Project: Bakery – Part 2

Deliverables

Your project files should be submitted to Web-CAT by the due date and time specified. Note that there is also an optional Skeleton Code assignment which will indicate level of coverage your tests have achieved (there is no late penalty since the skeleton code assignment is ungraded for this project). The files you submit to skeleton code assignment may be incomplete in the sense that method bodies have at least a return statement if applicable or they may be essentially completed files. To avoid a late penalty for the project, you must submit your completed code files to Web-CAT no later than 11:59 PM on the due date for the completed code assignment. If you are unable to submit to Web-CAT directly or via jGRASP, you should e-mail your project Java files in a zip file to your TA before the deadline. Test files are **not** required for this project. If submitted, you will be able to see your code coverage, but this will not be counted as part of your grade.

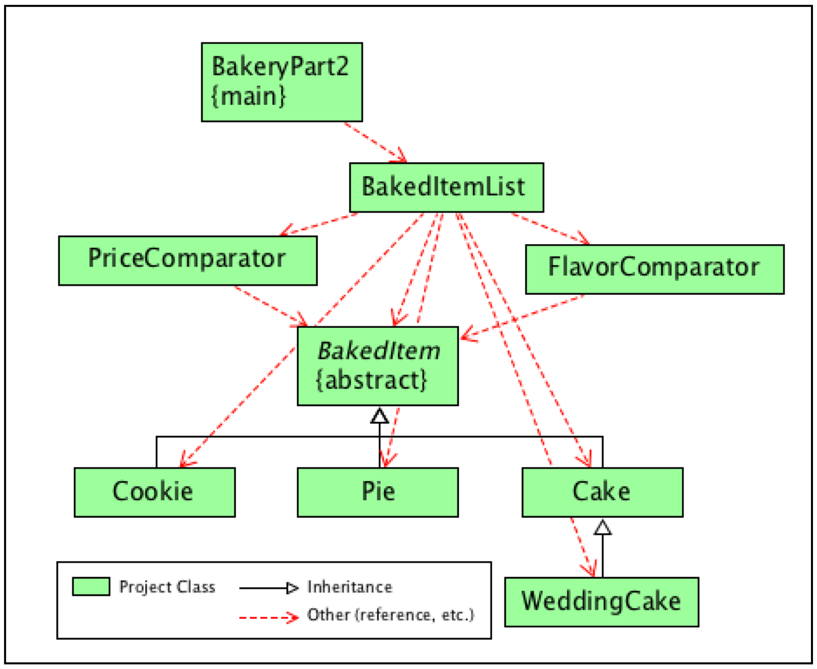
Files to submit to Web-CAT (*test files are optional*):

From Part 1

- BakedItem.java
- Cookie.java, *CookieTest.java*
- Pie.java, *PieTest.java*
- Cake.java, *CakeTest.java*
- WeddingCake.java, *WeddingCakeTest.java*

New in Part 2

- PriceComparator.java, *PriceComparatorTest.java*
- FlavorComparator.java, *FlavorComparatorTest.java*
- BakedItemList.java, *BakedItemListTest.java*
- BakeryPart2.java, *BakeryPart2Test.java*



Recommendations

You should create new folder for Part 2 and copy your relevant Part 1 source and optional test files to it. You should create a jGRASP project with these files in it, and then add the new source and optional test files as they are created.

Specifications – Use arrays in this project; ArrayLists are not allowed!

Overview: This project is Part 2 of three that involves a bakery and reporting for baked items. In Part 1, you developed Java classes that represent categories of baked items including cookies, pies, cakes, and wedding cakes. In Part 2, you will implement four additional classes: (1) PriceComparator, which implements the Comparator interface to compare baked items by price; (2) FlavorComparator, which implements the Comparator interface to compare baked items by flavor; (3)

BakeryItemList, which represents a list of baked items and includes several specialized methods; and (4) BakeryPart2, which contains the main method for the program. Note that the main method in BakeryPart2 should declare a BakeryItemList object and then call the readItemFile method which will add baked items to the BakeryItemList as the data is read in from a file. You can use BakeryPart2 in conjunction with interactions by running the program in the canvas (or debugger with a breakpoint) and single stepping until the variables of interest are created. You can then enter interactions in the usual way. 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. You should create a jGRASP project upfront and then add the source and test files as they are created. All of your files should be in a single folder.

- **Cookie, Pie, Cake, and WeddingCake**

  Requirements and Design: No changes from the specifications in Part 1.

- **BakedItem.java**

  Requirements and Design: In addition to the specifications in Part 1, the BakedItem class should implement the Comparable interface for BakedItem, which means the following method must be implemented in BakedItem.
  - `compareTo`: Takes a BakedItem as a parameter and returns an int indicating the ordering from lowest to highest based on the class name followed *name*, and *flavor*. Since this is essentially the same as the toString value, you could do the compareTo based on toString value. Otherwise, to do this, you will need to extract the class name, *name*, and *flavor* from the BakedItem and do the compareTo based these. Regardless of the method you use, to avoid uppercase and lowercase issues you should make the String values either all uppercase or all lowercase prior to comparing them.

- **BakedItemList.java**

  Requirements: The BakedItemList class provides methods for reading in the data file and generating reports.

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

  (1) **Fields:** All fields below should be private.
  (a) *listName* is a String that represents the name of the BakedItemList.
  (b) *itemList* is an array that can hold up to 100 BakedItem objects.
  (c) *itemCount* is an int that tracks the number of BakedItem objects in the *itemList* array.
  (d) *excludedRecords* is an array that can hold up to 30 String elements representing records that are read from the data file but not processed.
  (e) *excludedCount* is an int that tracks the number of records that have been added to the excludedRecords array.
  (f) listCount is a static field of type int initialized to zero, which tracks the number of BakedItemList objects created.

(2) **Constructor:** The constructor has no parameters, initializes the instance fields in BakedItemList as follows:

```
        listName = "";
        itemList = new BakedItem[100];
        itemCount = 0;
        excludedRecords = new String[30];
        excludedCount = 0;
```

The constructor should also increment the static field listCount. The readItemFile method defined below will populate the BakedItemList object.

(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 BakedItemList are described below.

- `getListName` returns the String representing the listName.
- `setListName` has no return value, accepts a String, and then assigns it to listName.
- `getItemList` returns the BakedItem array representing the *itemList*.
- `setItemList` has no return value, accepts a BakedItem array, and then assigns it to *itemList* (used in conjunction with `setItemCount` to provide a new BakedItem array and *itemCount*).
- `getItemCount` returns the current value of *itemCount*.
- `setItemCount` has no return value, accepts an int, and assigns it to *itemCount*.
- `getExcludedRecords` returns the String array representing the excludedRecords.
- `setExcludedRecords` has no return value, accepts a String array, and then assigns it to excludedRecords (used in conjunction with `setExcludedCount` to provide a new excluded records array and new excluded records count).
- `getExcludedCount` returns the current value of excludedCount.
- `setExcludedCount` has no return value, accepts an int, and sets excludedCount to it.
- `getListCount` is a static method that returns the current value of listCount.
- `resetListCount` is a static method that has no return value and sets listCount to 0.
- `readItemFile` has no return value and accepts the data file name as a String. Remember to include the `throws FileNotFoundException` clause in the method declaration. This method creates a Scanner object to read in the file, and then reads the file line by line. The first line contains the BakedItemList name and each of the remaining lines contains the data for a baked item. After reading in the list name, the “baked item” lines should be processed as follows. A baked item line is read in, a second scanner is created on the line, and the individual values for the baked item are read in. After the values on the line have been read in, an “appropriate” BakedItem object is created. 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 baked item lines, you need to set the delimiter to use a “,” by calling the `useDelimiter(",")` method on the Scanner object. Each baked item line in the file begins with a category for the baked item (C, P, K, and W are valid categories for baked items indicating **C**ookie, **P**ie, **K** for Cake, and **W**eddingCake respectively). The second field in the record is the name, followed by flavor, and quantity. The next field(s) may correspond, as appropriate, to the data needed for the particular category (or subclass) of BakedItem (e.g., a Pie has a crustCost and a Cake has layers). The last fields are the ingredients, which should be stored in an array (new String[50]) as they are read in. After all ingredients have been read in, use the java.util.Arrays copyOf method make a new

ingredients array with length equal to the number of ingredients. If the line/record has a valid category and it has been successfully read in, the appropriate BakedItem object should be created and added to itemList. If an item line has an invalid category, no BakedItem is created and the line should be added to the excluded records array with the String `"*** invalid category *** for line:\n"` appended to the front of it, and the excluded count should be incremented. The file *baked_item_data.csv* is available for download from the course web site. Below are example data records (the first line/record containing the BakedItemList name is followed by baked item lines/records):

```
Auburn's Best Bakery
C,Chips Delight,Chocolate Chip,12,flour,sugar,dark chocolate chips,butter,baking soda,salt
D,Chips Delight,Chocolate Chip,12,flour,sugar,dark chocolate chips,butter,baking soda,salt
P,Weekly Special,Apple,1,0,flour,sugar,apple,cinnamon,butter,baking soda,salt
R,Weekly Special,Apple,1,0,flour,sugar,apple,cinnamon,butter,baking soda,salt
P,Summer Special,Key Lime,1,2.0,flour,sugar,lime juice,lemon juice,graham crackers,butter,baking soda,salt
K,Birthday,Chocolate,1,1,flour,sugar,cocoa powder,vanilla,eggs,butter,baking soda,baking powder,salt
K,2-Layer,Red Velvet,1,2,flour,sugar,cocoa powder,food coloring,eggs,butter,baking soda,baking powder,salt
W,3-Layer/3-Tier,Vanilla,1,3,3,flour,sugar,buttermilk,coffee,eggs,butter,baking soda,baking powder,salt
```

- `generateReport` processes the item list array using the <u>original order</u> from the file to produce the bakery Report which is returned as a String. See the example output on pages 7 - 8.
- `generateReportByClass` makes a copy of the item list array and sorts the copy using the <u>natural ordering</u>, then processes the sorted array to produce the bakery Report (by Class) which is returned as a String. See the example output on pages 7 - 9.
- `generateReportByPrice` makes a copy of the item list array and sorts the copy <u>by item price</u>, then processes the sorted array to produce the bakery Report (by Price) which is returned as a String. See the example output on pages 7 - 8.
- `generateReportByFlavor` makes a copy of the item list array and sorts the copy <u>by flavor</u>, then processes the sorted array to produce the bakery Report (by Flavor) which is returned as a String. See the example output on pages 7 - 8.
- `generateExcludedRecordsReport` processes the excludedRecords array to produce the Excluded Records Report which is returned as a String. See the example output on pages 7 - 8.

Code and Test: See examples of file reading and sorting (using Arrays.sort) in the class notes. The natural sorting order for BakedItem objects is determined by the compareTo method from the Comparable interface. If *itemList* is the variable for the array of BakedItem objects, it can be sorted with the following statement.

```
BakedItem[] bList = Arrays.copyOf(itemList, itemCount);
Arrays.sort(bList);
```

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

```
BakedItem[] bList = Arrays.copyOf(itemList, itemCount);
Arrays.sort(bList, new PriceComparator());
```

The sorting order based on boarding cost is determined by the FlavorComparator class which implements the Comparator interface (described below). It can be sorted with statements similar to those above, except that a new FlavorComparator is passed to the Arrays.sort method.

- **PriceComparator.java**

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

  - `compare(BakedItem b1, BakedItem b2)` that defines the ordering from <u>lowest to highest</u> based on the price of the baked item.

  Note that the *compare* method is the only method in the PriceComparator class. An instance of this class will be used as one of the parameters when the Arrays.sort method is used to sort by "price". For an example of a class implementing Comparator, see class notes on Comparing Objects.

- **FlavorComparator.java**

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

  - `compare(BakedItem b1, BakedItem b2)` that defines the ordering from <u>lowest to highest</u> (i.e., alphabetical order, ignoring case) based on the flavor of each baked item.

  Note that the *compare* method is the only method in the FlavorComparator class. An instance of this class will be used as one of the parameters when the Arrays.sort method is used to sort by "flavor". For an example of a class implementing Comparator, see class notes on Comparing Objects.

- **BakeryPart2.java**

  Requirements and Design: The BakeryPart2 class has only a main method as described below.

  - `main` reads in the file name as the first argument, args[0], of the command line arguments, creates an instance of BakedItemList, and then calls the readItemFile method in the BakedItemList class to read in the data file and generate the five reports as shown in the output examples beginning on page 7. Since main invokes the readItemFile method, which throws the FileNotFoundException, main should also throw FileNotFoundException. If no command line argument is provided (i.e., `args.length == 0`), the program should indicate this and end as shown in the first example output on page 7. An example data file, *baked_item_data.csv* can be downloaded from the Lab assignment web page, and this file will be available in Web-CAT for use with your test files. Any .csv data files that you create for your test methods will need to be uploaded to Web-CAT along with your source and test files.

Code and Test: In your JUnit test file for the BakeryPart2 class, you should have at least two test methods for the main method. One test method should invoke BakeryPart2.main(args) where args is an empty String array, and the other test method should invoke BakeryPart2.main(args) where args[0] is the String representing the data file name. Depending on how you implemented the main method, these two test methods should cover the code in main. In the first test method, you should reset *listCount*, call your main method, then assert that `getListCount()` is zero for the test with no file name. In the second test method, you should reset *listCount*, call your main method assert that `getListCount()` is one with *baked_item_data.csv* as the file name passed in as args[0].

In the first test method, you can invoke main with no command line argument as follows:

```
// You should be checking for args.length == 0
// in BakeryPart2, and the following should exercise
// the code for true.
BakedItemList.resetListCount();
String[] args1 = {};  // an empty String[]
BakeryPart2.main(args1);
Assert.assertEquals("Baked Item List count should be 0. ",
                    0, BakedItemList.getListCount());
```

In the second test method, you can invoke main as follows with the file name as the first (and only) command line argument:

```
BakedItemList.resetListCount();
String[] args2 = {"baked_item_data.csv"};
// element args2[0] is the file name
BakeryPart2.main(args2);
Assert.assertEquals("Baked Item List count should be 1. ",
                    1, BakedItemList.getListCount());
```

If Web-CAT complains that the default constructor for `BakeryPart2` has not been covered, you should include the following line of code in one of your test methods.

```
// to exercise the default constructor
BakeryPart2 app = new BakeryPart2();
```

Example Output when no file name is provided as a command line argument

```
 ----jGRASP exec: java BakeryPart2
File name expected as command line argument.
Program ending.

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

Example Output when *baked_item_data.csv* is the command line argument

```
 ----jGRASP exec: java BakeryPart2 baked_item_data.csv

---------------------------------------
Report for Auburn's Best Bakery
---------------------------------------

Cookie: Chips Delight - Chocolate Chip   Quantity: 12   Price: $4.20
(Ingredients: flour, sugar, dark chocolate chips, butter, baking soda,
salt)

Pie: Weekly Special - Apple   Quantity: 1   Price: $12.00
(Ingredients: flour, sugar, apple, cinnamon, butter,
baking soda, salt)

Pie: Summer Special - Key Lime   Quantity: 1   Price: $14.00
(Ingredients: flour, sugar, lime juice, lemon juice, graham crackers,
butter, baking soda, salt)

Cake: Birthday - Chocolate   Quantity: 1   Price: $8.00
(Ingredients: flour, sugar, cocoa powder, vanilla, eggs,
butter, baking soda, baking powder, salt)

Cake: 2-Layer - Red Velvet   Quantity: 1   Price: $16.00
(Ingredients: flour, sugar, cocoa powder, food coloring, eggs,
butter, baking soda, baking powder, salt)

WeddingCake: 3-Layer/3-Tier - Vanilla   Quantity: 1   Price: $135.00
(Ingredients: flour, sugar, buttermilk, coffee, eggs,
butter, baking soda, baking powder, salt)


---------------------------------------
Report for Auburn's Best Bakery (by Class)
---------------------------------------

Cake: 2-Layer - Red Velvet   Quantity: 1   Price: $16.00
(Ingredients: flour, sugar, cocoa powder, food coloring, eggs,
butter, baking soda, baking powder, salt)

Cake: Birthday - Chocolate   Quantity: 1   Price: $8.00
(Ingredients: flour, sugar, cocoa powder, vanilla, eggs,
butter, baking soda, baking powder, salt)

Cookie: Chips Delight - Chocolate Chip   Quantity: 12   Price: $4.20
(Ingredients: flour, sugar, dark chocolate chips, butter, baking soda,
salt)

Pie: Summer Special - Key Lime   Quantity: 1   Price: $14.00
(Ingredients: flour, sugar, lime juice, lemon juice, graham crackers,
butter, baking soda, salt)

Pie: Weekly Special - Apple   Quantity: 1   Price: $12.00
(Ingredients: flour, sugar, apple, cinnamon, butter,
baking soda, salt)

WeddingCake: 3-Layer/3-Tier - Vanilla   Quantity: 1   Price: $135.00
(Ingredients: flour, sugar, buttermilk, coffee, eggs,
butter, baking soda, baking powder, salt)
```

```

---------------------------------------
Report for Auburn's Best Bakery (by Price)
---------------------------------------

Cookie: Chips Delight - Chocolate Chip   Quantity: 12   Price: $4.20
(Ingredients: flour, sugar, dark chocolate chips, butter, baking soda,
salt)

Cake: Birthday - Chocolate   Quantity: 1   Price: $8.00
(Ingredients: flour, sugar, cocoa powder, vanilla, eggs,
butter, baking soda, baking powder, salt)

Pie: Weekly Special - Apple   Quantity: 1   Price: $12.00
(Ingredients: flour, sugar, apple, cinnamon, butter,
baking soda, salt)

Pie: Summer Special - Key Lime   Quantity: 1   Price: $14.00
(Ingredients: flour, sugar, lime juice, lemon juice, graham crackers,
butter, baking soda, salt)

Cake: 2-Layer - Red Velvet   Quantity: 1   Price: $16.00
(Ingredients: flour, sugar, cocoa powder, food coloring, eggs,
butter, baking soda, baking powder, salt)

WeddingCake: 3-Layer/3-Tier - Vanilla   Quantity: 1   Price: $135.00
(Ingredients: flour, sugar, buttermilk, coffee, eggs,
butter, baking soda, baking powder, salt)


---------------------------------------
Report for Auburn's Best Bakery (by Flavor)
---------------------------------------

Pie: Weekly Special - Apple   Quantity: 1   Price: $12.00
(Ingredients: flour, sugar, apple, cinnamon, butter,
baking soda, salt)

Cake: Birthday - Chocolate   Quantity: 1   Price: $8.00
(Ingredients: flour, sugar, cocoa powder, vanilla, eggs,
butter, baking soda, baking powder, salt)

Cookie: Chips Delight - Chocolate Chip   Quantity: 12   Price: $4.20
(Ingredients: flour, sugar, dark chocolate chips, butter, baking soda,
salt)

Pie: Summer Special - Key Lime   Quantity: 1   Price: $14.00
(Ingredients: flour, sugar, lime juice, lemon juice, graham crackers,
butter, baking soda, salt)

Cake: 2-Layer - Red Velvet   Quantity: 1   Price: $16.00
(Ingredients: flour, sugar, cocoa powder, food coloring, eggs,
butter, baking soda, baking powder, salt)

WeddingCake: 3-Layer/3-Tier - Vanilla   Quantity: 1   Price: $135.00
(Ingredients: flour, sugar, buttermilk, coffee, eggs,
butter, baking soda, baking powder, salt)


---------------------------------------
Excluded Records Report
---------------------------------------

*** invalid category *** for line:
D,Chips Delight,Chocolate Chip,12,flour,sugar,dark chocolate chips,butter,baking soda,salt
*** invalid category *** for line:
R,Weekly Special,Apple,1,0,flour,sugar,apple,cinnamon,butter,baking soda,salt

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

Hints

1. In the methods that generate the reports, return value should begin with a \n and end with \n. The beginning \n is part of the heading, and the ending \n is part of the line for last BakedItem appended to the return value in the list. Note that you need to use a loop to go through the array of BakedItem objects to add each to the report. Each BakedItem added to the report should begin with \n and end with \n.

2. In the main method, use println, rather than print, to print each report. You should not need to add any \n characters before or after the report method is invoked within the println statement. For example, the following statement could be used to generate the first report where bList is a BakedList object:

   ```
   System.out.println(bList.generateReport());
   ```

3. If a baked item line has an invalid category, add "*** invalid category ***\n" plus the entire line read in from the file to the excluded record array.
   For example, the following record has an invalid category (D).

   ```
   D,Chips Delight,Chocolate Chip,12,flour,sugar,dark chocolate chips,butter,baking soda,salt
   ```

   This would result in the following two lines when the excluded record is eventually printed as part of the report.

   ```
   *** invalid category *** for line:
   D,Chips Delight,Chocolate Chip,12,flour,sugar,dark chocolate chips,butter,baking soda,salt
   ```