Programming Assignment 4

Course Instructors: Erkut Erdem, Hacer Yalım Keleş, Adnan Özsoy
TAs: Sümeyye Meryem Taşyürek, Hikmet Can Doğancı
Programming Language: Java (OpenJDK 11)
Due Date: **Friday, 16.05.2025 (23:59:59)**

Club Fair on Campus

Every spring, Hacettepe University comes alive with the annual University Club Fair a colorful event where student clubs showcase their work, welcome new members, and bring the community together. This year, the Student Council has decided to take things to the next level and make the fair more organized, efficient, and exciting than ever before.

You, as a talented computer engineering student, have been recruited to help plan and manage the setup of the fair and build a campus navigation assistant for visitors.

Thanks to your reputation for clean algorithms and efficient code, you've been given two important missions:

1. **Club Fair Setup Planning:** Help coordinate all booth setup tasks (like setting up tables, hanging banners, powering lights, etc.) by creating a schedule that respects the dependencies between tasks and ensures the earliest possible completion.
2. **Campus Navigator App:** Help visitors easily find their way from one booth to another using the fastest combination of walking paths and golf cart routes across campus.

> These missions rely on your understanding of graph algorithms and regular expressions, and your ability to implement them in Java using a provided starter codebase. Your work will ensure the club fair is efficiently prepared and easy to navigate — and maybe even make this the best club fair yet!

1 Part I – Club Fair Setup Planning

The first step in organizing the University Club Fair is making sure everything is set up on time. From building booths to decorating and setting up electronic displays, there are many tasks that need to be coordinated and some tasks can only begin after others are finished. You've been assigned to develop a scheduling tool that can determine when each task can start, taking into account task durations and dependencies. This will help the organizing committee ensure smooth setup and avoid delays.

1.1 Background Information and Objectives

One of the key projects is the preparation of the Environmental Club's booth for the University Club Fair. The booth setup involves several tasks such as arranging tables, hanging banners, painting decorations, and setting up displays. For each task, you are provided with a task ID, a description, the estimated duration in days, and a list of tasks that must be completed before the task can begin.

The task information is provided in the table below:

Task ID	Task Description	Duration	Depends on
0	Set up tables	2 days	-
1	Hang banners	1 day	0
2	Power up LED displays	1 day	1
3	Paint booth decorations	2 days	0
4	Decorate with posters and lights	1 day	1, 3
5	Final check and approval	1 day	2, 4

Table 1: Environmental Club Setup Task List

The dependencies are structured so that, for example, Task 4 (Decorate with posters and lights) can only start after both Task 1 (Hang banners) and Task 3 (Paint booth decorations) are completed. Task 0 is the first to begin, while Task 5 is the final task that concludes the setup.

Each task is assigned to a different group of student volunteers. Since many tasks are interdependent, it is important to determine the correct schedule so that no team is left waiting unnecessarily. Your role is to calculate the **earliest possible start and end times for each task**, considering the project's constraints, and help ensure that the entire setup is completed in the shortest total time.

Mission Objective: Create a task schedule for each club booth such that all tasks are completed at the earliest possible time, respecting the dependency constraints.

1.2 Input File Format

The input will be provided as an `.xml` file, given as the first command-line argument to your program. The structure is shown below:

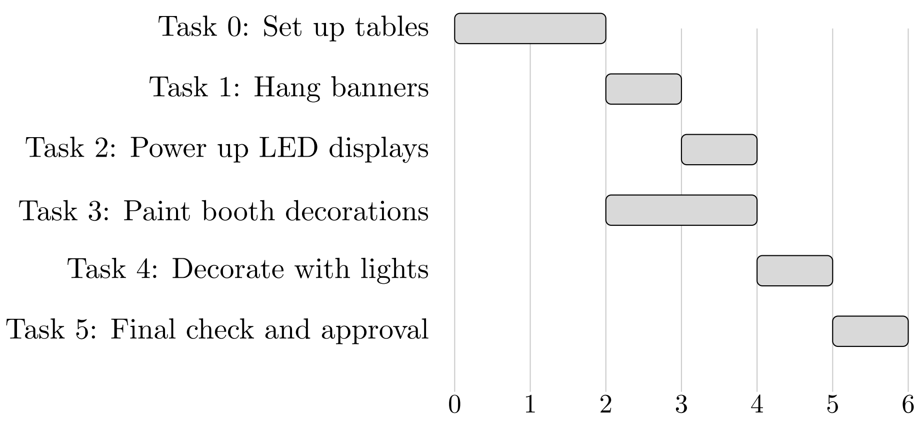
```
<Projects>
  <Project>
    <Name>Club Fair Preparation</Name>
    <Tasks>
      <Task>
        <TaskID>0</TaskID>
        <Description>Set up tables</Description>
        <Duration>2</Duration>
        <Dependencies></Dependencies>
      </Task>
      <Task>
        <TaskID>1</TaskID>
        <Description>Hang banners</Description>
        <Duration>1</Duration>
        <Dependencies>
          <DependsOnTaskID>0</DependsOnTaskID>
        </Dependencies>
      </Task>
      ...
    </Tasks>
  </Project>
</Projects>
```

Each project includes a list of tasks, where each task is defined with an ID, description, duration (in days), and optional dependencies.

1.3 Expected Solution and Output Format

For the project **Club Fair Preparation**, the expected schedule is given in the figure below, which ensures the earliest completion of the project:

As early as possible schedule for Project Club Fair Preparation



From the project timeline, observe that Tasks 0, 1, 4, and 5 are on the *critical path* of the project schedule, meaning that delaying any of these tasks would result in a delay in the overall project completion. Tasks 2 and 3 have some scheduling flexibility (also called mobility). The mobility of the tasks on the critical path is always zero.

Your output must display the earliest possible start and end times for each task:

```
+------------------------------------+
Task ID      | 0 | 1 | 2 | 3 | 4 | 5 |
+------------------------------------+
Start time   | 0 | 2 | 3 | 2 | 4 | 5 |
+------------------------------------+
End time     | 2 | 3 | 4 | 4 | 5 | 6 |
+------------------------------------+
Minimum project duration: 6 days
```

Your output must print the project name, each task's schedule (start and end day), and the minimum duration of the project. Tasks must appear in topological order.

You are expected to complete the following methods:

- `readXML()` and `printSchedule()` in the `ClubFairSetupPlanner` class
- `getEarliestSchedule()` and `getProjectDuration()` in the `Project` class

The expected STDOUT output format for a single project schedule is given below:

```
### CLUB FAIR SETUP START ###
-----------------------------------------------------------------
Project name: Club Fair Preparation
-----------------------------------------------------------------
Task ID Description                               Start  End
-----------------------------------------------------------------
0       Set up tables                               0      2
1       Hang banners                                2      3
2       Power up LED displays                       3      4
3       Paint booth decorations                     2      4
4       Decorate with posters and lights            4      5
5       Final check and approval                    5      6
-----------------------------------------------------------------
Project will be completed in 6 days.
-----------------------------------------------------------------
### CLUB FAIR SETUP END ###
```

The program must format its output to match this structure exactly to receive full points. Any deviation in spacing, header lines, or order may result in point loss during automated grading.

2 Part II - Campus Navigator App

After successfully planning the booth setup for the Club Fair, your next mission is to support visitors during the event. The campus is large and full of activity, and attendees often want to visit multiple booths in different areas. To help them, you are tasked with developing a **Campus Navigator App** that calculates the *fastest route* between any two locations.

2.1 Background Information and Objectives

Visitors can either *walk* or use *golf carts* provided along specific two-way paths. While golf carts move faster, they are only available on certain routes. The goal is to combine walking paths and golf cart routes to guide visitors from their starting point to their destination as efficiently as possible.

Mission Objective: Given the start and end coordinates and the golf cart lines (with their stations), find the fastest route across campus by combining golf cart rides and walking.

Assumptions:

- Golf cart lines operate between listed stations, in both directions.
- Walking is allowed between any two points; average walking speed is **10 km/h**.
- Golf cart speed (in km/h) is provided in the input.
- Waiting time is ignored.

2.2 Input File Format and Reading the Input Data Using Regular Expressions

The input will be given in a `.dat` file, provided as the second command-line argument, which contains:

- The number of golf cart lines on campus
- The X, Y coordinates of the visitor's starting and destination points
- The X, Y coordinates of the golf cart stations for each line
- The average cart speed (in km/h)

A sample input file is structured as follows:

```
num_cart_lines = 2
starting_point = (0, 0)
destination_point = (10000, 1000)
average_cart_speed = 400.00
cart_line_name = "Main Avenue Path"
cart_line_stations = (0, 200) (5000, 200) (7000, 200)
cart_line_name = "Library Loop"
cart_line_stations = (2000, 600) (5000, 600) (10000, 600)
```

Stations are listed in order for each line, with at least two stations per line. Golf carts operate in both directions between adjacent stations. All coordinates represent points on a 2D campus map, measured in meters.

Note: The order of variables and the placement of whitespace characters (tabs or spaces) around the variable names, equal signs, commas, parentheses, and even outside of quotes may vary. To handle these inconsistencies, you are required to use **regular expressions** to parse the input correctly.

You must implement several methods in the `CampusNavigatorNetwork` class to read the necessary parameters. An example method is provided below as guidance on how to implement this part:

```
public int getIntVar(String varName) {
    Pattern p = Pattern.compile("[\\t ]*" + varName +
    "[\\t ]*=[\\t ]*([0-9]+)");
    Matcher m = p.matcher(fileContent);
    m.find();
    return Integer.parseInt(m.group(1));
}
```

This method uses a regular expression to match an integer variable, allowing for flexible spacing and formatting.

You should apply similar logic in the following methods:

- `getStringVar()`: Modify the regular expression so that it matches the value of the given variable as `Group 1` and returns it as a `String`.
- `getDoubleVar()`: Modify the regular expression so that it matches a floating point value (supporting various formats like `5`, `5.2`, `5.0002`, etc.) and returns it as a `Double`.
- `getPointVar()`: Modify the regular expression to match two integers within parentheses, separated by a comma. Parse the values from `Group 1` and `Group 2`, and return a new `Point` instance.

2.3 Expected Solution and Output Format

You should output the time in minutes it will take the user to get to their desired destination using the fastest route to the `STDOUT`, as well as the steps they must go through along that route.

The time should be rounded to the nearest minute. For the given sample input, the expected output is shown below. The minute values in directions **should not be rounded**. Print them as doubles with a precision of **two (2)** floating points.

```
### CAMPUS NAVIGATOR START ###
The fastest route takes 8 minute(s).
Directions
----------
1. Walk from "Starting Point" to "Main Avenue Path Station 1"
for 1.20 minutes.
2. Ride the cart from "Main Avenue Path Station 1" to "Main
Avenue Path Station 2" for 0.75 minutes.
3. Walk from "Main Avenue Path Station 2" to "Library Loop
Station 2" for 2.40 minutes.
4. Ride the cart from "Library Loop Station 2" to "Library
Loop Station 3" for 0.75 minutes.
5. Walk from "Library Loop Station 3" to "Final Destination"
for 2.40 minutes.
### CAMPUS NAVIGATOR END ###
```

You are expected to complete the following methods:

- `getStringVar()`, `getDoubleVar()`, `getPointVar()`, `getCartLines()`, and `readInput()` functions from the class `CampusNavigatorNetwork`
- `getFastestRouteDirections()` and `printRouteDirections()` functions from the class `CampusNavigatorApp`

Your solution will be tested using pre-written test files and will be auto-graded. Be sure to format the output **exactly as shown**.

Grading Policy

- Implementation of the algorithms: 90%
  - Part I – Club Fair Setup Planning: 40%
  - Part II – Campus Navigator App: 50%
    - Regular Expressions: 15%
    - Fastest Route Calculation: 35%
- Output format test: 10%

Run Configuration

Your code will be compiled and run as follows:

```
javac *.java
java Main <projectsXMLFile> <campusnavigatorDATFile>
```

Important Notes

- Do not miss the deadline: **Friday, 16.05.2025 (23:59:59)**.
- Save all your work until the assignment is graded.
- The assignment solution you submit must be your original, individual work. Duplicate or similar assignments are both going to be considered as cheating.
- You can ask your questions via Piazza (https://piazza.com/hacettepe.edu.tr/spring2025/bbm204), and you are supposed to be aware of everything discussed on Piazza.
- You will submit your work via https://submit.cs.hacettepe.edu.tr/ with the file hierarchy given below:
  - b<studentID>.zip
    - Main.java <FILE>

  - ∗ CampusNavigatorNetwork.java <FILE>
  - ∗ Main.java <FILE>
  - ∗ Point.java <FILE>
  - ∗ Project.java <FILE>
  - ∗ RouteDirection.java <FILE>
  - ∗ Station.java <FILE>
  - ∗ Task.java <FILE>
  - ∗ CartLine.java <FILE>
  - ∗ ClubFairSetupPlanner.java <FILE>
  - ∗ CampusNavigatorApp.java <FILE>

- The name of the main class that contains the main method should be Main.java. You MUST use **this starter code** code. The main class and all other classes should be placed directly in your zip archive. Feel free to create other additional classes if necessary, but they should also be inside the zip.
- This file hierarchy must be zipped before submitted (not .rar, only .zip files are supported).
- **Usage of any external libraries is forbidden.**

Academic Integrity Policy

All work on assignments **must be done individually**. You are encouraged to discuss the given assignments with your classmates, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in pseudocode) **will not be tolerated**. In short, turning in someone else's work (including work available on the internet), in whole or in part, as your own will be considered as **a violation of academic integrity**. Please note that the former condition also holds for online/AI sources. Make use of them responsibly, refrain from generating the code you are asked to implement. Remember that we also have have access to such tools, making it easier to detect such cases.

The submissions will be subjected to a similarity check. Any submissions that fail the similarity check will not be graded and will be reported to the ethics committee as a case of academic integrity violation, which may result in suspension of the involved students.