Examen du cours de Programmation et Algorithmique II

1ère Session, Juin 2022

Partie 1

NOM : ..... PRENOM : ..... SECTION : .....

Consignes à lire impérativement !

L'examen est composé de **2 parties**. Chaque partie dure **2 heures**. Il vous est demandé de respecter les consignes suivantes.

- Commencez par écrire vos **nom**, **prénom** et **section** (math, info, ...) sur chaque feuille, y compris les feuilles de brouillon.
- Laissez vos calculatrice, téléphone portable et notes de cours dans votre sac. Leur usage n'est **pas autorisé**. Pensez à éteindre votre téléphone portable !
- Faites attention à la clarté et à l'organisation de vos réponses. Respectez les règles grammaticales et orthographiques.
- Utilisez pour vos réponses les **cadres** prévus à cet effet. Si davantage d'espace est nécessaire, utilisez le dos de la feuille ou une feuille supplémentaire et indiquez clairement où se situe le restant de la réponse.
- Vous devez terminer cette partie de l'examen avant de pouvoir sortir de la salle (pour aller à la toilette par exemple).
- Toutes les feuilles (énoncé et brouillon) doivent être remises en fin d'examen.
- Vérifiez que vous avez répondu à toutes les questions (il y a **2 questions** dans cette partie).

Question 1 – Echauffement (/4)

Q1a – Quel sera le résultat affiché par le programme ci-dessous ? (cochez la case correspondante)

- ☐ Le programme affiche « a ».
- ☐ Le programme affiche « b ».
- ☐ Le programme n'affiche rien.
- ☐ Aucune des solutions ci-dessus.

```
int x = 3;
if (x >= 0)
    if (x < 3)
        System.out.println("a");
else
    System.out.println("b");
```

Examen du cours de Programmation et Algorithmique II

1ère Session, Juin 2022

Partie 1

NOM : ..... PRENOM : ..... SECTION : .....

Q1b – Quel sera le résultat affiché par le programme ci-dessous ? (cochez la case correspondante)

- ☐ Le programme affiche « `true` ».
- ☐ Le programme affiche « `false` ».
- ☐ Le programme ne compile pas.
- ☐ Le programme génère une exception.
- ☐ Aucune des solutions ci-dessus.

```
Number [] numbers = new Number[2];
numbers[0] = Double.valueOf(3.1415);
numbers[1] = Double.valueOf(3.1415);
System.out.println(numbers[0] == numbers[1]);
```

Q1c – Quel sera le résultat affiché par le programme ci-dessous ?

.....

```
int n = 123;
while (n > 0) {
    int r = n % 2;
    n = n / 2;
    System.out.print((char) ('0' + r));
}
```

Q1d – Donnez une instruction permettant d'appeler la méthode `plop` de la classe `A` ?

.....

```
class A {
    public void plop() {
        System.out.println("A.plop");
    }
}
class B extends A {
    @Override
    public void plop() {
        System.out.println("B.plop");
    }
}
A ref = new B();
```

Examen du cours de Programmation et Algorithmique II

1ère Session, Juin 2022

Partie 1

NOM : ..... PRENOM : ..... SECTION : .....

Q1e – Dans quel ordre les clés de l'association suivante sont-elles affichées ? (cochez la case correspondante)

- ☐ Hervé, Henri, Henriette
- ☐ Henri, Henriette, Hervé
- ☐ Henriette, Hervé, Henri
- ☐ Aucun des ordres ci-dessus.

```
Map<String,Integer> m = new TreeMap<String,Integer>();
m.put("Hervé", 42);
m.put("Henri", 43);
m.put("Henriette", 41);
for (String s: m.keySet())
    System.out.println(s);
```

Q1f – Quel modificateur devez-vous ajouter à la définition de la classe `FonctionSinus` ci-dessous ?

.....

```
public interface Fonction {
        double eval(double x, double y);
}
public /* ... */ class FonctionSinus implements Fonction{
        double eval(double x) {
                return Math.sin(x);
        }
}
```

Q1g – Si `E` est un paramètre de type, parmi celles listées ci-dessous, quelles expressions seront acceptées par le compilateur et mèneront à une exécution correcte lors de leur utilisation ? (cochez les cases correspondant aux expressions correctes)

- ☐ `List<Object> l = new List<>();`
- ☐ `List<Object> lo = new ArrayList<String>();`
- ☐ `E[] tab = new E[100];`
- ☐ `E[] tab = (E[]) new Object[100];`
- ☐ `List<E> [] = (List<E>) new Object[100];`
- ☐ `List<E> [] = (List<E>) new List<Object>[100];`

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2022
Partie 1

NOM : ..... PRENOM : ..... SECTION : .....

Q1h – Le code suivant permet de rechercher un élément dans une liste chaînée triée par ordre croissant. Cette liste n'admet pas d'élément `item` égal à `null`. Complétez les parties manquantes du code.

- ligne 1 : .....
- ligne 8 : .....
- ligne 9 : .....
- ligne 10 : .....
- ligne 12 : .....

```
1  class Node</* ??? */> {
2      E     item;
3      Node next;
4  }
5  Node head = /* ... une liste chaînée triée ... */
6  Object toFind = /* ... un object à rechercher ... */
7  Node tmp = head;
8  while (/* ... */) {
9      int cmp = /* ... */
10     if (/* .. */)
11         break;
12     if (/* ... */) {
13         System.out.println("L'objec␣appartient␣à␣la␣liste");
14         break;
15     }
16     tmp = tmp.next;
17 }
```

Question 2 – Test de la propriété « bien parenthésée » (/6)

L'objectif de cette question est de concevoir et implémenter en Java un algorithme capable de vérifier si une chaîne de caractères est « bien parenthésée ». Une telle expression peut utiliser plusieurs types de parenthèses. Dans cet énoncé, nous considérons les types de parenthèses suivants : `()`, $\{\}$ et `[]`.

Une chaîne bien parenthésée peut être définie par récurrence comme suit :

- Une chaîne ne comprenant que des caractères autres que des parenthèses est bien parenthésée.
- Si x et y sont des chaînes bien parenthésées, alors leur concaténation xy est une chaîne bien parenthésée.
- Si x est une chaîne bien parenthésée, alors `(`x`)`, $\{x\}$ et `[`x`]` sont des chaînes bien parenthésées.

Examen du cours de Programmation et Algorithmique II

1ère Session, Juin 2022

Partie 1

NOM : ..... PRENOM : ..... SECTION : .....

Le tableau suivant donne des exemples de chaînes et indique lesquelles sont bien parenthésées ou non. Dans le cas négatif, une description du problème est fournie et le premier caractère posant problème est souligné et mis en gras.

Chaîne	**Description**
`(5+(3))`	bien parenthésée
`()`	bien parenthésée
`[(45*tadaam)+(9-2)]`	bien parenthésée
`()a+5{3}`	bien parenthésée
`(`**`]`**	parenthèses ouvrante (et fermante } de types différents
`5+2`**`}`**	parenthèse fermante } sans parenthèse ouvrante correspondant
`(((Ab)+cd`**`}`**`]`	ferme une parenthèse ouvrante d'un autre type : (versus }

Il vous est demandé de fournir l'implémentation en Java de la méthode `isWellParenthesized` qui vérifie si une chaîne de caractères est bien parenthésée. La signature de la méthode est fournie ci-dessous. La méthode prend en argument la chaîne s à vérifier. Elle retourne `true` si s est bien parenthésée, `false` sinon.

```
boolean isWellParenthesized(String s)
```

Veuillez respecter les contraintes données ci-dessous.

- L'algorithme fourni doit fonctionner de manière **itérative**. **Une implémentation récursive sera considérée incorrecte !** Il est nécessaire de garder trace dans une structure de données des parenthèses ouvrantes non encore fermées. L'ordre dans lequel les parenthèses sont ajoutées et retirées dicte le choix de la structure de données à utiliser.
- Afin de garder un code court et lisible, il est **vivement conseillé** de traiter les différents types de parenthèses dans un même cas. A cet effet, la méthode `String.indexOf(char c)` peut être utilisée. Cette méthode retourne -1 si la chaîne ne contient pas le caractère c. La méthode retourne la position de la première occurrence de c sinon.
- Votre implémentation doit ignorer (passer) les caractères autres que les parenthèses (), {} et [] tels que p.ex. des opérateurs, chiffres, lettres.

Q2 **(méthode `isWellParenthesized`)**

NOM : . . . . . PRENOM : . . . . . SECTION : . . . . . .

Q2 (suite) **(méthode `isWellParenthesized`)**

Examen du cours de Programmation et Algorithmique II

1ère Session, Juin 2022

Partie 2

NOM : ..... PRENOM : ..... SECTION : .....

Consignes à lire impérativement !

L'examen est composé de **2 parties**. Chaque partie dure **2 heures**. Il vous est demandé de respecter les consignes suivantes.

- Commencez par écrire vos **nom**, **prénom** et **section** (math, info, ...) sur chaque feuille, y compris les feuilles de brouillon.
- Laissez vos calculatrice, téléphone portable et notes de cours dans votre sac. Leur usage n'est **pas autorisé**. Pensez à éteindre votre téléphone portable !
- Faites attention à la clarté et à l'organisation de vos réponses. Respectez les règles grammaticales et orthographiques.
- Utilisez pour vos réponses les **cadres** prévus à cet effet. Si davantage d'espace est nécessaire, utilisez le dos de la feuille ou une feuille supplémentaire et indiquez clairement où se situe le restant de la réponse.
- Vous devez terminer cette partie de l'examen avant de pouvoir sortir de la salle (pour aller à la toilette par exemple).
- Toutes les feuilles (énoncé et brouillon) doivent être remises en fin d'examen.
- Vérifiez que vous avez répondu à toutes les questions (il y a **2 questions** dans cette partie).

Question 1 – Modélisation de matrices (/4)

Cette question concerne la modélisation de matrices $m \times n$ à l'aide d'une classe Java **immuable** nommée `Matrix`. Les dimensions m (lignes) et n (colonnes) ainsi que les composantes de la matrice doivent être fournies au constructeur. Il est demandé d'utiliser à cet effet un constructeur ayant un **nombre variable d'arguments**. Un exemple de création d'une instance de matrice 2×3 est illustré à la Figure 1. La matrice ainsi créée est montrée à la droite de la figure.

```
Matrix a= new Matrix(2, 3,
                     1, 2, 3,
                     4, 5, 6);
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

FIGURE 1 – Exemple d'instanciation de la classe `Matrix`.

La classe `Matrix` doit également fournir une méthode `multiply` qui calcule le produit de la matrice avec une autre (multiplication à droite) et une méthode `transpose` qui calcule la transposée de la matrice. Si les arguments fournis à un constructeur ou une méthode sont inconsistants, une exception **non-contrôlée** sera générée.

Ce qui vous est demandé : fournir la classe `Matrix` correspondant à la description ci-dessus. En particulier, fournir ses déclarations de variables, son ou ses constructeurs et les méthodes `multiply` et `transpose`

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2022
Partie 2

NOM : . . . . . PRENOM : . . . . . SECTION : . . . . . .

Q1 **(classe `Matrix`)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2022
Partie 2

NOM : . . . . . PRENOM : . . . . . SECTION : . . . . . .

Q1 **(classe `Matrix` — suite)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2022
Partie 2

NOM : ..... PRENOM : ..... SECTION : .....

Question 2 – Fusion de listes doublement chaînées (/6)

La classe `DLList` dont un extrait est donné ci-dessous permet de manipuler des listes doublement chaînées. A cet effet, elle définit la classe interne `DLNode` qui représente un noeud de la liste, muni des références de son prédécesseur (`prev`) et de son successeur (`next`) ainsi que de l'élément/donnée associé (`item`). Elle définit également deux variables `head` et `tail` qui référencent respectivement le début et la fin de la liste.

Les méthodes fournies par la classe `DLList` sont les suivantes :

- La méthode `add` permet d'ajouter un élément en fin de liste.
- La méthode `dumpForward` affiche à la console chacun des éléments de la liste en commençant par la tête.
- La méthode `merge` effectue la fusion de la liste avec une autre liste. Les deux listes doivent être préalablement triées (pré-condition). Le résultat est une nouvelle liste (avec de nouveaux noeuds) elle aussi triée. Les listes passées en argument ne sont pas modifiées par `merge`.

```
public class DLList<E extends Comparable<E>> {

  private class DLNode {
    public DLNode prev, next;
    public E item;
    public DLNode(DLNode prev, E item, DLNode next) {
        this.prev = prev;
        this.next = next;
        this.item = item;
    }
  }

  private DLNode head, tail;

  public void add(E item) { /* à implémenter ( Q2a ) */ }

  public void dumpForward() { /* à implémenter ( Q2b ) */ }

  public DLList<E> merge(DLList<E> other) { /* à implémenter ( Q2c ) */ }

}
```

Ce qui vous est demandé :

Q2a Implémentez la méthode `add` de la classe `DLList`

Q2b Implémentez la méthode `dumpForward` de la classe `DLList`

Q2c Implémentez la méthode `merge` de la classe `DLList`

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2022
Partie 2

NOM : . . . . . PRENOM : . . . . . SECTION : . . . . . .

Q2a **(méthode `add`)**

Q2b **(méthode `dumpForward`)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2022
Partie 2

NOM : . . . . . PRENOM : . . . . . SECTION : . . . . . .

Q2c **(méthode `merge`)**