Nom :	Prénom
Numéro d'étudiant :	Groupe (A, B, C, MI) :

Modélisation et programmation par objets 2

Seul document autorisé : une feuille A4 recto verso. Durée : 1h.
L'ensemble des réponses sera à donner sur les feuilles d'énoncé. Ne pas dégrapher les feuilles.

Une entreprise développe un logiciel pour gérer ses commandes de masques pour son personnel. Les types de masques (classe `Mask`) ont un prix unitaire hors taxe, et une méthode pour calculer le nombre de masques requis par mois en fonction du nombre de demi-journées d'utilisation. Pour les masques à usage unique (classe `SingleUseMask`), cette méthode retourne exactement le nombre de demi-journées travaillées. Pour les masques réutilisables (classe `ReusableMask`), cette méthode retourne le max entre 2 et le nombre de masques nécessaires, en prenant en compte le nombre de lavages autorisés pour le masque. Un masque à usage unique a une catégorie FFP qui peut être soit `none` (aucune), soit `FFP1`, soit `FFP2`, soit `FFP3`.

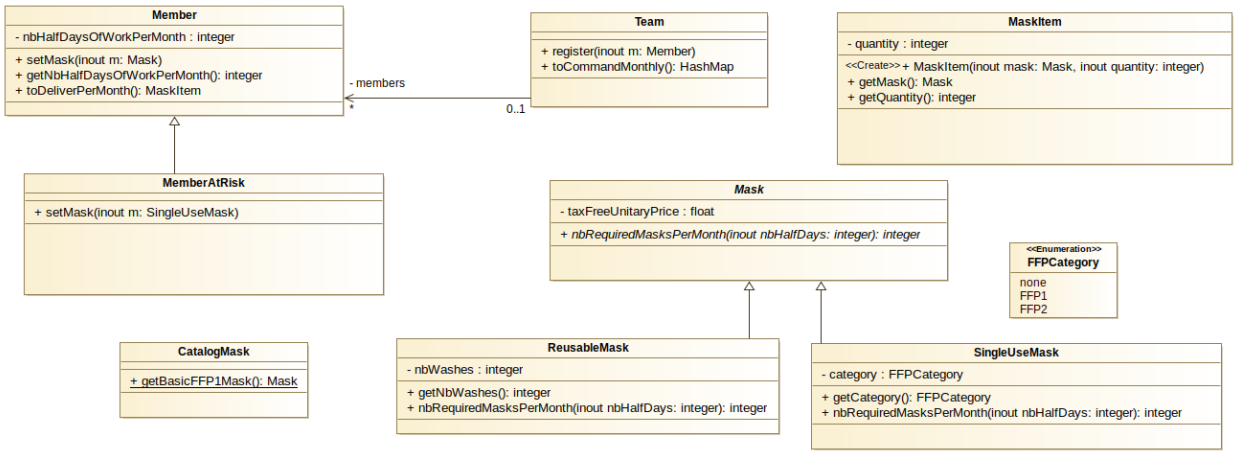

FIGURE 1 – Diagramme de classe partiel

```
public abstract class Mask {
  private float taxFreeUnitaryPrice;

  public Mask(float unitaryPrice) {
    this.taxFreeUnitaryPrice = unitaryPrice;
  }
  public abstract int nbRequiredMasksPerMonth(int nbHalfDays);
}
```

```
public class ReusableMask extends Mask {
  private int nbWashes;

  public int getNbWashes() {
    return nbWashes;
  }

  public ReusableMask(float unitaryPrice, int nbWashes) {
    super(unitaryPrice);
    this.nbWashes = nbWashes;
  }

  public int nbRequiredMasksPerMonth(int nbHalfDays) {
    return Integer.max(2,(int)(Math.ceil((float)nbHalfDays/nbWashes)));
  }
}
```

```java
public class SingleUseMask extends Mask {
  private FFPCategory category;

  public SingleUseMask(float unitaryPrice, FFPCategory category) {
    super(unitaryPrice);
    this.category = category;
  }

  public FFPCategory getCategory() {
    return category;
  }

  public int nbRequiredMasksPerMonth(int nbHalfDays) {
    return nbHalfDays;
  }
}
```

```java
public enum FFPCategory {
  none, FFP1, FFP2, FFP3;
}
```

Les membres de l'entreprise (classe `Member`) travaillent dans l'entreprise un certain nombre de demi-journées par mois (`nbHalfDaysOfWorkPerMonth`), et ils doivent choisir le type de masque (`chosenMask`) qui leur convient. Le choix se fait via l'accesseur `setMask`. Les membres du personnel à risque (`MemberAtRisk`) ne peuvent choisir que des masques à usage unique de catégorie au moins `FFP1` (pas `none`).

```java
public class Member {
  private int nbHalfDaysOfWorkPerMonth ;
  private Mask chosenMask;

  public Member(int nbHalfDaysOfWorkPerMonth) {
    this.nbHalfDaysOfWorkPerMonth = nbHalfDaysOfWorkPerMonth;
  }

  public void setMask(Mask m) {
    this.chosenMask=m;
  }

  public int getNbHalfDaysOfWorkPerMonth() {
    return nbHalfDaysOfWorkPerMonth;
  }

  public MaskItem toDeliverPerMonth() {
    return new MaskItem(chosenMask, chosenMask.nbRequiredMasksPerMonth(
        nbHalfDaysOfWorkPerMonth));
  }
```

```java
public class MemberAtRisk extends Member{
  public MemberAtRisk(int nbHalfDaysOfWorkHebdo) {
    super(nbHalfDaysOfWorkHebdo);
  }

  public void setMask(SingleUseMask m) {
    if (m.getCategory()!=FFPCategory.none) {
    super.setMask(m);
    } else {
      System.out.println("incorrect choice");
    }
  }
}
```

Question 1. On écrit le code suivant :

```
Mask mask1SingleUse=new SingleUseMask(0.5f, FFPCategory.none);
SingleUseMask mask2SingleUse=new SingleUseMask(0.5f, FFPCategory.none);
MemberAtRisk member1AtRisk =new MemberAtRisk(10);
Member member2AtRisk =new MemberAtRisk(8);

member1AtRisk.setMask(mask1SingleUse); // instruction 1
member1AtRisk.setMask(mask2SingleUse); // instruction 2
member2AtRisk.setMask(mask1SingleUse); // instruction 3
```

Indiquez parmi les 3 instructions avec un `setMask` laquelle ou lesquelles déclenchent/ne déclenchent pas l'affichage d'un message "incorrect choice", en justifiant votre réponse.

► *Seule la 2 (type statique du membre= MenmberAtRisk et type statique du masque=MasqueSingleUse) déclenche l'affichage de l'erreur. En effet, les 2 méthodes setMask sont en surcharge l'une de l'autre et pas en redéfinition. De ce fait, à la compilation la signature cible est choisie de manière statique. Donc ici :*
**cas 1 : on fixe la signature setMask(Mask). A l'exécution, on cherche cette signature à partir de MemberAtRisk MemberAtRisk (pas trouvée) puis dans Member (trouvée).*
** cas 2 : on fixe la signature setMask(SingleUseMask). A l'exécution, on cherche cette signature dans MemberAtRisk → trouvée*
** cas 3 : on fixe la signature setMask(Mask). A l'exécution, on cherche cette signature dans MemberAtRisk → pas trouvée puis dans Member (trouvée).*
✠*1 point pour cq instruction (réponse et justif)*
✠*total 3 points*

Question 2. Comment réécrire `setMask` pour que chacune des instructions déclenche bien l'affichage d'une erreur comme il serait souhaitable ?

► *Faire une redéfinition et pas une surcharge :*
✠*1 point pour l'explication*
✠*2 points pour la réécriture*
✠*total 3 points*

```
public void setMask(Mask m) {
  if (m instanceof SingleUseMask){
   if (((SingleUseMask)m).getCategory()!=FFPCategory.none) {
   super.setMask(m);
   }} else {
     System.out.println("incorrect choice");
   }
 }
```

Question 3. L'affichage d'erreur n'est pas une bonne solution pour signaler à l'appelant que le choix est incorrect. Proposez une meilleure solution (sans la mettre en œuvre).

► *Il serait mieux de lever une exception, qui pourrait ainsi être attrapée par l'appelant.*
✠*1 point*

La méthode `toDeliverPerMonth()` retourne une instance de `MaskItem` contenant le masque choisi et le bon nombre d'exemplaires à délivrer par mois. La classe `MaskItem` est donnée ci après.

```
public class MaskItem {
  private Mask mask;
  private int quantity;
  public MaskItem(Mask mask, int quantity) {
    this.mask = mask;
    this.quantity = quantity;
  }
  public Mask getMask() {
    return mask;
  }
  public int getQuantity() {
    return quantity;
  }
}
```

Question 4. La méthode `toDeliverPerMonth` va déclencher en l'état une `NullPointerException` dans le cas où le masque n'a pas encore été choisi. Pour éviter cela, faites en sorte que s'il n'y a pas de masque choisi, une exception *ad hoc* de type `NoMaskChosenException` soit jetée. Vous donnerez aussi le code Java de `NoMaskChosenException`.

▶ *voir ci-dessous*
✠*1 point pour l'exception*
✠*1 point pour le throws*
✠*1.5 point pour le throw et sa condition de lancement*
✠*total 3.5 points*

```
public class NoMaskChosenException extends Exception {}
 public MaskItem toDeliverPerMonthCorr() throws NoMaskChosenException {
    if (chosenMask==null) {
      throw new NoMaskChosenException();
    }
    return new MaskItem(chosenMask, chosenMask.nbRequiredMasksPerMonth(
        nbHalfDaysOfWorkPerMonth));
  }
```

Les membres sont regroupés dans des équipe (classe `Team`) et chaque équipe est munie d'une méthode `toCommandMonthly` permettant de calculer la quantité à acheter par mois de chaque sorte de masque pour toute l'équipe. Le résultat est stocké dans une `HashMap` dont la clef est un masque `m` et la valeur le nombre de masques de type `m` à commander. Comme on le voit, la méthode `put` des `HashMap` prend en premier paramètre la clef, et comme second la valeur.

```java
public class Team {
  private ArrayList<Member> members=new ArrayList<>();

  public void register(Member m) {
    if (!members.contains(m)) members.add(m);
  }

  public HashMap<Mask, Integer> toCommandMonthly(){
    HashMap<Mask, Integer> result=new HashMap<Mask, Integer>();
    for (Member m:members) {
      MaskItem mi=m.toDeliverPerMonth();
      if (result.containsKey(mi.getMask())) {
        result.put(mi.getMask(), result.get(mi.getMask())+mi.getQuantity());
      }
      else {
      result.put(mi.getMask(), mi.getQuantity());
      }
    }
    return result;
  }
```

```java
public class CatalogMask {
private static Mask basicMask=new SingleUseMask(0.05f, FFPCategory.FFP1);
  public static Mask getBasicFFP1Mask() {
    return basicMask;
  }
}
```

Question 5. Modifiez le code de la méthode `toCommandMonthly` de manière à prendre en compte le fait que l'appel à la méthode `toDeliverPerMonth` est suceptible de jeter une exception de type `NoMaskChosenException`. Si une telle exception est jetée, c'est que qu'un membre de l'équipe n'a pas choisi de masque. Dans ce cas, on décide de lui commander le nombre nécessaire de masques basiques de type `FFP1`. On utilisera pour cela la classe `CatalogMask` et sa méthode `getBasicFFP1Mask()` qui retourne un tel masque.

▶ *voir ci-dessous*
✠*absence de throws : 0.5pts*
✠*déclaration de mi à l'extérieur du try (pas de pb de portée), pas de point si pas de try ! : 0.5*
✠*try avec ce qu'il faut dedans : 1 point*
✠*catch la bonne exception : 0.5 points*
✠*positionnement du maskitem dans le catch :0.5 points si affectation de qq chose de raisonnable à mi + 0.5 points pour l'appel à la méthode statique –¿ total 1 point*
✠*1 point pour la poursuite normale à positionner après le catch.*
✠*total : 4.5 points*

```java
public HashMap<Mask, Integer> toCommandMonthlyCorr(){
    HashMap<Mask, Integer> result=new HashMap<Mask, Integer>();
    for (Member m:members) {
      MaskItem mi;
      try {
        mi=m.toDeliverPerMonthCorr();
      } catch(NoMaskChosenException exc) {
        mi=new MaskItem(CatalogMask.getBasicFFP1Mask(), m.
            getNbHalfDaysOfWorkPerMonth());
      }
      if (result.containsKey(mi.getMask())) {
        result.put(mi.getMask(), result.get(mi.getMask())+mi.getQuantity());
      }
      else {
      result.put(mi.getMask(), mi.getQuantity());
      }
    }
    return result;
```

```
}
```

Question 6. Pour des raisons comptables, tout ce qui est acheté comme dépense de fonctionnement dans cette entreprise doit implémenter une interface `OperationalExpenditure` permettant d'obtenir le fournisseur (ici sous forme de chaîne de caractères pour simplifier), le taux de TVA appliqué (un float), le prix hors taxe et le prix TTC (se calculant à partir du prix hors taxe et du prix TTC). Mettez en place l'interface `OperationalExpenditure` et modifiez la classe `Mask` et/ou la classe `ReusableMask` et/ou la classe `SingleUseMask` de manière à ce que les masques implémentent bien l'interface `OperationalExpenditure`. Ne vous intéressez pas aux constructeurs.

► *interface :*
✠*entete 0.5*
✠*que des méthodes (ou un attribut pour la TVA) : 1.5 point*
✠*price méthode default avec corps : 1 point*
implémentation :
✠*entete de Mask : 0.5points*
✠*attributs : 0.5 points*
✠*implémentation des trois méthodes et pas de la default : 1 point*

✠*total : 5 points*

```java
public interface OperationalExpenditure {
  public String provider();
  public float tvaRate(); // ou on peut mettre un attribut, si on considere la TVA
      constante
  public float taxFreePrice();
  public default float price() {
    return taxFreePrice()*(1+tvaRate());
  }
}

public abstract class Mask implements OperationalExpenditure{
  private float taxFreeUnitaryPrice;
  private String provider;
  private float tvaRate;

  public Mask(String provider, float unitaryPrice, float tvaRate) {
    this.taxFreeUnitaryPrice = unitaryPrice;
    this.provider=provider;
    this.tvaRate=tvaRate;
  }
  public abstract int nbRequiredMasksPerMonth(int nbHalfDays);

  public String provider() {
    return provider;
  }
  public float tvaRate() {
    return tvaRate;
  }
  public float taxFreePrice() {
    return taxFreeUnitaryPrice;
  }
}
// on ne touche pas aux sous classes (sauf si on a fait un constructeur comme moi
    ici mais il ne fallait pas !)
```