Homework #3: Active Passengers, Collections

Due: Monday, October 21st 11:59 pm Eastern time

Grade: 15%

Last Modified: Wednesday, 25-Sept-2019 15:03:34 EDT

Note: For this assignment you must to generate and include JavaDoc.

Specification of Requirements

Overview

This homework extends Homework 2 in the following ways:

Instead of tracking *counts* of passengers, individual objects that implement the Passenger interface are tracked. Elevator and Floor will track Passengers in collections, moving Passenger objects from one collection to another as part of the simulation.

This more detailed simulation correctly handles the direction in which a Passenger wants to travel. E.g., if the Elevator is going down and arrives on a Floor, a Passenger who is waiting to go up will not board.

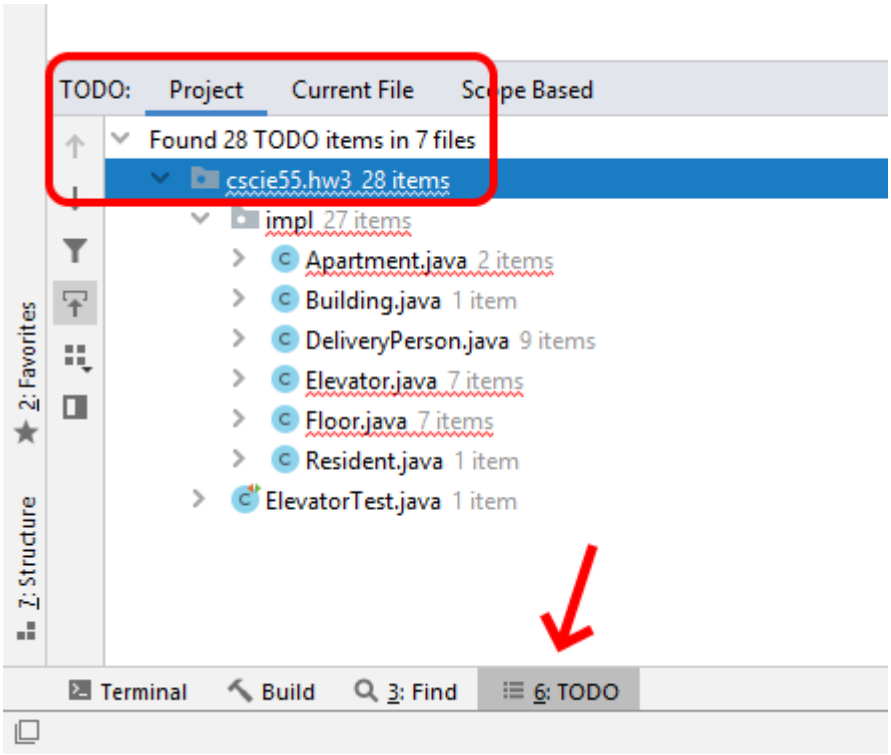
Passenger [the interface] is implemented by the Resident class. The Elevator manages its cargo as 'Passenger', but the concrete class is Resident. A Resident can do anything a Passenger can do.

Another class, DeliveryPerson, also extends Person and implements Passenger. But this class lacks a feature that allows the Resident to enter an Apartment. Instead, instances of this class may only ring an Apartment's doorbell.

As in Homework 2, your code will be tested by JUnit tests. While much of HW2 code may be reused, do not assume that everything is exactly the same. Test Everything!

The use of logging has been [gently] incorporated into this assignment. See the ElevatorTest class for usage. You will find dependencies defined in the pom.xml file for the project, and a configuration file in resources directories.

Throughout the classes provided you will see many methods with comments '***//TODO:***' Each of these must be completed. Hint: Use your IDE to quickly identify these:



Remember that the package names have changed!

Tasks

Code organization

The Elevator, Floor, Building and Resident , DeliveryPerson classes all go into the package ***cscie55.hw3.impl.*** If you define any additional classes, they should also go into this package (and should not be declared public). If you copy-paste HW2 classes, remember to refactor the package names.

The Passenger class is in the ***cscie55.hw3.api*** package.

Assuming you are happy with your Homework 2 code*, the best way to proceed is probably to revisit the Elevator and Floor classes, and modify them to track Passenger objects in collections as detailed below.

Building

In homework 2, the Building class contained an array of type Floor with a length of 7. Since our building has not grown higher, that will work for homework 3 just fine.

- Add a new ***public static int*** member named **UNDEFINED_FLOOR** and set its value to ***-1***.
- Remove the 'enter()' method from HW 2 if you are using HW 2 code, we won't use it.
- Be sure the Building constructor has method createFloors() that creates the 7 Floors in the building as in homework 2.
- Add the floors to the Elevator, as we did in homework 2.

Floor

Floor needs ***three*** collections of Passengers:

1. Passengers who are **residents** on the Floor, not waiting for the elevator;
2. Passengers who are **upwardBound**: waiting for an Elevator **going up**;
3. Passengers who are **downwardBound**: waiting for an Elevator **going down**.

Add a method named callElevator(Passenger passenger), that will place a passenger into the correct one of these collections. (we assume that a passenger does **not** call the Elevator from any floor to travel to that same floor.)

A Passenger on a Floor will be in exactly one of these collections. This means you must put Passengers into one of these collections as needed. When Floor.callElevator() is invoked, examine the Passenger's destination property to determine if that Passenger is going up or down, and add that passenger to the appropriate collection.

Design consideration: When an Elevator arrives at a Floor it should unload passengers destined for that floor, then ask the Floor for a collection of Passengers waiting for service: upwardBound when the Elevator is going up, downwardBound when the Elevator is going down. The Elevator will board the Passengers, one by one, subject to the condition that it must not exceed capacity.

Question: which Passengers board first? To ensure that Passengers on the wait-queues should get service in the order in which they joined the queue, the queues should be represented by an ordered collection. That is: First In, First Out [FIFO]

Passenger

This is an ***interface*** which defines several abstract methods, get/set a Passenger's currentFloor and destinationFloor. A Passenger who is resident on a Floor has a current floor number and an UNDEFINED_FLOOR destinationFloor. A Passenger who is on an Elevator has a destination number but an undefined current floor number. A Passenger who is waiting for an Elevator has a current floor number (the Floor on which the Passenger is waiting), and a destination floor number.

Passenger defines the following public methods:

- int getDestination(): returns the Passenger's destinationFloor.
- void setDestination (int newDestinationFloor): Sets the Passenger's destinationFloor.
- void boardElevator(): Sets the Passenger's currentFloor number to be UNDEFINED_FLOOR.
- void arriveOnFloor(): The Passenger is on an Elevator and arrives at the destination.
    - In the implementation of this method (i.e., ***in the Resident class***), you will do 2 things:
        1. set the value of the destinationFloor to the ***Passenger's currentFloor number***
        2. set the Passenger's destination floor number to be UNDEFINED_FLOOR.

- A class implementing Passenger starts out with current floor number = 1 (the ground floor) and the destination floor number UNDEFINED_FLOOR.

A word of advice: Be precise about undefined values. Use the **static final int UNDEFINED_FLOOR in your Building class** to be -1 and assign this as the value of the current or destination floor numbers as appropriate. If you do this correctly, (and print both current and destination floor numbers in the toString method), it will simplify debugging. E.g., if you see a Passenger in an Elevator collection, and the destination floor is UNDEFINED, then something has gone wrong.

Resident

As in homework 2, Resident implements Passenger. Add the following to the Resident class constructor (and be sure there are fields to support them):

```
currentFloor = 1;
destinationFloor = Building.UNDEFINED_FLOOR;
doorKey = address.hashCode();
```

Be sure there are public methods to retrieve these values. Examine the tests to see how these are tested and how they are to be used.

DeliveryPerson

The DeliveryPerson is a new class. It extends Person and implements Passenger, just like Resident. Populate the fields and constructor of DeliveryPerson just as you find in Resident

However, the DeliveryPerson's properties will differ in certain important respects. It does **not** have the enterApartment() method.

Implement the fields and methods in DeliveryPerson marked '//TODO:'

A DeliveryPerson can board the Elevator and ride up to a floor just like Resident. But s/he cannot enterApartment. Instead, add a method that takes an Apartment as a parameter, and returns a string. See ElevatorTest@DeliveryPersonTest for the format.

Elevator

Elevator needs a collection of Passengers who intend to disembark on each Floor. Note: this entails a collection that can resolve the destination [i.e., a Floor] and accounts for the possibility that multiple Passengers may be destined for each floor. A list of lists (List<List<Passenger>>) should do the trick.

Add the field: private List<List<Passenger>> passengers = new ArrayList<>();
Starting with Elevator from Homework 2, add the following public methods:

- Replace the homework 2 int[] passengers with List<List<Passenger>> passengers. Code for this is provided.
- Add an enum named 'direction' with two possible values: UP and DOWN.

- Modify the move() method to use the direction enum to handle the state of the Elevator's travel direction.
- Remove the boardPassenger(**int destination**) method and replace with boardPassenger(**Passenger passenger**). Get the destination floor by obtaining the Passenger's destinationFloor field value.
- Modify boardPassenger() method so each passenger is added to the List<List<Passenger>> passengers destined for the appropriate floor.
- Change the getPassengers() method to return the number of passengers in the new List<List<Passenger>> passengers
- Note the use of the ***LOGGER***. See the ElevatorTest class for its usage. When throwing Exceptions in try-catch blocks, (calling boardPassenger() in your move() method, for example), log messages rather than use System.out.println().

Floor

Floor should be modified as follows:

The int passengersWaiting() method is no longer needed and should be removed (or made non-public, if you need it for your implementation. Your choice.).

The ***callElevator()*** method needs to be modified. Change it to ***void callElevator(Passenger passenger).***

Add three new fields to each Floor to store Passengers in a polite line so the first one in line is the first one into the Elevator:

- **private Collection<Passenger> residents**;
- **private Collection<Passenger> upwardBound**;
- **private Collection<Passenger> downwardBound**;

Notice that classes implementing the Passenger interface (i.e., Resident) requires a property, int destinationFloor, that returns an int value when required methods int *getDestination()* and *setDestination(int destinationFloor)* are called.

When ***callElevator(Passenger passenger)*** is called, use the *getDestinationFloor* method of the Passenger to determine if this Passenger is waiting for the Elevator to go up or down. Add the Passenger to one of the new collection fields described above as appropriate.

When the passenger is unloaded on their destination floor, they must be added to the residents collection.

You will need to decide what fields the Floor class should have.

Apartment

In addition to the properties delivered with the Apartment class with homework 2, add the following methods:

@Override

public int hashCode()

public int getkey() -when getKey() is called, return the value of hashCode().

Exception classes

There are 4 exception classes included in the exception package. Examine the tests to see how they are used.

Testing

The test code as before is under the src/test branch. The test file is named ElevatorTest.

The unit tests in Homework 3 are adapted from the Homework 2 test. They have been modified to test the more detailed simulation you are implementing for Homework 3. Although they are similar in many cases, do not assume that they are identical to the tests in homework 3.

I recommend you begin by migrating the code from Homework 2 into homework 3 and studying the test methods starting with the simplest methods and progressing to the more complex. In combination with examining the TODO's, you should be able to get this working. My estimation of the order of difficulty, from easier to more difficult, is as follows:

- testElevatorCreation()
- testFloorCreation()
- testApartmentCreation()
- testDeliveryPerson()
- testElevatorMotion()
- testElevatorBoardPassenger()
- testArriveOnFloor()
- testEnterApartment()
- testLoadUnload()
- elevatorFull()