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

Examen de Programmation

(janvier 2010)

durée trois heures ; tous les documents sont interdits

2 pages

Le qcm est sur des feuilles à part. Le barème est donné à titre indicatif.

1. Le type abstrait de données *Pile* est défini par les opérations *pop* (qui dépile l'élément au sommet de la pile), *push* (qui empile un élément), et *estVide* (qui teste si la pile est vide).
   Définir une interface java paramétrée par un type `T` pour une pile de `T`. (1pt).
2. On considère la classe `Item` paramétrée par un type :

```
class Item<T> {
    private T Val;
    private Item Suiv;
    public Item(T t) {
        Val = t;
        Suiv = null;
    }
    public T getVal() {
        return Val;
    }
    public Item getSuiv() {
        return Suiv;
    }
    public void setVal(T t) {
        Val = t;
    }
    public void setSuiv(Item n) {
        Suiv = n;
    }
}
```

   Définir une implémentation `PileItem` (paramétrée par un type `T`) qui implémente l'interface `Pile` précédente en uilisant la classe `Item`. (2pts)
   Pour cette question et la suivante, l'implémentation devra bien sûr assurer les propriétés usuelles des opérations `pop`, `push` et `estVide`, et on ne se préoccupera pas des cas où ces opérations ne sont pas définies.
3. On veut définir une implémentation `PileTab` (paramétrée par un type `T`) qui implémente l'interface `Pile` précédente à partir d'un tableau de taille fixe.
   Est-il possible de représenter la pile par un tableau de `T` ? Si ce n'est pas le cas, comment peut-on contourner le problème et avec quel(s) inconvénient(s) ? (1pt)
   Définir une implémentation `PileTab` qui contiendra un constructeur ayant un paramètre entier indiquant la taille maximale de la pile (et du tableau utilisé pour sa représentation).(1pt)
4. Si `p` est déclaré comme `Pile<Integer>` (et instancié par un objet d'une classe qui implémente `Pile`), `p.push("un")` est-il possible (=ne provoque pas d'erreur à la compilation) ? Même question pour `p` déclaré comme `PileTab<Integer>` et `PileItem<Integer>`.(1pt)

Si p est déclaré comme `Pile` (et instancié par un objet d'une classe qui implémente `Pile`), `p.push("un")` est-il possible (=ne provoque pas d'erreur à la compilation) ? Même question pour p déclaré comme `PileTab` et `PileItem`.(1pt)

5. On s'intéresse au traitement des opérations non définies (pop sur une pile vide et push sur une pile pleine (pour `PileTab`)), pour cela on définira deux exceptions `PileVideException` et `PilePleineException`.
   Modifier les classes précédentes de façon à ce qu'en cas d'opérations non définies les exceptions `PilePleineException` ou `PileVideException` soient lancées. En maintenant la hiérarchie des classes est-il possible de ne pas modifier l'interface `Pile` ? Les modifications faites ont-elles des conséquences sur les programmes déjà écrits qui utilisent `Pile`, `PileItem` ou `PileTab` ? (2pts)
6. Définir une méthode statique `cmp` ayant comme argument deux piles et qui vérifie que ces deux piles sont composées des mêmes éléments.(0,5pt)
   En quoi cette question peut être considérée comme ambiguë ? (0,5pt)
   La méthode que vous avez écrite modifie-t-elle les piles qui lui sont passées en argument ? Si c'est le cas proposez des solutions pour contourner le problème.(1pt)
7. Définir une extension de `PileTab` qu'on appellera `PileTabUn` pour laquelle :
   (a) le clonage est possible et crée une copie de la pile. On demande de procéder de façon à ce que les invariants classiques des piles s'appliquent aussi bien à la copie qu'au clone. En particulier, pour le type de données pile, pour une une pile p, un *p.pop()* qui suit un *p.push(v)* (sans autres opérations sur la pile p entre ce *push* et ce *pop*) le *pop* doit retourner un v et, par exemple, dans votre implémentation, si `c` est un clone de `a`, alors pour un morceau de code comme `c.push(y) ; a.push(x) ; c.pop() ; a.pop()` ; alors `c.pop()` retourne `y` et `a.pop()` retourne `x`. (1pt)
   (b) il est possible pour un objet `p` de la classe `PileTabUn` d'utiliser une construction de la forme `for(Object o : p){ ... }`. (On rappelle qu'il faut pour cela que la classe implémente l'interface `Iterable` qui déclare la méthode `iterator()` retournant un `Iterator` définissant les méthodes `hasNext`, `next`, et `remove`) (2pts)
8. Définir une classe `PileTabBis` extension de `PileTabUn` telle qu'en cas d'un `push` sur une pile pleine au lieu de lancer `PilePleineException`, la taille du tableau de données pour représenter la pile soit augmentée. Pour cela, on redéfinira (entre autres choses) `push` de façon à appeler `push` de `PileTabUn` et à attraper l'exception `PilePleineException` et, dans ce cas, augmenter la taille du tableau. (2pts)
9. On va dans ce qui suit utiliser les piles définies précédemment de façon concurrente.
   (a) Définir une extension `PileThread` de `Thread` contenant une pile `p`, et telle qu'un objet de cette classe empile successivement tous les entiers de 0 à 99. Cette classe contiendra un constructeur ayant comme argument une `Pile` qui sera affectée à `p`. (1pt)
   (b) Définir un code qui crée et lance deux threads objets de `PileThread` partageant une pile `p` initialement vide. (0,5pt)
   (c) Écrire un programme réalisant la même chose en utilisant l'interface `Runnable` sans créer explicitement d'extension de `Thread`. (1pt)
   (d) Si on considère `p` après la terminaison de ces deux threads, est-on sûr que la pile contiendra 200 éléments ? Si ce n'est pas le cas comment modifier l'implémentation de pile utilisée pour le garantir ? (2pt)
   (e) On définit maintenant une méthode `pushpop` qui réalise un `push` suivi d'un `pop`. Si plusieurs threads travaillent sur la même pile, est-on sûr que la valeur retournée par le pop soit celle qui était en argument du `push` ? Comment le garantir ? (3pts)