CSC461 More Java and Visitor pattern

DUE: as posted on D2L

The purpose of this assignment is to give you practice with more Java and the visitor pattern. It is also to give you practice using try—catches to validate input. This must run on IntelliJ and JDK 17.

There is a checklist!

Overview

You will be coding a city which has empty spaces, streets, greenspace and buildings. To keep things simple, each object occupies one square (or tile) of the city, and the city is a 7X7 grid. You will allow users to start with a blank city, or update to a default setup. Afterwards, users can change individual tiles, count the number of each type of tile, 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**. For this to work properly in Java, you may only call new Scanner(System.in) **once** in the entire program. A **public static** variable for your console Scanner is permitted. This supersedes the code standard requirements for this course. A starter file is given, and you may edit it except for the name of the class (this is what my testing script needs).

You will create a first attempt of a diagram and submit before the assigned class lecture. You will then be assigned a group during the designated lecture day to merge your designs. We will then create class diagrams using the group results. You **must** use this diagram to code your project.

Java can use Unicode, and thus we can use symbols in place of full images. The characters **must be** associated with each type of object (COPY THESE into your code):

▫ → empty area
⌂ → building
▒ → green space
~ → water
— → initial road

IntelliJ also supports ANSI color encoding for text, and there is a static class provided that will alter the text color for you. For formatting, please see the "Making the Output Look Better" section.

You will be given starter files that allow for tests to be run. Drop the contents into the same level as the *.iml* file. All of your Java files must use a package renamed to your lastName_firstName (lowercase with no prefixes or suffixes*). My script adds the package name based on your D2L recorded name, so do not use a nickname.*

NOTE: I will be doing my plagiarism check on this class AND the prior years that had something similar.

Required Functionality

The menu must be in the following format for the initial state (with a space after the :>):

```
//extra new line here!
◦◦◦◦◦◦◦
◦◦◦◦◦◦◦
◦◦◦◦◦◦◦
◦◦◦◦◦◦◦
◦◦◦◦◦◦◦
◦◦◦◦◦◦◦
◦◦◦◦◦◦◦

1) Set Tile
2) Make Default City
3) Count Zones
4) Set Tile Color
5) Rezone
6) Fix Roads
0) Quit

Choice:>
```

Initially, the city will be composed of all empty tiles signified with a '◦' character. Put an extra new line right before the grid.

When inputting values in the menu, do NOT assume correct input. The program should continue when given an "e," 7, or even "pancake." If it is an incorrect integer, output "`Number is out of range`." If it is the wrong data type, output "`Please input an integer`" and then immediately reprint the grid followed by the menu.

You must use the ColorText class (in this document see *Making the Output Look Better*) to color your grid (and only the grid) to black *on the initial run* to pass the first tier. The test checks for the color encoding.

HINT: Java's try-catch block will greatly simplify this logic and the remaining input error handling requirements. In fact, **you are permitted ONE try-catch block** to handle all of your exceptions for this assignment, *including the output response*. This means the grader will be checking if the error response is in only one location as it must be. Using scanner.hasNext() will make your code very error prone, and **is therefore forbidden**. I suggest using only nextInt() for this project. **Tip**: Remember how to throw your own exceptions.

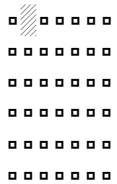
Reminder, redirected IO strips the new line from the user's <enters>. This limits the length of the line that has to be matched. RunTests ignores *new* line whitespace as much as possible.

Set an Area (Main Menu Option 1)

After the user selects option 1 to change an area's object, first ask the user to input the type, then ask for the location, and finally reprint the grid and menu. The upper left is (0, 0) and x is the column, and y is the row (e.g. math style) For example, the following will add a greenspace at index 1, 1:

```
Input tile type 1) greenspace 2) water 3) road 4) building 5) empty:> 1
Input location (x y): 1 1

◦◦◦◦◦◦◦
```


…menu…

Location (0, 0) is the upper left. If the input is not an integer, immediately output `"Please input an integer"` and reprint the grid and menu. If the type of object does not exist (for example, 100 or -1), immediately output "`Number is out of range`" and reprint the menu. If the input location is out of range, immediately output `"Number is out of range"` then reprint the grid and menu.

Default Grid (Main Menu Option 2)

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

Note, this may look a bit off due to the way Windows handles monospace fonts (aka, very poorly) it should look relatively OK in IntelliJ since I was careful with the character selection.

Counting the Amount of each Type (Main Menu Option 3)

Count how many of each object is present in the grid and then output the result. This must be done with a visitor. For example, the default grid from the previous section will print

```
Choice:> 3
Empty: 5
Buildings: 7
Greenspaces: 4
Roads: 27
Water: 6
```

and then reprint the grid and menu.

Tag, once, the call that starts the call chain for this visitor to do its task as "GRADING: COUNT."

Tip: If you have a loop per visitor in this program, you are doing it wrong!

Changing the Color (Main Menu Option 4)

This must be done with a visitor, and I have given a helper class to do this (see Section: Making the Output Look Better). The user can change the color of buildings, roads, and non-structure objects. Non-structure objects are greenspaces and water objects. Empty spaces are not recolored. The prompt to change the colors must ask for the type, then the color, and then reprints the menu and grid with the updated colors. For example:

```
Choice:> 4
Input tile type 1) building 2) road 3) non-structure:> 3
Input color 1) red 2) orange 3) blue 4) green 5) black:> 2
```

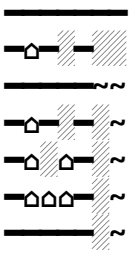
This effect should be permanent on all current *instances* of that type. Adding a new object will not be affected by the earlier color choice. For example, a new building object would be black. If the input is not an integer, immediately output `"Please input an integer"` and reprint the grid and menu. If the type of object does not exist (for example, 100 or -1), immediately output "`Number is out of range`" and reprint the menu.

Tag, once, the call that starts the call chain for this visitor to do its task as "GRADING: COLOR."

Rezoning (Main Menu Option 5)]

If there are fewer than 5 empty spaces, convert these spaces to greenspaces. Running this on the default city would result in no changes to the city but must print "Insufficient open areas" and then reprint the menu. This print statement must be in your try-catch block, and reached by throwing an exception. The exception is up to you.

If the (6, 1) spot is changed to a greenspace the result would be:



Hint: reuse the count visitor's results.

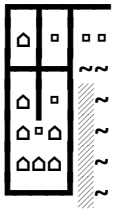
Tag, once, the call that starts the call chain for this visitor to do its task as "GRADING: REZONE."

Fixing the Roads (Main Menu Option 5)

This must be done with a visitor. Moreover, you must call another visitor inside the acceptX function(s) to make this work. The reason for the second visitor is that not only are you looking at all tiles when finding a road tile, but all adjacent tiles must also be checked.

When adding 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 of the tile 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, and you will not receive the visitor points. For example, if you stored the roads in a list and not a 2D array, the following should occur **once**:

```
for( Area a : myAreas)
    a.accept(someVisitor);
```

If you have this multiple times, you have broken O in SOLID as any new visitor would require reopening your grid class.

Hint: you will need some way to get and find areas at a particular (x, y) location. Passing in your grid instance is good start or even storing the (x, y) location in the area. You can then ask the grid for more information about a cell. Note, this is *not* the same as passing the variable that holds the areas.

Tag the *nested* visitor (nested, as in, the location a visitor is using another visitor), once, with "GRADING: NESTED." This most likely will be in your "fix roads" visitor class

If the roads look "broken" vertically, you can change the line height in the edit to remove the excess spacing. This is under File → Settings → Editor → Font. Find the line height option.

Additional Restrictions

- Objects must own the characters that describe how to display them.
- You must COPY the grading tags exactly into the inline comments.
- You must follow the OOP diagram developed in class. Minor adjustments are expected and mostly likely needed. Moderate changes may be permitted with written permission.
- Visitor requirements:
    - You must have an accept (or equivalent) function for the collection.
    - You may **not** add a function per visitor in your collection class or menu class. There should be a single function that accepts a *generic* visitor! Multiple *generic* visitors are ok. Generic means that the function takes in *any* derived visitor plus some extra parameters. For example,
      `acceptRowVisitor(visitor : Visitor, rowIndex : int)`
      is generic, but
      `acceptRowVisitor(mySpecialRowVisitor : MySpecialRowVisitor)`
      is not.

    - You may not have a Visitor member variable in your collection. This breaks O in SOLID. Make your visitor outside the collection class.
    - The visitor pattern must be enforced. In other words, a derived class must be forced to create an accept() function or not compile.
- NO public, protected, or package-private variables are permitted unless they meet the "rules."
- Direct access to the underlying collection is forbidden due to good encapsulation rules.
- You may not use scanner.hasNext(), or similar.
- **You are permitted ONE try-catch block** to handle all of your exceptions this assignment, *including the output response*.
- You may use the online "book" for code snippets, *if, and only if,* you cite it on the function that uses it. Any other source must get approval from me, and must have a license that would permit use in a project.

Tests

You are given a file called RunTests. This must be just inside your package folder. This uses the given redirected IO files, and checks your output against my personal run's output. If there is a mismatch, it will output the results, and flag the problem lines in red. The text files being checked against must be in the same folder as src.

This RunTests file should be selected as your starting file. If you want to turn off the tests temporarily, run directly from your starter file.

If you want to see the output of your file without the tests but still have the redirected IO, change the starting file to CityStart. Then, go to the configuration menu for CityStart. Select the redirect IO box, and find the desired file.

Note: To match the tests, you must use the Black colored text starting in the FIRST tier.

Making the Output Look Better

You are given a ColorText class that will wrap the ANSI color tags around giving the text a color. Simply call ColorText.colorString() with your text and color and the call 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 colors it supports. The class also supports bold and underlining, if you are curious.

Grading Tiers

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

1) Get the menu working, and show a blank city
2) Set and unset the 0, 0 tile with each of the tile types
3) Set and unset any tile with each of the tile types
4) Get the bad input handled at this point
5) Make the default city
6) Make the counting visitor
7) Make the color visitor (visitor needs parameters)

8) Make the rezone visitor (visitor needs more challenging parameters and reused results from a visitor)
    a. Also checks exception skill
9) Make the road fix visitor (visitor needs parameters and another visitor)

You must "reasonably" complete the lower tiers before gaining points for the later tiers. By "reasonably," I can reasonably assume you honestly thought it was complete.

Submission instructions

1. Check the coding conventions before submission.
2. Check that you are not in violation of the additional deductions in the main tab.
3. If anything is NOT working from the following rubric, please put that in the bug section of your main file header.
4. If given a file that is not supposed to be changed, I will be overwriting the file with the original when I grade.
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! This must be able to be unzipped with a single command.

    If you are on Linux, make sure you see the "hidden folders" and that 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.

Handling the bad input lines will be 0 if you used more than one try-catch.

Item	Points
Other deductions	
Framework for the visitor pattern is correct (-5 for each violation listed in additional restrictions)*	20
Tier 1: menu working	8
Displays a 7 by 7 empty city properly	4
Displays rest properly	4
Tier 2: set and unset at 0, 0	15
Can set/unset each (equally weighted)	15
Tier 3: set and unset anywhere	10
Can set/unset each (equally weighted)	10
Tier 4: error checking up to this point	12
Handles bad input (-4 per error)	12
Tier 5: default city displays properly	10
All space correct (-4 for minor error)	8
City still update correctly	2
Tier 6: count tile types	17
Count correct: (types equally weighted)	15
Formatting correct	2
Tier 7: color tile types	20
Any color effect	5
Colors correct tile group	5
Colors correct color	5
Handles bad input	5
Tier 8: rezone	14
Nothing happens if <=5 empty	3
Something happens if >5 empty	3
Correctly updates	4
Exception catches thrown	4
Tier 9: fix roads	24
Any road fixed	4
All work (-4 per error)	12
Uses a second visitor, not anything else*	8
Total	150

* these have required tagging in the comments. They also do not affect passing the tier.