Module 5 – Assignment A – SeatingPlan

Seating Plan Specification

Create an application that models the seats in a classroom. Each seat holds a String that will either say "EMPTY" or have the student's first name, such as "John". The user can create a classroom, add students to seats, and print out a visual representation of the seating plan.

You will need to create two classes – one called SeatingPlan with a 2D array to represent the seats, and client code called myClass that fills the array

Application output should be similar to (but need not be identical to):

```
How many rows of seating does your class have? 10
How many columns of seating does your class have? 10
Who do you want to add to the seating plan? Q to quit John
Which row do you want John to sit in? 1
Which column do you want John to sit in? 1
Who do you want to add to the seating plan? Q to quit Charlie
Which row do you want Charlie to sit in? 0
Which column do you want Charlie to sit in? 0
Who do you want to add to the seating plan? Q to quit Katie
Which row do you want Katie to sit in? 2
Which column do you want Katie to sit in? 2
Who do you want to add to the seating plan? Q to quit Shannon
Which row do you want Shannon to sit in? 5
Which column do you want Shannon to sit in? 5
Who do you want to add to the seating plan? Q to quit Q
```

Note that the 1st row is actually row 0. Most users will not expect this, so an extra mark will be provided for allowing the user to refer to row 0 as row 1

```
Here is your class seating plan!:
[Charlie][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY]
[EMPTY][John][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY]
[EMPTY][EMPTY][Katie][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY]
[EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY]
[EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY]
[EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][Shannon][EMPTY][EMPTY][EMPTY][EMPTY]
[EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY]
[EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY]
[EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY][EMPTY]
```

Pseudocode for Client Code provided:

```
myClass
main():
  SeatingPlan period1 = new SeatingPlan();
  period1.addStudent("Charlie");
  period1.addStudent("Katie");
  period1.displayClass;
```

Instructions

a) Write out your code design for the SeatingPlan classe (refer to the Tic-Tac-Toe sample program for an example). Refer to the provided client pseudocode for hints regarding parameters and function. Copy and paste your Code Design to the Google Doc Submission Document

SeatingPlan

variables:

methods:

constructor -

b) Create the two classes

c) Test your program. Write down how the application was tested. Add this to the Google Submission Doc.

d) BONUS: In order to achieve a mark of level 4 on this project, you must improve upon the basics of this program in a significant way by correctly implementing some different concepts. Here are some ideas for extras:

- prompt the user appropriately if the seat is already occupied
- print out a single row, or single column of students
- count the number of students in the seating plan
- check if any rows or columns are empty
- search the array for a particular student

e) Reminders: Project must be submitted to the dropbox, and your project folder name must be "Last Name, First Name – Mod 5 AsnA – SeatingPlan".

Module 5 – Assignment A – SeatingPlan Rubric

Categories	Level 1 (50 – 59%)	Level 2 (60 – 69%)	Level 3 (70 – 79%)	Level 4 (80 – 100%)
Code Design (weight: 2)	Variable names and method names along with a description of methods and any required parameters are missing crucial elements and do not communicate a clear plan	Variable names and method names along with a description of methods and any required parameters is communicated with limited clarity	Variable names and method names along with a description of methods and any required parameters is communicated with only minor omissions or errors	Variable names and method names along with a description of methods and any required parameters is communicated clearly
Program Testing (weight: 1)	Program testing was missing crucial elements.	Program testing was insufficient to conclude that the program runs properly	Program testing missed covered a considerable number of possible cases and was summarized in a logical way	Program testing was thorough, and summarized in a logical and succinct way
Program Submission (weight: 1)	Program was not submitted properly	Serious or multiple omissions with assignment submission (such as incorrectly placed in dropbox, incorrect naming convention, late)	Small omission with assignment submission (such as incorrectly placed in dropbox, incorrect naming convention, late)	Program was submitted to the dropbox correctly, using proper naming conventions, on time
Program Execution (weight: 2)	Program doesn't run properly, or the output has serious errors.	Program runs properly. Output has errors. The provided client code will not run properly, but student provided client does run properly	Program runs properly. Output is correct with only minor errors. The provided client code will run properly	Program runs properly as is. The output is identical or superior to the sample provided. The provided client code will run properly.
Source code (weight: 4)	Coding conventions are missing or incomplete.	Coding conventions are followed with some attention to detail	Coding conventions are followed with considerable attention to detail	Coding conventions are followed with thorough attention to detail
Programming Concepts (weight: 5) Focus: 2dArray Look for extras	Few of the programming concepts from the unit are used properly	Some of the programming concepts from the unit are used properly	Most of the programming concepts from the unit are used properly	Many or all of the programming concepts from the unit are used properly to maximize the efficiency of the code