**Review -FRQ**

**FRQ1- Control flow**

SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Assume that the classes listed in the Java Quick Reference have been imported where appropriate.

Unless otherwise noted in the question, assume that parameters in method calls are not null and that methods are called only when their preconditions are satisfied.

In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

A positive integer is called a “self-divisor” if every decimal digit of the number is a divisor of the number, that is, the number is evenly divisible by each and every one of its digits. For example, the number 128 is a self- divisor because it is evenly divisible by 1, 2, and 8. However, 26 is not a self-divisor because it is not evenly divisible by the digit 6. Note that 0 is not considered to be a divisor of any number, so any number containing a 0 digit is NOT a self-divisor. There are infinitely many self-divisors.

The following table contains a sample code execution and the corresponding results.

|  |  |
| --- | --- |
| isSelfDivisor(128) | true |
| isSelfDivisor(26) | false |
| isSelfDivisor(120) | false |
| isSelfDivisor(1068) | false |

Finish writing method isSelfDivisor below, which takes a positive integer as its parameter. This method returns true if the number is a self-divisor; otherwise, it returns false. The main method includes tests to check if this method is working correctly.

**(Hints)**

You need to loop through all the digits in the number one at a time and test if the current digit is 0 and if so return false. Otherwise you need to test if the passed number is evenly divisible (0 remainder) by the current digit. If it isn’t we return false. If you have looped through all the digits and not found a problem return true.

How to loop through all the digits in a number? You can use x % 10 to get the rightmost digit from a number and x / 10 to remove the rightmost digit from a number.

public class SelfDivisor{

/\*\* @param number the number to be tested

\* Precondition: number > 0

\* @return true if every decimal digit of

\* number is a divisor of number;

\* false otherwise

\*/

public static boolean isSelfDivisor(int number)

{

// put your answer on the answer sheet

}

**FRQ2--writting classes**

**Past exam paper-StepTracer**

This question involves the implementation of a fitness tracking system that is represented by the StepTracker class. A StepTracker object is created with a parameter that defines the minimum number of steps that must be taken for a day to be considered active. The StepTracker class provides a constructor and the following methods.

addDailySteps, which accumulates information about steps, in readings taken once per day

activeDays, which returns the number of active days

averageSteps, which returns the average number of steps per day, calculated by dividing the total number of steps taken by the number of days tracked

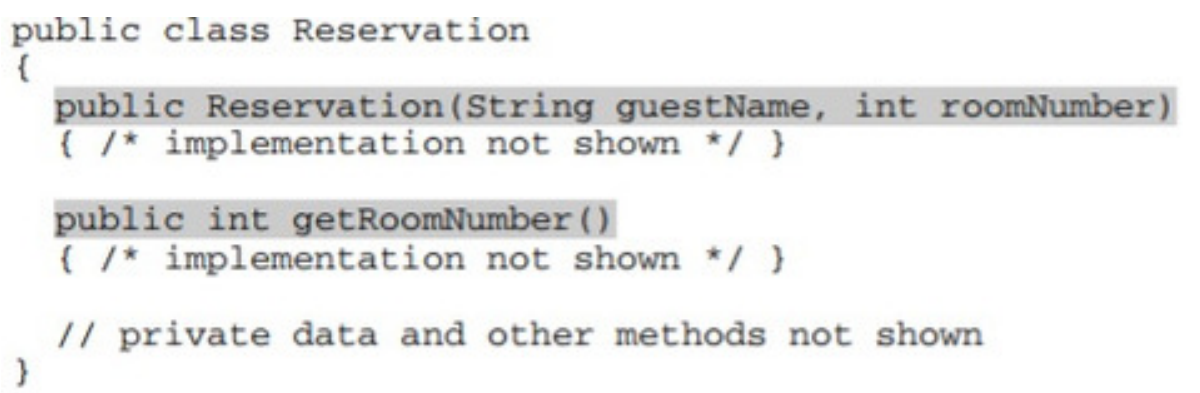
The following table contains a sample code execution sequence and the corresponding results.

|  |  |  |
| --- | --- | --- |
| **Statements and Expressions** | **Value Returned (blank if no value)** | **Comment** |
| StepTracker tr = new StepTracker(10000); |  | Days with at least 10,000 steps are considered active. Assume that the parameter is positive. |
| tr.activeDays(); | 0 | No data have been recorded yet. |
| tr.averageSteps(); | 0.0 | When no step data have been recorded, the averageSteps method returns 0.0. |
| tr.addDailySteps(9000); |  | This is too few steps for the day to be considered active. |
| tr.addDailySteps(5000); |  | This is too few steps for the day to be considered active. |
| tr.activeDays(); | 0 | No day had at least 10,000 steps. |
| tr.averageSteps(); | 7000.0 | The average number of steps per day is (14000 / 2). |
| tr.addDailySteps(13000); |  | This represents an active day. |
| tr.activeDays(); | 1 | Of the three days for which step data were entered, one day had at least 10,000 steps. |
| tr.averageSteps(); | 9000.0 | The average number of steps per day is (27000 / 3). |
| tr.addDailySteps(23000); |  | This represents an active day. |
| tr.addDailySteps(1111); |  | This is too few steps for the day to be considered active. |
| tr.activeDays(); | 2 | Of the five days for which step data were entered, two days had at least 10,000 steps. |
| tr.averageSteps(); | 10222.2 | The average number of steps per day is (51111 / 5). |

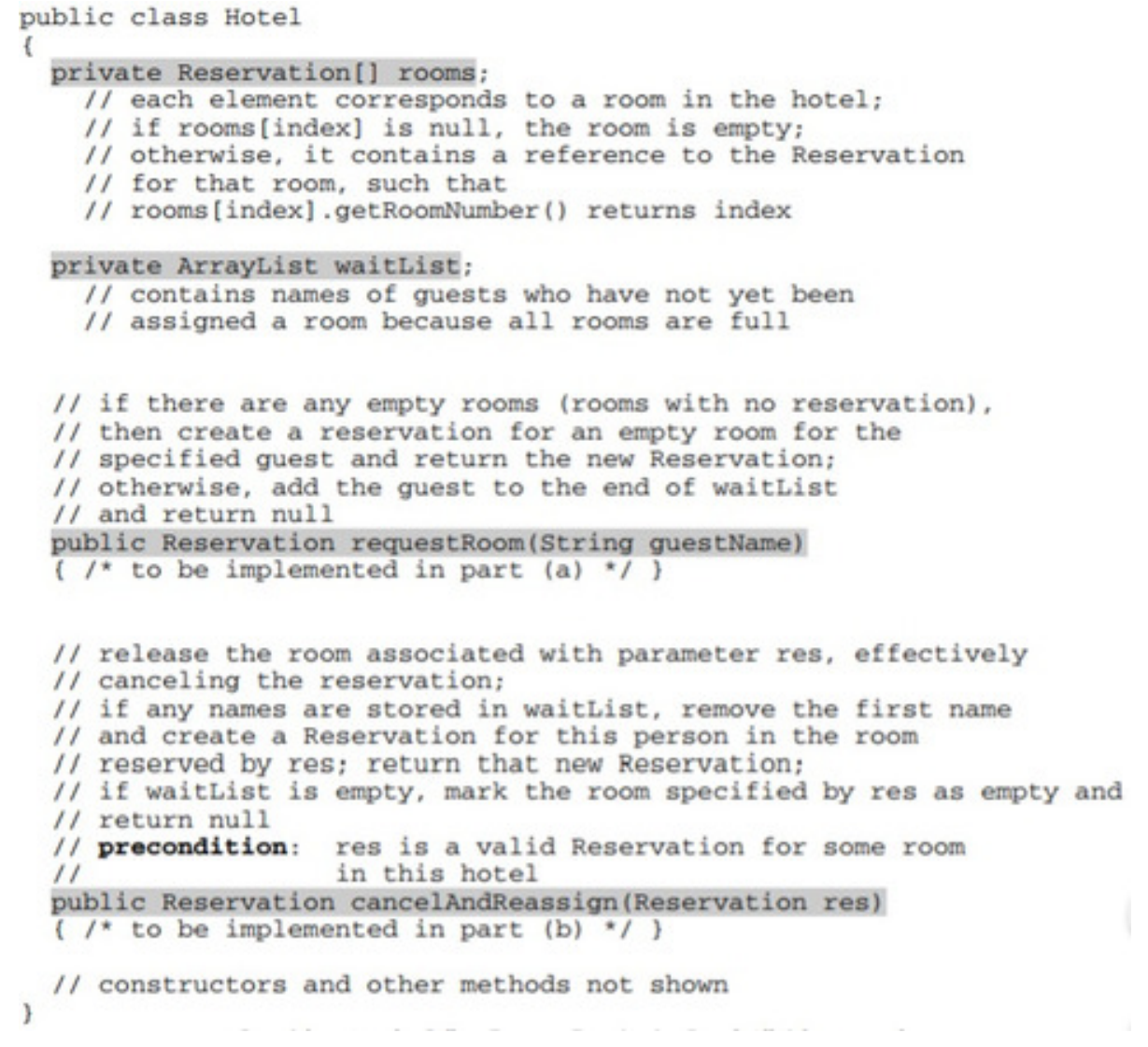
This question asks you to write the complete StepTracker class, including the constructor and any required instance variables and methods. Your implementation must meet all specifications and conform to the example.

**FRQ3--Array/ArrayList**

In this question, you will implement two methods for a class Hotel that is part of a hotel reservation system. The Hotel class uses the Reservation class shown below. A Reservation is for the person and room number specified when the Reservation is constructed.

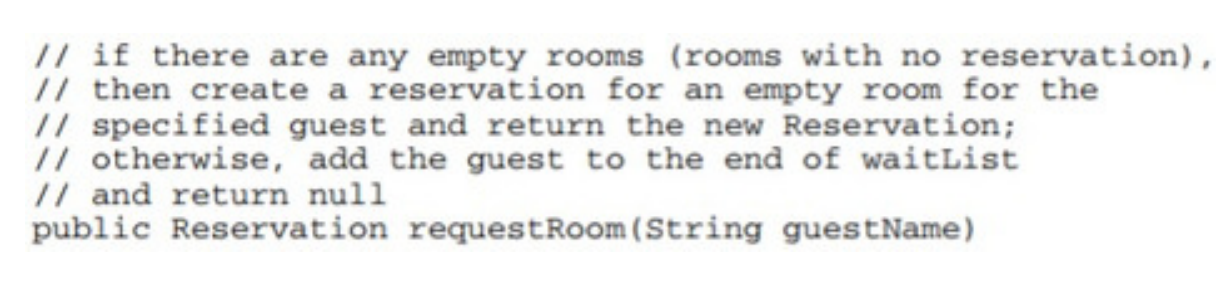


An incomplete declaration for the Hotel class is shown below. Each hotel in the hotel reservation system has rooms numbered 0, 1, 2, . . . , up to the last room number in the hotel. For example, a hotel with 100 rooms would have rooms numbered 0, 1, 2, . . . , 99.



a. Write the Hotel method requestRoom. Method requestRoom attempts to reserve a room in the hotel for a given guest. If there are any empty rooms in the hotel, one of them will be assigned to the named guest and the newly created reservation is returned. If there are no empty rooms, the guest is added to the end of the waiting list and null is returned.

Complete method requestRoom below.



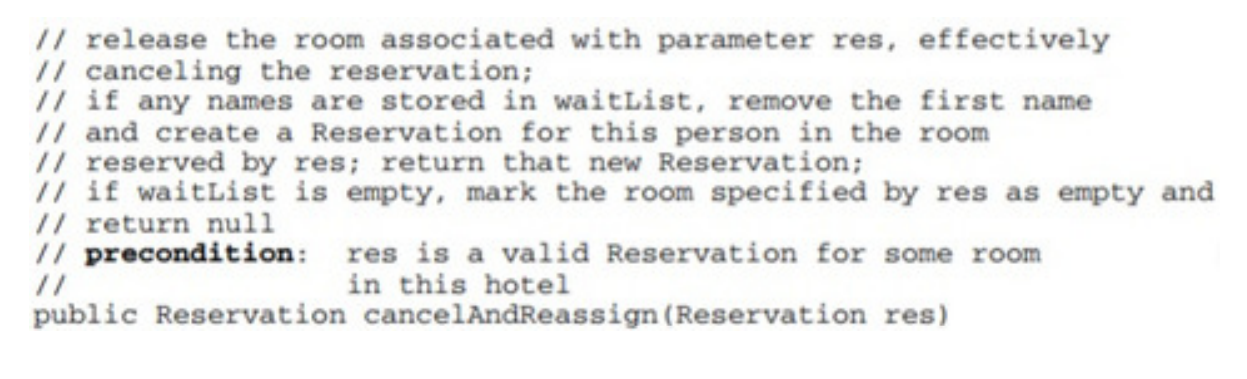
b. Write the Hotel method cancelAndReassign. Method cancelAndReassign releases a previous reservation.

If the waiting list for the hotel contains any names, the vacated room is reassigned to the first person at the beginning of the list. That person is then removed from the waiting list and the newly created reservation is returned. If no one is waiting, the room is marked as empty and null is returned.

In writing cancelAndReassign you may call any accessible methods in the Reservation and Hotel classes.

Assume that these methods work as specified.

Complete method cancelAndReassign below.



**FRQ4--2D Array**

The ExperimentalFarm class represents crops grown on an experimental farm. An experimental farm is a

rectangular tract of land that is divided into a grid of equal-sized plots. Each plot in the grid contains one type of crop.

The crop yield of each plot is measured in bushels per acre.

A farm plot is represented by the Plot class. A partial definition of the Plot class is shown below.

public class Plot

{

private String cropType;

private int cropYield;

public Plot(String crop, int yield)

{

/\* implementation not shown \*/

}

public String getCropType()

{

return cropType;

}

public int getCropYield()

{

return cropYield;

}

}

The grid of equal-sized plots is represented by a two-dimensional array of Plot objects named farmPlots, declared

in the ExperimentalFarm class. A partial definition of the ExperimentalFarm class is shown below.

public class ExperimentalFarm

{

private Plot[][] farmPlots;

public ExperimentalFarm(Plot[][] p)

{

/\* implementation not shown \*/

}

/\*\* Returns the plot with the highest yield for a given crop type,

as described in part (a). \*/

public Plot getHighestYield(String c)

{

/\* to be implemented in part (a) \*/

}

/\*\* Returns true if all plots in a given column in the two

dimensional array farmPlots

\* contain the same type of crop, or false otherwise, as described

in part (b).

\*/

public boolean sameCrop(int col)

{

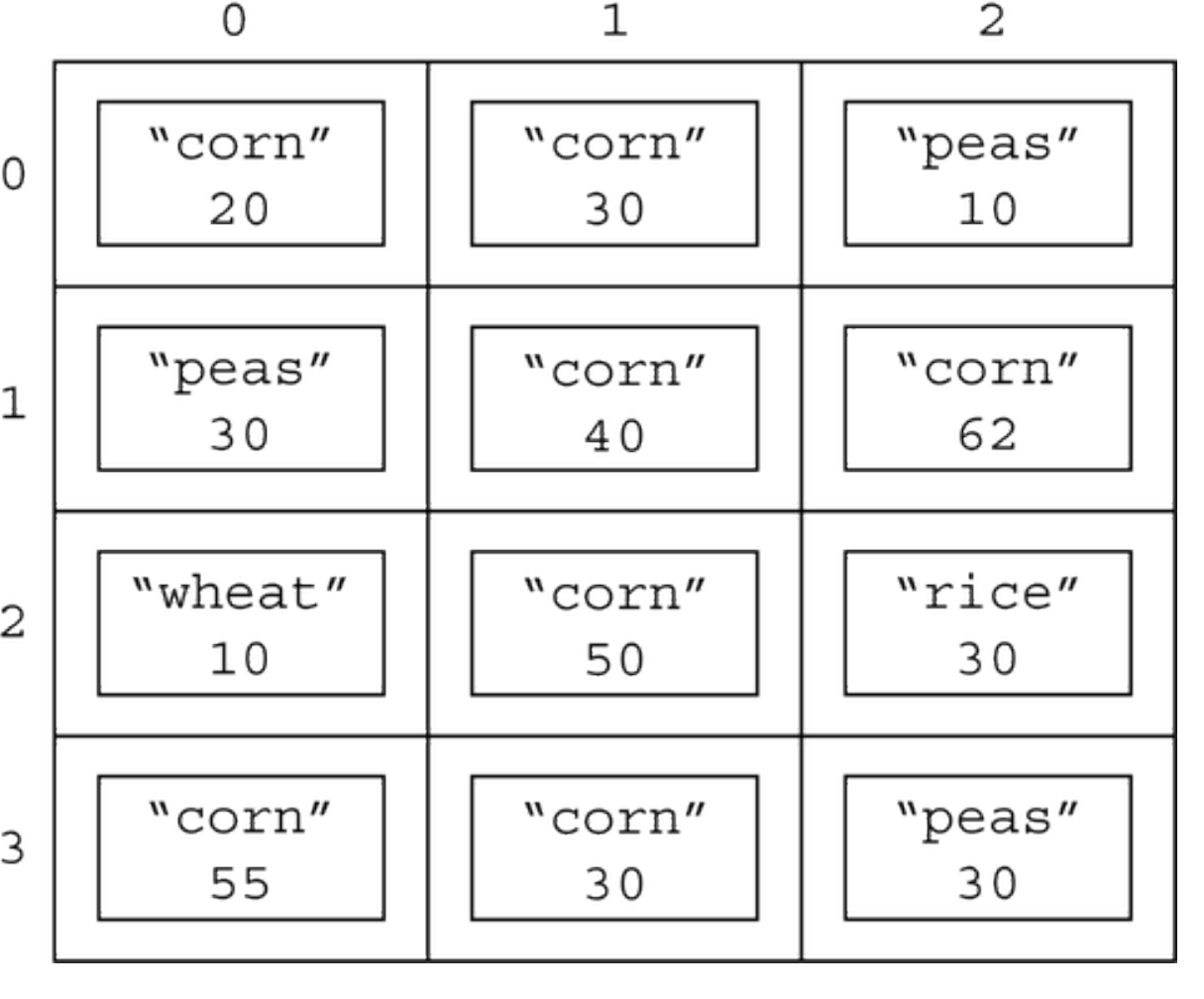
/\* to be implemented in part (b) \*/

}

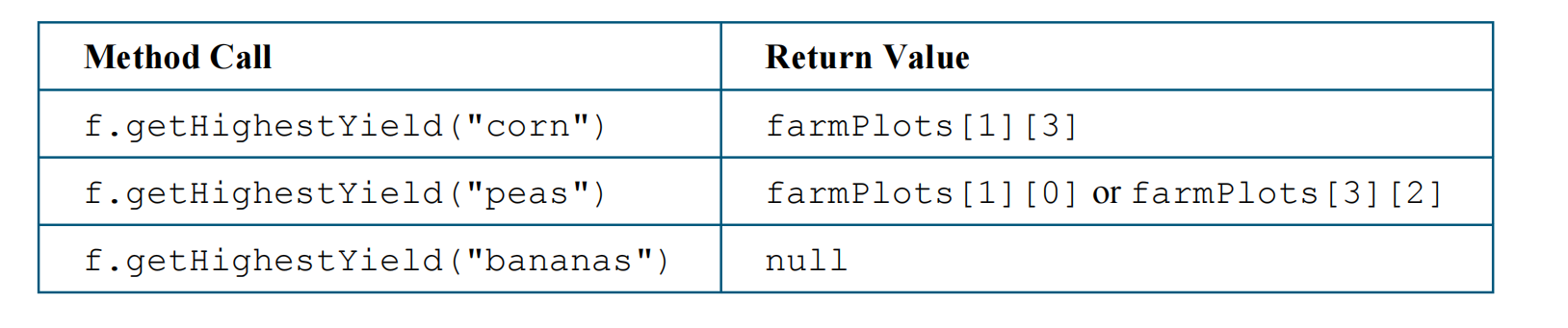
}

(a) Write the getHighestYield method, which returns the Plot object with the highest yield among the plots in farmPlots with the crop type specified by the parameter c. If more than one plot has the highest yield, any of these plots may be returned. If no plot exists containing the specified type of crop, the method returns null.

Assume that the ExperimentalFarm object f has been created such that its farmPlots array contains the following cropType and cropYield values.



The following are some examples of the behavior of the getHighestYield method.



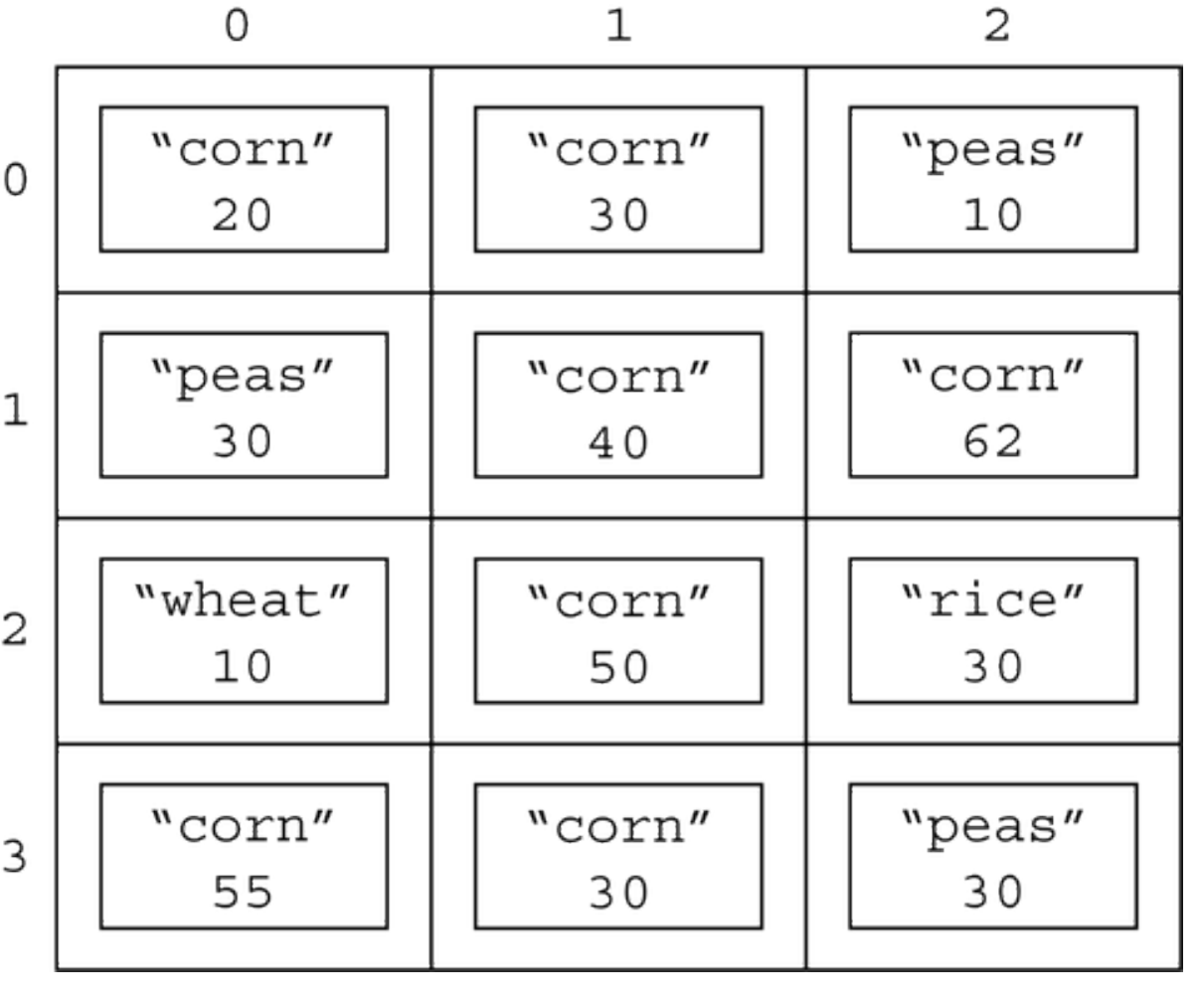
Write the getHighestYield method below.

/\*\* Returns the plot with the highest yield for a given crop type, as described in part (a). \*/

public Plot getHighestYield(String c)

(b) Write the sameCrop method, which returns true if all the plots in a given column of farmPlots grow the same crop and returns false otherwise.

Assume that the ExperimentalFarm object f has been created such that its farmPlots array contains the following cropType and cropYield values.



The following are two examples of the behavior of sameCrop.

The method call f.sameCrop(0) returns false because the values of cropType for the elements of column 0 ("corn", "peas", "wheat", and "corn") are not all the same.

The method call f.sameCrop(1) returns true because the values of cropType for all elements of column 1 are the same ("corn").

Write the sameCrop method below.

/\*\* Returns true if all plots in a given column in the two-dimensional array farmPlots

\* contain the same type of crop, or false otherwise, as described in part (b).

\*/

public boolean sameCrop(int col)