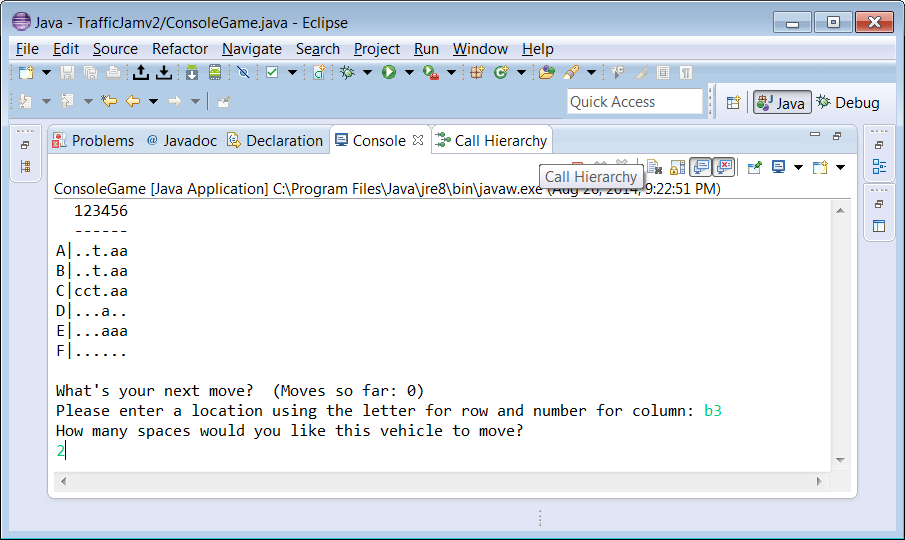
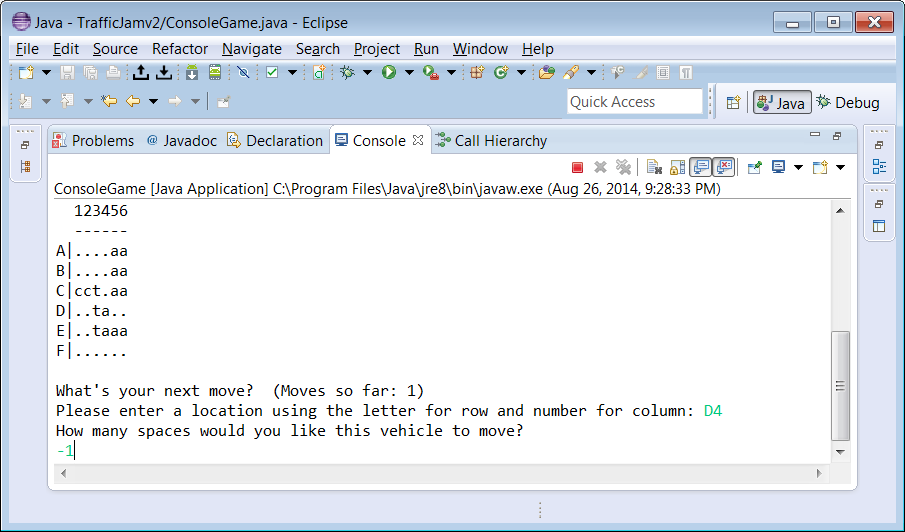
# COMP 55 – Traffic Jam Console Version

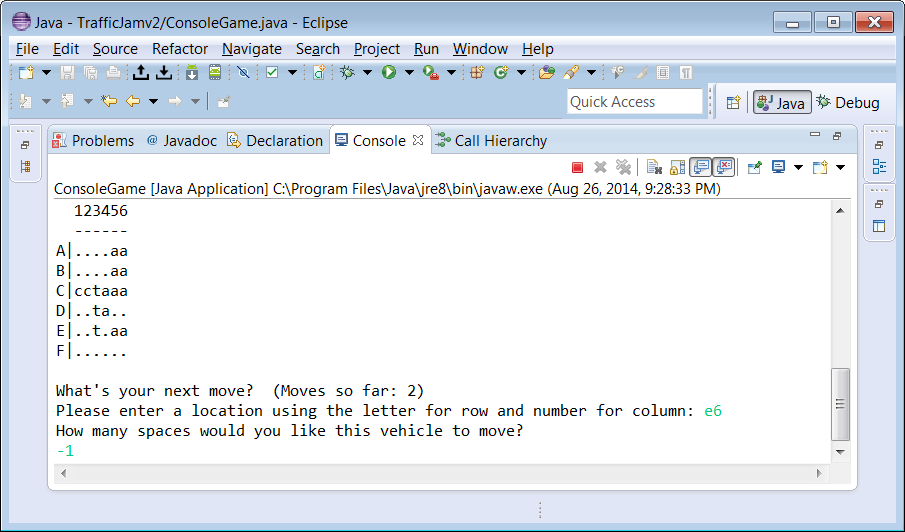
Now that you’ve worked on building the board, vehicle and space and have the underlying structure built, this next phase is dedicated to constructing the console game version that you’ll need to make traffic jam into a COMP 53 like project.

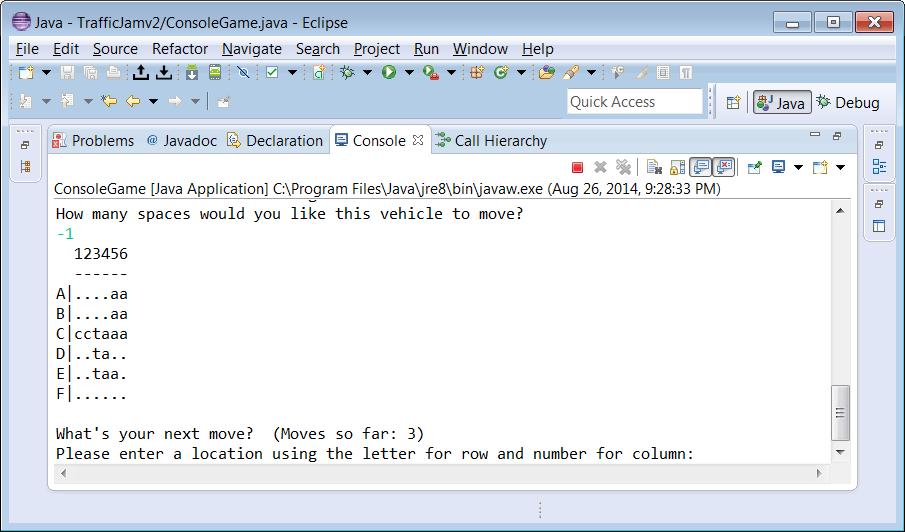
## The Text-Based Version

Here is a small set of text interactions with the text-based version, which is showing the same board as the graphics picture shown in previous assignments.

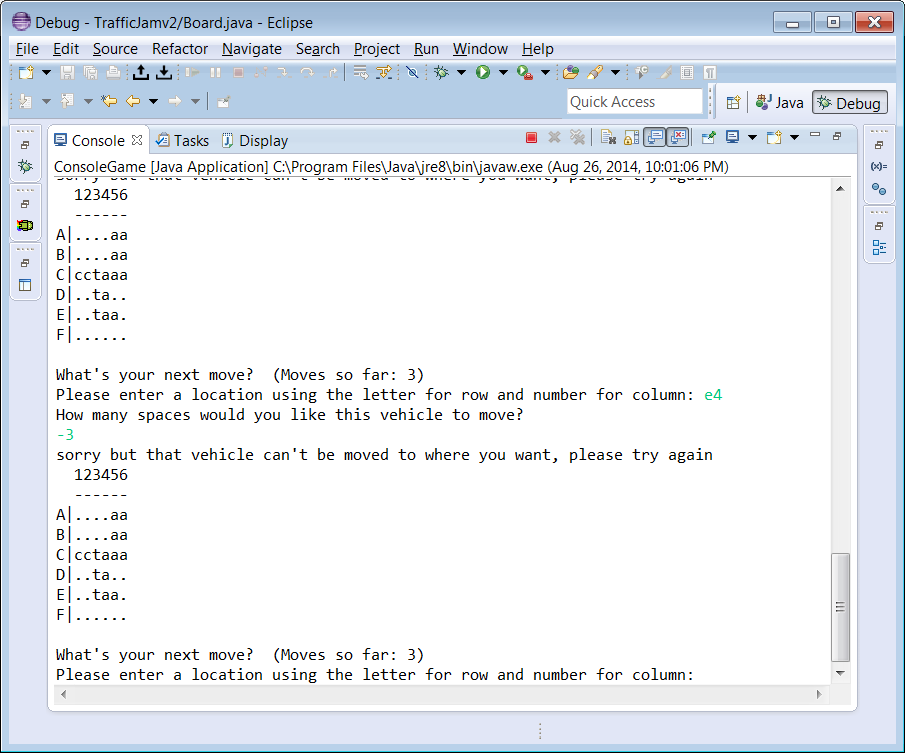








In this situation, we are asking the user to type in a location for a car they wish to move. After the user types in their location, they are then prompted to type in the number of spaces (positive or negative) they want that vehicle to move. If they picked a correct location, the vehicle would then move to that location. The rules in traffic jam are that a car must stay in the bounds of the board, can only move along its initial orientation axis, 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.

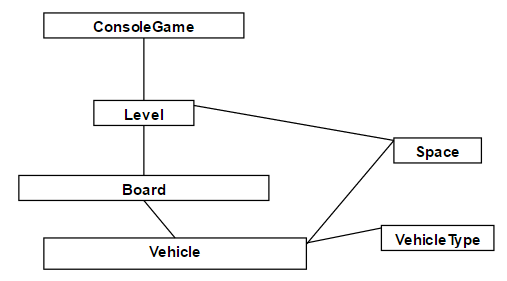


This game continually prompts the user for their next move until they get their car (represented as the cc) to the other side of the board. At this point the game prints a congratulatory message and then exits.

## The programming requirements

Your job is to take your four java files (Space, Vehicle, VehicleType & Board) and integrate them with ConsoleGame and with Level so that the work you’ve done so far really becomes a game. To help you along in the process, I’m going to explain the files in detail and give you a small plan of attack.

The two new java files each represent a different part of the game. For example, the ConsoleGame is directly responsible for all of the text input and output and is largely done for you, while 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 or not the board has been solved, and how many moves have been made. Here’s an updated diagram that shows how the different java files are related to each other.



This diagram represents how these six files are related to each other. If you implement these files (or finish implementing them in some cases, you will be able to have a fully functioning console version of Traffic Jam. Let’s look at each one in detail in the way you should implement them.

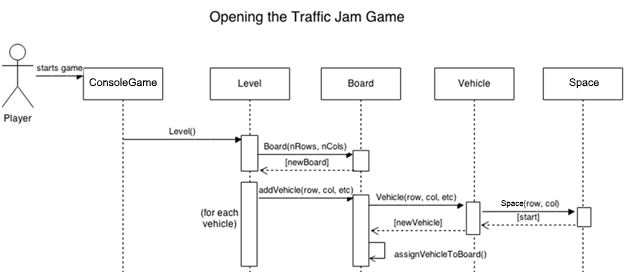
## The Plan of Attack

### Step 1) Re-Implement/Re-examine your Board and Vehicle class

*Board*.java – This is the heart of the program, and definitely one of the trickiest if not the trickiest file. You may have already noticed that there are certain complications to making the code work, particularly with canMoveNumSpaces and moveNumSpaces. For some of you, you’ll need to add the code for these and check to make sure it works once you discuss it with me. Again, the best implementation for board that I found was to have a 2D array of vehicle pointers, with each row and column that a vehicle occupies storing a pointer back to that vehicle. This test is by no means comprehensive, and it’s important that you try to think of more scenarios that you would like to test. Also included in this class is the toString method and the name of the 2d array of vehicles, which we’ll creatively call board. You’ll be in charge of writing methods that will add Vehicles on a board, access a vehicle from a certain Space, 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 spacesOccupiedOnTrail from Vehicle, which returns back an array of all of the Spaces that a vehicle would cross and/or occupy should they move that number of spaces.)

### Step 2) Implement the Level Class

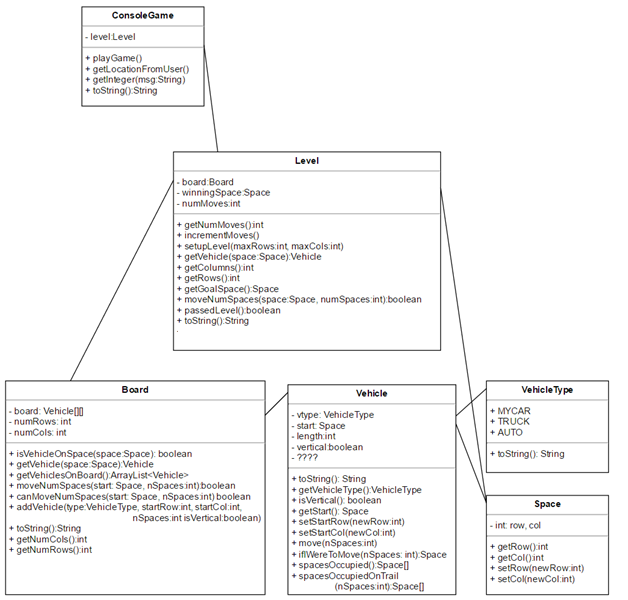
*Level*.java – Like we mentioned before, Level serves as the intermediary between Board and the Game. In a way this will be confusing to many of you because Level will not have much extra code, but helps become a layer between the ConsoleGame and Board. This will make it easy for us to switch between having a console program and a graphics program. Currently the program only has one level, so we will also manually setup the level by calling the appropriate methods to addVehicles to the board. Level has a board and a few extras like keeping track of the number of moves, setting up the level and knowing whether or not you have passed the level. Level also has other functionality, but the best way to write this functionality is by writing having it simply ask Board for its solution to the problem. For example, here’s how with the ConsoleGame class, when the program starts, the ConsoleGame, Board and Vehicle interact with each other…



### Step 3) Implement the ConsoleGame Class

*ConsoleGame*.java – This is the main driver for the program, but rather than have you waste time in the innards of Java’s text input capabilities, I’ve decided to write the text input mostly for you. Your job is to understand how to use the methods given to you, like GetInteger, and GetLocation, and to write the method playGame(), which should continually loop through asking for a location and an integer and potentially moving a vehicle until the user passes this level. Because of the size of this program, we are only going to concentrate on having it work for one level, so your playGame can simply print out a congratulatory message and finish. Later on, I’m going to ask you to write the file GraphicsGame, which will simply be the graphical version of this game. Starter code for that file will be given when the time comes for it. In order to make sure that you get a handle on everything that you need to write, I’ve given you a basic UML model below, which shows in more detail the methods you should write (including what to pass in and return), the instance variables you should store, and a little bit on how everything is related. It does not include how you should write your constructors.

Please Please PLEASE, start early, and if something does not make sense as you read the code or this handout, please don’t hesitate in asking me. This shouldn’t be a high pressure situation, it’s just meant to get you back in the swing of things from a programming perspective.



Woah! So this may look daunting, but again, if you do this piece by piece like I outlined above, it will make more sense. If you want more information, feel free to look at these help resources:

|  |  |  |
| --- | --- | --- |
| Traffic Jam Console Help | ["Narration - Board Help"](https://www.dropbox.com/s/3zlt5gvupjbmll2/Narration%20-%20Traffic%20Jam%20Console%20Game%20Help.pptx?dl=0) | Narrated powerpoint describing some of the issues you may encounter with board |
|  |
| Console Troubleshooting | ["SOS Website for Board"](http://appdevhelp.surge.sh/tjconsole) | Handy checklist to troubleshoot your consolegame |  |
|  |
| Traffic Jam Console Assignment | ["Assignment Description of Board (similar to Canvas)"](https://www.dropbox.com/s/trp7eqqah08111o/COMP%2055%20-%20Intro%20Project%20Console.docx?dl=0) | Most up-to-date version of this assignment that you are reading right now |  |