CSCI 1933 Lab 10

Discrete Event Simulation

Rules

You may work individually or with *one* other partner on this lab. All steps should be completed primarily in the lab time; however, we require that at least the starred steps (★) to be completed and checked off by a TA during lab time. Starred steps cannot be checked off during office hours. You have **until the last office hours on Monday** to have non-starred steps checked by ANY TA, but we suggest you have them checked before then to avoid crowded office hours on Monday. There is nothing to submit via Moodle for this lab. This policy applies to all labs.

Lab Overview

In this week's lab you will be creating a ferry simulator, which will hopefully give you an idea of how to complete Project 4. The ferry you will be simulating is similar to a traditional ferry going between three islands. Here are some things to know before you get started:

- The ferry has three stops at three islands goes in a circle (ex. 0,1,2,0 and so on). Islands will be represented as `int`s in 0,1,2.
- The ferry will stop at each island even if no one is at the island waiting for it.
- When a person gets on the ferry, they know what island they want to get off at. Each island is equally likely to be chosen, except they will never get off at the island they got on from.
- The ferry stops for one second for each person that gets on or off at that island.
- The ferry can hold 30 people at a time. People get off the ferry before people get on, but if the ferry is full they can't get on and must wait for it to come around again.
- The time between people arriving at a given island to start waiting for the ferry is n seconds, where n is a random number on the interval $[5, 10]$.

Hint: You might want to consider importing `Random` and `ArrayList`.

The rest of this page is intentionally left blank

1 Starting the simulator★

In this section we'll make the driver class for this simulation. Download the supporting files: *Event.java, PQ.java, PQInterface.java, Segment.java, Q2.java, Q.java, N.java.* These can be found on moodle in *Week 9 - Priority Queues, Discrete Event Simulation.*

Next, make another class called `FerrySim` with the following methods and fields:

- `public static PQ agenda` – This is the agenda that all of your `Events` will be run through. We make it `static` so that it can be accessed anywhere, without explicitly needing to pass around the `FerrySim` or `PQ` objects.
- `public static void main(String[] args)` – We'll leave this blank for now and get to it after we've written some other classes. It'll be similar to CarSim's main, however.

The rest of this page is intentionally left blank

2 Who wants to ride the ferry?★

In this section you will be making `Passengers` to ride on the ferry. You'll be using PQ and implementing Event.

Events are actions that will run at some point in the future, relative to the current time of the agenda. Notice that `PQ` does not operate on `FerryEvent` objects, but on `Event` objects. `Event` is simply an interface that provides a `run()` method this means we can put multiple types of events in the agenda, that have different run methods! This is going to be necessary for both this lab and Project 4.

Make a `Passenger` class with the following methods:

- `public int getPickupIsland()` – This is the island that the person got on the ferry at.
- `public int getDropoffIsland()` – This is the island that the person will get off the ferry at.
- `public Passenger(int pickupIsland)` – A constructor that takes in the island the passenger is starting at. This should save pickupIsland, and select a random drop off island that isn't pickupIsland.
- Passengers know what island they came from and where they are going.

Now that we have our Passenger objects made, we need a place to put them while they wait in line. Use a queue to store Passengers. You decide where the queue should be. Note that each island needs their own queue.

> **Hint:** Is it necessary to make a class to store these queues? How are we going to access these queues outside of the class that they're in?

Now lets make an event that creates `Passengers` for each island. Make an `PassengerEvent` class that implements Event and has the following methods and fields:

- `private int island` – This is the island that this `PassengerEvent` will make PassengerObjects for.
- `public void run()` – This function will add a new `PassengerEvent` to our agenda for the same island at a time that is a random number between 5 and 10 (inclusive). It will also create a new `Passenger` at the correct pickupIsland and add it to the correct queue.
- You will need some way of setting the island this `PassengerEvent` is for.

Question: If I have n islands, how many `PassengerEvent`s will there be in my agenda? Why is this the case?
Must be answered in order to get checked off

The rest of this page is intentionally left blank

3 Testing PassengerEvent ★

Now that we have our `Passenger` and `PassengerEvent` classes made, lets add it to main. First, lets add a new `PassengerEvent` to our agenda. We'll only add one for now so it's easier to see correctness. Then, we'll want to have a loop to run the agenda for a period of time, say 10000 seconds. For an example of how to do this, look at `CarSim`. `CarSim` can be found on moodle in *Week 9 - Priority Queues, Discrete Event Simulation.*

If we hit run as it is, we don't have any useful output telling us that our code works. Let's edit our `PassengerEvent`'s `run()` function so that when we run our main, our output will be something similar to:

```
Passenger Event Island: 0, Time is: 0.0, Next Passenger in: 6
Passenger Event Island: 0, Time is: 6.0, Next Passenger in: 10
Passenger Event Island: 0, Time is: 16.0, Next Passenger in: 9
Passenger Event Island: 0, Time is: 25.0, Next Passenger in: 6
```

Hint: If you're getting a NullPointerException, are you instantiating everything?

Once you're sure that your `PassengerEvent` works, add the other `PassengerEvent`s for each island to the agenda.

The rest of this page is intentionally left blank

4 All Aboard CSCI1933

Having `Passengers` is great an all, but they still have no way to get from island to island! Lets fix that problem by creating a `Ferry` class. Here are the requirements for `Ferry`:

- The ferry can hold up to 60 `Passengers`. You can choose what kind of data structure you want to use to store these Passengers. Make sure that a max of 60 passengers can be stored and that **ONLY** `Passengers` can be stored. To protect our `Passengers` from anything dangerous (ex. Passengers leaving the ship midway or unwanted passengers boarding the ship at sea), our data structure should be `private`.
- `public boolean addPassenger(Passenger p)` – Adds a `Passenger` to this ferry. Returns true if the `Passenger` was added successfully.
- `public Passenger[] removePassengersAtIsland(int island)` – Removes all of the `Passenger`-s that are trying to get dropped off at a island and returns them in an array.
- `public boolean isFull()` – Returns true if ship is full.

Now our `Ferry` needs a way to move from island to island. We'll do this by creating a `FerryEvent`! The `FerryEvent` will have similarities to `PassengerEvent` but will work on a `Ferry` object instead of creating `Passengers`. Here are requirements for `FerryEvent`:

- `private int island` – This is what island the `Ferry` that this Event is for is at.
- `private Ferry ship` – This is the Ferry object that we will remove and board passengers on.
- `public void run()` – This function will first remove all `Passengers` that want to get off at our current island, board new `Passengers`, and move our ferry to the next island.
- You will need a way to set `island` and `ship`.
- Ferries will drop off `Passengers` before they pick them up.
- Conditions for departure are: there are no passengers available for pickup OR this ferry is full.
- It takes 60 seconds to get from one island to another.
- To move our Ferry nicely (so that passengers that arrive while boarding is commencing can also board), we will need to play with scheduling times of this event. Essentially, we will schedule this event for multiple times during a ferry's stop at an island for each *phase*: dropping off passengers, picking up passengers until a departure requirement is met, and traveling.

5 Finishing The Simulator And Collecting Statistics

We've written `Ferry` and `FerryEvent`, now it's time to use it. Edit the main method so that your `Ferry` will do it's thing. You can use some print statements to help verify that your `Ferry` is working as intended.

Now add some basic statistics gathering to the simulator. Track and print the following:

- The mean (average) number of people on the `Ferry` (after departure).
- The total number of people who got in line for the `Ferry`.
- The total number of people who arrived at their destination.

Think about what these statistics tell you about the system. IF we increase the simulation time, how do they change? Try changing some of the parameters such as the average arrival time, number of ferries, size of the ferries, etc. What conclusions can you draw? Is the simulation at equlibrium? Be prepared to talk about your conclusions in order to get checked off.

The rest of this page is intentionally left blank