CSE 174 – Fall 2018
PROGRAM #13: 50 points – Due Sunday, Dec 01, by 11:59 p.m.

Outcomes:

- Write programs that use object oriented programming concepts.
- Write programs that use objects.
- Write programs that use ArrayList.
- Write applications using object oriented programming concepts.

Scoring:

- If you do not submit a .zip file containing your source code, your score will be zero.
- If you submit source code that does not compile, your score will be zero.
- If you submit source code without the correct class names or the correct method names (see below in the instruction), you will receive at most half credit for this assignment.
- Deductions will be made for not meeting the usual requirements:
    - Source code that is not formatted according to the usual guidelines, including commenting each method to explain its purpose. See program1 for an example of how we comment our methods.

	Full credit	**Partial credit**
Implement the ContactInfo class (15 points)	You implemented the class and it passes all tests.	You implemented the class and it passes tests partially.
Implement the PhoneBook class (25 points)	You implemented all the methods and it passes all tests.	You implemented all the methods and it passes tests partially.
Output format as shown in the sample output, comments, small methods, proper messages for invalid inputs and errors, and proper indentation. (10 points)	Checking all errors and invalid inputs. Proper indentation, having comments, structured classes/output similar to the sample output.	There are errors in formatting and not having clean codes and outputs as described.

Instructions:

Please read the instructions carefully line by line!!!!!

1. Create a folder called Program13.
2. Download Phone.java and Phonebook.java and put them inside the Program13 folder.
3. Open the Phone.java in DrJava file and study the class.
4. DO NOT CHANGE ANYTHING INSIDE THIS CLASS.
5. Study the following:
    - How many fields you have and how you can access them.
    - How many constructors you have, what are the inputs, and how they initialize the fields.
    - How many methods you have and which ones you have access to from outside the class (public/private methods).
    - Understand what each method does.
6. Compile Phone.java, there should be no compile error.
7. Let's test the class. In the interaction part try these (you can simple copy/past the commands):

```
> Phone p1 = new Phone("iphone", "(513)000-0000")
> p1.getLabel()
"iphone"
> p1.getPhonNum()
"(513)000-0000"
> Phone p2 = new Phone("mobile", "(513)111-1111")
> p2.getLabel()
" mobile "
> p2.getPhonNum()
"(513)111-1111"
> Phone p3 = new Phone(p2)
> p3.getLabel()
" mobile "
> p3.getPhonNum()
"(513)111-1111"
> p1.setLabel("mobile")
Static Error: No method in Phone has name 'setLabel'
```

8. At this point if you don't understand the concept and why some of those lines work and some don't, please stop right here! Spend 20~30 minutes to study the textbook and slides. You can also use the Discussion part to ask questions.
9. Now you are ready to implement your own class (use the Phone class as an example).
10. Create a class inside the Program13 folder and call it **ContactInfo**. Now implement your **ContactInfo** class with the following details:

Field Summary

Modifier and Type	*Field and Description*
String	name Holds a name i.e. Meisam Amjad
ArrayList<Phone>	phoneNums Holds a list phone numbers for this specific individual

Constructor Summary

Constructor and Description
ContactInfo() Default constructor and initializes a contact info with name = "unknown" and an empty list (length of zero)
ContactInfo (String name, ArrayList<Phone> phoneNumber) Constructor and initializes a contact info with the given name and the given list of phones.

Method Summary

Modifier and Type	*Method and Description*
string	getName() Returns the name field
ArrayList<Phone>	getPhones() Returns an ArrayList of phones
boolean	addPhone(Phone p) Adds the given phone to the list of phones. If it was successful the method returns true, otherwise false. If a similar label **or** phone number already exist in the list of phones, this method should return false without adding the given phone to the list.
boolean	removePhone(String phonNum) Removes the given phone number from the list of phones. If it was successful the method returns true, otherwise false. If the list is empty or there is no such phone number in the list of phones, this method should return false without removing anything.

11. Above methods and constructors are the only accessible things in this class for other users.
12. You can write as many helper methods as you like.
13. When you finished the class, don't forget comments at the top, for each method, and inside your code explaining what is going on.
14. You should not have any method longer than 12 lines.
15. Now if you're done with the class, let's test your class. Compile your code and run the following commands inside the interaction part and make sure that you are getting exactly the same results. In case you need to change your code, you must compile your code and run the commands again.

```
> ContactInfo pp1 = new ContactInfo ()
> pp1.getName()
"Unknonw"
> pp1.getPhones()
[]
>  Phone p1 = new Phone("mobile", "(555)444-4444")
> import java.util.ArrayList
> ArrayList<Phone> list1 = new ArrayList<Phone>()
> list1.add(p1)
true
> pp1 = new ContactInfo ("Jim Carrey", list1)
ContactInfo @3bd4bdab     (if this address is different for you, don't worry, can you say why?)
> pp1.getName()
"Jim Carrey"
> pp1.getPhones()
[Phone@70cedb55]
> for (Phone phone: pp1.getPhones()){
System.out.println(phone.getLabel() + "  " + phone.getPhonNum());
}
Mobile  (555)444-4444
> pp1.addPhone(new Phone("home", "(666)666-6666"))
true
> for (Phone phone: pp1.getPhones()){
System.out.println(phone.getLabel() + "  " + phone.getPhonNum());
}
mobile  (555)444-4444
home  (666)666-6666
> Phone p3 = new Phone("main", "(666)666-6666")
Phone@2fa30cf4
> pp1.addPhone(p3)
false
> pp1.addPhone(new Phone("home", "(123)000-8723"))
false
> pp1.removePhone("(000)000-0000")
false
> pp1.removePhone("(555)444-4444")
true
> for (Phone phone: pp1.getPhones()){
System.out.println(phone.getLabel() + "  " + phone.getPhonNum());
```

```
}
home  (666)666-6666
>
```

16. Above commands are just some examples to test your code with. Having the same results does not indicate that your code is fully functional. You still need to test your code with other instances. For example, when the list of phone numbers is empty and you are trying to remove a phone number from an empty list, does your method return false? So, come up with other test cases and make sure your class works properly with everything mentioned in the instructions. Your code will be checked for all cases (those mentioned in the instruction) and graded accordingly.
17. Congrats for making it this far.
18. Now you get to write your own **PhoneBook.java** application**.**
19. Open PhoneBook.java inside DrJava. You should have this file in your Program13 folder from step #2.
20. You need to implement a menu driven application that includes the following methods:

```
public static void add(ArrayList<ContactInfo> list,
                       String name, String label, String pNum) {
    // Adds a ContactInfo object to the list
    // If the name is already in the list
    // the method should print:
    // "The name already exists!"
    // If the contact is added successfully, the method should
    // print "*The contact has been added successfully*"
}
```

```
public static void append(ArrayList<ContactInfo> list,
                       String name,  String label, String pNum) {
    // Adds a new Phone object to a specific ContactInfo
    // If this person does not exist the method should print:
    //   ,"Couldn't find the name!"
    // If a similar label or phone number already exists for
    // that person, the result should be:
    //     "The label/number already exists for this person!"
    // If the number is added, the method should print
    // "*The number has been added successfully*"
}
```

```
public static void display(ArrayList<ContactInfo> list, String name) {
    //Displays all the phone numbers that belongs to the given
    // name.
    // If the name doesn't exist in the list, the method should print
    // "Couldn't find the name"
}
```

```
public static void displayAll(ArrayList<ContactInfo> list){
    // Displays all the names and the phone numbers.
    // If the list is empty, the method should print,
    //"The list is Empty!"
}
```

```
public static void remove(ArrayList<ContactInfo> list,
                          String name) {
    // Remove all data related to the given name from the
    // list.
```

```
    // If the list is empty, the method should print
    // "The list is Empty!",
    // If the name does not exist the method should print
    // ,"Couldn't find the name!"
    // If the contat is removed the method should print
    // *Contact is removed successfully*
}
```

21. Some codes have been already written for you inside the PhoneBook.java file as a hint to put you in a right direction, so use them.
22. You can add as many helper methods as you like.
23. Methods should not have more than 12 lines.
24. After completing each method use the interaction part to test that method. Do you recall how to test a single method!? (You have done it in Lab12)
25. Once you have your program working well, add the following menu in the main method (you can have a different method for the menu).
26. Use a loop to repeat your menu and implement error checking, so that the user cannot enter illegal inputs and make sure that it terminates the program only on user's choice 6.

Sample run: (Text shown in red are user's inputs)

```
> run PhoneBook

1. Add a contact
2. Add a new number to an old contact
3. Display a contact
4. Display All
5. Remove a contact
6. Exit
Enter your choice: 0
Invalid Input!!

1. Add a contact
2. Add a new number to an old contact
3. Display a contact
4. Display All
5. Remove a contact
6. Exit
Enter your choice: 7
Invalid Input!!

1. Add a contact
2. Add a new number to an old contact
3. Display a contact
4. Display All
5. Remove a contact
6. Exit
Enter your choice: 4
```

```
---Display all
The list is empty

1. Add a contact
2. Add a new number to an old contact
3. Display a contact
4. Display All
5. Remove a contact
6. Exit
Enter your choice: 1
---Add a contact
Name:  Jim Currey
Label: home
Phone number (i.e. (513)111-1111): (000)000-0000
*You contact has been added successfully*

1. Add a contact
2. Add a new number to an old contact
3. Display a contact
4. Display All
5. Remove a contact
6. Exit
Enter your choice: 1
---Add a contact
Name: Jim Currey
Label: cell
Phone number (i.e. (513)111-1111): (000)000-0000
The name already exists!

1. Add a contact
2. Add a new number to an old contact
3. Display a contact
4. Display All
5. Remove a contact
6. Exit
Enter your choice: 2
---Add a new number
Name:  Jim Curr
Couldn't find the name

1. Add a contact
2. Add a new number to an old contact
3. Display a contact
4. Display All
5. Remove a contact
6. Exit
Enter your choice: 2
---Add a new number
Name:  Jim Currey
Label: home
Phone number (i.e. (513)111-1111): (333)333-3333
```

```
The label/number already exists for this person!

1. Add a contact
2. Add a new number to an old contact
3. Display a contact
4. Display All
5. Remove a contact
6. Exit
Enter your choice: 2
---Add a new number
Name:  Jim Currey
Label: mobile
Phone number (i.e. (513)111-1111): (333)333-3333
*The number has been added successfully*

1. Add a contact
2. Add a new number to an old contact
3. Display a contact
4. Display All
5. Remove a contact
6. Exit
Enter your choice:  3
---Display a contact
Name: Jim Currey
home: (000)000-0000
mobile: (333)333-3333

1. Add a contact
2. Add a new number to an old contact
3. Display a contact
4. Display All
5. Remove a contact
6. Exit
Enter your choice: 4
---Display all
Jim Currey
     home: (000)000-0000
     mobile: (333)333-3333

1. Add a contact
2. Add a new number to an old contact
3. Display a contact
4. Display All
5. Remove a contact
6. Exit
Enter your choice: 1
---Add a contact
Name: Adam Sandler
Label: home
Phone number (i.e. (513)111-1111): (777)777-7777
*You contact has been added successfully*
```

```
1. Add a contact
2. Add a new number to an old contact
3. Display a contact
4. Display All
5. Remove a contact
6. Exit
Enter your choice: 4
---Display all
Jim Currey
     home: (000)000-0000
     mobile: (333)333-3333
Adam Sandler
     home: (777)777-7777

1. Add a contact
2. Add a new number to an old contact
3. Display a contact
4. Display All
5. Remove a contact
6. Exit
Enter your choice: 5
---Remove a contact
Name:  car
Couldn't find the name!

1. Add a contact
2. Add a new number to an old contact
3. Display a contact
4. Display All
5. Remove a contact
6. Exit
Enter your choice:  5
---Remove a contact
Name:  Jim Currey
*Contact is removed successfully*

1. Add a contact
2. Add a new number to an old contact
3. Display a contact
4. Display All
5. Remove a contact
6. Exit
Enter your choice:   4
---Display all
Adam Sandler
     home: (777)777-7777

1. Add a contact
2. Add a new number to an old contact
3. Display a contact
```

```
4. Display All
5. Remove a contact
6. Exit
Enter your choice: 6
>
```

Advice:

1. Your application uses an **ArrayList** called **list** that is capable of storing all **ContactInfo** objects for any number of individuals.
2. The **list.size()** returns the length of the list.
3. Compress your **Program13** with all files inside it and submit your **Program13.zip** on canvas.