CISC 3120 Fall 2012 Homework 4

Instructions

- This assignment **is due by midnight on Wednesday October 3rd** and you will submit this online — instructions below.
- **Follow these submission instructions:**
  1. Create a ZIP archive from the `.java` files that make up your homework.
  2. Go to:

     `http://agents.sci.brooklyn.cuny.edu/parsons/cisc3120/`
  3. Login and submit your ZIP archive.
  4. You can submit as many times as you like before the deadline—only the final version of the homework will be graded.
  5. Failure to follow these instructions will result in points being taken away from your grade. The number of points will be in proportion to the extent to which you did not follow instructions. . . (since it can make it a lot harder for me to grade your work — grrrr!)

1 Source

You should start from the full version of the Rabbit/Fox example that we looked at in class on Thursday.

You can download this from the website, from the page for Unit II. The file you want is the one called:

```
EcoSystem.zip
```

You are **strongly advised** to read through **all** these instructions before starting work on the homework.

2 First build your project

Once again you you need to use your Eclispe skills to set up a project (previous homeworks describe how to do this in detail).

1. Once youhave set up a project, put all the files from `EcoSystem.zip` in the `src` directory.
2. See what this code does — run the project.
3. The orange dot represents the `Fox`, that has an empty `move` method right now.
4. The black and/or white dots (colors are set randomly) represent two `Rabbits`. These curently move in a fixed direction.
5. Right now, pressing the button not only calls the `move` method for all the animals, but also calls the `eat` method (from the `Predator` interface for one `Rabbit`

3 Clean up `Animal`

1. The current version of `Animal` makes `myColor` `protected` so that subclasses can access it.
2. This is bad programming style.
3. Improve it by making `myColor` `private`
4. This will involve writing an API for `myColor` and retrofitting it into the code, so that `Animal` and `Fox` use the API

4 Modify the simulation

1. Right now the `EcoSystem` class creates three named animals and uses these names to make method calls that make the simulation work.
2. Change `EcoSystem` so that it creates 8 `Rabbits` and 2 `Foxes` without names, holds these in an array of `Animals`, and uses this array to make the necessary method calls.

5 Introduce `Grass`

1. Now let's add someting for `Rabbits` to eat.
2. Create a class `Grass` which contains a `Point` object called `location` that (as with `Rabbit` and `Fox`) gives its location in the simulation and on the screen.

   As with `Rabbit` and `Fox`, the location of `Grass` should be set randomly.
3. Naturally, since it is a plant, `Grass` does not move.
4. `Grass` should be represented as a green dot in the display.
5. Alter the `eat` method for `Rabbit` so that it reports `Yum!` only when there is `Grass` within 5 units of the `Rabbit`.
6. Add ten `Grass` objects to the `EcoSystem`.

6 `Plant` and `Living`

1. If you just implement `Grass` as described above, you will have to duplicate a lot of functionality from `Animal`.
2. A better solution (and why I told you to read through the instructions before doing anything) is to create a superclass that holds this common information.
3. Do this by creating an `abstract` class `Living` which holds `location`.
4. Make `Animal` a subclass of `Living`.

   Leave the `myColor` attribute in `Animal` along with the `abstract` `move` function.
5. Create another `abstract` class `Plant` which has a field to hold the color of the `Plant`. This field should, of course, be `private` so it will need access functions.
6. Make `Grass` a subclass of `Plant`
7. Add another subclass of `Plant`, `Flower`, which shows up as a red or yellow dot (randomly selected).
8. Add six `Flowers` to the `EcoSystem`.

7 `Predator` and `Prey`

1. Make `Fox` implement the `Predator` interface, and have its `eat` method print out a suitable message when it gets within 5 units of a `Rabbit`.
2. Now create a `Prey` interface which includes a function `beEaten`.
3. Make `Rabbit` and `Grass` implement the `Prey` interface — for now `Rabbit`'s `beEaten` should just print out:

   > Drat!

   and make `Grass`'s `beEaten` print out a message of your choice.
4. Ensure that when a predator `eat`s some object, that object runs its `beEaten` method.

   Hint: This will likely need you to change `eat` so that the predator has access to a reference to the object it is eating — then it can call `beEaten` on that object.

8 `Motion`

1. The current `move` methods are a bit dull.
2. Change `Rabbit` so that it chanegs direction randomly.

   That doesn't mean it has to change direction all the time, just that it changes direction now and again (which you can determine randomly), and picks a random `direction` when it chnages.
3. Alter `Fox` so that `move` does something.

9 Death

1. Right now objects keep doing things after they have been eaten.
2. Add a `private boolean` field `alive` to the class `Living`.

   Again this will need an API.
3. This should be set true when an object of class `Living` is created.
4. It should also be set false when a an object executes `beEaten`.
5. Alter the display method so that only objects that are `alive` are displayed.

   Note that you don't need to delete the objects that are dead.

10 Challenge problems

You will get additional credit for *any* of the following things that you do. Make sure that you document which of these features you have added.

1. Replace the `Move things!` button with `Start` and `Stop` buttons. When the user presses the `Start` button, the animals start moving, and when the user presses the `Stop` button they stop.
2. At the moment the messages from `eat` and `beEaten` appear on the console. Have them appear in a dialog box instead. Preferably have all the messages appear, one on each line, in a scrollable window.

3. Allow the number of `Rabbits`, `Foxes`, `Grass` and `Flower` to be set by the user. You choose how to do this, but describe how to do it somewhere in your documentation.
4. Alter the `move` method for `Fox` so that it pursues `Rabbits`. How you do this is up to you, but, for example, you might write a `sense` function for `Fox` which detects the `direction` of any `Rabbit` that is within 20 units of the `Fox` so that the `Fox` can move towards it.

   With `direction` as currently defined, the direction that the `Fox` moves will only be approximately towards the `Rabbit`, so you may want to refine `direction` also.
5. Have dead objects show up in their last location for a short while after they are eaten, in a suitable color to denote that they are dead.

11 How I will mark your work

1. You will get credit for each of the things listed above that you have implemented in your program and which work as I have described them.
2. You will also get credit for the **way** in which you do this. That is programs that do not adhere to what I consider to be good programming style will not gain as much credit.

   You can get an idea of what I consider to be good style by looking at the code I have given you.
3. However, your program **must compile and run** for you to get any credit at all.

   (At this point in your computer science career it is unacceptable to submit programs that don't compile and run).

   Bear in mind that I will be using Java version 1.6 to assess whether your code compiles and runs.
4. Thus it is better for you to submit a program that works and contains less functionality than it is to submit a program with more functionality that does not work.
5. You can get full credit for this homework **without** solving the challenge parts.