COMS 228, Fall 2020

Project 1: Profit Utilization for Internet Service

Due: at 11:59 PM
Tuesday, September 8

Problem Description:

You were hired by a fictional internet provider called Blaze to compute yearly profits. Customers pay fixed monthly fees. Blaze makes a profit when customers do not use all the bandwidth they are allowed. Such users are called “Casual’ users. The region to which Blaze supplies internet is in the form of an *n* x *m* grid. Blaze can provide internet to each cell in the grid if desired.

Each cell in the grid has either customers (one of three types as given below) or is in “Empty” or “Outage” states. Therefore, a cell can have one of five states:

C: Casual customer: checks their email 13 times a day
S: Streamer customer: an aspiring e-celebrity with an appetite for bandwidth
R: Reseller customer: provides lower-cost, lower-bandwidth internet to users off the grid
E: Empty grid cell
O: Outage grid cell

For example, below is a 4 x 4 grid:

```
O R O R
E E C O
E S O S
E O R R
```

Row and column indices start from 0. The top left cell is at coordinates (0,0). Coordinates are in the form (RowIndex,ColumnIndex). In the example above, the cell at coordinates (1,1) is E (Empty). It has a 3 x 3 neighborhood centered at the square:

```
O R O
E E C
E S O
```

Please note that the cell itself is not counted as its own neighbor, so it can have a maximum of 8 neighbors.

Cells which are on the edge of the grid will have smaller neighborhoods. In the above example, the cell at the upper left corner, (0, 0), has 3 neighbors in a 2 x 2 neighborhood:

```
O R
E E
```

Similarly, the cell at (3, 2), Reseller, has a 2 x 3 neighborhood which is:

```
S O S
O R R
```

In the example neighborhood, a cell with a 3 x 2 neighborhood is (1,0):

```
0 R
E E
E S
```

2. Cell Update Rules:

Each month, the state of the cell can change depending on its current state and the state of its neighbors. The rules for those are:

1. **C**: Casual: If the current cell is occupied by a casual user and
    a. If there is any reseller in its neighborhood, then the reseller causes outage in the casual user cell. Thus, the state of the cell changes from C (Casual) to O (Outage).
    b. Otherwise, if there is any neighbor who is a streamer, then the casual user also becomes a streamer, in the hopes of making it big on the internet.
2. **S**: Streamer:
    a. If there is any reseller in the neighborhood, the reseller causes outage for the streamer as well.
    b. Otherwise, if there is any Outage in the neighborhood, then the streamer leaves and the cell becomes Empty.
3. **R**: Reseller
    a. If there are 3 or fewer casual users in the neighborhood, then Reseller finds it unprofitable to maintain the business and leaves, making the cell Empty.
    b. Also, if there are 3 or more empty cells in the neighborhood, then the Reseller leaves, making the cell Empty. Resellers do not like unpopulated regions.
4. **O**: Outage: An Outage cell becomes an Empty cell, meaning internet access is restored on the billing cycle after an outage.
5. **E**: Empty: If the cell is empty, then a Casual user takes it and it becomes a C.
6. Additional rules:
    a. Any cell that (1) is not a Reseller or Outage and (2) has at most one empty neighbor OR one outage neighbor converts to Reseller.
    b. If none of the above rules apply, any cell with 5 or more casual neighbors becomes a Streamer.
7. If none of the rules apply, then the cell state remains unchanged for the next iteration.

All cells update concurrently: each new state of a cell is based on the previous state of its neighbors.

3. Determining Profit:

Blaze makes a profit of $1 from casual customers. Blaze does not make a profit from streamers or resellers.

Profit utilization is calculated as a percentage of potential profit for each billing cycle. For example If the region grid is 12x10, then potential profit is $120 per month, as each cell is occupied by a casual user. Thus, in a single month, profit utilization is 100*C/120, where C is the number of Casual cells. The software for this project needs to compute the profit utilization over 12 billing cycles (equivalent to one year), as a percentage of maximum profit, e.g. 23.

4. Implementation Details:

1. Implement the abstract class `TownCell` to represent a generic cell. It has five subclasses, one for each cell type, specifically: `Casual`, `Streamer`, `Reseller`, `Empty` and `Outage`. These 5 subclasses need to be implemented as well.

2. The `Town` class has a public member variable named `grid`, which is a 2D array of `TownCell`. It stores the state of each cell and holds the objects of subclasses of `TownCell`.

3. `State` is an enum for the cell's state. You may not modify this file.

4. Implement an `ISPBusiness` class, which simulates the changes in the grid for an entire year (12 monthly cycles). Note that all its functions are static. This class should:

4.1. Create the `Town` object and populate the grid inside it.

4.2. The `main` method needs to ask the user about grid generation:

"`How to populate grid (type 1 or 2): 1: from a file. 2: randomly with seed`"

4.2.1. If the user chooses Option 1, take a file path as input.

"`Please enter file path:`"

The format of the file is given in ISP4x4.txt. The first line contains the number of rows and columns, separated by space.

After the first line, the file contains lines equal to the number of rows, and each line contains as many characters as there are columns. Each character can be: C, S, R, E, or O (upper case only).

The file will not deviate from this format.

4.2.2. To generate the grid randomly, ask the user for three integer values for number of rows, number of columns and seed for random number generator (in order).

"`Provide rows, cols and seed integer separated by spaces:` "

Populate the grid with an array of size: rows x cols. Then assign each cell in it using random number generator with the given seed.

Java provides a random number generator. To use it, you need to import the package java.util.Random. Next, declare and initiate a Random object like below:

```
Random generator = new Random();
```

Then, random numbers can be obtained by:

```
int newRandomValue = generator.nextInt(5);
```

The above instruction will generate a random number between 0 and 4 that corresponds to one of the cell type as given by static variables in the `TownCell` class.

- Templates are provided for all classes, except subclasses. You should create 5 subclasses extending `TownCell`. Names of those 5 classes should be exactly: Casual, Streamer, Reseller, Empty and Outage, including case. Also use the package name exactly as: edu.iastate.cs228.hw1.

- Below is state of the grid as it changes for 12 billing cycles. This is just for illustration - you should not output the grid state but just the profit percentage. That is only output 38 for this case.

```
How to populate grid (type 1 or 2): 1: from a file. 2: randomly with seed

2

Provide rows, cols and seed integer separated by space:
4 4 10
Start:

O    R    O    R
E    E    C    O
E    S    O    S
E    O    R    R
 Profit:1
```

After itr: 1

E E E E
C C O E
C O E O
C E E E
Profit: 4
After itr: 2
R C C C
C C E C
C E C E
C C C C
Profit: 12
After itr: 3

E R R R
R O C C
R R S R
R R C R
Profit: 3
After itr: 4

R E E E
E E R R
E E R E
E E R E
Profit: 0
After itr: 5

E C C R
C C E E
C C E R
C C E R
Profit: 8
After itr: 6

R C O E
R S C C
R S C E
R C R E
Profit: 5
After itr: 7

```
E    R    E    R
E    R    S    C
E    R    O    R
E    R    E    R
Profit: 1
 After itr: 8

R    E    R    E
C    E    O    O
C    E    E    E
R    E    R    E
Profit: 2
 After itr: 9

E    C    E    C
O    C    E    E
O    C    C    C
E    C    E    C
Profit: 8
 After itr: 10

R    C    C    C
E    C    C    C
E    C    S    C
R    C    R    R
Profit: 9
 After itr: 11

E    R    R    R
R    O    R    R
R    O    R    R
E    R    E    E
Profit: 0
27.604166666666668    //only output a double as the profit %!!
```

- The only value returned by your software is profit utilization in %, as a `double`.
- While you may add more classes, variables and helper functions, please do not modify access specifiers or function signature for any of the given functions. Failure to abide by this rule is considered breach of contract in the real world and carries a 50% penalty. Any helper functions introduced need to be properly commented.

- You need not worry about validating the values of rows and columns input by the user. We are assuming that user is only providing positive integers for these.

5. Junit Classes:

JUnit classes include `CasualTest, StreamerTest, ResellerTest, EmptyTest, OutageTest, ISPBusinessTest, TownTestand TownCellTest`. You also need to create all these classes.

There should be at least one test method for each function for the respective classes.

You should use jUnit5.

6. Submission

Write your classes and JUnit test classes in the `edu.iastate.cs228.hw1` package. Also submit the text files with your JUnit tests. Your text files should be directly inside the project folder (i.e., along with src folder). Turn in the zip file, **not your class files**. Please follow the guideline posted under Documents & Links on Canvas.

Include the Javadoc tag `@author` in each class source file. Your zip file should be named `Firstname_Lastname_HW1.zip`.