CSI 1100 / 1500
Automne 2004
Introduction à l'informatique I
EXAMEN FINAL

Durée : 3 heures 9 Décembre 2004, 9h30

Professeurs : Daniel Amyot, Alan Williams

Nom (lettres moulées): _____

Numéro d'étudiant: _____

Instructions – lisez attentivement!

- **Écrivez votre nom et votre numéro d'étudiant au stylo.**
- Cet examen est à livres fermés. **Aucun livre ou appareil électronique (incluant les calculatrices) n'est permis.**
- Il y a 7 questions à l'examen, dont quelques-unes avec des sous-questions. Répondez à ces questions directement sur le questionnaire, dans l'espace prévu. **Les réponses écrites au crayon de plomb ne pourront pas être recorrigées.**
- Les points alloués à chaque question sont indiqués. Les questions n'ont pas toutes le même poids, alors planifiez votre temps en conséquence. Cet examen est noté sur 100 points et représente 55% de votre note finale.
- Les algorithmes doivent être décrits à l'aide du format de pseudocode vu en classe et dans les notes de cours.
- Vous pouvez utiliser le verso des pages pour vos calculs et brouillons. Les pages 12 et 13 peuvent être détachées car elles ne seront pas corrigées.
- Answers in English are accepted.

Question	Points	Points obtenus
1	12	
2	8	
3	15	
4	15	
5	15	
6	25	
7	10	
Total	100	

Question 1A) (4 points)

Pour cette question, votre expression Booléenne (en **pseudocode**) ne peut contenir que :

- les opérateurs de comparaison <, >, =, ≤, ≥, et ≠
- les connecteurs Booléens NON, ET, OU
- les opérateurs mathématiques +, -, *, /, et MOD (modulo)
- Les noms des variables, de même que des constantes.

Utilisez les parenthèses lorsque nécessaire, et évitez la syntaxe Java!

Environnement Canada va annoncer une valeur *humidex* dans ses prévisions météo si la température (T) est plus grande ou égale à 30 degrés, si la température est plus grande ou égale à 25 degrés et que le taux d'humidité (H) est plus grand que 35%, ou si la température est plus grande ou égale à 20 degrés et que le taux d'humidité est plus grand ou égal à 65%.

Écrivez une expression Booléenne qui sera vraie si Environnement Canada va annoncer une valeur *humidex*, et fausse sinon.

Réponse:

Question 1B) (4 points)

Observez le programme Java suivant :

```
MaClasse[] obj;
int index;

obj = new MaClasse [2];
index = 15;
while( index > 2 )
{
   obj[index%2] = new MaClasse ( );
   index = index / 2;
}
// Ligne X
```

i) Combien d'instances de `MaClasse` ont été créées pendant l'exécution de ce programme? (2)
Réponse:

ii) Combien d'instances de `MaClasse` sont encore accessible à la « Ligne X »? (2)
Réponse:

Question 1C) (4 points)

Considérez les deux classes suivantes:

```
class Foo
{
  private int x1;
  public static int x2;
  public static Bar x3;

  public Foo(int x4)
  {
       ...
  }
}
```

```
class Bar
{
  public int x5;

  public static int x6()
  {
         ...
  }

  public Foo x7()
  {
         ...
  }
}
```

Les instructions suivantes (indépendantes les unes des autres) sont utilisées dans la méthode `main()` d'une classe `Test`. Encerclez l'option qui causera une **erreur de compilation**.

(a)
```
Foo[] a = new Foo[5];
a[4] = new Foo(-1);
```

(b)
```
Foo f = Bar.x7();
```

(c)
```
Foo.x2 = Bar.x6();
```

(d)
```
int k = Foo.x3.x5;
```

(e)
```
Bar b = new Bar();
Foo f = b.x7();
```

Question 2: (8 points)

Voici un programme contenant une méthode récursive.

```
class Football
{
  public static void main(String[ ] args)
  {
    char[] t = {'G', 'e', 'e', '-', 'G', 'e', 'e'};
    Rec (t, t.length - 1);
  }

  public static void Rec(char[] var, int i)
  {
    if (i < 0)
    {
      ; // Ne rien faire
    }
    else
    {
      if ( (var[i] > 'A') && (var[i] < 'Z') )
      {
        System.out.print ( (char) (var[i] - 'A' + 'a') );
      }
      else
      {
        if ( (var[i] > 'a') && (var[i] < 'z') )
        {
          System.out.print ( (char) (var[i] - 'a' + 'A') );
        }
        else
        {
          ; // Ne rien faire
        }
      }

      Rec (var, i - 1);

    }
  }
}
```

Que sera-t-il affiché si nous exécutons la méthode `main` de cette classe?

Réponse :

Question 3: (15 points)

Traduisez ici l'algorithme de la page 13 en une **méthode Java**. Attention à la propreté…

Question 4: (15 points)

Aux Jeux Olympiques, le pointage pour un plongeon de la plateforme de 10 mètres est calculé de la façon suivante. Chaque plongeon a un « coefficient de difficulté » (CD) basé sur la complexité des éléments techniques inclus (par exemple, un 2 ½ saut périlleux avant en position groupée a un CD de 2.7). Chaque juge observe le plongeon de l'athlète, puis soumet sa note (sur 10 points). La note la plus haute et la note la plus basse sont enlevées, et le pointage pour ce saut est calculé en faisant la somme des notes qui restent et en la multipliant par le coefficient de difficulté.

Concevez un algorithme qui va calculer le pointage pour le plongeon d'un athlète à partir d'un tableau de notes soumises par N juges, pour un plongeon de coefficient de difficulté CD. Veuillez inclure les commentaires d'usage.

Question 5: (15 points)

La sous-matrice *bas-droite* d'une matrice est formée en sélectionnant un élément (selon sa ligne et sa colonne) et en excluant tous les éléments qui se trouvent à gauche ou au-dessus de l'élément sélectionné. Par exemple, dans la matrice ***M*** ci-dessous, si nous sélectionnons $\boldsymbol{M}_{11} = 5$, alors ***S*** est la sous-matrice bas-droite résultante.

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad\qquad S = \begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix}$$

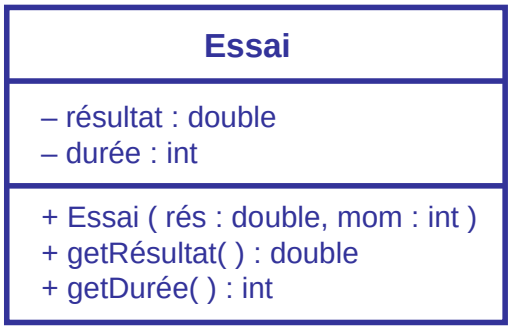
Écrivez une méthode Java qui prendra en entrée une matrice d'entiers ***M***, de même que des index de ligne et de colonne, et qui retournera une nouvelle matrice qui sera la sous-matrice bas-droite de ***M*** formée à partir de cette position. L'en-tête de cette méthode est la suivante:

```
public static int[][] sousMatrice( int[][] m, int ligne, int col )
```

Question 6: (25 points)

Pour cette question, vous devez écrire une classe **`Expérience`** qui représente un enregistrement d'une quelconque expérience scientifique. Afin de s'assurer que les résultats d'une expérience soient répétables, il y a une classe **`Essai`** qui contient les résultats d'*une* exécution de l'expérience. Une expérience contiendra donc un certain nombre d'objets **`Essai`**.

La classe **`Essai`** emmagasine un résultat mesuré pendant l'exécution de l'expérience, de même que la durée de cette expérience (en millisecondes). Cette classe a déjà été implémentée; elle contient un constructeur et des accesseurs simples, et son diagramme UML est le suivant :



Dans les deux pages suivantes, vous devrez développer les méthodes requises pour la classe **`Expérience`**. Votre classe devra offrir quatre méthodes publiques qui permettront à la méthode **`main`** de la classe **`TestExpérience`** d'être exécutée sans problème:

```
class TestExpérience
{
   public static void main (String[] args)
   {
      Expérience uneExpérience;

      uneExpérience = new Expérience( 2 );
      uneExpérience.ajouteEssai( new Essai( 99.1, 10000 ) );
      uneExpérience.ajouteEssai( new Essai( 97.1, 11000 ) );
      uneExpérience.ajouteEssai( new Essai( 94.1, 12000 ) );
      Expérience.setPrédiction( 98.6 );
      uneExpérience.affiche();
   }
}
```

Résultats affichés :

```
Aucun autre essai ne peut être ajouté à cette expérience.
Essai 0: Résultat 99.1, durée de 10000 (à 0.5 de la prédiction)
Essai 1: Résultat 97.1, durée de 11000 (à 1.5 de la prédiction)
```

Question 6 (suite)

// Classe Expérience (sur 2 pages), à compléter. Contient des champs, une méthode constructeur,
// une méthode pour mettre à jour la prédiction, une méthode pour modifier les essais, et une
// méthode d'affichage.

```
class Expérience
{
```
// DÉCLARATIONS DES CHAMPS (4 points)

// MÉTHODE CONSTRUCTEUR (5 points)
// Prend en paramètre un entier représentant le nombre maximum d'essais pour l'expérience.

// MÉTHODE MODIFICATEUR ***setPrédiction*** *: (4 points)*
// Paramètres de la méthode: nombre réel représentant le résultat prédit pour l'expérience.
// Résultat: (aucun)
// Modifiée: la prédiction pour l'expérience.

*// MÉTHODE **ajouteEssai** : (6 points)*
// Paramètres de la méthode: un objet Essai à ajouter à cette expérience. Résultat: (aucun)
// Modifiés: les champs de l'objet Expérience
// Cette méthode doit aussi être robuste: elle affichera un message si l'expérience
// n'a plus de place pour de nouveaux essais (voir 2e page précédente pour le format).

*// MÉTHODE **affiche** : (6 points)*
// Paramètres de la méthode: (aucun). Résultat: (aucun)
// Cette méthode affiche les résultats et durées de chaque essai, de même que la valeur
// absolue de la différence avec la valeur prédite.
// Voir les résultats de l'affichage pour le format exact (2e page précédente).

```
} // Fin de la classe Expérience
```

Question 7: (10 points)

Supposons que nous ayons un tableau d'entiers déjà trié en ordre croissant et qui ne contient pas de valeurs dupliquées. Une *recherche binaire* sur un tel tableau se fait en localisant l'élément au milieu du tableau et en comparant la valeur recherchée avec cet élément. Si nous n'avons pas trouvé la valeur recherchée, alors nous pouvons choisir le sous-intervalle approprié (à gauche ou à droite) pour restreindre la recherche.

Écrivez une méthode **récursive** qui va faire une recherche binaire de la valeur `trouveMoi` dans un tableau `t` entre les positions `indexDépart` et `indexFin`. Vous pouvez supposer que ces deux positions sont plus petites que la longueur du tableau. La méthode devra retourner `true` si la valeur se trouve dans le tableau, et `false` sinon.

L'en-tête de cette méthode est la suivante:

```
public static boolean rechercheBinaire(int[] t, int trouveMoi,
                                       int indexDépart, int indexFin)
```

Cette page est disponible pour vos calculs et autres brouillons (elle ne sera pas corrigée).

Algorithme à traduire pour la Question 3

NB : Vous pouvez détacher cette page car elle ne sera pas corrigée.

DONNÉES:

Base:	*(une base pour logarithme)*
Opérande:	*(tableau d'entiers pour lesquels on cherche le logarithme entier)*
N:	*(nombre de valeurs dans le tableau Opérande)*

RÉSULTAT:

IntLog:	*(tableau de N logarithmes entiers pour les valeurs du tableau Opérande ; la valeur -1 est utilisée si le logarithme n'existe pas)*

INTERMÉDIAIRES:

Index	*(index pour tableau)*
Valeur	*(utilisée pour divisions successives)*
Compteur	*(compte le nombre de fois qu'un opérande peut être divisé par la base)*

EN-TÊTE: IntLog ← Logarithmes(Base, Opérande, N)

MODULE: