USTL — Licence STA Mention Informatique — Semestre 4

UE Programmation Orientée Objet

Examen première session – 16 juin 2009

2 heures - notes et polycopiés de cours/TD/TP autorisés
livres, calculatrices et portables interdits
dictionnaires de langue autorisés pour candidats étrangers

▷ Les 3 exercices sont totalement indépendants et l'ordre dans lequel ils apparaissent ne préjuge pas de leur difficulté réelle ou supposée.

Exercice 1 : Gènes

Tous les types de cet exercice sont à placer dans un paquetage `genetic`.

Un gène est une séquence d'ADN. On représente un gène comme une séquence de bases nucléiques représentées par les lettres `A`, `C`, `T` et `G`.

Q 1 . Définissez le type énuméré `Base` ayant 4 valeurs : `A`, `C`, `T` et `G`

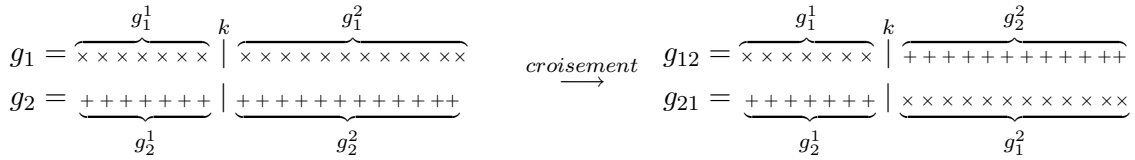
On va représenter un gène par la donnée d'un tableau de `Base`.

Q 2 . Donnez le code d'une classe `Gene` dont le constructeur prend en paramètre un tableau d'objets `Base`.

Vous définirez dans cette classe un accesseur pour ce tableau de bases, une méthode qui retourne la taille d'un gène, c'est-à-dire la longueur de sa séquence de bases, une méthode `toString` qui retourne la séquence de bases ainsi qu'une méthode `equals` : deux gènes sont dits égaux si ils ont la même séquence de bases..

Q 3 . Réaliser le croisement de deux gènes g_1 et g_2, de même longueur, à partir d'une position k consiste à produire deux nouveaux gènes à partir de g_1 et g_2 ainsi :

- on coupe chaque gène g_1 et g_2 à la même position k pour obtenir deux portions g_1^1, g_1^2 et g_2^1, g_2^2
- on crée les nouveaux gènes en "recollant" les portions g_1^1 et g_2^2 d'une part et g_2^1 et g_1^2 d'autre part.



Q 3.1. Donnez le code de la méthode `croisement` de la classe `Gene` dont la documentation est :

```
/** crée 2 nouveaux gènes par croisement à partir d'une position
*  donnée
* @param gene le gène avec lequel on croise celui-ci, supposé
*  de même longueur
* @param k la position de croisement
* @result le tableau des 2 nouveaux gènes créés par croisement à
*  partir de la position k de gene avec celui-ci
*/
```

On ne fera pas la vérification dans le code que les 2 gènes sont de la même longueur, ni que k est une position valide.

Q 3.2. Que se passera-t-il si les 2 gènes ne sont pas de la même longueur ou si k n'est pas une position valide ?

Q 4 . On appelle population de gènes une liste de gènes tous de même taille, supposés en nombre pair. La reproduction d'une population p de gènes consiste à produire une nouvelle population p'. Pour cela, on réalise le croisement, à partir d'une position k tirée aléatoirement, de gènes de p pris deux à deux également au hasard. Chaque gène de p intervient une et une seule fois dans la création d'un gène de p'.

Donnez le code java de la méthode `reproduit` d'une classe `Genetique` :

```
public List<Gene> reproduit(List<Gene> population)
```

qui réalise cette opération, on ne vérifiera pas que population est bien de taille paire, ceci est considéré comme une hypothèse de départ toujours vérifiée, ainsi que le fait que tous les gènes de la population sont de même taille.

Vous trouverez en annexe un extrait de la documentation sur la classe `java.util.Random`.

Exercice 2 : Identification

On va s'intéresser dans cet exercice à un système (très simplifié) d'accès à un compte basé sur une identification classique de type identifiant/mot de passe. L'utilisateur voulant accéder à un compte fournit un identifiant et un mot de passe. Si le mot de passe est bien celui associé à cet identifiant l'accès est accordé, dans le cas contraire il est refusé. Après trois refus consécutifs, le compte est désactivé et il n'est alors plus possible de s'y connecter.
Dans cet exercice, identifiant et mot de passe seront des objets de type `String`.
On stocke dans une table de hachage les associations entre les identifiants et les informations associées que nous appellerons *login*. Cette table sera appelée dans la suite `tableLogin`.
De manière simplifié, dans cet exercice, un *login* est défini par la donnée du mot de passe qui lui est associé, d'un état activé/désactivé, du nombre de refus de connexion consécutifs et de l'objet compte auquel de login donne accès.
Les comptes seront des objets d'un type `Compte` que l'on supposera défini ainsi :

Compte
+ toString(): String + equals(o : Object) : boolean

Q 1 . Donnez le diagramme de classe UML détaillé d'une classe `Login` permettant de représenter les *login*, sachant que les valeurs des attributs mot de passe et compte sont donnés à la création des objets et que les logins sont activés par défaut. On veut disposer des accesseurs sur ces attributs, ainsi que des méthodes permettant de modifier l'état et le nombre de refus consécutifs.

On considère maintenant une classe `Connexion` qui dispose de l'attribut `tableLogin` défini précédemment et d'un autre attribut `listeComptesConnectés` qui est une liste regroupant l'ensemble des objets de type `Compte` connectés.

Q 2 . Donnez la définition des attributs `tableLogin` et `listeComptesConnectes`.

Q 3 . Définissez une méthode `connexion` de cette classe qui prend en paramètre un identifiant *id* et une proposition de mot de passe *mdp*. Cette méthode a pour objet de vérifier si *mdp* est bien le mot de passe défini dans le login associé à *id*, si ce dernier est un identifiant connu. Si c'est le cas le compte associé au login est ajouté aux comptes connectés.

La méthode lève une exception `LoginDesactiveException` si *id* est associé à un login désactivé.

La méthode lève une exception `LoginInconnuException` si *id* n'existe pas dans `tableLogin`.

La méthode lève une exception `ConnexionRefuseeException` si *mdp* n'est pas le mot de passe défini dans le login associé à *id*. De plus, dans cette situation le compte sera désactivé à l'occasion du troisième refus consécutif.

Les trois classes d'exception mentionnées ont un constructeur qui prend comme seul paramètre une chaîne de caractères qui est le message de l'exception.

Q 4 . Donnez le code d'une méthode `afficheIdentifiants` de `Connexion` qui affiche la liste des identifiants connus et qui indique, pour chacun, si il est activé ou désactivé et connecté ou non.

Exercice 3 : Parcours d'arbres

On s'intéresse à des arbres binaires dont les nœuds sont étiquetés par des chaînes de caractères.
On dispose de la classe `ArbreBinaire` définie ainsi :

ArbreBinaire
- e : String - g : ArbreBinaire -d : ArbreBinaire + ARBRE_VIDE : ArbreBinaire
- ArbreBinaire() + ArbreBinaire(e :String, g: ArbreBinaire, d: ArbreBinaire) + racine() : String + gauche() : ArbreBinaire + droit() : ArbreBinaire + estVide() : boolean

Les méthodes `racine`, `gauche` et `droit` lèvent une exception `ArbreVideException` si elles sont appelées sur la constante statique `ARBRE_VIDE` représentant l'arbre vide.
On s'intéresse au parcours de tels arbres binaires. Différents parcours sont possibles : infixe, postfixe et préfixe.
Lors du parcours d'un arbre on souhaite pouvoir réaliser une opération sur chaque nœud de l'arbre.
Ces opérations sont définies par l'interface :

≪ interface ≫ *Operation*
+ traitement(s : String) : String

Q 1 . Définissez une opération dont le traitement affiche la chaîne passée en paramètre et retourne cette valeur.

Q 2 . Définissez une opération dont le constructeur est paramétré par deux caractères c_1 et c_2 et dont le traitement consiste à remplacer toutes les occurrences du caractère c_1 du paramètre s par le caractère c_2, la nouvelle chaîne obtenue est le résultat du traitement.
Une méthode utile de la classe `String` est décrite en annexe.

On définit pour ces parcours l'interface :

≪ interface ≫ *ParcoursAB*
+parcours(ArbreBinaire ab, Operation op) : ArbreBinaire

Le résultat de cette méthode est le nouvel arbre obtenu à partir de l'arbre `ab`, où chaque nœud rencontré (dans l'ordre du parcours) subit la transformation `op`.

Q 3 . Donnez le code d'une classe `ParcoursABInfixe` qui permet le parcours des arbres binaires selon un parcours infixe en appliquant une opération.

Q 4 . On suppose définies de manière équivalente les classes `ParcoursABPrefixe` et `ParcoursABPostfixe`.

Donnez les lignes de code qui, pour une variable ab de type `ArbreBinaire` supposée initialisée, permettent :

- un affichage préfixe de ab,
- un affichage postfixe de ab,
- d'obtenir un nouvel arbre bb à partir de ab en remplaçant par des `'b'` tous les `'a'` des étiquettes (ou éléments) des nœuds de ab.

Annexe

Classe `java.util.Random`

Random() Creates a new random number generator.

int nextInt(int n) Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

Classe `java.lang.String`

String replace(char oldChar, char newChar) Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.