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 this week 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. In order 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. 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 TA 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, in part, 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 Bakery – Part 1

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

From Bakery – Part 2

- PriceComparator.java, PriceComparatorTest.java
- FlavorComparator.java, FlavorComparatorTest.java
- BakedItemList.java, BakedItemListTest.java [modified as described below]

New in Bakery – Part 3

- InvalidCategoryException.java
- BakeryPart3.java, BakeryPart3Test.java

Recommendations

You should create a new folder for Project 11 and copy your relevant Project 10 source and test files to it. You should create a jGRASP project and add the class and test files as they are created.

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

Overview: This project is Part 3 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 implemented 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. In Part 3, you are to add exception handling. You will need to do the following: (1) create a new class named InvalidCategoryException which extends the Exception class, (2) add try-catch statements to catch a

IOException in the main method of the BakeryPart3 class, and (3) modify the readItemFile method in the BakedItemList class to catch/handle an InvalidCategoryException and a NumberFormatException in the event that either type of exception is thrown while reading the input file.

Note that the main method in BakeryPart3 should create a BakedItemList object and then invoke the readItemFile method on that BakedItemList object to read data from a file and add baked items to the list. You can use BakeryPart3 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 the indicated source files 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 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 Bakery – Part 1.

- **InvalidCategoryException.java**

  Requirements and Design: InvalidCategoryException is a user defined exception created by extending the Exception class. This exception is to be thrown and caught in the readItemFile method in the BakedItemList class when a line of input data contains an invalid baked item category. The constructor for InvalidCategoryException takes a single String parameter representing *category* and invokes the super constructor with the following String:

  ```
  "For category: " + "\"" + category + "\""
  ```

  This string will be the toString() value of an InvalidCategoryException when it occurs. For a similar constructor, see InvalidLengthException.java in Examples\Polygons from the class notes on Exceptions.

- **BakedItemList.java**

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

  Design: In addition to the specifications in Bakery – Part 1, the existing readItemFile method must be modified to catch the following: InvalidCategoryException, NoSuchElementException, and NumberFormatException.

  - `readItemFile` has no return value and accepts the data file name as a String. Remember to include the `throws IOException` 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. For each *incorrect* line scanned (i.e., a line of data contains an invalid category, invalid numeric data, or missing data), your method will need to handle the invalid items properly.

- If the line of data begins with an invalid category, your program should throw an InvalidCategoryException (see description above). The code that adds a record with an invalid baked item category to the excluded records array should be placed in the catch block for InvalidCategoryException.
- If a line of data has a valid category, but includes invalid numeric data (e.g., the value for *quantity* contains an alphabetic character), a NumberFormatException (see notes on last page) will be thrown automatically by the Java Runtime Environment (JRE).
- If a line of data has a valid category but has missing data, a NoSuchElementException will be thrown automatically by the JRE.

The code in the while loop that reads in the baked item records and checks for baked item category should be in a try statement followed by three catch blocks: one for each of InvalidCategoryException, NumberFormatException, and NoSuchElementException. In each catch block, a String object should be created consisting of

```
"*** " + e + " in:\n" + line
```

where *e* is the exception and *line* is the record with the invalid data. The String object should be added to the excludedRecords array.

The file *baked_item_data2.csv* is available for download from the course web site. Below are example data records (the first line/record containing the baked item list name is followed by baked item lines/records). Note that five of these baked item records will cause an exception to be thrown. These are listed in the "Excluded Records Report" on page 6.

```
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
P,Daily Special,Pecan,1.0,0,flour,sugar,pecans,dark corn syrup,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
K,2-Layer,Chocolate,1,two,flour,sugar,cocoa powder,eggs,butter,baking soda,baking powder,salt
W,3-Layer/3-Tier,Buttermilk,1,3
W,3-Layer/3-Tier,Vanilla,1,3,3,flour,sugar,buttermilk,coffee,eggs,butter,baking soda,baking powder,salt
```

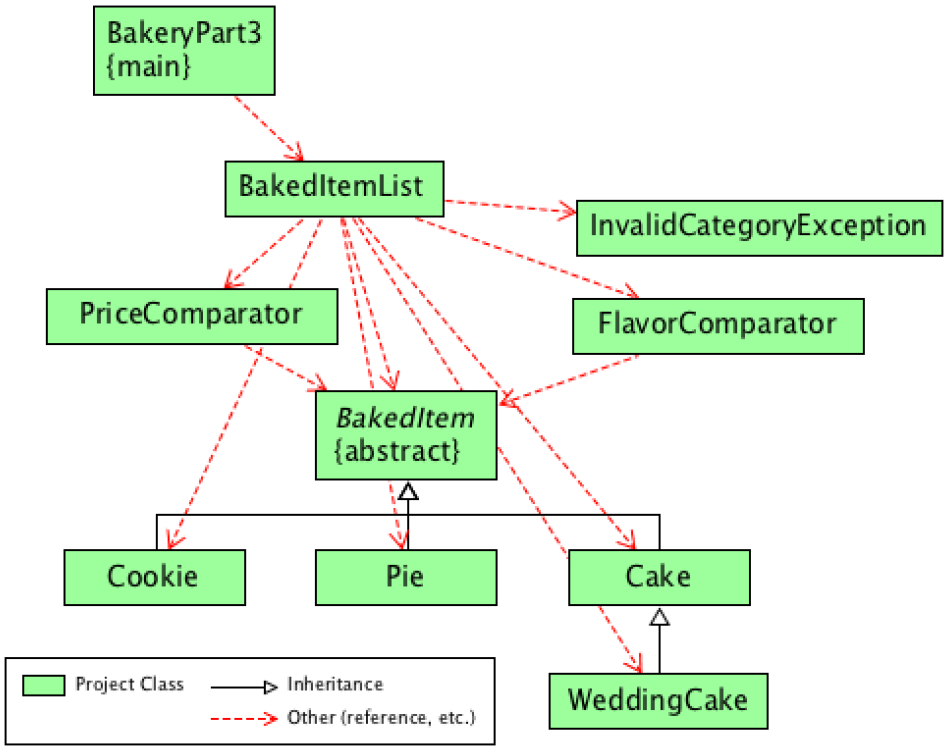
BakeryPart3.java

Requirements and Design: The BakeryPart3 class has only a main method as described below. In addition to the specifications in Bakery – Part 2, the main method should be modified to catch and handle an IOException if one is thrown in the readItemFile method.

- As before, the `main` method 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 the next page. The main method should not include the *throws IOException* in the declaration. Instead, the main method should include a try-catch statement to catch IOException when/if it is thrown in the readItemFile method in the BakedItemList class. This exception is most likely to occur when an incorrect file name is passed to the readItemFile method. After this exception is caught and the appropriate message is printed in the main method, your program should end. See the second example output on the following page.

  Example data files can be downloaded from the Lab web page, and the program output for *baked_item_data2.csv* begins on the next page.

UML Class Diagram



Example Output when file name is missing as command line argument

```
 ----jGRASP exec: java BakeryPart3
File name expected as command line argument.
Program ending.

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

Example Output when attempting to read a file that is not found (fake_data.csv)

```
 ----jGRASP exec: java BakeryPart3 fake_data.csv
Attempted to read file: fake_data.csv (No such file or directory)
Program ending.

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

Example Output for *baked_item _data2.csv*

```
 ----jGRASP exec: java BakeryPart3 baked_item_data2.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
---------------------------------------

*** InvalidCategoryException: For category: "D" in:
D,Chips Delight,Chocolate Chip,12,flour,sugar,dark chocolate chips,butter,baking soda,salt
*** java.lang.NumberFormatException: For input string: "1.0" in:
P,Daily Special,Pecan,1.0,0,flour,sugar,pecans,dark corn syrup,butter,baking soda,salt
*** InvalidCategoryException: For category: "R" in:
R,Weekly Special,Apple,1,0,flour,sugar,apple,cinnamon,butter,baking soda,salt
*** java.lang.NumberFormatException: For input string: "two" in:
K,2-Layer,Chocolate,1,two,flour,sugar,cocoa powder,eggs,butter,baking soda,baking powder,salt
*** java.util.NoSuchElementException in:
W,3-Layer/3-Tier,Buttermilk,1,3

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

Notes:

This project assumes that you are reading each double value as a String using next() and then parsing it into a double with Double.parseDouble(...) as shown in the following example.

```
. . . Double.parseDouble(myInput.next());
```

This form of input will throw a java.lang.NumberFormatException if the value is not a double.

If you are reading in each double value as a double using nextDouble(), for example

```
. . . myInput.nextDouble();
```

then a java.util.InputMismatchException will be thrown if the value read in is not a double.

You can either change your input to use Double.parseDouble(...) or you can catch the java.util.InputMismatchException and handle it the same way that you handled the NumberFormatException.

If you have mixed the two forms of input in your program and you want to keep both, then you will need to catch and handle both of the exceptions.