CS112 - INTRODUCTION TO PROGRAMMING

Programming Project #3 - **Part I due** Wednesday, March 25, at the start of class
Everything else due Wednesday, April 8, at the start of class

Purposes:

1. To give you experience working with a complete application.
2. To give you experience with testing a program's operation against specifications.
3. To give you experience reading and understanding code written by someone else.
4. To give you experience working with graphical user interfaces and event-driven programming

Introduction

This project is concerned with a very simple application program that maintains an address book. This application is used as a simple example of the process of designing an object-oriented system, which we will discuss more thoroughly in CS211. The project consists of three parts:

- In part I, you will be given a copy of the program that contains numerous deliberate errors. Your task will be to discover these errors by comparing the behavior of the program to its specifications.
- In Part II, you will fix these errors.
- In Part III, you will make some improvements to the program.

Parts I and II are required of all students. Part III is required for full credit on the project. Of course, all students will take the project quiz administered on the final due date.

Before proceeding further, read through the following portions of the web pages discussing the development of this application accessible as "Project 3" from the course home page.

- **"Requirements and User Interface"**
- **"Use Cases"**

Part I (Value 30 points)

1. Copy the `Project3-Executable` folder from the common volume to you server volume. This folder contains an executable jar file `Project3.jar` that (incorrectly) implements the requirements and use cases you read about in the web page. You will not have access to source code at this time.

2. Using test data of your own choosing, test this program with a view to discovering its errors.
   - Only erroneous behaviors that violate the stated requirements / use cases are to be considered errors.
   - There are ten intentional errors in the program. (There may be one or more unintentional errors in the program - bonus credit for discovering such :-)).
   - There is at most one intentional error in each use case. (Where there are two similar use cases - e.g. Save/Save As or Find/Find Again - the same error may affect both use cases. This should be considered just one error in just the the use case that is common to both). Of course, some use cases are completely correct.

3. Document each error you discover, as follows:

   • Give the name of the use case in which the problem occurs.

   • Clearly describe the problem. Note that, in this part, you are focussing on the program's behavior, not what you would do to fix it, so error should be described in terms of what the program does - not in terms of why you think it does it. Describe (1) any specific circumstances under which the problem occurs, (2) what the program does, and (3) what it should do instead.

   • Cite the specific statement from the requirements or use cases that has been violated.

   Example: the following is appropriate documentation for an error that actually does not occur in the program.

Use case:	Add a person
Problem:	If the "Save" item in the File menu was disabled before adding a person, it remains disabled after doing so. It should be enabled, since the address book has been changed.
Violates:	"The 'Save' option in the File menu is enabled."

4. Each error you discover is worth 3 points if it is properly documented as in the above example. (If your description of an error is partially incorrect, or you discover an error but your documentation is unclear or incomplete, you will get 1 or 2 points). **Please note that full credit for finding an error requires that your documentation be as described above.**

5. Each "error" you discover that is not an error (i.e. the program actually does the right thing in this case) will result in the loss of 1 point.

6. **Note that this part is due two full weeks before the due date for the project.** (No late submissions accepted except for emergencies, since I will be going over the errors in class)

Part II (Value 30 points)

Before proceeding further, read the "Further Analysis" page on the project web site.

1. Copy the `Project3-Code` folder from the common volume to you server volume. This folder contains a BlueJ project that was used to create the `Project3.jar` file you tested in Part I. The program can be run by running the `main()` method of class `AddressBookAppliction`.

2. Fix each of the problems identified during testing. (Note: I will go over the problems in class, so you can fix a problem even if you didn't discover it during your testing.)

   The changes required for each fix are very small - usually the addition or removal or change of a single line, or sometimes even a portion of a line. **If your plan for fixing a problem involves more than a few lines of code, or two different methods, look again!**

3. All of the changes will occur in the following classes, only. Though you are encouraged to look at and seek to understand the other classes, they may involve concepts you haven't seen before, and in any case are not the locus of the problems. You should not change any of the other classes.

   `AddressBookGUI`　　　　`AddressBook`　　　　`Person`

Note that some of the computation involved in quitting the program occurs in the `AddressBookApplication` class; however, the code in that class results in the `doClose()` method of `AddressBookGUI` being called, and any changes you need to make can be made there.

4. Document your fixes by giving the name of the class and method in which you made changes, plus "before" and "after" versions of the code you changed. The code you submit should include just enough context to make it clear to the reader where the change was made.

   Example (for the example problem mentioned in Part I - recall that this problem does not actually exist in the code you have been given - it is just meant as an example!)

   Use case: Edit
   Problem: The `doEdit()` method in `AddressBookGUI` is missing a line.

   "Before" code:

```
...
if (changedValues != null)
{
      person.update(changedValues);
}
...
```

   "After" code

```
...
if (changedValues != null)
{
      person.update(changedValues);
      saveItem.setEnabled(true);
}
...
```

5. Each error that is correctly fixed and documented as in the above example is worth 3 points, with partial credit possible for a fix that solves the problem partially but not completely, or that creates a new problem in some cases, or is not properly documented.

Part III (Value depends on what you choose to do - you may do any or all of these, in any order - maximum value 20 points)

Before proceeding further, read the "Maintenance Ideas" page on the project web site.

1. Store the individual's e-mail address in addition to all of the other information about the person, as discussed on the maintenance page. (5 points)

2. Add an option for printing mailing labels, as discussed on the maintenance page. (5 points)

3. Allow multiple address books to be open at the same time, as discussed on the maintenance page. (Again, take note of the comment in Part II above about the computation that takes place in the `AddressBookApplication` class resulting in the `doClose()` method of `AddressBookGUI` being called.) (5 points)

4. Add an option for searching for an exact match, as discussed on the maintenance page. The menu accelerator for exact match search should be shift-F, while that for contains search would remain plain F. (See Save/Save All for handle the accelerator key). (5 points)

5. Some other possibility **approved by me in advance**: (points to be negotiated) (Note: no credit for an additional feature not approved in advance. An added feature will not be approved until the requirements for Part II are complete.

Project Quiz - given on the Final due date (maximum value 20 points)

Turn in:

1. Part I: A description of the errors you found, documented as in the example above. (Please word-process your documentation, or write it very neatly by hand so I can read it!)

2. Parts II-III:

   a. Documentation for the changes you made for Part II as in the example above. You do not need to turn in complete printouts of the code as long as your documentation makes "before" and "after" clear.
   b. Printout of the relevant portions of the java sources for the classes you modified for Part III. Please highlight the changes you made in some way. You do not need to print portions of the code in which you made no changes.

Place in the drop box on the server:

Source and compiled versions of the program you turned in, in a folder called *yourname3* - e.g. Anthony Aardvark would place his project in the drop box in a folder labeled Aardvark3.