten patients and the patient details are shown below:

Patient ID	Patient Full Name	Admission Date	Disease	Discharge Date
10011001	Alex Crowe	12/30/2019	Tuberculosis	01/03/2020
10011002	Amelia Kaur	01/02/2020	Asthma	01/07/2020
10011003	Amanda Daya	01/15/2020	Heart Attack	01/23/2020
10011004	Ben Krish	01/20/2020	Stroke	01/23/2020
10011005	Brian Miller	12/25/2019	Urinary Tract Infection	12/30/2019
10011006	Chris Miller	12/12/2019	Asthma	12/15/2019
10011007	Drew Millan	01/12/2020	Hypertension	01/20/2020
10011008	Frank Derrick	01/12/2020	Sarcoidosis	01/26/2020
10011009	Maria Stuart	01/13/2020	Lung Cancer	01/28/2020
10011010	Rosy David	01/15/2020	Diabetes	01/17/2020

Table 1: Patient Database Table

4. The starter code is provided inside the lab repository in a file named, `Patient.java` and `PatientInfo.java`. Analyze the starter code and identify the errors in the starter code. The starter code may not compile without adding the required getters and setters. Recall in class we compiled multiple files using `javac *.java` command and `javac -d classes *.java` command to store the class files in the `classes` directory.
5. First, add the required getter and setter methods to the `Patient.java` file. Please note that the getters and setters should be added to initialize all the private members in the class.
6. Next, complete the `PatientInfo.java` file by implementing an object for every patient within the main function. Please note that if there are 10 patients then the solution should include ten objects namely [p1, p2, p3, ..., p10]. This may not be an efficient solution. An efficient solution by creating an array of objects shall be explored later.
7. To avoid complexity, the patient details may be hard-coded in the program. That is, it is not required to get the patient details as user input. If you have some additional time, then it is recommended to get these inputs from the user.
8. For simplicity the `PatientInfo.java` already includes two print functions. The `printPatientHeader` function is used to print the header information such as Patient ID, Patient Name, etc. The `printPatientDetails` functions accept the `Patient` object as input and prints the data that is connected to the patients. It is not required to make any changes to these functions. These functions should be called within the main function so as to print the patient header and details as the output of the program on the console.

9. A screenshot below shows the output of the program. Please note, to minimize space, the screenshot only includes the details connected to one patient. The lab submission should include the details connected to all 10 patients.

```
Aravinds-MacBook-Pro-916:code amohan$ java PatientInfo
Patient ID      Patient Name    Admission Date  Patient Disease Discharge Date
10011001        Alex Crowe      12/30/2019      Tuberculosis    01/03/2020
Aravinds-MacBook-Pro-916:code amohan$
```

Part 03 - Pair Discussion (5 points)

Students are required to complete this part only in the laboratory room, which is in **Alden 101**. It is required to complete the final commit by the end of the laboratory session, which is before **4:20 PM** on the same day that the lab is given to the students. There is no late submission accepted for this part unless prior arrangements are done with the Professor.

Pair up with at least one of your peers. Talk to your peer and discuss with them at least one idea of how to further enhance the `PatientInfo.java` and `Patient.java` program. Each person in the pair should discuss their own idea and write a short summary to explain their idea. The other person should critic the idea of their peer by providing them with at least one strong and weak point related to the idea. The summary should also include both the original idea and as well as the critic points from the peer. Name the summary file as **pd-summary.pdf**. Here **pd** refers to pair discussion. In order to pass the submission requirements, the file should be generated using a PDF format.

ADD the following statement to all your code files including the file named `pd-summary.pdf`.

This work is mine unless otherwise cited - Student Name

After completing all the parts in Section 01, it is acceptable for students to start working on Section 2 remotely. Although it is strongly recommended to be in the laboratory room till the end of the lab session, and work on the lab along with other colleagues.

Section 2: Take Home Exercise (25 points)

Students are recommended to get started with this part in the laboratory room, by discussing ideas and clarifying with the Professor and the Technical Leader(s). All the part(s) in this section is due in **one week** time. It is acceptable to discuss high-level ideas with your peers, while all the work should be done individually. The deadline for submitting the part(s) in this section is **5th Feb 2020, 8:00 AM**. Late submission is accepted for the part(s) in this section, based on the late policy outlined in the course syllabus.

Part 01 - Transaction Analysis Program (15 points)

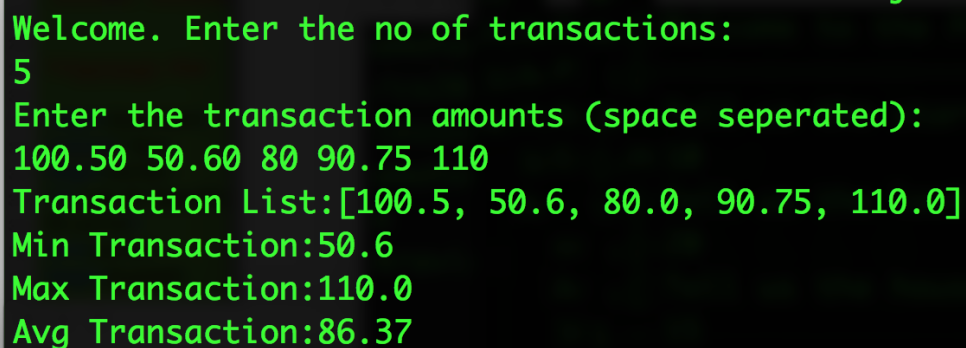


Write a Java program that implements a `Transaction Analysis` program for a bank teller using a series of requirements outlined below.

1. **ADD** the following statement to all your code files including the file named `Transaction.java` and `TransactionStub.java`.

This work is mine unless otherwise cited - Student Name

2. The starter code is provided inside the lab repository in a file named, `Transaction.java` and `TransactionStub.java`.
3. First, the program should greet the user with a welcome message. This is already implemented in the starter code file so as to get started.
4. After displaying the greeting message, the program should prompt the user to provide a list of transactions and the array `trans` is initialized based on the user-provided input. This part is already implemented in the starter code.
5. Next, prompt the user to enter a series of transaction amount(s) and load the array `trans` based on the user-provided input. This part should be implemented in the `setTrans` method in the `Transaction` class file. One way to do this is to use a while loop and load the elements in an array one at a time.
6. After loading the transaction amounts in to the array, complete the implementation in the `minTrans`, `maxTrans`, and `avgTrans` in the `Transaction` class file to find the min, max, and average from the `trans` array respectively.
7. The program should repeat the steps to guide the bank teller to load and analyze a different set of transactions. The program should exist if the teller specifies there is no more analysis needed.
8. The program should do some basic data validation procedure to correctly identify invalid inputs. For example, if the bank teller provided input other than a number as the transaction amount, then an invalid input message should be displayed on the console. In general, it is acceptable to assume that the bank teller provided input for the transaction that he/she is inputting a number ≥ 1 and less than 1 million.
9. Please note that step by step instructions to find the min, max, and average calculations are purposefully left out from this sheet. It is intended to do so, so as to challenge the students to think, learn, and develop problem-solving skills independently.
10. A screenshot displayed below shows the welcome greeting message, the user prompt, and a sample execution result of the program.



```
Welcome. Enter the no of transactions:
5
Enter the transaction amounts (space seperated):
100.50 50.60 80 90.75 110
Transaction List:[100.5, 50.6, 80.0, 90.75, 110.0]
Min Transaction:50.6
Max Transaction:110.0
Avg Transaction:86.37
```

Part 02 - Developing Software Analysis Report (10 points)

An important part of programming is to develop thinking skills. By now, we had implemented a few interesting ideas connected to object-oriented programming. Write a detailed report by analyzing the `Transaction.java` and `TransactionStub.java` program files that you had implemented from the previous part. Include a detailed summary of the analysis by incorporating the series of requirements outlined below.

1. **ADD** the following statement to all your code files including the file named `analysis-summary.pdf`.

This work is mine unless otherwise cited - Student Name

2. Identify the access modifiers connected to the members in the file named, `Transaction.java`. Describe in detail the general scope of these access modifiers. Access modifiers were discussed in detail in lesson 3.
3. Identify the object-oriented features connected to your completed version of the `Transaction.java` and `TransactionStub.java` program files and discuss how these features implemented in the program support the three software goals discussed in class.
4. Think about some new features that can be added to the completed code files from the earlier part to further enrich the object-oriented experience and to enhance the support of the software goals. Discuss your ideas in detail in the summary document.

Submission Details

For this assignment, please submit the following to your GitHub repository before the **deadline** by using the link shared to you by the Professor:

1. Commented source code from the “Patient” program.
2. Commented source code from the “PatientInfo” program.
3. Commented source code from the “Transaction” program.
4. Commented source code from the “TransactionStub” program.
5. A document containing the peer discussion points, named `pd-summary.pdf`.
6. A document containing the ideas to enrich the functionality, named `analysis-summary.pdf`.
7. It is highly important, for you to meet the honor code standards provided by the college. The honor code policy can be accessed through the course syllabus. Make sure to add the statement “This work is mine, unless otherwise cited.” in all your deliverables such as source code and PDF files. In your peer discussion summary file, please make sure to include your name and your partner’s name.

Grading Rubric

1. There will be full points awarded for the lab if all the requirements in the lab specification are correctly implemented. Partial credits will be awarded if deemed appropriate.
2. Failure to upload the lab assignment code to your git repo will lead you to receive no points given for the lab submission. In this case, there is no solid base to grade the work.
3. There will be no partial credit awarded if your code doesn’t compile correctly. It is highly recommended to validate if the correct version of the code is being submitted before the due date and make sure to follow the honor code policy described in the syllabus. If it is a late submission, then it is the student’s responsibility to let the professor know about it after the final submission in GitHub. In this way, an updated version of the student’s submission will be used for grading. If the student did not communicate about the late submission, then automatically, the most updated version before the submission deadline will be used for grading purposes. If the student had not submitted any code, then, in this case, there are no points awarded to the student.
4. If you need any clarification on your lab grade, talk to the Professor. The lab grade may be changed if deemed appropriate.

