Listen

Material covered:

- Review of COMP 1010 (arrays and simple strings) in Part A
- Creating basic classes and objects in Part B
- Using arrays of objects in Part B

top

Please ensure that you:

- solve all problems with programs written in the Java programming language;
- follow the programming standards;
- submit a separate complete program for each part that can be saved in a single Java source code file, and compiled and executed without any other Java files; and
- submit your solution to both parts of the assignment by the assignment due date.

If you submit a solution that does not meet these requirements (for example, if the program does not run), you will lose some or all marks.

top

Part A: Airline seating

Millions of commercial airline flights cross the world's cities every year, and each one of these flights must solve what seems like a simple problem: how do you arrange where each passenger on one of those flights sits? In this assignment, we will solve a variation on that problem, where we will observe one restriction on the seats: people who book a flight together will be — if possible — seated together.

Your program will process data from an array of strings (call this the "bookings" array). The first array element will contain a number indicating how many seats the aircraft has. The remaining elements will contain information about groups of passengers who have booked seats on the flight. The first element of this grouping will be a number that indicates how many people are in the group. The remaining elements identify the individual passengers in that group by their last and first name. For example, the array might contain the following 10 strings:

```
8

2

Nobbly, Greg

Nobbly, Jo-Anne

1
```

```
Lee, Sook

3

Lukas, Stephie

Lukas, Cambridge

Lukas, Ogden
```

which means there are 8 seats on the flight, the first group of passengers has 2 people (with their names below), the second group has 1 person, and the third group has 3 people.

> Normally this data would come from input, such as lines from a text file, not from an array. That's the next assignment.

The seating chart for the flight will be stored in an array of strings (call this the "seats" array), whose size is equal to the number of seats on the flight. Each position in the array corresponds to a seat on the flight; the first element in the array is seat 1, the second element is seat 2, etc. Initially all the seats are empty. Passengers will be seated by placing their name in this array.

Your program will seat passengers as follows:

- It will create a "seats" array of the appropriate size (given in the first position of the "bookings" array).
- It will process the remaining items in the "bookings" array of strings. For each group of passengers it will attempt to seat them as follows:
    1. First, it will see if there are enough seats remaining on the flight for everyone in the group; if not, it will display an error message and not assign seats to anyone in the group.
    2. Secondly, it will go through the "seats" array to determine if a block of empty seats large enough to seat the entire group together is available (for example, if the group size is 3, it will see if there are 3 consecutive empty seats).
        - If there is at least one such block of seats anywhere in the array, randomly assign the group to one of these blocks by randomly selecting an element in the "seats" array. If that seat is empty, determine if enough successive array elements (seats) are also empty to seat the entire group; if so, seat them there. Otherwise, randomly try another seat, repeating until successful. (Note that this is not the most efficient approach...)
        - If there is no such block, randomly assign each passenger in the group individually to a seat (i.e., the group will be split up). For each passenger, pick random seat numbers until you find an empty seat.

When the seats for the flight have been assigned, print the contents of the "seats" array (neatly) to the display. Show both filled and empty seats, describing empty seats as "empty" (do not display the word "null"). Label each line of output with the seat number, where the first seat on the aircraft is seat 1.

Note that using the algorithm above, passengers never move once their seat has been assigned. A group may be split up due to the result of the randomly chosen seat locations. For example, the sample data above may result in the following output:

```
Seat 1: Lukas, Ogden

Seat 2: Lee, Sook

Seat 3: -empty-

Seat 4: Lukas, Stephie

Seat 5: Nobbly, Greg

Seat 6: Nobbly, Jo-Anne

Seat 7: -empty-
Seat 8: Lukas, Cambridge
```

Because the seat assignments for the first two groups left no block of three seats empty, each person in the third group had to each be assigned a random seat. If the first two groups had been seated differently, the third group may have been able to sit together.

Your main program will seat three flights. Each flight will be described by a "bookings" array that you will pass to a method that processes the array and determines the flight seating. Call the method three times in your main program, once for each "bookings" array. Display the seating plan for each flight under an appropriate title. Note that your main program should include only declarations, assignments, method calls, and output statements (no loops or other control structures); all the processing should be performed within other methods.

> This is programming standard number 18, and should be followed in all your assignments, not just this one.

First, seat the flight containing the sample data shown above (eight seats, three groups). Next, create and seat your own booking array: give the flight 12 seats, and exactly five groups containing a total of 12 passengers. Make up the passenger names. Finally, download the class Bookings.class; inside is a method `public static String[] getBookings()` which will return an bookings array that you should seat.

top

Part B: Date book

A classic computer application is the electronic date book: a list of daily events stored in a calendar. Write a Java program that can be used as a simple date book. The date book will use a separate array for each month of the year, with one array entry for each day in the month (0 = first day of the month, 1 = second day of the month, etc.). The date book is "simple" because at most one event will be allowed on each day.

Your program should include an `Event` class that will be instantiated for each event in the date book. The Event class consists of three instance variables: the starting time of the event (a String), the name of the event (a String), and the priority of the event (an integer value in the range 1-3, with 3 being the highest priority). Make all your instance variables private and write appropriate instance methods for processing them, including a constructor, toString(), and any others needed by your program.

Each month in the date book will be stored as an array of Events. Days with an event will point to the appropriate Event object, and days without will contain null.

Your program will produce three types of output (three views) for each month:

- A calendar view, listing the days of the week (Sunday to Saturday) horizontally across the top, and the days of the month underneath. Next to each day where there is a scheduled event, print an asterisk "*". Like in a real calendar, the first day of the month can be any of the weekdays.
- A list of events, showing the day of the month and the details of the event (using toString). These can be displayed in the order that they occur during the month. Do not display lines of output for days without events.
- A list of the highest-priority events (those with priority 3). List *only* the names of the events, not the starting time or priority.

For example:

```
Sun Mon Tue Wed Thu Fri Sat
      1   2*  3   4   5   6
 7    8   9  10  11* 12* 13
14* 15* 16* 17  18  19* 20
21  22  23  24  25* 26* 27
28* 29  30

Events:
 2: 4:00 PM: Event #8 (priority 3)
11: 2:00 PM: Event #10 (priority 1)
```

```
12: 5:00 PM: Event #7 (priority 3)
14: 10:00 PM: Event #2 (priority 1)
15: 6:00 PM: Event #6 (priority 3)
16: 7:00 PM: Event #5 (priority 2)
19: 3:00 PM: Event #9 (priority 2)
25: 9:00 PM: Event #3 (priority 1)
26: 11:00 PM: Event #1 (priority 3)
28: 8:00 PM: Event #4 (priority 1)

High-priority events:
 2: Event #8
12: Event #7
15: Event #6
26: Event #1
```

Write a separate static method to generate each view. Test your program as follows:

- Create two arrays of Events, `september` and `october`; the first has 30 entries, the second 31.
- Place 10 events at random locations in each array. Write one method to do this, and call it twice (i.e. the two arrays will contain the same ten events on different days). Make up reasonable values for the names and starting times of the events: make them all recognizably different and meaningful (not just "Event 1", "Event 2", etc.). The method should choose a random number between 1-3 for event priority.
- Display the complete output (all three views) for September. Choose the first day of the month randomly (pick a random number 0-6 where 0 is Sunday).
- Display the complete output (all three views) for October. The first day of the month will be (the first day of September + 30) mod 7: that is, it will continue after September's last day.

Make sure to title both parts of your output (September and October) clearly. Remember from COMP 1010 that you can choose a random number between 1 and *n* using `(int)(Math.random() * n) + 1`.

top

Copyright © University of Manitoba, Distance and Online Education.
Distance and Online Education uses Google Analytics