COSI 12B Summer 2020 – Advanced Programming Techniques

Programming Assignment 4

Objective

- Practice developing a well encapsulated object-oriented program
- Practice using Eclipse Debugger Tool

Overview

In this assignment, you will be simulating a building that contains a single elevator. However, this elevator has a special characteristic, it can only be used once by a person as they enter the building. Once someone has used the elevator they will remain at their floor forever. You can think of this elevator as a one-way elevator.

Person instances will request to enter the building and go to a specific floor. Two things need to happen. As a person arrives at the building, the elevator may or may not be on the first floor, ready to take new passengers. To address this problem, we will put a "job" on the elevator to go to floor number 1, and immediately after put a second "job" for the person that has entered the building to go the their desired floor. More is said about this further on.

This elevator will not be very efficient, since it will process "jobs" in order (first-come-first- serve). So, if two people come to the building, the elevator will come to the first floor, take the first person and then take them to their desired floor, and then go back to the first floor for the second person that entered the building. The elevator is given a batch of "jobs" and then it should process the entire batch as fast as possible.

You will be coding an Object-Oriented solution, where you will need to implement the following classes:

Building: It should hold an array of floors and an instance of an elevator.

Floor: Every floor must contain an array of people (that are currently on a given floor).

Elevator: An elevator will contain a list of Jobs (as an array), where each Job

represents a person and the floor to which they desire to go. The elevator should also contain a reference to the building to which it belongs.

Person: Each person has a first name and a last name.

Job: This class is used to represent an elevator trip. This class holds a reference to a Person and a floor number. You can use a null object for the first parameter to represent an empty trip (calling the elevator to the first floor).

Simulation: This class should contain a main method, where you simulate the building. You should instantiate a building, and several person instances. Then you should simulate two different groups of people, arriving at different times, and requesting to use the elevator. Print the state of the building/elevator/people's locations in order to show what is going on in the building as the elevator runs. This is so the user can tell what is going on.

Implementation Details

The idea is that a client program should be able to create person instances and a building instance. Then, each person could call a method to enter the building, where the building instance is passed as a parameter. Then a method would be called from the building instance on its elevator, creating a "job" or trip to be executed. The details of each class and the necessary methods are below. Note that you may use additional methods in your classes in order to have an organized and modular solution. Analyze the code and decide how to implement your solution. We are not looking for a uniquely right solution, so use your own judgement as a programmer and describe your logic in a README.txt file.

Note: Feel free to pass as many parameters in each constructor method as you see fit. Comment your code very well, as we will be evaluating your logic and your understanding of object- oriented practices.

Building Class

- `public boolean enterElevator(Person person, int floor)` — This method will process the request made by a person to enter the building. Then, it should pass on the request to an elevator instance. Make sure that the elevator visits the first floor and then the floor requested by the person.

- `person`: the person that has requested access to the building
- `floor`: the number of the desired floor

- `public void startElevator()` – This will call a method in the elevator instance that should process all current jobs.

- `public boolean enterFloor(Person person, int floor)` – This method should simply access the given floor object and call a method to save a reference to the person in the given floor.
  - `person:` the person to access the floor
  - `floor:` the floor number to be entered

Person Class

- `public boolean enterBuilding(Building building, int floor)` – This method will simply call the appropriate method from the building instance to enter the elevator and request a job.
  - `building:` the building to be entered
  - `floor:` the floor requested
- `public String getLocation()` – You should hold a variable (that you will need to update at certain moments) that will say the location of a person. You can choose how to implement this, but you should return strings like "In lobby", "In Elevator", or "In floor 4", etc...

Floor Class

- `public boolean enterFloor(Person person)` – This method should save a reference to the person within the floor object
  - `person:` the person to enter the floor

Job Class

- `public Job(Person person, int floor)` – This class is used mostly to hold information.
  - `person:` the person that requested a job (or null if the elevator was requested to go to the lobby to pick up a person)
  - `floor:` the floor number that has been requested

Elevator Class

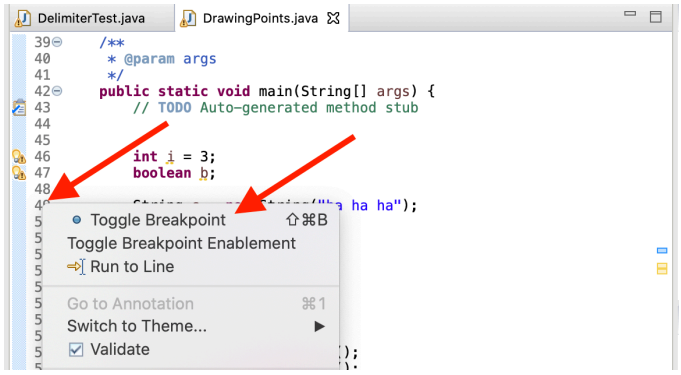
- `public public boolean createJob(Person person, int floor)` – This method should simply add, in the right place, a new job to be completed by the elevator.
  - `person:` the person that requested the elevator
  - `floor:` the desired floor number
- `public void cleanUpJobs()` – This method will be used after completing (and removing) a job. The use of this method should guarantee that the jobs array is in a valid state.
- `public void processAllJobs()` – This method should remove jobs one by one (in the right order) and process them individually.
- `public boolean processJob(Job job)` – This method should process a job, and move the elevator floor by floor (while printing updates) in order to reach the desired floor. Then, the exit method should be called, simulating the exit of a person (if necessary).
  - `job:` the job to be processed
- `public boolean exit(Person person, int floor)` – This method should call a method on the building for a person to enter a given floor (hold a reference to the person in the given floor).
  - `person:` the person exiting at a given floor
  - `floor:` the floor at which the person is exiting

Please make sure to read and understand the code provided before attempting a solution. There are additional comments that are very important to your task. If you draw a simple diagram of how the classes interact and what the flow of the program will be before you start coding, it will be much, much easier to complete this assignment.

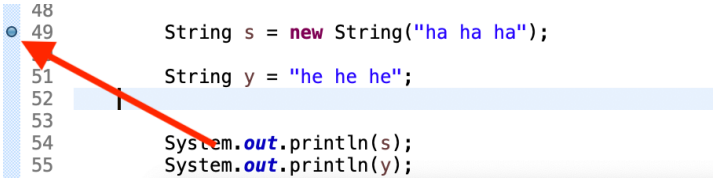
Debugging

Use the debugger included with Eclipse to debug your program step by step:

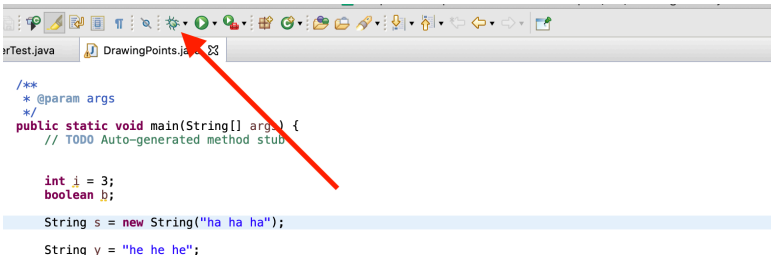
1. Set a breakpoint in the code just before where you are experiencing a problem. If you set the breakpoint on the 1st statement of your main method, you will be able to step through every line of code in your program.
    a. Right click on the number of the line of code where you would like to set the breakpoint
    b. Select Toggle Breakpoint to set or remove



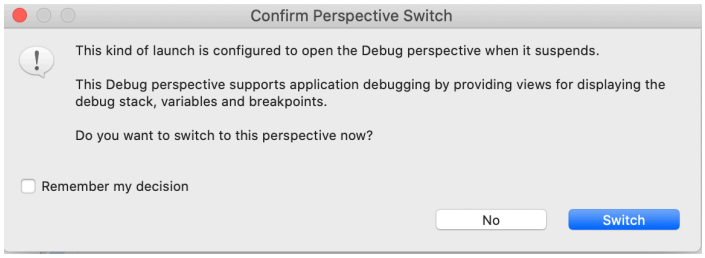
    c. You will see a little blue dot appear to the left of the line number when the breakpoint is set.



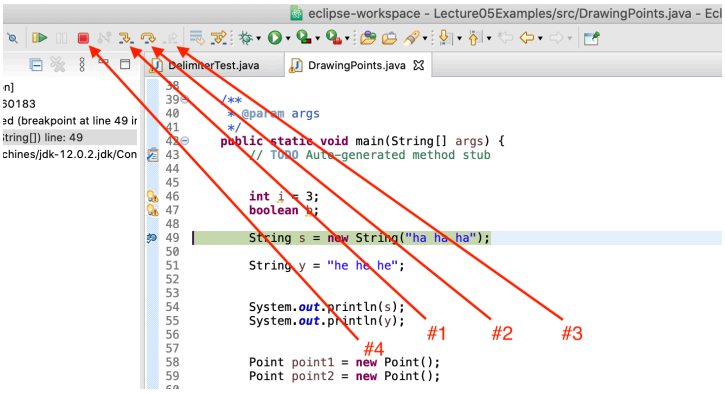
2. Run the program in Debug Mode
    a. Click on the "Debugger" button



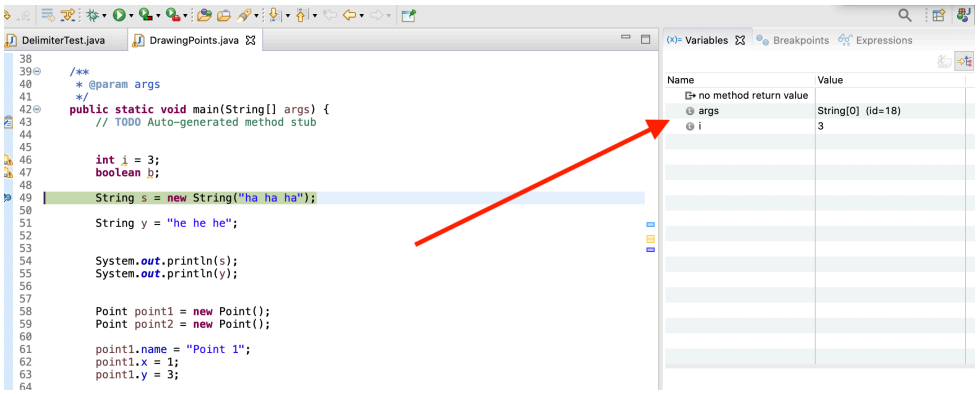
b. First time you run this, you will be prompted to switch perspectives. Agree to that. Eclipse will open additional windows useful for the debugging process



3. Set through your code to watch where each step takes. (1) You can jump into the functions (all the way to the Java source code). (2) You can jump over statements to execute them without going into them. (3) You jump out of functions to get back to the calling function. (4) You can stop execution to start again.



4. As you step through your code, you can watch how each step takes you and how it affects you variables



Alternatively, you can use the `toString` methods and `print` statements to track down where things are going wrong however, this is a less efficient method so I recommend you get comfortable with the debugger.

Regardless if you use the debugger or the print statements, in order to debug effectively, you should know EXACTLY what to expect from the code that you write in your main method. This way, you can compare what is actually happening to the expected results. Then, you can isolate the problem and know where in your code you need to revise the logic.

Important

You should NOT import any libraries in your code. The only data structure that you will be using is arrays, remember that you can use arrays of any type of object. The objects in the array can be instances of the specified class or null. Make sure you handle this correctly and be careful with potential `NullPointerExceptions`. You should NOT use any java collections.

If you think that you need extra classes, feel free to implement them. However, make sure to explain why did you need them and what advantage they you're your program.

Grading

We will be mostly looking at the way that you have organized your code and see if you understand how to code an object-oriented solution and how to practice encapsulation. We will be testing your ability to write concise classes, but also your ability to successfully relate the classes to one another.

You will be graded on:

- **External Correctness:** The output of your program should match exactly what is expected. Programs that do not compile will not receive points for external correctness.
- **Internal Correctness:** Your source code should follow the stylistic guidelines shown in class. Also, remember to include the comment header at the beginning of your program.

Submission

Make sure your project in Eclipse is called yourfirstname_yourlastnamePA4 before you export the final version for submission. Submit via Latte a **zip** file named yourfirstname_yourlastnamePA4.zip which contains all your .java files and javadocs. Please make sure to use **exactly** this file name, including identical capitalization.