A1 – 20% Due Aug 11th

**Object Oriented Programming**

The assignment can be done **individually** or in teams of **two**. Submit one assignment per team of two via Omnivox and NOT MIO. Assignments sent via MIO will be deducted marks. Assignments must be done alone or in groups and collaboration between individuals or groups is strictly forbidden. If you submit late, there is a 10% penalty for every day you submit late and you get 0 if it is submitted more than 5 days late.

**General Guidelines When Writing Your Programs:**

Include the following in comments at the top of your program:

// -----------------------------------------------------

// A1 Metro

// Written by: (include your name(s) and student ID(s))

// Short Description of your project/code and how you designed it.

// -----------------------------------------------------

Throughout your program, include comments in your program describing the main steps.

Write a program which controls a metro train and that keeps track of stations and who gets on and off. Design and implement a class called t**rain**, which has at least the following:

a. At least the following attributes (you can include others):

i. *metroID*

ii. *stationNum* which keeps track of the station that the train is at.

iii. *direction* which keeps track of the direction the train is travelling in.

iv. *passTotal* which keeps track of the number of passengers currently on the train.

b. *A default constructor* which sets the *metroID* to a random number between 1 and 1000, the station number to 0, the direction (int), and the *passTot* to 0 (train is empty).

c. *A constructor* with one parameter; the parameter is the metro id. The constructor assigns the passed integer to the *metroID* attribute; the rest of the attributes are set as described in the default contrustor.

d. Accessor methods for each attributes.

e. Mutator methods for each attributes.

f. *nextStation*(*int lastStation*) which determines the next station the train is to go to, and changes the direction of the train when necessary. This method should return a integer value which will specify the action that needs to be done: passengers leaving, passengers boarding or passengers getting off and boarding

g. *toString:* This method will return in a string the *metroID*, the current station number, the direction the train is travelling in as well as the number of passengers on the train.

Using all of these methods may not be necessary in your design but they must be present either way (can also be used to add something interesting).

Your program driver should:

- Display a welcome message

- Ask the user for the number of metro stations. There should be at least 3 stations. Make sure the user enters a number ≥ 3, if not keep asking until they enter at least 3. You may as well ask the user for input to name the stations.

- Create a Train object using the default constructor.

- As long as the user wants:

1. Advance the train to the next station. At each station, using a random number generate the number of people who will disembark the train (where applicable), and the number of people who have arrived wishing to embark on the train (where applicable). Rules:

- If the metro train is at the 1st station in any given direction, there will only be people waiting to get on as the train is empty.

- If the metro train is at the last station, all passengers must get off and no one can get on until the train turns around.

- Any other station, there are people waiting to get on (a max of 300), and some people will get off the train (can’t have more people getting off than there are people on the train).

- If the number of places available on the train, after people leave is less than the number of people waiting to get on, some people will be left waiting on the platform for the next train.

2. Keep track of the number of people who did not get on the train, as they will try and embark on the next train. (Recommend having a 2 dimensional array with size 2 by the number of stations; each row represents a direction).

3. Display the Train ID, the station that the train is leaving and the number of passengers. As well:

-indicate the number of people who got off the train

-the number of people who were left behind at this station the last time and are hoping to embark the train this time

-the new people who arrived on the platform

-the number of people who actually got on the train

-the number who are left behind on the platform because the train was full this time.

Include the train id, the station number and the direction the train is travelling in (info that is displayed when output a train object).

4. Ask the user if they would like to continue following the metro

- If yes, continue to next station and repeat the process

- If no, quit and end the program with a closing message.

The metro train travels from one end of the line to the other. When it reaches the end in either direction, all passengers must get off. The train turns around and starts again with an empty train from the same station heading in the opposite direction.

Additional features: Part of your grade will be to implement some extra feature of your own choosing, some potential ideas to use include: multiple trains, different directions (north, south), compare trains, gui, additional lines (blue, green, yellow) etc… Two points for two different ideas implemented that notably modifies your application.

Grading Schema:

|  |  |  |
| --- | --- | --- |
| **Common Criteria for Assignment – (2 pts.)** | | |
| Comments & Programming Style which includes:  - Description of the program (authors, date, purpose)  - Significant names for variables  - Description of variables  - Description of the algorithm | 2 | pts. |
| **Question 1 – (18 pts).** | | |
| No bugs occur | 5 | pts. |
| Meets the above requirements/description | 7 | pts. |
| Clarity of output/correctness | 3 | pts. |
| Code formatted correctly | 1 | pts. |
| Extra Features | 2 | pts. |
| TOTAL | 20 | pts |