

## CSC461 More Java and Visitor pattern

**DUE: Thursday, September 27<sup>th</sup>, at 9AM**

***The purpose of this assignment is to give you practice with more Java and the visitor pattern.***

**+1 Bonus points for submitted 1 days early!**

**+2 more bonus points for submitted 2 days early!**

### Overview

You will be coding a city which has empty spaces, streets, greenspace, and buildings. To keep things simple, the objects all occupy one square (or tile) of the city, and the city is only a 5X5 grid. You will allow users to start with a blank city, or update to a default setup. Afterward, users can change individual tiles, count how many of each type of tile there are, “fix” the streets, and change the color of a type of object. The count, fix, and color operations must be done with the visitor pattern.

We will first create the class diagram in class, and you must use this diagram to code your project.

Java can use Unicode, and thus we can use the “box plot” characters, and other symbols in place of full images. The characters associated with each type of tile will be (COPY THESE):

◻ → empty area

◼ → building

⌘ → green space (sorry, everyone, IntelliJ only supports up to Unicode value 26B0, and trees are past that)

— → initial road

Java also supports ANSI color encoding for text, and there is a static class provided that will alter the color of text for you. For formatting, please see the “Making the output look better” section.

### Required functionality

The menu must be in the following format as the initial state:

```
.....
.....
.....
.....
.....
0) Make default City
1) Set Tile
2) Remove tile
3) Count Zones
4) Set tile color
5) Fix roads
6) Quit

Choice:
```

Initially, the city will be composed of all “empty” tiles signified with a ‘.’ character.

You may NOT assume correct input. The program should continue when given an “e”, 7, or even “pancake” by simply outputting “Invalid option” and then reprinting the city followed by the menu. HINT: you can move past the error by calling next() on the Scanner object.

### Set a tile

To change a tile, first ask the user to input the tile type, then ask for the location, and then reprint the city and menu. For example, the following will add a greenspace at index 1, 1:

```
Choice: 1
Input tile type 1) greenspace 2) building 3) road #) empty: 1
Input location (x y): 1 1
. . . . .
. * . . .
. . . . .
. . . . .
. . . . .
...menu...
```

If they do not give a legal tile type, default to an empty tile. If it is not a number, output “Invalid option” and continue.

### Remove a tile

To remove a tile, simply ask for the location and reset the tile to empty. For example, the following will remove the flower from the above “set a tile” example:

```
Choice: 2
Input location (x y): 1 1
. . . . .
. . . . .
. . . . .
. . . . .
. . . . .
```

If the input location is not valid, output “Invalid option.” Then reprint the city and menu.

### Default city

To aid in testing, you must provide a function that will alter the city so that after the command it appears as follows:

```
———**
———*
———△
. . . △
. . . *
```

Note, this will look a bit off due to the way Windows handles monospace fonts (aka, very poorly).

### Counting how many of each type

This must be done with a visitor.

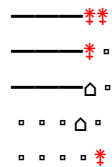
Count how many of each tile you have and then output the result. For example, the default city will print:

```
Choice: 3
Empty: 10
Greenspaces: 4
Roads: 9
Buildings: 2
Then reprint the city and menu.
```

## Changing the color

This must be done with a visitor. Ask for the tile type, and then the color (red, yellow, blue, green, or black). For example:

```
Choice: 4
Input tile type 1) greenspace 2) building 3) road #) empty: 1
Input color 1) red 2) yellow 3) blue 4) green #) black: : 1
```



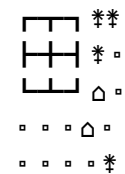
This effect should be permanent on all current tiles of that type. Adding a new tile will not be affected. For example, a new greenspace tile would be black. If they choose an invalid color, default to black. If they choose an invalid tile type, output “Invalid option.” Then reprint the city and menu.

## Fixing the roads

This must be done with a visitor. Moreover, use a nested visitor to make it work. The reason you need a nested visitor is that not only are you looking at all tiles, when you find a road tile, you must check all tiles adjacent to it.

When we add a road, we initially do not know its shape. We need to know the tile next to it to determine this. You are given a function called `setAdjacencies()` in the starting code. This function asks for whether there is a road on each side as true or false. It will figure out the correct symbol for you, but you will need to set the symbol afterwards.

After choosing the option, the default city should become:



If you have any additional loops to complete this task or have an “instanceof”, you are doing it wrong.

## Additional restrictions

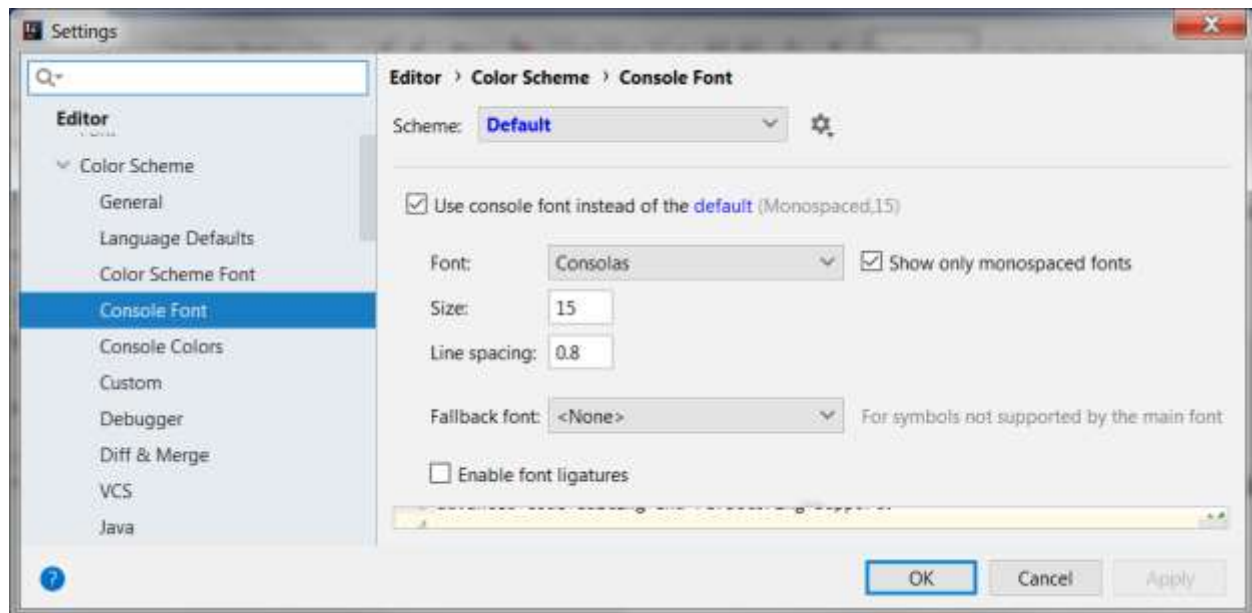
You may not make your member variable public unless they are constants.

The visitor pattern must be enforced. In other words, a derived class must be forced to create an `accept()` function or not compile.

## Making the output look better

You are given a `ColorText` class that will wrap the ANSI color tags around given text given a with a color. Just call `ColorText.colorString()` with your text and color and it will return a string that will be colored when output. There is a `Color` class in the Java “awt” library. Do NOT use this. The `ColorText` class’s enum will restrict to only the color it supports. The class also support bold and underlining if you are curious.

NOTE: Fonts depend on the OS, and even monospace fonts may not be monospace. The default font in IntelliJ does NOT display as monospace in Windows, and changing the console font in the settings to Consolas seemed to work the best, along with decreasing the distance between lines a bit. For example:



You are NOT required to change this.

## Grading Tiers

These tiers start with the simplest tasks, and go to the most involved.

- 1) Get the menu working
- 2) Have one square, with one tile
- 3) Set and unset that tile with each of the 4 classes
- 4) Make a blank city that is 5 by 5 and the square can be set and unset
- 5) Make the counting visitor
  - a. Not using a visitor is zero points
- 6) Make the color visitor
  - a. Not using a visitor is zero points
- 7) Make the road fix visitor
  - a. Not using both visitors is zero points

You must “reasonably” complete the lower tiers before gaining points for the later tiers. “Reasonably,” I can reasonable assume you honestly thought it was complete.

## Submission instructions ( correct submission [5 points] )

1. Check the coding conventions before submission.
2. If anything is NOT working from the following rubric, please put that in the bug section of your main file header.
3. If given a file that is not supposed to be changed, I will be overwriting the file with the original when I grade.
4. List any known bugs in the main file header comment.
5. All of your Java files must use a package named your lastName\_firstName (lowercase with no prefixes or suffixes)
6. Delete the out folders, then zip your **ENTIRE project** into **ONE** zip folder named your lastName\_firstName (lowercase with no prefixes or suffixes). Make sure this matches your package name!

**If you are on Linux, make sure you see “hidden folders” and you grab the “.idea” folder. That folder is what actually lists the files included in the project!**

7. Submit to D2L. The dropbox will be under the associated topic's content page.
8. *Check* that your submission uploaded properly. No receipt, no submission.

You may upload as many times as you please, but only the last submission will be graded and stored

If you have any trouble with D2L, please email me (with your submission if necessary).

## Rubric

All of the following will be graded all or nothing, unless indication by a multilevel score. You must reasonably complete the tier below before you can get points for the next tier.

**Missing files: -2 letter grades**

**Does not compile: -1 letter grade**

**Immediate crash: -1 letter grade**

**Sometimes crash (within bounds of assignment up to current tier): -half-letter grade**

These are on TOP of other deductions

Item	Points
Correct submission (-3 per error)	6
Coding standard (-3 per error)	8
Followed the class OOP diagram	8
Member variables follow access level rules	5
Framework for the visitor pattern is correct	7
Tier 1: menu working	4
Displays properly	2
Handles bad input	2
Tier 2: one tile working (2 for each type)	8
Tier 3: set and unset	10
Can set/unset empty	2
Can set/unset road	2
Can set/unset greenspace	2
Can set/unset building	2
Handles bad input	2
Tier 4: blank city	8
Displays a 5 by 5 empty city properly	2
Can set/unset empty at given location	1
Can set/unset road at given location	1
Can set/unset greenspace at given location	1
Can set/unset building at given location	1
Handles bad input	2
Tier X: default city displays properly	4
Tier 5: count tile types (2 for each tile count)	8
Tier 6: color tile types	10
Colors any tile once	2
Colors one tile permanently	2
Colors tile type once	2
Colors tile type permanently	2
Handles bad input	2
Tier 7: fix roads	14
Any road fixed	2
All corners work	2
All line roads work	2
Intersection works	2
Uses a second visitor, not anything else	6
Total	100