# Concordia University
# Comp 248 - Winter 2016
# Introduction to Programming
# Assignment # 5
**Due by 11:59 PM, April 8, 2016**

This assignment is partially built over assignment # 4. In specific, you may utilize parts of the CellPhone class that you have created in assignment 4, with possibly some modifications. To minimize any reference to the old assignment, the full details of the CellPhone class is indicated below, so there is no need to refer to the description given in Assignment 4.

A CellPhone object has three attributes, a brand (String), a serial number (long), and a price (double).

## Part 1:

You are required to design and implement the **CellPhone** class according to the following specifications:

- Upon the creation of a CellPhone object, the object must immediately be initialized with valid values; that is: brand, serial number and price. You may assume that serial numbers do NOT start with 0 (that is, they can only start with digits from 1 to 9).

- Your class should have a copy constructor, so new CellPhone objects can be created as exact copies of existing ones (for simplicity, we assume here that serial numbers can be duplicated).

- The design should allow enough flexibility so that the value of any of these attributes can be modified later on. For example, it should be possible to create a CellPhone object with a given price then change its price later on.

- The design should allow the user of the class to obtain the value of any of the attributes individually (i.e. getPrice()).

- The design should also allow the user to view all CellPhone information at once using the System.out.print() method. That is, a statement such as: ***System.out.println(c1);*** would display all information of CellPhone c1.

  **Hint:** Use *toString()* method.

## Part 2:

In this part, you are required to write a public class called **CellPhoneSearch**, which will utilize the CellPhone class created in part 1. In the main method of this class the following must be done:

- Create an array of 10 cell phones. All these cell phones must be initialized with proper values; that is: brand, price & serial number. You must use the copy constructor to create some of these objects.

- Ask the user to enter three pieces of information: a cell phone brand, a cell phone price, and either *Yes* or *No* for a search combination (see details below).

- Read the entered information.

- Depending on whether the user has entered Yes or No for matching combinations, your program will search the array for matching cell phones based on the following rules:

  → If the user enters *Yes* for matching combinations, then your program will search the array and displays all cell phones that have <u>both</u> the brand and the price entered by the user.

  → If the user enters *No* <u>(or anything different than Yes, or any of its variations i.e. YES)</u> for matching combinations, then your program will search the array and display all cell phones that have either the brand entered by the user or the price entered by the user.

  If no matches are found, the program must indicate that to the user.

  As a general rule, if the display is only for the prices, then it must be formatted using *printf*.

**<u>Guideline:</u>** For part 1 and 2, you will need to create one .java file, called **CellPhoneSearch.java**, which includes the *CellPhone* class as well as the *CellPhoneSearch* class (this is the public class), which will have the main() function.

## Part 3:

For this part, you are required to write a class called **ModifyCellPhones**, which is going to utilize the CellPhone class that you have created in part 1 (notice that Part 3 has nothing to do with Part 2). In that part, you are required to write a public class, the **ModifyCellPhones** class. Besides the main() method, this class will also include another *static* method called **ModifyPhonePrices()**. The details of this method are as follows:

- The method accepts 3 parameters: a 2-dimentional array of cell phones, and two double values. The method also returns an integer.

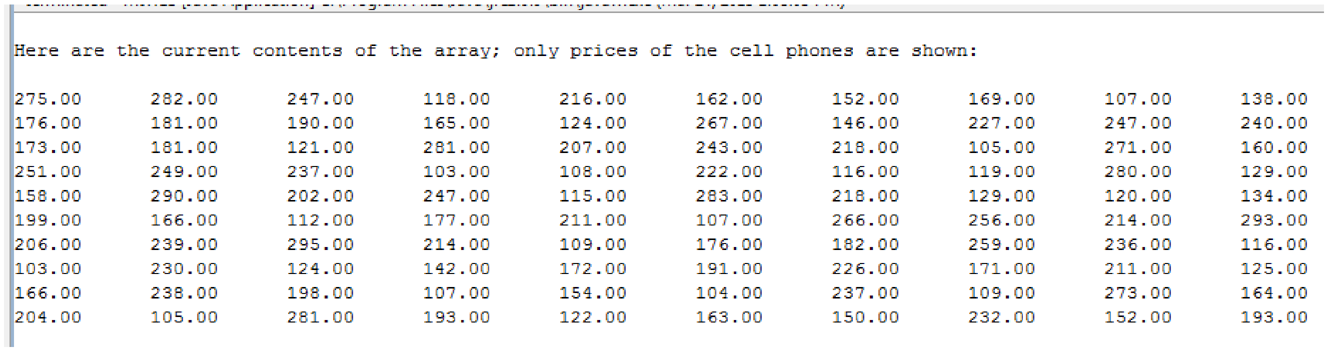
- The method will search the entire array looking for any cell phone with a price that matches the 2nd parameter, and if found, the method will replace the price of that cell phone with the value passed as the 3rd parameter.
- The method should display the array location of each cell phone that has been changed.
- The method should also keep track of how many changes were made in total, and finally return this number.

In the main() method of this class the following must be done:

- Create a 2-dimentional array of 10x10 cell phones; all cell phones in the arrays must be initialized with proper values; that is brand, price and serial number. Many of these phones may have identical values (you may initialize them in a *for* loop, and you do not need to worry about phones with duplicate serial numbers); however, at least 6 of these 100 cell phones must also be created using the copy constructor. The locations of these 6 phones in the array are up to you.

- Once the objects are created and placed in the array, change the price of all the objects. The price of any phone must now be set to a value between 100$ and 299$ inclusive. To simplify your task, we assume that all prices will be set to values that do not have any fractions (i.e. 214.00, 196.00, 284.00, etc.). You must use the **Random** class to set the prices. You can simply use a *for* loop and set the prices using a Random object.

- Display the contents of the array; however you only need to display the prices of the phones (not the rest of the attributes). This should look like the following:



```
Here are the current contents of the array; only prices of the cell phones are shown:

275.00      282.00      247.00      118.00      216.00      162.00      152.00      169.00      107.00      138.00
176.00      181.00      190.00      165.00      124.00      267.00      146.00      227.00      247.00      240.00
173.00      181.00      121.00      281.00      207.00      243.00      218.00      105.00      271.00      160.00
251.00      249.00      237.00      103.00      108.00      222.00      116.00      119.00      280.00      129.00
158.00      290.00      202.00      247.00      115.00      283.00      218.00      129.00      120.00      134.00
199.00      166.00      112.00      177.00      211.00      107.00      266.00      256.00      214.00      293.00
206.00      239.00      295.00      214.00      109.00      176.00      182.00      259.00      236.00      116.00
103.00      230.00      124.00      142.00      172.00      191.00      226.00      171.00      211.00      125.00
166.00      238.00      198.00      107.00      154.00      104.00      237.00      109.00      273.00      164.00
204.00      105.00      281.00      193.00      122.00      163.00      150.00      232.00      152.00      193.00
```

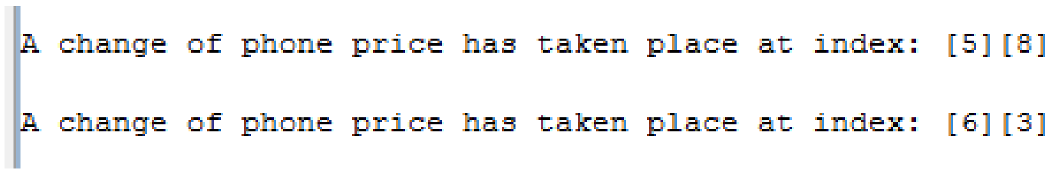
Figure 1 – Displaying original contents of the arrary –
Only prices of cell phones should be displayed

- Call the **ModifyPhonePrices()** method, which you created above, and pass to it the 2-dimenstional array, as well as 2 values. The passed values are up to you.

- Give an output message indicating how many changes are made.

- Display the array again (again, you are only displaying the prices), which will now show the modified array.

Here is an example that illustrates the requirements. Assume *arr* is the name of the 2-dimentional array that you created, and assume that its contents were as shown in Figure 1 above. Calling **ModifyPhonePrices(arr, 214, 196)** will result in the following displays by the method. Again, for simplicity, assume that all the passed values will not include any fractions; so calls similar to the one just indicated are okay (and actually, this is what you should use). Further; assume that the new given values will still be between 100$ and 299$ inclusive:

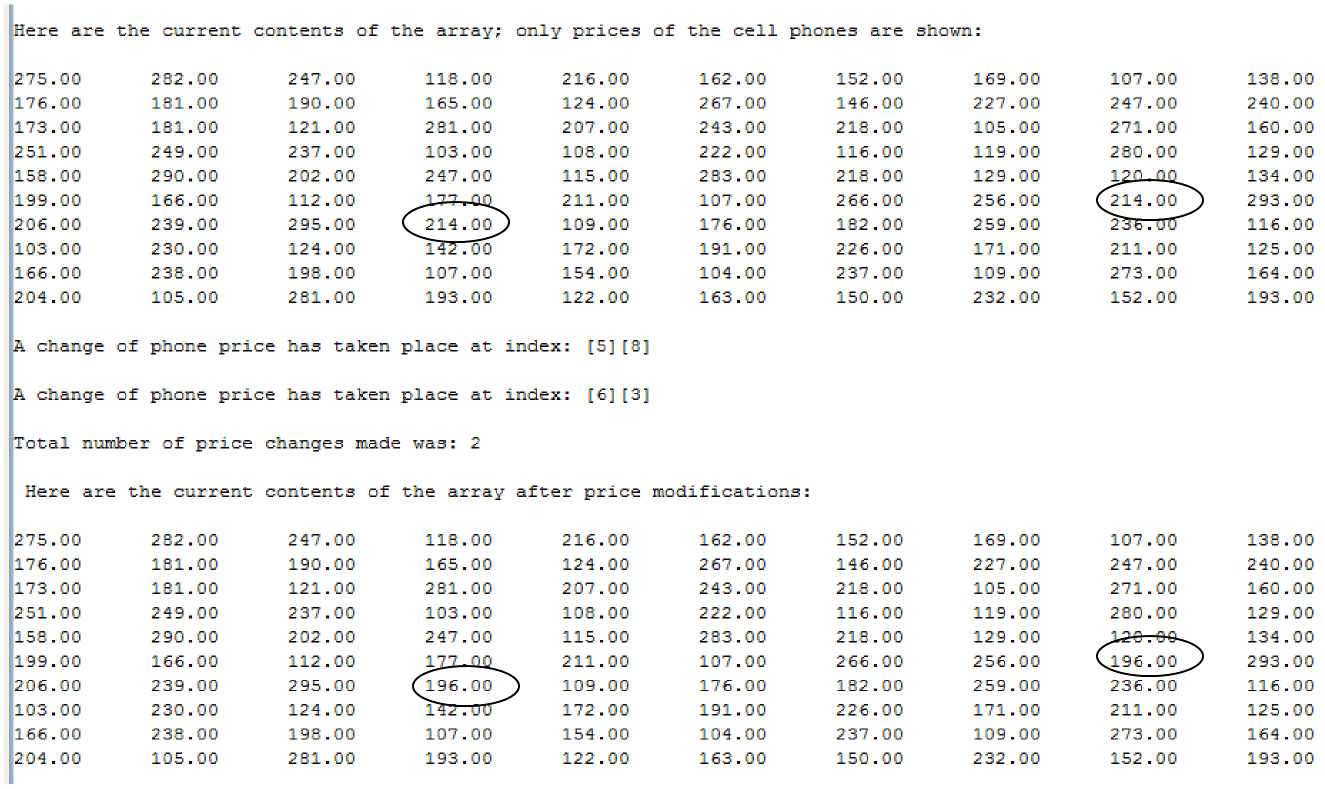


```
A change of phone price has taken place at index: [5][8]

A change of phone price has taken place at index: [6][3]
```

Figure 2 – Displaying locations where prices of cell phones have been modified

- Finally displaying the content of the array will consequently show the following (here is actually the full image of how the program should behave; the circled values are just for clarity to show where the changes were made):



```
Here are the current contents of the array; only prices of the cell phones are shown:

275.00      282.00      247.00      118.00      216.00      162.00      152.00      169.00      107.00      138.00
176.00      181.00      190.00      165.00      124.00      267.00      146.00      227.00      247.00      240.00
173.00      181.00      121.00      281.00      207.00      243.00      218.00      105.00      271.00      160.00
251.00      249.00      237.00      103.00      108.00      222.00      116.00      119.00      280.00      129.00
158.00      290.00      202.00      247.00      115.00      283.00      218.00      129.00      120.00      134.00
199.00      166.00      112.00      177.00      211.00      107.00      266.00      256.00      214.00      293.00
206.00      239.00      295.00      214.00      109.00      176.00      182.00      259.00      236.00      116.00
103.00      230.00      124.00      142.00      172.00      191.00      226.00      171.00      211.00      125.00
166.00      238.00      198.00      107.00      154.00      104.00      237.00      109.00      273.00      164.00
204.00      105.00      281.00      193.00      122.00      163.00      150.00      232.00      152.00      193.00

A change of phone price has taken place at index: [5][8]

A change of phone price has taken place at index: [6][3]

Total number of price changes made was: 2

 Here are the current contents of the array after price modifications:

275.00      282.00      247.00      118.00      216.00      162.00      152.00      169.00      107.00      138.00
176.00      181.00      190.00      165.00      124.00      267.00      146.00      227.00      247.00      240.00
173.00      181.00      121.00      281.00      207.00      243.00      218.00      105.00      271.00      160.00
251.00      249.00      237.00      103.00      108.00      222.00      116.00      119.00      280.00      129.00
158.00      290.00      202.00      247.00      115.00      283.00      218.00      129.00      120.00      134.00
199.00      166.00      112.00      177.00      211.00      107.00      266.00      256.00      196.00      293.00
206.00      239.00      295.00      196.00      109.00      176.00      182.00      259.00      236.00      116.00
103.00      230.00      124.00      142.00      172.00      191.00      226.00      171.00      211.00      125.00
166.00      238.00      198.00      107.00      154.00      104.00      237.00      109.00      273.00      164.00
204.00      105.00      281.00      193.00      122.00      163.00      150.00      232.00      152.00      193.00
```

Figure 3 – Illustrative example of program behavior

**Guideline:** For part 3, you will need to create one .java file, called ModifyCellPhones.java. The file will have an exact copy of the *CellPhone* class defined in Part 1, as well as the public class *ModifyCellPhones*. Inside that public class, you will have two *static* methods, the *ModifyPhonePrices*() method as well as the *main*() method.

⇨ As a general rule, displaying of the prices must be formatted using *printf*.

## Submitting Assignment 5

- IMPORTANT: You are allowed to work on a team of 2 students at most (including yourself!). Any teams of 3 or more students will result in 0 marks for all team members. If your work on a team, ONLY one copy of the assignment is to be submitted for both members.
- Zip together the source codes. (Please use WINZIP).
- Naming convention for zip file: Create one zip file, containing all source files for your assignment using the following naming convention:

  The zip file should be called *a#_studentID*, where # is the number of the assignment *studentID* is your student ID(s) number. For example, for the first assignment, student 12345678 would submit a zip file named *a1_12345678.zip*. If you work on a team and your IDs are 12345678 and 34567890, you would submit a zip file named *a1_12345678_34567890.zip*.
- Submit your zip file at: https://fis.encs.concordia.ca/eas/ as **Programming Assignment** and submission **#5**. Assignments submitted to the wrong directory would be discarded and no replacement submission will be allowed.

- Submit only ONE version of an assignment. If more than one version is submitted the first one will be graded and all others will be disregarded.

### **Evaluation Criteria for Assignment 5** (10 points)

| | |
|---|---|
| Comments - description of variables/ description of the steps in code/ purpose of program/ Indentation and readability of program/ Choice of variable names | 1 pts |
| Clarity of output | 0.5 pt |
| Part 1 | 1.5 pts |
| Part 2 (Achieving Requirements/Functionality) | 3 pts |
| Part 3 (Achieving Requirements/Functionality) | 4 pts |