CPSC 250L Lab 10
Comparable, Equals, and Encapsulation II

Fall 2017

1 Introduction

The focus of this lab is to define *comparator* methods for a class. A comparator is a method that compares two objects of the same type and returns whether or not another object is "less than", "equal to", or "greater than" the object that called the method. Examples of comparators are Java's `<=`, `==`, `>=`, `Comparable<T>.compareTo(T o)`, and `Object.equals(Object o)`.

This weeks exercise is dependent on last week's `Person` class.

2 Exercises

Fork and clone the `cpsc250l-lab10` repository from the CPSC 250 student group for this semester.

2.1 `Party`

Exercise 1

© Christopher Newport University, 2016

Create a class called `Party` with the following fields.

1. An `ArrayList` of invited people.
2. An `ArrayList` of people who RSVP'd yes.
3. An `ArrayList` of people who RSVP'd no.

Implement the following methods.

1. `public Party()`

   This constructor initializes each `ArrayList` field to a new `ArrayList`.

2. `public void addInvited(Person p)`

   This method adds a *copy* of `p` to the list of invited people. If `p` is already in the list, this method does nothing.

3. `public ArrayList<Person> getInvited()`

   This method returns a *deep copy* of the list of invited people. A deep copy of a list is another list with *copies* of the original list's elements in the same order as the original.

4. `public void addRSVP(Person p, boolean accepted)`

   If `p` is invited, this method will add `p` to the appropriate list. Specifically, if `p` was invited and is accepting the invitation, a copy of `p` should be added to the list of people who RSVP'd "yes". Otherwise, if `p` was invited and is not accepting a copy of `p` should be added to the list of people who RSVP'd "no". Additionally, a `Person` should be able to change their RSVP. Moreover, the list of people who RSVP'd "yes" and the list of people who RSVP'd "no" should be *mutually exclusive*. That is no `Person` should be on both the "yes" list and "no" list at the same time. Furthermore, neither list should contain two copies of the same `Person`.

5. `public ArrayList<Person> getRSVP(boolean accepted)`

   If `accepted` is `true`, this method returns a *deep copy* of the "yes" list. Otherwise, it returns a deep copy of the "no" list.

6. `public boolean equals(Object o)`

   If `o` is not a `Party` object, returns `false`. If this `Party`'s invited, "yes", and "no" lists contain the same elements as `o`'s invited, "yes", and "no" lists respectively then it returns `true`. Otherwise, it returns `false`.

Test your code against `PartyTest.java`.

Exercise 1 Complete

Run:

```
git add .
git commit -m "Completed exercise 1"
git push origin master
```

---

3 Common Mistakes

Some solutions to common mistakes are as follows.

1. To get the lexicographical comparison of two `String` objects, use the `String.compareTo(String)` method.
2. Two objects are equal if and only if `compareTo` returns 0.
3. In the getter methods for `Party`, you **must** return a deep copy of the desired list. To do so, create a new list and iterate through the old one, adding a `copy` of each `Person` to the new list.
4. Be sure that when you move a `Person` from the "yes" list to the "no" list or vice-versa, you remove it from the old list.

5. Ensure that you do not add a `Person` to any list twice!