Computer Science 1260 – Project 2

March 15, 2018 | Address Book – Part I | Due: Check Course Calendar

Introduction

Our company has been hired to develop a customized **Address Book** application. We intend to build the application in phases. In the first phase, we will develop a **Contact** class. An object of this class will eventually represent one entry in the completed **Address Book**. To test the **Contact** class and verify that it does what it is intended to do, we must also develop a **Contact Demo** driver class to demonstrate the functionality of the **Contact** class. All code you write for this project should be in the **addressBook package**.

The Contact Class

Each **Contact** should contain at least the following information:

a. **Contact type**: an enumerated type with one of the following values (The enum should be in a separate file)
    a. Family
    b. Church
    c. Friend
    d. BusinessColleague
    e. ServicePerson
    f. Customer
    g. Other

b. **Name**

c. **Street Address**

d. **City**

e. **State** (two letter official postal abbreviation such as TN, SC, GA, etc.)
    a. Must be one of the official state abbreviations. Anything else is invalid and an XX should be stored instead. The state abbreviations should be stored in an array. You need to use the appropriate method from the ArrayOperations class to determine if what they passed in is valid. A list of the state abbreviations has been provided in a text file that you can use to quickly populate the array.

f. **ZipCode**

g. **Phone** (including area code)
    a. Must be 10 characters long. Must not include hyphens, parenthesis or periods. You are to format it in the following format before storing it…(111)222-3333. If it is not 10 characters long, it is invalid and should be stored as (000)000-0000. (Hint: you can use the String method substring to retrieve parts of the String to build the new version of it.)

h. **Email** (email address)
    a. Must have an @ somewhere in the address with a '.' somewhere after the @. If it is not in the proper format, it should be stored as "invalid@address.given"

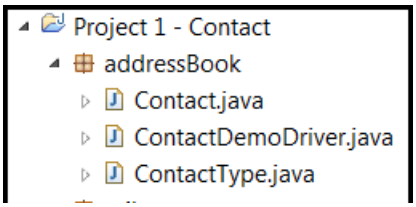
i. **Photo** (the name and relative path of a .jpg file containing the contact's photo)

   a. The photo can only be of type jpg. If the filename comes in without the ".jpg" or with a different extension, you are to store it as "InvalidFileName.jpg"

The class requires a minimum of three **constructors** (default, parameterized and copy), appropriate **getters** and **setters**, and a **toString** method that formats all of the **Contact** information for a user-friendly multiline String that can be displayed (through either **System.out** or a **JOptionPane** dialog) – but is not to be displayed in the toString method. **The Contact class should not do any input/output.**

The Driver Class

For the driver, we will not be creating a driver that will be used by the user yet. At this phase of the ongoing projects, this driver will be one that will simply demonstrate that you have fully tested the Contact class. Your grade on this part of the project will be determined by how thoroughly you tested the Contact class. There are no specific requirements other than the fact that your code must follow the department programming standards.



Project Structure

The structure of your solution should look something like this graphic in the Eclipse Package Explorer. You will have other enum files that are not listed, as well as the ArrayOperations class, but this should give you an idea of the structure.

Documentation

The **documentation standards** are posted on the course web site. **DOCUMENTATION IS NOT OPTIONAL**. If your version of **Eclipse** is set up as we did in the lab on the first day, and if you use **Eclipse** the way you were taught that day, **Eclipse** will do much of the documentation for you. However, you must fill in the details (such as the purpose of a method or a parameter). If you did not set up your version of **Eclipse** properly or you do not use it as you were taught in the lab to use it, you will have to provide all documentation manually.

Penalties for failure to **FULLY** document your code are **severe**.

Deliverables

Check the course calendar for due dates.

In addition to the project itself, you are to create and submit a design document for this project. Ideally, the design document is submitted before the project because it is the description of what the project is (the classes and how they will interact – **class descriptions and UML**), the plan for how you are going to create the class (what they will include and how it will be implemented – **UML and algorithms**) and how you plan to test them (**test cases**). For this project only, you will submit the design document along with the project submission. In future assignments you will be submitting it early (and possibly a revised version with the project submission). An example of a design document and the instructions for creating one are posted in D2L. It does not have to look exactly like the example…that is just something

to give you an idea of the structure and what it should contain. As long as it has all of the required contents, the layout and design of it is up to you. It must be word-processed (i.e., not a text file), readable, neatly organized and appropriately formatted. The UML diagram must be created with Astah or a similar UML modeling tool. You should have an algorithm for every method with the exception of getters and setters that simply store a value (no logic required).

You are to upload the following to D2L:

- Zipped folder containing:
  - the project folder (which contains all .java and .class files, as it is created by Eclipse), zipped (named **YourlastnameYourfirstnameProject2.zip**)
- Your design document in Word or PDF format
  - This should not be submitted in the zipped folder. It should be submitted separately.