# Distributed Systems - Assignment 1: Uber-like Application

**Objective:**

The goal is to build a basic Uber-like application where the server handles ride requests from customers and driver offers. The system should support multiple customers and drivers interacting with the server at the same time. You're required to implement it **using socket programming and multithreading.**

---

## Assignment Overview:

You are required to implement a server and client application using Java sockets, where the server will handle multiple customer and driver requests concurrently. The customers will request rides. The server will broadcast the ride request to all "free" drivers, who will offer to pick up customers with a certain price, then the server will send those offers back to customer to choose whichever he likes. The server should also manage "ongoing" rides till they are finished.

The application should have a multithreading server where each client (either customer or driver) will be handled in a separate thread. The system will need to support both customer and driver clients, allowing for basic interactions such as requesting rides, offering rides, and receiving updates about the status of the ride.

---

## Server-Side Requirements:

- **Server Initialization**:
  - The server should listen for client connections on a given port (e.g., port 12345).
  - The server should differentiate between customer and driver clients. Each client should send an initial message indicating whether they are a customer or a driver.
- **Multithreading**:
  - For each customer and driver that connects, the server should spawn a new thread. The server should allow simultaneous handling of multiple customers and drivers.
- **Client Management**:
  - The server should maintain separate lists for customers and drivers. Each time a new customer or driver connects, the server should assign them a unique ID and store this information.
  - Drivers can send status updates, such as whether they started or ended the ride.

## Client-Side Requirements:

- **Client Initialization**:
    - The client should be able to connect to the server by providing the server's IP address and port number (default port: 12345).
    - The client should identify itself as either a customer or a driver upon connecting.
- **User Interaction**:
    - The client should present a simple text-based menu for the user to interact with the server. The menu will offer different options based on whether the client is a customer or a driver.
        1. **Customer Options**:
            1. Request a ride by entering a pickup location and destination.
            2. View the current status of the requested ride.
            3. Disconnect from the server.
        2. **Driver Options**:
            1. Offer a fare for a ride request.
            2. Send status updates of the ride they have been assigned to (start or finish ride).
            3. Disconnect from the server.
- **Multithreading**:
    - The client should be able to handle sending and receiving data concurrently. For example, it should be able to send disconnect request while waiting for the server’s responses.
- **Graceful Disconnect**:
    - The client should handle the server disconnecting and should exit the program gracefully when the user chooses to "disconnect."

---

## Feature list:

1. **Server-Client Communication:** Implement communication between the server and clients using Java SE sockets.

2. **User Authentication**:
    - login and registration. Existing users would need to login, while new users need to register first.
    - To login, the user needs to send his username and password.
    - To register, the user needs to send his type (customer or driver), username, and password.
    - There should be a special “Admin” user that is already pre-defined and can login.
3. **Ride Request**:

- Customers can request a ride by providing their location (e.g., pickup location, destination).
- The server should broadcast this ride requests to all free drivers. If no driver is available, the customer should receive a message saying no drivers are currently available.

4. **Driver Availability**:
   - Drivers can offer fares to rides, but can't send offers to 2 customers at the same time (one ride at a time).
5. **Ride Assignment**:
   - The server should notify both the customer and the driver once a ride has been successfully assigned (the customer accepted one of the offers).
6. **Ride Status Updates**:
   - Drivers should be able to update the status of a ride (e.g., when the ride starts, when the ride ends).
   - The server should inform the customer when the ride status changes (e.g., "Driver on the way," "Ride completed").
7. **Disconnect**:
   - When the customer or driver sends a "disconnect" message, the server should gracefully close the connection with that specific client. (can't do that while he is in an "ongoing" ride)
8. **Statistics**:
   - Provide the admin with the ability to view the overall status, in terms of all rides and status of them, total number of customers, and total number of drivers.

9. **Driver rating:**
   - **Add a rate driver feature from a customer** based on specific user defined review format including **various data**. **Calculate on the server (and display)**, the overall accumulated rating for that driver, as per your user-defined review format.

10. **Proper Exception/Error handling**:
    - Implement error handling mechanisms to deal with various scenarios, such as invalid user inputs.

---

## Additional Features (For students working in teams of 3):

1. When a driver rating goes below specific user defined conditions, give such driver a lower priority in getting new rides. The priority should increase upon improvement in the drivers' rating.

2. Handle driver location updates in the period between matching the ride and starting it. (“6 minutes away”,,, “3 minutes away” and the server updating the customer with those updates)

3. **Database/Files Dump and load**: Provide the admin to save all application data (rides, customers, drivers … etc.) to persistence (whether its DB or Files), and when the application boots up again, the server loads the data from persistence if it was stored in the first place.

---

## Notes:

1. No graphical user interface is needed. The whole assignment can be run through the command line

2. Test the application by running at least 2 (or more) client customers and 2 (or more) client drivers and 1 admin.

---

## Submission:

- The project must be developed in the Java SE programming language only.
- Provide a brief explanation of how to run the server and client and the used java JDK.
- The assignment will be solved in a group of 2 or 3 students from the same lab or with the same TA.
- In case you’re forming a group of 3, then you **MUST** implement the additional features.
- If teams of 3 members submit only the features for a team of two, all team members will get **ZERO**.
- If more than 3 team members submit the assignment, all team members will get **ZERO**.
- You should submit your assignment as ONE zip file with the below naming convention:

  Assign1_GroupNumber_ID1_ID2_TAName

  (example: Assign1_S1_20116001_20116002_TA.Hassan)

- You SHOULD NOT copy any code from the internet or from your colleagues or generate code using AI. It will be detected and considered as a cheating case and get a minus grade

- Submission of the assignment will be on Google Classroom through a Google form link that will be shared later. **Deadline for the submission is Friday 28th of March 2025.**