Due: Skeleton Code (ungraded) **– Thursday, November 21, 2013 by 11:59 p.m.**
Completed Code – **Thursday, November 21, 2013 by 11:59 p.m.**

Updated: 11/18/2013
See highlighted text on pages 3 and 5.

Deliverables

Your project files should be submitted to Web-CAT by the due date and time specified above (see the Lab Guidelines for information on submitting project files). You may submit your skeleton code files until Wed 20 Nov 2013 11:59 p.m. to check for compilation but not for credit. You should do your best to get this done in lab on Wed or by the end of the Help session on Fri to ensure that you are making appropriate progress. You must submit your completed code files to Web-CAT no later than 11:59 PM on the due date for the completed code to avoid a late penalty for the project. You may submit your completed code up to 24 hours after the due date, but there is a late penalty of 15 points. No projects will be accepted after the one day late period. If you are unable to submit via Web-CAT, you should e-mail your project Java files in a zip file to your lab instructor before the deadline. The Completed Code will be tested against your test methods in your JUnit test files and against the usual correctness tests. The grade will be determined by the tests that you pass or fail and the level of coverage attained in your Java source files by your test methods.

Files to submit to Web-CAT:

From Project 9

- CellPhone.java, CellPhoneTest.java
- FlipPhone.java, FlipPhoneTest.java
- SmartPhone.java, SmartPhoneTest.java
- IPhone.java, IPhoneTest.java
- Android.java,AndroidTest.java

New in Project 10

- Provider.java, ProviderTest.java
- CellPhoneBillComparator.java, CellPhoneBillComparatorTest.java
- CellPhonesPart2.java, CellPhonesPart2Test.java

Recommendations

You should create new folder for Project 10 and copy your relevant Project 9 source and test files to it (i.e., do not include CellPhonesPart1.java, CellPhonesPart1Test.java). You should create a jGRASP project and add the class and test files as they are created. You may find it helpful to use the "viewer canvas" feature as you develop and debug your program.

Specifications

Overview: This project is Part 2 of three that will involve calculating the bills for cell phones. In Part 1, you developed Java classes that represent categories of cell phones including flip phones, generic smart phones, iPhones, and Android phones. In Part 2, you will implement three additional classes: (1) CellPhoneBillComparator that implements the Comparator interface, (2) Provider that represents a provider of cell phones services and includes several specialized methods, and (3)

CellPhonesPart2 which contains the main method for the program. Note that the main method in CellPhonesPart2 should create a Provider object, read the data file using the readCellPhonesFile method. CellPhonesPart2 then prints the summary, rates, cell phone list by number, cell phone list by billing amount, and the list of excluded records. You can use CellPhonesPart2 in conjunction with interactions by running the program in the canvas (or debugger with a breakpoint) until the Provider has been created and the data has been read in. You can then enter interactions in the usual way. You can also step into the methods of interest. In addition to the source files, you will create a JUnit test file for each class and write one or more test methods to ensure the classes and methods meet the specifications. I recommend that create a new Project 10 folder and copy your files from Project 9 into it. You should then create a jGRASP project and add the source and test files as they are created.

- **FlipPhone, SmartPhone, IPhone, and Android**

  Requirements and Design: No changes from the specifications in Project 9.

- **CellPhone.java**

  Requirements and Design: In addition to the specifications in Project 9, the CellPhone class should implement the Comparable interface for CellPhone objects.
  - `compareTo`: Takes a CellPhone as a parameter and returns an int indicating the results of comparing CellPhone objects based on their respective numbers. This method is required since the CellPhone class implements the Comparable interface for CellPhone.

- **Provider.java**

  Requirements: The Provider class provides methods for generating reports (summary, rates, list), adding a cell phone, deleting a cell phone, and sorting the cell phones by number and by billing amount.

  Design: The Provider class has fields, a constructor, and methods as outlined below.

(1) **Fields:** All fields below should be private.
(a) *name* is the name of the provider that is initialized to "not yet assigned".
(b) *phones* is an array of type CellPhone that is initialized to length zero.
(c) *excludedRecords* is a String array that is initialized to length zero.
Note that the phones array and excluded records array should grow and shrink as items are added and/or deleted. Hence the length of the array should be the same as the number of objects in the array. Imagine that we have lots processor capacity but not much memory.

(2) **Constructor:** The constructor has no parameters. The fields for name, phones, and excludedRecords can be initialized in the constructor or in the declaration of the fields.

(3) **Methods:** Usually a class provides methods to access and modify each of its instance variables (i.e., getters and setters) along with any other required methods. The methods for Provider are described below.

- `readCellPhoneFile` has no return value and accepts the data file name as a String. This method creates a Scanner object to read in the file and then reads it in line by line. The first line contains the provider name and each of the remaining lines contains the data for a cell phone. After reading in the provider name, the "cell phone" lines should be processed as follows. A cell phone line is read in, a second scanner is created on the line, and the individual values for the cell phone are read in. After the values on the line have been read in, an "appropriate" cell phone object is created and added to the phones array using the addPhone method. If the cell phone type is not recognized, the record/line should be added to the excluded records array using the addExcludedRecord method. The data file is a "comma separated values" file; i.e., if a line contains multiple values, the values are delimited by commas. So when you set up the scanner for the cell phone lines, you need to set the delimiter to use a ",". Each cell phone line in the file begins with a category for the cell phone. Your switch statement should determine which type of CellPhone to create based on the first character of the category (i.e., **F**, **S**, **I**, and **A** for **F**lipPhone, **S**martPhone, **I**Phone, and **A**ndroid respectively). The second field in the record is the phone number, followed by the values for texts, minutes, data, iMessages, and hotspot minutes as appropriate for the category of phone. That is, the items that follow minutes correspond to the data needed for the particular category (or subclass) of CellPhone. An example file, *provider1.dat*, is available for download from the course web site. Below are example data records (the first line/record containing the provider name is followed by cell phone lines/records):

```
AU Cellular Service
FlipPhone,111-243-5948,100,50
BrickPhone,111-534-5948,100,50
FlipPhone,111-342-7544,34,955
Android,111-934-9939,500,400,1000,30
Iphone,111-123-4567,20,548,220,55
SmartPhone(Windows),111-131-3131,40,21,10
```

- `getName` returns the String representing the name.
- `setName` returns nothing, accepts a String and assigns it to the name field.
- `getPhones` returns the CellPhone array representing the cell phones.
- `getExcludedRecords` returns the String array representing the excludedRecords.
- `addPhone` has no return value, accepts a CellPhone, increases the capacity of the phones array by one, and adds the new phone in the last position of the phones array.
- `deletePhone` has a boolean return value, accepts a String representing a phone number, searches the phones array for the number; if found, decreases the capacity of the phones array by one, and deletes the cell phone with the matching number in the phones array. If a cell phone is deleted, return true; otherwise return false. No more than one cell phone should be deleted each time this method is invoked.
- `addExcludedRecord` has no return value, accepts a String, increases the capacity of the excludedRecords array by one, and adds the String in the last position of the excludedRecords array.
- `toString` returns a String representing a list of cell phones in the phones array; accepts no parameters.
- `calculateTotalBill` returns a double representing the total bill for all of the phones in the phones array.

- `calculateTotalTexts` returns an int representing the total texts for all of the phones in the phones array.
- `calculateTotalMinutes` returns an int representing the total talk minutes for all of the phones in the phones array.
- `calculateTotalData` returns an int representing the total data for all of the smart phones in the phones array.
- `calculateTotalHotspotMin` returns an int representing the total hotspot minutes for all of the android phones in the phones array.
- `calculateTotalIMessages` returns an int representing the total iPhone messages for all of the iPhones in the phones array.
- `summary` returns a String representing summary information for all of the cell phones in the phones array including the number of phones and totals for texts, minutes, data, hotspot minutes, iMessages, and bills. Note that this method should call the "calculate" methods described. See example below.
- `rates` returns a String representing the cell phone rates represented by the constants in the cell phone classes. See the example below.
- `listByNumber` returns a String representing the cell phone list by number. The phone array should sorted by number before building the String to be returned. The resulting String should include the title and list of cell phones as shown in the output example below.
- `listByBill` returns a String representing the cell phone list by the billing amount. The phone array should sorted by billing amount before building the String to be returned. The resulting String should include the title and list of cell phones as shown in the output example below.
- `excludedRecordsList` returns a String representing the list cell phone records/lines that were read from the file but excluded from the phones array in Provider. The resulting String should include the title and list of excluded records/lines as shown in the output example below.

Code and Test: See examples of file reading and sorting in the lecture notes. The Array.sort method in the java.util package sorts the array in place. The natural sorting order for CellPhone objects is determined by the compareTo method from the Comparable interface. If *phones* is the variable for the array of CellPhone objects, it can be sorted in natural order with the following statement.

```
Arrays.sort(phones);
```

The sorting order based on billing amount is determined by the CellPhoneBillComparator class which implements the Comparator interface (described below). It can be sorted with the following statement.

```
Arrays.sort(phones, new CellPhoneBillComparator());
```

After the phones array is sorted, the array returned by the getPhones method should be in the order resulting from the most recent sort.

- **CellPhoneBillComparator.java**

  Requirements and Design: The CellPhoneBillComparator class implements the Comparator interface for CellPhone objects. Hence, it implements the following method.

  - `compare(CellPhone c1, CellPhone c2)` that defines the ordering from lowest to highest based on the bill for c1 and c2.

    Note that the *compare* method is the only method in the CellPhoneBillComparator class. An instance of this class should be used as one of the parameters when the Arrays.sort method is used to sort by "bill" (see above). For an example of a class implementing Comparator, see lecture notes 10B Comparing Objects.

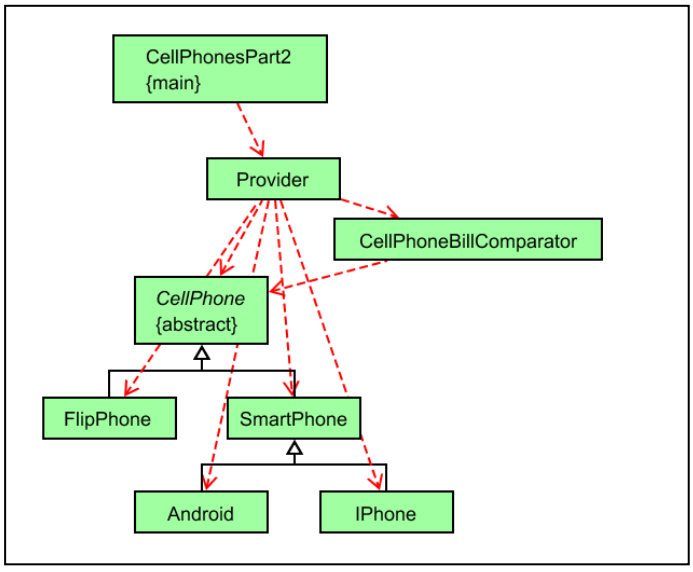
- **CellPhonesPart2.java**

  Requirements and Design: The CellPhonesPart2 class has only a main method as described below.

  - `main` reads in the file name from the command line (i.e., args[0]), creates an instance of Provider, and then calls its readCellPhonefile method to read in the data file to populate the Provider object. The main method then prints the summary, rates, cell phone list by number, cell phone list by billing amount, and the list of excluded records. An example data file, *provider1.dat*, can be downloaded from the Lab web page. The output from main for this file is on the following page.

  Code and Test: The example data file, *provider1.dat*, has been uploaded into Web-CAT and is available for your test methods to call as needed. Web-CAT submission appears to be filtering files with a ".dat" extension. If you want to use data files other than provider1.dat, you'll need to use .txt or .csv as the extension and then upload the data files along with your source files.

UML Class Diagram



Example Output

```
 ----jGRASP exec: java -ea CellPhonesPart2 provider1.dat

------------------------------
Summary for AU Cellular Service
------------------------------
Number of cell phones: 5
Texts: 694
Talk Minutes: 1974
Data: 1230
Hotspot Minutes: 30
iMessages: 55
Bill Total: $664.40

------------------------------
Rates for AU Cellular Service
------------------------------
FlipPhone Talk Rate: $0.15   Text Rate: $0.25
SmartPhone Talk Rate: $0.10   Text Rate: $0.50   Max Talk Time: 600.0
   iPhone iMessage Rate: $0.35
   Android Hotspot Rate: $0.75

------------------------------
Cell Phones by Number
------------------------------

Number: 111-123-4567 (IPhone)
Bill: $95.05 for 20 Texts, 548 Talk Minutes, 220 MB of Data, 55 iMessages

Number: 111-131-3131 (SmartPhone)
Bill: $22.60 for 40 Texts, 21 Talk Minutes, 10 MB of Data

Number: 111-243-5948 (FlipPhone)
Bill: $32.50 for 100 Texts, 50 Talk Minutes

Number: 111-342-7544 (FlipPhone)
Bill: $151.75 for 34 Texts, 955 Talk Minutes

Number: 111-934-9939 (Android)
Bill: $362.50 for 500 Texts, 400 Talk Minutes, 1000 MB of Data, 30 Hotspot Minutes


------------------------------
Cell Phones by Billing Amount
------------------------------

Number: 111-131-3131 (SmartPhone)
Bill: $22.60 for 40 Texts, 21 Talk Minutes, 10 MB of Data

Number: 111-243-5948 (FlipPhone)
Bill: $32.50 for 100 Texts, 50 Talk Minutes

Number: 111-123-4567 (IPhone)
Bill: $95.05 for 20 Texts, 548 Talk Minutes, 220 MB of Data, 55 iMessages

Number: 111-342-7544 (FlipPhone)
Bill: $151.75 for 34 Texts, 955 Talk Minutes
```

```
Number: 111-934-9939 (Android)
Bill: $362.50 for 500 Texts, 400 Talk Minutes, 1000 MB of Data, 30 Hotspot Minutes


------------------------------
Excluded Records
------------------------------

BrickPhone,111-534-5948,100,50


 ----jGRASP: operation complete.
```