**CSCI 3302 Programming Assignment 02 (100 Points)**

**Due: Oct 2, 8:00 AM**

GITHUB Link: [Program 02](https://classroom.github.com/a/7TlG_1X0)

Objectives:

* Demonstrate a basic understanding of Java programming.
* Demonstrate a basic understanding of recursion.

Assignment Assistance:

* This homework assignment is due prior to the date and time specified above.
* This assignment is restricted to individual effort. You may not receive help from any other person except the instructor or the AARC (help from the AARC must be well documented!).
* Any resource used (other than Dr. Becnel or the course text) must be documented in the code (as comments) detailing the source and describing exactly what was learned and how that information was used. Submissions will be severely penalized if copied in part or in whole from any source.

Problem Description:

1. Your task is to write a Java class named Ticket in a class file called Ticket.java.
2. The Ticket class contains a single **public** static method called findSeatID that returns a value of type int. This method has a single parameter of type String called partition.
   1. You may define any additional helper methods that you need. Make sure to declare them as private and static.
   2. Since all methods in this class are static, no constructor is needed. The main method should be able to call findSeatID without instantiating a Ticket object.
   3. **You are not allowed to use iteration within any method in your submission. Instead, you must use recursion.** Also, exhaustively listing out all possibilities is not considered a reasonable solution.
3. The Ticket class is part of a complex computer program used by a major airline company. The purpose of your method, findSeatID, is to calculate the seat id based on the following instructions:
   1. For boarding passes, instead of zones or groups, this airline uses ***binary space partitioning*** to seat people. A seat might be specified like FBFBBFFRLR, where F means "front", B means "back", L means "left", and R means "right". This binary space partition is the argument provided to the findSeatID method as a String.
   2. The first 7 characters will either be F or B; these specify exactly one of the 128 rows on the plane (numbered 0 through 127). Each letter tells you which half of a region the given seat is in. Start with the whole list of rows; the first letter indicates whether the seat is in the front (0 through 63) or the back (64 through 127). The next letter indicates which half of that region the seat is in, and so on until you're left with exactly one row.
   3. For example, consider just the first seven characters of FBFBBFFRLR:
      * Start by considering the whole range, rows 0 through 127.
      * F means to take the lower half, keeping rows 0 through 63.
      * B means to take the upper half, keeping rows 32 through 63.
      * F means to take the lower half, keeping rows 32 through 47.
      * B means to take the upper half, keeping rows 40 through 47.
      * B keeps rows 44 through 47.
      * F keeps rows 44 through 45.
      * The final F keeps the lower of the two, row 44.
   4. The last three characters will be either L or R; these specify exactly one of the 8 columns of seats on the plane (numbered 0 through 7). The same process as above proceeds again, this time with only three steps. L means to keep the lower half, while R means to keep the upper half.
   5. For example, consider just the last 3 characters of FBFBBFFRLR:
      * Start by considering the whole range, columns 0 through 7.
      * R means to take the upper half, keeping columns 4 through 7.
      * L means to take the lower half, keeping columns 4 through 5.
      * The final R keeps the upper of the two, column 5.
   6. So, decoding FBFBBFFRLR reveals that it is the seat at row 44, column 5.
   7. **Every seat also has a unique seat ID: multiply the row by 8, then add the column.** In this example, the seat has ID 44 \* 8 + 5 = 357.
   8. Here are some other boarding passes:
      * BFFFBBFRRR: row 70, column 7, seat ID 567.
      * FFFBBBFRRR: row 14, column 7, seat ID 119.
      * BBFFBBFRLL: row 102, column 4, seat ID 820.
4. Your file Ticket.java should NOT contain a main method or any extraneous testing code. You can include files with testing code; however, these will not be considered for grading. If you wish to include non-working code for insight into your thought process, make sure to contain it within comment blocks and ensure that submission successfully compiles.
5. Your program should work in the GitHub codespace (Linux environment) and locally (Windows environment).

Hints:

1. Your public method, findSeatID, should not be recursive.
2. You should probably have at least two helper methods; one to recursively determine the row, and the other to recursively determine the column.
3. You may use either head recursion or tail recursion, though tail recursion is the better fit for this problem. Your recursive helper method will require multiple parameters.
4. To assist with this assignment, study the binary search algorithm discussed in the text.

Submission:

* Review the Evaluation below to ensure you have met all the requirements.
* Commit electronic copy of Ticket.java to GitHub. Upload a backup copy to D2L.

**Evaluation**

|  |  |
| --- | --- |
| **Automatic Deductions:** |  |
| Late/Not Submitted | -100 |
| Code not submitted to GitHub | -30 |
| Code does not run/compile | -50 |
|  |  |
| **Earn Points for the following:** |  |
| Code has comment header with name, section, date | 5 pts |
| Code organization, structure, and indention is appropriate (SHFT + ALT + F in VS Code) | 5 pts |
| Code is well and meaningfully commented. | 5 pts |
| Appropriate variable and method names that follow Java conventions | 5 pts |
| Instructions correctly followed for fields, class, methods | 10 pts |
| Only recursion is present, no iteration or exhaustive list | 10 pts |
| Code uses recursion to provide correct solution (partial credit given) | 60 pts |