Université d'Ottawa
Faculté de génie

École d'ingénierie
et de technologie de l'information

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Information
Technology and Engineering

ITI 1120/1520 Automne 2008
Introduction à l'informatique I

EXAMEN FINAL - Solutionnaire

Durée de l'examen: 3 heures **19 décembre 2009, 14:00**

Professeur: Gilbert Arbez, Mohamad Eid

Nom de famille : _____

Prénom(s) : _____

Numéro d'étudiant : _____ Signature : _____

1. Cet examen est à **livres fermés**. Aucun livre ou appareil électronique (calculatrices) n'est permis.
2. Il y a 7 questions à l'examen, dont quelques-unes avec des sous-questions, et les points alloués sont indiqués. Répondez à ces questions directement sur le questionnaire, dans l'espace prévu.
3. Cet examen est noté sur 100 points et représente 50% de votre note finale.
4. Les algorithmes doivent être décrits à l'aide du format vu en classe et dans les notes de cours.
5. Vous pouvez utiliser le verso des pages pour vos calculs et brouillons. Les pages 9, 16 et 17 peuvent être détachées car elles ne seront pas corrigées (n'enlevez pas la broche!!!).
6. Si vous ne comprenez pas une question, présentez vos hypothèses et continuez à répondre.
7. Answers in English are accepted.
8. Il ne sera pas permis de sortir de la salle dans les 10 dernières minutes de l'examen. À la fin de l'examen, les surveillants vont ramasser vos copies et vous devrez demeurer assis pendant ce temps.

1 (12)	2 (10)	3 (15)	4 (15)	5 (10)	6 (15)	7 (25)	Tot. (100)

Question 1A) (4 points)

Pour cette question, utilisez le format suivant (**format algorithme**) pour les expressions Booléennes.

- Opérateurs de comparaison: <, >, =, ≤, ≥, et ≠
- Opérateurs booléens: NON, ET, OU
- Opérateurs arithmétique: +, -, *, /, et MOD (modulo)
- Noms de variables et de constantes.

Utilisez des parenthèses lorsque nécessaire. **NE PAS** utiliser la syntaxe Java!

Les classifications d'édifices sont utilisées pour définir des normes structurels, d'incendie et de sécurité. La variable *type* donne le type de base d'un édifice et peut être affectée une des valeurs constantes suivantes : BANQUE, ÉTABLE, ENTREPÔT_CHIMIQUE, ÉGLISE, CLINIQUE, CONFÉRENCE, GARDERIE, DORTOIRES, ÉCOLE_PRIMAIRE, CASERNE_POMPIERS, GARAGE, SOIN_SANTÉ, ÉCOLE_SECONDAIRE, HÔPITAL, HÔTEL, SALLE_CONFÉRENCE, BIBLIOTHÈQUE, MUSÉE, RÉUNION, FOYER_SOINS, BUREAU, PEINTURE, ASSEMBLAGE_PIECES, ENTREPÔT_PETROL, STATION_POLICE, BUREAU_POSTE, RESTAURANT, MAGASIN_DÉTAILS, MAISON_CHAMBRES, STADE, ESPACE_ENTREPÔT, PAVILLON_UNIVERSITAIRE, ENTREPÔT, TRAVAIL_BOIS.

La classification d'un édifice dépend de son *type* tel que démontré dans la table suivante :

Classification d'édifice	Description de clafficiation	Types d'édifice
GROUPA	Assembly Occupancies	CONFÉRENCE, ÉGLISE, SALLE_CONFÉRENCE, BIBLIOTHÈQUE, RÉUNION, MUSÉE, RESTAURANT, STADE, PAVILLON_UNIVERSITAIRE
GROUPB	Business Occupancies	BANQUE, CLINIQUE, CASERNE_POMPIERS , BUREAU, STATION_POLICE, BUREAU_POSTE
GROUPE	Educational	GARDERIE, ÉCOLE_PRIMAIRE, ÉCOLE_SECONDAIRE
GROUPF	Factory and Industrial	PEINTURE, ASSEMBLAGE_PIECES, TRAVAIL_BOIS
GROUPH	Hazardous Occupancies	ENTREPÔT_CHIMIQUE, ENTREPÔT_PETROL
GROUPI	Institutional Occupancies	SOIN_SANTÉ, HÔPITAL, FOYER_SOINS
GROUPM	Mercantile	MAGASIN_DÉTAILS
GROUPR	Residential Occupancies	DORTOIRES, HÔTEL, MAISON_CHAMBRES
GROUPS	Storage	ESPACE_ENTREPÔT, ENTREPÔT
GROUPU	Utility Occupancies	GARAGE, ÉTABLE

Donnez une expression booleénne **d'algorithme** qui est VRAI quand l'édifice est classifié comme GROUPF, ou GROUPH, ou GROUPS, ou GROUPU.

(type=PEINTURE OU type=ASSEMBLAGE_PIECES OU type=TRAVAIL_BOIS) OU
(type=ENTREPÔT_CHIMIQUE OU type=ENTREPÔT_PETROL) OU
(type=ESPACE_ENTREPÔT OU type=ENTREPÔT) OU
(type=GARAGE OU type=ÉTABLE)

Question 1B) (6 points)

Examinez les classes suivantes:

```
class Avion
{
  private int ID;
  private double X;
  private double Y;
  private static int nombreAvions;

  public double calculeDistance()
  { …   }
  public static void getNombreAvions()
  {¨ …   }
}
```

```
class Zone
{
  private String ID;
  public static double limite;
  private Avion[] avions =
                  new Avion[10];
  planes = new Avion[10]

  public Zone () { }
  public Zone (double l)
  { …   }
  public void vérifierDistances()
  { …   }
}
```

Supposons que les instructions suivantes sont utilisées dans la méthode `main()` de la classe `Test`. Chaque cas doit être considéré indépendamment – comme si les instructions de chaque cas sont exécutées dans des méthodes `main()` différentes. Encerclez la lettre de TOUS les cas qui CAUSERONT une erreur de compilation et donnez une courte description de la cause de l'erreur.

a) `Avion.nombreAvions = 5+2;`

// nombreAvions est une variable privée

b) `Zone z = new Zone (300, "Zone1");`

// Le constructeur avec 2 paramètres n'existe pas.

c)
```
Zone ref1 = new Zone();
ref1.limite = 1.25;
```

d) `Avion.getNombreAvions();`

e)
```
Zone zone1;
Avion avion1 = new Avion();
zone1.avions[0] = avion1;
```

// La variable de référence avions est privée dans la classe Zone.

f)
```
Zone instance = new Zone(800);
Zone.vérifierDistances();
```

// vérifierDistances est une méthode d'instance

Question 2: (10 marks)

Sur la page suivante, montrez comment le contenu de la mémoire de travail et la mémoire globale change à l'exécution du programme. Assurez-vous de montrer toutes les valeurs affectés aux variables. Le suivant est un exemple de comment indiqué le changement des valeurs dans une variable.

Variable	~~2~~, ~~6~~, ~~4~~, 10

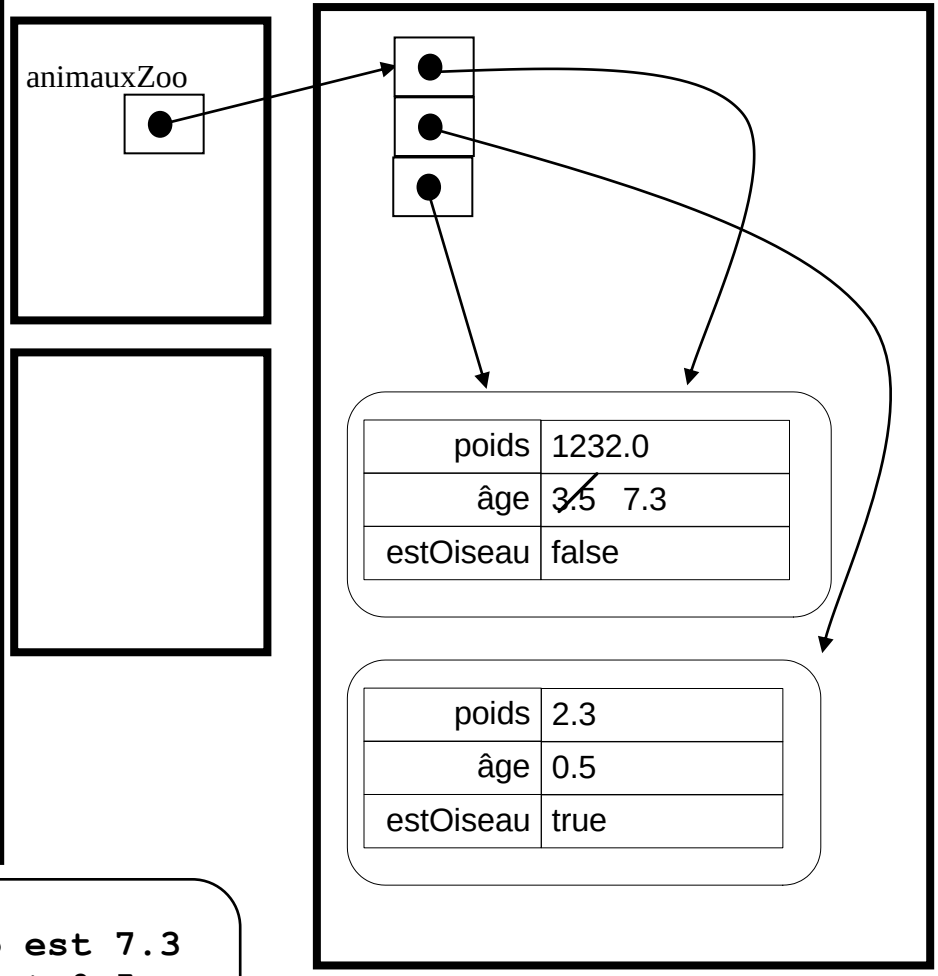
Montrez aussi ce que le programme afficher dans la console.

Mémoire de programme **Mémoire de travail** **Mémoire globale**

```
class Question2
{
  public static void main(String args[])
  {
    Animal[] animauxZoo;
    animauxZoo = new Animal[3];
    animauxZoo[0] = new Animal(1232.0, 3.5, false);
    animauxZoo[1] = new Animal(2.3, 0.5, true);
    animauxZoo[2] = animauxZoo[0];
    animauxZoo[2].âge = 7.3;
    System.out.println("L'âge de l'animal zéro est"
                                        + animauxZoo[0].âge);
    System.out.println("L'âge de l'animal un est "+
                                          animauxZoo[1].âge);
    System.out.println("L'âge de l'animal deux est "+
                                          animauxZoo[2].âge);
  }
}

class Animal
{
  public double poids;
  public double âge;
  public boolean estOiseau;

  public Animal(double poids, double années, boolean oiseau)
  {
    this.poids = poids;
    âge = années;
    estOiseau = oiseau;
  }
```



Console:

```
L'âge de l'animal zéro est 7.3
L'âge de l'animal un est 0.5
L'âge de l'animal deux est 7.3
```

Question 3: (15 marks)

L'algorithme dans l'annexe imprime une pyramide avec le format suivant:

```
                                1
                            1   2   1
                        1   2   4   2   1
                    1   2   4   8   4   2   1
                1   2   4   8  16   8   4   2   1
            1   2   4   8  16  32  16   8   4   2   1
        1   2   4   8  16  32  64  32  16   8   4   2   1
    1   2   4   8  16  32  64 128  64  32  16   8   4   2   1
```

L'algorithme prend un paramètre, *n*, qui donne le nombre de lignes à imprimer dans le pyramide. Sa valeur devra être plus petit ou égale à 10. Traduisez l'algorithme dans l'annexe à une **méthode** Java. La méthode Java `Math.pow(double a, double b)` donne le résultat a^b. L'algorithme suivant se fait appeler :

ImprimeNbr(nbr) imprime le nombre dans un champs de 4 caractères à l'écran, c'est-à-dire, précède le nombre avec assez d'espaces pour imprimer une chaîne de 4 caractères. Traduisez l'appel à la méthode Java *System.out.printf("%4d",nbr)* où *nbr* correspond à l'argument de *ImprimeNbr*.

Question 3 – suite

```
    public static void pyramid(int n)
    {
      // Declare variables
      int line;
      int i;
      int two2i;

      if(n>0  &&  n<11)
      {
        line=1;
        while(line <= n)
        {
          // Prints blank spaces at the start of the line
          System.out.print(" ");
          i=0;
          while(i<n-line)
          {
            System.out.print("    ");
            i=i+1;
          }
          // Print first half of the numbers
          i=0;
          while(i<line)
          {
            two2i = (int) Math.pow(2,i);
            System.out.printf("%4d", two2i);
            i=i+1;
          }
          // Print second half of the numbers
          i=line-2;
          while(i >= 0)
          {
            two2i = (int) Math.pow(2,i);
            System.out.printf("%4d", two2i);
            i=i-1;
          }
          System.out.println();
          line=line+1;
        }
      }
      else
      {
        System.out.println("n is invalid");
      }
    }
```

Question 4: (15 points)

Une matrice N x N est « **triangulaire supérieure** » lorsque tous les éléments au-dessous de la diagonal du coin gauche supérieur au coin droit inférieur sont zéros. La matrice 3 x 3 ci-dessous est un exemple d'une matrice triangulaire supérieure.

$$\mathbf{A} = \begin{bmatrix} x & x & x \\ 0 & x & x \\ 0 & 0 & x \end{bmatrix}$$

Développez une méthode Java qui reçoit la référence d'une matrice carrée (de n'importe quelle dimension) et donne le résultat `true` si la matrice est triangulaire supérieure et `false` autrement. Votre méthode doit être le plus efficace possible.

```
public static boolean estTriangulaireSupérieure( int[][] a)
{
   boolean estTriangulaireSupérieure = true;
   int i, j;

   for (i = 1; i < a.length && estTriangulaireSupérieure; i++)
   {
      for (j = 0; j < i && estTriangulaireSupérieure; j++)
      {
           if (a[i][j] != 0)
           {
                estTriangulaireSupérieure = false;
           }
           else
           {
                /* rien faire */;
           }
      }
   }
   return (estTriangulaireSupérieure);
}
```

Question 5: (10 marks)

Écrivez une méthode récursive, *dip(n)*, qui produit la sortie suivante quand *n*=5:

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

(Notez que vous pouvez utiliser une boucle pour imprimer une ligne individuelle. Jusqu'à 3 points bonis sera affecté si vous utilisez une autre méthode pour imprimer les lignes, c'est-à-dire, que votre solution n'utilise aucune boucle.)

L'en-tête de la méthod aura le format suivant:

```
public static void dip(int n)
{
  int i;  // number of the term

  dipline(n);

  if(n == 1) // Base case
  {
    // do nothing
  }
  else
  {
    dip(n-1);
  }

  dipline(n);
}

public static void dipline(int n)
{
  if(n == 0)
  {
    System.out.println();
  }
  else
  {
    System.out.print("*");
    dipline(n-1);
  }
}
```

Question 6: (15 marks)

La comparaison des prêts avec différents taux d'intérêt :

Développez un algorithme avec les DONNÉES qui recoivent un montant de prêt en dollars et une période de prêt en nombre d'années. L'algorithme affichera les paiements mensuels et le montant total

payé pour chaque taux d'intérêt de 5% à 8% avec incrémentation de 1/8%. Par exemple, pour un montant de prêt de 10,000 pour une période de 5 ans, l'algorithme devra afficher le suivant :

Montant du prêt: 10000
Nombre d'années: 5

Taux d'intérêt	Paiement mensuel	Total payé
5%	188.71	11322.74
5.125%	189.28	11357.13
5.25%	189.85	11391.59
…		
7.85%	202.16	12129.97
8.0%	202.76	12165.83

Le paiement mensuel, *m*, est calculé avec la formule suivante:

$$m = \frac{p * r * q}{12 * (q - 1)}$$

ou

r est le taux d'intérêt en forme décimale (ex. r=0.05 pour un taux de 5%)
p et le montant principal du prêt, i.e. le montant prêté (dans l'exemple ci-dessus p=10000)
q est calculé comme $\left(1 + \frac{r}{12}\right)^n$
n la période du prêt en mois (par exemple pour 20 ans, n = 240 mois)

[Vous n'avez pas 'a produire un algorithme qui aligne les colonnes de la sortie. Vous avez 2 autres pages pour répondre à la question]

Question 6 suite

GIVENS:
- **p (principle amount of the loan)**
- **nYears (loan period in years)**

RESULT:
- **none**

INTERMEDIATES:
- **nMonths (loan period in months)**
- **r (interest rate)**
- **q (q factor as defined)**
- **m (monthly payment)**

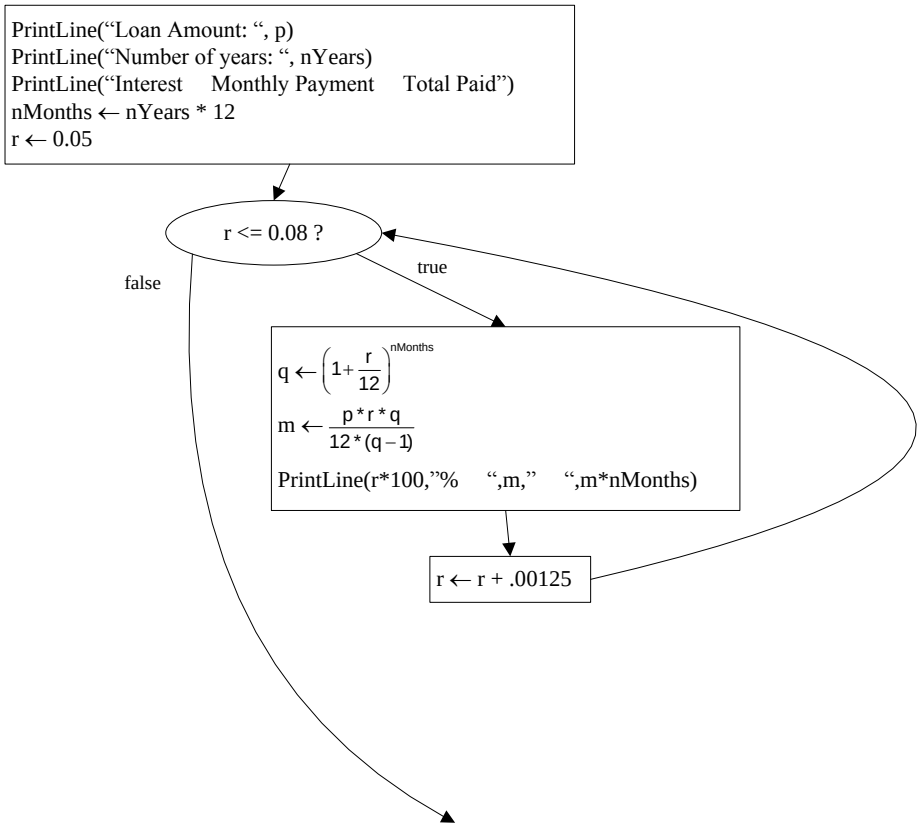
ASSUMPTIONS:

Monthly payment, m, is computed as: $m = \frac{p * r * q}{12 * (q - 1)}$ **, where q=** $\left(1 + \frac{r}{12}\right)^{nMonths}$

Total payment is computed as m * nMonths

HEADER: loanTable(p, nYears)

BODY:



Question 7 (25 marks)

La classe `InventairePièces` sera conçu pour contenir plusieurs objets `Pièce`.

La classe `Pièce` déjà implémentée et vous est fournie (voir l'annexe). Cette classe comprend les attributs : une description de pièce (*description*) , le numéro de pièce (*numéroPièce*), coût de la pièce (*coût*), et le nombre en stock dans l'inventaire (*nbrStock*). La classe offre deux constructeurs, quatre méthodes accesseurs, et deux méthodes modificateurs. Le premier constructeur affecte la valeur 0 à *coût*. Les deux constructeurs affectent la valeur 0 à *nbrStock*. Les méthodes *setCoût* et *setNbrStock* sont utilisées pour mettre à jour les valeurs des attributs *coût* et *nbrStock* respectivement. Le diagramme de classe UML pour la classe *Pièce* est:

Pièce
- description: String - numéroPièce: String - coût: double - nbrStock: int
+Pièce(laDescription: String, numPièce: String) +Pièce(laDescription: String, numPièce: String, leCoût: double) +getDescription(): String +getNuméroPièce(): String +getCoût(): double +setCôut(coût:double) +getNbrStock(): int +setNbrStock(nbr: int)

Sur les deux prochaines pages, vous devez compléter les méthodes pour la classe `InventairePièces`. Votre classe `InventairePièces` devra fournir cinq méthodes publiques (définissez un constructeur si nécessaire) qui permettra la classe `TestInventairePièces` d'exécuter :

```
class TestInventairePièces
{
   public static void main (String[] args)
   {
      InventairePiecès inventory = new InventairePièces( );
      inventaire.ajouterPièce( "Dakota Starter, 3.9L, 5.2L, 5.9L", "F5000-137756");
      inventaire.ajouterPièce( "Dakota Water Pump, 3.9L", "AWAW7160");
      inventaire.ajouterPièce( "Dakota Alternator 136 Amp", "MG1338", 399.95);
      inventaire.ajouterPièce( "Dakota Master Brake Cylindre", "CE131.67025", 124.94);
      inventaire.ajouterPièce( "Dakota Roll Up Tonneau Cover", "A7414079");
      inventaire.changerCoût("Dakota Starter, 3.9L, 5.2L, 5.9L",169.95);
      inventaire.changerCoût("MG1338",78.95);
      inventaire.changerNbrStock("Invalid",5);  // Notez le String invalid
      inventaire.changerNbrStock("AWAW7160",5);
      inventaire.changerNbrStock("F5000-137756",3);
      inventaire.changerNbrStock("A7414079",1);
      inventaire.imprimeInventaire();
   }
}
```

L'exécution de `main()` imprime le suivant à l'écran :

```
Pièce Invalid n'existe pas.
Inventaire:
   Dakota Starter, 3.9L, 5.2L, 5.9L; F5000-137756; 169.95; 3
   Dakota Water Pump, 3.9L; AWAW7160; 0.0; 5
   Dakota Alternator 136 Amp; MG1338; 78.95; 0
   Dakota Master Brake Cylindre; CE131.67025; 124.94; 0
   Dakota Roll Up Tonneau Cover; A7414079; 0.0; 1
```

Question 7:(suite)

Complétez la classe `InventairePièces` sur cette page et sur les prochaines 2 pages.

```
class InventairePièces
{
  // Déclarations d'attributs: (3 points)

  Part [] partList;

  // Constructeur (si nécéssaire): (2 points)

  // Not required

 // MÉTHODE MODIFICATEUR ajouterPièce: (8 points)
 // Paramètres de méthodes: String description - une description
 //                         String numéroPièce - un numéro de pièce
 // Il est possible d'appeler cette méthode avec un paramètre optionnel
 //         double coût – le coût de la pièce
 // La méthode ajoute un objet Pièce à l'object InventairePièces.
 // it is possible that the method will be called with an optional

  public void ajouterPièce(String description, String numéroPièce)
  {
    Part newPart;
    Part [] newPartList;
    int i;

    newPart = new Part(newDescription, newPartNum);

    if(partList == null)
    {
       partList = new Part[1];  // First creation of list
       partList[0] = newPart;
    }
    else
    {
       newPartList = new Part[partList.length+1];  // Creates new list
       for(i=0 ; i<partList.length ; i=i+1)
       {
          newPartList[i] = partList[i];
       }
       newPartList[newPartList.length-1] = newPart;
       partList = newPartList;
    }
  }

  public void ajouterPièce(String description, String numéroPièce, double coût)
  {
      ajouterPièce(newDescription, newPartNum);
      updatePartCost(newPartNum,newCost);
  }
```

```
 // MÉTHODE imprimeInventaire: (4 points)
 // Paramètres: (aucune)
 // Résultats: (aucun)
 // Cette méthode imprime une liste de tous les pièces dans l'inventaire (
 // y compris tout information au sujet des pièces)  Voyez la sortie exemple
 // de TestInventairePièces sur page 12 pour un format exacte de la sortie.

  public void imprimeInventaire()
  {
    int i;
    System.out.println("Inventory:");
    for(i=0 ; i< partList.length ; i=i+1)
    {
       System.out.println("   "+partList[i].getDescription()+"; " +
                          partList[i].getPartNumber() + "; " +
partList[i].getCost()
                          + "; "+partList[i].getStockAmount());
    }
  }

  // MÉTHODE changerCoût: (4 points)
  // Mets à jour l'attribut côut de la première Pièce ayant la description ou
  // numéro de pièce donné.  Si la pièce n'est pas trouvée, un message d'erreur
  // est affiché à l'écran (voyez la page 12 pour le format exact du message).
  // Rappelez vous que deux chaîne (i.e. objets String) peuvent être comparés pour
  // égalité avec « s1.equals(s2) » (s1 et s2 réfère aux objets String).
  // Paramètres :  descOuNumPièce – soit la description ou numéro de pièce.
  //               nouveauCoût – nouveau coût de la pièce.

   public void changerCoût(String descOuNumPièce, double nouveauCoût)
  {
    int i;
    boolean foundPart = false;

    i=0;
    while((i< partList.length) && !foundPart)
    {
       if(descOrPartNum.equals(partList[i].getDescription()) ||
          descOrPartNum.equals(partList[i].getPartNumber()) )
       {
          partList[i].setCost(newCost);
          foundPart = true;
       }
       else {/* do nothing */ }
       i = i + 1;
    }
    if(!foundPart) System.out.println("Part "+descOrPartNum+" does not exist.");
  }
```

```
  // MÉTHODE changerNbrStock: (4 points)
  // Met à jour l'attribut nbrStock de la première Piéce ayant la description
  // ou numéro de pièce spécifié. Si la pièce n'est pas trouvée, un message
  // d'erreur est affiché à l'écran (voyez la page 12 pour le format exact du
  // message).Rappelez vous que deux chaîne (i.e. objets String) peuvent être
  // comparés pour égalité avec « s1.equals(s2) » (s1 et s2 réfère aux objets
  // String).
  // Paramètres :  descOuNumPièce – soit la description ou numéro de pièce.
  //               nouveauNbrStock – nouveau nombre en stock pour la pièce.

  public void changerNbrStock(String descOuNumPièce, int nouveauNbrStock)
  {
    int i;
    boolean foundPart = false;

    i=0;
    while((i< partList.length) && !foundPart)
    {
       if(descOrPartNum.equals(partList[i].getDescription()) ||
          descOrPartNum.equals(partList[i].getPartNumber()) )
       {
          partList[i].setStockAmount(newStockAmount);
          foundPart = true;
       }
       else {/* do nothing */ }
       i = i + 1;
    }
    if(!foundPart) System.out.println("Part "+descOrPartNum+" does not exist.");
  }

} // Fin de la classe InventairePièces
```