

## Séance 2b: SÉANCE 2B.

Université Paris-Diderot

### Objectifs:

- S'exercer à écrire du JAVA.
- Effectuer des calculs arithmétiques dépendant | de l'indice d'une boucle.

Dans cette séance, vous allez tout d'abord apprendre à traduire du code décrit informellement en français en du code JAVA. Ensuite, vous allez progressivement concevoir des programmes qui (i) font des calculs arithmétiques (avec ou sans variables), (ii) définissent et utilisent des fonctions et (iii) utilisent des boucles de type `for` pour répéter des actions ou bien accumuler des résultats partiels dans des variables.

## 1 Du français au Java

Dans la série d'exercices qui suit, vous allez améliorer votre connaissance de la syntaxe de JAVA. Ce que doit faire le code JAVA à produire est écrit en français. (Il n'y a normalement rien à deviner.) Même si ce n'est pas précisé, vous devez systématiquement tester vos programmes.

### Exercice 1 (Expressions arithmétiques, ☆)

Complétez le fichier `Expression.java` pour écrire un programme qui affiche les résultats des calculs suivants :

- le double de 21;
- la division entière par 32 de 1025;
- la division entière par 128 de la somme de 1000 et de 24;
- la division entière par 3 du produit de 771 et de 80085.

□

### Exercice 2 (Expressions arithmétiques avec variable, ☆)

Écrivez dans le fichier `ExpressionVar.java` une séquence d'instructions qui :

1. affecte une variable `x` avec la valeur d'une expression arithmétique qui calcule le triple de 73;
2. affecte une variable `y` avec la valeur d'une expression arithmétique qui calcule les deux tiers de `x`;
3. affiche la division entière par 5 de la somme de `x` et de `y`.

□

### Exercice 3 (Définition et usage de fonctions, ☆)

Pour cet exercice, vous utiliserez le fichier `Fonctions.java`.

1. Écrivez une fonction `div10` qui attend un entier `x` et retourne le résultat de la division entière de 10 par `x`.

- Évaluez la fonction `div10` pour `x` valant 10, 2 et 0. Que se passe-t-il dans ce dernier cas ?
- À l'aide d'une unique expression `JAVA`, déterminez la somme des évaluations de la fonction `div10` pour `x` valant 1, 3, 5 et 7.
- Écrivez une fonction `sumproduct` qui attend trois entiers `x`, `y` et `z` et qui renvoie la somme du produit de `x` et de `y`, du produit de `x` et de `z` et du produit de `y` et de `z`.
- Évaluez la fonction `sumproduct` pour `x` valant 2, `y` valant « 2 + 1 » et `z` valant « `div10` (2) ».
- Quelles sont les erreurs produites par la compilation d'un programme essayant de calculer le résultat des expressions `JAVA` suivantes ?
  - `div10 ()`
  - `div10 (33, 37)`
  - `div10 ("Dark Vador")`

□

#### Exercice 4 (Instructions conditionnelles, ★)

Utilisez pour cet exercice le fichier `Conditions.java`.

- Écrivez une fonction `absolute` qui attend un entier `x` et qui renvoie `x` si `x > 0` et qui renvoie `-x` sinon.
- Que valent « `absolute (73)` » et « `absolute (-37)` » ?
- Écrivez une procédure `solve` qui attend trois entiers `x`, `y` et `z` et qui affiche :
  - La chaîne « `x + y = z` » si `x + y = z`.
  - La chaîne « `x - y = z` » si `x - y = z`.
  - La chaîne « `- x - y = z` » si `- x - y = z`.
  - La chaîne « `- x + y = z` » si `- x + y = z`.
  - La chaîne « Rien du tout! » sinon.
- Testez la fonction précédente pour solliciter tous les cas possibles d'évaluation.

□

#### Exercice 5 (Boucles qui répètent la même chose plusieurs fois, ★)

Traduisez dans le fichier `Yoda.java` les descriptions informelles de programme suivantes :

- Affiche 100 fois la ligne « May the force be with you. »
- Définis une fonction `yoda` qui attend un entier `n` et affiche « May the force be with you. » `n` fois. Évalue « `yoda (100)` ».
- Que fait le programme (à écrire dans la fonction `main`) :

```

1 for (int i = 0; i < 42; i++) {
2     yoda (100);
3 }

```

? Recopiez le code de la fonction `yoda` à la place de son appel en substituant correctement son paramètre `n` par la valeur passée en argument. Pourquoi dit-on que l'instruction obtenue est une boucle imbriquée ?

□

#### Exercice 6 (Boucles dont chaque étape dépend de l'indice d'itération, ★)

Traduisez les descriptions informelles de programme suivantes dans le fichier `BouclesDr.java` :

- Pour `i` allant de 0 à 99, affiche `i`.
- Définis une fonction `count_to` qui attend un entier `n` et qui pour `i` allant de 0 à `n`, affiche `i`. Évalue `count_to (100)`.
- Définis une procédure `filter_even` qui attend un entier `n` et qui affiche `n` si « `n % 2 == 0` » et affiche la chaîne « ... » sinon. Pour `i` allant de 0 à 65535, appelle `filter_even` pour `n` valant `i`.

□

À partir de ce point du sujet, vous commencez à savoir *écrire* des programmes JAVA. Il faut maintenant passer à l'étape suivante : *concevoir* des programmes JAVA. Cela signifie que vous allez devoir *chercher les programmes* qui répondent au problème posé. Pour cela, on vous propose d'utiliser la méthodologie suivante :

1. Lisez attentivement le sujet. Si le sujet demande d'écrire une fonction, imaginez des exemples d'entrées pour cette fonction et les sorties que l'on devrait obtenir pour chacun de ces exemples.
2. Écrivez en *français*, la description la plus précise d'un programme qui répond au problème posé.
3. Traduisez en JAVA le programme obtenu à l'étape précédente.
4. Testez votre programme sur *suffisamment* d'exemples pour vous convaincre que votre programme est correct : il faut au moins que chaque instruction de votre programme ait été exécutée par au moins un exemple.

Un fichier à remplir est proposé pour chaque exercice.

## 2 Définir des fonctions utilisant les opérations arithmétiques

### Exercice 7 (Moyenne, \*)

Fichier : *Moyenne.java*. Écrivez une fonction *moyenne* qui reçoit 5 paramètres et affiche leur somme et leur moyenne entière.

**Contrat:**

*moyenne(10,12,8,9,19)* affichera :

58

11

□

### Exercice 8 (Coup de chaud, \*)

Fichier : *Temperature.java*. Écrivez une fonction *fahrenheit* qui reçoit une température en degrés Celsius et affiche la température en degrés Fahrenheit correspondante. On rappelle la formule :

$$f = \frac{9c}{5} + 32$$

où *c* est la température en degrés Celsius et *f* la température en degrés Fahrenheit. Vérifiez que 100°C correspond bien à 212°F.

□

### Exercice 9 (Ô temps!, \*)

Fichier : *LapinBlanc.java*. Écrivez une fonction *secondes* qui prend un nombre entier *s* en entrée et l'affiche en heures, minutes, secondes.

**Contrat:**

*secondes(3725)* affichera :

1

2

5

□

### Exercice 10 (Alea jacta est., \*\*)

Fichier : *Alea.java*. La fonction *randInt(n,m)* fournie permet de tirer un nombre entier *a* pseudo-aléatoire avec  $n \leq a \leq m$ .

1. Écrire une fonction *de* qui ne prend pas de paramètre et renvoie le résultat du lancer d'un dé cubique non pipé.
2. Écrire une fonction *yams* qui simule le lancer de 5 dés cubiques et affiche leurs résultats dans l'ordre croissant.

□

### 3 Utiliser des boucles pour répéter ou accumuler

#### Exercice 11 (Compte à rebours, ★)

Fichier : *Decompte.java*. Écrire une fonction `rebours` qui reçoit un entier  $n$  et affiche un compte à rebours de  $n$  à 0.

**Contrat:**

`rebours(3)` affichera :

3

2

1

0

□

#### Exercice 12 (Somme des entiers, ★)

Fichier : *Somme.java*. Écrire une fonction `somme` qui renvoie la somme des entiers jusqu'à son paramètre.

**Contrat:**

$somme(n) = 1 + 2 + 3 + \dots + n$ .

□

#### Exercice 13 (Factorielle, ★)

Fichier : *Factorielle.java*. Écrire une fonction `factorielle` qui renvoie le produit des entiers jusqu'à l'entier entré en paramètre. Ainsi,  $factorielle(5) = 1 \times 2 \times 3 \times 4 \times 5 = 120$ .

On rappelle que par définition,  $factorielle(0) = 1$ .

□

#### Exercice 14 (Fibonacci, \*\*)

Fichier : *Fibonacci.java*. En 1202, le mathématicien Leonardo Fibonacci inventa l'énigme suivante : Un homme met un couple de lapins dans un lieu isolé de tous les côtés par un mur. Combien de couples obtient-on en un an si chaque couple engendre tous les mois un nouveau couple à compter du troisième mois de son existence ?

1. Si l'on connaît le nombre de lapins au mois  $n$  et au mois  $n+1$ , comment connaître le nombre de lapins au mois  $n+2$  ?
2. Résoudre l'énigme.
3. Plus généralement, écrire une fonction `fibonacci` qui calcule le nombre de lapins au bout d'un nombre  $n$  de mois donné en argument.

□

#### Exercice 15 (Il est pas pur, mon alliage ?, ★)

Fichier : *Bijoutier.java*. En ces temps de crise, rappelons utilement que le carat des bijoutiers mesure la pureté d'un alliage contenant un métal précieux : un carat représente un vingt-quatrième de la masse totale d'un alliage, de sorte que 24 grammes d'un alliage d'or à 16 carats contiennent 16 grammes d'or.

Écrivez une fonction `carat` qui attend deux entiers `alliage` et `metal` représentant la masse d'alliage et la masse de métal précieux qu'elle contient (exprimées en tonnes) et qui affiche la pureté (en carats) de l'alliage.

□

#### Exercice 16 (Conjecture de Syracuse, \*\*\*)

Fichier : *Syracuse.java*. On définit récursivement la suite de Syracuse d'un entier  $a$  par  $u_0(a) = a$  et  $u_{n+1}(a) = 3u_n(a) + 1$  si  $u_n(a)$  est impair et  $u_{n+1}(a) = \frac{u_n(a)}{2}$  si  $u_n(a)$  est pair.

1. Écrire une fonction `syracuse` qui attend deux entiers  $a$  et  $n$  en entrée et qui renvoie la valeur du terme  $u_n(a)$ .
2. Une conjecture ouverte affirme que quelque soit la valeur de l'entier  $a$ , la suite finira toujours pas atteindre 1. Écrire une fonction `conjecture` qui attend un entier en paramètre et renvoie le numéro du premier indice  $k$  tel que  $u_k(a) = 1$ .

3. *Vol en altitude* : Écrire une fonction `vol` qui attend un entier  $a$  en paramètre, et qui renvoie le premier indice  $j$  tel que  $u_j(a) \leq a$ .
4. *Altitude maximale* : Écrire une fonction `altitude` qui renvoie la valeur maximale de la suite  $(u_n(a))_n$ . Attention ! Pour de petites valeurs de  $a$ , l'altitude maximale peut-être atteinte après l'indice  $k$  tel que  $u_k(a) = 1$ . Par exemple,  $u_0(1) = 1$  mais  $u_1(1) = 4$ .

□

### Exercice 17 (Scrutin de liste, \*\*\*)

Fichier : `Election.java`. On s'intéresse au résultat d'une élection dans un scrutin de liste dans lequel il y a  $n$  sièges à pourvoir et 3 listes candidates ayant obtenu  $v_1, v_2$  et  $v_3$  voix.

Pour cette élection, les règles sont les suivantes :

1. Une liste ayant moins de 5% des voix ne peut obtenir aucun siège. Les sièges sont alors répartis comme si ses électeurs s'étaient abstenus.
2. Si  $n > 10$ , la liste arrivée en tête obtient un bonus de 10 sièges. Sinon, la liste arrivée en tête obtient tous les sièges.
3. Les  $n - 10$  sièges restants doivent être répartis proportionnellement au nombre de voix obtenues.
4. Les sièges restants sont répartis au hasard, la probabilité d'attribution correspondant au taux de voix obtenues.

Le but de cet exercice est de calculer le nombre de sièges obtenus par chaque liste. Nous allons avancer progressivement.

1. Écrivez une fonction proportionnelle qui prend quatre arguments `siege`, `v1`, `v2`, `v3` et qui affiche le nombre de sièges attribués proportionnellement à chaque liste ainsi que le nombre de sièges sans attribution (dans le cas où la division ne tomberait pas juste).

2. Modifiez la fonction proportionnelle pour attribuer les sièges restant au hasard.

Vous utiliserez la fonction `randInt` qui prend en paramètres deux entiers `n1` et `n2` et renvoie un entier aléatoire compris entre `n1` et `n2` inclus.

**Indication.** Pour choisir au hasard un député parmi les 3 listes, on peut ranger les députés en rang un par un, en regroupant côte-à-côte les députés d'une même liste. On tire ensuite un numéro de député au hasard parmi tous les numéros possibles, et on choisit le député du rang correspondant. Le nombre de députés de chaque liste permet alors de connaître son camp. Après avoir compris cette méthode et fait quelques essais sur une feuille de papier, vous pouvez l'adapter en JAVA.

3. À partir de la fonction proportionnelle, écrivez une fonction `election` qui ajoute 10 sièges à la liste arrivée en tête.
4. Modifiez la fonction `election` pour éliminer toute liste qui aurait réalisé un score inférieur à 5%.
5. N'oubliez pas de tester votre fonction !

□