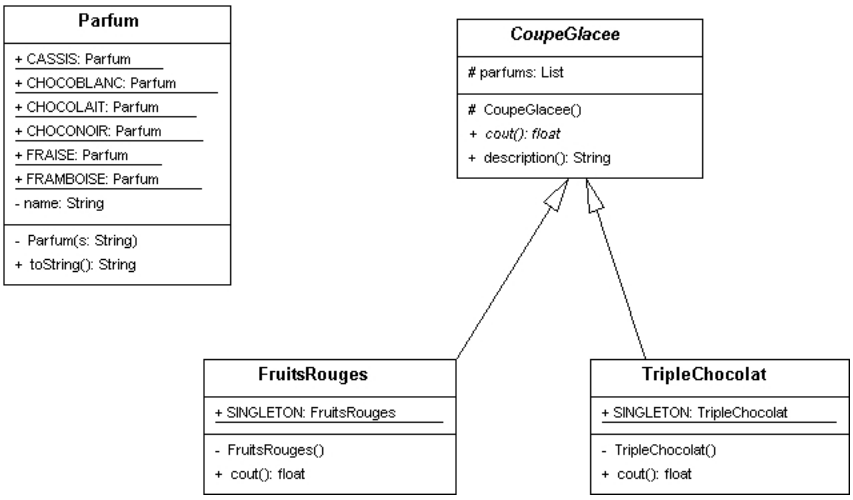
TD Glaces

Exercice 1 : Glaces

Dans le cadre de l'informatisation d'un glacier, il est nécessaire de modéliser les différentes coupes glacées que celui-ci peut proposer. Cela permet notamment la gestion des commandes et factures des clients.

Dans une première version, seules des coupes glacées composées de plusieurs boules de différents parfums sont gérées. Dans cette première version, on trouve le diagramme de classes suivant[1], auxquels on peut imaginer ajouter d'autres coupes glacées chacune correspondant à un singleton :



Une partie du code associé est fourni en annexe.

La classe `Commande` du logiciel de gestion du glacier comporte la méthode suivante qui permet d'afficher une facture sur le flux de sortie voulu et déterminé par le `PrintWriter` passé en paramètre :

```
public void publieFacture(PrintWriter writer) {
   float total = 0;
   for (CoupeGlacee coupe :  this.lesCoupes) {
     writer.println(coupe.description()+" + "+coupe.cout());
     total = total + coupe.cout();
   }
   writer.println("   TOTAL : "+total);
}
```

Pour une commande composée de deux coupes "fruits rouges" et d'une coupe "triple chocolat", la facture ainsi publiée ressemble à :

```
Coupe fraise framboise cassis + 5,50
Coupe fraise framboise cassis + 5,50
Coupe chocolat blanc chocolat au lait chocolat noir +6,00
   TOTAL : 17
```

Evolution du logiciel. Suite à de nombreuses demandes de ses clients, le glacier décide qu'il sera possible pour chaque client de personnaliser ses coupes par l'ajout d'un ou plusieurs agréments (ou "topping") tels que crème chantilly, coulis chocolat, coulis de fraise etc. Ceci évidemmment contre un modique surcoût dépendant de l'ingrédient. La description de la coupe glacée consommée par le client doit bien entendu se trouver modifiée en conséquence.

Voici quelques toppings possibles, leur surcoût et la description associée :

[1] `Parfum` pourrait être géré par une *énumération* en java 1.5.

topping	*surcoût*	*description*
chantilly	0,50	“chantilly”
sauce chocolat	0,70	“et sa délicieuse sauce chocolat”
coulis fraise	1	“au coulis de fraises fraiches”

Evidemment il est possible pour les gourmands de cumuler plusieurs toppings sur une même coupe glacée. Dans ce cas les surcoûts se cumulent et les descriptions s’enrichissent en conséquence.
Il est évidemment nécessaire de prendre en compte cette évolution au niveau des commandes et factures.
Voici donc un exemple de facture que l’on doit obtenir dans la seconde version :

```
Coupe fraise framboise cassis chantilly + 6,00
Coupe fraise framboise cassis chantilly au coulis de fraises fraiches + 7,00
Coupe chocolat blanc chocolat au lait chocolat noir  +6,00
Coupe chocolat blanc chocolat au lait chocolat noir
       et sa délicieuse sauce chocolat chantilly  +7,50
   TOTAL :   26,50
```

Q 1 . Proposez une solution élégante à l’ajout des classes permettant la gestion de coupes glacées avec “toppings” pour la seconde version de l’application de gestion du glacier.

Dans un premier temps vous **compléterez** le diagramme UML donné à la figure précédente, puis vous **donnerez tout le code** nécessaire à l’intégration du topping “Chantilly” dans l’application.

Votre solution doit permettre l’ajout de nouveaux toppings sans modification du code (respect du principe *ouvert-fermé*), ce qui n’est par exemple pas possible pour l’ajout de nouveaux parfums avec la conception actuelle. Vous **justifierez** votre respect de ce principe en indiquant brièvement mais clairement ce qu’il faut faire pour ajouter un nouveau topping (*noisettes caramélisées* par exemple).

Q 2 . Faut il maintenant modifier la méthode `publieFacture`, si oui comment, si non pourquoi ?

Q 3 . Donnez la ligne de code permettant de créer “*une Coupe chocolat blanc chocolat au lait chocolat noir et sa délicieuse sauce chocolat chantilly*”.

Encore une évolution (refactoring). Le succès toujours croissant de notre glacier l’amène à compléter sa carte en proposant également des boissons : des chocolats chauds et des cafés. Les cafés sont de plusieurs variétés possibles : *robusta*, *arabica*, etc.
Ces boissons ont également un coût et une description repris dans le tableau ci-dessous :

boissons	*coût*	*description*
café	1,50	“café” + *nom de la variété de café*
chocolat chaud	2,00	“chocolat chaud”

Le glacier pense tout de suite qu’il sera possible de compléter les cafés et chocolat avec de la chantilly, cela a la même incidence que précédemment pour les glaces.

Q 4 . Indiquez les modifications de conception qu’il faut apporter au modèle pour permettre de gérer à la fois les coupes glacées et les boissons de manière uniforme. C’est-à-dire que, par exemple, les commandes doivent aussi bien pouvoir contenir des coupes glacées que des boissons et qu’une simple et petite modification de code doit permettre de réutiliser la méthode `publieFacture`.

Vous proposerez un nouveau diagramme UML pour gérer les glaces et les boissons. Il n’est pas question ici de respecter le principe ouvert-fermé par rapport à la première version, mais d’y apporter le minimum de modifications nécessaires.

Vous indiquerez également les modifications à apporter à la publication de factures.

Annexe

Sources des classes `CoupeGlacee` et `FruitsRouges`.
Le code de la classe `TripleChocolat` est similaire à la classe `FruitsRouges` aux parfums près.

```
package glace;
import java.util.*;
public abstract class CoupeGlacee {
   protected List<Parfum> parfums;
   protected CoupeGlacee()  {
      this.parfums = new ArrayList<Parfum>();
   }
   public String description() {
      StringBuffer sb = new StringBuffer("");
      for(Parfum parfum : this.parfums) {
         sb.append(parfum.toString());
         sb.append(" ");
      }
      return sb.toString();
   }
   public abstract float cout();
}
-----------------------------------------------------------------------
package glace;
public class FruitsRouges extends CoupeGlacee {
   public static final FruitsRouges SINGLETON = new FruitsRouges();
   private FruitsRouges() {
      this.parfums.add(Parfum.FRAISE);
      this.parfums.add(Parfum.FRAMBOISE);
      this.parfums.add(Parfum.CASSIS);
   }
   public float cout() {
      return 6;
   }
}
```