CS 211 Project 3: Classes and Objects

due Monday, March 15th, 2021, 11:59 PM

- this assignment is worth approximately 5% of your total grade
- Please Submit to Blackboard

The objective of this project is to implement a dessert planner. This project uses deeper class hierarchies than previously, and introduces the use of abstract classes, interfaces, as well as enum datatypes. It also demonstrates the use of polymorphism in problem solving.

Overview:

1. Create the files indicated below and implement the methods described.
2. Download and use a tester module to ensure that your programs are correct. The programs can also be tested using Dr Java's interactive window.
3. Prepare the assignment for submission and submit it.

Files for this Project:
The following files are relevant to the project. Some are provided in the project distribution while others you must create. **You should not submit the files that are provided.**

File	State	Description
junit-cs211.jar	provided	JUnit library for command line testing
P3Tester.java	provided	Runs all tests
ID.txt	create	Create in setup to identify yourself
Dessert.java	create	An interface class
Cake.java	create	An abstract class
Pastry.java	create	An abstract class
FrozenDessert.java	create	An abstract class
Cookies.java	create	An abstract class
ButterCake.java	create	Subclass of Cake
FoamCake.java	create	Subclass of Cake
Pie.java	create	Subclass of Pastry
Cobbler.java	create	Subclass of Pastry
IceCream.java	create	Subclass of FrozenDessert
FrozenYogurt.java	create	Subclass of FrozenDessert
MoldedCookies.java	create	Subclass of Cookies
NoBakeCookies.java	create	Subclass of Cookies
PieType.java	create	enum
Ingredient.java	create	enum
DessertPlanning.java	create	Dessert Planning class that uses the Dessert type

Guidelines and Rules:

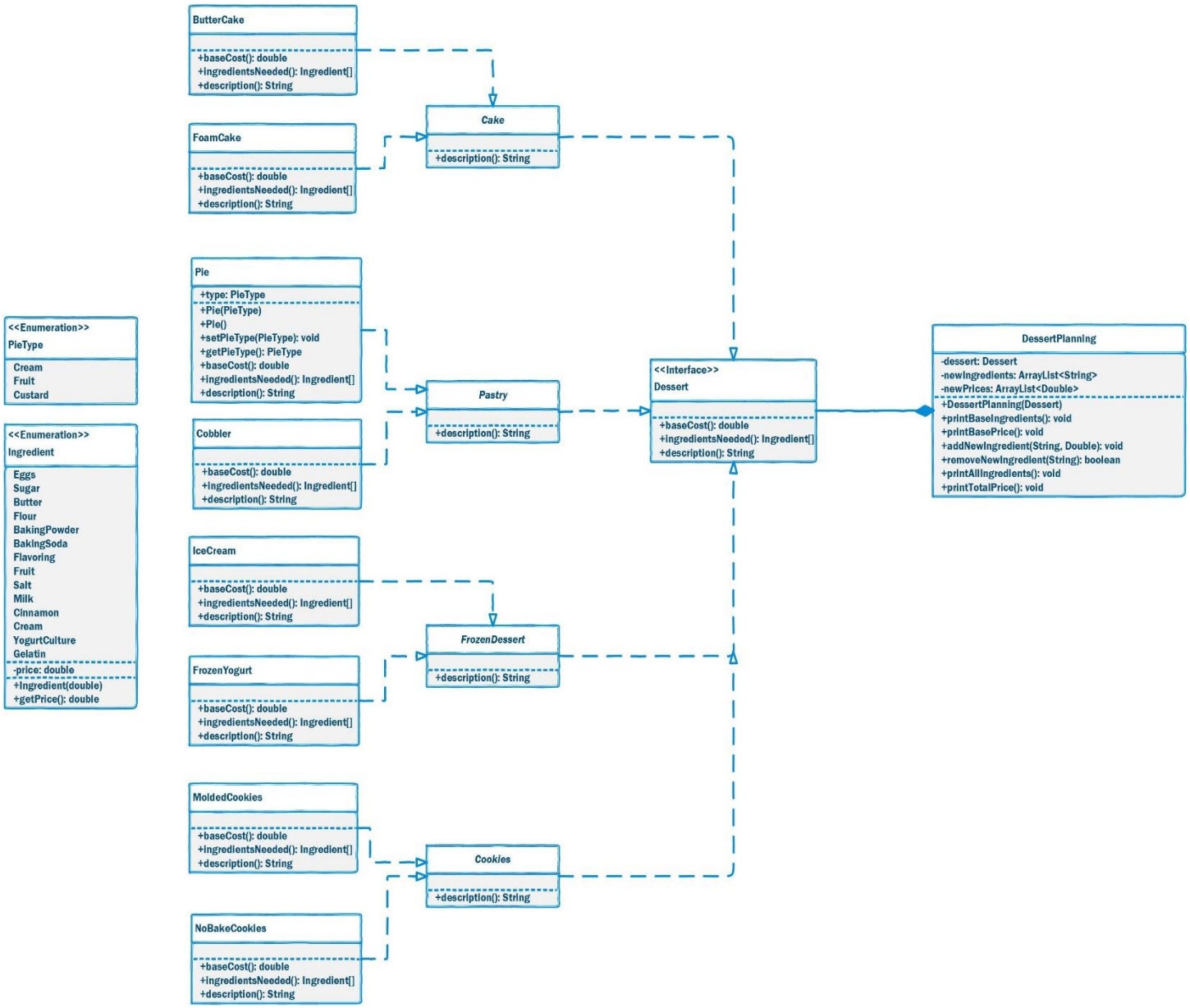
- **This project is an individual effort; the Honor Code applies.**
- The only library you can import is import java.util.ArrayList;
- The main method will not be tested; you may use it any way you want.
- You may use add(), remove(), get() for ArrayList.
- All data fields should be declared private or protected.
- You may add your own helper methods or data fields, but they should be private or protected.
- You may add additional constructors which are public.
- Format your code clearly and consistently.
- Each class definition will list the fields that must be present. Be sure to match the type and name exactly.
- Use of @Override tags on overridden methods is not required but is strongly recommended.
- When commenting code, use JavaDoc-style comments. Include @param and/or @return tags to explain what is passed in or returned wherever it isn't obvious. Any other JavaDoc tags are optional.

ASSIGNMENT:

A friend of yours love to bake but also live on a tight budget, so you decided to develop an application for them to budget their baking through a simple planner. The planner keeps track of 8 different desserts and their base ingredients and price of ingredient. The planner will allow your friend to determine how much it might cost them to make a certain dessert; the price won't be the exact cost of making the dessert; but more of an estimate on how much the ingredients will cost, thereby allowing your friend to continue their hobby. You are only in the beginning stages of this project and are currently working on defining some classes this application will need. You decided to use the UML below to define your classes.

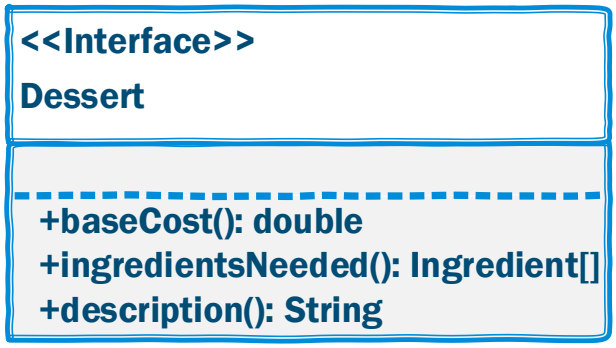
WARNING: THIS ASSIGNMENT WILL CAUSE CRAVINGS

UML:


ButterCake
+baseCost(): double
+ingredientsNeeded(): Ingredient[]
+description(): String
Cake
+description(): String
FoamCake
+baseCost(): double
+ingredientsNeeded(): Ingredient[]
+description(): String
Pie
+type: PieType
+Pie(PieType)
+Pie()
+setPieType(PieType): void
+getPieType(): PieType
+baseCost(): double
+ingredientsNeeded(): Ingredient[]
+description(): String
<<Enumeration>>
PieType
Cream
Fruit
Custard
Pastry
+description(): String
Cobbler
+baseCost(): double
+ingredientsNeeded(): Ingredient[]
+description(): String
<<Interface>>
Dessert
+baseCost(): double
+ingredientsNeeded(): Ingredient[]
+description(): String
DessertPlanning
-dessert: Dessert
-newIngredients: ArrayList<String>
-newPrices: ArrayList<Double>
+DessertPlanning(Dessert)
+printBaseIngredients(): void
+printBasePrice(): void
+addNewIngredient(String, Double): void
+removeNewIngredient(String): boolean
+printAllIngredients(): void
+printTotalPrice(): void
<<Enumeration>>
Ingredient
Eggs
Sugar
Butter
Flour
BakingPowder
BakingSoda
Flavoring
Fruit
Salt
Milk
Cinnamon
Cream
YogurtCulture
Gelatin
-price: double
+Ingredient(double)
+getPrice(): double
IceCream
+baseCost(): double
+ingredientsNeeded(): Ingredient[]
+description(): String
FrozenDessert
+description(): String
FrozenYogurt
+baseCost(): double
+ingredientsNeeded(): Ingredient[]
+description(): String
MoldedCookies
+baseCost(): double
+ingredientsNeeded(): Ingredient[]
+description(): String
Cookies
+description(): String
NoBakeCookies
+baseCost(): double
+ingredientsNeeded(): Ingredient[]
+description(): String

DESSERT

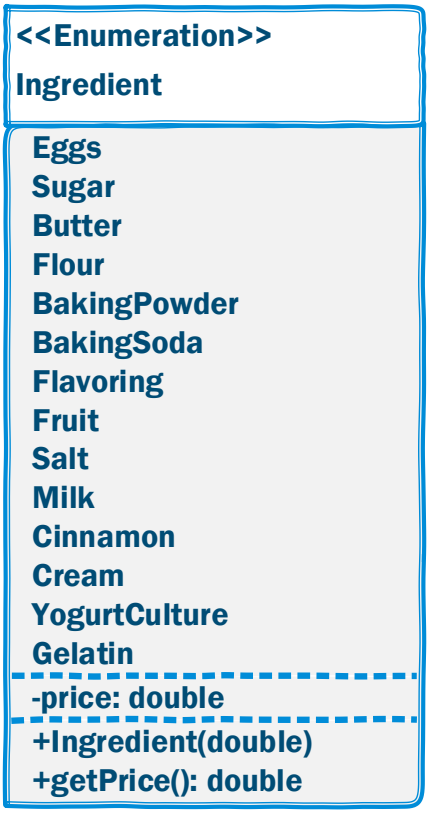
This will be used as a starting point for deriving any specific Dessert type. Every Dessert type has one or more ingredient; therefore, a Dessert has the following abstract methods:



+double baseCost() returns the total price based on all the base ingredients used for a dessert.

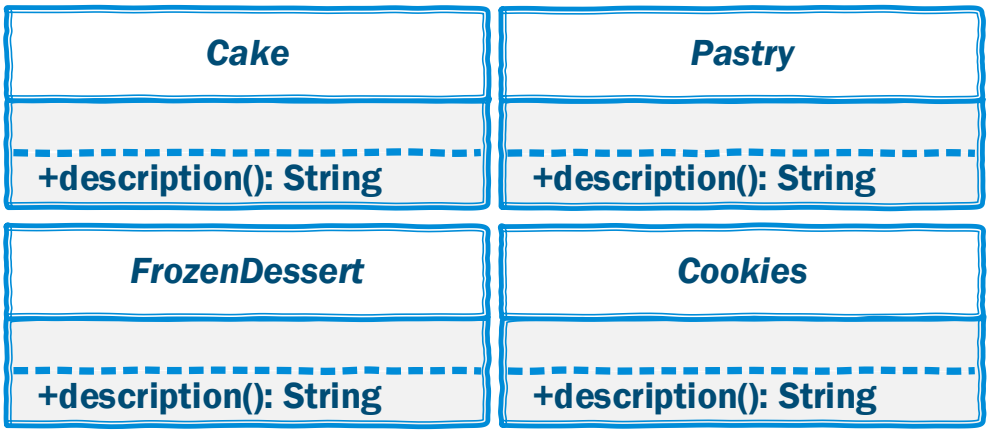
+String description() a method that returns the name of the dessert type.

+Ingredient[] ingredientsNeeded() a method that returns the ingredient(s) used by a dessert. Note that the return type of the method is Ingredient. Define an enum datatype called Ingredient. The enum contains the following values (see below UML) and a constructor which sets the price for each ingredient. The enum also contains a getter method which returns the price per ingredient. The prices of the ingredient values are initialized as: Eggs(3.48), Sugar(2.89), Butter(3.99), Flour(3.24), BakingPowder(1.99), BakingSoda(.85), Flavoring(2.99), Fruit(5.50), Salt(.99), Milk(2.72), Cinnamon(2.99), Cream(3.12), YogurtCulture(4.95), Gelatin(3.99).



CAKE, PASTRY, FROZENDESSERT, COOKIES

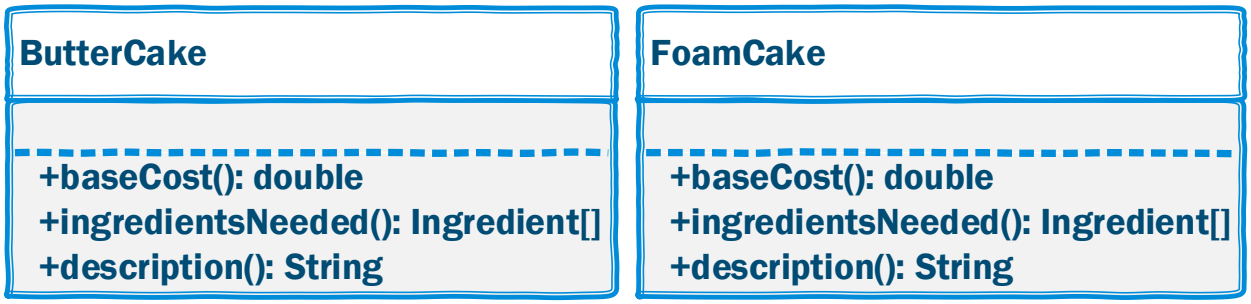
The different desserts are categorized into four different dessert groups. Namely: Cake, Pastry, FrozenDessert and Cookies. These four classes are abstract and implements the Dessert class.



The strings returned by each class are: “Cake”, “Pastry”, “Frozen Dessert”, “Cookies” (respectively).

BUTTERCAKE, FOAMCAKE

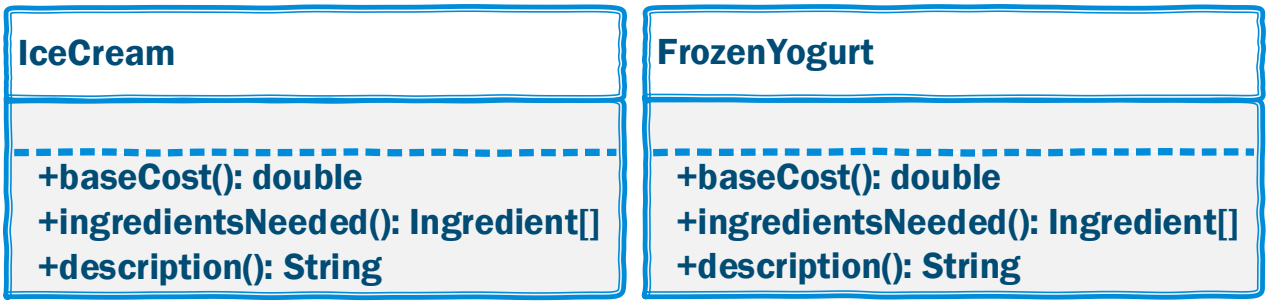
These classes derive from the Cake class. Both classes define the methods as set by the interface.



- The ingredients set for ButterCake are Eggs, Sugar, Butter, Flour, BakingPowder.
- The ingredients set for FoamCake are Eggs, Sugar, Butter, Flavoring.
- The description strings the classes return are: “Foam Cake”, “Butter Cake” (respectively).

ICECREAM, FROZENYOGURT

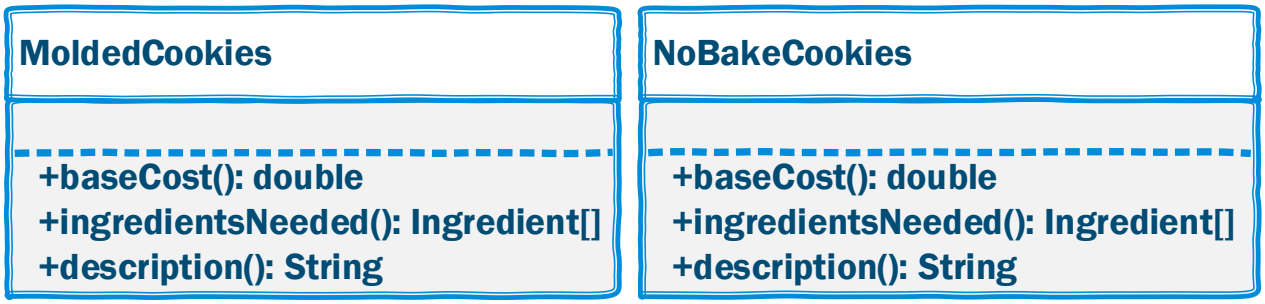
These classes derive from the FrozenDessert class. Both classes define the methods as set by the interface.



- The ingredients set for IceCream are Milk, Sugar, Gelatin, Eggs, Flavoring.
- The ingredients set for FrozenYogurt are Milk, Sugar, YogurtCulture, Flavoring.
- The description strings the classes return are: “Frozen Yogurt”, “Ice Cream” (respectively).

MOLDEDCOOKIES, NOBAKECOOKIES

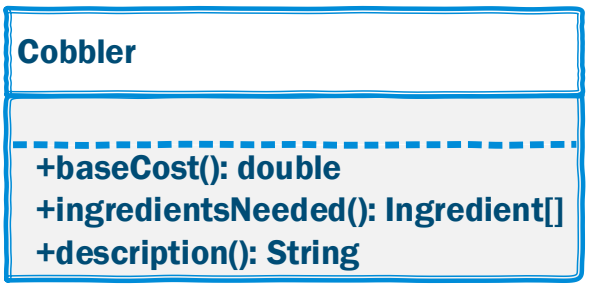
These classes derive from the Cookies class. Both classes define the methods as set by the interface.



- The ingredients set for MoldedCookies are Flour, Sugar, BakingSoda, BakingPowder, Milk, Cream, Butter.
- The ingredients set for NoBakeCookies are Sugar, Butter, Milk, Flavoring.
- The description strings the classes return are: “No-Bake Cookies”, “Molded Cookies” (respectively).

COBBLER

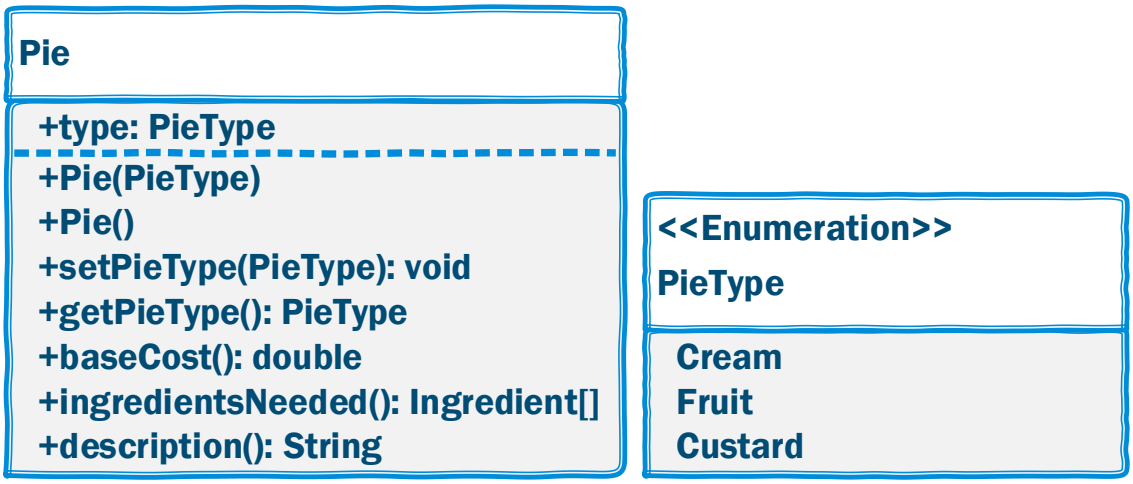
This class derive from the Pastry class. This class define the methods as set by the interface.



- The ingredients set for Cobbler are Fruit, Sugar, Salt, Butter, Flour, BakingPowder, Milk, Cinnamon.
- The description string the class return is: “Cobbler”.

PIE

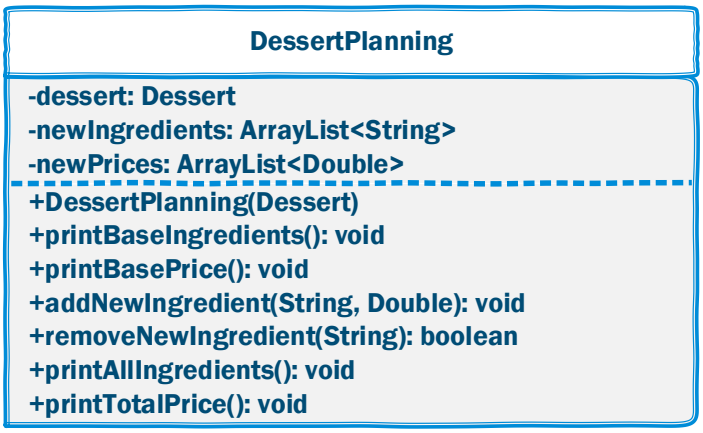
This class derive from the Pastry class. This class define the methods as set by the interface; and set ingredients based on three different PieType. PieType is an enum as shown in the UML below.



- The default constructor sets the member variable type to Cream by default.
- The parameterized constructor sets the member variable type to the passed in PieType.
- +setPieType(PieType) is the mutator for member variable type.
- +getPieType() is the accessor for member variable type.
- The ingredients set for the different pies are:
    - Cream Pie: Milk, Cream, Sugar, Flour, Eggs.
    - Fruit Pie: Fruit, Sugar, Salt, Flour, Butter, Eggs.
    - Custard Pie: Milk, Cream, Sugar, Eggs.
- The description string the class return is: “Custard Pie”, “Fruit Pie”, or “Cream Pie” based on the PieType.

DESSERTPLANNING

DessertPlanning is used to plan for any Dessert type. DessertPlanning allows the printing of base ingredients and price for any Dessert. It allows for new ingredients to be added or removed. It allows for calculating the total price of baking a particular dessert based on the base cost and any new costs.



- Member Variables: Dessert object, a String ArrayList and a Double ArrayList.
- Constructor allows the initialization of the member Dessert object to a passed in value. Sets newIngredients and newPrices to a new String ArrayList and Double ArrayList (respectively).
- +printBaseIngredients() uses System.out.println() to print out each of the base ingredients of the dessert.
  Example:
  Butter Cake Base Ingredients:
  Eggs $3.48
  Sugar $2.89
  Butter $3.99
  Flour $3.24
  BakingPowder $1.99
- +printBasePrice() uses System.out.println() to print out the base cost of the dessert.
  Example: Base Cost: $15.59
- +addNewIngredient(String, Double) populates the member String and Double ArrayLists.
- +removeNewIngredient(String) removes an ingredient from the String ArrayList and the corresponding price from the Double ArrayList.
- +printAllIngredients() uses System.out.println() to print out all the ingredients based on the base ingredients and the new ingredients added.
  Example:
  Butter Cake Base Ingredients:
  Eggs $3.48
  Sugar $2.89
  Butter $3.99
  Flour $3.24
  BakingPowder $1.99
  New Ingredients:
  Vanilla Extract $2.44
- +printTotalPrice() uses System.out.println() to print the total cost based on the base cost and the new ingredients.
  Example: Total Cost: $18.03

GRADING:

Rubric:

- Dessert Class (5 pts)
- Cake, Pastry, FrozenDessert, Cookies Classes (20 pts)
- ButterCake, FoamCake, Cobbler, IceCream, FrozenYogurt, MoldedCookies, NoBakeCookies Classes (35 pts)
- Pie Class (10 pts)
- Ingredient (5 pts)
- PieType (5 pts)
- DessertPlanning Class (10 pts)
- Documentation and Readability (5pts) [JavaDoc style Commenting, Proper Indentation/Overall Structure]
- Correct Project Setup (5pts) [zip format, file name, directory structure, files included, ID.txt]

To receive credit for this project you must submit it to Blackboard. Credit will be assigned based on the fraction of tests you pass. You must test your class with the provided tester to ensure that your code can compile and execute. Keep in mind, though, that the grader will have more tests than the provided tester and most of them will be different from the tester. Moreover, **it is your responsibility to verify that the tests pass on the command line, not on IDEs.** Test failures during grading will receive no credit and will not be examined for regrading.

Failure to submit your work in an appropriately named directory and with the ID.txt file in it with correct information will result in a 5% grade deduction.

After verifying that your code runs, zip your directory, and submit it to Blackboard. You may submit as many times as you like; only the final valid submission will be graded. If you turn in late work within 48 hours, there is a penalty (see syllabus). If you turn in late work beyond 48 hours, the submission will be ignored, and you will get zero points.

You should **always verify that your submission is successful**. Go back to the assignment and download your last submission; is there actually a file there? Can you unzip that file and see .java files in there as you would expect? Can you run those files? It turns out you can do quite a bit to ensure you have properly submitted your work. Do not get a zero just because of a failure to turn in the right files! It is your responsibility to correctly turn in your work.

TESTING:

Download the following files provided in the Project 3 assignment link in Blackboard:

- junit-cs211.jar
- P3Tester.java

This is a unit tester which is what we will use to test to see if your classes are working correctly.

The tester file may not be provided on the same day as the assignment instruction is posted. An announcement is made when the tester file is available.

When you think your classes are ready, do the following from the command prompt to run the tests.

On Windows:

```
javac -cp .;junit-cs211.jar *.java
java -cp .;junit-cs211.jar P3Tester
```

On Mac or Linux:

```
javac -cp .:junit-cs211.jar *.java
java -cp .:junit-cs211.jar P3Tester
```

In Dr Java:

- open the files, including `P3Tester.java`, and click the *Test* button.

In Eclipse:

- create a new JUnitTest Case (File --> New --> JUnit TestCase)
- Name it P3Tester
- Copy/paste the content of P3Tester.java in the new file
- Save the file
- Right click on the P3Tester.java on the Package explorer
- Chose Run As -->JUnit Test

SUBMISSION:

Submission instructions are as follows.

1. Let *xxx* be your lab section number and let *yyyyyyyy* be your GMU userid (netid). Create the directory `xxx_yyyyyyyy_P3/`
2. Place all of the `.java` files that you've created into the directory.
3. Create the file `ID.txt` in the format shown below, containing your name, netid, G#, lecture section and lab section, and add it to the directory.
   Full Name: Donald Knuth
   userID: dknuth
   G#: 00123456
   Lecture section: 001
   Lab section: 213

4. Compress the folder and its contents into a .zip file and upload the file to Blackboard.