Examen du cours de Programmation et Algorithmique II

1ère Session, Mai 2016

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 (/3)

Ecrivez une méthode de classe `checkIdentifier` qui vérifie si une chaîne de caractères est un identifiant valide dans un langage de programmation (identifiant = nom de variable/méthode/classe/...). La méthode ne vérifie pas si la chaîne est un mot réservé du langage (tel que `if`, `while`, `static`, ...). La méthode prend une chaîne de caractères en argument et ne retourne pas de résultat. Elle génère une exception sous-contrôle `InvalidIdentifierException` si la chaîne n'est pas un identifiant valide. Le constructeur de la classe `InvalidIdentifierException` prend une chaîne de caractères `message` en argument. Veillez à fournir un message d'erreur approprié.

Veillez à fournir une implémentation aussi **claire** et **concise** que possible. L'évaluation de votre réponse prendra ces deux critères en compte. Vous êtes autorisés à utiliser des méthodes auxiliaires dans votre implémentation. Si c'est le cas, vous devez en fournir également l'implémentation.

Pour accéder aux caractères d'une chaîne de caractères, vous pouvez par exemple utiliser la méthode `charAt(int i)` de la classe `String` où `i` est l'index du caractère.

Q1 **(méthode `checkIdentifier`)**

.....

.....

.....

Examen du cours de Programmation et Algorithmique II
1ère Session, Mai 2016
Partie 1

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

Q1 **(méthode `checkIdentifier` – suite)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Mai 2016
Partie 1

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

Question 2 – Plus longue sous-chaîne commune (/7)

Cette question vise à concevoir et implémenter en Java un algorithme qui détermine la plus longue sous-chaîne commune à deux chaînes de caractères a et b. Prenons, par exemple, les chaînes a=`ABCBC` et b=`BCCBCA`. La plus longue sous-chaîne commune à a et b est `CBC`. Attention, `BCBC` n'est pas une réponse correcte car elle n'est pas une sous-chaîne de b.

L'approche qui vous est proposée consiste à d'abord déterminer pour chaque paire de préfixes de a et b le plus long suffixe commun. Le plus long de ces suffixes communs pris pour toutes les paires de préfixes est la plus longue sous-chaîne commune recherchée.

Dans notre exemple précédent, les préfixes de a sont `A`, `AB`, `ABC`, `ABCB` et `ABCBC` et ceux de b sont `B`, `BC`, `BCC`, `BCCB`, `BCCBC` et `BCCBCA`. Considérons le cas particulier du préfixe de longueur $m = 3$ de a et celui de longueur $n = 5$ de b. Ceux-ci sont illustrés avec leurs suffixes dans le tableau suivant. On peut constater que le plus long suffixe commun à ces préfixes est `BC`.

Chaîne	**Longueur du préfixe**	**Préfixe**	**Suffixes**
a	$m = 3$	`ABC`	**C**, **BC** et A**BC**
b	$n = 5$	`BCCBC`	**C**, **BC**, C**BC**, CC**BC** et BCC**BC**
			`C` et `BC` sont communs
			`BC` est le plus long

TABLE 1 – Plus long suffixe commun à deux préfixes de a et b.

Afin de résoudre le problème de la recherche de la plus longue sous-chaîne commune, il est utile de raisonner sur la longueur de celle-ci. De manière générale, notons $L(m, n)$ la longueur du plus long suffixe commun aux préfixes de longueur m de a et de longueur n de b. On peut écrire la relation de récurrence suivante. Si les derniers caractères des deux préfixes sont égaux, alors la longueur du plus long suffixe commun est obtenue en ajoutant 1 à la longueur du plus long suffixe commun aux deux préfixes chacun diminués d'un caractère (premier cas). Si l'un des préfixes est de longueur nulle ou si les caractères finaux sont différents, alors $L(m, n) = 0$ (second cas).

$$L(m,n) = \begin{cases} L(m-1,n-1)+1 & \text{si } m > 0 \text{ et } n > 0 \text{ et } a_{m-1} = b_{n-1} \\ 0 & sinon \end{cases}$$

Par exemple, si on considère les préfixes `ABC` et `BCCBC`, alors $L(3, 5) = L(2, 4) + 1$ car les derniers caractères sont égaux. De même, $L(2, 4) = L(1, 3)+1$ car les deux derniers caractères de `AB` et `BCCB` sont égaux. Finalement, $L(1, 3) = 0$ car les deux derniers caractères de `A` et `BCC` diffèrent. Au final, $L(3, 5) = L(2, 4) + 1 = L(1, 3) + 2 = 0 + 2 = 2$.

La Table 2 donne les valeurs de $L(m, n)$ pour toutes les paires de préfixes de a et de b (i.e. pour toutes les valeurs de m et n). Le cas où l'un des deux préfixes a une longueur nulle n'est pas illustré car $L(., 0) = L(0, .) = 0$. La case encerclée montre le cas $L(3, 5)$ (préfixes `ABC` et `BCCBC`). La case en gras montre le cas $L(5, 5) = 3$ qui correspond à la plus longue sous-chaîne `CBC`.

Examen du cours de Programmation et Algorithmique II

1ère Session, Mai 2016

Partie 1

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

		Chaîne b					
		B	C	C	B	C	A
Chaîne a	A	0	0	0	0	0	1
	B	1	0	0	1	0	0
	C	0	2	1	0	②	0
	B	1	0	0	2	0	0
	C	0	2	1	0	**3**	0

TABLE 2 – $L(m, n)$. Longueurs des plus longs suffixes pour toutes les paires de préfixes.

La plus longue sous-chaîne commune à a et b est celle qui correspond au plus long suffixe commun pris sur l'ensemble des préfixes. Sa longueur peut donc être obtenue en calculant

$$\max_{m,n} L(m, n)$$

Notes importantes pour obtenir une implémentation efficace :

- Il est possible d'éviter de calculer récursivement la longueur $L(m, n)$ des plus long suffixes communs. Pour y arriver, il suffit d'avoir calculé $L(m-1, n-1)$ avant de devoir calculer $L(m, n)$. Les valeurs de $L(m, n)$ déjà calculées doivent donc être stockées par l'algorithme.
- La plus longue sous-chaîne commune correspond à la plus grande valeur ($\max$) de $L(m, n)$. Un moyen naïf de calculer ce maximum consiste à itérer sur toutes les valeurs de $L(m, n)$. Cependant, il est possible de le faire plus efficacement.
- La chaîne recherchée (plus longue sous-chaîne commune) peut être déterminée au moment où le maximum est obtenu.

Ce qui vous est demandé :

Implémenter la méthode `longestCommonSubstring`. Celle-ci prend deux chaînes de caractères a et b en arguments et retourne la plus longue sous-chaîne commune. L'implémentation repose sur l'approche décrite ci-dessus. Remarque : il est possible qu'il existe plusieurs plus longues sous-chaînes communes à a et b. Dans ce cas, la méthode ne doit en retourner qu'une seule (p.ex. la première trouvée).

Examen du cours de Programmation et Algorithmique II
1ère Session, Mai 2016
Partie 1

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

Q2 **(méthode `longestCommonSubstring`)**

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

Q2 **(méthode `longestCommonSubstring` – suite)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Mai 2016
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 – Type de données Ensemble (/7)

Cette question concerne le type de données abstrait (ADT) Ensemble. Celui-ci permet de stocker des éléments non dupliqués et non ordonnés. Il fournit à cet effet des primitives permettant d'ajouter un élément à l'ensemble, de tester l'appartenance d'un élément à l'ensemble et de supprimer un élément de l'ensemble.

La Figure 1 montre l'interface `Set` qui correspond à l'ADT Ensemble. La méthode `add` ajoute un élément à l'ensemble. La méthode `addAll` ajoute à l'ensemble tous les éléments d'un autre ensemble. La méthode `clear` vide l'ensemble. La méthode `contains` retourne `true` si et seulement si l'élément appartient à l'ensemble. La méthode `isEmpty` retourne `true` si et seulement si l'ensemble est vide. La méthode `iterator` retourne un itérateur pour cet ensemble. La méthode `remove` enlève un élément de l'ensemble. La méthode `size` retourne le nombre d'éléments de l'ensemble.

L'objectif de la question est de concevoir et implémenter en Java la classe `LinkedListSet`, une implémentation de l'interface `Set`. Comme son nom l'indique, la classe `LinkedListSet` repose sur une liste chaînée interne. Un extrait de la classe `LinkedListSet` est donné à la Figure 2. Les éléments de la liste chaînée sont des instances de la classe interne `Node`. Le premier élément de la liste chaînée est référencé par la variable d'instance `head`. La variable d'instance `size` maintient le nombre d'éléments actuellement contenu dans l'ensemble.

Ce qui vous est demandé :

Q1a Implémentation de la méthode `add`. Cette méthode ne peut ajouter deux fois le même élément. Si l'élément à

Examen du cours de Programmation et Algorithmique II
1ère Session, Mai 2016
Partie 2

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

```
public interface Set<E> extends Iterable<E> {

         /* Ajoute un élément. Retourne true ssi l'élément a été ajouté (non dupliqué).
            Note : l'élément null peut être ajouté à l'ensemble. */
        boolean add(E e);

        /* Ajoute les éléments d'un autre ensemble. Retourne true ssi au moins
           un élément a été ajouté (non dupliqué). */
        boolean addAll(Set<? extends E> c);

        /* Vide l'ensemble. */
        void clear();

        /* Teste l'appartenance d'un élément à l'ensemble.
           Note : l'appartenance de l'élément null peut être testée. */
        boolean contains(Object o);

        /* Teste si l'ensemble est vide. */
        boolean isEmpty();

        /* Retourne un itérateur pour cet ensemble. */
        Iterator<E> iterator();

        /* Enlève un élément de l'ensemble. Retourne true ssi l'élément
           appartenait à l'ensemble. */
        boolean remove(Object o);

        /* Retourne le nombre d'éléments de l'ensemble. */
        int size();

}
```

FIGURE 1 – Interface `Set`.

ajouter appartient déjà à l'ensemble la méthode retourne **`false`**, sinon elle retourne **`true`**. L'élément **`null`** peut être ajouté à l'ensemble.

Q1b Implémentation de la méthode `addAll`. Les mêmes contraintes que pour `add` s'appliquent. La méthode retourne **`true`** si et seulement si au moins un élément a été ajouté à l'ensemble.

Q1c Implémentation de la méthode `clear`.

Q1d Implémentation de la méthode `contains`. Attention, des précautions doivent être prises dans cette méthode pour le test d'égalité entre instances ! De plus, l'élément **`null`** peut appartenir à l'ensemble.

Q1e Implémentation de la classe interne `_Iterator`. Pour rappel, l'interface `Iterator<E>` définit deux méthodes. La méthode **`boolean`** `hasNext()` retourne **`true`** s'il y a encore un élément à itérer. La méthode `E next()` retourne l'élément courant et avance l'itérateur vers l'élément suivant.

Q1f Implémentation de la méthode `iterator`.

Examen du cours de Programmation et Algorithmique II

1ère Session, Mai 2016

Partie 2

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

```
public class LinkedListSet<E> implements Set<E> {

        private class Node {
                E elt;
                Node next;
        }

        private Node head;
        private int size;

        (...)

}
```

FIGURE 2 – Classe `LinkedListSet`.

Q1g Implémentation de la méthode `remove`. Cette méthode retourne **`true`** si et seulement si l'élément à enlever appartenait à l'ensemble.

Q1a **(implémentation de la méthode `add`)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Mai 2016
Partie 2

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

Q1b **(implémentation de la méthode `addAll`)**

Q1c **(implémentation de la méthode `clear`)**

Q1d **(implémentation de la méthode `iterator`)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Mai 2016
Partie 2

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

Q1e **(implémentation de la méthode `contains`)**

Q1f **(implémentation de la classe interne `_Iterator`)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Mai 2016
Partie 2

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

Q1g **(implémentation de la méthode `remove`)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Mai 2016
Partie 2

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

Question 2 – Good luck ! (/3)

Le programme suivant affiche plusieurs valeurs à la console. Il vous est demandé de fournir les valeurs affichées. Pour chacune, donnez une brève explication de votre raisonnement. Il est en particulier important d'indiquer quels types de liaisons (*bindings*) sont effectuées.

```
public class GoodLuck {

        public static class A {
                public static String s= "Hello";
                public int x= 10;
                public int magic() {
                        return s.length();
                }
                public int getX() {
                        setX(0);
                        return x + getX();
                }
                public void setX(int x) {
                        x= x;
                }
        }

        public static class B extends A {
                public final String s= "Bonjour";
                public int magic() {
                        return this.s.length() - s.length();
                }
                public void setX() {
                        ((A) this).setX(x);
                }
        }

        public static class C extends B {
                public static int x= 1;
                public int getX() {
                        if (x > 0)
                                return super.getX();
                        else
                                return -1;
                }
                public void setX(int x) {
                        this.x= x;
                }
        }

        (... suite page suivante ...)
```

Examen du cours de Programmation et Algorithmique II

1ère Session, Mai 2016

Partie 2

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

```
        public static void main(String [] args) {
                A x= new C();
                A y= new A();
                A z= new B();
                System.out.println(x.s);                                    ( Q2a )
                System.out.println(((B) x).s);                              ( Q2b )
                System.out.println(x.magic());                              ( Q2c )
                System.out.println(y.magic());                              ( Q2d )
                System.out.println(x.s == y.s);                             ( Q2e )
                z.setX(3);
                System.out.println(z.x);                                    ( Q2f )
                System.out.println(x.getX());                               ( Q2g )
                System.out.println(x.getX());                               ( Q2h )
                y.setX(100);
                System.out.println(y.x);                                    ( Q2i )
                System.out.println(y.getX());                               ( Q2j )
        }

}
```

Q2a **(x.s)**

.....

.....

Q2b **(((B) x).s)**

.....

.....

Q2b **(x.magic())**

.....

.....

Q2d **(y.magic())**

.....

.....

Examen du cours de Programmation et Algorithmique II
1ère Session, Mai 2016
Partie 2

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

Q2e **(`x.s == y.s`)**

. . . . . .

. . . . . .

Q2f **(`z.x`)**

. . . . . .

. . . . . .

Q2g **(`x.getX()`)**

. . . . . .

. . . . . .

Q2h **(`x.getX()`)**

. . . . . .

. . . . . .

Q2i **(`y.x`)**

. . . . . .

. . . . . .

Q2j **(`y.getX()`)**

. . . . . .

. . . . . .