Nom : Prénom :
Numéro de matricule : Année d'étude :
Groupe :

Examen SESP 1224 : Introduction à l'Informatique et la Programmation (année académique 2004-2005)

Professeur : Marco Saerens
Adresse : Université catholique de Louvain
Institut d'Administration et de Gestion
Information Systems Research Unit (ISYS)
Place des Doyens 1
B-1348 Louvain-la-Neuve, Belgique
Téléphone : 010 47.92.46. (Saerens)
Fax : 010 47.83.24.
Courriel : saerens@isys.ucl.ac.be
Web : http://www.isys.ucl.ac.be/etudes/cours/index.htm

Enoncés des problèmes

Vous avez **3 heures** pour résoudre les 4 problèmes suivants, cotés sur un total de 14 points. Lorsque vous aurez terminé, ou bien lorsqu'on vous le demandera, vous remettrez vos feuilles. Lisez attentivement les énoncés !

Attention:
- Vérifiez que vous avez huit feuilles au total: deux pages d'énoncé, quatre feuilles à compléter (feuilles réponses) et quatre feuilles de brouillon. Vous devez impérativement utiliser le papier brouillon fourni.
- Il y a donc une feuille réponse à compléter par problème à résoudre (4 problèmes); vous pouvez écrire sur le recto et le verso.
- Vous devez compléter vos **nom et prénom** pour chaque feuille réponse ainsi que l'énoncé.
- Il faut rendre les quatre feuilles réponses, **même celles qui sont vides**.
- N'oubliez pas d'inclure des **commentaires** expliquant ce que vous faites.

Voici les énoncés des quatre problèmes (de difficulté croissante) à résoudre. Notez que le dernier (4ème) est plus court et moins important.

Problème 1 (45 min; 4 points).
Ecrivez une méthode, *nombreAuCarre*, qui reçoit un nombre entier en argument et qui, sur cette base, calcule et renvoie un entier représentant ce nombre au carré. Le calcul du carré de ce nombre doit se faire de la manière suivante: le carré du nombre *x* est égal à la somme des *x* premiers nombres impairs positifs (à commencer par 1).

Par exemple, le carré du nombre 4 est égal à 1+3+5+7 = 16 (somme des 4 premiers nombres impairs positifs).

Signature de la méthode : `public int nombreAuCarre(int x)`

Problème 2 (45 min; 4 points).
Ecrivez une méthode, *sommeDiagonale*, qui calcule et renvoie la somme des nombres pairs ou impairs de la diagonale d'une matrice carrée contenant des nombres entiers. Une matrice carrée est une matrice dont le nombre de lignes est égal au nombre de colonnes. En effet, sur base de la valeur d'un booléen « *pair* », passé en paramètre, la méthode calculera:

- Soit la somme des nombres pairs si le paramètre booléen « *pair* » est égal à *true*.
- Soit la somme des nombres impairs si le paramètre booléen « *pair* » est égal à *false*.

Si la matrice n'est pas carrée, la méthode doit retourner « *null* » et imprimer une erreur.

Exemple

Soit la matrice:

1	…	…	…
…	2	…	…
…	…	3	…
…	…	…	4

La méthode appliquée à cette matrice renverra la valeur « 6 » (= 2 + 4) si *pair* est *true* et « 4 » (= 1 + 3) si *pair* est *false*.

Signature de la méthode : `public int sommeDiagonale(int[][] matrice, boolean pair)`

Problème 3 (45 min; 4 points).
Ecrivez une méthode, *sousChaine*, qui détecte si une chaîne de caractères **y** est comprise dans une autre chaîne de caractères **x**. Plus précisément, il faut vérifier si **y** est une sous-chaîne de **x** : tous les caractères de **y** apparaissent séquentiellement dans le même ordre dans **x**, mais avec éventuellement des insertions d'autres caractères. On peut déjà en déduire que **x** doit être au moins de la même longueur que **y**. Notez qu'il est inutile d'utiliser la méthode Java *substring*; il faut plutôt parcourir la chaîne **x** et comparer les caractères des chaines **x** et **y**.

Cette méthode renverra *true* si **y** est une sous-chaîne de **x** et *false* sinon.

Par exemple, si **x** = 'abcUdeCfgLhij' et **y** = 'UCL', il faut renvoyer *true* ('UCL' est compris dans 'abcUdeCfgLhij', dans le bon ordre). En revanche, si **x** = 'abcUdeLfgChij', il faut renvoyer *false* car 'UCL' n'apparaît pas dans le bon ordre dans **x**.

Signature de la méthode : `public boolean sousChaine(String x, String y)`

Problème 4 (15 min; 2 points).
Décrivez ce que réalise cette méthode; il s'agit de l'implémentation d'un algorithme qui effectue une opération bien précise, à partir de deux nombres entiers positifs, *nombre1* et *nombre2*. En d'autres termes, que renvoie cette méthode ? Essayez plusieurs combinaisons de valeurs afin d'analyser le comportement de la méthode.

```
// Précondition : nombre1 et nombre2 sont des nombres entiers positifs.
public static int calcul(int nombre1, int nombre2)
{
    while (nombre1 != nombre2)
    {
       if (nombre1 > nombre2) nombre1 = nombre1 - nombre2;
       else nombre2 = nombre2 - nombre1;
    }
    return(nombre1);
}
```

Il ne nous reste plus qu'à vous souhaiter bonne chance !

14/06/2005

Feuille de réponse au problème 1

Nom :
Prénom :
Numéro de matricule :
Année d'étude :
Groupe :

14/06/2005

Feuille de réponse au problème 2

Nom :
Prénom :
Numéro de matricule :
Année d'étude :
Groupe :

14/06/2005

Feuille de réponse au problème 3

Nom :
Prénom :
Numéro de matricule :
Année d'étude :
Groupe :

14/06/2005

Feuille de réponse au problème 4

Nom : Prénom :

Numéro de matricule : Année d'étude :

Groupe :