

# COMP1406 - Assignment #9

(Due: Monday, March 27th @ 10:00 am)



In this assignment we will simulate an **Taxi Service Dispatching System** that manages a fleet of taxis within a city. The system will keep track of roughly which taxis are in certain pre-defined areas of the city and it will have the ability to send taxis to certain areas of the city. Since some cities have many taxis (such as New York which has over 12,000 yellow cabs), finding an available taxi may be a time-consuming task. We will use two **HashMaps** in our system that will allow us to quickly find taxi information based on the taxi plate number and also to quickly find an available taxi to send out for pickup. The main purpose of the assignment is to get familiar with **Hashmaps** and **ArrayLists**, although we will also use 2D arrays and you will get some more practice towards improving your "problem-solving" skills. The GUI parts of the assignment have already been completed for you.

## (1) The Taxi Class

Download the **Taxi** class that represents a taxi in the system. It looks as follows:

```
public class Taxi {
    private int         plateNumber;
    private boolean     available;
    private String      destination;
    private int         estimatedTimeToDest;

    public int getPlateNumber() { return plateNumber; }
    public boolean getAvailable() { return available; }
    public String getDestination() { return destination; }
    public int getEstimatedTimeToDest() { return estimatedTimeToDest; }

    public void setAvailable(boolean avail) { available = avail; }
    public void setDestination(String d) { destination = d; }
    public void setEstimatedTimeToDest(int t) { estimatedTimeToDest = t; }

    public void decreaseEstimatedTimeToDest() {
        estimatedTimeToDest--;
    }

    public Taxi (int plate) {
        plateNumber = plate;
        available = true;
        destination = "";
        estimatedTimeToDest = 0;
    }

    public String toString() {
        if (available)
            return plateNumber + " (available)";
        return plateNumber + "(" + estimatedTimeToDest + ")";
    }
}
```

Note that taxis are considered *unavailable* if they are (1) on their way to pick up a passenger (i.e., *en-route*) or (2) if they currently have a passenger.

Create an **equals()** method for Taxis (making sure that the parameter is of type **Object**, not of type **Taxi!!!**). Two taxis should be *equal* if and only if their plate numbers are equal.

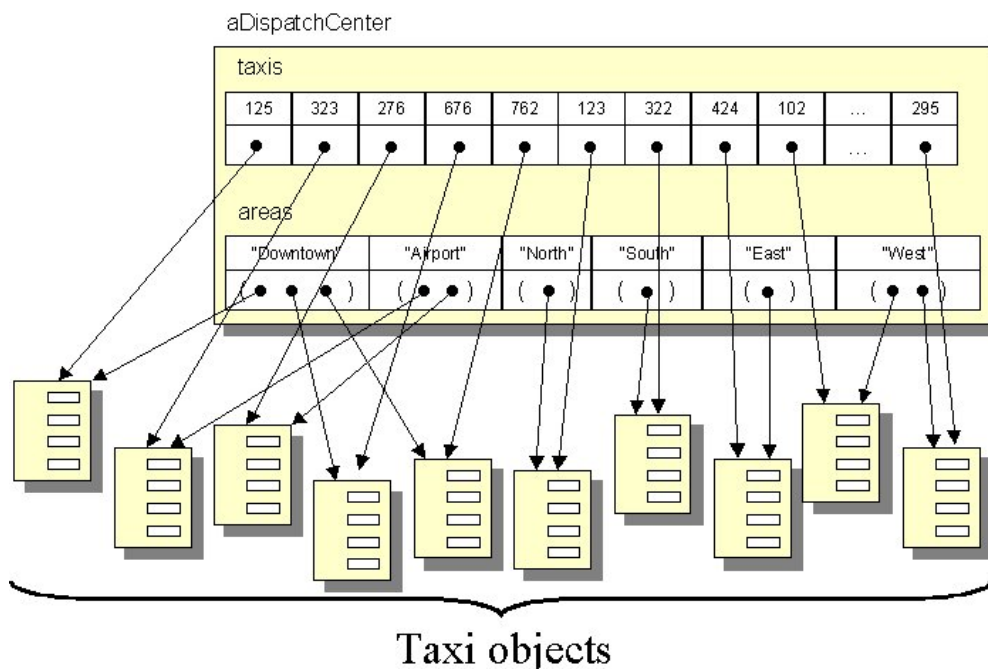
## (2) The DispatchCenter Class

Download the class called **DispatchCenter** that represents the model for the Taxi Service Dispatching System. You must add attributes to this code and complete the various methods as described below. The **DispatchCenter** should maintain a HashMap called **taxis** that keeps taxi plate numbers (i.e., Integers) as *keys* and the *values* being the Taxi object that has that plate number. This allows us to quickly find a taxi when given its plate number only.

The **DispatchCenter** should also maintain a HashMap called **areas** that keeps a city's "area name" as the *keys* and the *values* being ArrayLists of all taxis that are currently in that area. The areas are places in the city that can be serviced by the taxi cabs. Note that there are exactly 6 areas that we will specify in this assignment. They are: "North", "South", "East", "West", "Downtown" and "Airport" (conveniently defined as AREA\_NAMES in the code you were given). It will be assumed that taxis travel between these areas when making service calls (although a call may allow a taxi to stay within the same area). Note that when a taxi drives to a new area, its location will change within this HashMap. Also note that a taxi may be in only one of these ArrayLists at a time.

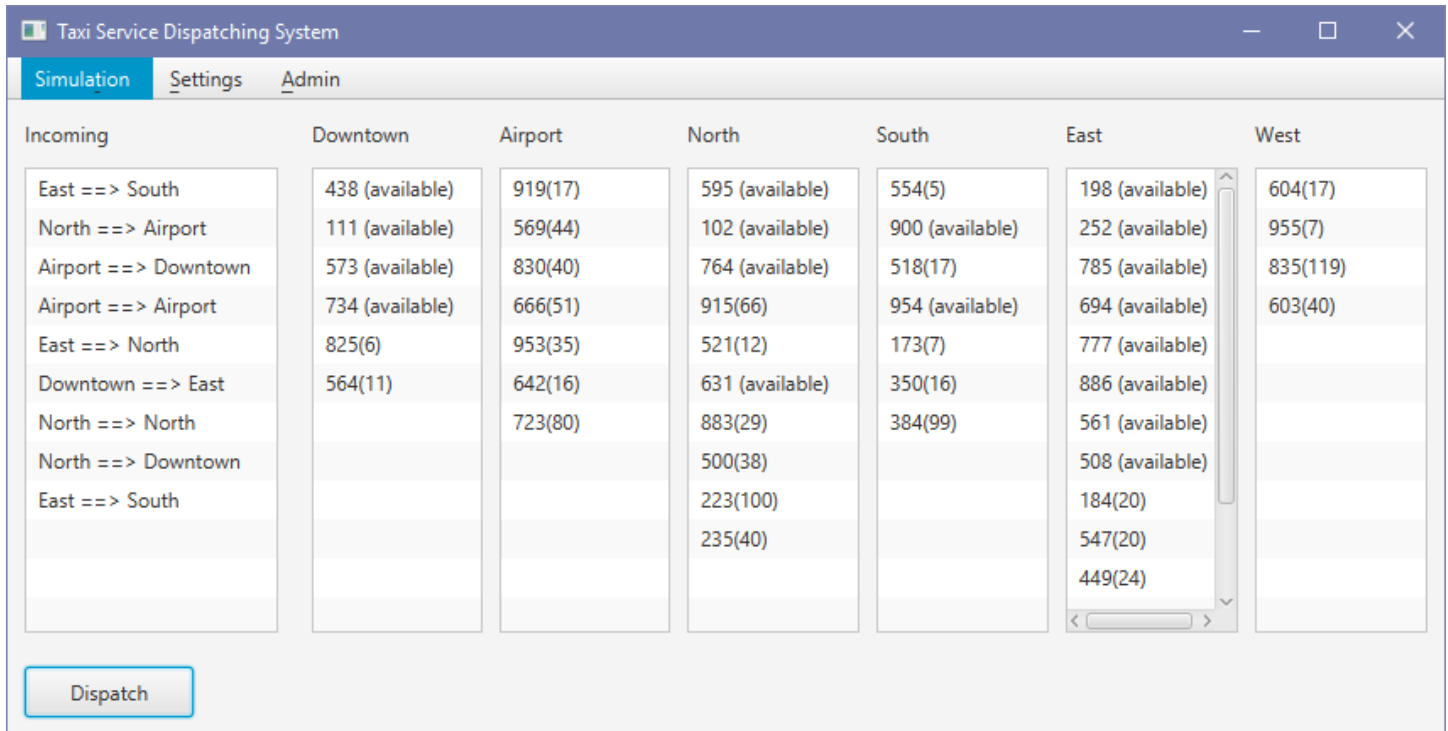
Complete the **addTaxi()** method that adds a taxi to the dispatch center. It should take a **Taxi** object as a parameter as well as the initial starting area for the taxi (e.g., "North"). Your method should add the taxi to both HashMaps in the appropriate manner.

Complete the zero parameter **constructor** so that it initializes the **taxis** HashMap with 50 taxis that have random plate numbers ranging from 100 to 999. Also, the **areas** HashMap should be initialized with the 6 fixed areas mentioned earlier. Each taxi should be randomly assigned to one of the 6 areas upon initialization. To assign the taxis to the HashMaps, you **MUST** make use of the **addTaxi()** method written above. See the picture below as an example of how things should be organized.



### (3) The **TaxiServiceDispatchSystem** Class

Download the **TaxiServiceDispatchSystem** code. It is an application that , when run, should display the information from the dispatch center. This window has been constructed for you already, using your **DispatchCenter** as its model:



The application displays the incoming client requests on the left. This represents people who are waiting for a taxi ... it indicates where they are waiting to be picked up and where they want to be dropped off. Each request is represented by the following **ClientRequest** class, which you should download as well:

```
public class ClientRequest {
    private String pickupLocation;
    private String dropOffLocation;

    public String getPickupLocation() { return pickupLocation; }
    public String getDropoffLocation() { return dropOffLocation; }

    public ClientRequest (String p, String d) {
        pickupLocation = p;
        dropOffLocation = d;
    }

    public String toString() {
        return pickupLocation + " ==> " + dropOffLocation;
    }
}
```

In the remaining 6 lists, it displays the Taxis that are currently in that area, or who have just left that area with a client. The Taxis are displayed showing their plate number and beside it in brackets it either shows that the Taxi is available to take on a new client, or it shows an estimate of the remaining number of minutes that that Taxi will be unavailable for as it is in the midst of bringing its client to their destination.

When pressed, the **Dispatch** button should dispatch a Taxi to handle the first (i.e., top) request on the incoming client requests list. The system should dispatch the first available Taxi that is within the clients pickup location, if there is one available. However, the dispatching code does not work yet.

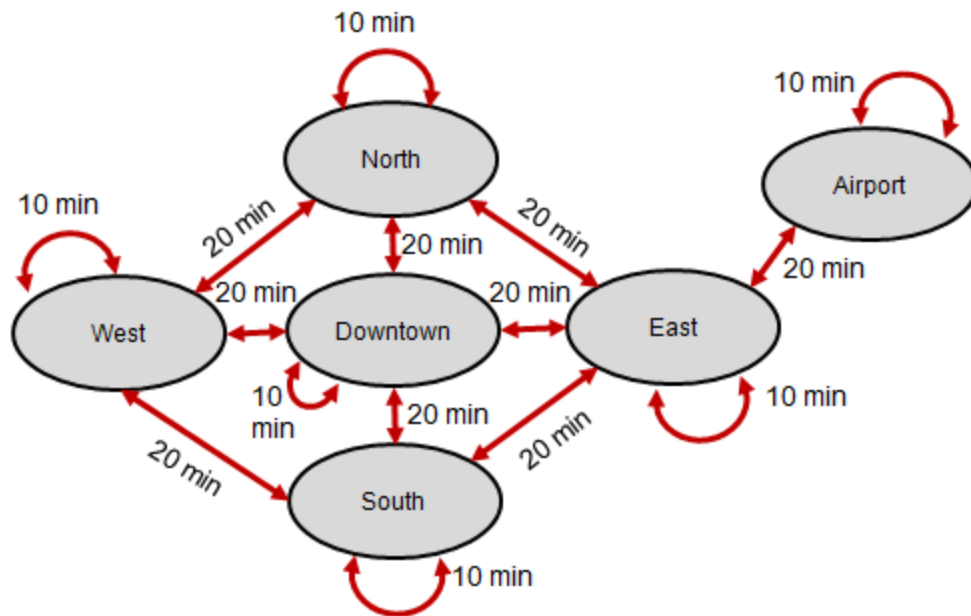
In the **Simulation** menu, there are two options. The **Start Clients Coming** option causes the system to start a timer that begins inserting random automatic client requests to the incoming client request list. The **Stop Clients From Coming** option stops the timer, putting a halt on incoming clients. The **Settings** menu contains a Simulation Rate... option that allows the user to set the rate at which the clients come into the system. You can use this to produce lots of clients quickly, or to go slowly to watch how the system behaves. Finally, in the **Admin** menu, there is a **Statistics** option that will bring up a Dialog showing various statistics regarding the pickups and dropoffs over time.

For this part of the assignment. Just run the code and make sure that the window appears with the 50 random requests as you created in part (2) of the assignment.

## (4) Dispatching

We now need to make everything work properly. In the **DispatchCenter** class:

1. Complete the **availableTaxisInArea(String s)** method that takes an area as a parameter and returns an **ArrayList** of all **Taxi** objects in that area that are **available**.
2. Complete the **getBusyTaxis()** method that returns a single **ArrayList** of all **Taxi** objects (in ALL areas combined) that are **unavailable**.
3. Complete the **sendTaxiForRequest(ClientRequest c)** method that sends a Taxi to pick up the client and returns that Taxi by following the rules below. Note that this method will be called from the **TaxiServiceDispatchingSystem** GUI when pressing the **Dispatch** button.
  1. if there are available taxis in the requested pickup area, the taxi that arrived first is chosen for the pickup (i.e., first come first served).
  2. if there are no available taxis in the pickup area, then the system must find a taxi in any one of the other areas and send/return that one.
  3. if there are no available taxis anywhere, the request for pickup is denied....no taxi is returned (i.e., **null**).
  4. when a taxi is sent, it should be removed from its current location in the **areas** HashMap and be appended to the bottom of the pile in the **areas** HashMap corresponding to the client's requested drop off location. The taxi should become **unavailable**. The **ClientRequest** should also be removed from the **incoming** list.
  5. the **estimatedTimeToDest** for the taxi should immediately be set to a value that corresponds to the following diagram:



For example to go from West to West takes 10 minutes, West to Downtown takes 20 minutes, West to East takes 40 minutes and West to the Airport takes 60 minutes. So, if a taxi is in the West and needs to go to another area, the time to travel is indicated in the picture above. However, if a taxi is needed from West to the Airport, for example, but the only taxi available is currently in the East, then the estimated time to the destination is the **combined time** for the taxi to go from the East to the West to pick up the client, then from West to Airport to drop him/her off. This is a total of 100 minutes. To accomplish this, it is best to complete the static method called **computeTravelTimeFrom(String pickup, String dropOff)**. In that method, it is wise to make a 2D array of travel times where the columns represent the pickup area and the rows represent the dropoff area.

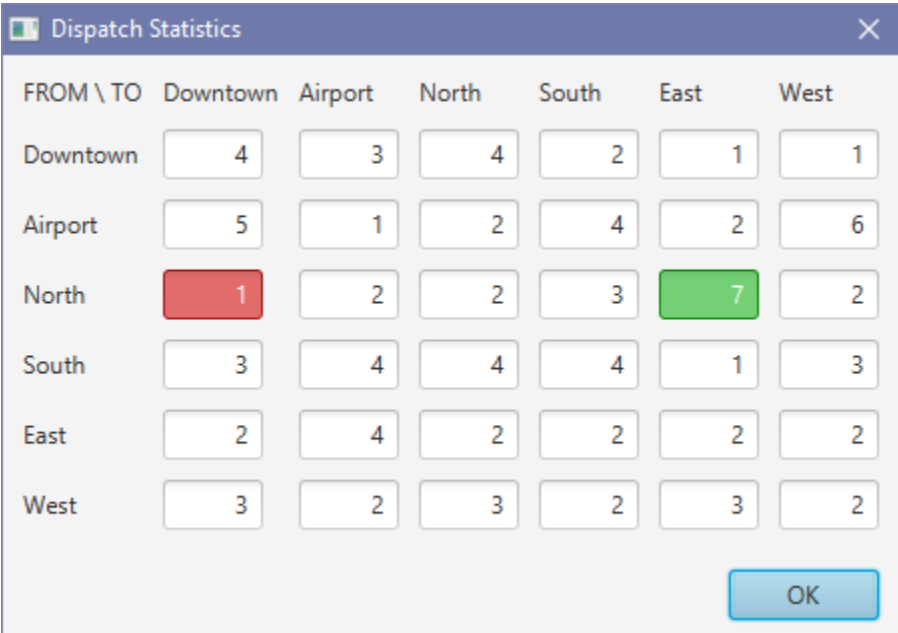
Test your code to make sure that the Dispatch button properly dispatches taxis and that the travel times are computed properly and that the taxis eventually return to *available* status after the delivery time has elapsed.

## (5) Statistics

Download the **DispatchStatsDialog** class, if you have not done so already. When the user selects **Statistics** from the **Admin** menu, this Dialog box will appear showing the statistics for each area pair combination as a grid of non-editable TextFields.

The **DispatchCenter** has a 2D **stats** array that keeps statistics on each taxi dispatch for all successful **ClientRequests**. That is, it keeps track of the number of pickup/dropoff combinations over time. A temporary 6x6 2D array has been defined already which is used by the dialog box in the GUI.

Complete the method called **updateStats(String pickup, String dropOff)** which updates this 2D array of stats each time a dropoff is made. This method should be called by the **sendTaxiForRequest()** method when a taxi is sent off. The code will automatically attempt to show the most busy route in green and the least busy in red.



The screenshot shows a dialog box titled "Dispatch Statistics" with a close button (X) in the top right corner. It contains a 6x6 grid of statistics for different area pairs. The rows and columns are labeled: Downtown, Airport, North, South, East, and West. The values in the grid are as follows:

FROM \ TO	Downtown	Airport	North	South	East	West
Downtown	4	3	4	2	1	1
Airport	5	1	2	4	2	6
North	1	2	2	3	7	2
South	3	4	4	4	1	3
East	2	4	2	2	2	2
West	3	2	3	2	3	2

An "OK" button is located at the bottom right of the dialog box. The cell for North to East (7) is highlighted in green, and the cell for North to Downtown (1) is highlighted in red.

Test your code to ensure that it works.

---

### IMPORTANT SUBMISSION INSTRUCTIONS:

Submit your **ZIPPED IntelliJ project file** as you did during the first tutorial for assignment 0.

- **YOU WILL LOSE MARKS IF YOU ATTEMPT TO USE ANY OTHER COMPRESSION FORMATS SUCH AS .RAR, .ARC, .TGZ, .JAR, .PKG, .PZIP.**
  - If your internet connection at home is down or does not work, we will not accept this as a reason for handing in an assignment late ... so make sure to submit the assignment **WELL BEFORE** it is due !
  - You **WILL** lose marks on this assignment if any of your files are missing. So, make sure that you hand in the correct files and version of your assignment. You will also lose marks if your code is not written neatly with proper indentation. See examples in the notes for proper style.
-