

Console

Now that you have worked on the underlying structure of the game, this next phase is dedicated to constructing the console version that you will need to make Traffic Jam into a more complete project.

The Text-Based Version

Here is a small set of interactions with the text-based version of the game. After this assignment, we will turn our attention to building the graphical version of the game.

```

123456
-----
A|..t.aa
B|..t.aa
C|cct.aa
D|...a..
E|...aaa
F|.....

What's your next move? (Moves so far: 0)
Please enter a location using the letter for row and number for column: B3
How many spaces would you like this vehicle to move?
2

123456
-----
A|....aa
B|....aa
C|cct.aa
D|..ta..
E|..taaa
F|.....

What's your next move? (Moves so far: 1)
Please enter a location using the letter for row and number for column: D4
How many spaces would you like this vehicle to move?
-1

```

```
123456
-----
A|....aa
B|....aa
C|cctaaa
D|..ta..
E|..t.aa
F|.....
```

What's your next move? (Moves so far: 2)

Please enter a location using the letter for row and number for column: E6

How many spaces would you like this vehicle to move?

-1

```
123456
-----
A|....aa
B|....aa
C|cctaaa
D|..ta..
E|..taa.
F|.....
```

What's your next move? (Moves so far: 3)

Please enter a location using the letter for row and number for column:

In this situation, we are asking the user to type in a location for a car they wish to move. After the user types in the location, they are then prompted to type in the number of spaces (positive or negative) they want that vehicle to move. If they picked a correct location, the vehicle would then move to that location. As a reminder, the rules in Traffic Jam are that a car:

- must stay in the bounds of the board,
- can only move in the direction it is currently facing,
- and must not collide with any of the other cars as it moves to its new position.

For example, if we try to move the auto on E4 -3 spaces (3 spaces to the left), the computer would simply print out that it's not possible to make that move and then repeat the question, like below.

```

123456
-----
A|....aa
B|....aa
C|cctaaa
D|..ta..
E|..taa.
F|.....

```

What's your next move? (Moves so far: 3)

Please enter a location using the letter for row and number for column: E4

How many spaces would you like this vehicle to move?

-3

Sorry, but that vehicle can't be moved to where you want. Please try again.

```

123456
-----
A|....aa
B|....aa
C|cctaaa
D|..ta..
E|..taa.
F|.....

```

What's your next move? (Moves so far: 3)

This game continually prompts the user for their next move until they get their car (represented as the cc) to the other side of the board. At this point the game prints a congratulatory message and then exits.

The Programming Requirements

Your task in this assignment is to take the four Java files (Location, Vehicle, VehicleType, and Board) and to integrate them with two new Java files – `ConsoleGame` and `Level` – so

that the work you have done so far really becomes a game. To help you along in the process, the following section explain the files in detail and provide you with a small plan of attack.

The two new files in this assignment each represent a different part of the game: **ConsoleGame** is directly responsible for all of the text input and output (and is largely complete already), whereas the **Level** class holds a single board configuration and largely acts as an intermediary between the Board and the Game. The Level also keeps track of whether the board has been solved and how many moves have been made. Figure 1 shows how the different Java files are related to each other.

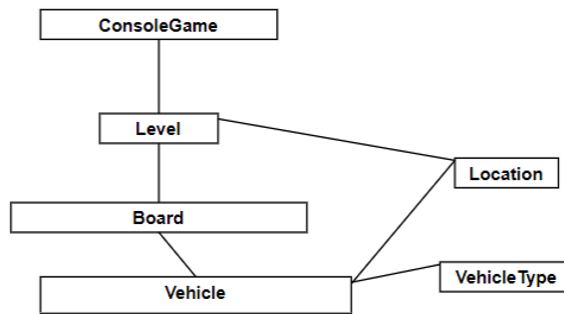


Figure 1: Relationship between classes in this assignment.

The Plan of Attack

Step 1) Test your Board and Vehicle classes

Board.java As mentioned in the previous assignment, this class is the heart of the program and one of the trickiest files. You may have already noticed that there are certain complications to making the code work, particularly with `canMoveAVehicleAt` and `moveVehicleAt`. Before you proceed, make sure to test your **Board** and **Vehicle** classes to make sure they work. (You may want to think of more scenarios that you would like to test.) Please feel free to discuss these classes with me or our TA if you are stuck or have any questions.

As a reminder, your **Board** class should have methods that will add Vehicles on a board (which you will hard-code for this assignment), access a vehicle from a certain **Location**, check to see if a vehicle occupies a certain location, and figure out whether or not a vehicle can actually move a certain number of spaces or whether there is a car in the way. To help you with this, you should use the method `locationsPath` from **Vehicle**, which returns an array of all of the **Locations** that a vehicle would cross and / or occupy should it move that number of spaces.

Step 2) Implement the Level class

`Level.java` `Level` serves as the intermediary between `Board` and the `Game`. This can be confusing because `Level` will not have much additional code but becomes a layer between the `ConsoleGame` and `Board`. This will make it easier for us to switch between having a console program and a graphics program in the future. Currently, the program only has one level, so we will manually set up the level by calling the appropriate methods to add vehicles to the board.

`Level` has a board, as well as some additional functionality like keeping track of the number of moves, setting up the level, and knowing whether or not the user has passed the level. There are a few other methods in `Level`, but the best way to write these is to have `Level` simply ask `Board` for its solution to the problem. For example, figure 2 shows how the `ConsoleGame`, `Board`, and `Vehicle` classes interact with each other when the program starts.

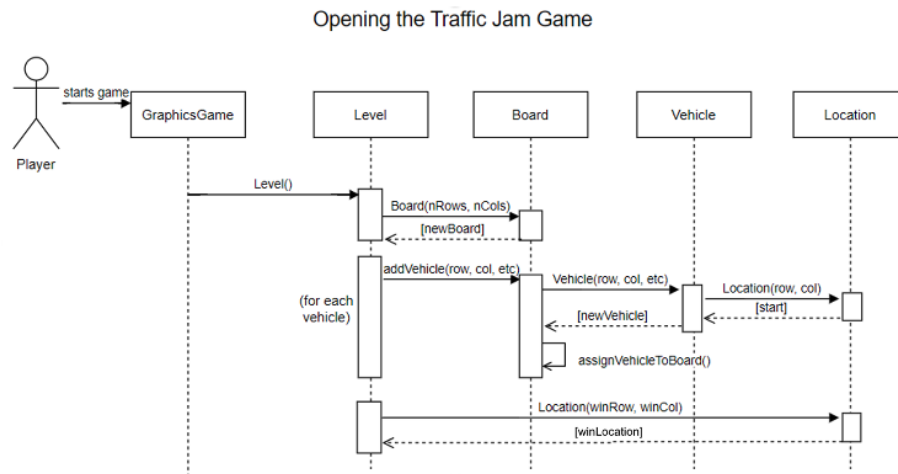


Figure 2: Interactions between classes when opening the game.

Step 3) Implement the ConsoleGame class

`ConsoleGame.java` This is the main driver for the program. The starter code already handles most of the text input for you. You will have to understand how to use the methods given to you, like `GetInteger` and `GetLocation`, and write the method `playGame()`, which should continually loop through asking for a location and an integer and moving a vehicle (when appropriate) until the user passes the level. Because of the size of this program, we are only going to focus one level for now, so `playGame` can simply print out a congratulatory message and finish.

UML Diagram

In order to help you keep track of everything that you need to write, we have provided you with a basic UML model below. This shows the methods that you should write (including what to pass in and to return) and the instance variables that you should store in more detail. It does not include how you should write your constructors.

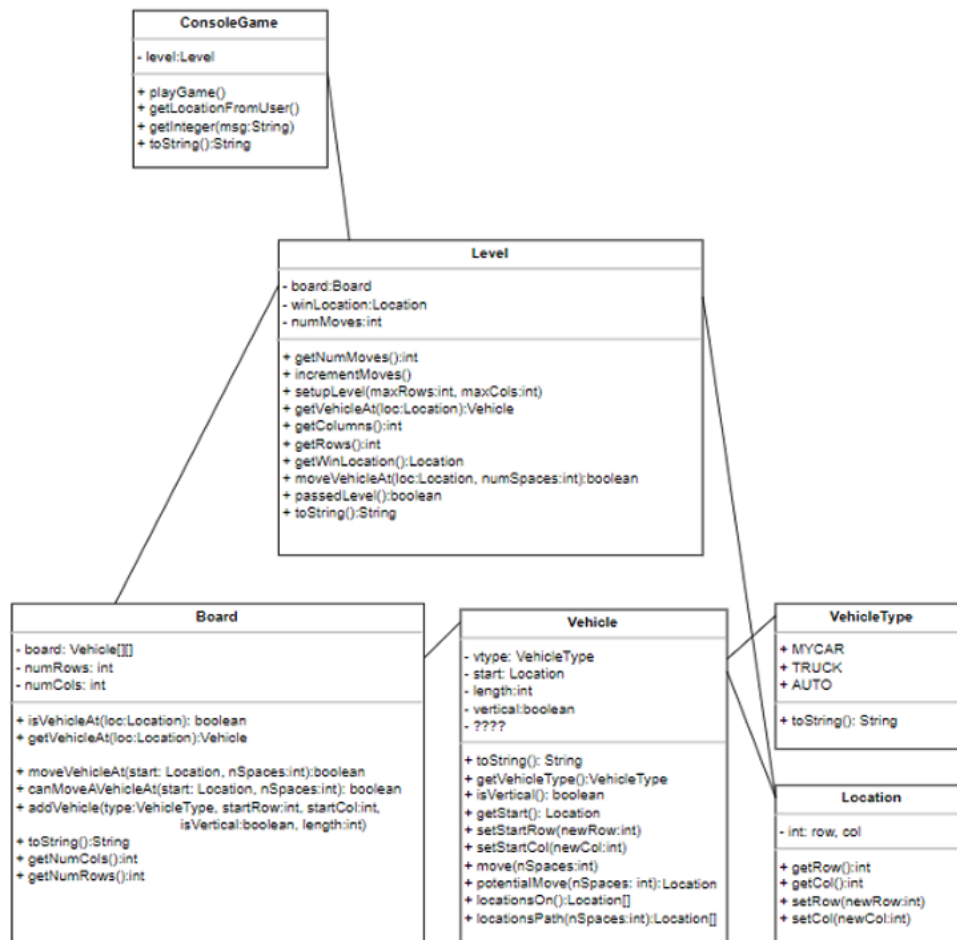


Figure 3: A UML diagram of the classes for this assignment.

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.