Homework #0: A Friendship Graph

Due Tuesday, January 21st at 11:59 p.m.

The goals of this assignment are to familiarize you with our course infrastructure and let you practice Java object-oriented programming. To complete this homework you will implement a simple graph class that could represent a network of friends.

Your learning goals for this assignment are to:

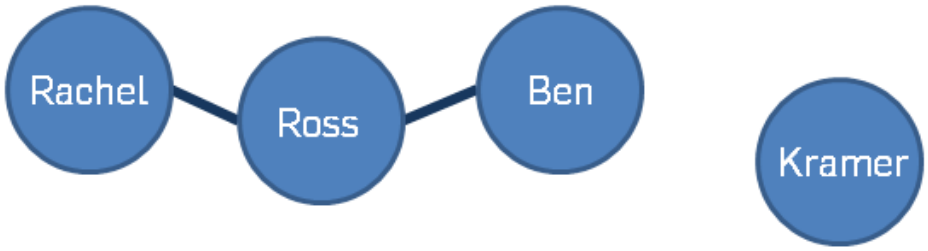
- Use Git and GitHub to revision and share work
- Get familiar with Java development environments
- Write a first Java program
- Practice Java style and coding conventions

Instructions

To begin your work, import the project into Eclipse from the `homework/0` directory in your repository.

Implement and test a `FriendGraph` class that represents friendships in a social network and can compute the distance between two people in the graph. You should model the social network as an undirected graph where each person is connected to zero or more people.

For example, suppose you have the following social network:



Your solution must work with the following client implementation (located in `Main.java`).

```
FriendGraph graph = new FriendGraph();
Person rachel = new Person("Rachel");
Person ross = new Person("Ross");
Person ben = new Person("Ben");
Person kramer = new Person("Kramer");
graph.addPerson(rachel);
graph.addPerson(ross);
graph.addPerson(ben);
graph.addPerson(kramer);
graph.addFriendship("Rachel", "Ross");
graph.addFriendship("Ross", "Ben");
System.out.println(graph.getDistance("Rachel", "Ross")); // should print 1
System.out.println(graph.getDistance("Rachel", "Ben")); // should print 2
System.out.println(graph.getDistance("Rachel", "Rachel")); // should print 0
System.out.println(graph.getDistance("Rachel", "Kramer")); // should print -1
```

Evaluation

Overall this homework is worth 50 points. To earn full credit you must follow these requirements:

- Your graph should have a `getDistance` method which takes two names (as `String`s) as arguments and returns the shortest distance (an `int`) between the people with those names, or `-1` if the two people are not connected.
- Do not use any libraries, including `java.util.*`.
- Use proper access modifiers (`public`, `private`, etc.) for your fields and methods[1]. If a field/method can be private, it should be private.
- Do not use any `static` fields or methods except for the `main` method.
- Follow the Java code conventions, especially for naming and commenting. Hint: use **Ctrl + Shift + F** to auto-format your code!
- Add short descriptive comments (`/** ... */`) to all `public` methods.

[1] Bloch, Joshua (2008). Item 13: Minimize the accessibility of classes and members. *Effective Java* (2^{nd} ed., pp. 67-70)

Additional hints/assumptions:

- For your implementation of `getDistance`, you may want to review breadth-first search.
- You may create any number of files to complete this task (for example, helper classes containing custom data structure implementations).
- You may assume that there is some arbitrary upper limit (for example, 50) on the number of nodes or connections to a node.
- You may assume that each person has a unique name.
- You may assume that there are no self-loops in the graph.
- You may handle incorrect inputs however you want (printing to standard out/error, silently failing, crashing, throwing a special exception, etc.)
- To print something to standard out, use `System.out.println`. For example:

```
System.out.println("DON'T PANIC");
```

We will grade your work approximately as follows:

- Submitting a compiling solution: 20 points
- Working `getDistance` implementation: 10 points
- Fulfilling technical requirements (the bullets directly under "Evaluation"): 10 points
- Documentation and code style: 10 points