IFT1025 Programmation 2
Examen Intra, Hiver 2015

Professeur : Pascal Vincent

Mardi 17 février 2015, 15h30 - 17h30.

— Prénom :
— Nom :
— Code permanent :
— Programme d'études :

Directives pédagogiques : Seule documentation permise : deux feuilles recto-verso, format letter ($8\frac{1}{2}'' \times 11''$), comprenant votre résumé de cours. Aucun appareil électronique n'est permis (à l'exception d'une montre pour connaître l'heure). L'examen est sur 100 pts. Répondez directement sur la feuille de l'énoncé, avec des réponses brèves mais précises.

1 Questions diverses (20 pts)

1. (1 pt) Que vaut l'expression Java suivante : **5/2**

2. (1 pt) Que vaut l'expression Java suivante : **"1"+2+3**

3. (1 pt) Que vaut l'expression Java suivante : **"1"+2*3**

4. (1 pt) Que vaut l'expression Java suivante : **3==2+1 && (3>3 || true)**

5. (2 pts) Qu'affiche l'extrait de programme suivant :

```
int i;
double k;
for(i=1, k=2; i<5 && k<10; i=i+1, k=k*2)
      System.out.println(k);

System.out.println("i="+i);
```

6. (2 pts) Pourquoi ne devrait-on généralement pas utiliser == pour comparer deux objets en Java ?

7. (2 pts) À l'intérieure d'une méthode non-static, quel mot clé Java peut-on utiliser qui contient une référence à l'objet sur lequel cette méthode a été appelée ?

8. (2 pts) Soit B une sous-classe de A où toutes deux déclarent une méthode non static *f()* avec la même signature, mais une définition (implémentation) différente. Soit la déclaration et instanciation suivante : *A x = new B();* Si on appelle *x.f()* quelle implémentation sera exécutée, celle définie dans A ou dans B ?

9. (1 pt) Comment, à l'intérieur d'une méthode d'une sous-classe, peut-on **faire appel** à la définition d'une méthode `void f()` de sa super-classe plutôt qu'à sa redéfinition `void f()` ... dans la sous-classe ? Écrivez l'instruction :

10. (1 pt) Soit B une sous-classe de A. Comment lors de la construction d'un objet B, peut-on faire appel à un constructeur de `A` en lui passant un paramètre x. Écrivez l'instruction qui fait cet appel :

11. (2 pts) On peut écrire *C x = new Z();* si Z est quoi par rapport à C ? (nommez tous les cas de figure)

12. (2 pts) Soit la déclaration et instanciation suivante : *C x = new Z();* que l'on supposera valide. Soit *m*() une méthode définie dans a classe Z mais qui n'est pas définie/connue au niveau de C. Écrivez la ou les instructions permettant d'appeler la méthode *m()* sur *x*.

13. (1pt) Au sein d'une classe, dans quel cas ne peut-on pas accéder (lire leur valeur) aux attributs qui ont été hérités de sa classe parent ?

14. (1 pt) Quelles classes ne peut-on pas instancier ?

2 Passage de paramètres (18 pts)

Dans le programme suivant, écrivez **à côté** de chaque *System.out.println* du main, ce qui s'affichera à l'écran.

Barème : +2 pts par valeur correcte, -2pts par valeur incorrecte, 0 si abstention.

Rappel et indication : en Java, les tableaux, tout comme les objets sont manipulés et passés en paramètre via des références. Pour réussir parfaitement cet exercice, il vous est fortement recommandé de dessiner et suivre l'évolution de l'état de la mémoire (les variables et ce qu'elles contiennent) au fur et à mesure de l'exécution du programme.

```
public class Obj {
        public int y;
        public static int f1(int x)
        { return x+1; }
        public static void f2(int x)
        { x = x+1; }
        public static void f3(int[] t)
        { t[0] = t[0]+1; }
        public static void f4(int[] t)
        {
                int x = t[0];
                t = new int[5];
                t[0] = x+1;
        }
        public static void f5(Obj c)
        {
                c = new Obj();
                c.y = 30;
        }
        public static void f6(Obj c)
        { c.y = 40; }
        public static void main(String[] args)
        {
                int[] t = new int[5];
                int x = 2;
                f1(x);
                System.out.println(x);
                x = 1;
                System.out.println(f1(x));
                x = f1(x);
                System.out.println(x);
                x = 20;
                f2(x);
                System.out.println(x);
                t[0] = 5;
                f3(t);
                System.out.println(t[0]);
                t[0] = 3;
                f4(t);
                System.out.println(t[0]);
                Obj a = new Obj();
                a.y = 200;
                f5(a);
                System.out.println(a.y);
                f6(a);
                System.out.println(a.y);
                a.y = 4;
                Obj b = a;
                b.y = a.y*2;
                System.out.println(a.y);
        }
}
```

3 Programmation objet, héritage et polymorphisme : classes de matrice (62 pts)

Vous allez écrire un début de hiérarchie de classes pour pouvoir représenter plusieurs genres de matrices (objet mathématique).

- les méthodes que l'on demande d'écrire sont **non static**, sauf si il est explicitement dit qu'elles doivent être static.
- Pour les méthodes de ces classes, on utilisera la convention d'indices de lignes et de colonnes numérotés à partir de 1 pour spécifier la position d'un élément dans une matrice (la convention usuelle en mathématique) plutôt qu'une numérotation à partir de 0 (tel que c'est le cas pour les tableaux en Java). À vous de gérer la conversion en des indices appropriés pour accéder à un élément de tableau java si besoin.
- si un utilisateur appelle une méthode correspondant à une opération invalide, vous pouvez simplement afficher un message d'erreur (avec *println*) suivi de l'instruction *System.exit(1);* qui quittera le programme. (Ceci en attendant que nous ayons vu plus en détail la gestion des erreurs et des exceptions en Java).

Écrivez :

1. Une *interface* **Printable** qui déclare une méthode **print** sans paramètre et ne retournant rien.

2. Une *classe* ***abstraite*** **Matrix** qui se conforme à l'interface Printable et comportant les méthodes suivantes :

   (a) des méthodes abstraites **nrows** et **ncols** qui auront pour rôle de retourner le nombre de lignes et de colonnes de la matrice respectivement.

   (b) une méthode abstraite **get** recevra en paramètre deux entiers indiquant une position i,j de l'élément dont on voudra récupérer la valeur. Cette valeur sera retournée sous forme d'un *double*.

   (c) une méthode abstraite **set** qui recevra 3 paramètres : indiquant la position i,j (tout comme get) et en plus la valeur (*double*) qu'on veut donner à l'élément à cette position. Cette méthode ne retourne rien.

   (d) la méthode **print** (pas abstraite) qui affichera la matrice à l'écran. Cette méthode ne retourne rien.

3. Une sous-classe de Matrix nommée **FullMatrix** qui pourra représenter une matrice arbitraire (pleine). En interne, les valeurs des éléments de cette matrice seront stockés dans un attribut nommé *elems* qui sera un tableau à 2 dimension de *double*. Tous les attributs (propriétés) de cette classe devront être *private*. Cette classe devra implémenter :

   (a) un constructeur prenant en paramètre les dimensions de la matrice (nombre de lignes et de colonnes) et une valeur initiale avec laquelle la remplir

   (b) toutes les méthodes nécessaires pour que cette classe soit instantiable

   (c) une méthode **equals** pour vérifier si deux matrices sont identiques (quant à leurs dimensions et contenu).

4. Une sous-classe de Matrix nommée **DiagMatrix** qui permettra de représenter une matrice diagonale (seuls les éléments sur la diagonale seront possiblement non nuls). Pour cela, en interne, il n'est besoin que de stocker les éléments de la diagonale, qui seront donc stockés dans un tableau de *double* à 1 dimension. Tous les attributs (propriétés) de cette classe devront être *private*. Cette classe devra implémenter :

   (a) un constructeur prenant en paramètre les dimensions de la matrice (ces matrices diagonales ne sont pas nécessairement carrées) et une valeur initiale avec laquelle remplir sa diagonale.

   (b) toutes les méthodes nécessaires pour que cette classe soit instantiable. Si l'utilisateur essaye d'effectuer une opération invalide (comme accéder en dehors de la matrice ou bien donner une valeur non-nulle à un élément hors-diagonale), affichez simplement un message d'erreur.

   (c) une redéfinition de la méthode **print** qui affichera **"Matrice diagonale"** avant d'appeler l'implémentation générique de print pour afficher le contenu de la matrice.

5. Une classe **MatrixMath** comportant les **méthodes static** suivantes :

   (a) une méthode static **diagonal** qui recevra en paramètre une référence de type Matrix et qui retournera une référence de type Matrix. Cette méthode extraira la diagonale de la matrice reçue en paramètre. Elle devra en fait créer et retourner un objet de type DiagMatrix qui contiendra une copie de la diagonale de la matrice reçue en paramètre.

   (b) une méthode static **add** qui recevra en paramètre deux Matrix et retournera une référence à une nouvelle Matrix qui sera l'addition des deux matrices reçues en paramètre (affichez une erreur si leurs dimensions ne sont pas identiques).

   (c) une méthode static **print** qui recevra en paramètre un tableau de Matrix et affichera toutes les matrices de ce tableau.

6. Un programme nommé **EssaiMatrix** qui comportera une méthode **main** qui effectuera les opérations suivantes :

   (a) L'utilisateur appellera ce programme en passant les dimensions de matrice (nombre de ligne et de colonnes) sur la ligne de commande. Votre méthode main devra donc récupérer ces dimensions. Toutes les matrices créées dans main auront ces même dimensions.

   (b) Créez une matrice *m1* de type FullMatrix initialisée avec des valeurs 1.0.

   (c) Ajoutez 33 à son élément à la position (1,2).

   (d) Affichez *m1*.

   (e) Créez une matrice *m2* de type DiagMatrix dont la diagonale contiendra la valeur 2.0.

   (f) Créez une autre matrice *m3* de type DiagMatrix dont la diagonale contiendra la valeur 3.0.

   (g) À l'aide des méthodes précédemment écrites, calculez la matrice *m4* qui sera la somme de *m1* et *m2*.

   (h) Affichez la matrice *m4*.

   (i) À l'aide des méthodes précédemment définies, extrayez la diagonale de *m4* et comparez si elle est égale à *m3*. Puis affichez un message qui dira si oui ou non elles sont égales.

   (j) Mettez toutes les matrices ainsi créées dans un tableau et affichez le en appelant la méthode static **print** précédemment définie.