Lycée Léonard de Vinci *Section Technicien Supérieur en Informatique SIO*
Option SLAM

CONTRÔLE DES CONNAISSANCES N° 1 Vendredi 18 mars 2022
DURÉE : 1h

Correction : Gabriel Demazeu, Ethan Bercier

Question 1 : En POO, l'encapsulation est typiquement un mécanisme qui permet : (**1 point**)

([X]) De limiter la redondance de code
[X] De cacher une implémentation à des programmes appelants
[ ] De cacher une interface à des programmes appelant **(Ne cache pas une interface)**
[X] De regrouper des données étroitement liées
[ ] De regrouper des données sans lien particulier **(Question organisation, ne soit pas foure tout)**
[X] De regrouper des données et des fonctions liées à ces données
[X] De ne pas faire de différence entre donnée calculée et donnée attribuée

Question 2 : Le symbole UML dièse (#) est traduit en Java par (**1 point**)

[X] Protected
[ ] Private
[ ] Invariant
[ ] Public
[ ] Package
[ ] Card

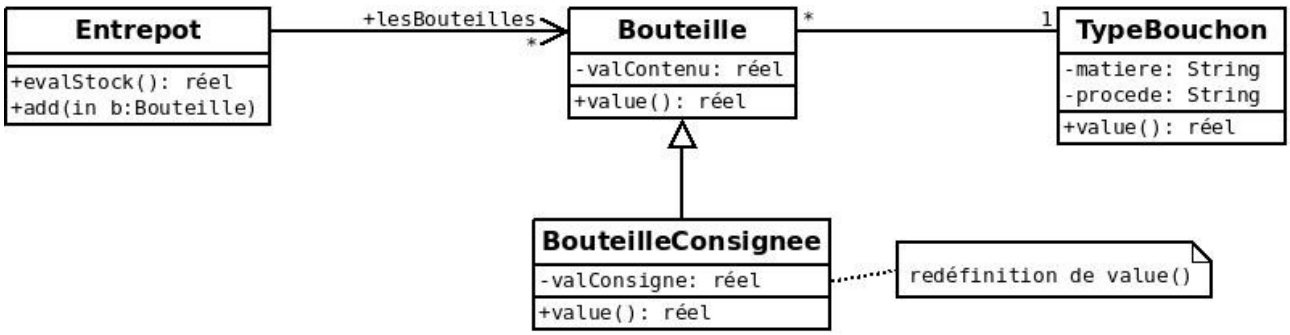
Question 3 Une classe d'implémentation instanciable : (**1 point**)
(Instanciable donc n'est pas abstraite)
[ ] Déclare forcément des attributs
[ ] Dispose forcément d'un seul constructeur
[X] Dispose forcément d'un ou plusieurs constructeurs
[ ] Est une classe qui ne peut pas être héritée.

Question 4 Une méthode de classe (*static*) : (**1 point**)
[ ] S'appuie toujours sur les attributs déclarés dans sa classe (s'ils existent)
[ ] Dépend au moins d'une instance de sa classe
[ ] Dépend de toutes les instances de sa classe
[X] Ne dépend d'aucune instance de sa classe **(Typique d'une classe static)**
[X] Est typiquement préfixée par sa classe au moment de son appel.
[ ] Est typiquement préfixée par un objet au moment de son appel. **(Il s'agit des méthodes d'instance)**

Question 5 Un constructeur: (**1 point**)
[ ] est une méthode d'instance qui rend *void* **(Les constructeurs ne sont pas des méthodes)**
[X] est obligatoirement utilisé avec **new (demande une allocution mémoire pour l'objet créer)**
[ ] est utilisé avec **new**, mais pas seulement
[X] sert à initialiser une instance
[X] sert à recevoir des valeurs pour les attribuer à l'instance en construction
[ ] sert à construire des méthodes
([X]) sert à déclarer des attributs

Question 6 : (8 points)

Un entrepôt dispose d'un stock de bouteilles. Les bouteilles sont classées en deux catégories: les consignées et les non consignées. La valeur d'une bouteille est fonction de la valeur de son contenu. Dans le cas d'une bouteille consignée, la valeur de la bouteille tient compte **en plus** de la valeur de la consigne. Le développeur conçoit les classes suivantes :



Un développeur vous affirme que la méthode **add** de **Entrepot** permet d'ajouter des objets **BouteilleConsignee**,

1/ En quoi le diagramme de classes confirme-t-il ou non ses propos ? (**2 points**)

Vrai, on peut voir sur le diagramme que BouteilleConsignee hérite de Bouteille qui a un lien avec Entrepot. On peut donc appliquer le principe de substitution de Barbara Liskov. add de Entrepot permet donc d'ajouter des objets BouteilleConsignee.

2/ Implémenter les méthodes suivantes (**langage de votre choix**) :

2.1 value() de la classe Bouteille **(2 points)**

```
public fonction value() : réel {
     return this.valContenu ;
}
```

2.2 value() de la classe BouteilleConsignee **(2 points)**

```
public fonction value() : réel {
     return this.valConsigne  + super.value() ;
     //ou +this.getValcontenu(); mais c'est moins propre
}
```

2.3 evalStock() de la classe Entrepot qui évalue la **valeur** du stock **(3 points)**
// La classe Entrepot dispose d'un attribut *lesBouteilles* de type *List<Bouteille>*
// => voir ANNEXE 1 (prendre Bouteille comme instance du type E)

```
fonction evalStock() : float {
        res : float=0
        for(b:Bouteille in this.lesBouteilles) {
            res += b.value()
        }
        return res
}
```

3/ Pour des raisons de sécurité, on souhaite connaître les types de bouchons de bouteilles présents dans un entrepôt. Pour cela on vous de demande :

3.1 L'interface d'une méthode qui pourrait répondre à cette demande, en signalant la classe dans laquelle cette méthode est déclarée. (**2 points**)

```
public getTypeBouchons : List<TypeBouchon>
```

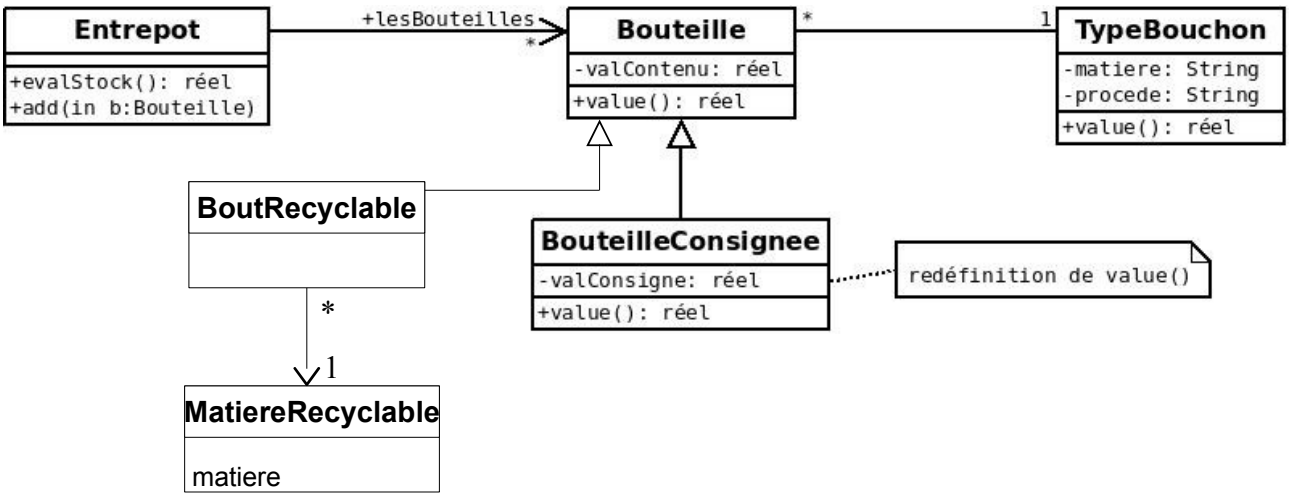
3.2 Implémenter cette méthode (**4 points**)

```
public getTypeBouchons : List<TypeBouchon> {
        res : List<TypeBouchon>
        res=new Arraylist() ;
        for(Bouteille b : this.lesBouteilles){
            if(!res.contains(b.getTypeBouchon()){
                res.add(b.getTypeBouchon()) ;
            }
        }
        return res ;
}
```

4/ (optionnel) On souhaite gérer des bouteilles recyclables, en matière plastique, sans valeur de consigne. Les types de plastiques à recycler sont :

- PET, qui constitue les bouteilles transparentes ou colorées
- PeHD, qui constitue les bouteilles opaques (lait ou lessive)

Proposer un diagramme de classes UML de votre solution (**4 points**)



ANNEXE 1 – L'interface List

packagr java.util

Interface List<E>

All Superinterfaces:

Collection<E>, Iterable<E>

All Known Implementing Classes:

AbstractList, AbstractSequentialList, ArrayList, AttributeList, CopyOnWriteArrayList, LinkedList, RoleList, RoleUnresolvedList, Stack, Vector

---

Method Summary

boolean	**add**(E e) Appends the specified element to the end of this list (optional operation).
void	**add**(int index, E element) Inserts the specified element at the specified position in this list (optional operation).
void	**clear**() Removes all of the elements from this list (optional operation).
boolean	**contains**(Object o) Returns `true` if this list contains the specified element.
boolean	**equals**(Object o) Compares the specified object with this list for equality.
E	**get**(int index) Returns the element at the specified position in this list.
int	**indexOf**(Object o) Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
boolean	**isEmpty**() Returns `true` if this list contains no elements.

. . .

int	**size**() Returns the number of elements in this list.
List<E>	**subList**(int fromIndex, int toIndex) Returns a view of the portion of this list between the specified `fromIndex`, inclusive, and `toIndex`, exclusive.
Object[]	**toArray**() Returns an array containing all of the elements in this list in proper sequence (from first to last element).