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

Examen de Programmation

(janvier 2011)

Durée deux heures trente ; tous les documents sont interdits
Le barème est donné à titre indicatif (total sur 40 + 4pts hors barème).
3 pages

1. On considère un type de données *Buffer* défini par les opérations *mettre(o)* qui ajoute un objet *o* dans le *Buffer* et *prendre* qui retourne et retire un objet du *Buffer*. *mettre* et *prendre* suivent une règle "*FIFO (first in first out)*" : *prendre* retirera l'objet qui a été introduit par *mettre* le plus anciennement. Par exemple : partant d'un *Buffer* vide, après la séquence *mettre(a), mettre (b), mettre(c)*, la séquence *prendre, prendre, prendre* retournera successivement *a*, *b* et *c*.
   Essayer de *prendre* un élément dans le *Buffer* vide est une erreur qui correspondra en java à une exception. De même, il est possible que le *Buffer* soit plein, auquel cas l'opération *mettre* est aussi considérée comme une erreur correspondant aussi à une exception.

   (a) On supposera que les éléments composant le *Buffer* sont des `Object` java. Ecrire une interface `Buffer` correspondant au type de données *Buffer*. On ajoutera deux méthodes `estVide` et `estPlein` qui testent respectivement que le *Buffer* est vide ou plein. On définira aussi deux exceptions `BufferPlein` et `BufferVide`. (1pt)

   (b) Proposer une première implémentation qu'on appellera `File` de cette interface `Buffer` en utilisant une liste chaînée définie par une classe statique interne dont le contenu est : (3pts)

```
static class Cellule{
     public Object d;
     public Cellule suivant;
     Cellule(Object o, Cellule s){d=o;suivant=s;}
     Cellule(Object o){this(o,null);}
     Cellule(){this(null,null);}
 }
```

   L'exception `BufferPlein` a-t-elle un sens ici ? Est-il nécessaire de la déclarer dans l'en-tête de la méthode `mettre` ? Pourrait-on sans modifier l'interface `Buffer` définir une exception plus spécifique que l'exception `BufferVide` pour `File` pour une tentative de *prendre* dans une *File* vide ? Si oui comment pourrait-on procéder ? (2pts)

   (c) On veut maintenant implémenter l'interface `Buffer` à l'aide d'un tableau `Tab` de `Object` de taille fixe (la taille étant fixée par un constructeur de la classe). Proposer une implémentation pour cela (on appellera `BufferBorné` cette implémentation). On demande de réaliser une implémentation "efficace" pour laquelle `mettre` un objet ou `prendre` un objet ne déplace pas d'autres objets dans `Tab`. (4pts)

   (d) On va maintenant utiliser le *Buffer* avec des threads. Pour cela, on considère une thread, le producteur, qui produit des objets et les met au fur et à mesure dans le buffer et une deuxième thread, le consommateur, qui consomme ces objets, c'est-à-dire qui prend successivement les objets dans le buffer. Le problème est que le consommateur doit attendre pour consommer un objet qu'au moins un objet ait été produit, ait été mis dans le buffer et n'ait pas encore été consommé. De même, de façon à ne pas perdre de données produites, le producteur doit attendre que le buffer ne soit pas plein avant de mettre les objets produits.

i. Définir une classe `Producteur` (extension de la classe `Thread`), qui contient une variable d'instance de type `Buffer`, cette variable `buf` étant initialisée par le constructeur à un `Buffer` particulier. Cette thread produira la liste des nombres de 1 à 100 qu'elle écrira dans le `Buffer` référencé par `buf`. Si l'écriture dans le buffer échoue, provoquant l'exception `BufferPlein`, la thread recommencera la tentative d'écriture jusqu'à ce qu'elle soit possible (parce que le consommateur aura consommé une valeur libérant ainsi un emplacement dans le buffer). (2pts)

ii. Définir une classe `Consommateur` (extension de la classe `Thread`), qui contient une variable d'instance de type `Buffer`, cette variable `buf` étant initialisée par le constructeur à un `Buffer` particulier. Cette thread consommera 100 valeurs contenues dans le `Buffer` référencé par `buf`. Si la lecture dans le buffer échoue, provoquant l'exception `BufferVide`, la thread recommencera la tentative de lecture jusqu'à ce qu'elle soit possible (parce que le producteur a produit une nouvelle valeur). (2pts)

iii. Ecrire un programme principal `main` qui crée un `Buffer` référençant un `BufferBorné` de taille 2 et ensuite crée et lance une thread `Consommateur` et une thread `Producteur`. (2pts)

Si au lieu d'utiliser un `BufferBorné`, on utilise une `File`, cela change-t-il quelque chose ? (1pt)

iv. Si les opérations `mettre` et `prendre` exécutées par les threads sont interrompues parce que les threads qui les exécutent ont perdu l'accès au processeur du fait de l'exécution concurrente des threads, quels problèmes peuvent se poser ? Quelle construction du langage permet d'y remédier. (2pts)

v. L'inconvénient de la solution précédente est que le consommateur peut en cas d'échec de la consommation être amené à refaire d'autres tentatives même si le buffer est encore vide. On peut faire une remarque similaire pour le producteur. Proposer une autre solution (indication : on pourra utiliser `wait`, `notify`). (4pts hors barème)

2. On rappelle le contenu de l'interface `Iterator` :

```
Interface Iterator<E>{
    boolean hasNext();
    E next();
    void remove(); // remove ne servira pas ici
}
```

Cette interface sert à obtenir successivement (par `next()`) les éléments d'une structure de données contenant des objets de classe `E` jusqu'à ce que `hasNext` retourne faux.

(a) A partir d'un objet `Iterator<Integer>`, définir une méthode statique `moyenne` qui calcule la moyenne (retournée sous forme d'un `double`) de toutes les valeurs retournées par le `next` jusqu'à ce que `hasNext` retourne faux. (2pts)

(b) Définir une méthode qui retourne un `Iterator` à partir d'un tableau de `Integer` (le `next` de cet `Iterator` retournera successivement tous les éléments du tableau). Utiliser cette méthode et la méthode `moyenne` précédente pour obtenir la valeur moyenne de ce tableau. (3pts)

(c) Supposons que $f(n)$ est une fonction à un paramètre entier qui retourne un réel. Si F est un ensemble (avec multiplicité) d'entiers, on veut calculer la valeur moyenne $M(f, F)$ de cette fonction sur cet ensemble F. Par exemple, si $f(n) = \sqrt{n}$ et $F = \{1, 3, 8, 4, 3\}$, $M(f, F) = \frac{1}{5}(\sqrt{1} + \sqrt{3} + \sqrt{8} + \sqrt{4} + \sqrt{3})$.

Dans ce qui suit, pour une fonction f et un ensemble F, on va définir un `Iterator` `it` de façon à ce que $M(f, F)$ soit la valeur retournée par `moyenne(it)`.

i. Définir une classe (abstraite) `IterFonc` : (1) contenant une méthode abstraite `fonction` ayant comme argument un `int` et retournant un `double`, (2) contenant une variable d'instance `F` qui est un tableau de `int` initialisé par un constructeur de la classe et (3) contenant une méthode `it` qui retourne un `Iterator` tel que les valeurs retournées par des appels successifs de `next`

soient les valeurs de `fonction(i)` `i` parcourant les valeurs du tableau `F`. Par exemple pour l'exemple précédent les appels successifs de `next` retourneront $\sqrt{1}$, $\sqrt{3}$, $\sqrt{8}$, $\sqrt{4}$ et $\sqrt{3}$. (3pts)

ii. Soit $F = \{2, 4, 6, 8, 10, 12\}$, en créant une extension de la classe `IterFonc` dans laquelle `fonction` est définie comme étant la fonction f qui à n associe n^2, écrire un code qui, en utilisant `moyenne`, calcule la moyenne de f sur l'ensemble F. Faire la même chose en utilisant une classe anonyme. (3pts)

3. On considère la classe :

```
class Pile{
    int n=0;
    Object[] data;
    Pile(){
        data=new Object[100];
    }
    Object pop(){return data[--n];}
    void push(Object o){data[n++]=o;}
    boolean estVide(){return n<=0;}
}
```

Une pile doit vérifier la propriété :
(1) dans toute séquence "*p.push(A); W; p.pop();* " telle que W ne contient ni *pop* ni *push* appliqué à p, le *p.pop()* doit retourner A.

(a) Vérifier que si p est la seule variable référençant un objet `Pile` la propriété (1) est toujours assurée. Est-ce encore vrai si plusieurs variables référencent le *même* objet `Pile` ? (Justifier la réponse ou donner un contre-exemple.) (3pts)

(b) Si les méthodes pop et push sont déclarées comme `synchronized` la réponse à la question précédente change-t-elle ? (1pt)

(c) On propose d'ajouter à la classe `Pile` une méthode `copie` :

```
public Pile copie(){
        Pile p=new Pile();
        p.n=n;
        p.data=data;
        return p;
}
```

Soit : " `Pile p=new Pile(); Pile q=p.copie(); Integer i;`", si on s'interdit ensuite les affectations entre `p` et `q`, `p` et `q` vérifient-ils la propriété (1) (justifier la réponse ou donner un contre-exemple) ? Si ce n'est pas le cas, modifier la méthode `copie` de façon à ce que la propriété (1) soit assurée. (3pts)

(d) Comment assurer la propriété (1) en utilisant la méthode `clone()` ? (On modifiera en conséquence la classe `Pile`.) (3pts).