John Lin

Ms.Schuet

AP Comp Sci

26 May 2016

Seating Arrangement Project

Brief overview: The code has five classes, Main, Student, Table, NaughtyGroup, and NiceGroup.

In Main, the program calls files from the project folder and prints out a randomized table group based on the inputs of the files. To understand how to use main, Please Read the “READ\_ME\_Intsruction\_On\_How\_To\_Use\_The\_Code” file. There were a lot more methods I could have implemented it is all in public static void main (String[] args) as that has comments to what each chunk of code does.

|  |
| --- |
| Main  public static void main (String[] args)  private static void fillNaughtyGroups(ArrayList<NaughtyGroup> naughtyGroups,  String[] seperateIntoGroups,Student[] students)  private static void fillNiceGroups(ArrayList<NiceGroup> niceGroups,  ArrayList<Student> helpers, ArrayList<Student> helpees)  private static void fillHelp(String[] seperate, ArrayList<Student> help,  Student[] students)  private static void printToTextDocument(ArrayList<Table> tables)  private static void print(ArrayList<Table> tables)  private static int getNumTables(int length, int tableSize)  public static String getFileAsString(String filename)  public static void writeStringToFile(String filename, String text) |

In Student, a student object is created and it has its name. When students are compatible or incompatible with each other, main adds them to each other’s compatible or incompatible or arrayList inside the student’s class. Most of the code is just structured around creating student objects and comparing students to each other by their name and their compatibility to select the seating arrangements.

|  |
| --- |
| Student  public class Student  public Student( String n)  public Student( String n , int num)  public String getName()  public void setName(String n)  public int getID()  public void setID(int num)  public boolean getSitInFront()  public void setSitInFront(boolean s)  public boolean getNaughtiness()  public void setNaughtiness(boolean s)  public void setInNiceGroup(boolean s)  public boolean getInNiceGroup()  public void addCompatible(Student s()  public void addinCompatible(Student s)  public void removeCompatible(Student s)  public void removeinCompatible(Student s)  public boolean isCompatible(Student s)  public boolean isinCompatible(Student s)  public void addSatWithStudent(Student s)  public void removeSatWithStudent(Student s)  public boolean satwithStudent(Student s) |

In Table, the code is solely focused on seeing if a student could have been added to a table group based on the student’s features. Its add method returns a string to see if the student is added. The test add method returns a Boolean to see if the add would be suitable.

|  |
| --- |
| Table  public class Table {  public Table ( int s, int n)  public String add( Student s)  public String testAdd( Student s)  public void remove( Student s)  public int sitInFrontValue()  public String toString()  public int getSize()  public int getCurrentSize()  public void setSize(int s() |

The naughty and nice groups have the same code and the reason for that is in Design Implementation section. The classes are just easier ways to visualize grouping students into their respective groups.

|  |
| --- |
| Naughty and Nice Group  public class NaughtyGroup {    public NaughtyGroup()  public void addStudent(Student s)  public void removeStudent(Student s)  public int getSize()  public Student get(int i) |

Design implementation:

A design implementation is using both Naughty and Nice Group classes. The classes literally do the same thing but when coding the project, it would get too confusing to think through so having two different classes allowed my mind to understand the code structure easier. If I do get some time, I would reduce my group but using just one group class instead.

Another implementation was to add the students in niceGroups first to the tables. This would be needed to be done first as if there was a table size of 4 and three students already filled, a group of students that were supposed to sit together could not be added to the table. Thus, it would be better just to loop through the niceGroups first and add both students in the group instead of breaking up a group due to the table size not being large enough as both students needed to be added at once.