CSIS0396A/COMP2396A - Assignment 3 (a)

Due: 18th Oct, 2014 23:30

Introduction

This assignment tests your understanding of **inheritance** and **polymorphism** in Java. This assignment consists of two parts. You need to implement the **basic requirements** of the program in part (a). In part(b), you have to **extend the program** by submitting **additional** files. Part (b) will be released on 20th October, 2014. **Files submitted in part (b) must work with the files in part (a) submitted on or before 18th Oct 2014.**

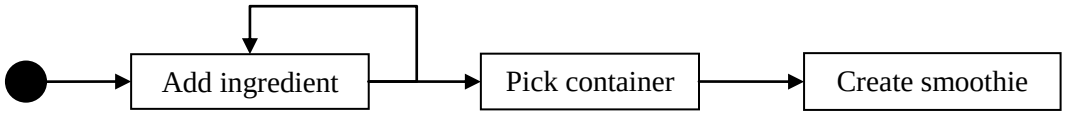
You are required to implement an ordering system for a shop that sells **smoothies**. The shop allows customers to customize their own smoothie by mixing from a variety of ingredients.

You need to design the program structure. Program design **will be evaluated** in this assignment. You must make good use of **inheritance** and **polymorphism** in order to get all marks for this assignment. You must make use of **abstract** class and **interface**, and **all instant variables must be private**. The marking criteria are included at the end of this assignment description.

You are also required to write **JavaDoc** for all non-private classes and non-private class members. **Submissions without JavaDoc will not be marked.**

Task

You need to implement the main program, `SmoothieShop.java`, which will be the ordering system for the shop. Here is the flow diagram of how a smoothie is created.



Ingredients

There are 8 possible ingredients that can be added to a smoothie. When an ingredient is added, it contributes a certain amount of **volume** in the smoothie.

Ingredient	Volume
Apple	40-50ml by random
Orange	50-60ml by random
Banana	35-40ml by random
Melon	50-70ml by random

Ingredient	Volume
Coconut	60-80ml by random
Chocolate chips	30-40ml by random
Milk	Always 100ml
Vanilla ice cream	40-45ml by random

When the program is executed, the following menu should be displayed:

```
1. Apple
2. Orange
3. Banana
4. Melon
5. Coconut
6. Chocolate chips
7. Milk
8. Vanilla ice cream
What would you like to add to your smoothie?
Please enter your choice (1-8, or 0 to finish the order):
```

If the user entered option 1-8, the program should show the **ingredient** and its **volume**:

```
Added Apple 45ml
```

The process should be repeated until the user entered `0` to finish the order. The user may choose to add the **same ingredient** again. In that case, the **volume may be different** because each ingredient added is considered a different entity.

Ask for input again if the user enters any other input. You can assume that the user will always input an integral value. Your program does not need to handle non-integer input.

Containers

There are 3 possible containers that can be used. Each container can only hold a certain amount of smoothie.

Container	**Capacity**
Plastic cup	Always 300ml
Melon	4 times the volume
Coconut	5 times the volume

If the user selected any one of the ingredients that can be used as a container, the user can choose either using the plastic cup (the default container), or the ingredient itself as the container. For example, the following menu should be shown if the user has chosen **one coconut** and **two melons** as the ingredients:

```
1. Plastic cup (300ml)
2. Melon (236ml)
3. Coconut (395ml)
4. Melon (232ml)
What would you like to use to hold your smoothie?
Please enter your choice (1-4):
```

Note that the order follows how the ingredients are added.

The user must select one of the options. Ask for input again if the user entered any other input. You can assume that the user will always input an integral value. Your program does not need to handle non-integer input.

If none of the ingredients can be used as a container, the **plastic cup is automatically used**. The menu should not be shown in such case.

Creating smoothie

If the total volume of the ingredients is less than the capacity of the container, **ice** is **automatically added** to the mix. The program should print out the amount of ice being added:

```
Added Ice 99ml
```

The program should then print out the **list of ingredients** and the **container**. For example:

```
Smoothie ingredients:
Melon (59ml)
Milk (100ml)
Coconut (79ml)
Melon (58ml)
Ice (99ml)
Container: Coconut (395ml)
```

In the above example, the ingredients contribute **296ml** of volume and the container has a capacity of **395ml**. Therefore **99ml** of ice is automatically added.

If too many ingredients are chosen and they cannot fit into the container, no ice is added and the amount of **waste** should be shown. For example:

```
Smoothie ingredients:
Apple (49ml)
Coconut (73ml)
Apple (50ml)
Milk (100ml)
Milk (100ml)
Container: Plastic cup (300ml)
Wasted 72ml!
```

Note there it is also possible that the amount of ingredients is exactly the capacity of the container. Then no ice is added and there is no waste.

The program should then terminate.

Part (a) requirements

You program must fulfill the following requirements:

- Model all possible **ingredients** and **containers** as **objects**. Allow **easy expansion** to the program by adding **more ingredients** that may or may not be used as a **container**.
- Define **abstract classes** for classes that are supposed to be inherited but not instantiated directly.
- All **instant variables** must be **private**. Define **getters** to access these variables only if necessary.
- `SmoothieShop.java` is the main program that controls the **program flow** and **user interaction** only. You are encouraged to define the program logic in other classes. The size of the main program, `SmoothieShop.java`, including comments and JavaDoc, must not exceed 4k bytes (4096 bytes).

Part (b) requirements

- Part (b) of the assignment will be released **after** you have submitted part (a).
- You will be asked to define some **new ingredients** that may or may not be used as a **container**.
- You may also be asked to **remove an ingredient** from the shop.
- You will have to submit the **new classes** for the new ingredients, as well as a **new main program** (in place of `Smoothieshop.java`). These files must work with your files submitted in part (a).
- You are **not allowed** to modify the files submitted in part (a) when you submit part (b).

Marking (65% for part a)

- **35% marks** are given to the **program design.**
  - You will be awarded all the marks if you are implementing the **ingredients** and **containers** by making use of **inheritance** and **polymorphism**.
  - You must use **abstract** class for classes that an object of such class should not be instantiated.
  - You must make use of **interface** in your program
  - You can check the by avoiding code duplication as much as possible.
  - **Economy is valuable in coding: the easiest way to ensure a bug-free line of code is not to write the line of code at all.**
- **15% marks** are given to the **functionality** of your program.
  - Your program output must be **identical** to what is described in this document, with the exception of the trailing spaces at the end of each line of output.
  - Please ask us on Moodle or by email if you are not sure about any of the requirements stated in this specification.
- **15% marks** are given to your **JavaDoc**. A complete JavaDoc includes documentation of every classes, member fields and methods that are not private. JavaDoc for the main method may be omitted.
  - You should know the JavaDoc requirements clearly now. So **you will either get all marks or none** for the JavaDoc written for this assignment.

Submission:

Please submit all source files (`*.java`) in a single compressed file (in `.zip` or `.7z`) to Moodle. **Late submission is not allowed**.

Do not submit *.class* file.

-- END --