Programmation avancée
TP 1 – Premiers programmes [1]

A-G. Bosser, ENIB
V. Padovani, PPS - IRIF

Exercice 1 : Hello World

Parce qu'il faut bien commencer par quelque chose...

1. Enregistrez sous `emacs` le programme suivant dans un fichier nommé `Bonjour.java`.

```java
// fichier Bonjour.java
public class Bonjour {
  public static void main(String[] args){
    System.out.println("Bonjour !") ;
  }
}
```

2. La commande `javac` permet de compiler un code-source Java vers du bytecode, interprétable par une machine virtuelle. Utilisez cette commande avec votre programme source en tapant `javac Bonjour.java` dans le shell. Regardez quel est le fichier créé par cette opération.
3. La commande `java` permet de lancer la machine virtuelle Java pour l'interprétation de tout fichier de bytecode disposant d'un point d'entrée (une méthode `main`). Lancez votre premier programme en tapant `java Bonjour`.
4. On peut passer des arguments à un programme. Les arguments de la ligne de commande sont représentés dans le code source par le tableau de chaînes de caractères `args`, paramètre de la méthode `main`. Le premier argument est à la position `0`. Le nombre d'arguments est la longueur du tableau, il s'écrit `args.length`.

   Modifiez le programme précédent pour qu'il dise bonjour à chaque personne dont le nom sera passé en paramètre lors de l'exécution du programme. L'opérateur `+` permet de concaténer deux chaînes de caractères.

Exercice 2 : Première classe

On se propose d'enrichir le programme suivant, réparti dans deux fichiers distincts :

```java
// fichier Pair.java
public class Pair {
  private int fst;   // premier  element d'une paire
  private int snd;   // deuxieme element d'une paire
}
```

1. Les exercices de cette feuille, des deux suivantes ainsi que l'exercice du TP 5, ont été rédigés par Anne-Gwenn Bosser (ENIB). Ils ont été remaniés, mais restent conformes à leur articulation d'origine.

```java
// fichier TestPair.java
public class TestPair {
  public static void main (String[] argv) {
    Pair p = new Pair () ;
    System.out.println (p) ;
  }
}
```

1. Ajoutez ces deux classes à un nouveau projet Java créé avec Eclipse.
2. Ajoutez à la classe `Pair` une méthode de signature `public String toString()` renvoyant les valeurs des champs d'une paire sous forme de chaîne de caractères. Compilez et executez à nouveau le programme. Qu'observez-vous ?
3. Ajoutez à cette classe un constructeur prenant en argument des entiers, permettant d'initialiser les champs d'une paire. Testez ce constructeur dans `main`.

   Peut-on encore écrire `new Pair ()` dans `main` ? Pourquoi ? Ajoutez à la classe un constructeur sans arguments et de corps vide et invoquez-le dans `main`. Qu'observez-vous ?
4. Ajoutez à la classe les méthodes `getFst` et `getSnd` renvoyant les valeurs des champs d'une paire. Ces méthodes permettent d'accéder indirectement aux champs d'une paire. Testez ces méthodes dans `main`.
5. Ajoutez à la classe les méthodes `setFst` et `setSnd` permettant de modifier indirectement les valeurs des champs d'une paire. Testez ces méthodes dans `main`.

Exercice 3 :

Le but de cet exercice est d'écrire une classe `Stack` représentant une pile d'entiers, en utilisant un tableau. Vous définirez cette classe dans un nouveau projet d'Eclipse. Une autre classe `TestStack` du même projet contiendra le point d'entrée du programme (`main`) afin de tester votre code au fur et à mesure où vous l'écrirez.

1. La classe `Stack` a un champ `content` de type tableau d'entiers. Elle a également un champ `nbr` de type entier, indiquant le nombre courant d'éléments dans une pile. Les valeurs seront empilées dans le tableau de la gauche vers la droite : `nbr` est donc aussi l'indice du tableau considéré comme étant immédiatement à droite du sommet de la pile.
2. Le constructeur de `Stack` prend en paramètre un entier représentant la capacité de la pile, c'est à dire le nombre de cases du tableau.
3. Ecrire une méthode `empty` renvoyant le booléen `true` si la pile est vide, `false` sinon.
4. Ecrire les méthodes `push` (ajoutant un élément au sommet d'une pile) et `pop` (retournant l'élément au sommet d'une pile en l'enlevant de la pile). Que se passe-t-il si l'on tente d'invoquer `pop` sur une pile vide ?
5. Raffiner la méthode `push` de manière à agrandir à la volée la capacité de la pile si l'on tente d'empiler un élément dans une pile pleine.
6. Ecrire une méthode `public String toString ()` qui retourne le contenu de la pile sous forme de chaine de caractères.

Exercice 4 :

Pour occuper ceux qui ont déjà fini... Les tours de Hanoi. Etant donné trois piles et n entiers tous différents empilés sur la première, les plus petits au dessus des plus grands : un déplacement consiste à choisir une pile A non vide, et à enlever l'entier au sommet pour le mettre au sommet d'une autre pile B, si le sommet de cette pile n'est pas un entier plus petit (sinon, le déplacement est impossible).

Le but du jeu est de déplacer tous les entiers de la première pile sur la troisième, en se servant de la seconde. L'algorithme proposé est le suivant :

> Pour déplacer n entiers de la pile A sur la pile C en utilisant la pile B :
> - on déplace (récursivement) $n-1$ sommets de A vers B,
> - on déplace l'entier restant sur le sommet de la pile A sur la pile C,
> - on déplace (récursivement) $n-1$ sommets de B vers C.

1. Dans le même projet que `Stack`, ajouter une classe `Hanoi` représentant le jeu : ses champs seront trois piles et le nombre total d'entiers. Ajouter à cette classe une méthode `main` pour tester au fur et à mesure votre travail (vous pouvez commenter le `main` de `TestStack`).
2. Définir dans `Hanoi` un constructeur initialisant le jeu à partir d'un nombre d'entiers n à empiler – les entiers 1, 2, 3, ... n seront tous empilés sur la première pile, par ordre décroissant.
3. Définir la méthode `afficher` affichant le contenu des trois piles sur la console.
4. Définir la méthode privée (et récursive) `deplace` et la méthode `joue` lançant les déplacements des n entiers de la première pile vers la dernière – invoquez `afficher` dans cette méthode pour suivre chaque déplacement.
5. Enfin, finaliser la méthode `main` qui, à partir d'un entier saisi en ligne de commande, initialisera et lancera le jeu pour un entier n donné.