```java
import lt.infobalt.itakademija.javalang.exam.socialnetwork.Friend;
import lt.infobalt.itakademija.javalang.exam.socialnetwork.FriendNotFoundException;
import lt.infobalt.itakademija.javalang.exam.socialnetwork.SocialNetwork;

import java.util.*;
import java.util.stream.Collectors;

public class SocialNetworkImpl implements SocialNetwork {

    LinkedList<Friend> friends;

    public SocialNetworkImpl() {
        this.friends = new LinkedList<>();
    }

    @Override
    public void addFriend(Friend friend) {
        if (friend == null || friend.getFirstName() == null && friend.getLastName() == null) { // patikrinimas ar draugas turi duomenis  + varda, pavarde
            throw new IllegalArgumentException();
        }

        if (!friends.contains(friend)) { //patikrinimas ar tokio draugo sarase dar nera/yra
            friends.add(friend);
        }
    }

    @Override
    public int getNumberOfFriends() {
        int count = 0;
        count = friends.size();
        return count;
    }

```

```
34      @Override
35      public Friend findFriend(String name, String surname) throws FriendNotFoundException {
36          if (name == null || surname == null) { //patikinimas ar yra duomenys
37              throw new IllegalArgumentException();
38          }
39
40          for (Friend friend : friends) { //patikrinimas palyginant vardus ir pavardes
41              if (friend.getFirstName().equals(name) && friend.getLastName().equals(surname)) {
42                  return friend;
43              }
44          }
45          throw new FriendNotFoundException(name, surname);
46      }
47
48      @Override
49      public Collection<Friend> findByCity(String city) {
50          if (city == null) { //patikrinimas ar miestas turi duomenis
51              throw new IllegalArgumentException();
52          }
53          LinkedList<Friend> friendsByCity = new LinkedList<>(); //sukuriam dar viena lista i kuri desim
    atrinktus draugus,
54          // o po to si lista returninsim (nes metodas praso grazinti draugus sarasa)
55
56          for (Friend friend : friends) {
57              if (friend.getCity().equals(city)) {
58                  friendsByCity.add(friend);
59              }
60          }
61          return friendsByCity;
62      }
63
64      @Override
65      public Collection<Friend> getOrderedFriends() {
66          return friends.stream()
```

```
67                  .sorted(Comparator.comparing(Friend::getCity)
68                          .thenComparing(Friend::getLastName)
69                          .thenComparing(Friend::getFirstName)).collect(Collectors.toList());
70          //alternatyva:
71  //          Comparator<Friend> sortFriends = Comparator.comparing(Friend::getCity)
72  //                  .thenComparing(Friend::getLastName)
73  //                  .thenComparing(Friend::getFirstName);
74  //          Collections.sort(friends, sortFriends);
75  //          return friends;
76      }
77  }
```