1 Project Description

Create a transport simulation on a two-dimensional map measured in miles. The simulation will model trucks of different sizes moving between different locations, picking-up shipments in one location and dropping them off in another location. There are no roads specified, instead, trucks can move from one point to another in a direct and incremental fashion.

The simulation is broken into hours of time, such that, each hour, trucks are doing one of the following things:

1. moving towards a destination,
2. picking up a shipment at a specified source location,
3. unloading a shipment of goods at a specified destination, or
4. waiting for an empty loading dock at a destination to unload a shipment.

Similarly, at each hour the different locations will be tracking the situation at the loading docks and picking the next truck to load and/or unload any cargo at that location.

Once the simulation is configured from a file, a central clock object will tick off the hours, during that hour each truck's action method will be executed by iterating through using a user-created list containing all trucks. To ensure that the trucks implement the correct methods, the following interface will be implemented. Similarly, every location will be examined to perform any needed actions and record any required information.

```
public interface Schedule {

  // Called each hour, allowing the object to perform an action.
  public void action();

  // Will store the object's current information into a log file.
  public int log_status();
}
```

At the start of the simulation trucks, warehouses, and shipments will be randomly generated. The trucks and warehouses, represented as objects of the appropriate class will be randomly placed on the map. Once the trucks and warehouses are placed, all shipments will be created (as objects) at each warehouse based on a randomly generated travel manifest for each truck.

Each warehouse will have between one to three loading docks, randomly determined, where trucks can drop off and pick up shipments. Trucks will come in four different types, capable of carrying between two to five loads of cargo. Trucks will move at one of four different speeds, where: a five-load truck moves one mile per hour, a four-load truck moves two miles per hour, and so on.

Shipments will be randomly determined in sizes of one, two, or three size units. The total number of shipment units cannot exceed the load size of the truck that is moving the shipment. For example, a size five-load truck will be able to carry one two-unit shipment and one three-unit shipment or two two-unit shipments and one one-unit shipment. As shipments are created, a unique identification code will be created starting from zero and incremented with each new shipment generated.

As previously indicated, at the start, empty trucks are created randomly around the map with randomly generated shipment manifests. The manifest of shipments will be randomly determined with each entry containing a source of the shipment and a destination. Trucks will perform pickups with each pickup being stored towards the front of the truck, with later shipments blocking earlier shipments. Thus, deliveries will be performed in a Last-in First-out fashion, and only the most recently loaded shipment will be delivered next.

The order of pickups will be based on warehouse distance from the truck's current position and the order of shipment creation identified by the sequential identifier. This combined criteria will be ordered by distance in ascending order and id in descending order, indicating that nearby pickups and newer shipments have a greater priority than those farther away and older. This ordering could change as the truck moves, thus priority will be adjusted with each new shipment added.

When a truck reaches a warehouse for pickup or drop-off it must wait for a loading dock to become empty. If other trucks are waiting for a loading dock, then the truck must wait its turn. Thus, trucks must access loading docks in the order of arrival.

The simulation will be finished when the last truck has made its last delivery.

You do not need to create any graphical interface. Whenever the central clock ticks off an hour, the simulation should write out (in text) information about each truck, each shipment, and each warehouse. All debugging and execution information generated by your simulation should be written to a log file. Also, there should be as little information as possible written to `System.out` and `System.err`, only what is needed. During the simulation, your program should write to a data file on the truck, shipment, and warehouse activities during the simulation. This data should be formatted into a comma-separated file.

Lastly, do not forget to unit-test as many classes as possible, and comment on all the classes in the project.

2 Functional Design Requirements

Your program must implement the following requirements.

1. All specified interfaces must be implemented.
2. All data structures will be designed and implemented by the programmer as **generic** classes. Thus, the Java API cannot be used in your program. Also, each data structure should be unit-tested, along with other classes that can also be unit-tested.
3. The design must be well divided across several classes where each class has specific functionality.
4. Along with the implementation, a report will be provided specifying what data will be tracked in the simulation and how the system will define simulation efficiency.
5. Input configuration will be stored in a file and read into the simulation at the beginning of the model run. There should be a logging file, capturing all activity coming out of the system and can be used for debugging. All output data will be written to data files.

3 Some Advice

- **Design first!**
- **Get something small to work, then make it more complex.**
- **Break the functionality into parts and test those parts.**
- **For the multiple model runs, this can be automated with a class that runs the model for different configurations.**

What to turn in:

1. A design document in Word. The document should include at least the following information:

   (a) Your plan of generating random trucks, warehouses, and shipments' schedules for the simulation. How would each file look like? What is the format of the generated data?

   (b) The definition of each class. You need to specify the properties of the objects that the class represents and the potential behaviors (methods) of the objects. As this is only a design document, you do not need to write the full definition of the methods, but you do need to specify the header of a method. For each method/class, comment how your design addresses the project requirements.

   (c) The format of the logs.

   (d) Essential algorithms in pseudo code.

   (e) Unit test cases for each class. For each testing case, you need to specify the input and the expected output. Again, you do not need to provide the implementation details for the test methods.

   The goal of this design document is for you to understand the whole project. When you finish your design document, it should be clear how to implement each module of the code.

   Due date: Friday 4/19 10:00 pm. Full grade: 5 points.

2. Final implementation and a README text file for reflection of your project. Compress all documents into a zip file named `Firstname_Lastname_P3.zip` containing all your source code for the project. Include the Javadoc tag @author in each class source file.

   Note: the reflection text file should contain the following information:

   (a) The functionalities that you planned to implement.

   (b) The functionalities that you actually finished.

   (c) Why did you fail to implement some of the functionalities? How could you fix the problem?

   (d) What is the lesson learned from the project?

   Due date: Sunday 5/5 10:00 pm. Full grade: 10 points.

   Grading for the final implementation:

   - Program compiles: required for a grade
   - Documentation and style: up to −2 points
     Make sure that your projects are commented on properly. This includes file headers, method headers, and appropriate inline comments in the source code and the answers to all the questions in the reflection file.
   - Correctness of your Interface/classes: 8 points
   - Unit-testing and explanation of your tests: 2 points