

Board

The code you have written may feel a long way to go from the picture in figure 1. But, like most challenging projects, this will not get built in a day and as long as your `Location` and `Vehicle` classes are passing tests, you should be good to go. **Before proceeding, make sure that you have tested your move method on both a horizontal and vertical vehicle in both positive and negative directions and that it is giving you back the expected results.**

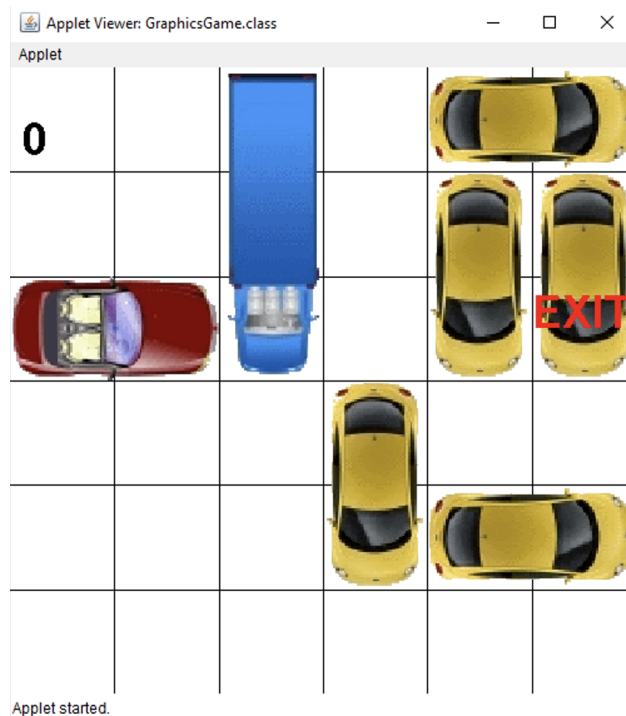


Figure 1: A screenshot of the Traffic Jame game.

The Text-Based Version

In this assignment, we will continue to build the text-based version while we learn to work with graphics in Java. Your goal for this assignment is to pass the board stress test. This is a set of tests which have already been written for you and which look like the output in figure 2.

The Programming Requirements

For this assignment, you will only have to deal with one class. While it may seem trivial, the `Board` class is anything but that. The `Board` is the workhorse and heart of the program.

```

..ta.
ccta.
..t..
...B.
...B.

Ok, now testing some moves...
These should all be true
Moving truck down true
Moving truck down true
Moving truck down true
Moving lower auto up true
Moving lower auto up true
And these should all be false
Moving truck down false
Moving the car into truckfalse
Moving the car into truckfalse
Moving lower auto up false
Moving lower auto up false
Moving upper auto up false
Moving upper auto up false
just moving some stuff around
...B.
.cca.
..t..
..ta.
..ta.

```

Figure 2: A screenshot of the test cases in the Board class.

A Board will hold all of the vehicles and will decide whether or not a Vehicle will be able to move to a new location. Figure 3 shows how Board is related to Location and Vehicle.

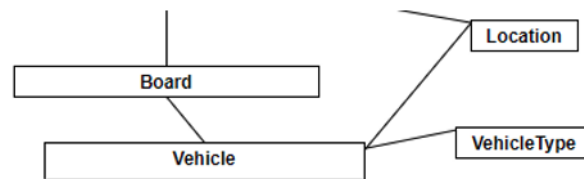


Figure 3: Relationship between classes in this assignment.

If you were not able to finish the previous assignment, you should continue to work on getting those classes right, as Board will be much more difficult. **Board needs Location and Vehicle to be correct.** You will still be able to get some credit for having a completed Location and Vehicle if you were not able to get credit on the first assignment.

The Plan of Attack

Step 1) Make sure your Location and Vehicle are correctly implemented

At the very least your Location and Vehicle should pass the test code provided to you in the previous assignment to ensure that they are correctly implemented. If this does not work, you should go back and look at the additional resources for help in how to implement it. You can also leverage office hours and tutoring for this.

Step 2) Understand how the Board class works

The Board class is the heart of Traffic Jam and it is one of the trickiest classes to implement. For this class, you will need to think about how to store all of the vehicles and we will do so in a 2D-array of Vehicles. You will also have to set up the vehicles on the 2D-array (you should check if the coordinates are in bounds) and check to see whether you can move them. If you have a working `locationsOn` function, this will help you immensely, since given some Vehicle you will always be able to figure out what locations it is occupying. **This means that Board should not check whether or not a Vehicle is vertical.** Vehicle already does this for you!

Based on the previous paragraph, your Board class should have a Vehicle 2D-array as one of its instance variables, with each row and column that a vehicle occupies storing a reference (pointer) back to that Vehicle object. Since there are no pointers in Java per se, you only have to worry about creating a 2D-array of vehicles and Java will do the rest.

As an example, assume that we want to use a 3x3 board and that we have the player's car (the "auto" VehicleType) and another car (the "car" VehicleType). We want the auto (assume its address is 0x40) to be placed vertically at the top of the right-most column, while the car (assume at 0x15) is placed horizontally on the top row. Given those addresses for these two vehicles, the 2D Vehicle array would look like this:

0x15	0x15	0x40
null (0x0)	null	0x40
null	null	null

When printed out, this will appear in a way that is much friendlier to the user (the printing mechanism has been implemented for you already):

```
c c a
. . a
. . .
```

The Board.java file also contains a main function which will run a small test on your Board class by creating a board object, adding a few vehicles, and trying to move them. This test is by no means comprehensive, and it is important that you try to think of more scenarios that you should test. The Board class also includes the `toString` method and the name of the instance variable for the 2D-array of vehicles, which is not-so-creatively called `board`.

The next steps outline the methods that you will be in charge of, like writing the constructor and methods to add Vehicles on a board, to return a vehicle that occupies a certain `Location`, to check if there is a vehicle on a certain location, and to figure out whether or not a vehicle can actually move a certain number of spaces legally.

Step 3) Setup the constructor for the Board class

To help you with this step, you should look at the test code that was provided for you to see how the `Board` will be created and how vehicles will be added to the board. Then review the previous step about how the board should be implemented.

Step 4) Write `getVehicleAt()` and `addVehicle()`, as well as any other methods listed in the diagram below (except `moveVehicleAt` and `canMoveAVehicleAt`)

These methods will simply test your knowledge about how the board class is represented. Neither of these methods is complex to write, but they do require an understanding of how everything is related. **This also requires that you try to leverage `locationsOn` in your code for `addVehicle`, since it can help you to find the locations that need have pointers to your new vehicle.** If `getVehicleAt` is called on a location that is empty, it should return `null`.

Step 5) Write `moveVehicleAt`, `canMoveAVehicleAt`, and then debug your code to pass as much of the Board class test as possible

The biggest hurdle you will have is in implementing `moveVehicleAt` and `canMoveAVehicleAt`, so you should tackle those last. The trick is to try to break down each one of these functions so that you can reduce the amount of if-statements that you have.

These methods become much simpler if you leverage the `locationsOn` and `locationsPath` functions you wrote for `Vehicle`, as well as other methods in the `Board` class. **Do not just try to jump into writing these methods.** Spend some time thinking, planning, and writing ideas out on paper. Think of some test cases before you implement the solution and make sure that you understand what should happen in different scenarios. It may also be useful to leverage your implementation of `addVehicle` here.

UML Diagram

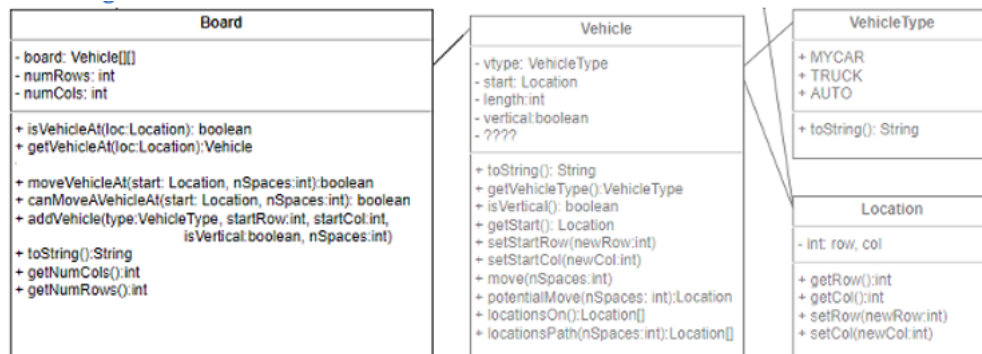


Figure 4: A UML diagram of the classes for this assignment.

Deliverables & Advice

Once you are done, submit the entire project as a .zip file like you did with the previous assignment. Please start early and if something does not make sense, do not hesitate to ask.