Examen du cours de Programmation et Algorithmique II

1ère Session, Juin 2018

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 – Evaluateur d'expression RPN (/5)

On utilise généralement la notation *infixe* pour noter des expressions arithmétiques. Dans cette notation, les opérateurs apparaissent **entre** les termes sur lesquels ils portent. Un exemple d'une telle expression est `(13 + 2) / 2.5`. Des parenthèses sont nécessaires pour grouper des sous-expressions, ce qui rend plus complexe leur analyse par ordinateur.

La notation *postfixe*, aussi appelée *notation polonaise inverse* ou *reverse polish notation* (RPN), est plus facile à analyser. Dans cette notation, les opérateurs apparaissent **après** les termes sur lesquels ils portent. Les parenthèses ne sont plus nécessaires. L'exemple ci-dessus pourrait s'écrire en notation *postfixe* comme `13 2 + 2.5 /`.

Un algorithme évaluant une expression *postfixe* peut procèder de gauche à droite et traiter les nombres ou opérateurs rencontrés les uns à la suite des autres. Lorsqu'un nombre est rencontré, il est mis de côté sur une pile. Lorsqu'un opérateur est rencontré, il est immédiatement appliqué sur les deux derniers nombres au sommet de la pile et le résultat obtenu remplace ceux-ci. Cette approche est illustrée ci-dessous pour deux expressions différentes.

Expression	**Pile**
`13 2 + 2.5 /`	
`2 + 2.5 /`	13
`+ 2.5 /`	13, 2
`2.5 /`	15
`/`	15, 2.5
	6

Expression	**Pile**
`2.5 13 2 + /`	
`13 2 + /`	2.5
`2 + /`	2.5, 13
`+ /`	2.5, 13, 2
`/`	2.5, 15
	0.1666...

Examen du cours de Programmation et Algorithmique II

1ère Session, Juin 2018

Partie 1

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

Ce qui vous est demandé : Ecrire en Java l'implémentation d'un évaluateur d'expression *postfixe* (RPN). L'expression est fournie sous la forme d'une chaîne de caractères. Les nombres et opérateurs qui y apparaissent sont séparés par un ou plusieurs espaces.

L'évaluateur d'expression doit prendre la forme d'une classe `RPNParser`. L'expression *postfixe* est fournie au constructeur de la classe. Le travail d'évaluation de l'expression est réalisé par les deux méthodes suivantes dont vous devez fournir l'implémentation.

Q1a La méthode `nextToken` extraît de la chaîne de caractères le nombre ou opérateur suivant. Si la fin de la chaîne de caractères est atteinte, la méthode retourne `null`. Par exemple, si la chaîne est `"12 5 +"`, les appels successifs de `nextToken` retourneront `"12"`, `"5"`, `"+"` et `null`.

Q1b La méthode `eval` analyse l'expression et retourne sa valeur. La méthode `eval` utilise la méthode `nextToken` pour obtenir les nombres et opérateurs de l'expression. Si l'expression est mal formatée, la méthode génère une exception `RPNException`. Des exemples d'expressions mal formatées sont `"12 5"` et `"5 +"`.

Notes : Le tableau ci-dessous résume des classes et méthodes offertes par la bibliothèque Java qui pourraient être utiles à votre implémentation.

String	
`char charAt(int` i`)`	retourne le caractère à l'index i.
`String substring(int` bi`, int` ei`)`	retourne la sous-chaîne commençant à l'index bi et se terminant à l'index $ei - 1$.
Double	
`static double parseDouble(String` s`)`	retourne le `double` représenté par la chaîne s. Si la chaîne ne représente pas un `double`, une expression `NumberFormatException` (hors-contrôle) est générée.
Stack<E>	
`void push(E` e`)`	Ajoute l'élément e au sommet de la pile.
`E pop()`	Retire l'élément au sommet de la pile et le retourne.
`int size()`	Retourne le nombre d'éléments sur la pile.

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2018
Partie 1

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

Q1a **(méthode `nextToken`)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2018
Partie 1

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

Q1b **(méthode `eval`)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2018
Partie 1

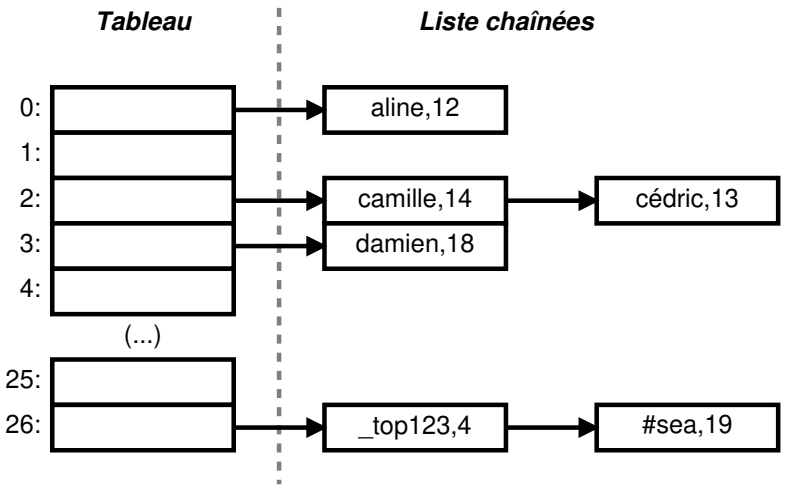
NOM : ..... PRENOM : ..... SECTION : .....

Question 2 – Table associative (/5)

Une table associative permet de conserver des associations (k, v) entre une clé k et une valeur v. Il ne peut exister simultanément dans la table deux couples de même clé.

Une table de hachage est une structure de données qui implémente une table associative à l'aide d'un tableau. La cellule du tableau dans laquelle un couple (k, v) est conservé est déterminée par une fonction de hachage h appliquée à la clé. La fonction de hachage est généralement non-injective, ce qui implique que plusieurs couples clé/valeur peuvent devoir être conservés dans la même cellule. Ceci est typiquement réalisé à l'aide d'une liste chaînée par cellule.

Un exemple d'une telle table de hachage est illustré ci-dessous. Le tableau contient 27 cellules. Les clés sont des chaînes de caractères et les valeurs des entiers. La fonction de hachage détermine la cellule du tableau sur base du premier caractère de la clé. Les 26 premières cellules du tableau correspondent aux 26 lettres de l'alphabet latin, tandis que la 27ème cellule correspond à tout autre caractère.



On considère la classe `HashMap<V>` qui maintient des associations entre clés de type `String` et valeurs de type `V` à l'aide d'une table de hachage. Un tableau interne à la classe, nommé `tab` référence les têtes de listes chaînées d'associations. Ces listes sont maintenues triées selon les clés, en utilisant l'ordre lexicographique sur les chaînes de caractères. Le tableau `tab` peut être de taille quelconque (mais non nulle). Les listes chaînées reposent sur la classe interne `Node` pour représenter chaque maillon. `Node` contient la référence de la cellule suivante (`next`), une clé (`key`) de type `String` et une valeur (`value`) dont le type est donné par le paramètre `V`.

Ce qui vous est demandé : Une implémentation partielle de `HashMap` est fournie ci-après. Vous devez en implémenter les éléments suivants :

Q2a La méthode `put(String k, V v)` ajoute une association (k, v) à la structure de données. Elle utilise à cet effet la méthode abstraite `hash` pour déterminer la cellule du tableau. Si une association de même clé que k existe, elle est remplacée par la nouvelle.

Q2b La classe interne `HashMapIterator` implémente un itérateur pour `HashMap`.

Q2c La méthode `iterator` retourne une instance d'itérateur.

Examen du cours de Programmation et Algorithmique II

1ère Session, Juin 2018

Partie 1

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

```
public abstract class HashMap<V> implements Iterable<V>
{

        /* Definition du maillon d'une liste chainee */
        private class Node {
                Node next;
                String key;
                V value;
                public Node(String key, V value, Node next) {
                        this.key = key;
                        this.value = value;
                        this.next = next;
                }
        }

        /* Tableau de listes chainees */
        private Node [] tab;

        /* Fonction de hachage (abstraite) */
        private abstract int hash(String key);

        /* Ajoute un couple (key, value) */
        public void put(String key, V value) {
                /* ... a implementer ...                                    ( Q2a) */
        }

        /* Itérateur */
        private class HashMapIterator
        implements Iterator<V> {

                /* ... a implementer ...                                    ( Q2b) */

        }

        /* Retourne un itérateur */
        public Iterator<V> iterator {
                /* ... a implementer ...                                    ( Q2c) */
        }

}
```

Notes : Le tableau ci-dessous résume les définitions des interfaces `Iterable` et `Iterator`.

Iterator<T>	
`boolean hasNext()`	Indique s'il reste des éléments à retourner.
`T next()`	Retourne l'élément suivant.
Iterable<T>	
`Iterator<T> iterator()`	Retourne un itérateur.

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2018
Partie 1

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

Q2a **(méthode `put`)**

Q2c **(méthode `iterator`)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2018
Partie 1

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

Q2b **(classe `HashMapIterator`)**

Examen du cours de Programmation et Algorithmique II

1ère Session, Juin 2018

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 **3 questions** dans cette partie).

Question 1 – Comptage de cellules (/4)

Soit un tableau de `boolean` partitionné en deux parties. Celle de gauche ne contient que des cellules de valeur `false` tandis que celle de droite ne contient que des cellules de valeur `true`. Un exemple d'un tel tableau est donné ci-dessous où la partie de gauche contient 5 cellules `false` et celle de droite 2 cellules `true`.

`false`	`false`	`false`	`false`	`false`	`true`	`true`

L'objectif de cette question est de concevoir un algorithme qui compte le nombre de cellules `true`. Comme cet algorithme est destiné à être appliqué sur des tableaux de très grande taille (p.ex. 10^9 éléments), il doit être conçu de manière à **minimiser le nombre d'itérations** dans le pire des cas.

Ce qui vous est demandé : Ecrire une méthode `countTrue` qui compte le nombre de cellules `true` d'un tableau. La signature de la méthode est donnée ci-dessous. Elle prend comme seul argument `tab`, le tableau sur lequel l'algorithme s'applique et elle retourne le nombre de cellules `true`. Pour le tableau donné en exemple ci-dessus, la méthode retourne 2.

```
int countTrue(boolean [] tab);
```

ATTENTION ! Un algorithme dont le nombre d'itérations dans le pire des cas varie linéairement avec la taille du tableau ne sera pas considéré comme suffisant pour réussir cette question !

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2018
Partie 2

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

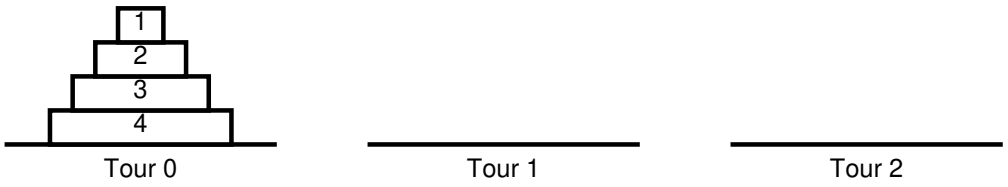
Q1 **(méthode `countTrue`)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2018
Partie 2

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

Question 2 – Tours de Hanoï (/4)

L'objectif de cette question est de fournir un algorithme qui résoud le problème dit des « Tours de Hanoï ». Il s'agit de déplacer n palets circulaires empilés les uns sur les autres de façon à former une tour. Chaque palet i a un diamètre entier d_i pris dans l'ensemble $\{1, \ldots, n\}$ de sorte que deux palets n'ont pas le même diamètre. Les palets sont empilés en ordre décroissant de diamètre, le plus petit palet étant situé au sommet de la tour. Cette situation est illustrée ci-dessous pour le cas où $n = 4$ (tour de gauche).



Le problème consiste à déplacer les palets de la tour de gauche (0) à la tour de droite (2) en utilisant éventuellement temporairement la tour centrale (1). Les contraintes suivantes doivent être respectées :

1. Un palet A ne peut être posé sur un palet B que si $d_A < d_B$.
2. Il n'est pas possible de déplacer plus d'un palet à la fois.

Ce qui vous est demandé : Fournissez une méthode de classe `solveHanoi(int n)` qui effectue la résolution du déplacement de n palets de la tour 0 à la tour 2. La méthode utilise un tableau de 3 piles représentant chacune l'état d'une tour. La méthode se charge d'initialiser ce tableau de sorte que l'ensemble des palets soit placé sur la pile de la tour 0, en commençant par le plus large. La méthode procède ensuite à la résolution. A chaque déplacement d'un palet i d'une tour j à une tour k, la méthode doit afficher à la console un message de la forme « $i : j \rightarrow k$ ».

Notes : il peut être opportun de découper le problème, notamment en utilisant une méthode supplémentaire pour la résolution. Par ailleurs, la class `Stack` de la bibliothèque Java peut être utilisée pour implémenter une pile. Ses méthodes importantes sont résumées ci-dessous.

Stack<E>	
`void push(E e)`	Ajoute l'élément e au sommet de la pile.
`E pop()`	Retire l'élément au sommet de la pile et le retourne.
`int size()`	Retourne le nombre d'éléments sur la pile.

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2018
Partie 2

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

Q2 **(méthode de classe `hanoiSolve`)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2018
Partie 2

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

Q2 **(méthode de classe `hanoiSolve` – *suite*)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2018
Partie 2

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

Question 3 – Now, scratch your head ! (/2)

Le programme ci-dessous est composé de plusieurs classes et interfaces dont le code source vous est fourni. Le programme principal en bas de page affiche 8 résultats à la console. Vos réponses sont les résultats affichés par le programme.

```
public interface I {
        int compute(int x);
}
```

```
public class B extends A {

        public static int y = 20;

        public int compute(int x) {
                return 2 * x;
        }

}
```

```
public abstract class A implements I {

        public int x = 2;
        public static int y = 10;

        public void setX(int value) {
                this.x = value;
        }
        public int getX() {
                return x;
        }
        public void work() {
                setX(compute(getX()));
                y = compute(y);
        }

}
```

```
public class C extends B {

        int x = 5;

        public void setX(int x) {
                this.x = x;
                y = x;
        }
        public int compute(int x) {
                try {
                        return x * x;
                } finally {
                        return x / 2;
                }
        }

}
```

```
A b = new B();
C c = new C();

b.work();

System.out.println(b.x);                                    /* ( Q3a) */
System.out.println(c.x);                                    /* ( Q3b) */
System.out.println(b.y);                                    /* ( Q3c) */
System.out.println(c.y);                                    /* ( Q3d) */

c.work();

System.out.println(b.x);                                    /* ( Q3e) */
System.out.println(c.x);                                    /* ( Q3f) */
System.out.println(b.y);                                    /* ( Q3g) */
System.out.println(c.y);                                    /* ( Q3h) */
```

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2018
Partie 2

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

Q3a

. . . . . .

. . . . . .

Q3b

. . . . . .

. . . . . .

Q3c

. . . . . .

. . . . . .

Q3d

. . . . . .

. . . . . .

Q3e

. . . . . .

. . . . . .

Q3f

. . . . . .

. . . . . .

Q3g

. . . . . .

. . . . . .

Q3h

. . . . . .

. . . . . .

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2018
Partie 2

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