Homework Assignment #7

CS5004 – Object-Oriented Design
Northeastern University – Silicon Valley
Summer 2020

Due Sunday 06/28 at 11:00pm PT

Grading: Each programming problem is graded as follows

- A submission which does not compile gets 0.
- A submission which compiles but does something completely irrelevant gets 0.
- A submission which works (partially) correctly, gets (up to) %80 of the total credit.
- %20 is reserved for the coding style. Follow the coding style described in the book.

Problem 1 [80pts]. The goal for this project is to create a simple 2D predator–prey simulation. In this simulation, the prey is ants, and the predators are doodlebugs. These critters (i.e. both species) live in a world composed of a 20×20 grid of cells. Only one critter may occupy a cell at a time. The grid is enclosed, so a critter is not allowed to move off the edges of the grid. Time is simulated in time steps. Each critter performs some action every time step.
The ants behave according to the following model

- Move. Every time step, randomly try to move up, down, left, or right. If the cell in the selected direction is occupied or would move the ant off the grid, then the ant stays in the current cell.
- Breed. If an ant survives for three time steps, then at the end of the third time step (i.e., after moving), the ant will breed. This is simulated by creating a new ant in an adjacent (up, down, left, or right) cell that is empty. If there is no empty cell available, no breeding occurs. Once an offspring is produced, the ant cannot produce an offspring until three more time steps have elapsed. The maximum number of spawned ants is 10 per ant.

The doodlebugs behave according to the following model:

- Move. Every time step, if there is an adjacent cell (up, down, left, or right) occupied by an ant, then the doodlebug will move to that cell and eat the ant. Otherwise, the doodlebug moves according to the same rules as the ant. Note that a doodlebug cannot eat other doodlebugs.
- Breed. If a doodlebug survives for eight time steps, then at the end of the time step, it will spawn off a new doodlebug in the same manner as the ant. The maximum number of spawned doodlebugs is 10 per doodlebug.
- Starve. If a doodlebug has not eaten an ant within the last three time steps, then at the end of the third time step, it will starve and die. The doodlebug should then be removed from the grid of cells.

During one turn, all the doodlebugs should move before the ants. Write a program to implement this simulation and draw the world using ASCII characters of "o" for an ant and "X" for a doodlebug. Below is an example of an 8 × 8 grid with 3 ants and 5 doodlebugs.

```
 - - - - - - - - -
|X           o  |
|               |
|X          o   |
|  o            |
|               |
|  X            |
|      X        |
|            X  |
 - - - - - - - - -
```

Create a class named `Organism` that encapsulates basic data common to both ants and doodlebugs. This class should have an abstract method named `move` that is defined (overridden) in the derived classes `Ant` and `Doodlebug`. You probably need additional data structures to keep track of which critters have moved. Initialize the world with 5 doodlebugs and 100 ants (at random). After each time step, prompt the user to press Enter to move to the next time step. You should see a cyclical pattern between the population of predators and prey, although random perturbations may lead to the elimination of one or both species. Note that the simulation end only if the user enters END instead of just pressing enter.
Note 1: This programming assignment requires using random number generators in three places.

1. The "move" step where a critter must choose one of the four (or less depending on the availability) directions.
2. The breeding step where a critter must choose a neighboring cell (if available) to breed in.
3. Three is the initial placement of critters.

To generate random numbers between 0 and $n - 1$ in Java, use the following code.

```
import java.util.Random;
Random rand = new Random();
int next = rand.nextInt(n);
```

Try to think of a clever way to place critters randomly on the grid.
Note 2: One approach to implementing this project is to have an array of 105 `Organisms` where the first 5 are `Doodlebugs`. Each round of the simulation corresponds to an iteration over the array where each `Organism` performs its `move()`. You must somehow keep track of starved critters. Finally, you can track spawned critters by having as an instance variable an array (of size 10) of spawned `Organisms` in `Organism` (in this case, `move()` should recursively call the `move()` method of its spawned critters). Again, you should "know" whether the spawned critters are alive or not.
Note 3: Another approach which is perhaps more natural, is the have a 2 dimensional array of `Organism`. This way, you don't have to keep track of each creature. But you need to be able know which critters are bugs and which ones are ants so that you can move bugs first.
Submission Format: You are are required to submit 4 files:

- `Organism.java` which contains the class `Organism`.
- `Ant.java` which contains the class `Ant`.
- `Doodlebug.java` which contains the class `Doodlebug`.
- `Simulate.java` which contains the class `Simulate` and has a `main()`.

The first 3 files must be part of a package named `Critters`. The other file, `Simulate.java`, imports `Critters` package and its `main()` contains the logic for reading from input and controlling the simulation. Note that the files that are part of the same package must reside in a folder named after the package. However, you only need to submit the `.java` files; we will take care of the folder structure.