Université Paris-Diderot Paris
UFR d'informatique
L3 Licence d'informatique

Examen de Compléments de Programmation Orientée Objet

(deuxièem session 21 juin 2016
Durée deux heures ; tous les documents sont interdits
Le barème est donné à titre indicatif : total 30
3 pages

Exercice 1 : Une *Pile* (LIFO : Last In First Out) est une strcuture de données contenant essentiellement deux méthodes : *push* et *pop*. Pour un objet `o`, `push(o)` met `o` dans la pile `pop()` retire l'objet de la pile qui a été mis dans la pile le plus récemment. Par exemple dans la séquence des opérations `push(o1); push(o2);v1=pop();v2=pop();` la pile sera dans le même état qu'au début, `v1` contiendra `o2` et `v2` contiendra `o1`.

On suppose que la pile ne peut contenir plus de `TailleMax` éléments. La pile contiendra aussi une méthode `isEmpty` (la pile est vide) retournant un booléen et une méthode `isFull` (la pile est pleine) retournant un booléen ainsi qu'une méthode `top` retournant sans le supprimer le dernier objet qui a été mis dans la pile).

Quand la pile est vide, l'opération `pop` retournera null. De même quand la pile est pleine `push` retournera `null` (sinon `push` retourne l'objet qui a été mis dans la pile.

1. (a) Ecrire une interface `Pile` correspondant à cette structure de données. (1pt)

   (b) En utilisant comme classe *interne statique* la classe ci-dessous :

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

   définir une implémentation `PileList` de `Pile`. (2pts)

   (c) Définir une implémentation `PileTab` de `Pile` en stockant les éléments de la File dans un tableau de `Object`. La taille de ce tableau sera initialisée par un constructeur. (On prendra soin à ce que `push` et `pop` se fassent en temps constant.) (2pts)

   (d) Si maintenant on veut que la `Pile` soit générique (paramatrée par un type `T`) quelles modifications doit-on faire (préciser ces modifications pour l'interface et les deux implémentations proposées précédemment). (2pts)

   (e) (bonus) Proposez une modification qui éviterait à vos méthodes de retourner `null` (afin d'éviter des bugs souvent difficile à détecter).

2. Dans la suite on va utiliser les structures de données définies précédemment dans un contexte de concurrence (multi-thread).

   Un *producteur* est un thread qui utilise une instance de `Pile` et insère (push) dans la `Pile` des entiers de 1 à 100. Un *consommateur* est un thread partageant cette instance de `Pile` et qui extrait (pop) indéfiniment de la file les données et les écrit sur la sortie standard. Toutes les données des producteurs doivent être consommées par les consommateurs.

   (a) Ecrire une classe `Producteur` et une classe `Consommateur`, héritant toutes deux de `Thread`, pour réaliser les threads décrits ci-dessus. Pour produire (respectivement consommer) une valeur, le producteur (resp. le consommateur) feront, dans leur méthode `run()`, une boucle while jusqu'à ce que la valeur soit mise dans la pile (resp. prise de la pile). (On rappelle que `pop` et `push` retournent `null` en cas d'échec c'est-à-dire respectivement si la pile est vide et si la pile est pleine). (2pts)

(b) Ecrire un main qui instancie une pile de type `PileTab` et lance un thread producteur et un thread consommateur partageant cette instance de pile. Si vous exécutez ce main, peut-on garantir que la sortie standard contiendra tous les entiers entre 1 et 100? Justifiez votre réponse. On pourra éventuellement modifier les implémentations dans une classe `PileSynch` en utilisant des verrous `synchronized` de façon à assurer que tous les entiers entre 1 et 100 seront affichés sur la sortie standard (3pts).

(c) On suppose maintenant qu'il y a plusieurs producteurs et plusieurs consommateurs partageant la même pile `PileSynch` de `TailleMax=3`. Quel est, du point de vue des performances, l'inconvénient de la solution proposée dans la question précédente?

Pour éviter ces inconvénients, modifier l'implémentation dans une classe `PileConcur` utilisant `wait` et `notifyAll`). (3pts)

Exercice 2 : Rappelons que dans l'interface `Stream<T>` :

- `Stream<T> filter(Predicate<? super T> p)` : retourne le *stream* où l'on ne conserve que les éléments satisfaisant p.
- `void forEach(Consumer<? super T> action)` : applique l'action à chaque élément.
- `boolean anyMatch(Predicate<? super T> p)` : retourne vrai si et seulement si `p` est vrai pour au moins un élément du *stream*.
- `long count()` : retourne le nombre d'éléments dans le stream courant.
- `<R> Stream<R> map(Function<? super T,? extends R> mapper)` : retourne le *stream* des éléments obtenus en appliquant la fonction `mapper` à chaque élément du stream courant.
- `T reduce(T id, BinaryOperator<T> op)` : opère une réduction sur le stream, combinant tous ses éléments par l'opération associative `op`; `id` est l'élément neutre de `op`.

Par ailleurs, on se donne la classe `Appartement` :

```
public class Appartement {
    public int nbPieces;
    public int prix;
    public String lieu;
    ...
}
```

et on considère que la variable `List<Appartement> appartements` est définie, initialisée et visible dans les programmes des questions qui suivent.

1. Expliquez ce que calcule l'instruction suivante puis traduisez-la par une boucle `for` qui calcule la même valeur dans une variable. (3pts)

```
appartements.stream()
  .filter(a -> a.nbPieces >= 3)
  .map(a -> a.prix)
  .reduce(Integer.MAX_VALUE, (a, b) -> (a<b)?a:b)
```

2. Expliquez ce qu'affiche le programme ci-dessous, puis écrivez le bloc `for` comme un *pipeline* d'opérations d'agrégation utilisant des lambda-expressions. (3pts)

```
int count = 0;
for (Appartement a: appartements)
    if (a.lieu.equals("Paris 13e"))
        count++;
System.out.println(count);
```

3. Même question bour la boucle suivante : (3pts)

```
boolean trouve = false;
    for (Appartement a : appartements)
        if (a.lieu.equals("Paris 13e") && a.nbPieces >= 3 && a.prix <= 400000) {
            trouve = true;
            break;
        }
System.out.println(trouve);
```

4. Parmi les *pipelines* suivants, lesquels garantissent-ils la sortie "`52914`" ? (6pts)
   (a) `Arrays.asList(5,2,9,1,4).stream().parallel().forEachOrdered(System.out::print);`
   (b) `Arrays.asList(5,2,9,1,4).stream().forEach(System.out::print);`
   (c) `Stream.of(5,2,9,1,4).parallel().forEach(System.out::print);`
   (d) `Arrays.asList(5,2,9,1,4).parallelStream().sequential().forEach(System.out::print);`
   (e) `Stream.of(5,2,9,1,4).forEach(System.out::print);`
   (f) `Stream.of(5,2,9,1,4).parallel().sequential().forEach(System.out::print);`