CPSC 250L Lab 6
Comparable, Equals, and Encapsulation

Fall 2016

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)`.

2 Exercises

Fork and clone the `cpsc250l-lab06` repository in the `cpsc250-students` group.

2.1 `Person`

In this exercise, you will create a class that implements the `Comparable` interface.

Exercise 1

Create a class called `Person` that implements the `Comparable` interface. The `Person` class should have a single field of type `String` that will store the `Person`'s name. Implement the following methods.

1. `public Person(String _name)`

   This constructor should set the `Person`'s name to the `_name` parameter.

2. `public String getName()`

   This method returns the `Person`'s name.

3. `public void setName(String _name)`

   This method changes the `Person`'s name to the `_name` parameter.

4. `public Person copy()`

   This method returns a new `Person` with the same name as this `Person`.

Override the following methods.

1. `public int compareTo(Person other)`

   This method returns the lexicographical comparison of this `Person`'s name and `other`'s name.

2. `public boolean equals(Object other)`

   This method returns whether or not `other` is the same as this `Person`. If `other.getClass` does not equal `Person`, return `false`. Otherwise, return whether or not their names are equal.

Test your code against `PersonTest.java`. Do **NOT** proceed to the next exercise until all tests pass.

Exercise 1 Complete

Run:

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

2.2 `Party`

Exercise 2

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 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.

2. `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.

3. `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`.

4. `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.

5. `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 2 Complete

Run:

```
git add .
git commit -m "Completed exercise 2"
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!