UQÀM
Université du Québec à Montréal

INF1120 Programmation I

Automne 2009

Examen final

19 décembre 2009

Solutions

#1	_____	/ 15
#2	_____	/ 15
#3	_____	/ 15
#4	_____	/ 15
#5	_____	/ 15
#6	_____	/ 25
TOTAL	_____	/ 100

COMMENTAIRES

Aide-mémoire pour l'examen final

Quelques méthodes publiques de la classe `String` :

char charAt(**int** indice)
Retourne le caractère à l'indice spécifié.

int indexOf(**int** ch)
Retourne l'indice à l'intérieur de cette chaîne de la première occurrence du caractère ch.

int indexOf(**int** ch, **int** fromIndex)
Retourne l'indice à l'intérieur de cette chaîne de la première occurrence du caractère ch à partir de l'indice fromIndex.

int lastIndexOf(**int** ch)
Retourne l'indice à l'intérieur de cette chaîne de la dernière occurrence du caractère ch.

int length()
Retourne la longueur de cette chaîne.

String replace(**char** oldChar, **char** newChar)
Retourne une nouvelle chaîne résultant du remplacement de toutes les occurrences de oldChar par newChar.

String substring(**int** beginIndex, **int** endIndex)
Retourne une nouvelle chaîne qui est une sous-chaîne de cette chaîne constituée des caractères de chaîne de la position beginIndex incluse et finissant à la position endIndex exclue. La sous-chaîne contient donc endIndex - beginIndex caractères.

String substring(**int** beginIndex)
Retourne une nouvelle chaîne qui est une sous-chaîne de cette chaîne constituée des caractères de chaîne à partir de la position beginIndex jusqu'à la fin.

String toLowerCase()
Retourne une nouvelle chaîne constituée des caractères de cette chaîne, chacun converti en minuscule s'il s'agit d'une lettre majuscule.

String toUpperCase()
Retourne une nouvelle chaîne constituée des caractères de cette chaîne, chacun converti en majuscule s'il s'agit d'une lettre minuscule.

String trim()
Retourne une copie de la chaîne à laquelle on a enlevé les blancs au début et les blancs à la fin.

static String valueOf(**int** i)
Retourne la chaîne correspondant l'entier i.

Méthode publique de la classe `Integer` :

```
//Transforme la chaîne de caractères s (qui représente un entier) en entier (int) et
//retourne cet entier.  Si la chaîne s ne représente pas un nombre entier, lance une
//exception NumberFormatException.
public static int parseInt(String s) throws NumberFormatException
```

Méthode publique de la classe `Double` :

```
//Transforme la chaîne de caractères s (qui représente un nombre réel) en nombre réel
//(double) et retourne ce nombre réel.  Si la chaîne s ne représente pas un nombre réel,
//lance une exception NumberFormatException.
public static double parseDouble(String s) throws NumberFormatException
```

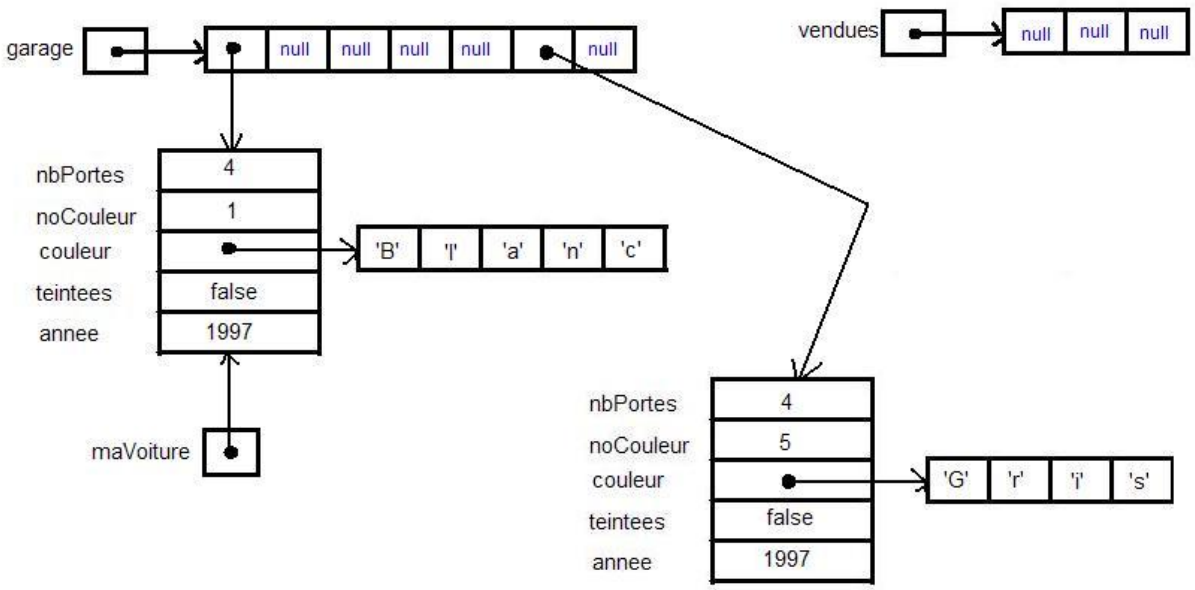
Question 1 (15 points)

Considérez les deux classes suivantes et répondez aux directives énoncées dans les deux commentaires en gris de la classe `TestVoiture`.

```
public class Voiture {

    //Constantes de classe
    public static int PORTES_MIN = 2;
    public static int PORTES_MAX = 4;
    public static int ANNEE_MIN = 1960;
    public static int ANNEE_MAX = 2009;
    public static int AN_DEFAUT = 1997;
    public static int COULEUR_DEFAUT = 5;
    public static String []  COULEURS =
                          {"Argent", "Blanc", "Bleu foncé", "Bleu pâle","Jaune", "Gris", "Noir",
                           "Rouge", "Vert foncé", "Vert pâle"};
    // constructeurs
    public Voiture (){
    }

    public Voiture(int p, int c, boolean t, int a){
        setNbPortes(p);
        setCouleur(c);
        teintees = t;
        setAnnee(a);
    }

    //Méthodes d'instance
    public void inverserTeintees(){
        teintees = !teintees;
    }

    public Voiture faireUneCopie(){
        return new Voiture(nbPortes, noCouleur, teintees, annee);
    }

    //Modificateurs
    public void setNbPortes(int n){
        if (n == PORTES_MIN || n == PORTES_MAX) {
            nbPortes = n;
        }
    }

    public void setCouleur(int c){
         if (!(c >= COULEURS.length)) {
            couleur = COULEURS[c];
            noCouleur = c;
        }
    }

     public void setAnnee(int a){
        if (a >= ANNEE_MIN && a <= ANNEE_MAX){
            annee = a;
        }
    }


    //Variables d'instance
    private int nbPortes = PORTES_MAX;
    private int noCouleur = COULEUR_DEFAUT;
    private String couleur = COULEURS[noCouleur];
    private boolean teintees = false;
    private int annee = AN_DEFAUT;

} // Voiture
```

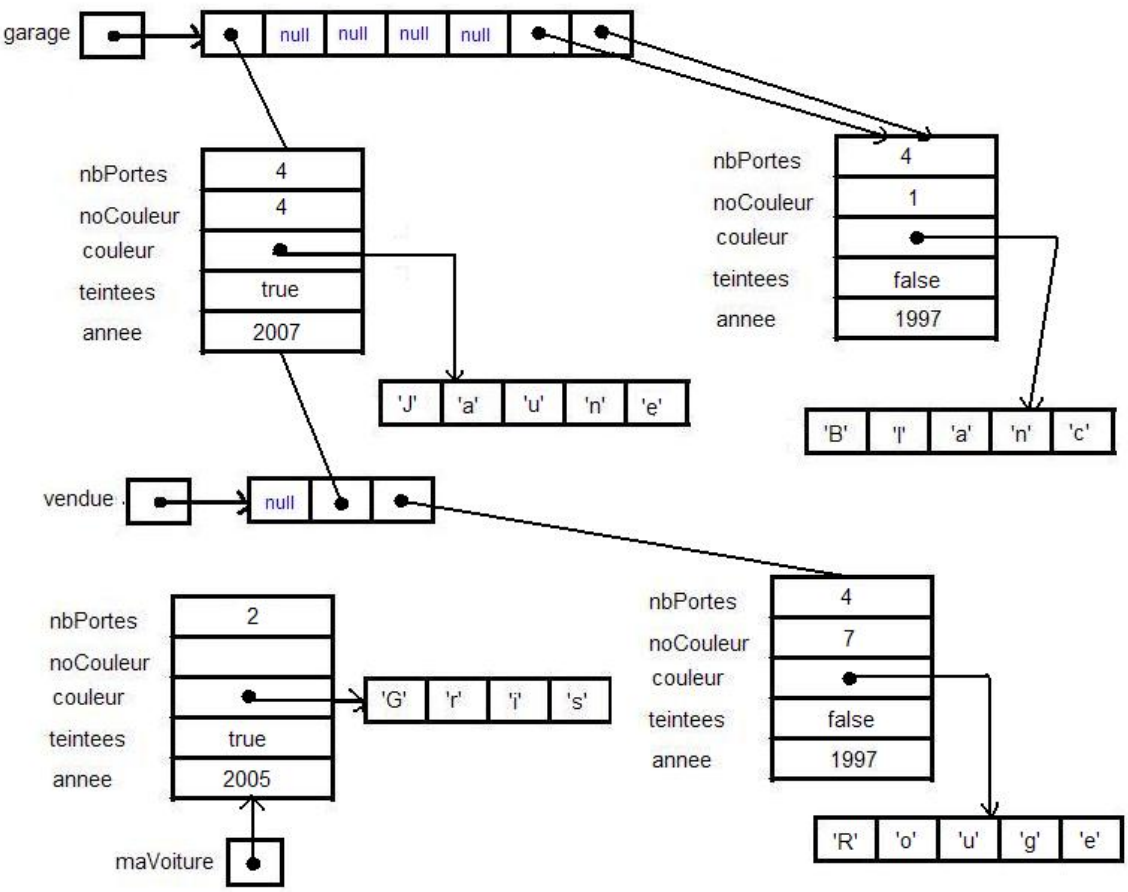
```
public class TestVoiture {

    // Autres méthodes s'il y a lieu

    public static void main (String[] params) {

        Voiture [] garage = new Voiture [7];
        Voiture maVoiture = new Voiture(3, 1, false, Voiture.AN_DEFAUT);
        Voiture [] vendues = new Voiture [3];

        garage[0] = maVoiture;
        garage[5] = new Voiture();

      // 1A - dessinez l'état de la mémoire
      // suite aux intructions précédentes...

        vendues[1] = garage[5];
        vendues[1].inverserTeintees();
        garage[6] = garage[0].faireUneCopie();
        garage[0].setCouleur(7);
        garage[0] = garage[5];
        vendues [2] = maVoiture;
        vendues[1].setCouleur(4);
        maVoiture = new Voiture(2, 10, true, 2005);
        garage[5] = garage[6];
        garage[0].setAnnee(2007);


      // 1B - dessinez l'état de la mémoire ici
      // suite à l'exécution complète des 10
      // instructions précédentes

    } // main

} // TestVoiture
```

Réponse 1 a)



(espace supplémentaire page suivante)

Réponse 1B


garage
null
null
null
null
nbPortes
noCouleur
couleur
teintees
annee
4
4
true
2007
'J'
'a'
'u'
'n'
'e'
nbPortes
noCouleur
couleur
teintees
annee
4
1
false
1997
'B'
'l'
'a'
'n'
'c'
vendue
null
nbPortes
noCouleur
couleur
teintees
annee
2
true
2005
'G'
'r'
'i'
's'
maVoiture
nbPortes
noCouleur
couleur
teintees
annee
4
7
false
1997
'R'
'o'
'u'
'g'
'e'

Question 2 (15 points)

Écrivez une méthode dont le nom sera `comprimer` qui prend en paramètre une chaîne de caractères (`String`) et qui retourne une nouvelle chaîne dans laquelle on retrouvera tous les caractères de celle passée en paramètre, dans le même ordre, sauf les espaces. Si la chaîne est `null`, le résultat retourné doit être `null`. Voici des exemples

Paramètre	Résultat
"Bonjour"	"Bonjour"
"1 2 3 4 5 6 toto "	"123456toto"
" x "	"x"
" "	""

```
public static
                  // à compléter (y compris l'entête)
```

Solutions possibles parmi plusieurs :

```
public static String comprimer ( String s ) {
    String reponse = null;
    if ( s != null ) {
        reponse = "";
        s = s.trim(); // enlever les espaces de début et fin
        int i = s.indexOf ( ' ' );  // pos du premier espace dans s
        while ( i >= 0 ) {
            reponse = reponse + s.substring ( 0, i ); // coller le morceau avant l'espace
            s = s.substring ( i + 1 ); // se débarrasser du morceau et de l'espace
            i = s.indexOf ( ' ' ); // pos du prochain espace
        }
    }
    return reponse + s;
}
```

```
public static String comprimer ( String s ) {
    String reponse = null;
    char c;
    if ( s != null ) {
        reponse = "";
        for ( int i = 0; i < s.length(); i++ ) {
            c = s.charAt(i);
            if ( c != ' ' ) {
                reponse = reponse + c;
            }
        }
    }
    return reponse;
```

Question 3 (15 points)

Considérez le programme suivant :

```
public class Max {

    public static int saisirNbElements() {
        int n;

        System.out.println("Veuillez fournir le nombre d'éléments : ");
        n = Clavier.lireInt();
        return n;
    }

    public static int trouverMaximum (int n) {
        int max = Integer.MIN_VALUE;   // Le plus petit entier = -2147483648
        int nombreLu;

        System.out.println("Veuillez fournir " + n + " nombres entiers.");

        for (int i = 0; i < n; i++){
            nombreLu = Clavier.lireInt();
            if ( nombreLu > max ) {
                max = nombreLu;
            }
        }
        return max;
    }

    public static void main (String[] params) {

        int nbElements;
        int leMaximum;

        nbElements = saisirNbElements();
        leMaximum = trouverMaximum ( nbElements );

        System.out.println ( "Le maximum des " + nbElements + " nombres est " + leMaximum +
"." );    } // main
}
```

(a) Écrivez une nouvelle version de la méthode `saisirNbElements()` dans laquelle une exception est levée et transmise à la méthode qui l'a appelée. Cette exception doit être une exception contrôlée (doit dériver de la classe `Exception`). L'exception en question doit être levée lorsque le nombre lu est strictement inférieur à 2. Si vous avez besoin de déclarer une nouvelle classe, indiquez-le clairement en donnant le code de celle-ci.

Solution :

```
public static int saisirNbElements() throws plusPetitQue2Exception {
    int n;


    System.out.println("Veuillez fournir le nombre d'éléments : ");
    n = Clavier.lireIntLn();
    if ( n < 2 ) {
        throw new plusPetitQue2Exception ();
    }
    return n;
}
class plusPetitQue2Exception extends Exception {
}
```

(b) Écrivez une nouvelle version de la méthode `main`. Dans cette nouvelle version, vous devez intégrer un gestionnaire d'exceptions (try-catch). Ce gestionnaire doit traiter les exceptions que les deux autres méthodes sont susceptibles de lever. Pour chaque exception, on doit faire afficher un message d'erreur approprié. On ne doit pas ajouter la clause "`throws ...`" dans l'entête de la méthode `main`.

Solution :

```
public static void main (String[] params) {


    int nbElements;
    int leMaximum;


    try {
        nbElements = saisirNbElements();
        leMaximum = trouverMaximum ( nbElements );


        System.out.println ( "Le maximum des " + nbElements + " nombres est "
                             + leMaximum + "." );
    } catch ( NumberFormatException e ) {
        System.out.println ( "La valeur entrée n'est pas numérique." );
    } catch ( plusPetitQue2Exception e ) {
        System.out.println ( "La valeur saisie doit être supérieure à 1." );
    }
} // main
```

Question 4 (15 points)

Qu'affiche la méthode `main` ?

```
public class Question4 {

    public static void mure (char[] tab, String fruit, int graine) {
        tab[0] = Character.toLowerCase (fruit.charAt(0));
        tab[1] = fruit.charAt(tab.length -1);
        tab[2] = "Mango".charAt(3);
        tab[3] = fruit.charAt(graine);
    } // mure

    public static void sucre (char[]tab) {
        char c;
        for (int i = 0; i < tab.length - 2; i++) {
            c = tab[i];
            tab[i] = tab[i+2];
            tab[i+2] = c;
        }
    } // sucre

    public static void sure (char[] tab, int ind1, int ind2) {
        if ( ind1 < ind2 ) {
            tab[ind1] = tab[ind2];
            tab[0] = tab[ind1-1];
        } else {
            tab[ind2] = 'O';
            tab[ind1-ind2] = 'P';
        }
    } // sure

    public static char[]deguster (char[] tab) {
        char[] avocat = { 'A', 'V', 'O', 'C' };
        for (int i = avocat.length - 1; i > 0; i--) {
            if (i < tab.length - 2) {
                avocat[i] = tab[i+2];
            } else {
                avocat[i] = 'T';
            }
        } // for
        return avocat;
    } // deguster

    public static void mangerFruit (char[] tab) {
        for (int i = 0; i < tab.length; i++) {
            System.out.print(tab[i]);
        }
        System.out.println();
    } // mangerFruit

    public static void main (String[] arg) {
        char[] panier = new char[4];
        char[] panier2;
        mure (panier, "Raisins", 3);
        mangerFruit (panier);
        sucre (panier);
        mangerFruit (panier);
        sure (panier, 2, 3);
        mangerFruit (panier);
        panier2 = deguster (panier);
        mangerFruit (panier2);

    } // main

} // Question4
```

Réponses :

rsgs

gsrs

ssss

AsTT

Question 5 (15 points)

La banque BP (Banque des Patrons) dispose d'un ensemble d'informations sur ses employés patrons regroupées dans un fichier nommé `employesPatronsSalaires.txt`. Chaque employé patron y est décrit par 4 informations. Chacune de ces informations est écrite sur une ligne différente, dans cet ordre : Le matricule, le nom, le salaire et le nombre d'années d'ancienneté. Ci-dessous l'exemple d'un fragment du fichier `employesPatronsSalaires.txt` contenant 3 employés patrons (donc 12 lignes).

```
RICD147681
Doma Riche
648123.50
2
KALJ103466
John Kalekas
869123.35
4
PROD147682
Daniel Propescu
1049231.30
5
```

Dans le souci de passer à travers la grande crise de 2008-2009 sans faire faillite, la BP décide de diminuer les salaires de ses employés patrons. Les intervalles de salaire et les taux de diminution à appliquer sont mentionnés dans le tableau ci-dessous.

Intervalle de salaire ($)	Taux de diminution de salaire (%)
> 500 000 et < = 750 000	5.5
> 750 000 et < = 1 000 000	7.5
> 1 000 000	10.5

Il s'agit d'écrire un programme (une méthode `main`) qui va lire le fichier `employesPatronsSalaires.txt` et créera un fichier `employesPatronsSalairesRevises.txt`. Dans le nouveau fichier créé, vous devez écrire toutes les informations de chaque employé patron plus le taux de diminution du salaire appliqué, le salaire après diminution. Les informations de chaque employé doivent être affichées sur une seule ligne avec une barre (|) comme séparateur selon l'ordre suivant : le matricule, le nom, le salaire avant diminution, le nombre d'années d'ancienneté, le taux de diminution du salaire appliqué, le salaire après diminution. Selon le fragment du fichier `employesPatronsSalaires.txt` et le tableau des règles et taux montrés ci-dessus, votre programme devrait écrire le fichier `employesPatronsSalairesRevises.txt` exactement comme suit :

```
RICD147681 | Doma Riche      | 648123.50  | 2 | 5.5  | 612476.70
KALJ103466 | John Kalekas    | 869123.35  | 4 | 7.5  | 803939.09
PROD147682 | Daniel Propescu | 1049231.30 | 5 | 10.5 | 939062.01
```

Le salaire après diminution se calcule selon la formule suivante :

Salaire après diminution = salaire - (salaire x (taux de diminution du salaire / 100))

Directives supplémentaires :

1. Vous ne devez pas utiliser les tableaux pour ce numéro.
2. Si l'exception `FileNotFoundException` est déclenchée (peu importe où dans votre programme), votre programme doit afficher le message d'erreur « `Impossible de lire le fichier.` » et se termine.
3. Si l'exception `IOException` est déclenchée (peu importe où dans votre programme), votre programme doit afficher le message d'erreur « `Erreur d'entrée/sortie` » et se termine.
4. Si l'exception `NumberFormatException` est déclenchée (peu importe où dans votre programme), votre programme doit afficher le message d'erreur « `Format invalide` » et se termine
5. Si le fichier `employesPatronsSalairesRevises.txt` existe, votre programme doit l'écraser.

(Consultez l'aide-mémoire au début de cet examen pour les méthodes `trim`, `parseInt` et `parseDouble` qui vous seront utiles)

```
public static void main (String [] params) {
    // à compléter
    // noms de fichiers à lire ou écrire
    // Vous devez utiliser ces constantes
    final String NOM_FIC_EMPLOYES_PATRONS_SALAIRES = "employesPatronsSalaires.txt";
    final String NOM_FIC_EMPLOYES_PATRONS_SALAIRES_REVISES = "employesPatronsSalairesRevises.txt";
    //Les flux de lecture ou d'écriture
    FileReader fichierEmployes = null;
    BufferedReader lecteurEmployes = null;
    PrintWriter ecrivainEmployes = null;

    //informations à lire ou à écrire
    String matricule;
    String nom;
    double salaire;
    int anneeAnciennete;
    double salaireApresDiminution;
    double tauxDiminutionSalaire = 0.0;


    try {
        fichierEmployes = new FileReader(NOM_FIC_EMPLOYES_PATRONS_SALAIRES);
        lecteurEmployes = new BufferedReader(fichierEmployes);
        ecrivainEmployes = new PrintWriter(NOM_FIC_EMPLOYES_PATRONS_SALAIRES_REVISES);

        while (lecteurEmployes.ready()) {

            //lecture de employees.txt
            matricule = lecteurEmployes.readLine();
            nom = lecteurEmployes.readLine();
            salaire = Double.parseDouble(lecteurEmployes.readLine().trim());
            anneeAnciennete = Integer.parseInt(lecteurEmployes.readLine().trim());

            //Obtenir le taux de diminution de salaire à appliquer
            if (salaire > 1000000) {
                tauxDiminutionSalaire = 10.5;
            } else if (salaire > 750000) {
                tauxDiminutionSalaire = 7.5;
            } else if (salaire > 500000) {
                tauxDiminutionSalaire = 5.5;
            }

             //Calculer le nouveau salaire
             salaireApresDiminution = salaire - (salaire * (tauxDiminutionSalaire/100));
             //Écrire la ligne dans le nouveau fichier
             ecrivainEmployes.println(matricule + " | " + nom + " | " + salaire
                                                + " | " + anneeAnciennete
                                                + " | " + tauxDiminutionSalaire
                                                + " | " + salaireApresDiminution);

       }

       //liberation des ressources
       lecteurEmployes.close();
       ecrivainEmployes.close();

    //gestionnaire d'exceptions
    } catch (FileNotFoundException e) {
        System.out.println("Erreur, le fichier ne peut pas etre lu.");

    } catch (IOException e) {
        System.out.println("Erreur d'entree / sortie.");

    } catch (NumberFormatException e) {
        System.out.println(" Format invalide.");

    }
} // fin Main
```

Question 6 (25 points)

(A) Vous devez compléter la classe suivante.

N'utilisez pas de String pour les attributs.

```
/**
 * I N F 1 1 2 0 - FINAL AUT 09
 *
 * DESCRIPTION de la classe BonhommeDeNeige.
 *
 *  Cette classe sert à fabriquer des bonhommes des neiges. Il existe 2 grandeurs de
 *  bonhomme : petit et grand. Par défaut, on fabrique un grand bonhomme.
 *
 *  Un bonhomme peut avoir des bras ou non. Par défaut, un bonhomme n'a pas de bras.
 *
 *  Il existe 3 choix de couvre tête :
 *
 *          0 = aucun       1 = chapeau     2 = tuque
 *
 *      Par défaut, un bonhomme a un chapeau. Un bonhomme de neige ne PEUT PAS avoir
 *      un chapeau ET une tuque.
 *
 *  Un bonhomme peut avoir 0 à plusieurs accessoires parmi les suivants :
 *
 *          0 = foulard     1 = mitaines    2 = pipe    3 = moustache
 *          4 = boutons de manteau
 *
 *      Un petit bonhomme ne PEUT PAS avoir de moustache ni de pipe. Un bonhomme
 *      qui n'a pas de bras ne PEUT PAS avoir de mitaine. Par défaut, un bonhomme
 *      n'a pas d'accessoire.
 *
 *
 *  Il existe 5 sortes de nez possible :
 *
 *          0 = carotte     1 = patate      2 = bouton      3 = roche
 *          4 = bille de verre
 *
 *      Par défaut, le nez est une carotte (0).
 *
 *  Il existe 4 sortes d'yeux possibles :
 *
 *          0 = boutons     1 = roches      2 = bille de verre     3 = raisins
 *
 *      Par défaut, les yeux sont faits avec des billes de verre (2).
 *
 * En tout temps, on peut appeler des méthodes pour savoir combien de petits bonhommes
 * de neige ont été fabriqués et combien de bonhommes de neige (peu importe la taille)
 * ont été fabriqués en tout.
 */

public class BonhommeDeNeige {
    final public static String [] ACCESSOIRES = {"foulard", "mitaines", "pipe",
                                    "moustache", "boutons de manteau"};
    final public static String [] COUVRES_TETE = {"Aucun", "chapeau", "tuque"};
    final public static String [] TYPE_NEZ = {"carotte", "patate", "bouton", "roche",
                                    "bille de verre"};
    final public static String [] TYPE_YEUX = {"boutons", "roches", "bille de verres",
                                    "raisins"};
    final public static int NEZ_DEFAUT = 0;
    final public static int YEUX_DEFAUT = 2;
```

```
// constantes supplémentaires
    final private static int INDICE_FOULARD = 0;
    final private static int INDICE_MITAINES = 1;
    final private static int INDICE_PIPE = 2;
    final private static int INDICE_MOUSTACHE = 3;
    final private static int INDICE_BOUTONS = 4;

// variable(s) de classe pour stocker des données servant aux statistiques.
// Voir les méthodes de classe plus loin.
    private static int nbrBonhommes = 0;
    private static int nbrPetits = 0;

// variable(s) d'instance pour stocker les informations pour un bonhomme de neige
    private boolean avecBras = false;
    private boolean grandBonhomme = true;
    private boolean avecChapeau = true;
    private boolean avecTuque = false;
    private boolean [] accessoires = new boolean [ACCESSOIRES.length];
    private int nez = NEZ_DEFAUT;
    private int yeux = YEUX_DEFAUT;
    /*
     *  Sert à construire un grand bonhomme de neige avec les valeurs par défaut.
     *  UTILISEZ LE MOINS D'INSTRUCTIONS POSSIBLES
     */
    public BonhommeDeNeige (){
        nbrBonhommes ++;

        //pas besoin de faire de code supplémentaire à cause des initialisations aux
        // valeurs par défaut dans la partie de la déclaration des variables d'instance.
    } // BonhommeDeNeige
```

```
/*
 *  Sert à créer un bonhomme de neige SANS couvre tête (pas de chapeau ni de tuque)
 *  en fonction des informations reçues en paramètre.
 *
 *  IMPORTANT:
 *  Seul un bonhomme de neige avec des bras (paramètre bras à true) PEUT avoir des
 *  mitaines. Seul un GRAND bonhomme de neige (paramètre grand à true) PEUT avoir une
 *  moustache et/ou une pipe. Il faut donc que le constructeur effectue des
 *  vérifications pour savoir s'il peut attribuer les valeurs des paramètres aux
 *  variables d'instance concernées.
 *
 *  Si les valeurs des paramètres nez et yeux ne sont pas dans l'intervalle des
 *  valeurs permises, la(les) valeur(s) par défaut sera(ont) attribuée(s) aux
 *  variables d'instance concernées.
 */
public BonhommeDeNeige (boolean bras, boolean grand, int nez, int yeux,
                        boolean foulard, boolean mitaines, boolean pipe,
                        boolean moustache, boolean boutonsM){
            this();
            avecBras = bras;
            if(bras){
                accessoires[INDICE_MITAINES] = mitaines;
            }//false par défaut...

            grandBonhomme = grand;
            if (!grand){
                nbrPetits ++;
            }
            if(grand){
                accessoires[INDICE_MOUSTACHE] = moustache;
                accessoires[INDICE_PIPE] = pipe;
            }//false par défaut...
            avecChapeau = false;
            avecTuque = false; //pas obligatoire car false par défaut...
            accessoires[INDICE_BOUTONS] = boutonsM;
            accessoires[INDICE_FOULARD] = foulard;
            if (nez >= 0 && nez < TYPE_NEZ.length){
               this.nez = nez;
            }
            if (yeux >= 0 && yeux < TYPE_YEUX.length){
                this.yeux = yeux;
            }
} // BonhommeDeNeige

/*
 * Retourne le nombre total de bonhommes fabriqués.
 */
 public static int obtenirNombreTotalBonhommes () {

     return nbrBonhommes;
}// obtenirNombreTotalBonhommes
```

```
/**
 * Retourne le nombre total de PETITS bonhommes fabriqués.
 */
 public static int obtenirNombreTotalPetitsBonhommes () {

     return nbrPetits;
 }// obtenirNombreTotalPetitsBonhommes

/*
 * retourne la chaine de caractères qui correspond au type de nez
 * du bonhomme de neige.
 */
public String getTypeNez(){

    return TYPE_NEZ[nez];
} // getTypeNez

/*
 *  NOTE: Il faut s'assurer que ce « setter » respecte la contrainte d'intégrité qui
 *       veut que seul un bonhomme de neige qui a des bras puisse avoir des mitaines
 */
 public void setMitaines (boolean mitaines) {

     if (avecBras){

         accessoires[INDICE_MITAINES] = mitaines;

     } //false par défaut
} // setMitaines


/*
 *  NOTE: Dans le cas où le bonhomme de neige a une tuque,
 *  on doit lui "enlever" avant de lui mettre un chapeau car
 *  un bonhomme de neige ne PEUT PAS avoir un chapeau
 *  ET une tuque en même temps.
 */
 public void setChapeau (boolean c) {

     avecChapeau = c;


     if (c && avecTuque){

         avecTuque = false;

     }
}  // setChapeau
```

```
/*
 *  NOTE: Dans le cas où il s'agit d'un petit bonhomme de neige,
 *  aucune modification n'est effectuée car un petit bonhomme
 *  ne PEUT PAS avoir de pipe.
 */
 public void setPipe (boolean p) {


     if (grandBonhomme){

         accessoires[INDICE_PIPE] = p;

     }
}// setPipe

 /*
  * Retourne la chaîne de caractères contenant tous les attributs du bonhomme
  * de neige. Utilisez les constantes définies au début de la classe.
  */
  public String toString () {

      String temp;

      String listeAccess = "";

      boolean access = false;


      temp = "Taille = ";


      if(grandBonhomme){

          temp = temp + "grand \n";

      }else{

          temp = temp + "petit \n";

      }


      temp = temp + "Bras = ";


      if(avecBras){

          temp = temp + "OUI\n";

      } else {

          temp = temp + "NON\n";

      }


      temp = temp + "Couvre tête = ";


      if(avecChapeau){

          temp = temp + COUVRES_TETE[1];

      } else if (avecTuque){
```

```
            temp = temp + COUVRES_TETE[2];
        } else {
            temp = temp + COUVRES_TETE[0];
        }

        temp = temp + "\nType de nez = " + TYPE_NEZ[nez];
        temp = temp + "\nType d'yeux = " + TYPE_YEUX[yeux];

        for(int i = 0; i < accessoires.length; i++){
            if(accessoires[i]){
                listeAccess = listeAccess +  ACCESSOIRES[i] + "   ";
                access = true;
            }
        }

        if (! access ) {
            listeAccess = "Aucun";
        }

        temp = temp + "\nAccessoires = " + listeAccess;
        return temp;
}//toString
```