**Programming Project 7**

Due: Monday, 4/25/16 at 11:59 pm

**Doodlebugs and Ants**

For this programming project, write a Java program that will simulate the population growth algorithm between two groups of insects: doodlebugs and ants.  This program idea is described in the textbook Absolute Java by Walter Savage and has many items related to the GridWorld projects from the AP CS Exam and to John Conway's Game of Life.

This project will use the java GridDisplay class to display each generation which can be found in the file GD3.java. We will use one color to show the positions in the grid that contains a doodlebug, another color to show the positions in the grid that contains an ant and a third color to show the empty positions in the grid.   You may use any three colors that you like.

The simulation is a predator/prey scenario.  Such simulations are fairly common and many variations of such simulations exist.  The primary differences between these simulations are the rules by which the predator/prey entities live.  Here, the doodlebugs will be the predator creature and the ants will prey creature.  As the simulation runs, you will see the doodlebug population and the ant population grow and shrink in a cyclic manner; however, the built-in randomness may cause one or both creatures to die out completely.

In this simulation, the time will advance in "days".  The simulation will occur in a grid of 20x20 locations.  We will randomly place down 5 doodlebugs and 100 ants into the grid to start the start of the simulation.  Only one creature can exist in a location at any given time.  So if the original intended location of a doodlebug or ant is already occupied, you will need to find a different location.  At the end of each "day", your program is to update an instance of the GridDisplay class showing the current locations of all doodlebugs, all ant and all empty positions.  You are to use the mySleep( ) method to allow the changes per day to be noticeable by the user.

**During each day of the simulation, the doodlebugs must all move first, then the ants will move after the doodlebugs.**  You will need some mechanism to keep track of which creatures have moved or have not moved during a day.  A set or list type data structure would be a good way to keep track of this information.

When a **doodlebug** moves on a given day it will do all of the following operations in the order given:
1.      **hunt** – if there is an ant at an adjacent location (up, down, left or right), the doodblebug will move to that location and eat the ant.  Otherwise **move** - (when it does not eat an ant), the doodlebug will pick a random direction (up, down, left or right) and attempts to move to the adjacent location in that direction.  If that location is already occupied by another doodlebug or that location does not exist (because the doodlebug is at the edge of the grid), the doodlebug will not move to a different location but stay in its current location.
2.      **spawn** – if the doodlebug has survived for 8 days, it will attempt to create a new doodlebug.  The new doodlebug will "be born" in an empty location adjacent (up, down, left or right) to the parent doodlebug.  If there is no empty adjacent location, no new doodlebug will be born.  Once a doodlebug has produced an offspring, it cannot produce another offspring until 8 more days has elapsed.

3.      **starve** – if the doodlebug has not eaten an ant during the last 3 days, it will starve and die.  When a doodlebug dies, the grid location it is on becomes empty.

When an **ant** moves on a given day it will do all of the following operations in the order given:
1.      **move** – the ants will pick a random direction (up, down, left or right) and attempts to move to the adjacent location in that direction.  If that location is already occupied by another doodlebug or another ant or that location does not exist (because the ant is at the edge of the grid), the ant will not move to a different location but stay in its current location.
2.      **spawn** – if the ant has survived for 3 days, it will create a new ant.  This new ant will "be born" in an empty location adjacent (up, down, left or right) to the parent ant.  If there is no empty adjacent location, no new ant will be born.  Once an ant has produced an offspring, it cannot produce another offspring until 3 more days has elapsed.

Note that in one day a doodlebug could move to an adjacent location, spawn a new doodlebug, and die of starvation.  Also note that if a doodlebug cannot spawn on the $8^{th}$ day because there is no empty adjacent location, it will try to spawn on the $9^{th}$ day.  Also, note that ants only "die" when eaten, they never starve.

## Object Development

You MUST write the code for the doodlebugs in separate class called Doodlebug that exists in a file called Doodlebug.java.

You MUST write the code for the ants in separate class called Ant that exists in a file called Ant.java.

These two classes share some common algorithms like basic moving and spawning.  You are to use inheritance when writing your program so you don't need to re-write the code common to both classes twice and to specify the instance variables that are also common to both.  The base class that contains that common information is to be called Creature and is to exist in a file called Creature.java.  Failure to use inheritance will result in at least a 10 point penalty.

## Notes on the Grid

At any point in time during the simulation, each location in the grid MUST be in one of three states:
   1. Empty – Containing no creature
   2. D State – Containing a Doodlebug
   3. A State – Containing an Ant

When referring to the adjacent locations, those locations will be the ones immediately above, below, left, or right of the current position. (Diagonal locations are NOT adjacent.)  So for position x,y, its 4 adjacent locations are at:

   ▪ x-1, y                              ▪ x+1, y
   ▪ x, y-1                              ▪ x,  y+1

## GridDisplay API

**Note that the GridDisplay class is to only be used the output for your program**. You may not modify the GridDisplay class to add any get operations or to read in information from the class.

You will find a class GridDisplay in the file GD3.java. This class will allow you to display a 2-D grid of any size. You can also place any character at any grid position and color any grid position. The GridDisplay API (Application Program Interface) is as follows:

- public GridDisplay(int rows, int cols)

  This constructor will take two integers which will set up the rows and columns for the grid. Each grid position will initially display the space character and will be WHITE.

- public void setChar (int row, int col, char c)

  This method will display the character given in the third parameter as the grid position specified by the row and column parameter values given. This methods is not being used for this project (so you can ignore it).

- public void setColor (int row, int col, Color c)

  This method will display the color given in the third parameter as the grid position specified by the row and column parameter values given.

The file GridDisplay class also contains the following static method which will cause the code to sleep for the number of milliseconds specified by the parameter. This method can allow the changes made to the GridDisplay grid to appear as animation. This is needed because the screen gets updated too fast that no one can see which changes occurred when. Call the mySleep( ) method after one day has been displayed and before the next day is determined.

- public static void mySleep( int milliseconds)

See the sample code in GD3.java for examples on how to use the GridDisplay class.

## Internal Storage Structure

You are not allowed to use any of the classes from the Java Collections Framework in this program. These classes include ArrayList, Vector, List, Set, HashMap, etc. **If you need such a class, you are to write it yourself.** A full listing of the Java Collections Framework can be found at:

   http://docs.oracle.com/javase/7/docs/technotes/guides/collections/reference.html

You must have some way to keep track of each location in the 20x20 grid. While you are not required to use a 2-D array to keep track of each location, this is a good starting point for this project. A 2-D array in Java is created by creating an array of arrays. This is described in:

   http://www.willamette.edu/~gorr/classes/cs231/lectures/chapter9/arrays2d.htm

There are many other places to find out about 2-D arrays (however, the Head First Java text did decide to only briefly discuss them).

## Command Line Arguments

The command line may contain the –d flag which will allow the user to specify the delay value between days that will be given to the mySleep( ) method.  Recall that mySleep( ) takes an integer value that specifies how many milliseconds the program is to pause.  When the –d flag is not given, you can hard code a reasonable value to be used for the delay time.

The use of the –d flag will require the use of two command line arguments.  The first command line argument will be the –d flag while the second command line argument will be this integer value.  Recall all command line arguments are actually strings, so the second command line argument will need to be parsed from a string to an integer.

Valid command lines for this program would be:
- java ptroy1Proj7
- java ptroy1Proj7 –d 100
- java ptroy1Proj7 –d 750

Invalid command lines for this program would be:
- java ptroy1Proj7 –d hello
- java ptroy1Proj7 500 –d


## Program Submission

You are to submit the programs for this lab via the Assignments Page in Blackboard.  You should zip all of your .java files together for a single submission to Blackboard.  To help the TA, name the file that contains the main( ) method used as the starting point of your project with your net-id and the assignment name, like:

- ptroy1Proj7.java

This way, to compile your project, we will just need to unzip your submission and type:

- javac ptroy1Proj7.java

Then to run your program, we will just need to type:

- java ptroy1Proj7