Homework #3: Design and Implementation of a Transit Simulator

Due Sunday, February 8th at 11:59 p.m.

In this assignment you will design and build an extensible simulator for an urban transit system. When you are done, your homework solution will allow a user to simulate the behavior of buses and people (bus riders) in the transit system and observe how the transit system is affected by varying bus properties and rider behavior.

Your learning goals for this assignment are to:

- Demonstrate mastery of earlier learning goals, especially the concepts of information hiding and polymorphism, software design based on informal specifications, and Java coding and testing practices and style.
- Interpret, design, and implement software based on a UML interaction diagram.
- Use inheritance and delegation effectively to achieve design flexibility and code reuse.
- Discuss the relative advantages and disadvantages of alternative design choices.
- Gain experience working with a medium-sized application, including programming against existing interfaces and implementations.

This document continues by describing the basic requirements for your transit simulator, the resources we have provided, and explicit design challenges: simulation scenarios that your design must be sufficiently flexible to model. We finally discuss what you must turn in for this assignment, including all of your source code and a written discussion for the rationale for part of your design.

An extensible transit simulator

Fundamentally, the goal of a transit system is to move *entities* (such as people and buses) from one location to another in a consistent and predictable way. Buses are typically scheduled by a transit agency far in advance of their travel, and each bus's driver tries to approximately follow the schedule given the real constraints of vehicular traffic and passenger loading and unloading times. Because these constraints are not fully predictable, the actual behavior of a transit system can substantially deviate from its schedule.

Your transit simulator must model the location and behavior of buses and people as they use the transit system throughout a single day. When you are done, your simulator should be able to simulate an entire day and analyze the delays observed throughout the system (see below) from 4 a.m. to midnight. Specifically, it should be able to track the location of buses and people for that day while the simulation is running (which can be visualized using our graphical user interface (GUI), which we describe below).

Time in your transit simulation should proceed in discrete steps, with *every step representing a single second in the day.* In every step each entity in the world can act, altering its internal state and (possibly) changing locations.

In Appendix A we provide sequence diagrams that describe selected expected interactions between your simulator and the entities it simulates. Your solution must conform to the design visualized in those sequence diagrams, but will of course extend it.

A simulator can answer many different kinds of questions. For this homework you will measure the average delay experienced by bus riders throughout the day, which we call the *performance* of the transit system. The delay of a rider is the difference between her expected/scheduled arrival time and her actual arrival time at her destination, measured in seconds. (Arriving early is not considered to be a delay.) Your simulator should report the average delay over all riders that day. The simulation ends at midnight; any riders still traveling or actually stuck at a stop are discarded.

You have considerable freedom for how you simulate the behavior of buses and people. You do not need to explicitly simulate external entities such as non-bus vehicles or weather, but your buses and people may be implicitly based on the effects of such external entities. For example, your bus simulation might randomly delay buses during rush hour, the times of day when road traffic is heavy. Your simulation of people might similarly increase or decrease the overall number of bus riders based on arbitrary, unspecified events.

Overall, you must design your buses and people so that you can demonstrate modeling flexibility and code reuse with methods such as inheritance and delegation. We strongly recommend that you introduce appropriate interfaces and/or classes to achieve this flexibility and reuse; if your solution does not allow you to demonstrate modeling flexibility and code reuse with inheritance and delegation, you should rethink your design so that you can successfully demonstrate the learning goals of the assignment.

Simulating buses

Your simulation of buses must be loosely based on the sample transit schedule – the Oakland data set – we have provided on Piazza. In other words, each bus should progress along its trip from stop to stop as described in the transit schedule, but not necessarily at the times specified by the schedule. A bus's location should be represented simply as its latitude and longitude, which are provided for each stop in the schedule. You should generate each bus's simulated timing based on some combination of the schedule data plus external factors (such as random traffic delays) and the behavior of bus riders, which themselves can delay a bus. In your simulation, the *travel time* for each bus should be determined as 90% of the scheduled time (between stops in the transit schedule) plus other delays due to rider boarding/unboarding or other external factors in your model.

We recommend that you do not simulate specifically how a bus travels between stops. Instead, you should identify the travel time each bus needs to get to its next stop as the bus leaves a stop (including all delays and speedups). The bus may appear as jumping magically to the next stop after that travel time has passed; alternatively, if you want, you may simulate each bus as moving smoothly and linearly between stops at some speed based on the precomputed travel time.

Your design must provide flexibility to model a variety of bus characteristics and behaviors which we do not fully specify. One obvious dimension of flexibility is adding or removing bus trips from a schedule, which can be changed easily just by altering the transit data files. Other dimensions of flexibility might be the capacity of buses (single-length and double-length buses) or more complex notions of capacity; we discuss some of these dimensions of flexibility in our design challenges below.

When you are done, your simulation should allow a user to answer basic questions such as: If we double the frequency of the bus trips along some specified route, how does that affect the overall performance of the transit system?

Simulating people

As with your simulation of buses, your simulation of people must be loosely based on the behavior of real bus riders in a transit system. We have provided some utility methods that generate a distribution of Pittsburgh bus riders on which you may optionally base your simulation. Specifically, our methods select rider densities and random bus stops using a distribution of times at which bus riders leave during the day, and a distribution of bus stops used by riders as their source and destination.

Your simulation should insert a fixed number of bus riders into the virtual transit system at random times based on the distribution. For example, if 10000 riders are simulated over the entire day and 10% of riders leave between 8 a.m. and 9 a.m., then your system should insert about 16 riders per minute during that hour, which corresponds to about one rider every four seconds.

You should insert each rider at a random stop in the transit system with a random destination stop, both based on the provided distribution. (E.g., it is more likely that a rider leaves from CMU than from a random stop in Shadyside.) If there is no possible route from the source to the destination stop, the simulation should select new random source and destination stops.

After a rider is placed at a bus stop in the simulation, the rider's behavior is as follows.

The bus rider uses a route planner (such as from Homework 2) to determine an itinerary for her trip, and then follows that itinerary. Specifically, if the itinerary says to get on a 61C at some specified time, the rider will board the next 61C at the bus stop regardless of whether the next 61C is the intended bus mentioned in the itinerary, or an earlier or a later one. If the itinerary says to get off at Forbes and Murray to change to the 64, the rider will get off and wait for the next 64 bus regardless of how long that takes. The rider should follow the itinerary's route and not perform any replanning during the trip, even though alternative buses (such as the 61D instead of 61C) might speed the rider to her destination. Overall, your simulator should calculate the delay for a trip as being the actual (simulated) arrival time minus the scheduled arrival time from the original itinerary, or 0 if the rider arrived before her scheduled time.

Bus riders are always either at a bus stop or in a bus. Bus riders never walk to or between bus stops.

As with buses, your design must provide flexibility to model a variety of person characteristics and behaviors. Again one obvious dimension of flexibility is the number of bus riders, and their sources and destinations. Other dimensions of flexibility might include modeling riders with special needs such as loading or unloading a bike or using the wheelchair ramp; we discuss some of these dimensions of flexibility in our design challenges below.

When you are done, your simulation should allow a user to answer basic questions such as: If we double the number of bus riders (e.g. for a football game) or increase the fraction of riders with special needs, how does that affect the overall performance of the transit system?

Design challenges

Above we described some basic requirements of your transit simulation as well as some broad expectations for modeling flexibility. In this section we pose three specific design challenges which your simulation must be sufficiently flexible to model. In each case we describe some aspect of the transit domain that requires flexibility for an accurate simulation, as well as some example questions that should be answerable using your simulator. Your goal is to design your simulator to be able to model each scenario while achieving general design goals as well as any additional design goals that we specify in the scenarios below.

Challenge 1: Behavioral variations during boarding

Various aspects of the transit system can affect the system's performance, and your simulator should allow a user to determine the effect of various changes in the transit system.

In Pittsburgh, for example, buses usually open just the front door so that the bus driver can accept payment. Pittsburgh has recently started accepting payments via RFID cards in addition to cash payments; paying by an RFID card can speed up the payment process. Alternative transit designs might have ticket machines at bus stations or in the buses, so that riders can board the bus without delaying the boarding process. Another alternative might be for the transit agency to randomly check whether passengers have tickets, which could allow bus drivers to open all doors simultaneously and further speed the boarding process. All these considerations affect how long a bus waits at each stop.

Your system must be sufficiently flexible to model the following three dimensions of variability:

- D1: Different delays by travelers using different kinds of payment systems, such as cash, RFID cards, credit cards, and/or no payment on entry.
- D2: Travelers who need extra time to board/unboard a bus, such as riders with wheelchairs who need much extra time to board.
- D3: Variable boarding/unboarding delays or speedups such as when multiple bus doors are used.

Design your system so that it is easily extensible along these dimensions without requiring modification of the simulation code or existing entities in the simulation. Notice that these dimensions are independent and, in a real transit system, can potentially be composed. You may make reasonable assumptions about boarding times; there is no need for these assumptions to be accurate if these parameters can be easily changed in your simulation. When you insert agents into your simulation using your `PersonFactory` object, you may also estimate a distribution or experiment with different distributions (e.g., 20% of travelers have an RFID card, 5% have wheelchairs, etc.).

When you are done, you should be able to use your simulation to evaluate questions such as whether Pittsburgh's transit system should encourage RFID cards or whether they should replace the driver-checked payment with a ticket system.

To successfully achieve the learning goals of this assignment, your solution must positively demonstrate working extensions with substantial flexibility for each of the dimensions above. For this design challenge (behavioral variations while boarding) you must also describe how you achieve flexibility as part of your discussion (see below), and we also encourage (but do not require) you to demonstrate these extensions in the demo simulators you provide.

Challenge 2: Variable bus behavior

The transit schedule already contains reasonable estimates for how long buses need to travel between stops at various times of day. Your simulation, though, should be sufficiently flexible to model changeable bus behavior along several dimensions:

- D4: Delays added to travel times: The system must be able to simulate different delays, e.g. a 100% delay between specific stops due to construction, or an overall 10% delay due to bad weather conditions, or an overall 10% speedup at night, etc.
- D5: Configurable bus driver behavior: If a bus is early, the bus might wait at every stop or at specific stops. If a bus is late, the bus driver might attempt to speed (e.g. reducing the transit time by 5%), but perhaps not all buses are capable of speeding, depending on the bus size and engine.

Your simulation should allow a user to answer questions such as: How bad would a 5% slowdown affect the average delay for bus trips? Should bus drivers wait when they are early? As with the first design challenge, your solution must demonstrate flexibility by providing extensions for both of the dimensions above.

Challenge 3: Variable bus capacities and traveler needs

Real transit systems include different kinds of buses with different kinds of capacity, such as different amounts of space for persons, luggage, wheelchairs, and so on. Different travelers also have different needs that might only be met by some buses. Furthermore, some travelers travel in groups and are unable to split the group if the bus cannot fit their group. If a bus cannot accommodate a traveler, the traveler will not board the bus and will wait for the next bus.

Your simulation must be sufficiently flexible to model buses and travelers in the following dimensions:

- D6: Buses with different amounts of various kinds of capacity, including at least different capacity for persons, capacity for luggage, and capacity for wheelchairs. Two persons may use a wheelchair space if it is not used by a wheelchair, but neither persons nor wheelchairs may use luggage space for anything but luggage. If dedicated luggage space is not available, each piece of luggage may take the space that could otherwise be used by a single person.
- D7: Travelers with different needs, including at least different quantities of luggage, a wheelchair, and members of groups of travelers (who each might have additional needs).

A successful design will model buses and travelers in such a way that capacity-related decisions can be made in the simulation. You must be able to add new bus variations without changing existing buses or travelers, and you also must be able to add new traveler variations without changing existing buses or travelers.

Your solution should allow a user to answer questions such as: If each bus has a single wheelchair space, how would the performance of the transit system be affected by a 10% increase in travelers needing wheelchair space?

Discussing your work

There are many different ways to solve the design challenges above, each with different trade-offs. In at most one page of text, justify your design decisions for *Challenge 1: Behavioral variations while boarding.* You should explicitly describe how you achieved design flexibility for dimensions D1–D3.

Specifically, explain why you chose your design over *alternatives*, e.g. why you used inheritance instead of delegation or why you split concepts into distinct objects (if you made choices like this). Explain which *design goals* you tried to achieve and which *design principles* and design patterns you used to achieve those goals. Your discussion should *mention all design patterns that you used*, if any.

Turn in your discussion as a `rationale.md` (GitHub markdown) or `rationale.pdf` (PDF) file in your `homework/3` directory. Highlight (in bold or underline) all design goals, design principles, and design patterns in your discussion. Your document should not exceed one page, or about 500 words.

Our graphical user interface

The basic simulation can run silently without any interaction from 4 a.m. to midnight to produce a single performance report, the average trip delay. To help you visualize the transit system and debug, we provide a simple graphical user interface (GUI) that displays entities on the screen. You may modify the provided GUI if you wish.

Our GUI is initialized by creating a simulation and passing the simulation to the GUI's constructor. The simulation must provide a list of all entities that are part of the simulation, as well as a method returning the current time and the result of the analysis (a `String`). Each entity implements the `Entity` interface consisting of three required methods:

- `getImage()`: Returns an `ImageIcon` object that will be used to display the `Entity` on the GUI.

- `getLocation()`: Returns a `Location` object that consists of both latitude and longitude coordinates.
- `getName()`: Returns the name or some string of the entity. An example name for a bus might be "61D" or its current bus stop.

If the simulation is run with the GUI, the GUI triggers steps in the simulation. It will call the `Simulation#step()` method one or multiple times and will subsequently update the visual representation. Each call of `Simulation#step()` represents the passage of one second in the simulation.

Demonstrating your simulator

You have considerable design flexibility in this assignment. For full credit you must demonstrate how you achieved flexibility for the design challenges by preparing three separate simulation programs—separate classes with their own `main` methods—that can be readily executed. Place all three simulations in the `edu.cmu.cs.cs214.hw3.simulation` package.

Your simulation programs should demonstrate the flexibility of your simulator by choosing three very different scenarios. For example, one simulation might include heavy traffic, many riders with wheelchairs, and lots of luggage, while another simulation might model all bus riders as having RFID payment cards.

Testing

It can be very difficult to test complex interactions in a simulator because there can be many interdependencies between objects. We recommend that you reasonably test your implementation, but we do not specify any specific testing requirements beyond the following:

You must systematically test the capacity-negotiation mechanisms of *Challenge 3: Variable bus capacity and traveler needs*. Specifically, you should write unit tests to test different persons trying to board different buses, including your capacity-related extensions such as wheelchair users, groups of travelers, and luggage.

Use your judgment in the testing process. Coverage tools might help you analyze your test suite, but they cannot replace human judgment.

Evaluation

Overall this homework is worth 110 points. To earn full credit you must:

- Design your items to maximize information hiding and code reuse. Avoid copying-and-pasting code; instead, design your code with useful abstractions, class hierarchies, and/or delegation.
- Build a simulator that conforms to the behavioral specification of the sequence diagrams we provide in Appendix A.
- Meet the basic requirements we describe for your transit simulator and achieve the modeling flexibility along the dimensions described in the three design challenges.
- Specify meaningful contracts and test your implementation for at least the functionality to negotiate capacity when boarding buses.
- Make sure your code is readable and conforms to standard Java programming conventions.
- Discuss the rationale of your design related to Challenge 1, justifying your design compared to alternatives based on design goals and design principles, and also listing all used design patterns.

We will grade your solution approximately as follows:

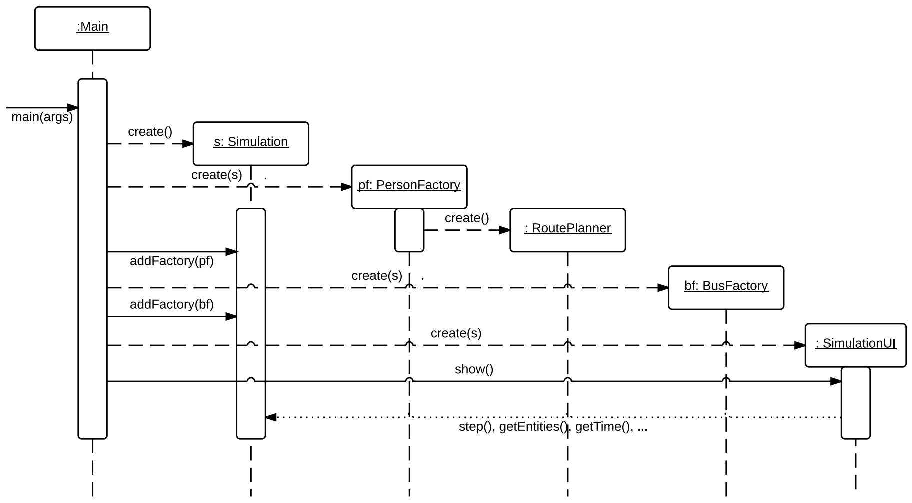
- Mastery of earlier learning goals, especially the concepts of information hiding and polymorphism, software design based on informal specifications, and Java coding, specification, and testing practices and style: 60 points
- Interpret, design, and implement software based on a UML interaction diagram: 10 points
- Use inheritance and delegation effectively to achieve design flexibility and code reuse: 30 points
- Discuss the relative advantages and disadvantages of alternative design choices: 10 points

Hints:

- The tasks and the UML diagrams are intentionally underspecified in some circumstances. In case of doubt, use your judgment or ask a question on Piazza. If you want to communicate your assumptions, use comments in the source code.
- You may create additional classes and method calls to implement the assignment, as long as your code implements the general interactions provided in the UML interaction diagrams.
- You may reuse all or parts of your implementations from prior homework assignments. If you do so, please copy all relevant code to your Homework 3 directory.
- Some simulator-specific requirements for this homework are similar to the pine tree simulation demonstrated in early lectures. You may reuse all or parts of that example simulator. If you do so, please copy all relevant code to your Homework 3 directory.
- As usual, avoid `instanceof` and static methods, where ever possible and reasonable.
- Remember that design patterns describe well known solutions to common problems. You may use want to use design patterns if you discover that design challenges fit your problem. Avoid however using design patterns everywhere or trying to use as many design patterns as possible; this typically leads to situations where design patterns are counterproductive. Many design challenges might suffice with simpler solutions without design patterns.
- You do not need to modify the `/.travis.yml` and `gradle.build` files, but you may perform reasonable changes. Do not however change `/settings.gradle` or the time limit in `/.travis.yml`. We enforce a timeout for all builds to balance Travis-CI capacity for all students.

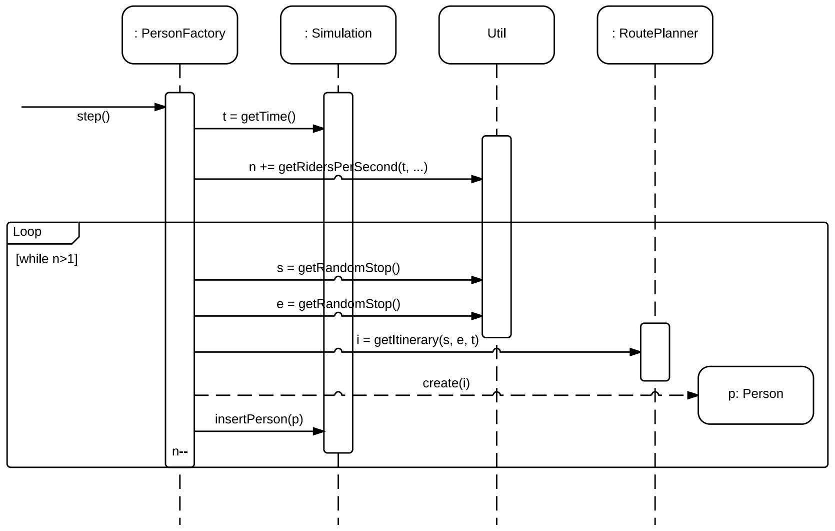
Appendix A: Partial UML Interaction Diagrams for a Transit Simulator

Initialization of the simulator:

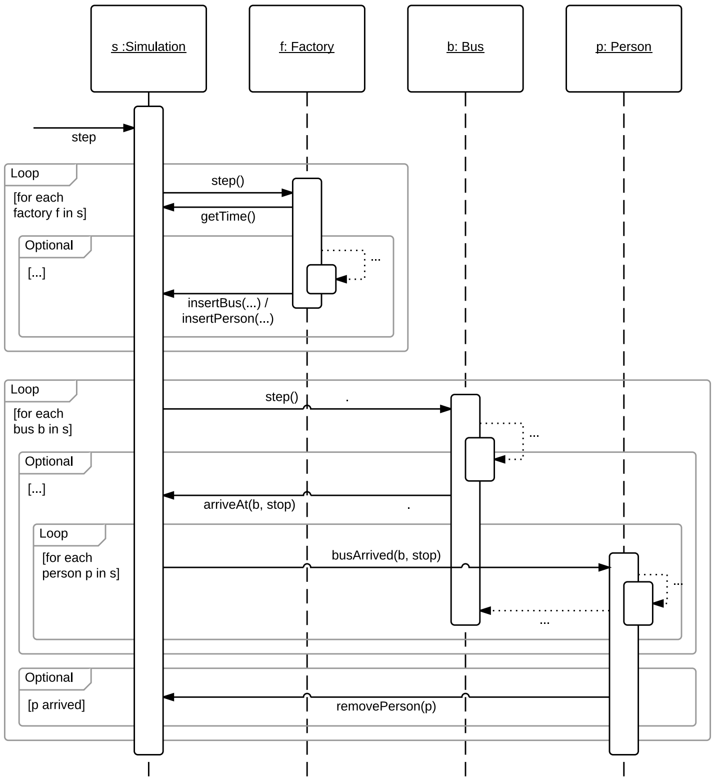


The diagram shows the core functionality but is intentionally not complete.
After this initialization process, the GUI will start communicating with the simulator, who will communicate with the factories and entities as illustrated below.

Random insertion of additional riders during a simulation step:



The major interactions for a single simulation step:



The diagram shows the core functionality but is intentionally not complete. Dotted lines are representing functionality or computations that are not specified and may differ in each implementation.