Nom :
Prénom :
Numéro d'étudiant :

Modélisation et programmation par objets 2

Tous documents sur support papier autorisés. Durée : 1h00
L'ensemble des réponses sera à donner sur les feuilles d'énoncé. Ne pas dégrapher les feuilles.

On se place dans le contexte de la conception d'une application dédiée à un magasin de produits en vrac. On s'intéresse par la suite à la représentation des produits et des contenants.

1 Des produits (et des interfaces)

Question 1. Ecrivez en Java une interface ProduitAuPoids. Un produit vendu au poids a un nom, un prix par kilo, une masse volumique en kg par l, et est un produit sec ou pas.

Question 2. Ecrivez en Java une interface ProduitSecAuPoids. Un produit sec vendu au poids est un produit vendu au poids ayant la particularité d'être forcément sec.

Question 3. Ecrivez en Java *l'entête* d'une classe `Riz` implémentant l'interface de la question 2.

2 Des contenants (et des classes génériques)

On met en place une classe Contenant. Les contenants permettent de contenir les produits vendus au poids. Les types de contenants sont des sacs en papier kraft ou en tissu, ou des bocaux. Un contenant a une certaine contenance maximale (`contenuMax`), et donc un contenu, et un volume pour ce contenu (`volume`). La classe contenant est paramétrée par le type de contenu qu'il peut recevoir. Le constructeur permet de construire un contenant vide au sens où le volume du contenu est nul. La méthode remplir permet ensuite de remplir le contenant d'un certain pourcentage de la contenance maximale. La méthode prix permet ensuite de calculer le prix approximatif du contenant rempli.

Question 4.

Complétez la première ligne de la classe Java Contenant.

```
public enum TypeContenant {
sacKraft,
sacTissu,
bocalVerre;
}

public class Contenant<                                                    >
{
  private TypeContenant type;
  private float volume;
  private final float contenuMax;
  private Contenu contenu;


  // builds an empty Contenant
  public Contenant(Contenu c, TypeContenant type,  float contenuMax) {
    this.type = type;
    this.volume = 0;
    this.contenu=c;
    this.contenuMax = contenuMax;
  }


  public void remplir(int pourcentage) {
    volume =(contenuMax*pourcentage)/100f;
  }

  public float prix() {
    return contenu.getPrixParKg()*volume*contenu.getMasseVolumique();
  }
}
```

Question 5. Donnez en Java une classe ContenantNonEtanche minimale, un contenant non étanche est un contenant paramétré par le type de contenu qu'il peut recevoir, qui doit forcément être un produit sec vendu au poids.

Question 6. On suppose disposer d'un constructeur par défaut pour la classe Riz. Donnez en Java le code permettant de créer un nouveau contenant non étanche pour produit sec, en papier kraft, de contenance maximale 2 litres, dont le contenu sera du riz, puis de le remplir à moitié. On nommera sacDeRiz ce contenant.

Question 7. Donnez tout le code Java nécessaire pour que la ligne suivante :

```
System.out.println(sacDeRiz);
```

affiche : `sac en papier kraft rempli de 1.0l de Riz`. Attention au type de contenant qui doit s'afficher correctement (sac en papier kraft et pas sacKraft). On suppose que la classe Riz est munie d'une méthode toString() qui retourne "Riz".

Question 8. Comment proposez-vous de tester la classe Contenant (avec JUnit) ? Expliquez brièvement les éventuelles difficultés rencontrées.

3 Annotations

Question 9. Donnez le code Java d'une annotation `RequiredTestIntensity` permettant de donner pour une méthode ou un constructeur le niveau de test requis (un entier) indiquant la difficulté des tests requis pour la méthode ou le constructeur, ainsi que la raison (une chaîne de caractères) expliquant cette difficulté.

Question 10. Donnez ci-dessous un exemple d'utilisation pour cette annotation en annotant la méthode m() qui suit avec les valeurs de votre choix.

```
public void m(){
// very complicated code
}
```

Question 11. Donnez le code Java d'une annotation `RequiredTestIntensityAndTime` qui en plus des informations présentes dans une annotation `RequiredTestIntensity` ajoute le temps (en minutes) que le testeur devrait consacrer à la méthode ou au constructeur.