Homework Assignment #6

CS5004 – Object-Oriented Design
Northeastern University – Silicon Valley
Summer 2019

Due Sunday 07/07 at 11:00pm PDT

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.

Formatting: Each class must reside in its own file. If you need to instantiate objects of class `A` in your class `B`, your `B.java` file must import `A.java` first. The names you choose for your classes should match the ones specified in each problem. Also, for each problem i, you must have a `Problem_i.java` file containing (only) the class which has the `main` method. Finally, you must put all files related to problem i into a folder named `problem_i`. The files *must be placed in the right sub-folder*. Make sure that your code complies. See the note in Problem 1 for an example.
Submission: For each problem i submit one `problem_i.zip` file to Blackboard. The zip file should maintain the structure of the sub-folders. See the note in Problem 1 for an example.

Problem 1 [70pts]. The goal for this programming project is to create a simple 2D predator–prey simulation. In this simulation, the prey is ants, and the predators are doodlebugs. These critters 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.

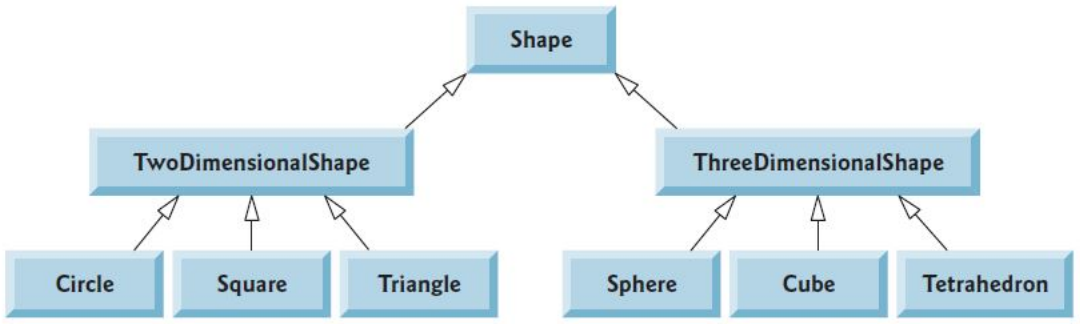
- Breed. If an ant survives for three time steps, then at the end of the third time step (i.e., after moving), the ant will breed. This is simulated by creating a new ant in an adjacent (up, down, left, or right) cell that is empty. If there is no empty cell available, no breeding occurs. Once an offspring is produced, the ant cannot produce an offspring until three more time steps have elapsed.

The 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.
- 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. Create a class named `Organism` that encapsulates basic data common to both ants and doodlebugs. This class should have an overridden method named `move` that is defined in the derived classes of `Ant` and `Doodlebug`. You may need additional data structures to keep track of which critters have moved. Initialize the world with 5 doodlebugs and 100 ants. 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.

Problem 2 [80pts]. Implement the `Shape` hierarchy shown in figure below. Note that each `TwoDimensionalShape` should contain method `getArea` to calculate the area of the two-dimensional shape. Each `ThreeDimensionalShape` should have methods `getArea` and `getVolume` to calculate the *surface area* and volume, respectively, of the three-dimensional shape. Create a program that uses an array of `Shape` references to objects of each concrete class in the hierarchy. The program should print a text description of the object to which each array element refers. Also, in the loop that processes all the shapes in the array, determine whether each shape is a `TwoDimensionalShape` or a `ThreeDimensionalShape`. If it's a `TwoDimensionalShape`, display its area. If it's a `ThreeDimensionalShape`, display its area and volume.


Shape
TwoDimensionalShape
ThreeDimensionalShape
Circle
Square
Triangle
Sphere
Cube
Tetrahedron