CSE222-DATA STRUCTURES AND ALGORITHMS

HW8 REPORT

ADEM MURAT ÖZDEMİR-2001004004110

I created two main classes named for “Main.java” and “TestMain.java”
The Main.java pdf was adjusted according to the desired output layout in the pdf you shared.
TestMain.java was prepared for the convenience of the demo (TimeStamp is not used in this).
In order to avoid the hassle of using TimeStamp for the user, some functions (addFriendShip(), findShortestPath(), suggestFriends() methods) do not take Timestamp as an input in TestMain.java .

Compiling and Running

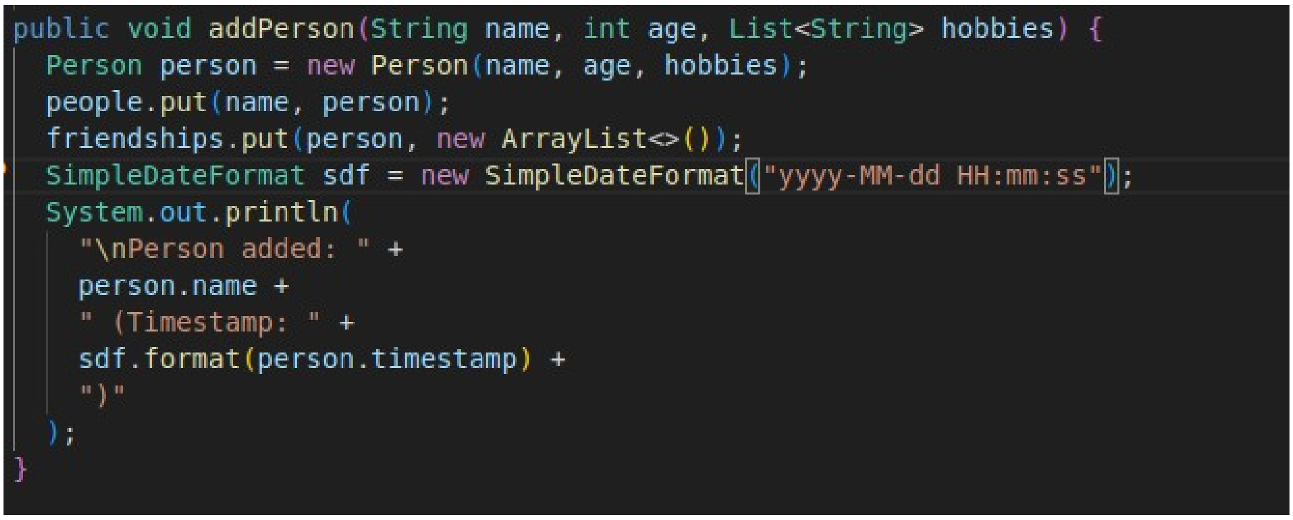
Compile For Main.java : “make”
For Running : “make run”
Clean the .class files : “make clean”
Create Javadoc document : “make doc”
Clean the Javadoc : “make cleandoc”

For TestMain.java :

```
javac TestMain.java
java TestMain
```

SocialNetworkGraph Class Explanations

addPerson:

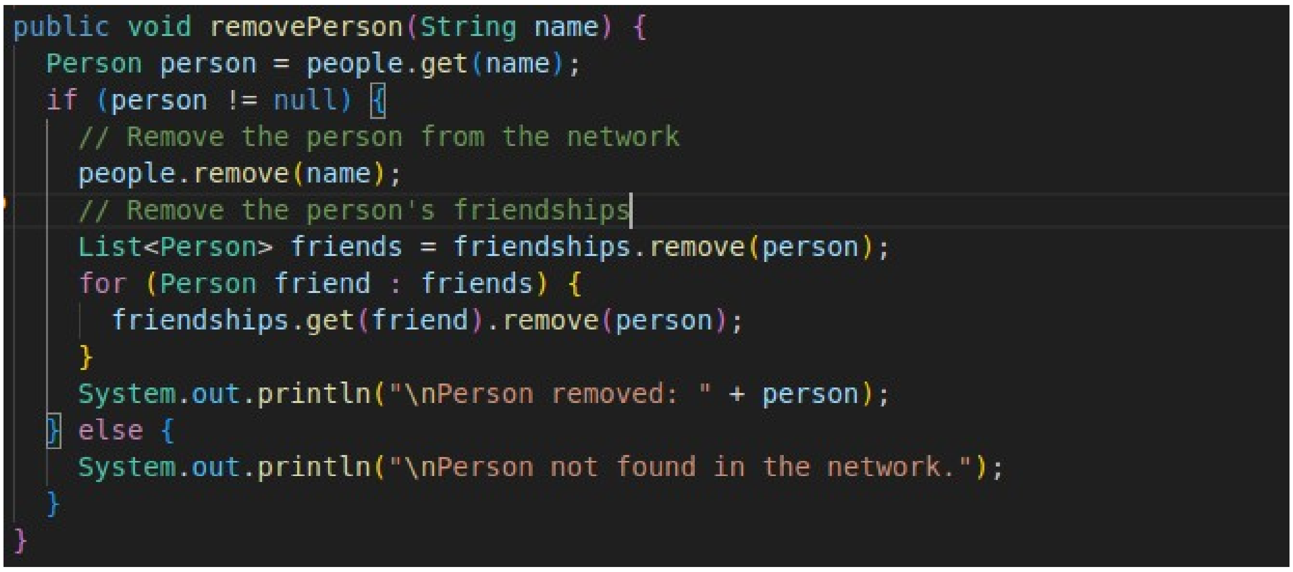


```java
public void addPerson(String name, int age, List<String> hobbies) {
  Person person = new Person(name, age, hobbies);
  people.put(name, person);
  friendships.put(person, new ArrayList<>());
  SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
  System.out.println(
    "\nPerson added: " +
    person.name +
    " (Timestamp: " +
    sdf.format(person.timestamp) +
    ")"
  );
}
```

This function, addPerson, takes a name, age, and list of hobbies as input parameters. It performs the following steps:

1. Creates a new Person object with the given name, age, and hobbies.
2. Adds this new person to the people map with the name as the key.
3. Initializes an empty friendship list for the new person in the friendships map.
4. Prints a confirmation message indicating that the person has been added, including the person's name and timestamp of when they were added to the network.

removePerson :

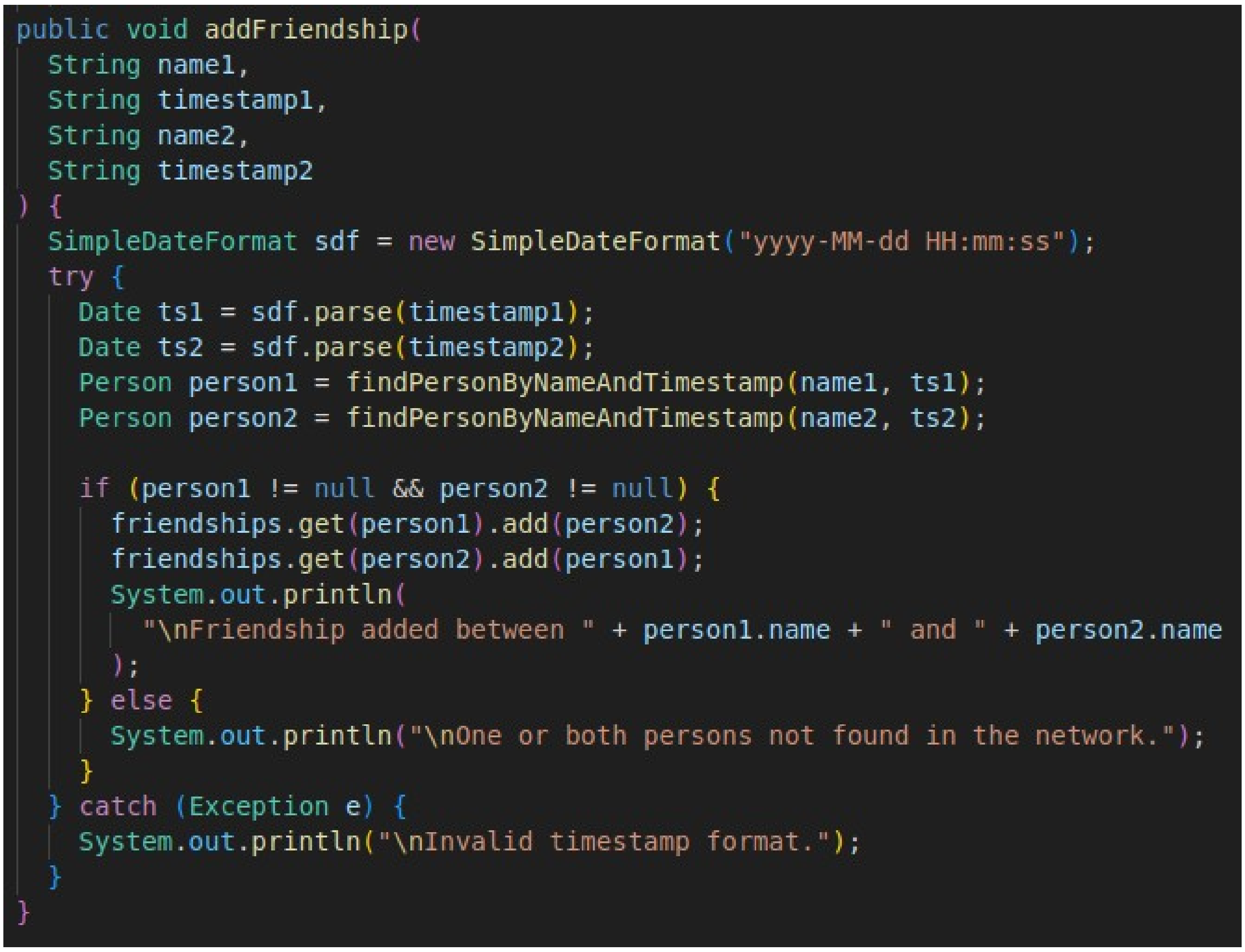


```java
public void removePerson(String name) {
  Person person = people.get(name);
  if (person != null) {
    // Remove the person from the network
    people.remove(name);
    // Remove the person's friendships
    List<Person> friends = friendships.remove(person);
    for (Person friend : friends) {
      friendships.get(friend).remove(person);
    }
    System.out.println("\nPerson removed: " + person);
  } else {
    System.out.println("\nPerson not found in the network.");
  }
}
```

This function, removePerson, takes a name as an input parameter. It performs the following steps:

1. Retrieves the Person object corresponding to the given name from the people map.
2. Checks if the person exists in the network.
3. If the person exists:
    - Removes the person from the people map.
    - Removes the person's friendships from the friendships map.
    - Iterates through the person's friends and removes the person from each friend's friendship list.
    - Prints a confirmation message indicating that the person has been removed.
4. If the person is not found in the network, it prints an error message stating that the person was not found.

addFriendship :

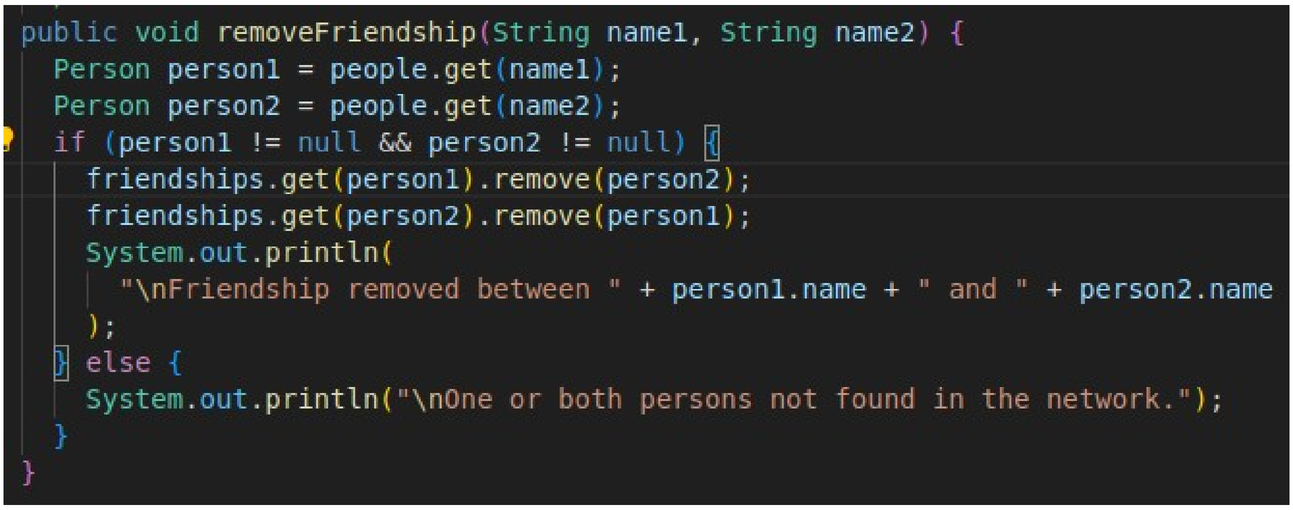


```java
public void addFriendship(
  String name1,
  String timestamp1,
  String name2,
  String timestamp2
) {
  SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
  try {
    Date ts1 = sdf.parse(timestamp1);
    Date ts2 = sdf.parse(timestamp2);
    Person person1 = findPersonByNameAndTimestamp(name1, ts1);
    Person person2 = findPersonByNameAndTimestamp(name2, ts2);

    if (person1 != null && person2 != null) {
      friendships.get(person1).add(person2);
      friendships.get(person2).add(person1);
      System.out.println(
        "\nFriendship added between " + person1.name + " and " + person2.name
      );
    } else {
      System.out.println("\nOne or both persons not found in the network.");
    }
  } catch (Exception e) {
    System.out.println("\nInvalid timestamp format.");
  }
}
```

This function, addFriendship, takes two names and their corresponding timestamps as input parameters. It performs the following steps:

1. Parses the timestamps from string format to Date objects using SimpleDateFormat.
2. Finds the Person objects corresponding to the given names and timestamps using the findPersonByNameAndTimestamp method.
3. Checks if both persons exist in the network.
4. If both persons exist:
    - Adds each person to the other's friendship list, creating a bidirectional friendship.
    - Prints a confirmation message indicating that the friendship has been added.
5. If one or both persons are not found in the network, it prints an error message stating that one or both persons were not found.
6. If there is an issue with parsing the timestamps, it prints an error message indicating an invalid timestamp format.

removeFriendship :

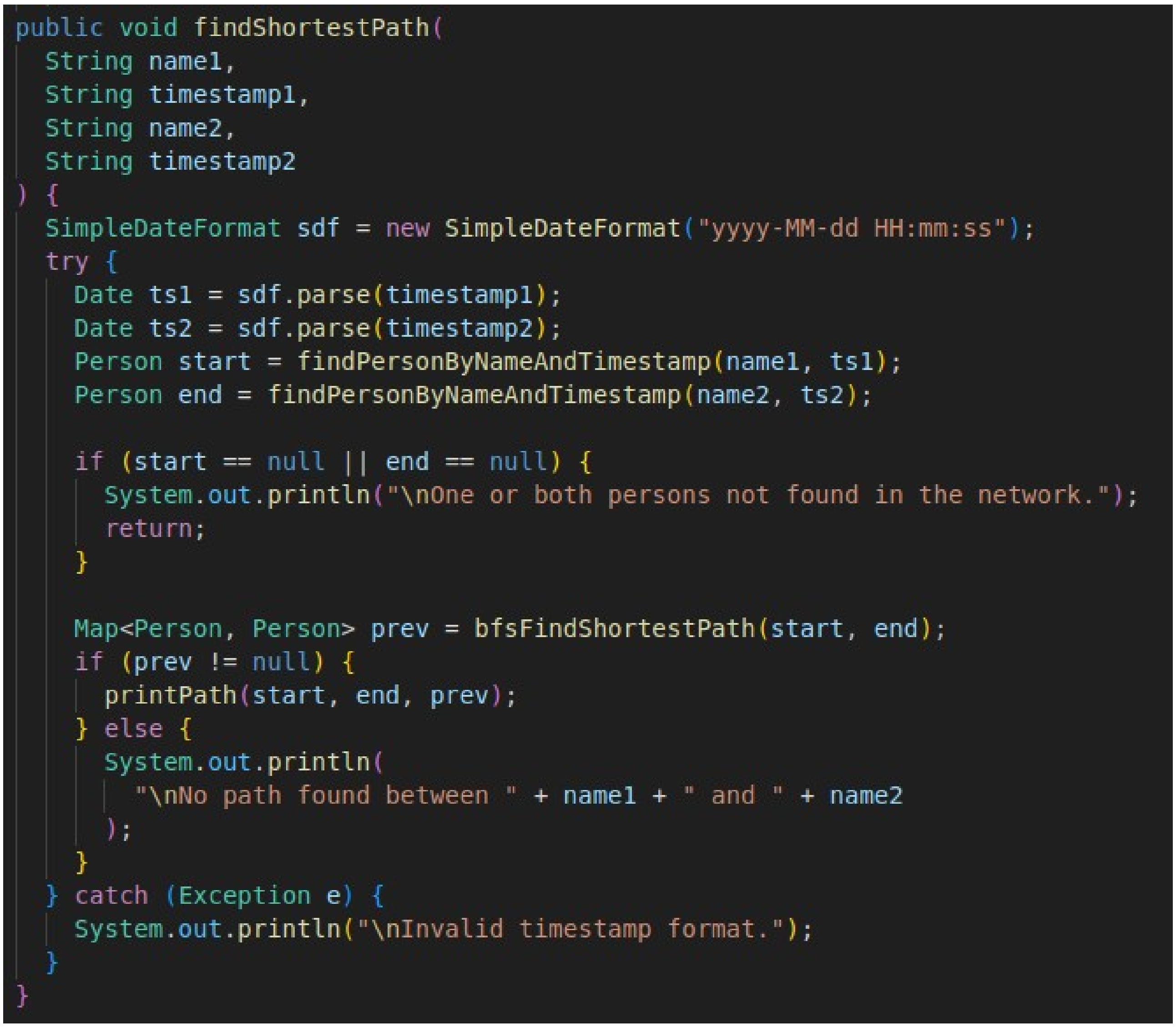


```java
public void removeFriendship(String name1, String name2) {
  Person person1 = people.get(name1);
  Person person2 = people.get(name2);
  if (person1 != null && person2 != null) {
    friendships.get(person1).remove(person2);
    friendships.get(person2).remove(person1);
    System.out.println(
      "\nFriendship removed between " + person1.name + " and " + person2.name
    );
  } else {
    System.out.println("\nOne or both persons not found in the network.");
  }
}
```

This function, removeFriendship, takes two names as input parameters. It performs the following steps:

1. Retrieves the Person objects corresponding to the given names from the people map.
2. Checks if both persons exist in the network.
3. If both persons exist:
    - Removes each person from the other's friendship list, effectively deleting the bidirectional friendship.
    - Prints a confirmation message indicating that the friendship has been removed.
4. If one or both persons are not found in the network, it prints an error message stating that one or both persons were not found.

findShortestPath :



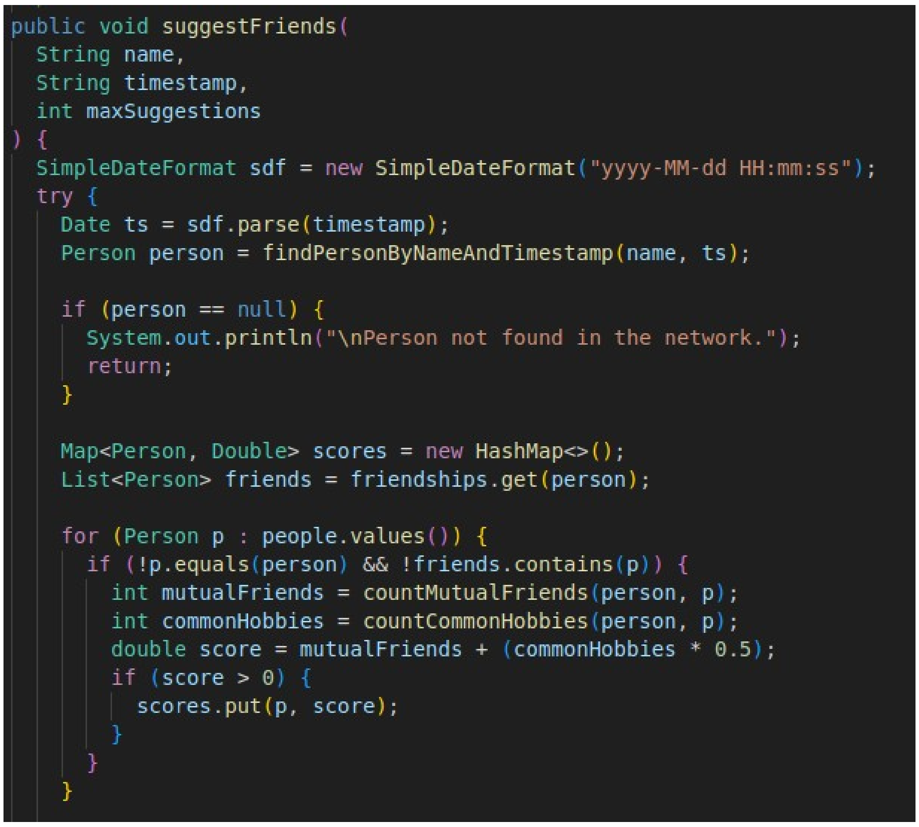
```java
public void findShortestPath(
  String name1,
  String timestamp1,
  String name2,
  String timestamp2
) {
  SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
  try {
    Date ts1 = sdf.parse(timestamp1);
    Date ts2 = sdf.parse(timestamp2);
    Person start = findPersonByNameAndTimestamp(name1, ts1);
    Person end = findPersonByNameAndTimestamp(name2, ts2);

    if (start == null || end == null) {
      System.out.println("\nOne or both persons not found in the network.");
      return;
    }

    Map<Person, Person> prev = bfsFindShortestPath(start, end);
    if (prev != null) {
      printPath(start, end, prev);
    } else {
      System.out.println(
        "\nNo path found between " + name1 + " and " + name2
      );
    }
  } catch (Exception e) {
    System.out.println("\nInvalid timestamp format.");
  }
}
```

This function, findShortestPath, takes two names and their corresponding timestamps as input parameters. It performs the following steps:

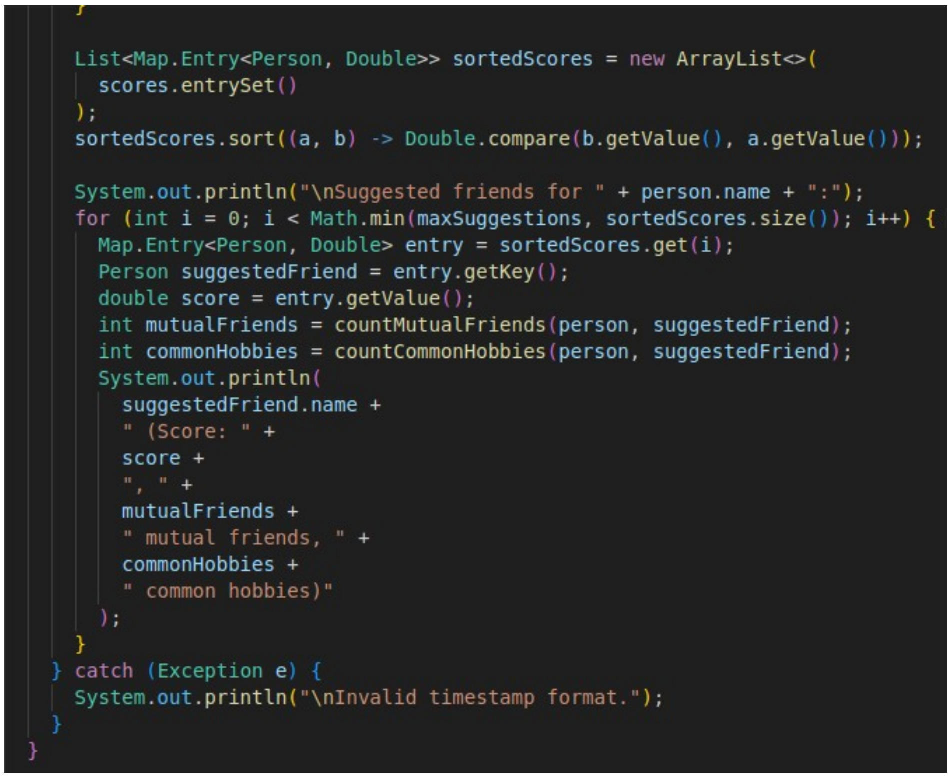
1. Parses the timestamps from string format to Date objects using SimpleDateFormat.
2. Finds the Person objects corresponding to the given names and timestamps using the findPersonByNameAndTimestamp method.
3. Checks if both persons exist in the network.
4. If one or both persons are not found, it prints an error message and exits the function.
5. Uses the bfsFindShortestPath method to perform a Breadth-First Search (BFS) to find the shortest path between the two persons.
6. If a path is found, it calls the printPath method to print the path.
7. If no path is found, it prints a message indicating that no path exists between the given persons.

8. If there is an issue with parsing the timestamps, it prints an error message indicating an invalid timestamp format.

suggestFriends :



```java
public void suggestFriends(
  String name,
  String timestamp,
  int maxSuggestions
) {
  SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
  try {
    Date ts = sdf.parse(timestamp);
    Person person = findPersonByNameAndTimestamp(name, ts);

    if (person == null) {
      System.out.println("\nPerson not found in the network.");
      return;
    }

    Map<Person, Double> scores = new HashMap<>();
    List<Person> friends = friendships.get(person);

    for (Person p : people.values()) {
      if (!p.equals(person) && !friends.contains(p)) {
        int mutualFriends = countMutualFriends(person, p);
        int commonHobbies = countCommonHobbies(person, p);
        double score = mutualFriends + (commonHobbies * 0.5);
        if (score > 0) {
          scores.put(p, score);
        }
      }
    }
```

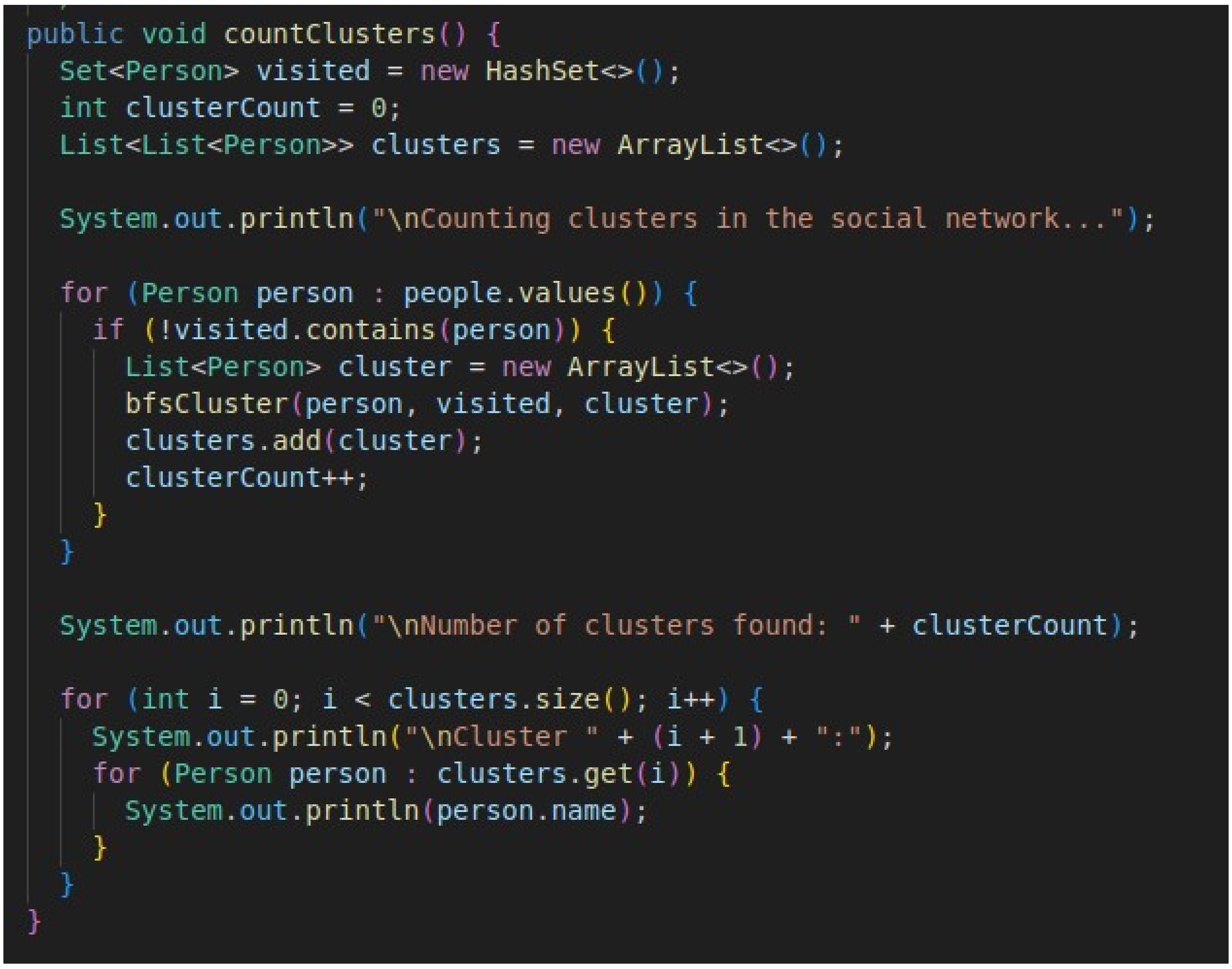


```java
    List<Map.Entry<Person, Double>> sortedScores = new ArrayList<>(
      scores.entrySet()
    );
    sortedScores.sort((a, b) -> Double.compare(b.getValue(), a.getValue()));

    System.out.println("\nSuggested friends for " + person.name + ":");
    for (int i = 0; i < Math.min(maxSuggestions, sortedScores.size()); i++) {
      Map.Entry<Person, Double> entry = sortedScores.get(i);
      Person suggestedFriend = entry.getKey();
      double score = entry.getValue();
      int mutualFriends = countMutualFriends(person, suggestedFriend);
      int commonHobbies = countCommonHobbies(person, suggestedFriend);
      System.out.println(
        suggestedFriend.name +
        " (Score: " +
        score +
        ", " +
        mutualFriends +
        " mutual friends, " +
        commonHobbies +
        " common hobbies)"
      );
    }
  } catch (Exception e) {
    System.out.println("\nInvalid timestamp format.");
  }
}
```

This function, suggestFriends, takes a name, a timestamp, and the maximum number of suggestions as input parameters. It performs the following steps:

1. Parses the timestamp from string format to a Date object using SimpleDateFormat.
2. Finds the Person object corresponding to the given name and timestamp using the findPersonByNameAndTimestamp method.
3. Checks if the person exists in the network.
4. If the person is not found, it prints an error message and exits the function.
5. Calculates scores for potential friends based on mutual friends and common hobbies.
6. Sorts the potential friends by their scores in descending order.
7. Prints the top suggested friends along with their scores, mutual friends, and common hobbies.
8. If there is an issue with parsing the timestamp, it prints an error message indicating an invalid timestamp format.

countClusters :

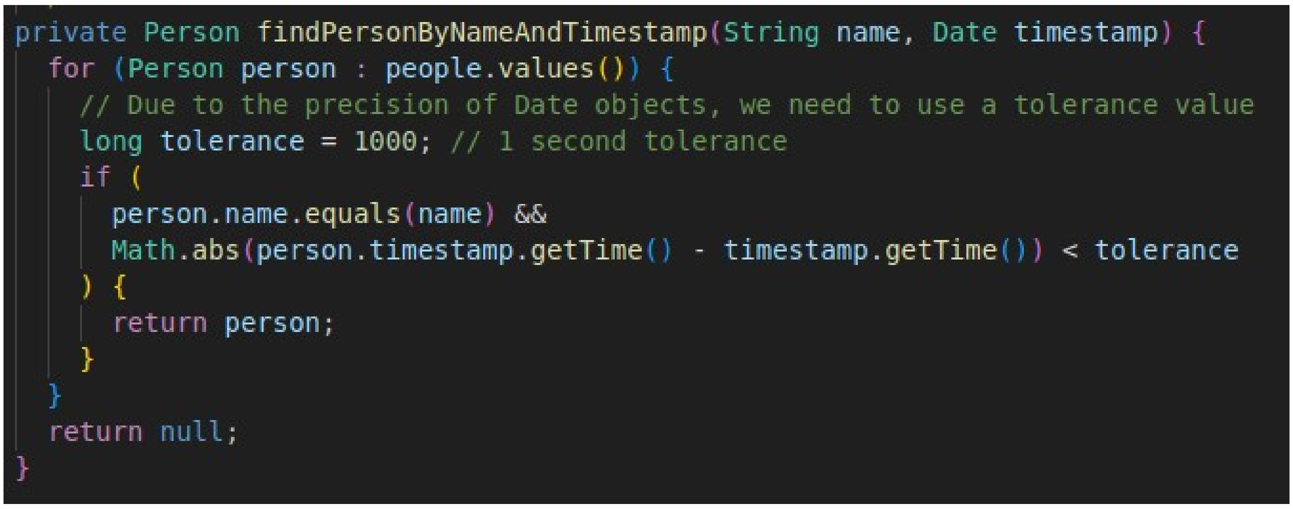


```java
public void countClusters() {
  Set<Person> visited = new HashSet<>();
  int clusterCount = 0;
  List<List<Person>> clusters = new ArrayList<>();

  System.out.println("\nCounting clusters in the social network...");

  for (Person person : people.values()) {
    if (!visited.contains(person)) {
      List<Person> cluster = new ArrayList<>();
      bfsCluster(person, visited, cluster);
      clusters.add(cluster);
      clusterCount++;
    }
  }

  System.out.println("\nNumber of clusters found: " + clusterCount);

  for (int i = 0; i < clusters.size(); i++) {
    System.out.println("\nCluster " + (i + 1) + ":");
    for (Person person : clusters.get(i)) {
      System.out.println(person.name);
    }
  }
}
```

This function, countClusters, identifies and counts clusters in the social network. It performs the following steps:

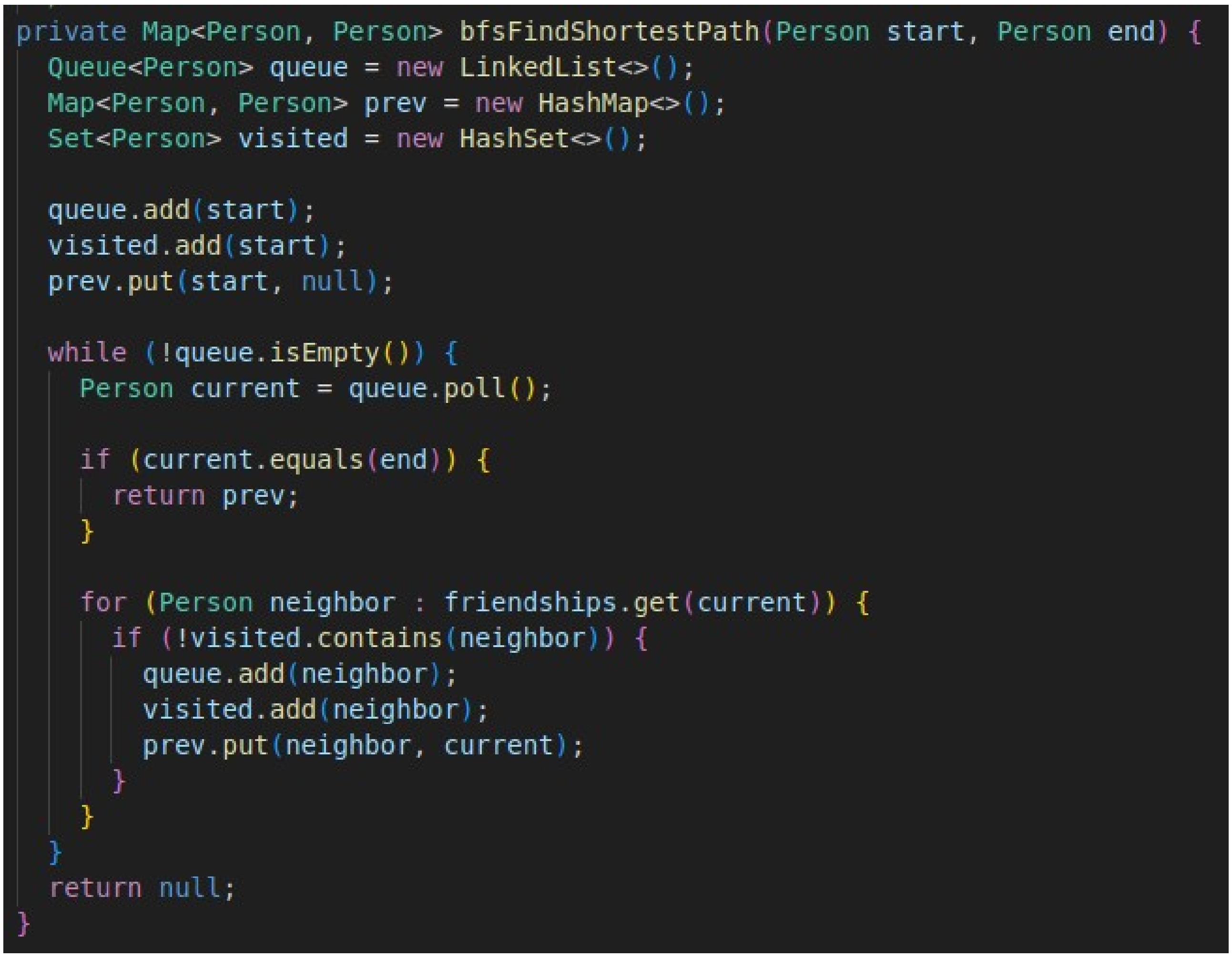
1. Initializes a set to keep track of visited persons and a list to store the clusters.
2. Prints a message indicating that it is counting clusters.
3. Iterates through all persons in the network.
4. For each person not already visited:
    - Initializes a new cluster.
    - Uses the bfsCluster method to perform a Breadth-First Search (BFS) to find all connected persons, adding them to the cluster.
    - Adds the cluster to the list of clusters and increments the cluster count.
5. Prints the total number of clusters found.
6. Iterates through the list of clusters and prints the members of each cluster, one cluster at a time.

Helper Methods :



```java
private Person findPersonByNameAndTimestamp(String name, Date timestamp) {
  for (Person person : people.values()) {
    // Due to the precision of Date objects, we need to use a tolerance value
    long tolerance = 1000; // 1 second tolerance
    if (
      person.name.equals(name) &&
      Math.abs(person.timestamp.getTime() - timestamp.getTime()) < tolerance
    ) {
      return person;
    }
  }
  return null;
}
```

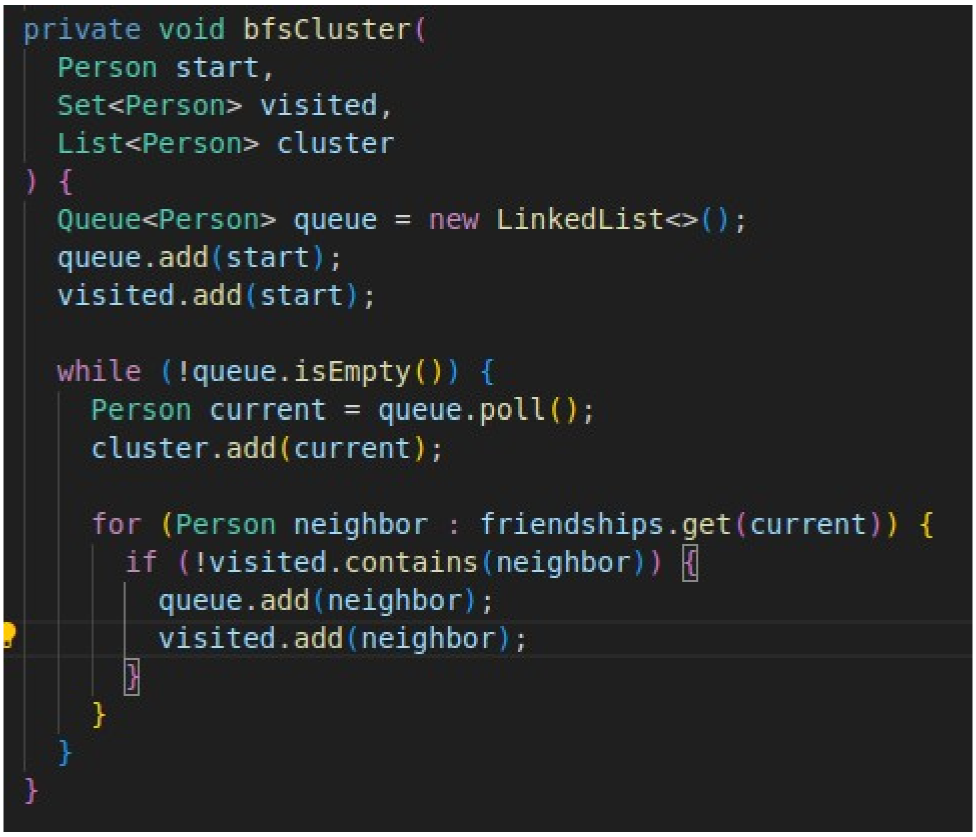
This function, findPersonByNameAndTimestamp, searches for a person in the network by their name and timestamp with a 1-second tolerance. If a matching person is found, it returns the Person object; otherwise, it returns null.



```java
private Map<Person, Person> bfsFindShortestPath(Person start, Person end) {
  Queue<Person> queue = new LinkedList<>();
  Map<Person, Person> prev = new HashMap<>();
  Set<Person> visited = new HashSet<>();

  queue.add(start);
  visited.add(start);
  prev.put(start, null);

  while (!queue.isEmpty()) {
    Person current = queue.poll();

    if (current.equals(end)) {
      return prev;
    }

    for (Person neighbor : friendships.get(current)) {
      if (!visited.contains(neighbor)) {
        queue.add(neighbor);
        visited.add(neighbor);
        prev.put(neighbor, current);
      }
    }
  }
  return null;
}
```

This function, bfsFindShortestPath, uses Breadth-First Search (BFS) to find the shortest path between two persons in the network. It performs the following steps:

1. Initializes a queue, a map to track previous nodes, and a set to track visited nodes.
2. Adds the starting person to the queue and marks them as visited.
3. While the queue is not empty:
    - Dequeues the current person.
    - If the current person is the end person, returns the map of previous nodes.
    - Enqueues all unvisited neighbors of the current person, marks them as visited, and records the current person as their predecessor.
4. If no path is found, returns null.

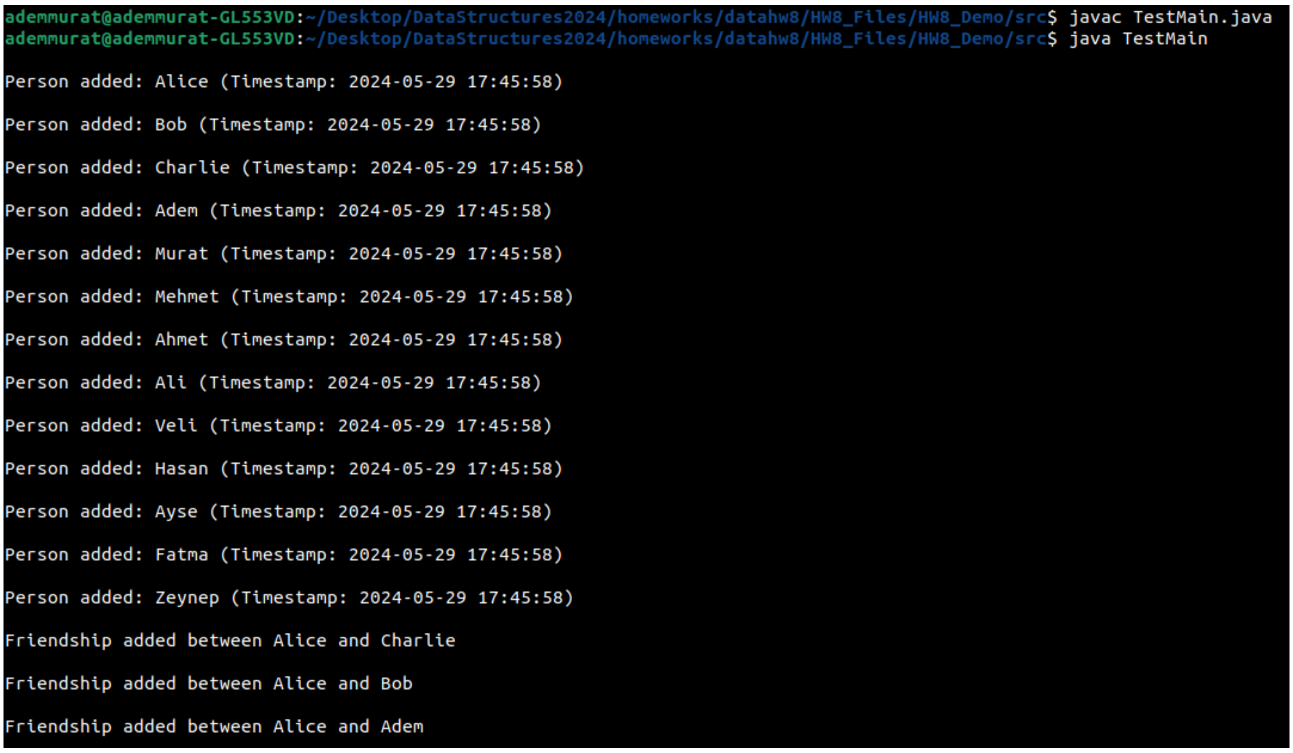


```java
private void bfsCluster(
  Person start,
  Set<Person> visited,
  List<Person> cluster
) {
  Queue<Person> queue = new LinkedList<>();
  queue.add(start);
  visited.add(start);

  while (!queue.isEmpty()) {
    Person current = queue.poll();
    cluster.add(current);

    for (Person neighbor : friendships.get(current)) {
      if (!visited.contains(neighbor)) {
        queue.add(neighbor);
        visited.add(neighbor);
      }
    }
  }
}
```

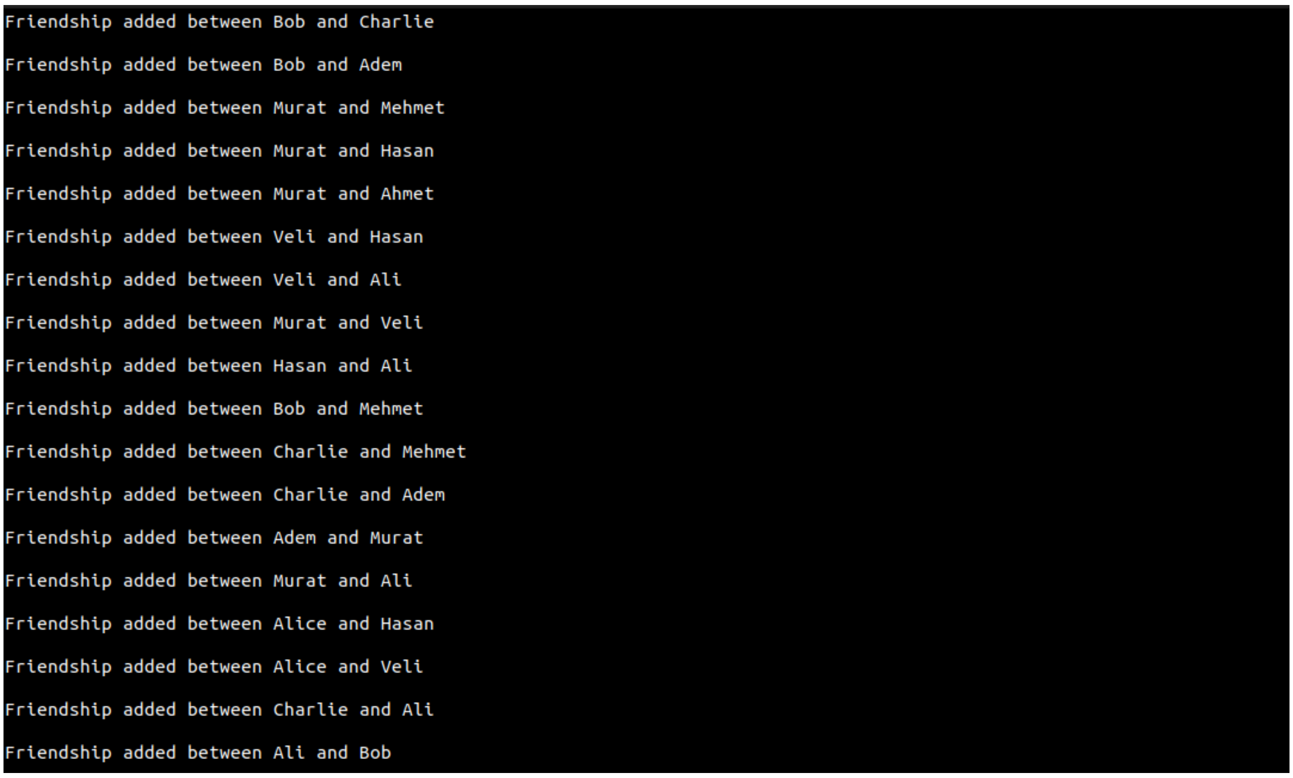
This function, bfsCluster, uses Breadth-First Search (BFS) to explore and identify all persons in a cluster starting from a given person. It performs the following steps:

1. Initializes a queue and adds the starting person to it.
2. Marks the starting person as visited.
3. While the queue is not empty:
    - Dequeues the current person.
    - Adds the current person to the cluster list.
    - Enqueues all unvisited neighbors of the current person and marks them as visited.

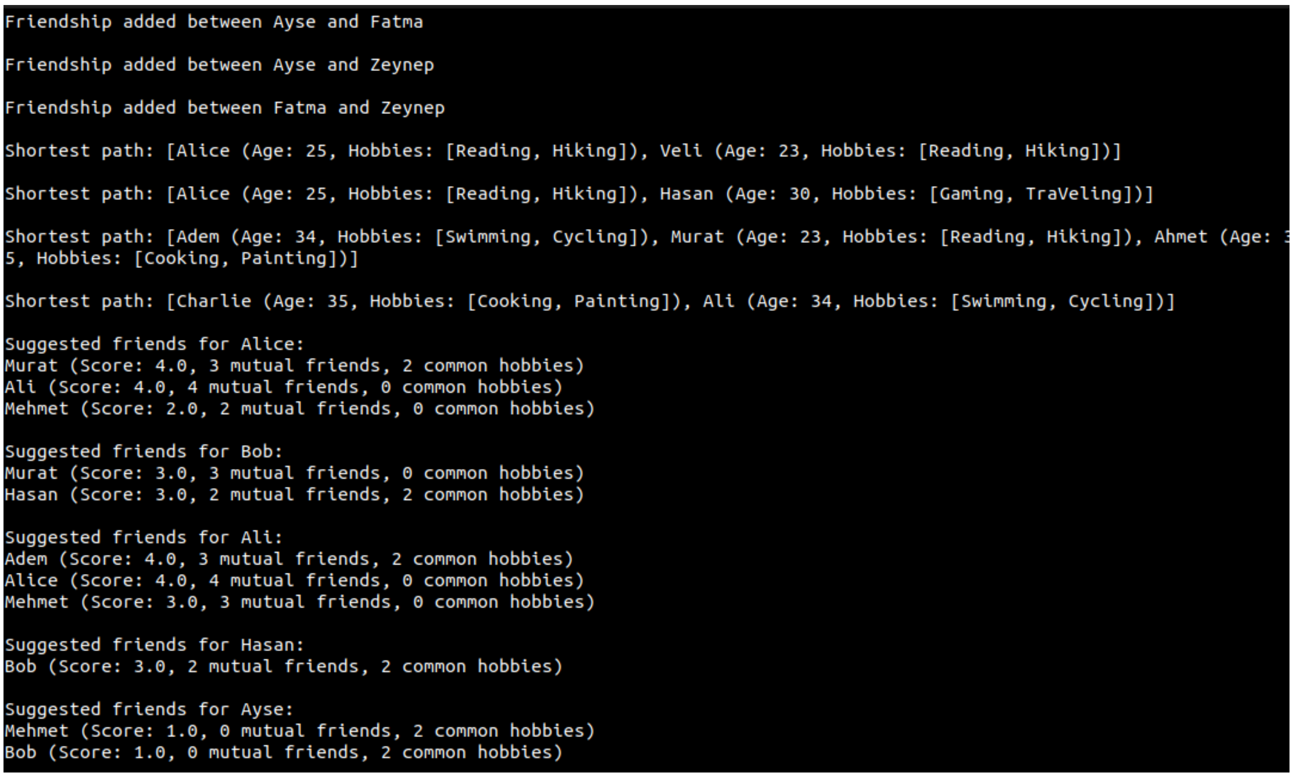
Test Results :   (I used TestMain.java for results)



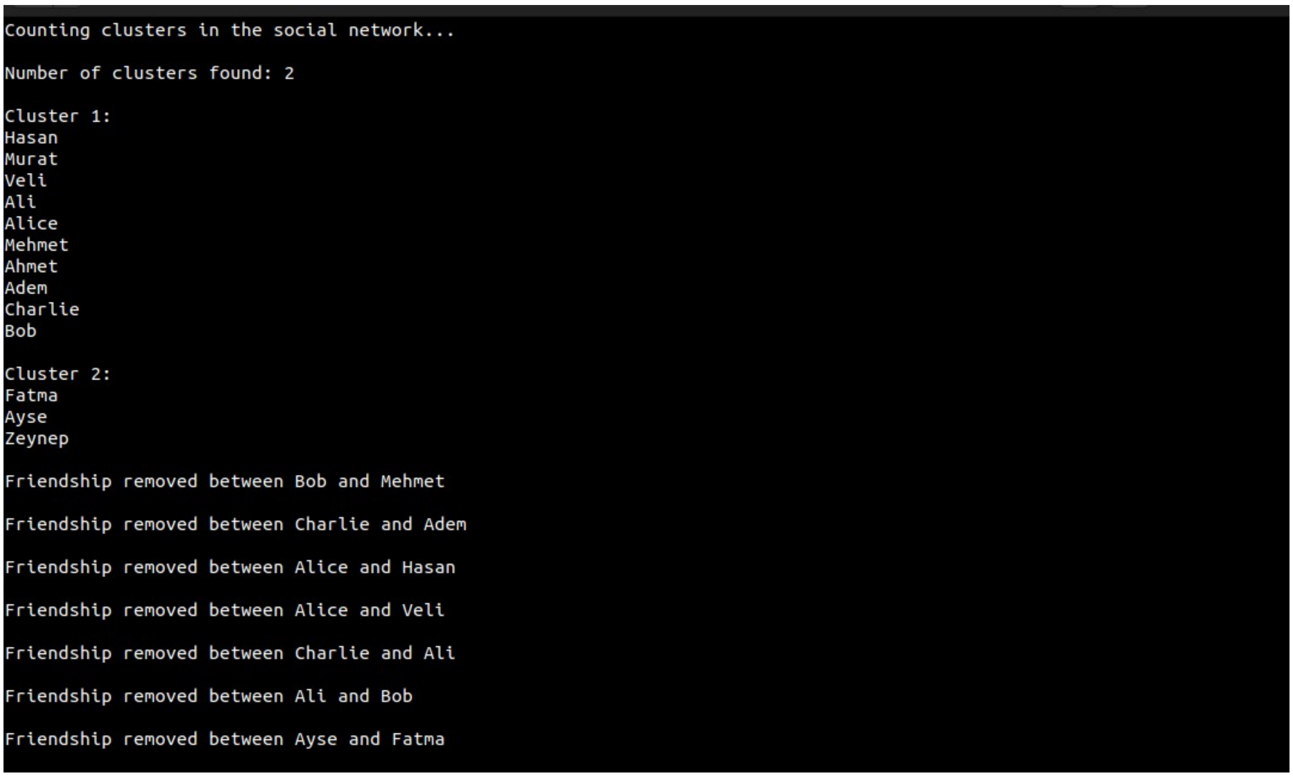
```
ademmurat@ademmurat-GL553VD:~/Desktop/DataStructures2024/homeworks/datahw8/HW8_Files/HW8_Demo/src$ javac TestMain.java
ademmurat@ademmurat-GL553VD:~/Desktop/DataStructures2024/homeworks/datahw8/HW8_Files/HW8_Demo/src$ java TestMain

Person added: Alice (Timestamp: 2024-05-29 17:45:58)

Person added: Bob (Timestamp: 2024-05-29 17:45:58)

Person added: Charlie (Timestamp: 2024-05-29 17:45:58)

Person added: Adem (Timestamp: 2024-05-29 17:45:58)

Person added: Murat (Timestamp: 2024-05-29 17:45:58)

Person added: Mehmet (Timestamp: 2024-05-29 17:45:58)

Person added: Ahmet (Timestamp: 2024-05-29 17:45:58)

Person added: Ali (Timestamp: 2024-05-29 17:45:58)

Person added: Veli (Timestamp: 2024-05-29 17:45:58)

Person added: Hasan (Timestamp: 2024-05-29 17:45:58)

Person added: Ayse (Timestamp: 2024-05-29 17:45:58)

Person added: Fatma (Timestamp: 2024-05-29 17:45:58)

Person added: Zeynep (Timestamp: 2024-05-29 17:45:58)

Friendship added between Alice and Charlie

Friendship added between Alice and Bob

Friendship added between Alice and Adem
```



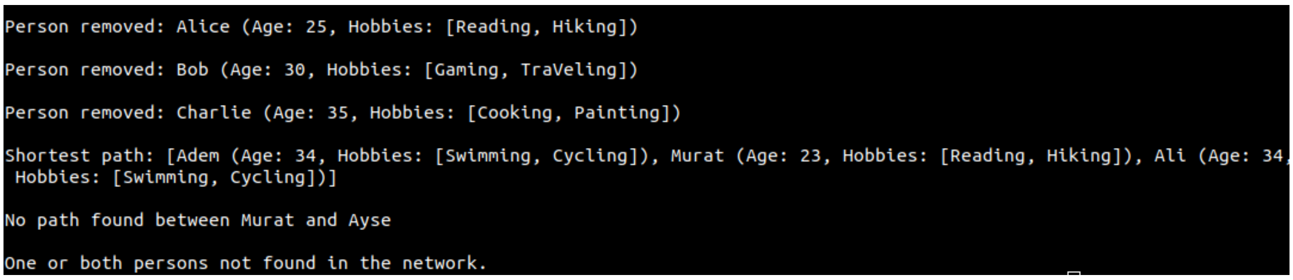
```
Friendship added between Bob and Charlie

Friendship added between Bob and Adem

Friendship added between Murat and Mehmet

Friendship added between Murat and Hasan

Friendship added between Murat and Ahmet

Friendship added between Veli and Hasan

Friendship added between Veli and Ali

Friendship added between Murat and Veli

Friendship added between Hasan and Ali

Friendship added between Bob and Mehmet

Friendship added between Charlie and Mehmet

Friendship added between Charlie and Adem

Friendship added between Adem and Murat

Friendship added between Murat and Ali

Friendship added between Alice and Hasan

Friendship added between Alice and Veli

Friendship added between Charlie and Ali

Friendship added between Ali and Bob
```



```
Friendship added between Ayse and Fatma

Friendship added between Ayse and Zeynep

Friendship added between Fatma and Zeynep

Shortest path: [Alice (Age: 25, Hobbies: [Reading, Hiking]), Veli (Age: 23, Hobbies: [Reading, Hiking])]

Shortest path: [Alice (Age: 25, Hobbies: [Reading, Hiking]), Hasan (Age: 30, Hobbies: [Gaming, TraVeling])]

Shortest path: [Adem (Age: 34, Hobbies: [Swimming, Cycling]), Murat (Age: 23, Hobbies: [Reading, Hiking]), Ahmet (Age: 3
5, Hobbies: [Cooking, Painting])]

Shortest path: [Charlie (Age: 35, Hobbies: [Cooking, Painting]), Ali (Age: 34, Hobbies: [Swimming, Cycling])]

Suggested friends for Alice:
Murat (Score: 4.0, 3 mutual friends, 2 common hobbies)
Ali (Score: 4.0, 4 mutual friends, 0 common hobbies)
Mehmet (Score: 2.0, 2 mutual friends, 0 common hobbies)

Suggested friends for Bob:
Murat (Score: 3.0, 3 mutual friends, 0 common hobbies)
Hasan (Score: 3.0, 2 mutual friends, 2 common hobbies)

Suggested friends for Ali:
Adem (Score: 4.0, 3 mutual friends, 2 common hobbies)
Alice (Score: 4.0, 4 mutual friends, 0 common hobbies)
Mehmet (Score: 3.0, 3 mutual friends, 0 common hobbies)

Suggested friends for Hasan:
Bob (Score: 3.0, 2 mutual friends, 2 common hobbies)

Suggested friends for Ayse:
Mehmet (Score: 1.0, 0 mutual friends, 2 common hobbies)
Bob (Score: 1.0, 0 mutual friends, 2 common hobbies)
```



```
Counting clusters in the social network...

Number of clusters found: 2

Cluster 1:
Hasan
Murat
Veli
Ali
Alice
Mehmet
Ahmet
Adem
Charlie
Bob

Cluster 2:
Fatma
Ayse
Zeynep

Friendship removed between Bob and Mehmet

Friendship removed between Charlie and Adem

Friendship removed between Alice and Hasan

Friendship removed between Alice and Veli

Friendship removed between Charlie and Ali

Friendship removed between Ali and Bob

Friendship removed between Ayse and Fatma
```



```
Person removed: Alice (Age: 25, Hobbies: [Reading, Hiking])

Person removed: Bob (Age: 30, Hobbies: [Gaming, TraVeling])

Person removed: Charlie (Age: 35, Hobbies: [Cooking, Painting])

Shortest path: [Adem (Age: 34, Hobbies: [Swimming, Cycling]), Murat (Age: 23, Hobbies: [Reading, Hiking]), Ali (Age: 34,
 Hobbies: [Swimming, Cycling])]

No path found between Murat and Ayse

One or both persons not found in the network.
```