II.1102 – Algorithmique et Programmation
TP 2 : Structures de contrôle et Tableaux

Patrick Wang

30 septembre – 01 octobre 2019

1 Objectifs du TP

— Savoir écrire une condition et l'utiliser dans une structure conditionnelle.
— Savoir construire des structures conditionnelles complexes.
— Savoir distinguer les différents types de boucle.
— Savoir utiliser les différents types de boucle de façon appropriée.

2 Rappels

2.1 Structures conditionnelles

Les structures conditionnelles permettent l'exécution d'instructions spécifiques en fonction de conditions précises. Il est donc impératif de s'assurer que ces conditions soient bien écrites aussi bien syntaxiquement que *sémantiquement*.

Pour rappel, voici un exemple de structure conditionnelle en Java :

```
if (condition1) {
   // Bloc d'instructions a executer si condition1 est vrai
} else if (condition2) {
   // Bloc d'instructions a executer si condition1 est faux ET condition 2 est vrai
} else {
   // Bloc d'instructions a executer si condition1 ET condition2 sont faux
}
```

Nous avons vu en cours l'existence du mot-clé `switch` qui permet aussi de construire des structures conditionnelles. Cependant, les conditions qu'il est possible de spécifier avec le mot-clé `switch` ne peuvent pas être aussi complexes qu'avec un `if`.

2.2 Structures itératives

Les structures itératives permettent de répéter l'exécution d'un bloc d'instructions. Cela est particulièrement pratique puisque l'on va souvent chercher à *modulariser* notre code (i.e., identifier des processus simples et souvent utilisés).

Le cours mentionne l'existence de trois types de boucle :
— Les boucles `for` ;
— Les boucles `while` ;
— Les boucles `do while`.

Chacun de ces types de boucles ont une spécificité par rapport aux autres :

— la boucle `for` est à utiliser lorsque l'on sait exactement combien de fois les instructions sont répétées ;
— La boucle `while` est à utiliser lorsque la répétition est définie par une condition ;
— La boucle `do while` est un cas spécifique de la boucle `while` puisque le bloc d'instructions est ainsi exécuté au moins une fois.

2.3 Tableaux

Les tableaux sont des variables permettant de stocker plusieurs valeurs d'un même type. On dit que le contenu d'un tableau est **homogène**. En Java, les tableaux sont aussi dits **statiques** : une fois créé, on ne peut plus modifier la taille d'un tableau. On rappelle qu'il y a deux façons de créer des tableaux, et que les éléments sont accessibles par leur indice (qui démarre par 0).

```java
// Si on connait tous les elements
int[] tableauConnu = {4, 1, -1, 0};
int unElement = tableauConnu[3];

// Si on ne connait que la taille
String[] tableauInconnu = new String[4];
```

3 Exercices

Pour chacun de ces exercices, nous allons créer une nouvelle *fonction*.

Pour créer une nouvelle fonction, nous allons **pour le moment** écrire les quelques lignes suivantes sous la fonction `main()`. La classe `Main` ressemble alors à ça :

```java
public class Main {
   public static void main(String[] args) {
      // Des choses ici
   }

   public static void nomFonction() {
      // Contenu de la fonction
   }
}
```

Pour tester votre fonction, il faudra *l'appeler* dans la fonction `main()`. Si l'on souhaite appeler la fonction `nomFonction()`, le contenu de la classe `Main` deviendrait alors :

```java
public class Main {
   public static void main(String[] args) {
      nomFonction();
   }

   public static void nomFonction() {
      // Contenu de la fonction
   }
}
```

3.1 Structures conditionnelles

3.1.1 Calcul du discriminant du second degré

Pour rappel, le discriminant est obtenu grâce à la forumule suivante :

$$\text{Soit } P(x) = ax^2 + bx + c \text{ avec } (a, b, c) \in \mathbb{R}^3,\ \Delta = b^2 - 4ac$$

Le programme suivant permet de calculer le discriminant d'un polynome du second degré. Puis, il est capable de traiter le cas $\Delta < 0$. Cependant, les instructions ont été mélangées.

Question : Remettez les instructions dans l'ordre afin que le programme calcule effectivement le discriminant d'un polynome du second degré. On fera particulièrement attention à bien indenter le code une fois les instructions remises dans l'ordre.

```
public static void discriminant() {
   int delta = Math.pow(b, 2) - 4 * a * c;
   int c = scanner.nextInt();
   System.out.println("Quelle est la valeur de a ?");
   System.out.println("Ce polynome n'a pas de racine reelle");
   System.out.println("Quelle est la valeur de b ?");
   }
   int a = scanner.nextInt();
   int b = scanner.nextInt();
   if (delta < 0) {
   System.out.println("Quelle est la valeur de c ?");
   Scanner scanner = new Scanner(System.in);
}
```

Lorsque le discriminant est positif ou nul, le polynôme présente une racine double ou deux racines distinctes. Pour rappel :

- Si $\Delta = 0$, on a une racine double $x_0 = \frac{-b}{2a}$
- Si $\Delta > 0$, on a deux racines $x_0 = \frac{-b+\sqrt{\Delta}}{2a}$ et $x_1 = \frac{-b-\sqrt{\Delta}}{2a}$

Questions :

- Modifiez le programme afin de traiter le cas $\Delta = 0$. **Note** : On pourra utiliser `Math.sqrt()` afin de calculer la racine carré d'un nombre.
- Modifiez le programme afin de traiter le cas $\Delta > 0$.

Lorsque le discriminant est strictement négatif, on dit que le polynôme présente deux racines complexes :

$$x_0 = \frac{-b + i\sqrt{-\Delta}}{2a}, \quad x_1 = \frac{-b - i\sqrt{-\Delta}}{2a}$$

Questions : Modifiez le programme afin de traiter le cas $\Delta < 0$.

3.1.2 Calcul de la parité d'un nombre

Questions :

1. Créez une fonction `parite()` ;
2. Complétez cette fonction pour demander à l'utilisateur de saisir un entier ;
3. Si cet entier est pair, affichez que ce chiffre est pair en utilisant une concaténation de chaîne de caractères avec une valeur numérique ;
4. Si cet entier est impair, affichez que ce chiffre est impair en utilisant une concaténation de chaîne de caractères avec une valeur numérique.

3.1.3 Calcul d'extremum

Questions :

1. Créez une fonction `max()` ;
2. Complétez cette fonction pour demander à l'utilisateur de saisir deux entiers ;
3. Déterminez le maximum de ces deux entiers, puis affichez la valeur du maximum à l'utilisateur ;
4. Reprendre les étapes 1 – 3, mais cette fois-ci pour déterminer le minimum entre deux entiers.

3.1.4 Égalité entre chaînes de caractères

En Java, les chaînes de caractères ne sont pas considérées comme des variables primitives. Une des conséquences à cela est que les tests d'égalité entre chaînes de caractères ont une syntaxe différente à ce dont on est habitué.

Étudions les quelques lignes suivantes :

```
String premiereChaine = "Bonjour, le monde !";
String deuxiemeChaine = "Hello, world!";
String troisiemeChaine = "Bonjour, le monde !";

boolean test1 = (premiereChaine == deuxiemeChaine);   // <- false
boolean test2 = (premiereChaine == troisiemeChaine);   // <- false
boolean test3 = premiereChaine.equals(deuxiemeChaine); // <- false
boolean test4 = premierChaine.equals(troisiemeChaine); // <- true
```

Ici, il faut bien comprendre la distinction entre `==` et `equals()`.

Dans le cas de `==`, Java va comparer les *variables* entre elles et non pas leurs valeurs. Il est donc assez facile de comprendre pourquoi `test1` est `false`. Quant à `test2`, il faut comprendre que `premiereChaine` et `troisiemeChaine` sont deux variables distinctes se trouvant dans deux emplacements mémoire différents et ayant certes la même valeur.

Dans le cas de `equals()`, Java va comparer les *valeurs* des variables. C'est donc pourquoi `test4` est `true`.

Questions :

1. Créez une fonction `egaliteStr()` ;
2. Complétez cette fonction pour demander à l'utilisateur de saisir deux chaînes de caractères ;
3. Déterminez si ces chaînes de caractères sont égales ou non, puis affichez le résultat.
4. Essayez votre programme avec plusieurs jeux de données (chaînes identiques, presque identiques, différentes, avec majuscules ou sans, etc.).

3.2 Structures itératives

Le programme suivant permet de calculer la factorielle d'un entier, mais les instructions ont été mélangées. Pour rappel, la factorielle d'un entier est définie de la façon suivante :

$$\forall n \in \mathbb{N}^*,\ n! = 1 \times \ldots \times n \, et \, 0! = 1$$

```
public static void factorielle() {
   }
   factorielle *= i;
   for (int i = 0; i <= n; i++) {
   int n = scanner.nextInt();
   System.out.println(n + "! = " + factorielle);
   Scanner scanner = new Scanner(System.in);
   int factorielle = 1;
   System.out.println("Saisir un entier positif ou nul");
}
```

Questions :

1. Remettez les instructions dans l'ordre afin que le programme calcule effectivement la factorielle d'un entier.
2. Testez votre programme avec $n = 1$. Que se passe-t-il ? Corrigez le programme puis testez une nouvelle fois avec cette valeur.
3. Testez votre programme avec $n = 0$. Que se passe-t-il ? Corrigez votre programme puis testez une nouvelle fois avec cette valeur.
4. Testez désormais votre programme avec des valeurs quelconques afin de vérifier qu'il est correct. Corrigez-le si ce n'est pas le cas.

3.2.1 Compte à rebours

Questions :

1. Créez une fonction `countdown()` ;
2. Complétez cette fonction pour afficher un compte à rebours allant de 10 jusqu'à 0 ;
3. Une fois que le compte à rebours a atteint 0, affichez le message « BOOM ! »

3.2.2 Valeurs de carrés

Questions :

1. Créez une fonction `carres()` ;
2. Complétez cette fonction afin d'afficher côte à côte les valeurs de x et de x^2. Pour cet exercice, nous privilégierons la multiplication de deux termes plutôt que la fonction `Math.pow()` et nous séparerons les valeurs par des tabulations.

3.2.3 Tables de multiplication

Pour cet exercice, nous souhaitons afficher la table de multiplication affichée en Figure 1.

Pour cela, nous allons utiliser la fonction `System.out.print()` (et non pas `println()`) pour afficher du texte sans créer de nouvelle ligne à la fin. De plus, nous utiliserons des tabulations pour séparer les éléments sur chaque ligne.

Nous allons dans un premier temps nous concentrer sur l'affichage de la première ligne.

Questions :

1. Quel est le type de boucle le plus approprié pour afficher la première ligne ?
2. Créez une fonction `tableMultiplication()` ;
3. Complétez ce programme pour afficher la première ligne de la table de multiplication.

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

FIGURE 1 – Table de multiplication à afficher

La table de multiplication est construite de sorte que :
— Les éléments de la ligne 1 sont les éléments de la ligne 1 multipliés par 1 ;
— Les éléments de la ligne 2 sont les éléments de la ligne 1 multipliés par 2 ;
— Les éléments de la ligne 3 sont les éléments de la ligne 1 multipliés par 3 ;
— Les éléments de la ligne n sont les éléments de la ligne 1 multipliés par n.

Question :

1. Modifiez votre programme pour imbriquer une seconde boucle `for` à l'intérieur de la première. Quelles sont les bornes de cette seconde boucle ?
2. Quelle est la relation entre un élément de la table de multiplication et ses indices de ligne et de colonne ?
3. Complétez votre programme pour afficher la table de multiplication complète.

3.2.4 Saisies de l'utilisateur

Une règle d'or en informatique est de ne jamais faire confiance à l'utilisateur. C'est pourquoi, lorsque l'on demande à l'utilisateur de saisir des valeurs, il est en général souhaité de vérifier la validité de cette saisie.

Par exemple, le calcul de la factorielle nécessite un entier positif ou nul. Le code suivant permet de s'assurer que l'utilisateur a bien saisi une telle valeur, sinon on répète la saisie.

```
Scanner scanner = new Scanner(System.in);
int entier;
do {
   System.out.println("Veuillez saisir un entier positif ou nul");
   entier = scanner.nextInt();
} while (entier < 0);
```

3.2.5 Division d'entiers

Questions :

1. Créez une fonction `division()` ;
2. Complétez cette fonction afin de demander à l'utilsateur de saisir deux nombres entiers ;
3. Calculez puis affichez la division de ces deux nombres ;

4. Modifiez cette fonction afin de répéter la saisie du dénominateur si celui-ci est égal à 0.

3.3 Structures de contrôle

Nous allons désormais traiter des exercices qui auront besoin de structures conditionnelles et itératives.

3.3.1 Règle graduée

L'objectif de cet exercice est de *dessiner* une règle graduée en console. La longueur de cette règle sera donnée par l'utilisateur. Le programme pourra, par exemple, avoir le comportement suivant :

```
Longueur ? 53
-----------------------------------------------------
```

Questions :

1. Créez une fonction `regle()` ;
2. Complétez cette fonction en demandant à l'utilisateur de saisir une valeur entière tant que celle-ci n'est pas strictement positive ;
3. Affichez la règle graduée.
4. Modifiez votre programme pour afficher les dizaines à l'aide d'une graduation.

Ainsi, l'exemple précédent devient :

```
|---------|---------|---------|---------|---------|---
```

3.3.2 Nombres premiers

Un nombre premier est un nombre qui n'est divisible que par 1 et par lui-même. Un corollaire à cette définition est qu'un nombre premier possède *exactement* deux diviseurs.

Questions :

1. Créez une fonction `nombrePremier()` ;
2. Complétez cette fonction en demandant à l'utilisateur de saisir une valeur entière tant que celle-ci n'est pas strictement positive ;
3. Déterminez si ce nombre saisi est premier ou non, puis affichez le résultat en console.

3.4 Tableaux

3.4.1 Manipulations sur un tableau

Le programme suivant initialise un tableau et demande à l'utilisateur de le remplir avec des valeurs réelles.

```
public static void initialisationTableau() {
   int[] tableau = new int[20];
   Scanner scanner = new Scanner(System.in);
   for (int i = 0; i < tableau.length; i++) {
      System.out.println("Saisir un entier");
      int entier = scanner.nextInt();
      tableau[i] = entier;
```

```
    }
}
```

Questions :

1. Complétez ce programme pour déterminer le minimum et le maximum du tableau saisi par l'utilisateur ;
2. Complétez ce programme pour calculer la somme des éléments du tableau ;
3. Complétez ce programme pour afficher les éléments pairs du tableau ;
4. Complétez ce programme pour afficher les éléments d'indice pair du tableau ;
5. Créez une nouvelle fonction `inverseTableau(int[] tableau)` qui prend en paramètre un tableau et qui inverse l'ordre de ses éléments.

3.5 Menu d'exercices

Pour simplifier les tests de vos exercices, vous pouvez développer un menu permettant à l'utilisateur d'appeler une fonction spécifique. Vous pourrez écrire ce menu directement dans la fonction `main()`. Pour ce TP, ce menu pourrait avoir le fonctionnement suivant :

```
Quel exercice ? Saisissez :
1. Discriminant
2. Parité d'un nombre
3. Calcul d'extremum
4. Égalité entre chaînes de caractères
5. Factorielle
6. Compte à rebous
7. Valeurs de carrés
8. Table de multiplication
9. Division d'entiers
10. Règle graduée
11. Nombres premiers
12. Manipulations sur un tableau
```

Selon la saisie de l'utilisateur, on appellera la fonction correspondante. On pourra utiliser le mot-clé `switch` pour cet exercice.