Contrôle de Programmation

Mercredi 11 janvier 2012
Durée : 2h

Ce sujet comporte 3 pages.
Documents autorisés : notes de cours, TD et TP, spécifications des types abstraits.

1. Répertoire téléphonique.

Le but de cette partie est de permettre à une entreprise de gérer les répertoires téléphoniques de ses employés.

Un répertoire téléphonique permet de mémoriser le nom et le numéro de téléphone de personnes appelées *contacts*. Nous considérons qu'il n'y a pas d'homonymes : ainsi le nom permet d'identifier de manière unique un contact.

Un répertoire téléphonique dispose des opérations suivantes :
- estPresent : permet de savoir si un nom correspond à un contact dans le répertoire ;
- getNumero : obtenir le numéro de téléphone d'un contact à partir de son nom ; le contact doit être présent dans le répertoire ;
- enregistrer : enregistrer un nouveau nom et le numéro de téléphone associé dans le répertoire ; le nom ne doit pas être présent dans le répertoire ;
- modifier : modifier le numéro de téléphone d'un contact en précisant le nom ; le contact doit être présent dans le répertoire ;
- supprimer : supprimer un contact en précisant son nom ; le contact doit être présent dans le répertoire ;
- nbContacts : déterminer le nombre de contacts enregistrés.

Nous considérons plusieurs sortes de répertoires téléphoniques :
- une simple page sur laquelle les nouveaux contacts sont ajoutés à la fin de la page ;
- une page ordonnée sur laquelle les contacts sont rangés dans l'ordre alphabétique de leur nom ;
- un carnet composé de 26 pages ; chaque page correspond à une lettre de l'alphabet et permet de ranger les contacts dont le nom commence par cette lettre.

1. Dessinez un diagramme de classes (sans attribut ni méthode) qui fait apparaître les notions de Répertoire, SimplePage, PageOrdonnée et Carnet et les relations qui les lient.
2. Pour quelle raison est-il important pour l'entreprise que les 3 sortes de répertoires possèdent une spécification commune ?
3. Écrivez en java l'interface Répertoire, en précisant les pré-conditions des opérations lorsque c'est nécessaire.

   Remarque : vous n'utiliserez pas d'autre structure de données que celles de l'API java ;

   Rappel : il existe deux implémentations de l'interface List<E> :
   - LinkedList<E> : liste chaînée ; parcours bidirectionnel avec itérateur ;
   - ArrayList<E> : tableau ; parcours bidirectionnel avec itérateur et accès direct avec indice.

   Pour programmer la classe SimplePage, on décide d'utiliser une *liste chaînée* dont les éléments sont de type Contact (voir Annexe) ;
4. Donnez la déclaration de la classe SimplePage et programmez les fonctionnalités suivantes :
   (a) déclaration des attributs en expliquant leur rôle précis ;
   (b) constructeur ;

   (c) méthode (protected) chercher qui cherche un nom donné dans la liste des contacts ; cette méthode renvoie null si le nom cherché est absent ; s'il est présent, elle renvoie un itérateur it tel que it.previous() donne l'élément trouvé ;
   (d) méthode estPresent ;
   (e) méthode modifier ;

5. Donnez la déclaration des classes PageOrdonnée et Carnet et programmez les fonctionnalités suivantes :
   (a) déclaration des attributs en expliquant leur rôle précis ;
   (b) constructeur ;
6. Indiquez quelles sont les méthodes de la classe PageOrdonnee qui diffèrent de celles de SimplePage en expliquant ce qui change ; faites-de même pour Carnet.
7. Ajoutez dans le diagramme précédent une classe dont le rôle est de mutualiser le plus grand nombre de fonctionnalités des trois classes SimplePage, PageOrdonnée et Carnet ; quelle est la particularité de cette classe ?
8. Donnez la déclaration de cette nouvelle classe en indiquant :
   (a) les méthodes qui *ne peuvent pas* être programmées dans cette classe ;
   (b) les méthodes qui peuvent être programmées dans la classe ; vous donnerez en outre le corps de la méthode supprimer.
9. Modifiez la déclaration de la classe Carnet pour l'adapter à la nouvelle organisation puis :
   (a) listez les méthodes qui doivent être programmées dans cette classe ;
   (b) programmez la méthode enregistrer ; on supposera définie la méthode
   protected int numeroPage(**char** c); qui donne le numéro de page correspondant à une lettre de l'alphabet ;
   par ailleurs, on rappelle la méthode **char** charAt(**int** ic); de la classe String qui donne le caractère situé à l'indice ic d'une chaîne this, avec $0 \leq ic < longueur\ de\ la\ chaîne$.
10. Expliquez ce qu'il faut modifier dans la classe Carnet pour utiliser des pages ordonnées plutôt que des simples pages, ou le contraire, selon vos choix précédents.
11. Expliquez ce qu'il faut modifier dans la hiérarchie de classes pour exprimer qu'un carnet peut être composé de simples pages ou de pages ordonnées mais pas de carnets ; dessinez le diagramme ainsi modifié.

2. Cours

1. Quels sont les points communs et les différences entre une interface et une classe abstraite ?
2. Expliquez en quelques lignes le mécanisme de liaison dynamique et ce qu'il apporte en programmation objet.

Annexe

class Contact

```
public class Contact {
 private String a_nom, a_numero;

 public Contact(String p_nom, String p_numero) { a_nom = p_nom ; a_numero = p_numero; }
 public  String getNom()                        { return a_nom; }
 public  String getNumero()                     { return a_numero; }
 public    void setNumero(String p_nouveau)     { a_numero = p_nouveau; }
 protected void setNom(String p_nouveau)        { a_nom = p_nouveau; }
 public  String toString()                      { return a_nom + ":" + a_numero; }
}
```

Interface ListIterator<E>

Ci-dessous, les commentaires de spécification en français de quelques opérations importantes.

```
// opérations de parcours
// Renvoie vrai s'il reste un élément lors d'un parcours "en marche avant"
boolean         hasNext();

// renvoie le prochain élément de la liste et avance le curseur
// @pre : hasNext() est vérifié
E       next();

// Renvoie vrai s'il reste un élément lors d'un parcours "en marche arrière"
boolean         hasPrevious();

// renvoie l'élément précédent de la liste et recule le curseur
// @pre : hasPrevious() est vérifié
E       previous();

// modification de la liste
// Insère l'élément nouveau à la position du curseur
void    add(E nouveau);

// Remplace l'élément renvoyé par le dernier next() ou previous() par l'élément nouveau
void    set(E nouveau);

// Supprime de la liste l'élément renvoyé par le dernier next() ou previous().
 void   remove();
```