Neighbour Helpers

In the Algorithms and Data structures project you will implement a **Neighbour Helpers app** that helps connect people who need help with household chores, yard work, home repairs, etc. with others who are capable and willing to help. On one side, users can post jobs they need done, on the other side are users who would like to do the jobs.

You will use the data structures you learn about and work with on the theoretical sessions and exercises throughout the semester. You can use only the code you programmed **yourself**, or the code provided in the context of this project (during the lectures and exercises). You should work on the project throughout the semester, as there are 2 intermediate submissions.
You should work on the project alone; this is NOT a team project. You cannot submit code you did not write yourself. All submissions will be checked, and any suspicion of a fraud will be reported to the dean according to the exam regulations.

You cannot use any third-party code or Java library (e.g. java.Util) without an explicit and prior confirmation by us.
The project is divided into three parts and the assignments will be released throughout the semester. There are 2 intermediate deadlines, and you need to submit your code into canvas.

Code Guidelines

Please follow Oracle's Java Code Conventions, which can be downloaded here:

http://www.oracle.com/technetwork/java/codeconv---138413.html
http://www.oracle.com/technetwork/java/codeconventions---150003.pdf
Most importantly:

- Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive.
  - E.g. *MyClass*
- Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.
  - E.g. *runMyMethod()*
- Variables are in mixed case with a lowercase first letter. Internal words start with capital letters.
  - E.g. *myVariable*
- The names of variables declared class constants should be all uppercase with words separated by underscores ("_").
  - E.g. *MY_CONSTANT*
- Opening curly brackets on the same line, closing curly brackets on a new line.
- Use spacing between statements and operators, not between consecutive brackets.
  - E.g. *if ((a == b) && (c == d)) { …*

Mandatory Classes and methods

Your project should include a class named Main that contains a main method with all your test cases and examples.
All classes should implement the ToString() method.

Part 1: Basic classes

Deadline: 24.11.2024

Feedback: 27.11.2024 and 28.11.2024

Design and implement classes to represent users and jobs. There are two categories of jobs, paid and unpaid. All classes should store a unique ID of the object and store the following:

- A user has a name, email address, and street address
- A job has a title, description, and category. A paid job also has a price.

Each class needs to implement the ToString() method which prints the details of the object in the following format:

- User ID, name, email address, street address
- Job ID, title, description, category, and in case it is a paid job, also the price

Implement the interface iNeighbourHelper and provide the implementation for the methods listed in the "Part 1" section, namely:

- addUser(…)
- addJob(…)
- printAllUsers(…)
- printAllJobs(…)
- findUser(…)
- findJob(…)

What to consider: How would you store the relationship between users and jobs? What is the best data structure to keep track of users and jobs in the system?

Important: Do not change the provided interface. Add the interface file to your project and implement the interface.

Part 2: System logic

Deadline: 15.12.2024

Feedback: 18.12.2024 and 19.12.2024

Design and implement job posting and job application mechanisms. Each user can post a job, and/or apply for a job. However, a user cannot apply for a job he or she posted.

When applying for a job, use the first-come, first-served policy. Once someone applies for a job, the job is automatically given to them. All other applicants should then be informed that the job is no longer available. The methods findAvailableJobs(…) and findAvailableJobsInCategory(…) return only the available jobs, ordering the paid jobs first.

Implement the methods of the provided interface which are listed in section “Part 2”, namely:

- findAvailableJobs(…)
- findAvailableJobsInCategory(…)
- removeJob(…)
- applyForJob(…)

What to consider: You might need to update the implementation of Part 1. What is the best data structure to keep track of users and jobs in the system? Identify the most frequent operations of your system and optimize their implementation (in terms of data structures and algorithms).

Important: Do not change the provided interface. Add the interface file to your project and implement the interface.

Part 3: Final

Deadline: 10.01.2025 (one week before the written exam)

To do the jobs, users must go to the homes of those who posted the jobs. In this final part, you will use a graph to represent the city/neighborhood. The nodes of the graph will represent the streets and edges will represent the distances between them. The street numbers will be omitted. You will also need to implement a shortest path algorithm.

Implement the method of the provided interface listed in the section “Part 3”, namely:

- addStreet(…)
- connectStreets(…)
- getDirections(…)

What to consider: It might be necessary to update the implementation of Part 1 and Part 2. What are the best data structures to keep track of users and jobs? What is the most efficient way to store relationships between users and jobs?

Important: Do not change the provided interface. Add the interface file to your project and implement the interface.

Interface

```
public interface iNeighbourHelper  {

	/************************* PART 1 *************************/
	/*
	 * Add a new user with given parameters: name, email address, and street
address.
	 *
	 * @param name
	 *
	 * @param email
	 *
	 * @param street address
	 *
	 * @return ID of the user
	 */
	public int addUser(String name, String email, String street);

	/*
	 * Add a job with given parameters: title, description, category, isPaid,
price.
	 *
	 * @param title
	 *
	 * @param description
	 *
	 * @param category, e.g. repair, general housework, yard work, ironing,
window washing, etc.
	 *
	 * @param isPaid - indication whether it is a paid job or not
	 *
	 * @param price
	 *
	 * @param userID - of the user who is creating the job
	 *
	 * @return ID of the job
	 */
	public int addJob(String title, String description, String category,
boolean isPaid, float price, int userID);

	/*
	 * Print all users in the following format:
	 * "user ID, name, email, street address"
	 */
	public void printAllUsers();

	/*
	 * Print all jobs in the following format:
	 * "job ID, title, description, category, price (in case of a paid job)"
	 */
	public void printAllJobs();

	/*
	 * Return user based on userID or null if the user does not exist.
	 *
	 * @param userID
	 *
	 * @return User object
	 */
```

```
        public User findUser(int userID);

        /*
         * Return job based on jobID or null if the job does not exist.
         *
         * @param jobID
         *
         * @return Job object
         */
        public Job findJob(int jobID);


        /************************** end of PART 1 **************************/

        /************************** PART 2 **************************/

        /*
         * Return Vector of available jobs.
         *
         * @return Vector of Job objects
         */
        public Vector findAvailableJobs();

        /*
         * Return a Vector of available jobs in the given category.
         *
         * @return Vector of Job objects
         */
        public Vector findAvailableJobsInCategory(String category);

        /*
         * Remove job from the app.
         *
         * @param job ID
         *
         * @return true if successful, false otherwise
         */

        public boolean removeJob(int jobID);

        /*
         * User applies for job.
         *
         * @param userID
         *
         * @param jobID
         *
         * @return true if successful, false otherwise
         */
        public boolean applyForJob(int userID, int jobID);

        /************************** end of PART 2 **************************/

        /************************** PART 3 **************************/

        /*
         * Add a street.
         *
         * @param name of the street
```

```
         */
        public void addStreet(String street);

        /*
         * Connects two streets.
         *
         * @param street1
         *
         * @param street2
         *
         * @param distance
         */
        public void connectStreets(String street1, String street2, int distance);

        /*
         * Return the shortest path from the user's home to all the jobs the user
applied.
         *
         * @param userID
         *
         * @return Vector of streets to visit
         *
         */
        public Vector getDirections(int userID);

        /************************** end of PART 3 **************************/
}
```