Licence Informatique
Université de Bordeaux
2015-2016

UE J1IN4W01
Programmation 2
Travaux Dirigés - feuille 1

1 Travaux Dirigés

Compréhension de code

Exercice 1. Le but de cet exercice est de se familiariser avec la syntaxe Java.

```
public class Monnaie {
  public static void usage() {
    System.err.println("␣Usage␣:␣<somme␣>");
  }

  public static void afficheMonnaie(int[] rep, int[] val) {
    for (int i = 0; i < rep.length; ++i) {
      if (val[i] < 5) {
        System.out.println("Nbre␣de␣pieces␣de␣" + val[i] + "␣euros␣:" + rep[i]);
      } else {
        System.out.println("Nbre␣de␣billets␣de␣" + val[i] + "␣euros␣:" + rep[i])
            ;
      }
    }
  }

  public static int[] calculMonnaie(int somme, int[] val) {
    int n = val.length;
    int[] r = new int[n];
    for (int i = 0; (i < n && somme != 0); i++) {
      while (somme >= val[i]) {
        r[i]++;
        somme -= val[i];
      }
    }
    return r;
  }

  public static void main(String[] args) {
    if (args.length == 0) {
      usage();
    } else {
      int[] val = { 100, 50, 20, 10, 5, 2, 1 };
      int a = Integer.parseInt(args[0]);
      int[] t = calculMonnaie(a, val);
      afficheMonnaie(t, val);
    }
  }
}
```

— Retrouvez dans la définition de la classe `Monnaie` les différentes notions de syntaxe vues en cours : les conventions de codage, la déclaration de classes et de méthodes de classe, le type de données tableaux et le passage de paramètres en ligne de commande.
— Qu'affiche le programme si on lui passe en paramètre la valeur 587 ? De façon générale, à quoi sert la classe `Monnaie` ?

Exercice 2. Dans cet exercice on va se familiariser avec les deux catégories de types de données manipulés par les programmes Java : les *types primitifs* (caractères, entiers, flottants, booléens) et les *types non-primitifs* (les tableaux et les classes). On considère les déclarations de classes suivantes :

```
public class TestPrimitifs {

  public static void f(int x) {
    x = 5;
  }

  public static void main(String[] argv) {
    int y = 1;
    f(y);
    System.out.println(y);

    char c = 'a';
    int z = c;
    System.out.println(z);

    float f = 1.5f;
    //float f = 1.5;
    float g = 0.1f;
    double h = (double) g;
    System.out.println(0.1 - h);
    boolean b = true;
    int v;
    // v = b;

    v = y;
    if (v == y) {
      System.out.println("v␣==␣y");
    } else {
      System.out.println("v␣!=␣y");
    }
  }
}
```

— Quelle est la valeur de `y` après l'appel de la fonction `f()` ? Expliquez.
— Quelle est la valeur de `z` ? Expliquez la conversion qui s'est produite.
— Quel est le résultat de l'instruction `float f = 1.5` ? Et de l'instruction `float f = 1.5f` ?
— Que produit l'instruction `v = b` ?
— Quel est le texte affiché après l'évaluation de l'expression `v == y` ?

```
class Point {
  public int x;
  public int y;
}

public class TestNonPrimitifs {

  public static void main(String[] argv) {
    Point p1 = new Point();
    p1.x = 1;
    p1.y = 2;
    Point p2 = new Point();
    p2.x = 1;
    p2.y = 2;
    if (p1 == p2) {
      System.out.println("p1␣==␣p2");
    } else {
      System.out.println("p1␣!=␣p2");
    }
    p1 = p2;
    if (p1 == p2) {
      System.out.println("p1␣==␣p2");
    } else {
```

```
      System.out.println("p1 != p2");
    }
    p2.x = 5;
    p2.y = 10;
    System.out.println(p1.x + " " + p1.y);
  }
}
```

— Quel est le texte affiché après l'évaluation de la première expression `p1 == p2` ? Et après la deuxième ?
— Quelles sont les valeurs affichées pour p1.x et p1.y ? Comment expliquez vous ce comportement ?

Tableaux

Exercice 3. Compléter les définitions des méthodes de la classe `TableauEntiers` manipulant des tableaux d'entiers signés distincts deux à deux de façon à ce que :
— la fonction `somme` retourne la somme des éléments du tableau d'entiers
— la fonction `minimum` retourne le minimum d'un tableau d'entiers
— la fonction `indiceMaxi` retourne l'indice du maximum d'un tableau d'entiers
— la fonction `opposes` retourne un nouveau tableau contenant les opposés des éléments du tableau d'entiers passé en paramètre
— la fonction `afficher` affiche le tableau d'entiers passé en paramètre
— la fonction `triBulles` qui trie de manière croissante le tableau d'entiers passé en paramètre.
Écrire ensuite une classe `TestTableauEntiers` qui fait appel aux méthodes de la classe `TableauEntiers` pour un tableau d'entiers constitué des arguments du programme. Un message d'erreur avec des précisions sur l'usage sera affiché dans le cas où ceux-ci seraient absents.

```
public class TableauEntiers{
    public static void afficher(int[] t){

    }
    public static int somme( int[] t){

    }
    public static int minimum( int[] t){

    }
    public static int indiceMaxi( int[] t){

    }
    public static int[] opposes(int[] t){

    }
    public static void triBulles(int[] t){

    }
}
```

Exercice 4. Écrire une classe `Stats` qui comporte les méthodes suivantes :
— une méthode `moyenne` qui retourne la moyenne des entiers dans le tableau d'entiers passé en paramètre
— une méthode `variance` qui retourne la variance des entiers dans le tableau d'entiers passé en paramètre, grâce à la formule :

$$var = \frac{1}{n}\sum_{i=0}^{n-1}(x_i - m)^2$$

où n, m et x_i représentent respectivement la taille du tableau, la moyenne de ses éléments, et le $i$ème élément du tableau

— une méthode `ecartype` (*std*) qui retourne l'écart type des entiers dans le tableau d'entiers passé en paramètre, grâce à la formule :

$$std = \sqrt[2]{\frac{1}{n}\sum_{i=0}^{n-1}(x_i - m)^2}$$

où m est la moyenne des entiers

— une méthode `mediane`, qui retourne une valeur M partageant la suite des entiers en deux sous-ensembles ayant le même nombre d'éléments. Par exemple pour `int[] tabEnt={5, 3, 2, 4, 1}`, $M = 3$ et pour `int[] tabEnt={1, 7, 5, 2}`, toute valeur comprise entre 3 et 4 pourrait être la médiane, mais en pratique, dans le cas d'une liste de nombre, c'est la moyenne arithmétique de ces deux valeurs centrales qui est le plus souvent utilisée, c.à.d. 3.5.

Pour cela, servez-vous des méthodes de la classe `TableauEntiers`. Compléter ensuite la classe `TestTableauEntiers` pour tester le fonctionnement de ces méthodes.

2 Travaux Pratiques

Edition, compilation, exécution avec Eclipse

Exercice 5.

1. Etudiez les premières parties (i.e. "Introduction" et "Premiers contacts") de l'introduction à eclipse disponible à l'adresse suivante : `http://dept-info.labri.fr/ENSEIGNEMENT/programmation2/intro-eclipse/`
2. Une fois ce travail effectué, vous devez avoir
   (a) un espace de travail ("workspace") spécifique pour l'UE dans le répertoire `~/workspaces/J1IN4W01`
   (b) un projet `td_0` contenant l'exemple "Hello World"

   Si ce n'est pas le cas, reprenez l'étude de l'introduction à eclipse
3. Créez un projet nommé `td_1` qui contiendra les classes de cette feuille d'exercices

Arguments d'un programme

Si ce n'est pas déjà le cas, parcourez la partie "Exécution d'un programme avec des arguments" pour comprendre comment les arguments du programme peuvent être défini sous eclipse.

Exercice 6. Écrire un programme `Combien` qui affiche le nombre d'arguments de la ligne de commande. Par exemple, le programme `Combien` appelé avec les arguments `a xx 546` affiche :

```
>Le programme a 3 arguments
```

Exercice 7. Ecrire un programme `HelloPerso` qui affiche "Hello <nom>!" où <nom> est passé en argument de la commande.

Exercice 8. Ecrire un programme `Hello` qui affiche "Hello <nom>!" pour chaque <nom> passé en argument de la commande et affiche "Hello World!" pour terminer. Exemple avec les arguments `Pierre Paul Julie` :

```
Hello Pierre !
Hello Paul !
Hello Julie !
Hello World !
```

Manipulation de tableaux

Exercice 9.

— Importer dans `Eclipse` la classe `TableauEntiers` à l'aide d'un copier-coller (c.f. partie " Incorporations de sources existantes").

```
public class TableauEntiers {
  public static void afficher(int[] t) {
    for (int i = 0; i < t.length; i++) {
      System.out.print("␣" + t[i]);
    }
    System.out.println();
  }

  public static void main(String[] args) {
    int[] t = new int[20];
    for (int i = 0; i < t.length; i++) {
      t[i] = 2 * i;
    }
    t[14] = 50;
    t[10] = -2;

    afficher(t);

    // System.out.println("La somme des elements est " + somme(t));
    // System.out.println("Le min des elements est " + minimum(t));
    // System.out.println("L'indice du max des elements est " + indiceMaxi(
        t));
    // System.out.println(" Le tableau des opposes est: ");
    // afficher(opposes(t));
  }

}
```

— Ajouter les méthodes suivantes puis les tester en décommentant les lignes correspondantes :

1. une méthode `somme` qui retourne la somme des éléments d'un tableau d'entiers.
2. une méthode `minimum` qui retourne le minimum d'un tableau d'entiers.
3. une méthode `indiceMaxi` qui retourne l'indice du maximum d'un tableau d'entiers.
4. une méthode `opposes` qui retourne un nouveau tableau contenant les opposés des éléments du tableau d'entiers passé en paramètre.

Types primitifs et non-primitifs

Exercice 10.

On considère l'exemple suivant (vu en cours) :

```
class Point {
  public int x;
  public int y;
}

public class PrimitifVsComposite {
  public static void main(String[] args) {
    int x, y;
    x = 3;
    y = 3;
    if (x == y) {
      System.out.println("x:" + x + "␣y:" + y + "␣␣x␣==␣y");
    } else {
```

```
      System.out.println("x:" + x + "␣y:" + y + "␣␣x␣!=␣y");
    }

    Point p1, p2;
    p1 = new Point();
    p2 = new Point();

    p1.x = 0;
    p1.y = 1;
    p2.x = 0;
    p2.y = 1;

    if (p1 == p2) {
      System.out.println("p1:" + p1 + "␣p2:" + p2 + "␣␣p1␣==␣p2");
    } else {
      System.out.println("p1:" + p1 + "␣p2:" + p2 + "␣␣p1␣!=␣p2");
    }
  }

}
```

— Importer dans `Eclipse` la classe `PrimitifVsComposite`.
— Quel est le résultat du premier test ? Que se passe-t-il si on le déplace juste après la déclaration de `x` et `y` ?
— Quel est le résultat du second test ? Que se passe-t-il si on le déplace juste après la déclaration de `p1` et `p2` ? Juste après l'instanciation de `p1` et `p2` ?
— Proposer deux façons différentes de modifier le code pour obtenir un message indiquant l'égalité des deux points.

Exercice 11.
— Les tableaux java sont-ils de type primitif ou non-primitif ?
— Complétez la classe `TableauEntiers` avec les méthodes suivantes :

1. une méthode qui échange deux éléments d'un tableau donné en paramètre. Les indices des éléments échangés sont également passés en paramètre.
2. une méthode qui inverse l'ordre des éléments d'un tableau passé en paramètre.
3. une méthode qui teste l'égalité de deux tableaux passés en paramètre.
4. une méthode qui renvoie une copie du tableau passé en paramètre.

Structures de contrôle

Exercice 12. Le but de cet exercice est d'écrire une classe `Arithmetique` qui regroupe les méthodes suivantes :

1. calcul du plus grand diviseur commun (pgcd) de deux entiers,
2. calcul du plus petit commun multiple (ppcm) de deux entiers,
3. une méthode `premier` qui retourne *true* si le nombre entier passé en paramètre est premier et *false* sinon,
4. une méthode `premiersEntre` qui affiche tous les nombres premiers compris, au sens large, entre les 2 valeurs entières min et max passées en paramètres.

Rappels :

Algorithme d'Euclide : a et b sont deux entiers. Tant que le reste de la division de a par b est non nul, a prend la valeur de b, b prend la valeur du reste. Le pgcd est le dernier reste non nul.

Le produit de deux entiers est égal au produit de leur pgcd par leur ppcm.