Indiana University
CSCI-C212: Introduction to Software Systems (S25) Lab 08
Joshua Crotts, Muazzam Siddiqui

Object-Oriented Programming

What To Do:

Follow each step carefully. As you complete the lab, submit the source files (`.java`) problems to the autograder. After finishing, please let one of the TAs know.

For this lab, since we are now working with classes that have meaningful names, please name the files and testers as we specify.

Problem 1

In this exercise you will design a very simple social media platform.

(a First, design the `Profile` class, which represents an individual on the platform.

(i) A `Profile` has a `String` name, an `int` age, and a `List<Profile>` of friends. The constructor should receive the name and age and assign them to `private` and `final` instance variables. Instantiate the `List<Profile>` as an `ArrayList<Profile>` in the constructor. Do *not* pass a `List<Profile>` to the constructor as a parameter!

(ii) Design the relevant accessor methods for each instance variable. These instance variables are definitionally `final`, so you should not write any mutator methods. When writing the accessor method for the `List<Profile>` instance variable, do not simply return the list. Instead, wrap it in a new `ArrayList<Profile>`. This ensures that the original list reference cannot be mutated by someone outside of the class.

(iii) Override the `public boolean equals(Object o)`, `public int hashCode()`, and `public String toString()` methods accordingly:

- Two `Profile` objects are equal according to `.equals` if they have the same name and age. (Yes, this means that two different people with the same name cannot be on the platform. Sorry!)
- The hash code of a `Profile` is the hash code of its name and age. Remember to use *only* `Objects.hash` and not something else.
  Note: think about why we can't include the `List<Profile>` as part of the hash.
- The `toString` representation of a `Profile` is their name followed by a space, then their age enclosed by parentheses. For example, `"Joshua Crotts (25)"`.

(iv) Design the `boolean addFriend(Profile p)` method that adds a `Profile p` to the friends list of this `Profile`. If that friend already exists in the list, return `false`. Otherwise, add `p` to the list. Be aware that in this context friendship is a symmetric relation (i.e., x is friends with y implies y is friends with x). So, you should also update `p`'s friends list. You should ask yourself, "How do we add `this` instance to `p`'s list?" At the end of the method, return `true`.

(v) Design the `boolean isFriendsWith(Profile p)` method that returns whether this `Profile` is friends with `p`.

When testing `Profile`, you only need to test `equals`, `toString`, `addFriend`, and `isFriendsWith`.

In the next step, you will work with the `SocialMediaPlatform` class. But, we need to discuss writing tests, since this class uses `Profile` objects and contains several methods that must be individually tested.

There are a few ways to do this, and one way is to define an instance of `SocialMediaPlatform`, add `Profile` instances to the system, then test one method, e.g., `oldest`. Doing this for each test method is very cumbersome and prone to typos.

We will, instead, take advantage of JUnit's `@BeforeEach` annotation to establish a `SocialMediaPlatform` with `Profile` s that you can then use to test the methods.

Follow along with your lab instructor.

(b Now, design the `SocialMediaPlatform` class, which manages `Profile` instances. It stores a `Set<Profile>` as an instance variable, indicating all `Profile` instances on the platform.

(i) The constructor of a `SocialMediaPlatform` should receive no arguments. Instantiate the underlying set as a `HashSet` inside the constructor.

(ii) Design the `boolean addProfile(Profile p)` and the `boolean addProfile(String name, int age)` methods that each add a `Profile` to the platform. When two methods that have the same name receive different parameters, it is called *method overloading*. Inside of the latter `addProfile`, call the other version by instantiating a `new Profile` with the provided arguments. Both of these methods return `true` if the system did *not* contain the given `Profile` (and therefore successfully added it) and `false` otherwise.

(iii) Design the `Optional<Profile> mostPopular()` method that returns the `Profile` with the most friends. If there are no `Profile` instances in the system, return `Optional.empty()`.

(iv) Design the `Optional<Profile> oldest()` method that returns the `Profile` that is the oldest according to age. If there are no `Profile` instances in the system, return `Optional.empty()`.

(v) Design the `Set<Profile> mutualFriends(Profile p, Profile q)` method that returns a set of all `Profile` s that are friends with both `p` and `q`. If `p` and `q` are not friends or the platform does not contain `p` nor `q`, return `null`.