Université de Caen Normandie
Département d'informatique
L3 informatique, 2020-2021
Unité INF5A1, Session 2
Le 30 juin 2021, 14h-16h

Documents autorisés. Les « import » ne sont pas demandés.

On souhaite créer une application permettant de gérer des factures, et on suppose que l'on dispose des types ci-dessous :

```
public abstract class AbstractListenableModel {
   private ArrayList<ModelListener> listeners;
   public AbstractListenableModel() {
      this.listeners = new ArrayList<>();
   }
   public void addListener(ModelListener l) {
      listeners.add(l);
      l.modelUpdated(this);
   }
   public void removeListener(ModelListener l) {
      listeners.remove(l);
   }
   protected void fireChange() {
      for (ModelListener l : listeners)
      {
         l.modelUpdated(this);
      }
   }
}
```

```
public interface ModelListener {

   void modelUpdated(Object source);
}
```

A. Partie Modèle

On dispose des deux interfaces définies ci-dessous, qui permettent de définir des factures contenant le nom de l'acheteur, le montant à payer, et qui sont écoutables (rappel : l'héritage d'interface permet d'hériter des méthodes de l'interface parent, et d'en définir un sous-type) :

```
public interface Facture {
   String getNom();
   double getPrix();
   void addListener(ModelListener l);
}
```

```
public interface FactureModifiable extends Facture{
   void setNom(String nom);
   void setPrix(double prix);
}
```

Question 1 : implémentation simple (3 points)

Ecrire une classe d'implémentation de FactureModifiable nommée FactureImpl.

Question 2 : decorator (4 points)

Ecrire un decorator de FactureModifiable nommé FactureAvecTaxe, qui permet de décorer une instance de FactureModifiable en lui ajoutant une certaine taxe lorsqu'on lui

demande son prix. Ce decorator prend en paramètre de constructeur, en plus d'une FactureModifiable, un double correspondant à la taxe à appliquer. Attention à bien implémenter les setters en les propageant vers l'objet décoré, et à bien transmettre les événements si l'objet décoré est modifié.

Question 3 : adapter (3 points)

On dispose par ailleurs du type suivant, et on suppose que l'on dispose de son implémentation BillImpl (son code n'est pas demandé) :

```
public interface Bill {
   String getName();
   double getPrice();
   void addListener(ModelListener l);
}
```

Créer l'adapteur BillToFacture permettant d'adapter le type Bill au type Facture. Comme un Bill est une facture américaine exprimée en dollars, et que Facture est en euros, cet adapteur prendra soin de faire la conversion correspondante (taux=1.1).

Question 4 : utilisation d'adapter et decorator (1 point)

Ecrire le code (une ligne) permettant de créer une instance de VueFacture sur une instance de Bill (utiliser BillImpl en supposant qu'il a une constructeur classique).

Ecrire le code (une ligne) permettant de créer une instance de Facture de « beurre » à 2 euros, décorée avec une taxe de 20% de TVA.

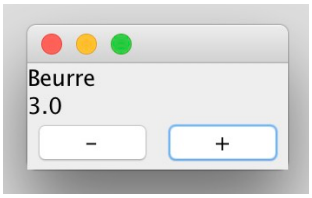
B. Partie interface graphique

Question 5 : Vue (4 points)

Créer la classe VueFacture, sous classe de JPanel, qui montre le nom et la valeur d'une instance de Facture avec des JTextField (rappel : la méthode setText(String t) permet de modifier son contenu). Voir le screenshot pour s'en inspirer. Cette vue doit bien sûr être réactive aux changements du modèle.

Question 6 : GUI (5 points)

Créer la classe FactureGUI, sous-classe de JFrame, qui est une vue-contrôleur sur une FactureModifiable (et non pas sur une Facture de base). Les boutons plus et moins permettent d'incrémenter ou de décrémenter la valeur du prix.



Exemple d'exécution avec le code : `new FactureGUI(new FactureImpl("Beurre",2));`
après avoir cliqué sur le bouton « + » une fois.