Cours de Programmation & Algorithmique II
1ère session 2014 — PARTIE 2

Consignes à lire impérativement !

L'examen est composé de **2 parties** et d'un total de **4 questions**. Chaque partie dure **2 heures**. Il vous est demandé de respecter les consignes suivantes.

- L'usage d'une calculatrice, d'un téléphone portable ou de notes de cours n'est **pas autorisé**, laissez-les dans votre sac. Pensez à éteindre votre téléphone portable !
- A l'issue de l'épreuve, rendez toutes les feuilles d'énoncé, les réponses ainsi que les feuilles de brouillon.
- Commencez par **écrire vos nom, prénom et section (math, info, ...) sur chaque feuille** que vous rendrez, y compris les feuilles de brouillon.
- Répondez à chacune des questions.
- Faites attention à la clarté et à l'organisation de vos réponses. Respectez les règles grammaticales et orthographiques.
- Veuillez utiliser pour vos réponses les cadres prévus à cet effet. Si une partie de votre réponse n'entre pas dans le cadre, utilisez le dos de la feuille ou une feuille supplémentaire et indiquez clairement à quelle question la réponse correspond.
- Vous devez terminer cette partie de l'examen avant de pouvoir sortir de la salle (pour aller à la toilette par exemple).

Question 1: Manipulation de Matrices (/7)

L'objectif de cette question est d'implémenter une classe Java appelée `Matrix` qui permette la manipulation de matrices. Bien que des tableaux à deux dimensions puissent être utilisés à cet effet, ils ont le désavantage qu'il s'agit de tableaux de tableaux et qu'il n'y a pas moyen de contraindre les tableaux "internes" à avoir tous la même taille. La classe `Matrix` contiendra un tableau interne à deux dimensions et devra se charger de contraindre les tableaux internes à avoir tous la même taille.

La Figure 1 donne un extrait de la classe `Matrix` à implémenter. Le tableau interne `content` est de type `double [][]`. Les tableaux internes de ce tableau représentent les lignes de la matrice. Ainsi, l'index de tableau le plus à gauche désigne la ligne et celui le plus à droite désigne la colonne. Les variables d'instance N et M de la classe doivent contenir respectivement le nombre de lignes et de colonnes de la matrice. La dimension de la matrice ne peut être changée.

- **Constructeurs :** La classe `Matrix` supporte 3 constructeurs. Le premier crée une matrice de N lignes et M colonnes dont les éléments valent initialement 0. Le second crée une matrice carrée de dimension $N \times N$. Le troisième permet de créer une matrice à partir d'un tableau de tableaux. Ce dernier constructeur doit vérifier que toutes les lignes de la matrice ont la même taille. Si ça n'est pas le cas, une exception doit être générée. De plus, ce constructeur doit effectuer une copie du tableau de tableaux passé en argument.
- **Multiplication :** La méthode `multiply` effectue la multiplication de cette matrice (A) par une autre matrice `B` dont la référence est passée en paramètre. Le produit des deux matrices est une nouvelle matrice $P = AB$. La méthode `multiply` doit vérifier que les dimensions des matrices A et B permettent la multiplication. Si ça n'est pas le cas, une exception doit être générée.

- **Représentation sous forme ce chaîne de caractères :** La méthode `toString` doit retourner une instance de `String` qui contient une représentation de la matrice. Cette représentation doit être telle que les valeurs d'une même colonne soient alignées sur leur virgule, c.-à-d. que la distance en caractères de la virgule au début de la ligne doit être identique pour tous les éléments d'une même colonne. La Figure 2 donne une matrice d'exemple A et illustre la chaîne de caractères produite par `toString`. On peut constater notamment qu'à la première ligne, il y a 2 caractères espace entre les valeurs $A_{1,1}$ et $A_{1,2}$. Cet espace est composé de l'espace inter-colonnes auquel est ajouté un espace supplémentaire afin de garantir l'alignement avec $A_{2,2}$.

```
public class Matrix
{

  public final int N, M;

  private double [][] content;

  public Matrix(int N, int M)
  {
    /* à implémenter */
  }

  public Matrix(int N)
  {
    /* à implémenter */
  }

  public Matrix(double [][] m) throws Exception
  {
    /* à implémenter */
  }

  public Matrix multiply(Matrix B) throws Exception
  {
    /* à implémenter */
  }

  public String toString()
  {
    /* à implémenter */
  }

}
```

FIGURE 1 – Extrait de la classe `Matrix`.

$$A = \begin{pmatrix} 20 & 1 & 372.33 & -2 \\ 3 & -1 & 1 & 7 \\ -1 & 3 & 4 & -5 \end{pmatrix}$$

```
20.0   1.0  372.3  -2.0
 3.0  -1.0    1.0   7.0
-1.0   3.0    4.0  -5.0
```

FIGURE 2 – Matrice exemple A et chaîne de caractères utilisée pour la représenter.

Il vous est demandé de : Compléter l'implémentation de la classe `Matrix` de la Figure 1. Vous devez pour cela fournir

Q1a Une implémentation de la méthode `multiply`.

Q1b Une implémentation des 3 constructeurs. Une partie de ces constructeurs sera probablement commune. Si c'est le cas, veillez à ne pas l'écrire plusieurs fois.

Q1c Une implémentation de la méthode `toString`.

Notes :

- Il est suggéré d'utiliser la méthode `System.arraycopy` afin de copier des tableaux. Attention, cette méthode n'est pas utilisable directement pour effectuer la copie d'un tableau de tableaux.
- Afin d'aligner horizontalement les valeurs de la matrice dans la méthode `toString`, il est nécessaire de déterminer le nombre de caractères que chaque valeur occupe. Ceci peut être réalisé en s'aidant de la méthode `format(String fmt, Object... args)` de la classe `String`. Cette méthode peut être invoquée comme suit pour convertir la variable `x` de type `double` en chaîne de caractères, avec 1 chiffre pour la partie fractionnelle du nombre : `String.format("%.1f", x)`. Si par exemple `x` vaut 372.33, alors la chaîne résultante sera `"372.3"`.

Q1a **(implémentation de la méthode `multiply` de la classe `Matrix`)**

Q1b **(implémentation des constructeurs de la classe `Matrix`)**

Q1c **(implémentation de la méthode `toString` de la classe `Matrix`)**

Question 2: Casse-tête (/3)

Cette question s'intéresse au *binding* de variables et de méthodes et mixant leur utilisation dans un enchevêtrement de classes. Vous disposez ci-dessous de 3 classes `A`, `B` et `C` qui contiennent un certain nombre de variables et de méthodes. Vous disposez également d'un court programme utilisant ces classes.

```
public class A {
  public static int x= 5;
  public int y= 20;
  public int getX() {
    return x;
  }
}

public class B extends A {
  public int y= 25;
  public B() {
    super();
  }
  public int getX(int x) {
    return x + 50;
  }
  public void setY(int y) {
    this.y+= y + 1.75;
  }
}

public class C extends B {
  public static int x= 10;
  public int getX() {
    return x + ((A) this).x;
  }
  public void setY(int y) {
    try {
      ((A) this).y= y;
    } finally {
      ((C) this).y= y;
    }
  }
}
```

```
A x= new A();
A y= new B();
A z= new C();
System.out.println(z.x);                    /* Q2a */
System.out.println(((C) z).x);              /* Q2b */
System.out.println(y.getX());               /* Q2c */
System.out.println(z.getX());               /* Q2d */
((B) y).setY(200);
((B) z).setY(300);
System.out.println(y.y);                    /* Q2e */
System.out.println(((B) y).y);              /* Q2f */
System.out.println(z.y);                    /* Q2g */
System.out.println(((C) z).y);              /* Q2h */
```

Ce qui vous est demandé : Il vous est demandé de fournir les 8 valeurs affichées par le programme suivant. Pour chacune des valeurs veuillez donner une brève explication ; par exemple quel(s) binding(s) sont réalisés. Une réponse sans justification sera considérée comme incorrecte.

Q2a **(valeur et justification)**

. . . . . .

Q2b **(valeur et justification)**

. . . . . .

Q2c **(valeur et justification)**

. . . . . .

Q2d **(valeur et justification)**

. . . . . .

Q2e **(valeur et justification)**

. . . . . .

Q2f **(valeur et justification)**

. . . . . .

Q2g **(valeur et justification)**

. . . . . .

Q2h **(valeur et justification)**

. . . . . .