CSC461 More Java and Visitor pattern (150pt)

DUE: Thursday, October 1th, at 7AM

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 11.

There is a checklist this time!

Overview

You will be coding a fish feeding application, which has empty spaces, fish food, habitat, and fish. To keep things simple, each object occupies one square area of the grid, and the grid is of 5X5 size. You will allow users to start with a blank grid, or to update to a default setup. Afterwards, users can change individual areas, count the number of each type of object, feed the fish, and change the color of a type of object. The count, feed, 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 normal code standards 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
∝ →fish
☁ → habitat
✻ → food

Java 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.

All of your Java files must use a package named 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:

```
//extra new line here!
⛶⛶⛶⛶⛶
⛶⛶⛶⛶⛶
⛶⛶⛶⛶⛶
⛶⛶⛶⛶⛶
⛶⛶⛶⛶⛶

1) Set Area
2) Remove Area
3) Make Default Grid
4) Count Types
5) Color Fish
6) Feed Fish
0) Quit

Choice:
```

Initially, the grid will be composed of all "empty" objects signified with a '⛶' character. Put an extra new line right before the grid.

You may NOT assume correct input in the menu. The program should continue when given an "e," 7, or even "pancake." If it is an integer, output "Unknown menu option." 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 (see Section: Making the output look better) to color your grid to black *immediately* to pass the first tier. The test check 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*. Using scanner.hasNext() will make your code very error prone, and **is therefore forbidden**.

Set an Area

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. For example, the following will add a habitat at index 1, 1:

```
Input area type 1) habitat 2) fish 3) food #) empty: 1
Input location (x y): 1 1
⛶⛶⛶⛶⛶
⛶☁⛶⛶⛶
⛶⛶⛶⛶⛶
⛶⛶⛶⛶⛶
⛶⛶⛶⛶⛶
…menu…
```

Location (0, 0) is the upper left. If the user gives a number (#), but not a legal object type, default to an empty object. If the input is not a number, output "`Please input an integer`" and immediately reprint the grid and menu. If the input location is not valid, output `"Please input a number between 0 and 5".` Then reprint the grid and menu.

Default grid

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

```
✻✻✻⛶☁
✻∝✻⛶⛶
✻✻✻✻✻
⛶⛶✻∝∝
☁∝⛶⛶☁
```

Note, this will 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 how many of each type

Count how many of each object you have and then output the result. This must be done with a visitor. For example, the default grid above will print

```
Empty: 7
Habitats: 3
Food: 11
Fish: 4
```

Then reprint the grid and menu.

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

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

Changing the Color

This must be done with a visitor, and I have given a helper class to do this (see Section: Making the output look better). Count up the number of fish, food, and habitats. If the number of fish are less than food or habitats, color the fish green. If the number of fish are more than food or habitats, color the fish red. If the number of fish do not meet either of the prior rules, color the fish yellow. For example,

```
✻✻✻⛶☁
✻∝✻⛶⛶
✻✻✻✻✻
⛶⛶✻∝∝
☁∝⛶⛶☁
```

The fish turn yellow since there are more fish than habitats, but more food than fish.

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 habitat object would be black. If they choose any other number, default to black. If they do not input an integer, output `"Invalid option."` Then reprint the grid and menu.

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

Tip: reuse the results from the count visitor!

Feed the fish

This must be done with a visitor. Moreover, you must call another visitor inside the acceptX function(s) to make it work. The reason you need a second visitor is that not only are you looking at all areas when finding a fish, you must check all areas adjacent to it. When fish feed, they eat the food around them. If there is no food, they die. We need to know the objects in the areas next to it to determine this. Walk through the fish, left to right and top to bottom. When eating, fish eat one food. Check the food starting at the top, and search clockwise (top, right, bottom, left).

After choosing the option, the original default grid should become:

```
✻⛶✻⛶☁
✻∝✻⛶⛶
✻✻✻⛶⛶
⛶⛶✻∝∝
☁⛶⛶⛶☁
```

Feed again, and you should have this:

```
✻⛶✻⛶☁
✻∝⛶⛶⛶
✻✻✻⛶⛶
⛶⛶⛶∝⛶
☁⛶⛶⛶☁
```

Feed again and you should have this:

```
✻⛶✻⛶☁
✻∝⛶⛶⛶
✻⛶✻⛶⛶
⛶⛶⛶⛶⛶
☁⛶⛶⛶☁
```

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 them in a list, and not a 2D array, we should see the following **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 square 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 “feed fish” visitor class.

Additional restrictions

- Object must own the characters that describe how to display them.
- You must COPY the grading tags exactly into the inline comments, otherwise they will not be counted.
- You must follow the OOP diagram developed in class. Minor adjustments are expected and 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 use a loop to start up each visitor that works on the entire collection. If you have multiple loops, you have repetition code smell. This is part of the visitor pattern framework.
    - Specifically, 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!
    - Specifically, 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 unless they meet the “rules”.
- Good encapsulation states you should NOT allow direct access to the underlying collection. You may not make the square area collection available through an access level or a getter. A getter to access a specific area object is permitted, and likely will be necessary.
- 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*.
- Reminder: You may only call `new Scanner(System.in)` **once** in the entire program to allow the grading script to work. The starter file breaks coding conventions a bit to permit this.
- 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 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.

This RunTests should be selected as your starting file. If you want to turn off the tests temporarily, set the RUN_TESTS constant to false.

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

Note: to match the test, 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 given text given 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 color 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 grid
2) Set and unset the 0, 0 square area with each of the 4 classes
3) Make a blank grid that is 5 by 5, where the squares can be set
4) Make the default grid
5) Make the counting visitor (Not using a visitor is zero points, and the tier fails)
6) Make the color visitor (Not using a visitor is zero points, and the tier fails)
7) Make the fish visitor (Not using both visitors is zero points for that line item)

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.

For partial credit in the last unpassed tier, you must explain in the main file header, how to confirm what sublines are working. No comment, no partial credit.

Submission instructions

1. Check the coding conventions, redownload test files, and rerun tests before submission.
2. Check that you are not in violation of the additional deductions in the main tab.
3. Complete the checklist and past into the top of SolarStart.
4. All of your Java files must use a top level package named 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.
5. Delete the “out” folder(s), 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!

6. Submit to D2L. The dropbox will be under the associated topic's content page.
7. *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 individual lines 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.

Item	Points
Other deductions	
Framework for the visitor pattern is correct\restrictions met (-6 for each violation listed in additional restrictions)*	24
Tier 1: menu working	12
Displays a 5 by 5 empty grid properly	6
Displays the rest properly	6
Tier 2: set and unset at 0, 0	16
Can set empty	4
Can set food	4
Can set habitat	4
Can set fish	4
Tier 3: handles bad input at this point	15
Handles bad input at the menu level	3
Handles bad input at the object level	3
Handles bad input at location level	3
Handles bad input with 1 try-catch	6
Tier 4: set and unset at x, y	12
Can set empty at given location	3
Can set food at given location	3
Can set habitat at given location	3
Can set fish at given location	3
Tier 5: default grid displays properly (-5 for a minor error)	15
Tier 6: count types (5 for each object count)*	20
Tier 7: color fish	18
Colors any fish, any color	5
Colors all fish, any color in one command	5
Colors all fish correctly (-50% if one of the rules are violated) *	8
Tier 8: Feed Fish	18
Any fish\food changed	2
Fish lives\dies correctly	2
Any food removed next to a fish	2
Correct food removed for each fish	4
Uses a second visitor, not anything else*	8
Total	150

* These have required tagging in the comments.