Chapitre 7

Les structures conditionnelles et les boucles

1. Les structures conditionnelles
2. Les boucles
3. Exemples d’application

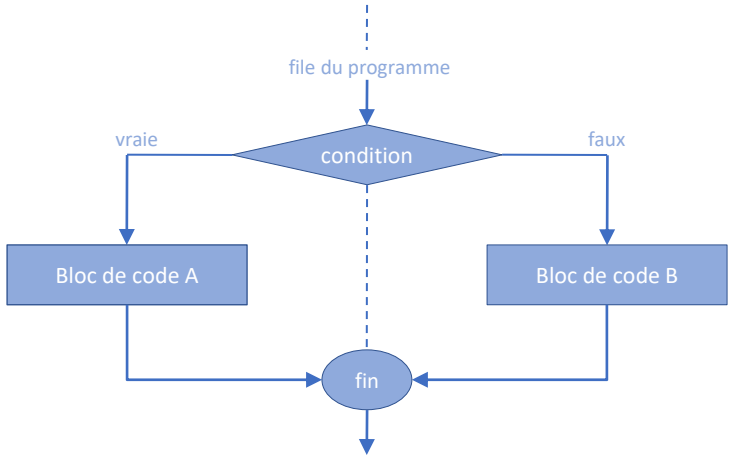
1 Les structures conditionnelles

2 Les boucles

3 Exemples d’application

Les structures conditionnelles

- Une structure conditionnelle est une instruction spécifique du langage qui permet de tester qu'une condition donnée est vraie ou non. Et en fonction du résultat du test, autorise ou non l'exécution d'un **bloc de code**.
- Un bloc de code est tout simplement une suite d'instructions balisées par les accolades ouvrantes ({) et fermante (})
- Plusieurs types de structures conditionnelles en Java :
  - l'instruction if
  - l'instruction if..else
  - l'instruction if..elseif..else
  - l'instruction ternaire
  - l'instruction switch..case



Les structures conditionnelles

L'instruction if

- Une instruction **if** caractérise la conjonction de subordination **si** dans un programme. C'est-à-dire, un bloc de code subordonné à une condition. Son format est :

```
if (condition) {
     /* bloc de code qui
       s'exécutera si la
      condition est vraie
      */
}
```

- Le bloc de code entre les accolades ne s'exécutera que si *condition* est évaluée à **true**. C'est-à-dire, si elle est vraie
- *condition* est tout simplement une **expression logique** composée *d'opérateurs logiques* et de *comparaison*
- Exemple :
  - Soit *a* et *b* deux variables entières dans un programme. Comment traduiriez-vous le besoin suivant ? :

    *On souhaite interchanger les valeurs de a et de b **si et seulement si**, a est supérieur à 10 et b est inférieur ou égal à 101*

  - Réponse :

```
if ( (a > 10) && (b <= 101) ) {
     int var = 0;
     var = b;
     b = a;
     a = var;
}
```

- Si dans le programme, on se retrouve dans un scénario où a = 13 et b = 54 par exemple,

  Alors la condition (a > 10) && (b <= 101) => true
  Et donc en sortie on aura : a = 54 et b = 13
  On a donc pu interchanger leurs valeurs

- A contrario, pour toute valeur de a et b telle que la condition (a > 10) && (b <= 101) => false,

  le bloc de code dans l'instruction if ne sera pas exécuté et au sortir a et b conserverons leurs valeurs initiales

Les structures conditionnelles

L'instruction if..else

- Une instruction **if..else** traduit l'expression si..sinon.. C'est-à-dire : si **une *condition est vérifiée*** alors **exécuter un *bloc de code A***, sinon, ***exécuter plutôt le bloc de code B*** :

```
if (condition) {
  // bloc de code A
} else {
  // bloc de code B
}
```

Les blocs de code A et B sont exclusifs entre eux.
Ils ne peuvent pas être exécutés en même temps dans le même if..else. C'est soit l'un, soit l'autre.

- Exemple :
  - Soit *a* et *b* deux variables entières dans un programme. Comment traduiriez-vous le besoin suivant ? :

*On souhaite interchanger les valeurs de a et de b **si** a est supérieur à 10 et b est inférieur ou égal à 101, **sinon** affecter -1 à chacune d'elles*

  - Réponse :

```
if ( (a > 10) && (b <= 101) ) {
  int var = 0;
  var = b;
  b = a;
  a = var;
} else {
  a = -1;
  b = -1;
}
```

- Si dans le programme, on se retrouve dans un scénario où a = 13 et b = 54 par exemple,

Alors la condition (a > 10) && (b <= 101) => true
Et donc le bloc de code situé entre les accolades du if sera exécuté et au sortir a = 54 et b = 13

- A contrario, pour toute valeur de a et b telle que la condition (a > 10) && (b <= 101) => false,

le bloc de code situé entre les accolades du **else** sera exécuté et au sortir a = b = -1

Les structures conditionnelles

L'instruction if..elseif..else

- Une instruction **if..elseif..else** traduit une série d'hypothèses conditionnelles de type si..sinon si.. Sinon.. :

```
if (condition1) {
  // bloc de code A
} else if (condition2) {
  // bloc de code B
} else {
  // bloc de code C
}
```

- **Si** condition1 est vraie, on exécute exclusivement le bloc de code A et aucun autre bloc de code B et C n'est exécuté
- **Sinon si** c'est plutôt la condition2 qui est vraie, alors seul le bloc de code B sera exécuté et les blocs A et C ne le seront pas. NB : On peut avoir plusieurs conditions **sinon si** dans l'instruction
- Enfin, **si ni la condition1 ni la condition2 ne sont vraies**, alors le bloc de code C sera exécuté

- Exemple :
  - Soit *a* et *b* deux variables entières dans un programme. Comment traduiriez-vous le besoin suivant ? :

    *On souhaite interchanger les valeurs de a et de b **si** a est supérieur à 10 et b est inférieur ou égal à 101, **ou alors si** a supérieur à b, ajouter la valeur b à a, **sinon** affecter -1 à chacune d'elles*

  - Réponse :

```
if ( (a > 10) && (b <= 101) ) {
  int var = 0;
  var = b;
  b = a;
  a = var;
} else if ( a > b ) {
  a += b;
} else {
  a = -1;
  b = -1;
}
```

Les structures conditionnelles

L'instruction ternaire

- C'est une forme contractée d'expression de l'instruction **if..else** et même **if..elseif..else :**

```
(condition) ? [//bloc code A si condition vraie] : [//bloc code B si condition fausse]
```

⟷

```
if (condition) {
  // bloc de code A
} else {
  // bloc de code B
}
```

- Elle est souvent utilisée comme diminutif d'une instruction if..else plus verbeux. A contrario, elle n'est pas très lisible
- Exemple :
  - Soit *a* et *b* deux variables entières dans un programme. Comment traduiriez-vous le besoin suivant en ternaire ? :

    *On souhaite interchanger les valeurs de a et de b **si** a est supérieur à 10 et b est inférieur ou égal à 101, **sinon** affecter -1 à chacune d'elles*
  - Réponse :

```
int c = a > 10 && b <= 10 ? a : -1;
int d = a > 10 && b <= 10 ? b : -1;
a = d;
b = c;
```

⟷

```
if ( (a > 10) && (b <= 101) ) {
  int var = 0;
  var = b;
  b = a;
  a = var;
} else {
  a = -1;
  b = -1;
}
```

Les structures conditionnelles

L'instruction switch..case (1/2)

- Une façon d'exprimer par des **cas de scénario**, les hypothèses de valeurs que peut prendre une variable utilisée comme condition initiale :

```
switch (variable) {

    case Valeur1 :
    //bloc de code 1
    break;

    case Valeur2 :
    //bloc de code 2
    break;

    case ValeursN :
    //bloc de code N
    break;

    default:
    //bloc de code exécuté en dernier ressort si aucune des hypothèses ci-dessus n'a été vraie
    break;
}
```

- Seules les variables de **types primitifs (byte, short, char, int, long), les Wrappers, String et Enum** sont utilisées dans un switch..case
- L'instruction break est très importante, car elle **met fin à un bloc**, sans quoi l'exécution se poursuit au bloc du cas suivant
- Le bloc default s'exécutera uniquement si aucun des *cases* n'a pu être exécutés. C'est le bloc que l'on aimerait que le switch..case exécute par défaut si aucun des cas de scénario du test de la variable n'a pu être trouvé

Les structures conditionnelles

L'instruction switch…case (2/2)

- Exemple : soit une variable de type String nommée *jour* dans un programme. Ecrivez un code Java qui affiche, pour les jours ouvrés de la semaine : « 1 » si *jour* vaut Lundi, « 2 » si *jour* vaut Mardi, « 3 » si *jour* vaut Mercredi, etc, ou sinon affiche « -1 » si *jour* ne correspond à aucun des jours ouvrés

- Réponse :

```
switch (jour) {

    case "Lundi" :
    System.out.println(1);
    break;

    case "Mardi" :
    System.out.println(2);
    break;

    case "Mercredi" :
    System.out.println(3);
    break;

    case "Jeudi" :
    System.out.println(4);
    break;

    case "Vendredi" :
    System.out.println(5);
    break;

    default:
    System.out.println(-1);
    break;
}
```

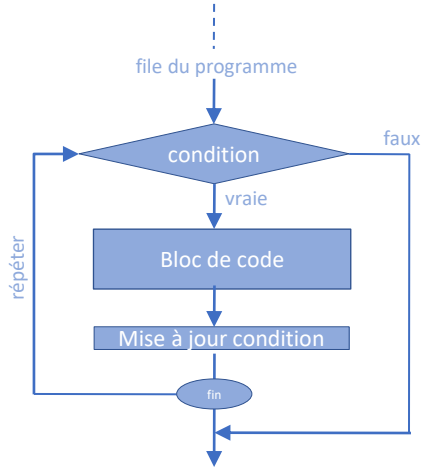
1 Les structures conditionnelles

2 Les boucles

3 Exemples d’application

Les boucles

- Une boucle est une instruction spécifique du langage qui permet de répéter plusieurs fois l'exécution d'un **bloc de code** tant qu'une condition donnée est vérifiée
- Elle balise le bloc de code à répéter entre les accolades ouvrantes ({) et fermante (})
- Plusieurs types de boucles en Java :
  - la boucle for
  - la boucle foreach
  - la boucle while
  - la boucle do…while



Les boucles

La boucle for

- Elle utilise un *compteur* ayant une valeur initiale, lui fixe une condition d'arrêt via un seuil à ne pas franchir, et enfin incrémente ou décrémente ce compteur jusqu'à ce qu'il franchisse ce seuil pour arrêter la boucle.
  Son format est le suivant :

```
for (compteur=ValeurInitiale; ConditionDarrêt; incrémentation/décrémentation du compteur) {
   // bloc de code
}
```

- Il est possible de déclarer la variable *compteur* à l'extérieur comme à l'intérieur de la boucle.
- Exemple :
  - Soit le tableau *tab* déclaré comme suit : **String[] tab = {"lundi", "mardi", "mercredi", "jeudi", "vendredi", "samedi", "dimanche"}** ;
    C'est un tableau de chaine de caractères contenant les jours de la semaine.
    Comment listeriez-vous tous les jours de la semaine contenus dans le tableau tab ?
  - Réponse :

```
for (int i = 0; i < 7; i++) {
  System.out.println(tab[i]);
}
```

⟺

```
int i;
for (i = 0; i < 7; i++) {
  System.out.println(tab[i]);
}
```

⟺

```
for (int i = 0; i < tab.length; i++) {
  System.out.println(tab[i]);
}
```

Pour rappel, *tab.length* retourne la taille du tableau qui ici vaut 7 (cf. chap5, slide 16)

Les boucles

La boucle foreach

- C'est une sorte de *boucle for* améliorée qui permet de s'affranchir de la gestion du compteur lorsqu'on veut boucler sur les éléments d'un tableau ou d'une collection. Ainsi dans l'instruction *for*, on récupère directement et itérativement chaque élément du tableau/collection.

  Son format est le suivant :

```
for (TypeDeDonnéesDesElementsDuTableau  unElement : tableau/Collection) {
   // bloc de code
}
```

- Exemple :
  - Soit le tableau *tab* déclaré comme suit : `String[] tab = {"lundi", "mardi", "mercredi", "jeudi", "vendredi", "samedi", "dimanche"};`

    Ou même la collection suivante : `ArrayList<Integer> liste = new ArrayList<>(); liste.add(1); liste.add(2); liste.add(3); liste.add(4); liste.add(5);`

    Comment itéreriez-vous sur ces deux objets avec la boucle foreach pour afficher chacun de leurs éléments ?
  - Réponse :

```
for (String elt : tab) {
  System.out.println(elt);
}
```

```
for (Integer elt : liste) {
  System.out.println(elt);
}
```

Les boucles

La boucle while

- C'est une boucle qui commence d'abord par vérifier qu'une **condition d'arrêt** est fausse :
  - Si oui, elle exécute le bloc de code, puis répète l'action plusieurs fois tant que la condition d'arrêt reste vraie.
    Dans le bloc de code, les éléments faisant partir de la condition d'arrêt doivent nécessairement être mis-à-jour
    Dès que la condition d'arrêt devient fausse, la boucle s'arrête.
  - Si non, le bloc de code dans la boucle n'est même pas exécuté. Il est ignoré

Son format est le suivant :

```
while (ConditionDarrêt) {
   // bloc de code
   // + mis-à-jour conditionDarrêt
}
```

- Exemple :
  - Ecrire une boucle while qui prend une variable *n* entière positive et affiche tous les nombres compris entre ***n* et 0**.
  - Réponse :

```
int n = 100;
while (n >= 0) {
  System.out.println(n);
  n--;
}
```

Les boucles

La boucle do…while

- C'est la même chose que la *boucle while* à la différence qu'elle commence d'abord par exécuter une première fois le bloc de code. Une fois fait, elle vérifie ensuite qu'une **condition d'arrêt** est fausse :
  - Si oui, elle répète l'exécution du bloc de code
    Dans le bloc de code, les éléments faisant partir de la condition d'arrêt doivent nécessairement être mis-à-jour
  - Si non, la boucle s'arrête.

Son format est le suivant :

```
do {
   // bloc de code
   // + mis-à-jour conditionDarrêt
} while (ConditionDarrêt);
```

- Exemple :
  - Ecrire une boucle do..while qui prend une variable *n* entière positive et affiche tous les nombres compris entre **0 et *n***.
  - Réponse :

```
int n = 100;
int i = 0;
do {
  System.out.println(i);
  i++;
} while (i <= n);
```

Les boucles

Autres caractéristiques

- Une boucle dont la *condition d'arrêt* est toujours fausse, est qualifiée de **boucle infinie**. Elle doit être utilisée avec précaution et en connaissance de cause, car mal utilisée elle empêche tout un programme de se terminer

```
do {
   // bloc de code
} while (true);
```

```
while (true) {
   // bloc de code
}
```

- Pour toute boucle *for, foreach, while* et *do..while*, on peut utiliser l'instruction break pour la « casser » (l'arrêter).
  Une fois break invoquée, toutes les instructions qui la suivent ne seront pas exécutées et la boucle s'arrêtera d'itérer.
  On l'utilise donc très intelligemment avec des structures conditionnelles
  Exemple de format :

```
while (ConditionDarrêt) {
   // bloc de code 1
   break;
   // bloc de code 2
}
```

- Dans une boucle *for, foreach, while* et *do..while*, on peut utiliser l'instruction continue pour « bypasser » les instructions qui la suivent et recommencer une nouvelle itération.
  Une fois continue invoquée, toutes les instructions qui la suivent ne seront pas exécutées et la boucle démarrera une nouvelle itération.
  On l'utilise donc très intelligemment avec des structures conditionnelles
  Exemple de format :

```
while (ConditionDarrêt) {
   // bloc de code 1
   continue;
   // bloc de code 2
}
```

1 Les structures conditionnelles

2 Les boucles

3 Exemples d’application

Exemple 1

L'exemple ci-contre, présente un programme complet (que vous pouvez reproduire dans Eclipse) utilisant la structure conditionnelle *if....else* et les *boucles for* et *foreach*, pour répondre à l'énoncé suivant :

Ecrire un programme exécutable qui crée une collection de type ArrayList, y ajoute des nombres entiers de 0 à 10. Puis itère sur chaque élément e de cette liste et affiche :

- *e est un nombre pair, s'il est est pair*
- *e est un nombre impair, sinon*

```java
import java.util.ArrayList;
import java.util.List;

public class TestExemple1 {


   public static void main(String[] args) {

      //déclaration de la liste
      List<Integer> liste = new ArrayList<>();

      /*boucle for pour initialiser la liste
       * avec les valeurs de 0 à 10.
       */
       for (int j = 0; j <= 10; j++) {
           liste.add(j);
       }

       /*boucle foreach pour parcourir la liste
        * et identifier les nombres pairs et impairs
        */
       for (int e : liste) {
           if(e%2 == 0) {
               System.out.println(e + " est un nombre paire");
           } else {
               System.out.println(e + " est un nombre impaire");
           }
       }
    }

}
```

Résultat :

```
0 est un nombre pair
1 est un nombre impair
2 est un nombre pair
3 est un nombre impair
4 est un nombre pair
5 est un nombre impair
6 est un nombre pair
7 est un nombre impair
8 est un nombre pair
9 est un nombre impair
10 est un nombre pair
```

Exemple 2

L'exemple ci-contre, présente un programme complet (que vous pouvez reproduire dans Eclipse) utilisant la structure conditionnelle *if* et les *boucles for* et *while*, l'instruction *continue* pour répondre à l'énoncé suivant :

Ecrire un programme exécutable qui crée une collection de type ArrayList, y ajoute des nombres entiers de 0 à 10. Puis comptabilise le total de nombres pairs dans cette liste et affiche :
- Total des nombres pairs entre 0 et 10 : n

NB : la boucle while n'est pas la plus optimale à utiliser dans ce programme. A la place on aurait pu utiliser une boucle foreach plus simple. Mais j'ai fait le choix du while pour vous faire observer les différences de conception que peuvent avoir les uns et les autres lors de l'écriture d'un programme et vous passer le message suivant :

Pensez toujours à rechercher les instructions les plus optimales lorsque vous développez. Un code optimalement écrit et qui compile est meilleur qu'un code qui compile tout simplement.

```java
import java.util.ArrayList;
import java.util.List;

public class TestExemple2 {


   public static void main(String[] args) {

      //déclaration de la liste
      List<Integer> liste = new ArrayList<>();

      /*boucle for pour initialiser la liste
       * avec les valeurs de 0 à 10.
       */
       for (int j = 0; j <= 10; j++) {
           liste.add(j);
       }

       /*boucle while pour parcourir la liste
        * et identifier les nombres pairs et les comptabiliser
        */
       int count = 0;
       int i = 0;
       while (i < liste.size()) {
           int elt = liste.get(i);
           if(elt%2 != 0) {
               i++;
               continue;
           }
           i++;
           count++;
       }
       System.out.println(" Total des nombres pairs entre 0 et 10 : " + count);
    }

}
```

Résultat : Total des nombres pairs entre 0 et 10 : 6


SUCCESS