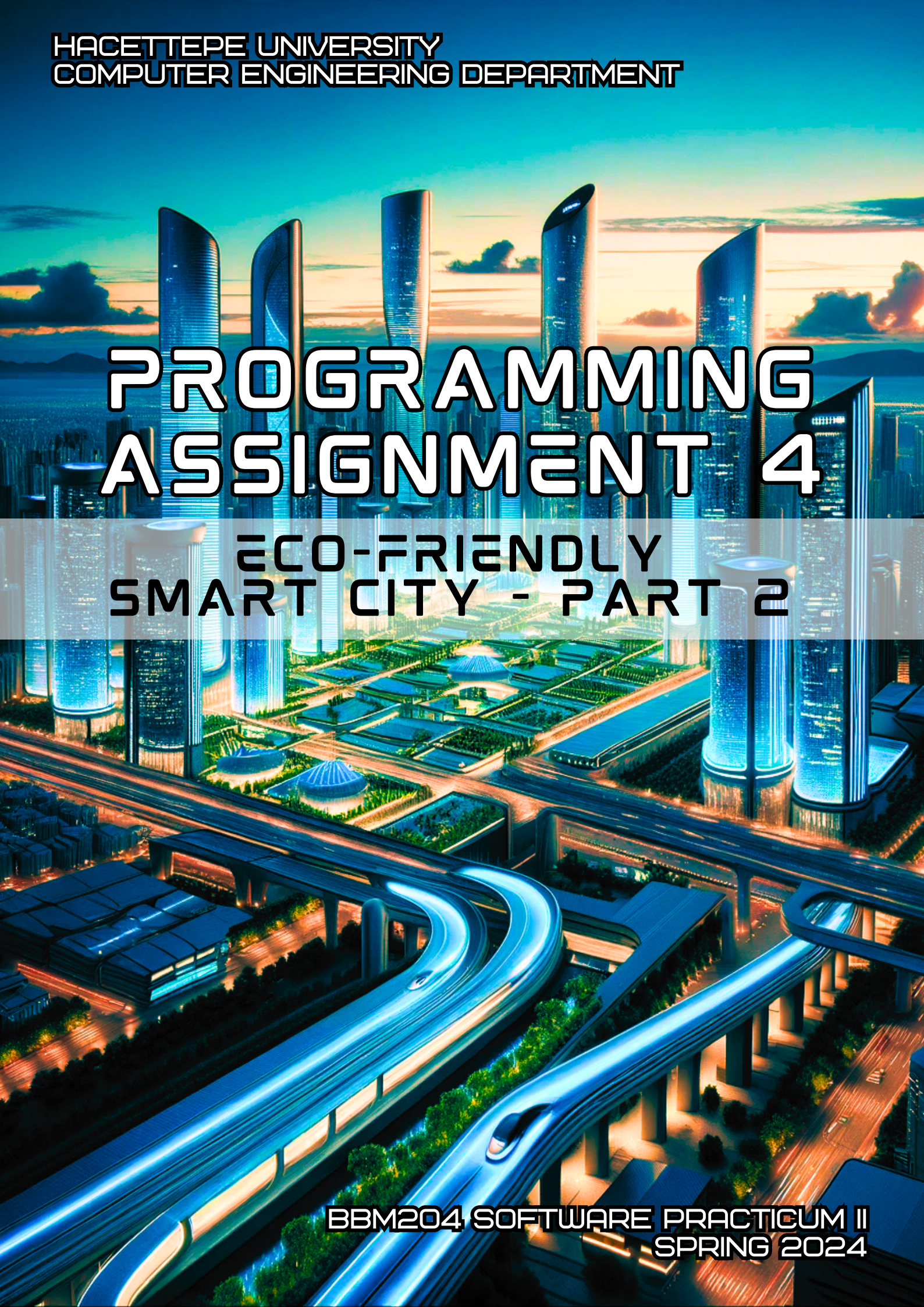
HACETTEPE UNIVERSITY
COMPUTER ENGINEERING DEPARTMENT



PROGRAMMING ASSIGNMENT 4

ECO-FRIENDLY SMART CITY - PART 2

BBM204 SOFTWARE PRACTICUM II
SPRING 2024

HACETTEPE COMPUTER ENGINEERING - BBM204 - SPRING 2024
PROGRAMMING ASSIGNMENT 4 - DEADLINE: 17/05/2024 AT 23:59:59

Topics: Topological Sort, Shortest Path, Regular Expressions

Course Instructors: Assoc. Prof. Dr. Erkut Erdem, Assoc. Prof. Dr. Hacer Yalım Keleş, Assoc. Prof. Dr. Adnan Özsoy
TAs: Alperen Çakın, Dr. Selma Dilek
Programming Language: OpenJDK 11 - **You MUST USE this starter code**
Due Date: **Friday, 17.05.2024 (23:59:59)**

Eco-Friendly Smart City - Part 2: Building the Future

In PA2, you assisted the Ministries of Environment, Urbanization and Climate Change, and Energy and Natural Resources in their initiative titled *Eco-cities: the Future of Urban Türkiye*. This initiative aims to transform urban landscapes across Türkiye by transitioning its major cities into smart eco-cities. **Ankara** was chosen as the pilot city where the new eco-friendly power grid optimization system would be deployed. As brilliant students of the Hacettepe University Computer Engineering Department, you played a pivotal role in optimizing the system's efficiency.

Due to the great success of your energy optimization algorithms, you have been tasked once more to contribute to the next stage of this important initiative. This project is founded on the principles of sustainability, efficiency, and technological innovation. Ankara is being redesigned to be a model for future urban living, integrating advanced technologies such as AI-driven management systems, autonomous transportation, and sustainable energy solutions. As a future hub for technological advancement and ecological harmony, Ankara seeks to revolutionize urban development by implementing a hyperloop train network. This network will not only enhance mobility but also serve as a foundation for expanding smart city concepts globally.

As skilled future engineers, your mission is to contribute to this groundbreaking project by designing and implementing the critical algorithms needed for the projects that will involve scheduling and timely operational management of the hyperloop train network construction, and other projects that will be crucial for the citizens.

1 Part I - Urban Infrastructure Development

The first mission in this urban development series is to lay the foundational groundwork for a hyperloop train network in the smart city. This mission involves planning and constructing the crucial infrastructure, systems, and facilities necessary for the hyperloop train network. Through the dedicated efforts of city planners and cost analysts, you have been assigned a list of projects and their related tasks. Your job is to schedule these tasks for each project and create a timeline for the construction and engineering teams.

1.1 Background Information and Objectives

One of the key projects is the construction of the Main Hyperloop Station. The project task information is provided as follows: for each task, you will receive the task ID, description, duration in days, and the list of tasks that must be completed before that task can commence (see Table below).

Project: Main Hyperloop Station Construction			
Task ID	**Task Description**	**Duration**	**Depends on**
0	Initial Groundwork and Site Preparation	5 days	-
1	Structural Framework Installation	4 days	0
2	Power Systems Integration	2 days	1
3	Control Systems Setup	1 days	2, 4
4	Passenger Facilities Construction	4 days	0, 1
5	Safety Checks and Approvals	3 days	3, 4

The task dependencies are structured so that, for instance, Task 3 (Control Systems Setup) requires the completion of Tasks 2 (Power Systems Integration) and 4 (Passenger Facilities Construction). Task 0 is always the starting point, and Task $N-1$ is the final task in the sequence.

Each task will be handled by a different team, and since some tasks are dependent on the completion of earlier tasks, it is crucial that the assigned teams are informed when their work can begin. This ensures efficient planning and execution. **The goal of this mission is to schedule all tasks within the projects such that they are completed as early as possible, given the dependency constraints.**

Mission Objective:
Develop a task schedule for each given project such that each task is completed at the earliest possible time, adhering to the dependency constraints within the project.

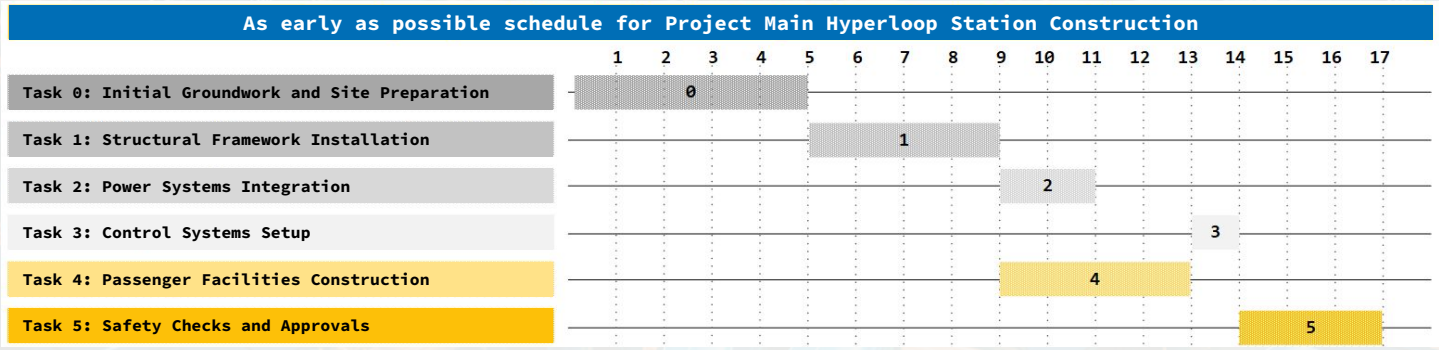
1.2 Input File Format

The input file with all projects will be given in the XML format as the ***first command-line argument***. Your program should parse all the projects into a `List` of `Projects` for further operations. The input file format is shown below:

```
<Projects>
    <Project>
        <Name>Main Hyperloop Station Construction</Name>
        <Tasks>
            <Task>
                <TaskID>0</TaskID>
                <Description>Initial Groundwork and Site Preparation</Description>
                <Duration>5</Duration>
                <Dependencies></Dependencies>
            </Task>
            <Task>
                <TaskID>1</TaskID>
                <Description>Structural Framework Installation</Description>
                <Duration>4</Duration>
                <Dependencies>
                    <DependsOnTaskID>0</DependsOnTaskID>
                </Dependencies>
            </Task>
            ...
        </Tasks>
    </Project>
    ...
</Projects>
```

1.3 Expected Solution and Output Format

For the project ***Main Hyperloop Station Construction***, the expected schedule is given in the figure below, which ensures the earliest completion of the project:



From the project timeline, observe that Tasks 0, 1, 3, 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. Task 2 has scheduling flexibility (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 | 5 | 9 | 13 | 9 | 14 |
+-----------------------------------+
| End time   | 5 | 9 | 11 | 14 | 13 | 17 |
+-----------------------------------+
Minimum project duration: 17 days
```

You are expected to complete:

- ***`readXML()`*** *and* ***`printSchedule()`*** *functions from the class* ***`UrbanInfrastructureDevelopment`***,
- ***`getEarliestSchedule()`*** *and* ***`getProjectDuration()`*** *functions from the class* ***`Project`***.

The expected STDOUT output format for a single project schedule is given below. Each project schedule should be printed in the same format one after another without extra blank lines. Note that your output must match the given output format for full credit.

```
### URBAN INFRASTRUCTURE DEVELOPMENT START ###
-----------------------------------------------------------------
Project name: Main Hyperloop Station Construction
-----------------------------------------------------------------
Task ID   Description                                  Start  End
-----------------------------------------------------------------
0         Initial Groundwork and Site Preparation      0      5
1         Structural Framework Installation            5      9
2         Power Systems Integration                    9      11
3         Control Systems Setup                        13     14
4         Passenger Facilities Construction            9      13
5         Safety Checks and Approvals                  14     17
-----------------------------------------------------------------
Project will be completed in 17 days.
-----------------------------------------------------------------
### URBAN INFRASTRUCTURE DEVELOPMENT END ###
```

2 Part II - Urban Transportation App

After successfully completing the urban infrastructure development planning project, you have been contracted by the Metropolitan Municipality of Ankara to develop a transportation application. This application will assist citizens and visitors in finding the fastest routes from their starting points to their destinations using the newly developed two-way hyperloop train lines that run throughout the city's hyperloop train network.

2.1 Background Information and Objectives

Hyperloop train lines are defined by the stations they connect. Not all train lines intersect directly; therefore, passengers may need to walk between stations to switch lines. Assume the average walking speed is 10 km/h. The speed of a hyperloop train will be provided as input data, also in km/h. For simplicity, waiting times for trains are disregarded. Your task is to develop an algorithm that identifies the fastest route from a user's starting point to their destination.

Mission Objective:
Use the information about the user's desired starting and destination points, and the hyperloop train lines with their stations to find the fastest route from the starting to the destination point.

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 data about the number of train lines in the network, the X, Y coordinates for the user's desired starting and destination points, the X, Y coordinates of the hyperloop train stations for each train line, and the average train speed in km/h. A sample input file shows how this input may be structured:

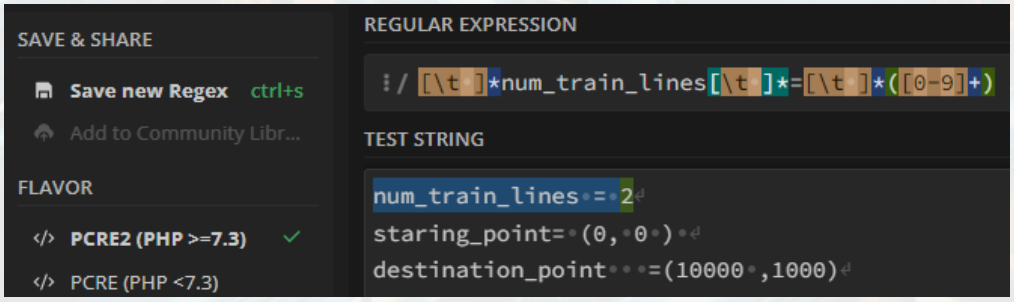
```
num_train_lines = 2
starting_point= (0, 0 )
destination_point =(10000 ,1000)
average_train_speed = 400.00
train_line_name="Kizilay-Sincan"
train_line_stations=(0 , 200) (5000, 200) (7000 , 200)
train_line_name = "Ulus-Beytepe"
train_line_stations = (2000,600) ( 5000 ,600 ) (10000, 600)
```

Stations are listed in order for each train line, with at least two stations per line. Trains travel in a two-way straight line between adjacent stops. All coordinates represent points on a 2-D city map in meters.

Note that the **order of variables** and the **placement** of whitespace characters (tabs or spaces) around the variable names, equal sign, commas, parentheses, and outside of quotes may vary. To handle these variations efficiently, you will use **regular expressions** for input parsing. You must implement three methods in the `HyperloopTrainNetwork` class from the starter code to read the necessary parameters. An example method is provided below as guidance on how to implement this part.

```java
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));
}
```

The first line of the method contains the necessary regular expression for matching an integer variable with an optional number and placement of whitespace characters. Using the variable `num_train_lines` as an example, the figure below illustrates the match and the groups (blue, encompassing the whole match, is **Group 0**, while green, encompassing only the relevant part in parentheses, is **Group 1**).



A logic similar to the example `getIntVar()` method should be used for other methods while parsing the input file.

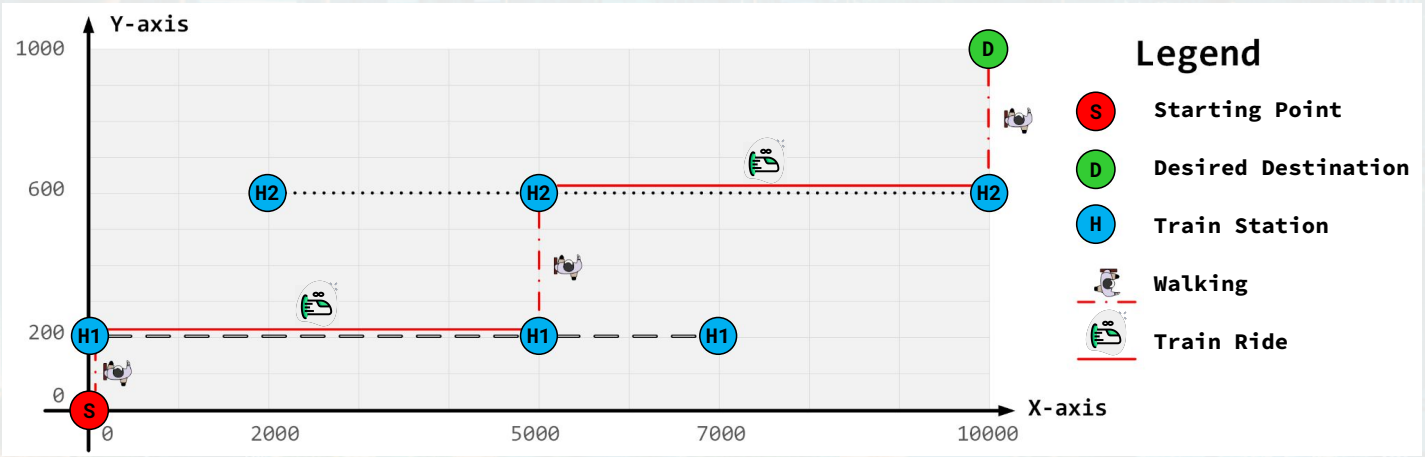
In the `getStringVar()` method, you should alter the given regular expression such that it matches the value of the variable given by `varName` as **Group 1** and returns it.

In the `getDoubleVar()` method, you should alter the given regular expression such that it matches the value of the variable given by `varName` as **Group 1**, parses it as a `Double` and returns it. Your regular expression should support floating point numbers with an arbitrary number of decimals or without any (e.g. 5, 5.2, 5.02, 5.0002, etc.).

In the `getPointVar()` method, you should alter the given regular expression such that it matches the value of the variable given by `varName` as two integers between parentheses separated by a comma as **Group 1** and **Group 2** respectively. Your method should then create an instance of the `Point` class by two integers parsed using **Group 1** and **Group 2**, then return the `Point` instance.

2.3 Expected Solution and Output Format

Following the sample input given in the sample `.dat` file, the expected solution is illustrated below.



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 stops 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.

```
### URBAN TRANSPORTATION APP START ###
The fastest route takes 8 minute(s).
Directions
----------
1. Walk from "Starting Point" to "Kizilay-Sincan Line Station 1" for 1.20 minutes.
2. Get on the train from "Kizilay-Sincan Line Station 1" to "Kizilay-Sincan Line Station 2" for 0.75 minutes.
3. Walk from "Kizilay-Sincan Line Station 2" to "Ulus-Beytepe Line Station 2" for 2.40 minutes.
4. Get on the train from "Ulus-Beytepe Line Station 2" to "Ulus-Beytepe Line Station 3" for 0.75 minutes.
5. Walk from "Ulus-Beytepe Line Station 3" to "Final Destination" for 2.40 minutes.
### URBAN TRANSPORTATION APP END ###
```

You are expected to complete:

- *`getStringVar()`, `getDoubleVar()`, `getPointVar()`, `getTrainLines()`, and `readInput()` functions from the class `HyperloopTrainNetwork`,*
- *`getFastestRouteDirections()` and `printRouteDirections()` functions from the class `UrbanTransportationApp`.*

HACETTEPE COMPUTER ENGINEERING - BBM204 - SPRING 2024
PROGRAMMING ASSIGNMENT 4 - DEADLINE: 17/05/2024 AT 23:59:59

Must-Use Starter Codes

You MUST use **this starter code**. All classes should be placed directly inside your **zip** archive. Feel free to create other additional classes if necessary, but they should also be directly inside **zip**.

Grading Policy

- Implementation of the algorithms: 90%
  - Part I - Urban Infrastructure Development: 40%
  - Part II - Urban Transportation App: 50%
    - Regular Expressions: 15%
    - Fastest Route Calculation: 35%
- Output test: 10%

Note that you need to get a NON-ZERO grade from the assignment to get the submission accepted. Submissions graded with 0 will be counted as NO SUBMISSION!

Important Notes

- Do not miss the deadline: **Friday, 17.05.2024 (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/spring2024/bbm204), and you are supposed to be aware of everything discussed on Piazza.
- You must test your code via Tur[6]Bo Grader (**does not count as submission!**).
- You must submit your work via https://submit.cs.hacettepe.edu.tr/ with the file hierarchy given below:
  - **b<studentID>.zip**
    - HyperloopTrainNetwork.java <FILE>
    - Main.java <FILE>
    - Point.java <FILE>
    - Project.java <FILE>
    - RouteDirection.java <FILE>
    - Station.java <FILE>
    - Task.java <FILE>
    - TrainLine.java <FILE>
    - UrbanInfrastructureDevelopment.java <FILE>
    - UrbanTransportationApp.java <FILE>
- The name of the main class that contains the main method should be **Main.java**. **You MUST use this starter 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.**

Run Configuration

Your code will be compiled and run as follows:

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

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 the material found on the web as everything on the web has been written by someone else.

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.