Université Denis Diderot Paris 7
UFR d'informatique
L3 Licence d'informatique

3/6/2013
souten.

Examen de Programmation

(janvier 2013)

durée deux heures ; tous les documents sont interdits

2 pages

Le barême est donné à titre indicatif (total 26)

1. On considère le programme suivant :

```
class F {
    void f(){System.out.print("F.f() ");this.g();}
    void g(){System.out.println("F.g() ");}
}
class Fbis extends F{
    void f(){System.out.print("Fbis.f() ");this.g();}
    void g(){System.out.println("Fbis.g() ");super.f();}
}
```

   Le code suivant : `F f=new Fbis(); ((F)f).f();`
   n'est pas correct, que provoque-t-il et pourquoi ? (2 points)

2. Soit :

```
class A{}
class B extends A{}
```

   (a) Que provoque le code "`A[] tab1= new B[10]; tab1[0]= new A();`". S'il provoque une erreur, expliquez laquelle et pourquoi ? (2 points)

   (b) Que provoque le code "`ArrayList<A> L1= new ArrayList<B> (); L1.add(new A());`". S'il provoque une erreur expliquez laquelle et pouquoi ? (1,5 points)

3. On considère un type de données `XObj` qui contient des éléments de type `Object`. Un `XObj` contient les méthodes suivantes : `mettre`, `prendre`, `lire`, `taille`, `table`. Si `X` est un `XObj`, `mettre(o)` ajoute un objet `o` à `X`, `prendre` retourne un objet contenu dans `X` et supprime cet objet de `X`, `lire` retourne un objet contenu dans `X` sans le supprimer, `taille` retourne le nombre d'éléments contenus dans `X` et `table` retourne un tableau de `Object` contenant les éléments contenus dans `X`. Dans le cas où une méthode ne peut être appliquée (par exemple, `lire` si `X` est vide), la méthode devra lancer une exception qu'on appellera `Echec`.

   (a) Après avoir défini `Echec`, définir une interface correspondant à `XObj`. (1 point)

   (b) On veut maintenant réaliser des implémentations de `XObj` telles que `prendre` et `lire` concernent l'élément qui a été mis (par la méthode `mettre`) le plus *récemment* (une telle implémentation correspond à une pile).

      i. Définir une classe `PileT` qui réalise une telle implémentation de `XObj` en utilisant un tableau de `Object` (on supposera que la taille du tableau est donnée par une variable d'instance `TMAX` qui sera initialisée par un constructeur). (2 points)

      ii. Définir une classe `PileL` qui réalise une telle implémentation de `XObj` en utilisant la classe suivante :

```
class Noeud{
    Object val;
    Noeud suivant;
    Noeud(Object o){val=o;suivant=null;}
    Noeud(Object o, Noeud i){val=o;suivant=i;}
}
```

(2 points)

iii. Après le code : "`XObj x=new PileT(10); XObj y=x; x.mettre("a");x.mettre("b"); y.prendre();`", que retournera "`x.prendre()`" ? (1,5 points) Comment assurer que, si `y` a *initialement* le même contenu que l'objet `Pile` référencé par `x`, ensuite, les opérations `mettre` et `prendre` sur `y` ne modifient pas `x` (on pourra utiliser la méthode `clone`) (2 points)

iv. Le code suivant : `XObj x=new PileL(); x.mettre("a"); x.mettre(1);` provoque-t-il une erreur à la compilation ? à l'exécution ? (1 point)

4. Dans cette partie on va "paramétrer" le type de données par une classe `E`.

   (a) Définir une interface `XClasse` similaire à `XObj` mais pouvant contenir des éléments d'un type paramètre `E` au lieu de `Object` (on aura les mêmes méthodes que pour `XObj`). On veut que cette interface soit la plus "générale" possible : par exemple si `F` est une extension de `E`, alors pour une `XClasse` construite sur un type de données `E`, on doit pouvoir mettre un objet de classe `F`). On veut de plus que `XClasse` implémente l'interface `Iterator<E>`. On rappelle que l'interface `Iterator<E>` est définie pour faire des itérations sur un type de données : son but est de retourner successivement les divers éléments contenus dans ce type de données. Elle contient les déclarations des méthodes :
   - `boolean hasNext()` qui retourne `true` si et seulement si il y a d'autres éléments à retourner dans l'itération,
   - `E next()` retourne l'élément suivant pour l'itération,
   - `void remove()` qui supprime de la structure de données le dernier élément retourné par `next()`. On ignorera cette méthode et l'implémentaion de `remove` sera toujours vide (`void remove(){}`).

   Dans la suite, les implémentations de `Iterator<E>` ne devront pas être "destructives" : après des appels de `next`, l'état du type de données `XClasse` ne doit pas être modifié. (2 points)

   (b) On veut maintenant réaliser des implémentations de `XClasse` telles que `prendre` et `lire` concernent l'élément qui a été mis (par la méthode `mettre`) le plus *anciennement* (une telle implémentation correspond à une file).

      i. Définir une classe `FileT` qui réalise une telle implémentation de `XClasse` en utilisant un tableau de `Object` (on supposera que la taille du tableau est donnée par une variable d'instance `TMAX` qui sera initialisée par un constructeur). On demande de plus que pour cette implémentation `mettre` ou `enlever` ne déplace pas d'éléments du tableau. Est-il possible, au lieu d'un tableau de `Object`, d'utiliser un tableau de `E`, si oui comment, si non pourquoi. (2 points)

      ii. Définir une classe `FileL` qui réalise une telle implémentation de `XClasse` en utilisant une classe interne `Nd` similaire à `Noeud` définie précédemment mais avec un paramètre de type `E` au lieu de`Object`. (2 points)

      iii. Définir une fonction statique `afficher` s'appliquant à un itérateur qui affiche tous les éléments retournés par `next()` jusqu'à ce que `hasNext()` retourne `false`. Utiliser cette fonction pour définir une fonction qui affichera le contenu d'une `XClasse`. (2 points)

5. On utilise maintenant `XClasse` pour qu'une thread transmette des informations à une autre thread.

   Définir une thread `Prod` et une thread `Cons` partageant `F`, objet de la classe `XClasse<Integer>`, telles que (1) `Prod` tire au hasard des entiers entre 1 et 1000 (on rappelle que `Math.random()` retourne un "double" compris entre 0.0 et 1.0) et les met successivement dans `F` et (2) `Cons` lit successivement les éléments de `F` et les affiche sur la sortie standard. On veillera à assurer que `Cons` ne prend des éléments de `F` que quand de tels éléments sont disponibles. (5 points)