CS 646 iPad/iPhone Mobile Application Development
Fall Semester, 2016
Assignment 2

San Diego State University -- This page last updated 9/14/16

Assignment 2
Due Sept 18 11:59 pm

Version 1.1

100 Points Total

Objectives

1. Use more Swift features - classes, enum, struct, protocols, extending classes, etc

Programming Problems

1. (5 points) Create an enum with associated values for Currency. We will support four currencies: Euro, US Dollar, Indian Rupee, Mexican Peso. Each currency has three letter code (EUR, USD, INR, MXN), a symbol ("\u{20AC}", "\u{24}", "\u{20B9}", "\u{24}") and an exchange rate with respect to USD (0.8904, 1, 66.7, 18.88).

2. (15 points) Create a Money struct. Creating a new money instance requires an amount and a type of currency, which is indicated by using the enum from problem one. Once created a money instance does not change in value. Each instance of Money needs access to the same instance of a Currency enum to access the same conversion rate. The Money struct needs three methods: +, - and description. Both - and + operate on two Money instances. They return a new Money instance. If the two instances are of different currencies the result is in the first (left operand) currency. The method description returns a string representation of the money instance. The string starts with the currency symbol, followed by the amount, followed by a space and ends with the three character symbol for the currency. For example $123.12 USD. Do not worry about formatting the amount.

3. (5 points) A phone number in this country has 10 digits like 6195946191 or 619JYDN191. It is hard to read a phone number formatted like that. One common format is 619-594-6191. Add the method **phoneFormat** to the String class. The method **phoneFormat** converts strings like "6195946191", "619 594 6191", "619 5946191" and "619-594-6191" to "619-594-6191". That is all the methods below will return "619-594-6191". You can assume that phoneFormat will be called only on valid phone numbers.

   "6195946191".phoneFormat()
   "619 594 6191".phoneFormat()
   "619 5946191".phoneFormat()
   "619-594-6191".phoneFormat()

4. (15 points) Create a PhoneNumber structure. A PhoneNumber instance has a number and a type. We will use only US phone numbers. A type can be mobile, home, work, main, home fax, work fax, pager or other. Define an enumeration for phone types. The

PhoneNumber structure needs to have the following methods. It can have more methods if needed. It should have an initializer that accepts only the phone number with default type being home. It also needs an initializer that accepts both a phone number and a phone type.

isMobile()-> Bool
Returns true if the phone number is for a mobile phone.

isLocal() -> Bool
Returns true if the phone number has the area code 619 or 858.

description() -> String
A property that returns a string description of the phone number in the format "type: number". For example 619 594 6191 is a work number so it would be "work: 619-594-6191"

5. (5 points) Add a method **asPhoneNumber** to the String class. The method takes a string of the format "work: 619-594-6191" and returns a PhoneNumber instance with the given types and number. If the string is not of the format "type: number" then the method **asPhoneNumber** returns nil.

6. (15 points) Create a Name structure. A name has two parts a first name (also called a given name or personal name) and a last name (family or surname). The Name structure should implement the Comparable protocol.

7. (20 points) Create a Person class. A Person has a first name, last (or family) name and 0 or more phone numbers. The person class should have the following methods (more if needed):

addPhoneNumber(String,PhoneType)
Adds a string representing phone number with given type, which has default type home

addPhoneNumber(PhoneNumber)
Add the PhoneNumber.

phoneNumber(PhoneType)->PhoneNumber?
Returns the person's phone number of the given type. Or nil if number does not exist.

hasNumber(String) -> Bool
Returns true if person has given phoneNumber.

8. (20 points) Create a ContactList class. Internally the ContactList maintains one list of your contacts. Your class should have at least the following methods. You may find that you want/need more methods in the class.

addPerson(Person)
Add a Person object to the list.

orderedByName() -> Array

Returns an Array of all your contacts ordered by last name.

phoneNumberFor(String) -> PhoneNumber?

Given the lastName return phone numbers for the first person in the list that has that last name. Return nil if no such person exists.

nameForNumber: (String) -> Person?

Return the person with the given phone number. Return nil if no one has the phone number.

Grading

In addition to the points indicated for each problem there will be 10 points for style. Style includes formatting your code reasonably and consistently, using Swift naming conventions and using the appropriate language constructs.

What to Turn in

Put all your code into a playground. Place the playground into a zip file. Turn in your zipped file using blackboard under assignment 2.

Late Penalty

An assignment turned in 1-7 days late, will lose 3% of the total value of the assignment per day late. The eight day late the penalty will be 40% of the assignment, the ninth day late the penalty will be 60%, after the ninth day late the penalty will be 90%. Once a solution to an assignment has been posted or discussed in class, the assignment will no longer be accepted. Late penalties are always rounded up to the next integer value.

Version History

1.1 In problem 2 replace “They return a new currency instance” with “They return a new Money instance”.