**ESS 201 Programming in Java**
**T1 2020-21**
**Lab 6**
**4 Oct 2020**
**Version 4**

*This is an update to the description provided in Version 3. The first part describes the changes made in this version. Versions 3, 2 and 1 are reproduced later for reference. The corresponding source files are provided in the zip file named Lab6b-v4.zip. Please review them to understand the changes, which are also partly explained below.*

**Version 4 changes:**

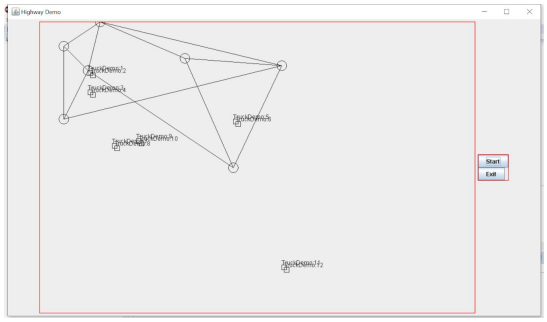
Changes to base classes: There are no changes to Network, Truck, Highway, Location classes. As discussed in class, getHighways of class Hub has been changed to public.
Some clarifications:

- all speeds are in screen-coordinates (or pixels) per second
- Hubs have distinct locations, so you don't need to worry about hubs that are 0 distance apart

Changes to Display and Driver classes: Significant revamp has been done to these classes, though it should not affect the rest of base classes, and hence no impact on derived classes you implement. Two major changes:

- A graphical display class using the Java Swing package has been implemented. This is available as SwingDisplay.java. In DemoDriver, you can switch between SwingDisplay and TextDisplay by commenting out the appropriate statement in the initialization part
- The Driver and Display classes have been restructured to follow the MVC or Model-View-Controller style of design for UI applications, including adding an interface AppControl. This makes it easier to add and manage graphical UI (such as the one here with Swing). You can use these "as is" - it should not affect the rest of the code.
- In addition, the DemoDriver code has been reorganized to improve modularity and remove dependence on static variables and methods.

So, if you set up DemoDriver to use SwingDemo.java, you should be able to run it like before, once the input data has been processed. You will see a Java UI open up (may be in minimized form). Open this, and you should see a window like the one below. Click on Start to start the simulation. When the position of Trucks change, you should be able to see the changes here.
**Again, note that none of this should require changes to your derived classes.**



**Version 3 changes**:

1. The base classes Hub and Truck extend Thread, so that each instance can run as different threads. As a consequence, the earlier start() method has been renamed as run() - which is the method that is invoked when a Thread is ready to be executed the first time. The run() methods essentially execute a loop, where they call the process/update method of the class, go to sleep for some time, and then continue with the loop.
    a. Note: you should ideally not override the run() method. If you do, make sure the base run() method is called as the last step of your method
    b. This also required that we change the method getName to getTruckName, since this clashed with an existing final method of the Thread class
2. The duration of sleep for each instance is now a “set” method - currently set from the driver class. You can play around with this as needed for your development/testing
3. The draw methods of the base classes have been updated
4. The driver class DemoDriver has been changed so that main reads in data from System.in to set up the instances of various network elements and trucks.
    a. A sample file, lab6In.txt, is provided as part of the zip file. The actual file used in the demo evaluation will likely be different
5. There are a few other minor changes to make all the classes consistent.

**Naming convention:**

To allow us to integrate classes from different students, we need to exactly comply with certain naming conventions:

1. Create a package (and any sub-packages if needed) with unique names. To simplify this, your package should be named demoNNNNN where NNNNN are the last 5 digits of your roll no.
2. The truck names should also be unique to each student, so that we can track the progress of each truck. Hence, the name of each truck should start with “TruckNNNN” where NNNNN are the last 5 digits of your roll no

Note: you can test your code by creating a copy of your package, give a different name to the package copy, and thus test the use of multiple factories.

**Version 2 changes:**

1. All base classes have been made abstract and empty default methods have been made abstract
2. Network has also been made an abstract class, and the Factory is expected to also generate a derived class of Network
3. For clarity, the following is the expected flow:
    a. Truck moves from start location to the nearest hub
    b. When it is moving on a road/highway, it checks if the next hub has capacity, and if so, adds itself to the Hub (using the add(Truck truck) method of Hub)
    c. When the truck reaches the Hub closest to the destination location, it starts moving towards the destination location. No capacity checks here
    d. When a truck is added to a Hub, the hub then manages the truck. It checks on the highway that will take the truck towards the destination, and adds the truck to the highway (using the add(Truck truck) method of the highway
    e. At the same time, the truck is notified that is has entered the highway through a call to its enter(Highway ) method.
4. Static methods have been added to Network and Location for convenience
5. We can assume there will be only one instance of Network in a given program. So, the static methods can be safely invoked
6. So, the main design and coding work would be in your derived classes of Truck, Hub, Highway and Network.
7. Base classes should not be modified. In the final demo, you will be provided the .class files of the base classes that should be used. If any changes are needed, updates will be provided as new versions to the whole class.

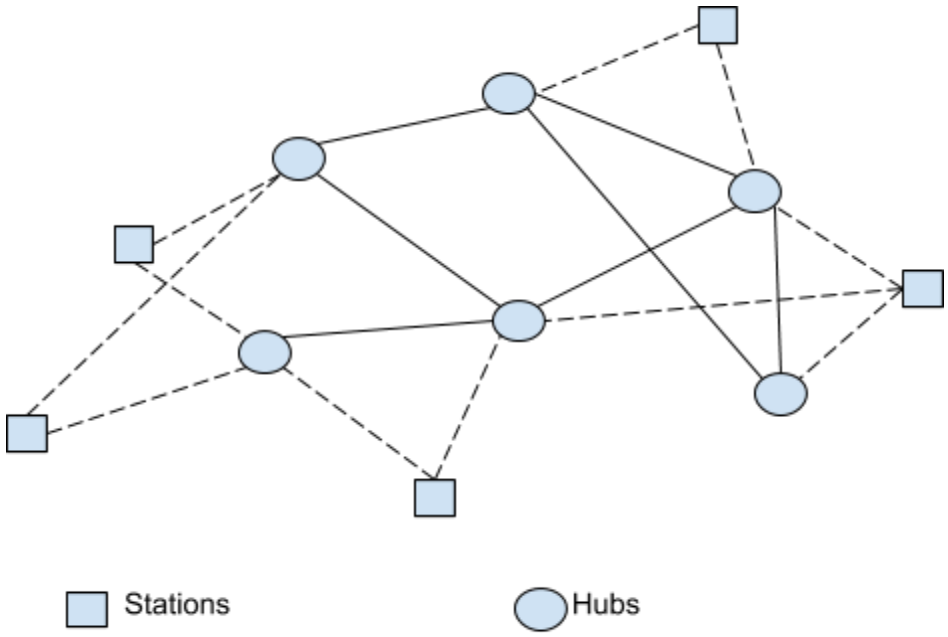
**Version 1 Description**

*This is the same as what was in the v1 document*

Consider a logistics company that manages a large volume of packages. It has a number of Stations distributed all over the map. Packages can be shipped from any station to any other station. To simplify things, all packages from a given station to the same destination are loaded onto one or more Trucks. Stations are connected to Hubs, which are the nodes of a Highway network. Thus a Truck that goes from a source Station to a destination Station is routed as follows:

- move to a Hub that is close to the source Station
- Navigate from Hub to Hub using Highways to the Hub that is close to the destination station.
- Move from this Hub to the destination Station

The figure below shows a typical network.




We have the following constraints:

- Trucks move at a certain maximum speed between Hubs and Stations and at a different maximum speed between Hubs (on a Highway)
- A Highway has a fixed capacity. At any given instant there is a maximum number of trucks that can be on the Highway. If a truck reaches a Hub, and it needs to get onto to Highway, it has to wait at the Hub until that Highway has spare capacity
- Hubs have a max capacity. At any given instant, there are a max number of trucks it can process, and that can be waiting at the Hub
- A truck travelling towards a Hub has to wait (on the road/highway) if that Hub is at full capacity
- To manage scale, entities like Hub, Highway, Truck (and their derived classes) keep minimal state information. For example, Truck does not keep track of the full route - just the source, destination locations, and previous, current and next Hubs.

Different companies manage the hubs and trucks. However, they all conform to some basic design. To model this, we have the following:

- Base classes Hub, Truck,Highway (we will not model Station and Road since they can be implicit)
- Each company creates its own derived classes of these, and specifically derived classes of Hub, Highway and Truck.
- A Network class maintains information about all the elements of the network
- A given network can have instances of different sub-types of these classes, corresponding to the different companies involved.
- To help manage the construction of these objects, we have a *Factory class* for each student. This class creates instances of the appropriate derived type for that student. Details are in Factory.java

The files Hub.java, Truck.java, Highway.java contain definitions of the base classes.
In this assignment, each student represents one company, and creates derived classes of these base classes. For the demo/evaluation, we will create a system that mixes up (derived class instances of) Hub, Truck, Highway etc of different students and show then working seamlessly together.

**The demo/simulation**:

1. We create multiple instances of Hub, Truck, Highway, and add these to a Network.
2. We define source and destination locations for Trucks and start time. Trucks start moving at their start time.
3. Each Truck navigates from start location to nearest Hub, and from there through other Hubs (and Highways) to reach the destination location/station.
4. On their route, if the next Hub is full, they wait until the Hub becomes free for them to move there. Similarly for getting on to Highways
5. Trucks can move at any speed that does not exceed the speed limit for the road/highway. Of course, we would like them to move as fast as possible towards their destination. Set up the demo such that a typical route can be covered in 10-20 secs end to end.

You are provided code of the base classes, as well as the “main” or driver class.
The code is written so that it is independent of how the objects are displayed. This is abstracted through a Display class, and you are provided a text output version of this. A graphical version will be provided later.

Subsequent versions of this document (and code) will detail some of these further.