CS 101 - Algorithms & Programming I

Fall 2025 - Lab 6

Due: Week of November 17, 2025

*Remember the **honor code** for your programming assignments.*
*For all labs, your solutions must conform to the CS101 style **guidelines**!*
All data and results should be stored in variables (or constants where appropriate) with meaningful names.

The objective of this lab is to learn static methods. Remember that analyzing the problems and designing the solutions on a piece of paper *before* starting implementation/coding is always a best practice.

For the methods below, you should **not** use any built-in methods that perform the mentioned tasks directly unless explicitly stated otherwise. The mentioned methods should be implemented from scratch.

0. Setup Workspace

Start VSC and open the previously created folder named `labs`. Now, under the `labs` folder, create a new folder named `lab6`. In this lab, you are to have 3 Java classes/files (under `labs/lab6` folder) as described below. A third Java file containing the revision should go under this folder as well. We expect you to submit a total of 4 files, including the revision, **without compressing** them. Do *not* upload other/previous lab solutions in your submission. The user inputs in the sample runs are shown in blue.

First, we will design and implement some *static* methods. These methods will then be used in a program to determine the associated properties of the integers input by the user.

1. Theater Seating

Create a new/empty file of your own under the `lab6` folder named `Lab06_Q1.java` with a class with the same name. The program will simulate a theater seating arrangement. First, the program should ask the user to enter the number of rows (`r`) and the number of columns (`c`) in the theater. The seats will be represented using a two-dimensional array, and at the start, all seats are empty.

After that, the user will enter `r` integers, where each integer represents the number of people assigned to that row. For example, if the user specifies that the theater has 5 rows and 7 columns, the input for the groups could be:

```
Enter the groups for 5 rows and 7 columns: 1, 4, 6, 7, 4
```

Your program should read exactly `r` group sizes, first validate the input, if any group size `k` is greater than `c`, print an error, and stop. For each row, you must seat the `k` people so that the minimum distance between adjacent seated people is **as large as possible** by spreading them out as evenly as the discrete grid allows, using both ends of the row.

To achieve this, you should divide the total distance from the first seat `(0)` to the last seat `(c-1)` into `k-1` gaps. The base gap size is calculated as: `g = (c-1) / (k-1)` using integer division, and the remainder: `R = (c-1) % (k-1)` tells us how many of those gaps need to be slightly larger.

The first `R` gaps are assigned size `g+1`, while the remaining gaps get size `g`. Starting at seat `0`, each subsequent position is determined by adding the appropriate gap, ensuring the last person always ends up at seat `c-1`. This method spreads people out as evenly as possible, guaranteeing that no two seated individuals are closer together than necessary, thereby maximizing the minimum spacing. Represent occupied seats with `'x'` and empty seats with `'-'`. After seating all rows (or detecting an error), print the result.

```
Example 1:
Enter number of rows: 5
Enter number of columns: 7
Enter the groups for 5 rows and 7 columns: 1, 4, 6, 7, 4
x------
x-x-x-x
x-xx-xx
xxxxxxx
x-x-x-x

Example 2:
Enter the groups for 3 rows and 4 columns: 2, 3
Error: expected 3 group sizes but got 2.

Example 3:
Enter the groups for 2 rows and 5 columns: 6, 3
Error: group size 6 exceeds columns 5 in row 1.
```

2. Dynamic Task Scheduler

Create a new/empty file of your own under the `lab6` folder named `Lab06_Q2.java` with a class with the same name. In this part, you will design and implement a console-based Task Scheduling System in Java that begins with fixed-size arrays and then transitions to dynamic ArrayList structures once the initial capacity is exhausted.

Each task has a task name and a priority, where `1: High`, `2: Medium`, and `3: Low`. The program starts by asking the user for the initial capacity; this number defines the sizes of the following two arrays:

- `taskNames[]` stores task names.
- `priorities[]` stores task priorities.

When the arrays are full and the user tries to add another task, the program prints a message that the array is full. At that point, it creates two new lists (ArrayLists for names and priorities) and copies all existing tasks from the arrays into these lists, and continues using the lists thereafter. From then on, any new tasks are added directly to these ArrayLists, allowing the system to grow dynamically without restrictions.

The program is menu-driven and repeatedly prompts the user until they choose to exit. All name comparisons should be case-insensitive, task names must be non-empty and unique, and priority inputs must be validated to be within 1–3.

Methods to Implement

- `addTask(String name, int priority)`

  Adds a new task to the system. If there is space in the arrays, store the task in the arrays; if the arrays are full, switch once to ArrayLists, copy all existing tasks, and then add the new task. Reject duplicate or empty names and priorities outside 1–3 with a clear message.

- `removeTask(String name)`

  Removes a task by its name. This operation must work whether the system is using arrays or ArrayLists. If the task does not exist, display “Task not found!”.

- `updateTaskPriority(String name, int newPriority)`

  Updates the priority of an existing task. First, display the current priority to the user, then request a different new priority (1–3). If the user enters the same priority as the current one, reject and re-prompt. If the task is not present, display “Task not found!”.

- `searchTask(String name)`

  Searches for a task by name and, if found, displays its current priority. If the task does not exist, print “Task not found!”. (You are allowed to use built-in methods for this question)

- `viewAllTasks()`

  Displays all tasks (pending tasks only—this simplified version has no completed-task list) with their names and priorities. If there are no tasks, print an informative message such as “No tasks to display.”.

- `exit()`

  Terminates the program gracefully by ending the menu loop.

```
Enter initial task capacity: 2
=== Task Scheduler ===
1. Add Task
2. Remove Task
3. Update Task Priority
4. Search Task
5. View All Tasks
6. Exit
Choose an option: 1
Enter task name: T1
Enter priority (1=High, 2=Medium,
3=Low): 1
Task added successfully!

=== Task Scheduler ===
1. Add Task
2. Remove Task
3. Update Task Priority
4. Search Task
5. View All Tasks
6. Exit
Choose an option: 5
Tasks:
1. T1 (Priority 1)

=== Task Scheduler ===
1. Add Task
2. Remove Task
3. Update Task Priority
4. Search Task
5. View All Tasks
6. Exit
Choose an option: 2
Enter task name to remove: T1
Task removed successfully.

=== Task Scheduler ===
1. Add Task
2. Remove Task
3. Update Task Priority
4. Search Task
5. View All Tasks
6. Exit
Choose an option: 5
No tasks to display.

=== Task Scheduler ===
1. Add Task
```

```
=== Task Scheduler ===
1. Add Task
2. Remove Task
3. Update Task Priority
4. Search Task
5. View All Tasks
6. Exit
Choose an option: 5
Tasks:
1. T1 (Priority 1)
2. T2 (Priority 2)
3. T3 (Priority 1)

=== Task Scheduler ===
1. Add Task
2. Remove Task
3. Update Task Priority
4. Search Task
5. View All Tasks
6. Exit
Choose an option: 3
Enter task name to update: T3
Current priority: 1
Enter priority (1=High, 2=Medium,
3=Low): 1
New priority cannot be the same as the
current priority. Try again.
Enter priority (1=High, 2=Medium,
3=Low): 3
Priority updated.

=== Task Scheduler ===
1. Add Task
2. Remove Task
3. Update Task Priority
4. Search Task
5. View All Tasks
6. Exit
Choose an option: 5
Tasks:
1. T1 (Priority 1)
2. T2 (Priority 2)
3. T3 (Priority 3)

=== Task Scheduler ===
1. Add Task
2. Remove Task
3. Update Task Priority
```

```
2. Remove Task
3. Update Task Priority
4. Search Task
5. View All Tasks
6. Exit
Choose an option: 1
Enter task name: T1
Enter priority (1=High, 2=Medium,
3=Low): 1
Task added successfully!

=== Task Scheduler ===
1. Add Task
2. Remove Task
3. Update Task Priority
4. Search Task
5. View All Tasks
6. Exit
Choose an option: 1
Enter task name: T2
Enter priority (1=High, 2=Medium,
3=Low): 2
Task added successfully!

=== Task Scheduler ===
1. Add Task
2. Remove Task
3. Update Task Priority
4. Search Task
5. View All Tasks
6. Exit
Choose an option: 1
Enter task name: T3
Enter priority (1=High, 2=Medium,
3=Low): 1
Array full! Switching to dynamic
ArrayList...
Task added successfully!
```

```
4. Search Task
5. View All Tasks
6. Exit
Choose an option: 4
Enter task name to search: T3
Task found. Priority: 3

=== Task Scheduler ===
1. Add Task
2. Remove Task
3. Update Task Priority
4. Search Task
5. View All Tasks
6. Exit
Choose an option: 4
Enter task name to search: T4
Task not found!

=== Task Scheduler ===
1. Add Task
2. Remove Task
3. Update Task Priority
4. Search Task
5. View All Tasks
6. Exit
Choose an option: 6
Quit!
```