Jackson Eshbaugh

April 19, 2024

CS 150: Data Structures and Algorithms

Professor John Dahl

Project 3: Transport Simulation

I. Class Representations

Schedule. Firstly, there will be a schedule interface that allows for all parts of this simulation to be run on a "schedule."

```
public interface Schedule {
    // Called every hour.
    void action();
    // Stores the object's current information into a log file.
    void log();
}
```

Point. This class will represent a point on a 2D grid. It will have an x and y coordinate.

```
public class Point {
  private int x, y;
  // Creates a new point at position (x, y).
  public Point(int x, int y) {...}
  // Returns the point's x-coordinate.
  public int getX() {...}
  // Returns the point's y-coordinate.
  public int getY() {...}
  // Returns the distance from this point to the given point.
  public int distanceTo(Point point) {...}
}
```

Truck. Trucks will have an id, a position point, load limit, speed (which is determined by the load limit), and a (cargo) hold (which is essentially a stack), also with a manifest.

```
public class Truck implements Schedule {
  private Point position;
  private int id, loadLimit, speed;
  private Stack<Shipment> hold;
  private Manifest manifest;
  private Warehouse destination;
  // Creates a new truck with loadLimit at position (x, y), setting speed base
  public Truck(int id, Point point, int loadLimit) {...}
  // Get the truck's id.
  public int getId() {...}
  // Set the truck's manifest.
  public void setManifest(Manifest manifest) {...}
  // Returns the truck's manifest.
  public Manifest getManifest() {...}
  // Loads a shipment onto the truck. Throws a TruckFullException if the truck
  public void load(Shipment shipment) {...}
  // Offloads the last loaded shipment from the truck. Throws TruckEmptyExcept
  public Shipment unload() {...}
```

```
  // Sets the truck's destination. See algorithm below.
  // Would be private, but is public for testing purposes.
  public void setDestination() {...}
  // Sets the truck's position.
  public void setPosition(Point position) {...}
  // Required methods from the Schedule interface:
  @Override
  public void action() {...}
  @Override
  public void log() {...}
}
```

Warehouse. Warehouses will have an id, a position point, and a certain number (between 1–3) of loading docks. The docks will have a boolean value to determine if they are occupied and a truck object to represent the truck currently at the dock. Also, if all docks are occupied, any trucks that arrive at the warehouse will have to wait until a dock is available. These trucks will be stored in a truckQueue.

Note: Once a shipment has reached its destination, it will be removed from the truck and not added to the destination warehouse. Additionally, shipments aren't actually removed from the warehouse; they are actually loaded in the truck from the manifest.

```
public class Warehouse {
  private Point position;
  private int id;
  private List<Dock> docks;
  private Queue<Truck> truckQueue;
  // Creates a new warehouse at the position given by point with the given num
  public Warehouse(int id, Point point, int docks) {...}
  // Returns the warehouse's id.
  public int getId() {...}
  // Join the queue of trucks waiting for a dock.
  public void joinQueue(Truck truck) {...}
  // Returns the number of docks at the warehouse.
  public int getNumDocks() {...}
  // Returns the number of docks with a truck at the warehouse.
  public int getNumOccupiedDocks() {...}
  // Returns the warehouse's position.
  public Point getPosition() {...}
  // Returns the number of trucks in the queue.
  public int getNumQueuingTrucks() {...}
  // Required methods from the Schedule interface:
  @Override
  public void action() {...}
  @Override
  public void log() {...}
  private class Dock {
    private boolean occupied;
    private Truck truck;
    // Creates a new dock.
    public Dock() {...}
    // Assigns a truck to the dock.
    public void dock(Truck truck) {...}
    // Removes the truck from the dock.
    public void remove() {...}
    // Handles the dock (to be called by action())
    public void handle() {...}
    // Returns whether the dock is occupied.
    public boolean isOccupied() {...}
    // Returns the truck at the dock.
    public Truck getTruck() {...}
  }
```

```
}
```

Manifest. The manifest is a list of shipments that are to be delivered. This is implemented as some type of List (probably an ArrayList). The manifest will have a pickUps list of shipments that are to be picked up. The manifest will also have a `nextPickup()` method that will return the next shipment to be picked up, and a`pickUp()` method that will remove the next shipment from the list of shipments to be picked up.

```
public class Manifest {
  private List<Shipment> pickUps;
  // Creates a new manifest.
  public Manifest() {...}
  // Adds a shipment to the manifest.
  public void add(Shipment shipment) {...}
  // Called when a truck picks up a shipment. Removes the shipment from the ma
  public Shipment pickUp(Position position) {...}
  // Returns the next shipment in the manifest.
  public Shipment nextPickup(Position position) {...}
  // Returns the number of shipments in the manifest.
  public int size() {...}
}
```

Shipment. Shipments will have an id (increments from 0), a destination Warehouse, an origin Warehouse, and a size (which is between 1–3 units).

```
public class Shipment {
  private int id, size;
  private Warehouse origin, destination;
  // Creates a new shipment with the given id, destination, and size.
  public Shipment(int id, Warehouse origin, Warehouse destination, int size) {
  // Returns the shipment's id.
  public int getId() {...}
  // Returns the shipment's size.
  public int getSize() {...}
  // Returns the shipment's origin.
  public Warehouse getOrigin() {...}
  // Returns the shipment's destination.
  public Warehouse getDestination() {...}
  // Returns the distance from the shipment's origin to the given position.
  public int distanceTo(Point point) {...}
}
```

TransportSimulation. This is the main class that will run the simulation. It will have a time variable to keep track of the current time. Additionally, it will have a maxX and maxY to represent the size of the grid, and a number of trucks, shipments (per truck) and warehouses to be used in the simulation.

```
public class TransportSimulation {
  private int time;
  private int maxX, maxY;
  private int numTrucks, numWarehouses, numShipments;
  private List<Truck> trucks;
  private List<Warehouse> warehouses;
  // Creates a new transport simulation.
  // Sets up the simulation with the given number of trucks, warehouses, and s
  public TransportSimulation(int maxX, int maxY, int numTrucks, int numWarehou
  // Alternatively, create an empty simulation and add trucks, warehouses, and
  public TransportSimulation(int maxX, int maxY) {...}
  // Adds a truck to the simulation.
  // Before adding, a truck should be assigned a manifest.
  public void addTruck(Truck truck) {...}
  // Adds a warehouse to the simulation.
  public void addWarehouse(Warehouse warehouse) {...}
```

```
  // Runs the simulation for a given number of hours.
  public void run(int hours) {...}
}
```

II. Algorithms

Truck. Trucks have 4 actions to take during an hour of simulation:

1. Moving towards a destination.
2. Picking up a shipment from a warehouse. (Next hour, the truck will undock and move towards the next destination, or drop off another shipment, or load a new shipment.)
3. Dropping off a shipment at a warehouse. (Same as above.)
4. Waiting at a warehouse.

Picking up and dropping off shipments will both be handled by the **Warehouse** class.

So, the `action()` method will check the current state of the truck and perform the appropriate action. The `log()` method will print the current state of the truck.

```
action():
  // truck should keep moving towards destination.
  if destination is not null:
    action = "(" + truck.getPoint().getX() + ", " + truck.getPoint().getY() +
    move(destination.getPoint().getX(), destination.getPoint().getY())
    log()
  if at destination:
    join the queue of trucks waiting for a dock

move(x, y):
  // move towards the destination
  // Use similar triangles to calculate the new position, based on the truck's
  destinationDistance = truck.getPoint().distanceTo(x, y)
  if destinationDistance <= truck.getSpeed():
    truck.setPoint(x, y)
    action = action + "(" + x + ", " + y + ")"
  else:
    // calculate the new position based on the truck's speed
    ratio = truck.getSpeed() / destinationDistance
    (int) newX = ceil(truck.getPoint().getX() + (x - truck.getPoint().getX())
    (int) newY = ceil(truck.getPoint().getY() + (y - truck.getPoint().getY())
    setPoint(newX, newY)
    action = action + "(" + newX + ", " + newY + ")"

setPoint(x, y):
  this.point = new Point(x, y)

log():
  print "Truck %id%: %insert type of action here% [Destination: (%x%, %y%); Ca
```

Additionally, the truck needs to set its destination to the next shipment in the manifest, or to the next dropoff, whichever is closer. Again, with any ties, the newer shipment will be given priority.

```
setDestination():
  nextPickup = manifest.nextPickup()
  nextDropoff = hold.peek()
  if nextPickup is null:
    destination = nextDropoff
    manifest.remove()
  else if nextDropoff is null:
```

```
    destination = nextPickup
  else if nextPickup.distanceTo(truck.getPoint()) < nextDropoff.distanceTo(tru
    destination = nextPickup
  else if nextPickup.distanceTo(truck.getPoint()) > nextDropoff.distanceTo(tru
    destination = nextDropoff
  else:
    // if distances are equal, newer shipment is given priority
    if nextPickup.id < nextDropoff.id:
      destination = nextPickup
    else:
      destination = nextDropoff
```

Warehouse. Warehouses will have a `load()` method that will load a shipment onto a truck. If all docks are occupied, the truck will be added to the `truckQueue`. The `unload()` method will remove a shipment from the truck and update the dock status. The `action()` method will check the status of the docks and trucks and perform the appropriate action.

```
action():
  for each dock:
    dock.handle()
    else:
      // Dock is empty
      if !truckQueue.isEmpty():
        assign truckQueue.next() to dock

log():
  print "Warehouse %id%: " + action
```

Dock. Docks will have an `dock()` method that will dock a truck at the dock and a `remove()` method that will remove the truck from the dock. The `isOccupied()` method will return whether the dock is occupied, and the `getTruck()` method will return the truck at the dock.

```
handle():
  if isOccupied():
    if truck needs to unload:
      Shipment s = truck.unload()
      action = "truck " + truck.getId() + " unloaded shipment " + s.getId() //
      log()
    else if truck needs to load:
      Shipment s = truck.getManifest().pickUp()
      action = "truck " + truck.getId() + " loaded shipment " + s.getId() // .
      truck.load()
      log()
    else:
      // Truck is done with its business at the warehouse.
      truck.setDestination()
      remove()
```

Manifest. The manifest will keep track of all shipments and their priorities. It will have a `nextPickup()` method that will return the shipment with the highest priority. The manifest will also have an `add()` method that will add a shipment to the manifest and a `pickUp()` method that will remove a shipment from the manifest.

Determining Priority. The priority of a shipment is determined by the distance from the truck to the shipment. If two or more distances are equal, a newer shipment is given priority over an older one.

```
nextPickup():
  Shipment next = null
  for each shipment in pickUps:
    if next is null:
      next = shipment
```

```
    else if shipment.priority > next.priority:
      next = shipment
    else if shipment.priority == next.priority:
      if shipment.id > next.id:
        next = shipment
  return next

add(shipment):
  pickUps.add(shipment)

pickUp():
  Shipment next = pickUps.remove(nextPickup())
  truck.load(next)
```

Shipment. Shipments will have a `distanceTo()` method that will return the distance from the shipment to a given position. This simply invokes the `distanceTo()` method of the warehouse's `Point` instance variable:

```
distanceTo(Point point):
  return point.distanceTo(point)
```

Point. The `Point` class will have a `distanceTo()` method that will return the distance from the point to a given position. This will simply use the distance formula:

$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

```
distanceTo(Point point):
  return ceil(sqrt((point.getX() - x)^2 + (point.getY() - y)^2))
```

TransportSimulation. The simulation will run for a given number of hours. During each hour, the simulation will call the `action()` and then `log()` method of each truck and warehouse. Once instantiated, the simulation can be started using the `run(int)` method.

Note: This allows for a truck to arrive at a warehouse, and if there is an available dock, the truck can dock and load/unload.

```
run(int hours):
  int count = 1
  while(count < hours):
    println("Hour " + count)
    for each truck:
      truck.action()
      truck.log()
    for each warehouse:
      warehouse.action()
      warehouse.log()
    count++
```

Main Class. The main class will instantiate the simulation and run it for a given number of hours. The `main()` method will take an argument at command line to determine wether to randomize the simulation (overwrite a config) or to read in the config.

```
main(String[] args):
  if args.length > 0 && args[0] == "random":
    randomize()
  readConfig() // read in the config file, assign values to variables
  TransportSimulation simulation = new TransportSimulation(...) // pass in the
  simulation.run(hours)
```

III. Log Format

The log will be printed in the following format:

```
Time: 0
Truck 0: (0, 0) -> (1, 1) [Destination: (2, 4); Cargo Hold: (1/3)]
Truck 1: (0, 0) -> (2, 2)
Truck 2: (0, 0) -> (3, 3)
Truck 3: (1, 1) Loaded Shipment id 0 at Warehouse 0 [Destination: (1, 1); Carg
.
.
.
Time: 3
Truck 0: (1, 1) -> (2, 4) [Destination: (8, 2); Cargo Hold: (1/3)]
Truck 1: [Manifest Empty] [Destination: no destination; Cargo Hold: (0/3)]
Truck 2: (3, 3) -> (2, 2) [Destination: (1, 1); Cargo Hold: (1/3)]
Truck 3: (1, 1) Loaded Shipment id 1 at Warehouse 0 [Destination: (1, 1); Carg
```

Each truck will print:

- Its ID
- Its current position (and move if applicable)
- Its destination
- The number of shipments in its cargo hold

This makes the following type of log:

```
Truck %id%: %insert type of action here% [Destination: (%x%, %y%); Cargo
Hold: (%current hold size%/%max hold size%)]
```

Or, if the action is at a warehouse:

```
Warehouse %id%: truck %id% %insert type of action here% [Destination:
(%x%, %y%); Cargo Hold: (%current hold size%/%max hold size%)]
```

For the action, here are the possibilities:

- `(%prevX%, %prevY%) -> (%x%, %y%)`: The truck is moving from one position to another.
- `Loaded Shipment id %id% at Warehouse %id%`: The truck is loading a shipment at a warehouse.
- `Unloaded Shipment id %id% at Warehouse %id%`: The truck is unloading a shipment at a warehouse.
- `[Manifest Empty]`: The truck has no shipments in its manifest.

IV. Configuration File

Type. The configuration file will be a plaintext file named `simulation.properties`.

Syntax. The configuration file will have the following syntax:

```
setting=value
stringSetting="value"
# Comment
```

Settings. The configuration file will have the following settings:

- Simulation Settings
    - `hours`: The number of hours the simulation will run. Default is `8`.
    - `numTrucks`: The number of trucks in the simulation. Default is `4`. Note: The size of the trucks is randomized.
    - `numWarehouses`: The number of warehouses in the simulation. Default is `3`.
    - `minShipments`: The minimum number of shipments on a truck's manifest. Default is `3`.
    - `maxShipments`: The maximum number of shipments on a truck's manifest. Default is `10`.
    - `maxX`: The maximum x-coordinate of the simulation. Default is `10`.
    - `maxY`: The maximum y-coordinate of the simulation. Default is `10`.

Note: The configuration file will be read in by the `TransportSimulation` class and used to set the simulation settings. It will be overwritten by the `main()` method of the whole project if the user passes in the command line argument "`random`":

```
java TransportSimulation random
```

V. Testing

Here are test cases for each class (aside from straightforward getters and setters):

- **Point**:
    - Test the `distanceTo()` method.
- **Shipment**:
    - Test the `distanceTo()` method.
- **Truck**:
    - Test the `load()` method.
    - Test the `unload()` method.
    - Test the `action()` method.
    - Test the `move()` method.
    - Test the `setDestination()` method.
- **Warehouse**:
    - Test the `Dock` class.
    - Test the `dock()` method.
    - Test the `remove()` method.
    - Test the `isOccupied()` method.
    - Test the `handle()` method.
    - Test the `joinQueue()` method.
    - Test the `getNumOccupiedDocks()` method.
    - Test the `getNumQueuingTrucks()` method.
    - Test the `action()` method.
- **Manifest**:
    - Test the `add()` method.
    - Test the `nextPickUp()` method.
    - Test the `pickUp()` method.

Point. The `distanceTo()` method will be tested by creating two points and calculating the distance between them.

1. P(0, 0) and Q(3, 4) should have a distance of 1.

2. P(0, 0) and Q(0, 0) should have a distance of 0.
3. P(1, 7) and Q(0, 3) should have a distance of sqrt(36 + 9).

Shipment. The `distanceTo()` method will be tested by creating a shipment and a point.

1. P(0, 0) and Shipment(3, 4) should have a distance of 1.
2. P(0, 0) and Shipment(0, 0) should have a distance of 0.
3. P(1, 7) and Shipment(0, 3) should have a distance of sqrt(36 + 9).

Truck. The `load()` and `unload()` methods will be tested by creating a truck and warehouse. The truck will load a shipment and then unload it. The test will pass if the shipment is loaded and then unloaded. The `load()` method will also be tested to ensure that the truck cannot load more shipments than its capacity (should throw a TruckFullException), and the `unload()` method tested to ensure that the truck cannot unload when empty (should throw a TruckEmptyException). Additionally, test the `setDestination()` method to ensure that the truck's destination is set correctly.

1. Truck with capacity 3 loads 3 shipments.

- Verify Before: Truck has 0 shipments.
- Verify After: Truck has 3 shipments, and those have the same ID as the shipments loaded.

1. Truck with capacity 3 loads 4 shipments (should throw TruckFullException).

- Verify After: the TruckFullException is thrown.

1. Truck with capacity 3 unloads 3 shipments.

- Verify Before: Truck has 3 shipments.
- Verify After: Truck has 0 shipments.

1. Empty truck unloads (should throw TruckEmptyException).

- Verify After: the TruckEmptyException is thrown.

To test the `setDestination()` method:

1. [Pickup is closer] Truck at (0, 0) with cargo destined for (3, 4) and a pickup at (1, 1) calls `setDestination()`.

- Verify After: Truck's destination is (1, 1).

1. [Dropoff is closer] Truck at (0, 0) with cargo destined for (1, 1) and a pickup at (3, 4) calls `setDestination()`.

- Verify After: Truck's destination is (1, 1).

1. [Pickup and dropoff are equidistant] Truck at (1, 0) with cargo (id = 0) destined for (0, 0) and a pickup (id = 1) at (2, 0) calls `setDestination()`.

- Verify After: Truck's destination is (2, 0). (This is because in the case of equidistance, the larger ID is the tiebreaker)

1. [Pickup and dropoff are equidistant] Truck at (1, 0) with cargo (id = 1) destined for (0, 0) and a pickup (id = 0) at (2, 0) calls `setDestination()`.

- Verify After: Truck's destination is (0, 0). (This is because in the case of equidistance, the larger ID is the tiebreaker)

To test the `move()` method:

1. Truck at (0, 0) with speed 1 and destination (1, 1) calls `move()`.

- Verify Before: Truck is at (0, 0).
- Verify After: Truck is at (1, 1).

1. Truck at (0, 0) with speed 1 and destination (0, 0) calls `move()`.

- Verify Before: Truck is at (0, 0).
- Verify After: Truck is at (0, 0).

1. Truck at (0, 0) with speed 3 and destination (1, 1) calls `move()`. [Test potentially overshooting the destination]

- Verify Before: Truck is at (0, 0).
- Verify After: Truck is at (1, 1).

1.

Truck at (0, 0) with speed 2 and destination (4, 4) calls `move()`.

- Verify Before: Truck is at (0, 0).
- Verify After: Truck is at (2, 2).

To test the `action()` method:

1. Truck at (0, 0) with speed 1 and destination (1, 1) calls `action()`.

- Verify Before: Truck is at (0, 0).
- Verify After: Truck is at (1, 1).

1. Truck at (1, 1) with speed 1 and destination (1, 1) calls `action()`.

- Verify Before: Truck is at (1, 1).
- Verify After: Truck is at (1, 1).
- Verify After: Truck is in the queue for the warehouse at (1, 1).

Warehouse. Test functionality of the `Dock` class. Test `dock()` and `remove()`, and `isOccupied()` methods. Test that the dock can only load and unload one truck at a time. Also ensure that `joinQueue()` works and that trucks are loaded out of the queue after `action()` is called. Test for both scenarios where trucks are left over in the queue and scenarios where all trucks have docked.

1. Dock with capacity 1 docks 1 truck.

- Verify Before: Dock is empty.
- Verify After: Dock has 1 truck.
- Verify After: `isOccupied()` returns true.

1. The above dock removes the truck.

- Verify Before: Dock has 1 truck. (Same as above)
- Verify After: Dock is empty.
- Verify After: `isOccupied()` returns false.

To test the `handle()` method:

1. Dock with capacity 1 docks 1 truck with no more buisness at the dock and calls `handle()`.

- Verify Before: Dock has a truck.
-

Verify After: Dock is empty.

1. Dock with capacity 1 docks 1 truck with a shipment to load and calls `handle()`.

- Verify Before: Dock has a truck.
- Verify After: Dock has a truck.
- Verify After: `truck` has loaded a shipment.

1. Dock with capacity 1 docks 1 truck with a shipment to unload and calls `handle()`.

- Verify Before: Dock has a truck.
- Verify After: Dock has a truck.
- Verify After: `truck` has unloaded a shipment.

1. Dock with capacity 1 docks 1 truck with a shipment to load and unload and calls `handle()`.

- Verify Before: Dock has a truck.
- Verify After: Dock has a truck.
- Verify After: `truck` has unloaded a shipment.

1. The above dock calls `handle()` again.

- Verify Before: Dock has a truck. (as above)
- Verify After: Dock has a truck.
- Verify After: `truck` has loaded a shipment.

1. The above dock calls `handle()` again.

- Verify Before: Dock has a truck. (as above)
- Verify After: Dock is empty.

To test the `action()` method, consider the whole `Warehouse` class (using `joinQueue()`):

1. Warehouse with 1 dock and 1 truck (with empty manifest) in the queue. `action()` is called.

- Verify Before: There are 0 docked trucks.

- Verify Before: Queue has 1 truck.
- Verify After: There is 1 docked truck.
- Verify After: Queue has 0 trucks.

Extend this to a multi-dock scenario:

1. Warehouse with 2 docks and 2 trucks (with empty manifests) in the queue. `action()` is called.

- Verify Before: There are 0 docked trucks.
- Verify Before: Queue has 2 trucks.
- Verify After: There are 2 docked trucks.
- Verify After: Queue has 0 trucks.

1. Warehouse with 2 docks and 3 trucks (with empty manifests) in the queue. `action()` is called.

- Verify Before: There are 0 docked trucks.
- Verify Before: Queue has 3 trucks.
- Verify After: There are 2 docked trucks.
- Verify After: Queue has 1 truck.

1. Call `action()` again.

- Verify Before: Handled above.
- Verify After: There is 1 docked truck.
- Verify After: Queue has 0 trucks.

1. Call `action()` again.

- Verify Before: Handled above.
- Verify After: There are 0 docked trucks.
- Verify After: Queue has 0 trucks.

1. Warehouse with 2 docks and 3 trucks (with manifests with one shipment each and with one dropoff each) in the queue. `action()` is called.

- Verify Before: There are 0 docked trucks.
- Verify Before: Queue has 3 trucks.
- Verify After: There are 2 docked trucks.
- Verify After: Queue has 1 truck.

Verify After: Both docked trucks have unloaded a shipment.

1. Call `action()` again.

- Verify Before: Handled above.
- Verify After: There is 1 docked truck.
- Verify After: Queue has 0 trucks.
- Verify After: The docked truck has unloaded a shipment.
- Verify After: The previously docked trucks have loaded a shipment each.

1. Call `action()` again.

- Verify Before: Handled above.
- Verify After: There are 0 docked trucks.
- Verify After: Queue has 0 trucks.
- Verify After: The previously docked truck loaded a shipment.

Manifest.

To test `add()`:

1. Create a manifest and add a shipment.

- Verify Before: manifest.size() returns 0
- Verify After: manifest.size() returns 1

To test `nextPickUp()`:

1. Create a manifest with 3 shipments. Call `nextPickUp()` 3 times.

- Verify Before: manifest.size() returns 3
- Verify After: manifest.size() returns 3
- Verify After: manifest.nextPickUp() returns the shipment that was added second

To test `pickUp()`:

1. Create a manifest with 3 shipments. Call `pickUp()` 3 times.

- Verify Before: manifest.size() returns 3
- Verify for Each: manifest.size() returns 2, 1, 0
- Verify for Each: manifest.pickUp() returns the shipment that was added first, second, third