Contrôle continu no 2

Durée : 2H. Documents non autorisés. Les sections sont indépendantes. Les points sont globalements répartis (environ 5,5 points pour les parties 1, 2 et 3 ; 3,5 points pour la partie 4. Même si ce n'est pas explicitement demandé, quand vous voyez un lien entre les questions et des architectures ou concepts liés au cours, n'hésitez pas à les nommer et commenter. Tout commentaire bien placé vaut bonus ; mais tout commentaire hors sujet vaut malus (dans le doute mieux vaut ne rien dire).

1 Une architecture logicielle pour des éléments audio

On s'intéresse ici à la lecture de d'éléments audio (*AudioElement*)) stockés en version numérique dans des fichiers. Les éléments audio sont par exemple des chansons ; chaque chanson est représentée par une instance de la classe `Song`. On souhaite de plus mettre en place des listes de lecture, représentées par des instances de la classe `PlayList`. Les listes de lectures se composent de chansons, et/ou d'autres listes de lecture. Les listes de lecture et les chansons ont une durée, qui pour une chanson est initialement fournie en argument au constructeur de `Song`. La durée d'une liste de lecture est la somme des durées de ses composants. L'envoi du message `duration()` à un élément audio donne sa duré. Les chansons et les listes de lecture disposent d'une méthode `play()`, pour jouer la chanson ou toutes les chansons qu'elle contient dans le cas d'une liste de lecture. Pour simplifier ici, jouer une chanson sera réalisé en affichant le titre de la chanson au terminal. L'interface des éléments audio (comme les chansons et les listes de lecture) est fixée et donnée par le listing ci-dessous.

```
public interface AudioElement{
  void play();
  String name();
  Int duration(); }
```

Question 1. Donnez un diagramme de classes UML permettant de modéliser la situation décrite ci-dessus (ce diagramme fera apparaître les classes, associations et relations d'héritage, ainsi que pour chaque classe les constructeurs, méthodes et attributs nécessaires). Commentez.

Question 2. Donnez tous les éléments de code permettant de mettre en œuvre les méthodes `duration()` et `play()` pour les chansons et les listes de lecture.

Question 3. Considérez votre réalisation ci-avant comme une bibliothèque extensible (on pourra imaginer d'autres sortes d'éléments audio comme des enregistrements d'émissions de radio, par exemple). Indiquez si vous pensez que votre modélisation et votre code le permettent. Votre code utilise-t-il le concept d'affectation polymorphique. Expliquez. Si oui indiquez à quel(s) endroit(s) du code on en trouve un(des) exemples. Si non indiquez pourquoi ce n'est pas utile.

2 Une architecture logicielle pour des sources de données

On souhaite proposer une API permettant de fournir des sources de données qui peuvent être selon les besoins : cryptées, et/ou compressées. Les sources de données doivent implémenter l'interface simplifiée suivante :

```
public interface DataSource(){
    void write(String s);
    String read();
```

La méthode `write` écrit la chaîne `s` passée en argument dans la source de donnée, et la méthode `read` lit la source de donnée et retourne la chaîne qui s'y trouve. On ne s'intéresse pour l'instant qu'aux sources de données qui utilisent des fichiers (la donnée est lue/écrite dans un fichier). Les sources de données peuvent être cryptées et/ou compressées.
– Quand une source de donnée est cryptée, alors : **(i)** écrire une chaîne `s` dans la source de donnée consiste à crypter `s` et à écrire le résultat dans la source de donnée ; **(ii)** lire la source de donnée consiste à récupérer le contenu de la source de donnée, le décrypter et retourner le résultat ;
– Quand une source de donnée est compressée, alors : **(i)** écrire une chaîne `s` dans la source de donnée consiste à compresser `s` et à écrire le résultat dans la source de donnée ; **(ii)** lire la source de donnée consiste à récupérer le contenu de la source de données, le décompresser et retourner le résultat.

Question 4. Donnez un diagramme de classes UML définissant une architecture permettant de modéliser la situation décrite ci-dessus (ce diagramme fera apparaître les classes, associations et relations d'héritage, ainsi que pour chaque classe les constructeurs, méthodes et attributs nécessaires au vu de la situation décrite plus haut). Placer dans les classes qui vous semblent adéquates des méthodes : `crypt(s:String):String`, `uncrypt(s:String):String`, `compress(s:String):String` et `uncompress(s:String):String` qui assurent respectivement l'encryptage, le décryptage, la compression et la décompression d'une chaîne de caractères.

Question 5. Donnez un diagramme d'objet illustrant quels objets (et quels liens) représentent une source de données cryptée et compressée basée sur un fichier.

Question 6. Donnez les éléments de code permettant la mise en œuvre des méthodes de nom `read` et `write` déclarées dans l'interface `DataSource`, en supposant disposer, sans les écrire, des implémentations des méthodes d'encryptage/décryptage et compression/décompression listées précédemment.

3 Tester une architecture pour panier/produit d'un site marchand

On dispose (listing ci-dessous) d'une classe `Produit` dotée d'une méthode `prix()` dont le code n'est pas donné ici mais qui calcule le prix du produit en euros en utilisant un service web permettant de récupérer le taux de conversion de l'unité monétaire utilisée vers l'euro (si l'appel au service web échoue, une exception de type `ConnectException` est lancée). On dispose également (voir listing ci-dessous) d'une classe `ItemPanier` possédant aussi une méthode `prix()`. On souhaite tester la méthode `prix()` d' `ItemPanier`.

```
public class Produit{
    protected float prix;
    protected String uniteMonetaire;
    ...
    public float prix() throws ConnectException { ... }
}

public class ItemPanier{
    protected int qte;
    protected Produit produit;

    // retourne le prix de l'item de panier ou un nombre negatif si le calcul n'a pas abouti
    public float prix() {
        float prixUnitaire;
        try{ prixUnitaire=produit.prix(); }
        catch (ConnectException c){
          prixUnitaire=-1; }
        return qte*prixUnitaire; }
}
```

Question 7. Ecrivez deux méthodes de test avec *JUnit* et *Mockito*. La première méthode de test teste qu'une exécution de la méthode `prix()` de `ItemPanier` retourne bien une valeur négative dans le cas où l'exécution de `produit.prix()` signale (lance) une exception de type `ConnectException`. La deuxième méthode de test teste que cette même méthode `prix()` fonctionne correctement (fait le bon calcul) dans le cas où produit est un produit en dollars américains ("USD"), et que `produit.prix()` a bien retourné le prix converti du `produit`, sachant que vous n'êtes pas censés connaître le taux de conversion dollars américains/euros. On supposera que tous les imports nécessaires à l'utilisation classique de *JUnit* et *Mockito* ont bien été faits dans la classe où se situent les 2 méthodes de test.

Question 8. On peut considérer la classe *ItemPanier* comme un micro-framework, utilisant un paramétrage par composition, capable de fonctionner avec toutes sortes de produits différents, éventuellement définis par des sous-classes de la classe `Produit`. On souhaite ainsi faire tourner les tests précédents pour des instances de sous-classes de `Produit`. Si nécessaire (si le code existant ne traite pas déjà la question), ajoutez à la classe `ItemPanier` une solution pour réaliser des injections de dépendance au sens de la technologie des *frameworks*. Si necessaire (idem), ajoutez à la classe `ItemPanier` une solution pour réaliser l'inversion de contrôle au sens de la technologie des *frameworks*.

4 Lignes de produits

On souhaite créer une ligne de produits logiciels pour générer différents programmes de simulation d'ordinateurs proches les unes des autres. La Figure 1 présente un *feature model* simplifié modélisant les ordinateurs et de leurs variations possibles.

Question 9. Donnez un exemple de configuration valide, expliquez. Donnez un exemple de configuration non valide. Expliquez pourquoi la configuration que vous donnez n'est pas valide.

Question 10. Quel est le problème de modélisation concernant la feature "écran" ? Proposez une solution pour corriger ce problème sans modifier la sémantique de configurations du modèle.

Question 11. Expliquez succintement la principale différence entre un framework et une ligne de produits logiciels.

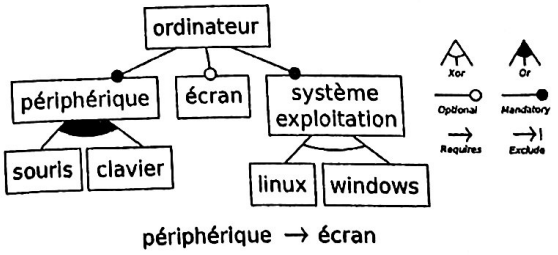

Fig. 1 : un *feature model* pour configurer des variantes d'ordinateurs.