Examen ITI 1620 de 2005

Question 1A [4]

- Le programme « Air pur Ontario » exige qu'une voiture doive passer un test d'émissions pour le renouvellement de l'immatriculation si les deux conditions suivantes sont satisfaites :
  - L'année de fabrication (AF) est impaire et l'année courante (AC) est paire, ou vice versa;
  - La différence entre l'année courante et l'année de fabrication est d'au moins 3 ans mais pas plus de 20 ans.
- Écrivez une expression Booléenne qui sera vraie seulement si une voiture doit passer un test d'émissions « Air pur », et fausse sinon.
- Devoir 2 2006!

(((AF MOD 2 = 1) ET (AC MOD 2 = 0)) OU ((AF MOD 2 = 0) ET (AC MOD 2 = 1))) ET ((AC – AF ≥ 3) ET (AC – AF ≤ 20))

Solution alternative:

((AF MOD 2 + AC MOD 2) = 1) ET ((AC – AF ≥ 3) ET (AC – AF ≤ 20))

Question 1B [4]

- Que sera-t-il affiché en sortie de ce programme? Écrivez votre réponse (seulement ce qui est affiché) sous le programme.

```
class ABC
{
    public static void main(String[ ] args)
    {
        char[][] x = {{'i', 'j'}, {'4', '5'}};
        char[][] y = {{'x', 'y'}, {'a', 'b'}};
        int i;
        char t;

        for (i = 0; i < 2; i=i+1)
        {
           t = y[(i+1)%2][1];
           y[i][(i+1)%2] = x[i][0];
           y[(i+1)%2][1] = t;
        }

        System.out.print(y[0][1]);
        System.out.println(y[1][0]);
    }
}
```

Réponse: i4

Question 1C [4]

```
class C1
{
    private int[] v1 = {1,4,9};
    public int v3;

    public static int m1 (C2 p)
    {
       ...
    }

    private C1 m2 (int m)
    {
       ...
    }
}
```

```
class C2
{
    public static char v3;

    public C2 (int n)
    {
       ...
    }

    private void m3 (int m)
    {
       ...
    }
}
```

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

```
(a)  int v2 = C1.m1(this);   // Erreur: this pas un objet de type C2
(b)  char v3 = C2.v3;        // Pas d'erreur!
(c)  C1 x = new m2(4);       // Erreur: méthode privée, pas constructeur!
(d)  C1 w = new C1();
     w.v1[2] = 3;            // Erreur: champ v1 privé
(e)  C2 y = new C2();        // Erreur: constructeur par défaut n'existe plus
     int z = C1.m1(y);       // (remplacé par constructeur avec un int)
```

Question 2 [8]

```
class UneClasse
{
  public static void main(String[ ] args)
  {
    uneMéthode(137210);
    System.out.println( );
  }

  public static void uneMéthode(int i)
  {
    if (i == 0)
    {
      ; // ne rien faire
    }
    else
    {
      if ( i%10 == 1 )
      {
        System.out.print( "un " );
      }
      else
       {
          if ( i%10 == 2 )
          {
            System.out.print( "deux " );
          }
          else
          {
             if ( i%10 == 3 )
             {
               System.out.print( "trois " );
             }
             else
             {
                ; // ne rien faire
             }
          }
       }
       uneMéthode (i/10);
    }
  }
}
```

Réponse: un deux trois un

Question 3A [10]

- Traduisez ici l'algorithme en une **méthode Java**
- Cet algorithme détermine si une phrase est un palindrome ou non. Dans cette question, un palindrome est un mot ou une phrase qui s'écrit exactement de la même façon de gauche à droite comme de droite à gauche. Par exemple « été », « 2002 », et « un nu » sont des palindromes, alors que « Laval » ne l'est pas ('l' est différent de 'L').

DONNÉES:
- Phrase — *(un tableau de caractères représentant une phrase à vérifier)*
- Bas — *(indice de gauche pour Phrase)*
- Haut — *(indice de droite pour Phrase)*

RÉSULTAT:
- EstPal — *(vrai la Phrase est un palindrome)*

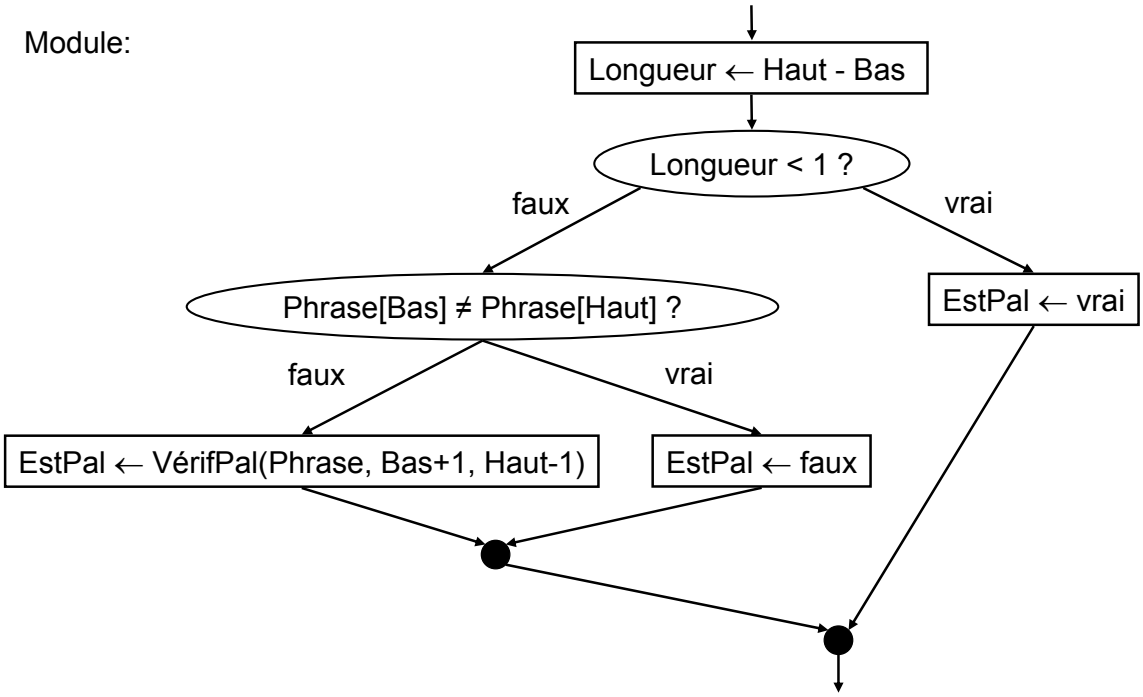
INTERMÉDIAIRES:
- Longueur — *(taille de l'intervalle de la Phrase considéré)*

EN-TÊTE: EstPal ← VérifPal (Phrase, Bas, Haut)

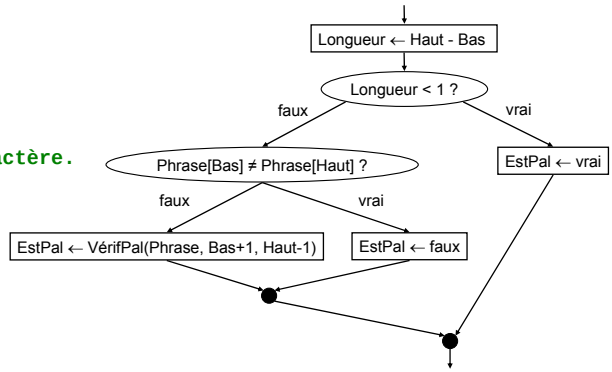
Traduction partielle

```
// MÉTHODE vérifPalindrome: vérifie si phrase est un palindrome
// Paramètres:
// - phrase: un tableau de caractères représentant une phrase à vérifier
// - bas: indice de gauche pour phrase
// - haut: indice de droite pour phrase

public static boolean vérifPal (char [] phrase, int bas, int haut)
{
   // DÉCLARATION DES VARIABLES / DICTIONNAIRE DE DONNÉES
   int longueur;   // INTERMÉDIAIRE: taille de l'intervalle de la Phrase considéré
   boolean estPal; // RÉSULTAT: vrai si phrase est un palindrome
```

Module:



```
   // MODULE DU SOUS-ALGORITHME
   longueur = haut - bas;
   if (longueur < 1)
   {
      // Cas de base
      estPal = true;  // Zéro ou un caractère.
   }
   else
   {
      // Cas récursif, efficace
      if (phrase[bas] != phrase[haut])
      {
         estPal = false;
      }
      else
      {
         // Les premier et dernier caractères sont égaux. Vérifie le reste.
         estPal = vérifPal(phrase, bas + 1, haut - 1);
      }
   }

   // RÉSULTAT RETOURNÉ
   return estPal;
}
```



Question 3B [5]

- Ajoutez une méthode de démarrage qui invoquera votre méthode de la Question 3a). Cette méthode devrait être invocable de la façon suivante : estPalindrome = palindrome(uneString);
- Pas besoin d'ajouter les commentaires d'usage pour cette sous-question.

```
// Méthode de démarrage…
public static boolean palindrome (String phrase)
  {
    boolean résultat;
    char[] phraseTableau;

    phraseTableau = phrase.toCharArray(); // Ou boucle avec charAt()...
    résultat = vérifPal(phraseTableau , 0, phrase.length() - 1);
    return résultat;
  }
```

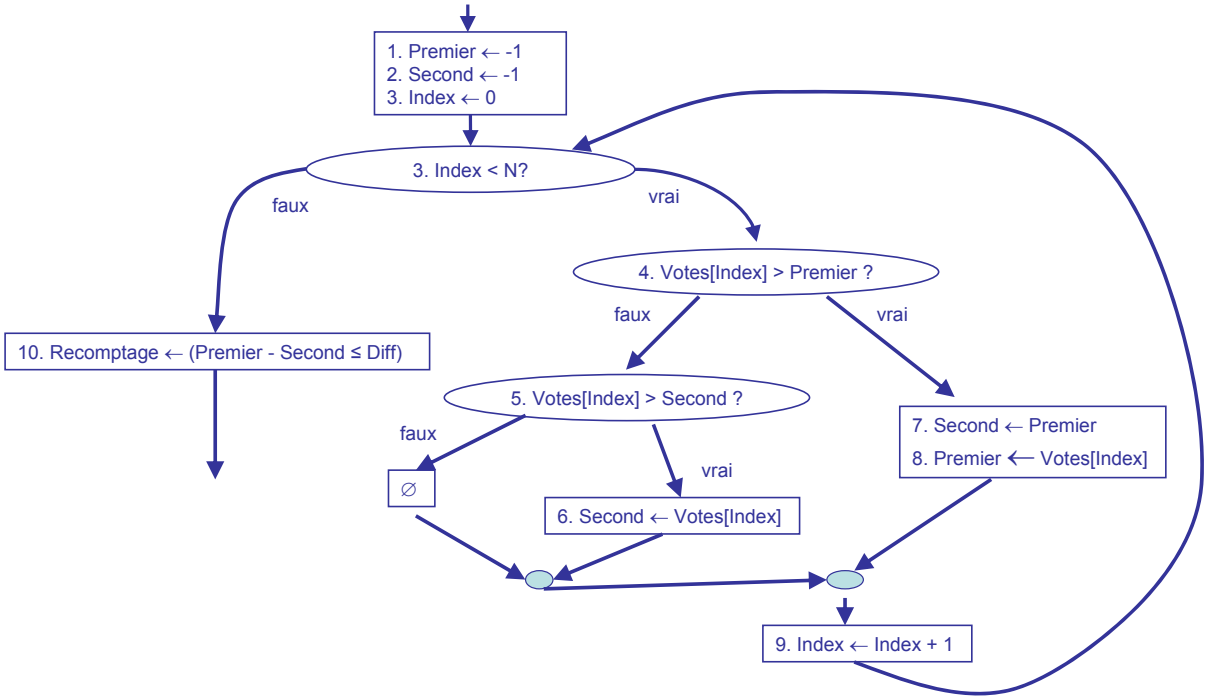
Question 4 [15]

- Pour l'élection qui approche, le candidat qui a le plus de votes dans un district électoral remporte et devient Député. Cependant, si la différence entre les nombres de votes des deux premiers candidats (c'est-à-dire, celui qui a le plus de votes et celui qui le suit) est très faible, alors un recomptage aura lieu.
- Écrivez un algorithme qui va prendre en entrée un tableau **Votes** contenant le nombre de votes de chacun des **N** candidats dans un district électoral, ainsi qu'une valeur **Diff** (où si la différence entre les nombres de votes des deux meilleurs candidats est plus petite ou égale à **Diff**, un recomptage aura lieu). Votre algorithme devra retourner *vrai* si les votes devraient être recomptés et *faux* sinon. Le tableau **Votes** n'est pas trié.

Question 4 - Solution

DONNÉES:
  Votes *(tableau de nombre de votes)*
  N *(taille du tableau)*
  Diff *(différence maximale pour recomptage)*
INTERMÉDIAIRES:
  Premier *(plus grand nombre de votes dans le tableau)*
  Second *(2e plus grand nombre de votes dans le tableau)*
  Index *(index pour parcourir le tableau)*
RÉSULTAT:
  Recomptage *(vrai ssi un recomptage est nécessaire)*
EN-TÊTE:
  Recomptage ← DétermineRecomptage(Votes, N, Diff)
HYPOTHÈSE:
  Votes contient au moins 2 éléments, et ils sont tous positifs.
MODULE:

Question 4 - Module



Question 5 [10]

- Un « carré magique » est une matrice carrée d'entiers où la somme de chaque ligne et de chaque colonne est la même. Par exemple, dans la matrice ***A*** suivante, la somme de chaque ligne et de chaque colonne est 15.

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

- Écrivez une méthode Java qui prendra en entrée une matrice d'entiers ***A*** et qui retournera *true* si A représente un « carrée magique », et *false* sinon. Votre méthode devrait être efficace.

```
public static boolean estMagique( int[][] m )
```

Question 5 (suite)

```
public static boolean estMagique( int[][] m )
   {
        // Note: si l'hypothèse est que m est un carré 3x3,
        // on peut vérifier que la somme est 15.
        // L'approche ici est: toutes les lignes/cols ont une
        // somme égale. (Peu importe la taille de m!)

        int sommeMagique; // INTERMÉDIAIRE: Somme attendue
        int somme;        // INTERMÉDIAIRE: Somme courante
        int ligne;        // INTERMÉDIAIRE: index
        int col;          // INTERMÉDIAIRE: index
        boolean magique;  // RÉSULTAT

        maqique = true;
        sommeMagique = 0; // Seulement pour satisfaire le compilateur qui
                          // pourrait penser que sommeMagique est utilisé
                          // avant d'être initialisé. La valeur utilisée
                          // n'a pas d'importance
```

Question 5 (suite)

```
        // Vérifie les lignes
        ligne = 0;
        while ( ligne < m.length && magique ) // Efficace
        {
            somme = 0;
            col = 0;
            while ( col < m[ligne].length )
            {
                somme = somme + m[ligne][col];
                col = col + 1;
            }
            if ( ligne == 0 )
            {
                sommeMagique = somme;
            }
            else
            {
                magique = ( somme == sommeMagique );
            }
            ligne = ligne + 1;
        }
```

Question 5 (suite)

```
        // Vérifie les colonnes
        col = 0;
        while ( col < m[0].length && magique )  // Efficace
        {
            somme = 0;
            ligne = 0;
            while ( ligne < m.length )
            {
                somme = somme + m[ligne][col];
                ligne = ligne + 1;
            }
            maqique = ( somme == sommeMagique );
            col = col + 1;
        }
        return magique;
    }
}
```

Question 6 [25]

- Pour cette question, vous devrez créer une classe **Annuaire** qui pourra contenir un ensemble d'objets **Ami**.
- La classe **Ami**, qui a déjà été implémentée, permet d'emmagasiner le nom et le numéro de téléphone d'une personne. Elle contient une méthode constructeur, deux méthodes accesseur, et une méthode modificateur. Voici son diagramme de classe UML :

Ami
- nom : String - numéro: String
+ Ami(sonNom : String, sonNuméro : String) + getNom( ) : String + getNuméro( ) : String + setNuméro (nouveauNuméro : String) : void

Question 6 (suite)

- Dans les deux pages suivantes, vous devez compléter la classe **Annuaire** en fournissant un constructeur et trois méthodes publiques qui permettront à la méthode **main** de la classe **TestAnnuaire** d'être exécutée:

```
class TestAnnuaire
{
   public static void main (String[] args)
   {
      Annuaire monAnnuaire = new Annuaire( 2 );
      monAnnuaire.ajouteAmi( new Ami("Alice", "555-1212" ) );
      monAnnuaire.ajouteAmi( new Ami("Tommy", "555-3434" ) );
      monAnnuaire.ajouteAmi( new Ami("Pizza", "737-1111" ) );
      monAnnuaire.changeNuméro( "Tommy", "867-5309" );
      monAnnuaire.affiche();
      monAnnuaire.changeNuméro ( "Pizza", "310-1010" );
   }
}
```

Résultats affichés :

```
   L'annuaire est plein.
   Nom: Alice, Téléphone: 555-1212
   Nom: Tommy, Téléphone: 867-5309
   Pizza n'est pas un nom connu de l'annuaire.
```

Question 6 (suite)

```
class Annuaire
{
  // DÉCLARATIONS DES CHAMPS (3 points)
  private Ami [] amis;
  private int nbAmis;
  private int maxAmis;

  // MÉTHODE CONSTRUCTEUR : (5 points)
  // prend en paramètre un entier représentant le nombre
  // maximum d'amis que peut contenir l'annuaire.
  public Annuaire(int max)
  {
    amis = new Ami[max];
    maxAmis = max;
    nbAmis = 0;
  }
```

Question 6 (suite)

```
// MÉTHODE MODIFICATEUR ajouteAmi: (6 points)
// Paramètres de la méthode: un objet Ami à ajouter à l'annuaire.
// Résultat:  affichera un message si l'annuaire est déjà plein
   (voir page 8 pour son format)
// Modifié:  l'objet Annuaire

  public void ajouteAmi(Ami unAmi)
  {
    if (nbAmis < maxAmis)
    {
      amis[nbAmis] = unAmi;
      nbAmis = nbAmis + 1;
    }
    else
    {
      System.out.println("L'annuaire est plein.");
    }
  }
```

Question 6 (suite)

```
// MÉTHODE affiche : (5 points)
// Paramètres de la méthode: (aucun)
// Résultat: (aucun)
// Cette méthode affiche la liste des noms et numéros de
// téléphone de l'annuaire. Voir les résultats de
// l'affichage de TestAnnuaire pour le format exact

  public void affiche()
  {
    for (int index=0; index < nbAmis; index = index + 1)
    {
      System.out.print("Nom: " + amis[index].getNom());
      System.out.println(", Téléphone: " +
                         amis[index].getNuméro());
    }
  }
```

```
// MÉTHODE changeNuméro: (6 points)
// Met à jour le numéro de téléphone du premier Ami dont le nom est spécifié.
// Paramètres de la méthode: le nom de l'ami (String) et le nouveau numéro de téléphone de
    cet ami (String)
// Résultat: un message d'erreur si l'ami spécifié n'est pas dans l'annuaire (voir page 8
    pour format)
// NOTE: l'égalité de deux objets String s1 et s2 est vérifiable avec s1.equals(s2)

  public void changeNuméro(String nom, String num)
  {
    int index = 0;
    boolean trouvé = false;
    while ( (index < nbAmis) && (!trouvé) )
    {
      if ( nom.equals(amis[index].getNom()) )
      {
        amis[index].setNuméro(num);
        trouvé = true;
      }
      else
      {
        index = index + 1;
      }
    }
    if (!trouvé)
    {
      System.out.println(nom + " n'est pas un nom connu de l'annuaire.");
    }
    else
    {
      ; // ne rien faire
    }
  }
}
```

Question 7 [10]

- Écrivez une méthode **récursive** qui prendra comme paramètre un entier non négatif et qui générera un dessin composé d'étoiles tel qu'affiché ci-dessous. Vous pouvez utiliser une boucle pour générer une ligne d'étoiles, mais **pas** le dessin en entier. Si l'entier non négatif est 4, alors le dessin généré sera :

```
****
***
**
*
*
**
***
****
```

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

```
public static void étoiles(int n)
```

```
public static void étoiles( int n )
   {
        int index;
        if ( n == 0 )
        {
            ; // Cas de base? On ne fait rien!
        }
        else
        {
            // On affiche une ligne de n étoiles
            for (index = 0; index < n; index=index+1 )
            {
                System.out.print("*");
            }
            System.out.println( );
            // On affiche le diagramme plus petit
            étoiles( n - 1 );
            // On affiche une ligne de n étoiles
            for (index = 0; index < n; index=index+1 )
            {
                System.out.print("*");
            }
            System.out.println( );
        }
   }
```