Project 15

CS 1323, Fall 2015

# Learning Objectives

1. Implement a class with private instance data. (10 points)
2. Implement a constructor properly. (10 points)
3. Implement accessor and mutator methods. (10 points)
4. Use ArrayList in two programmer defined classes. (20 points)
5. Fill in method stubs in two classes (Campaign, Fall2016). (20 points)

10 points will be awarded for the documentation of your program. That means using good names for variables, proper and consistent indentation of code, and meaningful use of whitespace.

Section 10: When your program is completed and running, have the teaching assistants check it to get credit for the lab. If you do not submit your assignment during the laboratory, it will be due on Wednesday, December 9 by 11:59 p.m.

Sections 1 and 995: The lab is due by Wednesday, December 9 by 11:59 p.m.

# Description

With presidential elections coming around, we will write software to keep track of a political campaign. Campaigns run on money. Money comes from donors, so keeping track of donors is the main purpose of campaign software.

Each donor has a name and a history of making donations. The Donor class is written below in UML. For the lab on December 1, ignore the + and – signs. We’ll talk about them in class soon.

Donor

-name: String

-donations: ArrayList<Double>

+Donor(name: String)

+Donor(name: String, donation: double)

+getName(): String

+getTotalDonations(): double

+toString(): String

+addDonation(donation: double)

Each Campaign has a candidate and donors. The candidate will have a name. The donors will be stored in an ArrayList<Donor>. Using an ArrayList<Donor> is almost exactly like using an ArrayList<String> . The only difference is that the objects being stored are Donor objects, and will behave like them. The Campaign class is summarized in UML below.

Campaign

-candidateName: String

-donors: ArrayList<Donor>

+Campaign(name: String)

+getCandidateName(): String

+getDonors(): String

+getAllDonations(): double

+addDonor(name: String): void

+getDonation(donor: String) :double

+getDonationList(donor: String) :String

+addDonation(donorName: String, donation: double) : void

Notice which methods are accessors and mutators. Also, some accessors apply only to one of the many Donors (getDonation() is an example, you can tell it applies to only one donor because of the parameter). Accessors like getAllDonations apply to all donors. In order to implement this class, you will have to implement Donor first since this class uses Donor.

I’ve made many of the methods in this class stubs, just to make implementing it less time consuming. This means that I have put a fake return type in the method where necessary so that the program will compile.

The main method that runs this program is menu driven. I’ve written all but two methods for you. This class (Fall2016) is designed like our previous programs have been—with no instance data. The design is below in UML. The underlined methods and data are static (class methods and class data).

Fall2016

-ADD\_DONOR: int

-DONATION: int

-SUM\_DONATIONS: int

-SINGLE\_DONOR\_DETAILS: int

-QUIT: int

+main(args: String[]) : void

-menu(keyboard: Scanner): int

-addDonor(keyboard: Scanner, candidate: Campaign): void

-addDonations(keyboard: Scanner, candidate: Campaign): void

-sumDonations(candidate: Campaign): void

-singleDonorDetails(keyboard: Scanner, candidate: Campaign):void

I left a neat little surprise for you in the menu method. If you plan to continue in programming, check it out!

# Hints

The purpose of separating programs into classes is to make things simpler. It may not seem like this goal is achieved at first. The best piece of programming advice I can give you as we make this transition is that you need to know what type of object you have at all times. Make sure you only use methods that are appropriate for the type of object.

For example:

ArrayList<Donor> list; // assume this is constructed and initialized with some data

If we do list.get() we will get a Donor object because that was what list stores.

Donor person = list.get(0);

If, on the other hand we have this:

ArrayList<Double> donations; // assume this is constructed and initialized with some data

When we do:

double money = donations.get(0) we have a Double (changed to a double, thanks to autoboxing).

The methods in these classes are just a few lines of code (usually 2-10) and that’s the way it’s supposed to be. If you find you’re writing long methods, please ask for help as soon as possible.