MIT AITI Mobile Development in Java
Java Lab 04: Inheritance

This lab builds on the previous lab, and uses the concepts of Encapsulation covered in Lecture 3. Please go back to those slides if you need a refresher.

You will be provided with source code for a test program called Lab04Test that will interface with your implementation. Lab04Test will call your code. Thus, your code needs to adhere to the API defined in this document. The main method for this lab is located in Lab04Test.

A user may want to include different types of contacts in their contact list: businesses and persons. They hold different information, so you do not want to create a single class that models both. Since they share some structures, you want to create a class hierarchy to represent the different types of contacts. This leads to better code organization and reuse. You must create a superclass, `Contact`, and subclasses for the two different types of contacts, one for businesses and one for persons. The two subclasses inherit all `public` and `protected` properties of the class `Contact`. In addition, you will create a `ContactList` class as described in the following.

You will have to test your code with a test program we provide.

1. Create a new project for this lab, title it "lab04". In your new project, create three new classes, `Contact`, `Business`, and `Person`, and store them in the src folder. These classes will not have a main method. Also, import Lab04Test.java into this project.

2. Implement the *Application Programming Interface* (API) for the `Contact`, `Business`, and `Person` classes. Remember that the `Business` and `Person` classes inherit all the properties of the `Contact` class. Keep this in mind as you are writing your code. The API definition begins on the next page. You must abide by this API in order for Lab04Test to test your code properly.

   The API is an interface contract that your code must adhere to. You are required to implement the internals of the classes for the lab. The API does not specify the fields required. Also, the API does not specify the visibility modifiers. Keep the ideas of abstraction and encapsulation in mind when implementing this lab.

3. Create the `ContactList` class. This class will be used to keep track of your contacts. Its API is given below.

   Instead of using an array of `Contact` in the `ContactList`, you should use an `ArrayList`. `ArrayList` is a class that models an array. However `ArrayLists` do not have a specified size, so you can keep adding elements to them. The `ArrayList` class defines many useful methods for `ArrayLists`. See the API at:

http://java.sun.com/javase/6/docs/api/java/util/ArrayList.html

Here is some sample code that uses an `ArrayList` of `String`s. The code will output "Mike" then "2" then "Mike thinks AITI is cool" then "Juice thinks AITI is cool":

```
ArrayList<String> names = new ArrayList<String>();
names.add("Mike");
names.add("Cory");
names.add("Juice");

names.remove("Cory");

System.out.println(names.get(0));
System.out.println(names.size());

for (int j = 0; j < names.size(); j++)
{
      System.out.println(names.get(j) +
                         " thinks AITI is cool");
}
```

You also need to add the following line to the beginning of your ContactList class:

```
import java.util.ArrayList;
```

4. Test your code using lab04's Lab04Test.java. If the test is successful, Lab04Test.java should only print out "Passed" a number of times.

5. Have your Contact class implement the `Comparable<T>` interface:

   http://java.sun.com/javase/6/docs/api/java/lang/Comparable.html

   Your `compareTo` method should compare two Contacts based on their names. Look at the `compareTo` method in the `String` class. Next, add a method to your ContactList that will sort the Contact ArrayList by name. You should not implement a sort yourself, you should instead call the `sort(List<T>)` method of the Collections class:

   http://java.sun.com/javase/6/docs/api/java/util/Collections.html

   Do not forget to add "`import java.util.Collections;`" at the beginning of your class.

Class Contact

A `Contact` is constructed by giving the contact's name, phone number, and address to the `Contact` constructor.

Constructor Summary

```
<modifiers> Contact()
```

Creates a new Contact with blank information.

```
<modifiers> Contact(String name, String phoneNumber, String address,
                    String email)
```

Creates a new Contact with the given name, phone number, address, and email.

Method Summary

```
<modifiers> String getName()
```

Returns the contact's name.

```
<modifiers> String getPhoneNumber()
```

Returns the contact's phoneNumber.

```
<modifiers> String getAddress()
```

Returns the contact's address.

```
<modifiers> String getEmail()
```

Returns the contact's email.

```
<modifiers> void print()
```

Prints all the contact information, each piece on a separate line. It should print as so:

Name: Contact name

Phone Number: Contact phone number

Address: Contact address

Email: Contact email

```
<modifiers> void changeEmail(String newEmail)
```

Changes the email to the new email.

```
<modifiers> void changePhoneNumber(String newPhoneNumber)
```

Changes the phone number to the new phone number.

```
<modifiers> void changeAddress(String newAddress)
```

Changes the address to the new address.

```
<modifiers> void queryUser()
```

Create a method that will prompt the user to input an email, phone number, and address for the contact and have the method change these fields for the contact. This method can be used when querying the user for contact information. If using this method, you can first use the no argument constructor, which does not set a value for any of the fields of the contact.

Your code should prompt the user to enter input in this format:

Email:

Phone number:

Address:

Class Business

A business is a specific type of contact that derives from the `Contact` class. A `Business` contact models a business such as a restaurant or a market. A business is constructed by giving a name, phone number, and age to the `Business` constructor.

Constructor Summary

```
<modifiers> Business(String name, String phoneNumber,
        String address, String email, String operatingHours,
        String industry, String size)
```

Creates a new Business contact with the given name, phone number, address, and e-mail. Operating hours should be in the form 08:00-15:00. Examples of industries are retail, restaurant, transportation, tourism, etc. Business size will be either micro, small, medium, or large.

Method Summary

```
<modifiers> String getOperatingHours()
```

Returns the Business's operating hours.

```
<modifiers> String getIndustry()
```

Returns the Business' industry.

```
<modifiers> String getSize()
```

Returns the Business' size.

```
<modifiers> void changeOperatingHours(String newOperatingHours)
```

Returns the Business's operating hours.

```
<modifiers> void changeIndustry(String newIndustry)
```

Returns the Business' industry.

```
<modifiers> void changeSize(String newSize)
```

Changes the Business' size. If the input is not “micro”, “small”, “medium”, or “large”, the method should do nothing.

`<modifiers> void print()`

Prints the full information of the business in the form of Size Industry + "Business", two examples would be: Small Transportation Business or Medium Retail Business.

`<modifiers> void queryUser()`

The business has grown and has also changed its operating hours to reflect this growth. Create a method that allows the user to input a new business size and operating hours for the business. Then call on other methods you've written to change these values for the Business contact.

Class Person

Person contacts are a specific type of contact that derives from the `Contact` class. A `Person` contact is constructed by giving a gender, birthday, and age to the `Person` constructor.

Constructor Summary

```
<modifiers> Person(String name, String phoneNumber, String address,
                   String email, String birthday, int age,
                   char gender)
```

Creates a new Person contact with the given name, phone number, address, e-mail, birthday, age, and gender (M for male and F for female)

Method Summary

```
<modifiers> String getBirthday()
```

Returns the person's birthday.

```
<modifiers> int getAge()
```

Returns the person's age.

```
<modifiers> char getGender()
```

Returns the gender.

```
<modifiers> String correctBirthday(String newBirthday)
```

Corrects the person's birthday (used if entered incorrectly).

```
<modifiers> void correctAge(int newAge)
```

Corrects the person's age (used if entered incorrectly).

```
<modifiers> void correctGender(char newGender)
```

Changes the person's gender (used if entered incorrectly).

```
<modifiers> void birthday()
```

Adds 1 to the person's age. (Called on a birthday).

```
<modifiers> void print()
```

Print out "Born on " [birthday] "and is " [age] "years old."

```
<modifiers> void queryUser()
```

You've accidentally initialized a person contact with the wrong information. Create a method that will prompt the user for the correct birthday, age, and gender of the person contact. The method should prompt the user for information using the format below:

Correct birthday:

Correct age:

Correct gender:

Class ContactList

ContactList is used to manage a list of `Contacts`.

Constructor Summary

`<modifiers> ContactList()`

Creates a ContactList with an empty `ArrayList` of `Contacts`.

`<modifiers> ContactList(ArrayList<Contact> cs)`

Creates a ContactList with the given `ArrayList` as the initial set of `Contacts`.

Method Summary

`<modifiers> ArrayList<Contacts> getContacts()`

Returns the `ContactList`'s `Contacts`.

`<modifiers> Contact getContact(String name)`

Returns the `Contact` with the given name if the `Contact` is in the `ContactList`. Otherwise returns null.

`<modifiers> void addContact(Contact c)`

Adds the given `Contact` to the `ContactList`.

`<modifiers> boolean changeEmail(String name,String newEmail)`

Changes the e-mail of the `Contact` with the given name to the new e-mail. Returns `true` if a `Contact` with the given name is in the `ContactList`, otherwise returns `false`.

`<modifiers> boolean changeAddress(String name,String newAddress)`

Changes the address of the `Contact` with the given name to the new address. Returns `true` if a `Contact` with the given name is in the `ContactList`, otherwise returns `false`.

`<modifiers> boolean changeNumber(String name,String newNumber)`

Changes the phone number of the `Contact` with the given name to the new phone number. Returns `true` if a `Contact` with the given name is in the `ContactList`, otherwise returns `false`.

`<modifiers> void removeContact(Contact c)`

Removes the given `Contact` from the `ContactList`.

`<modifiers> void removeContact(String name)`

Removes the `Contact` with the given name from the `ContactList`. If there are multiple `Contacts` with the same name, you may decide whether to remove one or to remove them all.

`<modifiers> boolean hasContact(Contact c)`

Returns `true` if the given `Contact` is in the `ContactList`. Otherwise, returns `false`.

`<modifiers> boolean hasContact(String name)`

Returns `true` if a `Contact` with the given name is in the `ContactList`. Otherwise, returns `false`.

`<modifiers> void printContacts()`

Prints out a *numbered* list of the `Contacts` in the `ContactList`. Each `Contact` should print out as it normally does according to its `print()` method.