Introduction à JAVA

Séance A de cours/TD

Université de Paris

2021-22

Objectifs:

- Comprendre les différences entre PYTHON et JAVA.
- Identifier et donner un sens aux différentes constructions du langage (déclaration et utilisation des variables, boucles for, conditionnelles, procédures et fonctions).
- Découvrir les types `int`, `char`, `boolean` et `String`.
- Maîtriser les expressions booléennes dans les conditions.
- Comprendre l'utilisation des boucles avec dépendances.
- Manipuler des boucles avec accumulateurs ou imbriquées

1 De Python à Java

Qu'est-ce qu'un programme et un ordinateur ? [Cours]

— Un **programme** est la représentation d'une suite d'instructions à exécuter. Par exemple, une recette est un programme exécuté par un cuisinier. Un code source Java est un programme exécuté par un ordinateur.
— En première approximation, un **ordinateur** est une machine à calculer munie d'une mémoire et connectée à des périphériques (comme un écran, une carte réseau, etc).
Il peut **faire des calculs** qui produisent des **valeurs** ou **transmettre** des valeurs à sa mémoire ou à des périphériques.

Les constituants d'un programme [Cours]

— Les **expressions** décrivent des calculs à faire. Par exemple, `j*i <=n`.
— Les **instructions** décrivent des actions à effectuer. Par exemple, `if` (section 5), `for` (section 6), `while` (section 14) ou encore `return` (section 7).
— Les **déclarations** donnent des noms et des types aux constituants du programme.

Java un langage compilé [Cours]

En Java le code doit d'abord être transformé en un code intermédiaire (appelé *bytecode*) avant d'être interprété.

Pour voir le résultat de l'exécution du programme `toto.java`, il vous faudra procéder en *deux étapes* :
— transformer le code java en bytecode (`javac toto.java` dans votre terminal favori) : c'est la compilation
— interpréter le bytecode grâce à la machine virtuelle de java (`java toto`)

Exercice 1 (Traduction, ⋆)
Comparer les programmes ci-dessous : quelles différences y a-t-il ? Classifier les parties du code source correspondant aux expressions, aux instructions et aux déclarations.

```python
// renvoie un tableau l de taille n+1 tq l[i]=True ssi i est premier
// entrée : un entier n, sortie : un tableau
def crible(n):
  l = [False] + [False] + [True] * (n-1)
  for i in range(2,n+1):
  if l[i]:
    j = 2
    while j*i <= n:
      l[j*i] = False
      j += 1
  return l
```

```java
// renvoie un tableau l de taille n+1 tq l[i]=true ssi i est premier
    public static boolean[] crible (int n) {
        boolean[] l=new boolean[n+1];
        l[0]=false;
        l[1]=false;
        for (int k=2;k<=n;k++){
            l[k]=true;
        }
        for (int i = 2; i < n+1; i++) {
            int j=2;
            while (j*i<=n){
                l[j*i]=false;
                j++;
            }
        }
        return l;
    }
```

Listing 1 – Un même code en Python et en Java

□

Exercice 2 (Traduction2, ⋆)
Comparer les programmes ci-dessous : quelles différences y a-t-il ? Classifier les parties du code source correspondant aux expressions, aux instructions et aux déclarations.

```python
def somme_carres(n):
  return sum(i*i for i in range(n+1))
```

```java
    public static int somme_carres (int n) {
        int s=0;
        for (int i=0;i<=n;i++){
            s+=i*i;
        }
        return s;
    }
```

Listing 2 – Un même code en PYTHON et en JAVA

□

2 Expressions arithmétiques (`int`)

Le type `int` [COURS]

- Les expressions sont classifiées à l'aide de **types** correspondant à la forme des valeurs qu'elles calculent.
- Le type `int` est celui des expressions qui calculent des valeurs entières (entre `-2147483648` ($= -2^{31}$) et `2147483647` ($= 2^{31} - 1$)), aussi appelées expressions arithmétiques.
- Une **expression arithmétique de type `int`** peut être :
  (a) une constante entière (`0`, `1`, `2`, `-1`, `-2`, `-2147483648`, `2147483647` ...) ;
  (b) deux expressions séparées par une opération arithmétique binaire (`1 + 2 * 3 / 4`, ...), telle que `+`, `*`, `/`, `%` ;
  (c) une expression entourée de parenthèses (`(1 + 2)`, `(1 + 2 * 3 / 4)`, ...) ;
  (d) une expression précédée du signe moins (`-2`, `-(1 + 2)`, ...).
- Les opérateurs ont des priorités relatives, par exemple les opérations {`*`, `/`, `%`} sont prioritaires sur {`+`,`-`} . À priorité égale, les opérations s'effectuent de gauche à droite. Par exemple, l'expression "`1 + 2 * 3 - 4`" est équivalente à l'expression "`(1 + (2 * 3)) - 4`" évaluée en "`(1 + 6) - 4`" puis "`7 - 4`" qui donne 3.
- Les opérations ($+$, $-$, $/$, $*$, ...) ont le sens usuel *tant que l'on reste entre les bornes du type* `int`.
- Dans le type `int`, la division $/$ est une *division entière*. Par exemple, "`31 / 7`" vaut 4 et "`-15 / 9`" vaut -1.
- L'opérateur `%` désigne le *modulo* : "`a % b`" est le reste dans la division entière de `a` par `b`.
  Par exemple, "`31 % 7`" vaut 3 car $31 = 7 \times 4 + 3$ et "`-15 % 9`" vaut -6 .

Exercice 3 (Java comme une calculatrice, ⋆)
Prévoir l'évaluation des expressions arithmétiques suivantes :

```java
6 * 7 + 3
6 * (7 + 3)
45 / 7
3 * 7 / 4
(3 * 7) / 4
(45 / 7) * 7 + 45 % 7
(1 + 2 - 3 + 4 - 5 + 6 - 7 + 8 - 9 + 10 - 11 + 12 - 13) / (1 - 2 + 3 - 4 + 5 - 6 + 7 - 8
     + 9 - 10 + 11 - 12 + 13)
```

□

DIY

Exercice 4 (Valeurs d'expressions entières, ⋆)
Donnez la valeur des trois expressions suivantes : `3 + 5 / 3` `4 * 1 / 4` `2 / 3 * 3 - 2` □

Exercice 5 (Modulo 9, ⋆⋆)
Donnez la valeur des expressions suivantes :
`18 % 9` `81 + 18 % 9` `(81 + 18) % 9`

□

3 Variables

Types [Cours]

Les types étudiés ce semestre seront

- `int` pour un nombre entier (compris entre -2147483648 et +2147483647) (vu Section 2)
- `boolean` pour un booléen
- `char` pour un caractère unicode (par exemple a) et `String` pour une chaîne de caractère
- les tableaux.

Attention, un type ne peut pas être utilisé pour un autre.

Déclaration et utilisation des variables [Cours]

- Une **variable** a un *nom*, un *type*, c'est-à-dire que ce qu'elle contiendra est d'un format prédéterminé qui ne peut être modifié en cours d'instructions, et contient une *valeur*, c'est-à-dire le résultat d'un calcul.
- Une déclaration Java est de la forme `type nom_de_variable = valeur_initiale`. Par exemple, on déclare une variable `x` de type `int` qui contient initialement le résultat du calcul `6 * 7` ainsi :

```
1 int x = 6 * 7;
```

- Dans l'exemple précédent, la valeur de la variable `x` est 42.
- Exemples de noms de variable : `x`, `y`, `foo`, `foobar42`.... Par convention, elles ne commencent pas par une majuscule. Il faut aussi éviter les mots réservés (une variable ne peut pas avoir `int` pour nom).
- On peut utiliser la valeur d'une variable dans une expression en faisant référence à son nom. Ainsi, l'expression "`x + 1`" vaut 43 si `x` vaut 42.
- On peut changer la valeur d'une variable qui a été déclarée auparavant en lui *affectant* le résultat d'un nouveau calcul. L'opérateur d'affectation est « = » :

```
1 x = 2 * 10;
```

- Sur papier, une mémoire contenant les variables `x = 42`, `y = 73` et `z = 37` sera écrite :

x	y	z
42	73	37

Exercice 6 (Nommer, ⋆)
Quelle est la valeur des variables, à la suite des instructions suivantes.

```
1 int x = 6 * 7;
2 int y = x + x;
```

□

Exercice 7 (Affectations, ⋆⋆)
Quelle est la valeur des variables, à la suite des instructions suivantes.

```
1 int x = 1;
2 int y = 4;
3 x = y + 2;
```

DIY □

Exercice 8 (Affectations, ⋆⋆)
Quelle est la valeur des variables, à la suite des instructions suivantes.

```
1 int x = 1;
2 int y = x - 1;
3 x = 2;
```

□

Exercice 9 (Affectations, ⋆⋆)
Quelle est la valeur des variables, à la suite des instructions suivantes.

```
1 int x = 1;
2 x = x - 1;
```

□

4 Expressions booléennes (`boolean`)

Le type `boolean` [Cours]

— Le type `boolean` est celui des expressions booléennes qui ne peuvent prendre que deux valeurs : `true` et `false`. Celles-ci peuvent être stockées dans des variables de type `boolean`.
— Une **expression booléenne** peut être :
  (a) un booléen (`true`, `false`)
  (b) une comparaison entre élément de type `int` (par exemple `1<=3`). Les comparateurs sont les suivants : `<`, `<=`, `>`, `>=`, `==` (aussi utilisé pour le type `char`), `!=`.
  (c) deux expressions booléennes séparées par `&&` ("et") ou `||` ("ou").
  (d) une expression booléenne précédée par `!` ("négation").
— Si `a` et `b` sont deux expression booléennes, `a && b` est vraie si et seulement si `a` et `b` le sont.
— Si `a` et `b` sont deux expression booléennes, `a || b` est vraie si et seulement si au moins `a` ou `b` l'est.
— Si `a` est une expression booléenne, `!a` est vraie si et seulement si `a` est fausse.
— La négation `!` est prioritaire sur la conjonction `&&` qui est elle-même prioritaire sur la disjonction `||`.
— Par exemple, les déclarations suivantes sont valides :

```
boolean b1 = false; // b1 vaut false.
boolean b2 = (3 + 2 < 6); // b2 vaut true.
b1 = !b2; // b1 vaut la negation de true, donc false.
b2 = b1 || b2; // b2 vaut false || true, donc true.
```

— **Attention :** le symbole `=` est utilisée dans les instructions d'affectation tandis que le symbole `==` est un opérateur de test d'égalité entre entiers.

Exercice 10 (Évaluer, ⋆)
Quelles sont les valeurs des variables `x`, `b`, `d`, `e` *et* `f` *après avoir exécuté les instructions suivantes ?*

```
int x = 3 + 5;
boolean b = (x == 5 + 3);
boolean d = (x > x + 1);
boolean e = b || d;
boolean f = b && d;
f = (e != b);
```

□

Exercice 11 (Parité, ⋆)
Écrire une expression booléenne qui vaut `true` *si un entier* `a` *est pair,* `false` *sinon.*

□

Exercice 12 (Équivalence d'expressions, ⋆⋆)
On dit que deux expressions booléennes sont équivalentes *quand ces deux expressions s'évaluent toujours vers la même valeur booléenne pour toutes les valeurs possibles des variables qui apparaissent dans ces expressions.*

Par exemple, "`x > 10 && x < 12`*" est équivalente à "*`x == 11`*". Par contre, "*`x > y`*" et "*`x != y`*" ne sont pas équivalentes.*

Dire si les expressions suivantes sont équivalentes :

1. *"*`x > y || x < y`*" et "*`x != y`*";*
2. *"*`x != 3 && x != 4 && x != 5`*" et "*`x <= 2 || x >= 6`*";*
3. *"*`x ==  y && x == z`*" et "*`x == z`*";*
4. *"*`x ==  y && x == z`*" et "*`x == y && y == z`*".*

□

Exercice 13 (Simplification des expressions booléennes, ⋆⋆)

Simplifier *une expression booléenne consiste à la remplacer par une autre expression booléenne équivalente qui est plus courte. Simplifier les expressions suivantes :*
— `x > 5 && x > 7`

— `x == y && x == z && y == z`
— `x == 17 || (x != 17 && x == 42)`
— `x > 5 || (x <= 5 && y > 5)`

□

5 Instruction conditionnelle

Instruction conditionnelle [Cours]

— Une instruction conditionnelle permet d'exécuter des instructions en fonction d'une expression booléenne.
— Elle est de la forme

```
if (expression_booléenne) {
    instructions1
} else {
    instructions2
}
```

pour effectuer `instructions1` si `expression_booléenne` est vraie et `instructions2` dans le cas contraire. On peut omettre le `else` s'il n'y a pas d'`instructions2`.
— Il est possible de considérer des disjonctions de cas grâce à `else if`, ou en utilisant l'opérateur `switch` (voir Section 15)

Exercice 14 (Le max, ⋆)
À la suite des instructions ci-dessous, quelle est la valeur de la variable `max` *?*

```
int x = 3;
int y = 4;
int max = 0;
if (x > y) {
    max = x;
} else {
    max = y;
}
```

Transformez la suite d'instructions pour calculer le minimum de `x` *et* `y` *et le mettre dans une variable* `min`.

□

Exercice 15 (Différents tests, ⋆⋆)
Quelle est la valeur des variables `a`, `b` *et* `c` *après la suite d'instructions suivante :*

```
int a = 2;
int b = a * a + 3;
int c = b - a;
if (c == a) {
    a = 1;
} else {
    a = a + 3;
}
if (b + c < a) {
    b = 2;
} else {
    b = 4;
}
if (b != c * c) {
    c = 12;
} else {
    c = -6;
}
```

□

Exercice 16 (Savoir évaluer une condition, ⋆)
Quelle est la valeur de la variable `x` *après la suite d'instructions suivante ?*

```
int a = 2;
```

```
a = a * a * a * a + 1;
int b = a / 2;
int c = a - 1;
int x = 0;
if ((b == 0 ) && (c == 16)) {
    x = 1;
} else {
    x = 2;
}
```

□

DIY

Exercice 17 (Valeur absolue, ⋆)
En vous inspirant de l'exercice 14, donner une suite d'instructions permettant de calculer la valeur absolue d'une variable. □

Exercice 18 (Condition et division entière, ⋆)
Écrire une condition permettant de tester si le tiers de l'entier `x` *appartient à l'intervalle* $[2; 8]$. □

Exercice 19 (Condition, ⋆)
Écrire une condition permettant de tester si les entiers `a`, `b` *et* `h` *peuvent correspondre aux longueurs des côtés d'un triangle rectangle, où* `h` *serait la longueur de l'hypothénuse. C'est le cas par exemple si* $(a, b, h) = (3, 4, 5)$, *mais pas si* $(a, b, h) = (1, 2, 3)$.- □

6 Boucles

Boucles [Cours]

— Une boucle permet de répéter plusieurs fois les mêmes instructions.
— Elle est de la forme

```
for (initialisation; condition; incrémentation) {
    instructions
}
```

— Le numéro de l'itération est disponible dans une variable qui s'appelle le *compteur de boucle* qui est déclarée, initialisée et incrémentée dans l'en-tête de la boucle. La boucle s'arrête quand la `condition` n'est plus vérifiée.
— Le *corps de la boucle*, entre accolades, est exécuté à chaque itération de la boucle.
— L'*instruction d'incrémentation* peut être de la forme `i++` (équivalent à "`i = i + 1`"), `i--` (équivalent à "`i = i - 1`"), ou encore `i+=n` (équivalent à "`i = i + n`"), pour `n` un entier quelconque donné.
— Par exemple, la boucle suivante affiche les entiers de 0 à 9 :

```
for (int i = 0; i <= 9; i++) {
    System.out.println (i);
}
```

— Nous verrons qu'il existe un autre type de boucle introduite par le mot-clé `while`.

Exercice 20 (Afficher des suites d'entiers, ⋆)

1. *Écrivez une boucle qui affiche les 100 premiers entiers, en commençant à 0.*
   Quel est le dernier entier affiché ?
2. *Écrivez une boucle qui affiche les 50 premiers entiers pairs, en commençant à 0.*
   Quel est le dernier entier affiché ?
3. *Écrivez une boucle qui affiche 1000 fois le nombre 3.*

□

Exercice 21 (Des entiers qui s'ajoutent, ⋆⋆)
Quelle est la valeur de `sum` *après la suite d'instructions suivante :*

```
int i = 0;
int sum = 0;
sum = sum + i;
```

```
i = i + 1;
sum = sum + i;
i = i + 1;
sum = sum + i;
i = i + 1;
sum = sum + i;
i = i + 1;
sum = sum + i;
i = i + 1;
sum = sum + i;
i = i + 1;
```

Si on veut adapter le code ci-dessus pour aller non plus jusqu'à 5 mais jusqu'à 100, 1000 ou plus, on a un problème : on obtiendrait un programme beaucoup trop long !

On peut remplacer la longue liste d'instructions par une boucle `for` *en remarquant que l'instruction "*`sum = sum + i;`*" est exécutée 10 fois avec* `i` *prenant pour valeurs les différents entiers entre 1 et 5 car la variable est systématiquement incrémentée (sa valeur est augmentée de 1) grâce à l'instruction "*`i = i + 1;`*".*

On obtient ainsi la boucle :

```
int bound = 5;
int sum = 0;
for (int i = 0; i <= bound; i++) {
    sum = sum + i;
}
```

1. *On remplace la première ligne du code précédent par "*`int bound = 100;`*". Quelle est la valeur de* `sum` *après l'exécution de cette suite d'instructions ? Même question si on remplace la première ligne par "*`int bound = 1000;`*" ?*
2. *Modifiez le code précédent pour calculer la somme des entiers de* 10 *à* 100.

□

DIY

Exercice 22 (Boucles simples, ⋆⋆)

1. *Quel est l'affichage produit par la suite d'instructions suivante :*

```
for (int i = 0; i < 5; i++) {
    System.out.println (8);
}
```

2. *Quelle est la valeur de* `x` *après la suite d'instructions suivante :*

```
int x = 0;
for (int i = 0; i < 5; i++) {
    x = x + 8;
}
```

3. *Quelle est la valeur de* `x` *après la suite d'instructions suivante :*

```
int x = 0;
for (int i = 0; i < 5; i++) {
    x = 10 * x + 8;
}
```

□

Exercice 23 (Somme des cubes, ⋆)
Écrire une boucle permettant de calculer la somme des cubes des entiers de 1 à 100. □

7 Fonctions et procédures

Qu'est-ce qu'une fonction ? [Cours]

— Une fonction est une série d'instructions appliquée à des paramètres qui *renvoie une valeur*.
— Les fonctions permettent de rendre un programme plus compréhensible en augmentant le vocabulaire du langage : on donne un nom à une série d'instructions calculant une certaine valeur et il n'est alors plus nécessaire de se souvenir de cette série d'instructions car on peut utiliser le nom de fonction introduit comme si il s'agissait d'une nouvelle primitive du langage.
— Les fonctions permettent de factoriser du code : plutôt que d'écrire plusieurs fois la même série d'instructions (avec éventuellement quelques variations artificielles), on écrit une seule fois ces instructions en *explicitant des paramètres* pour représenter les variations. Par exemple, à la place de :

```
int z = y * y + (1 + y) * (1 + y) + (2 + y) * (2 + y);
```

On peut introduire la fonction `square` suivante :

```
public static int square (int x) {
    return x * x;
}
```

et remplacer la déclaration de la variable `z` par :

```
int z = square (y) + square (1 + y) + square (2 + y);
```

— On peut utiliser des fonctions prédéfinies dans des *bibliothèques de fonctions*.
— On fait référence à une procédure écrite dans une bibliothèque en utilisant une notation "pointée". Par exemple, `System.out.println` est le nom de la procédure `println` de `System.out`.
— Une fois définie, il est possible d'*utiliser la fonction dans toute expression*, on parle alors d'**appel** (où d'**invocation**) d'une fonction. On peut par exemple écrire :

```
int a = 3 + power (4, 2);
```

Dans cette déclaration, la sous-expression « `power (4, 2)` » est remplacée par la valeur retournée par le corps de la fonction `power` pour `base` valant 4 et `exponent` valant 2, c'est-à-dire 16. On évalue ensuite $3+16$ pour obtenir la valeur 19 qui sera la valeur initiale de `a`.
— L'appel d'une fonction doit respecter le prototype de cette fonction, c'est-à-dire le nombre et les types des paramètres ainsi que le type de la valeur de sortie.
— Par exemple, la fonction « `void factorial (int n)` » ne peut pas être utilisée avec un argument de type `boolean`. Ainsi, écrire "`factorial (true)` provoquera une **erreur de typage** au moment de la compilation.

Syntaxe des fonctions [Cours]

— Une fonction est caractérisée par :
  — **un corps** : le bloc d'instructions qui décrit ce que la fonction calcule ;
  — **un nom** : par lequel on désignera cette fonction ;
  — **des paramètres** (l'*entrée*) : l'ensemble des variables extérieures à la fonction dont le corps dépend pour fonctionner et qui prennent une valeur au moment de l'appel de la fonction ;
  — **une valeur de retour** (la *sortie*) : ce que la fonction renvoie à son appelant.

  Par exemple, le code du programme `ExoExecution` ci-dessous définit une fonction de nom `power`, qui prend en paramètre deux valeurs entières `base` et `exponent`, de type `int`, et renvoie en sortie une valeur de type `int` qui vaut $\texttt{base}^{\texttt{exponent}}$:

```
    /* Retourne base élevée à la puissance exponent. */
    public static int power (int base, int exponent) {
        int result = 1;
        while (exponent > 0) {
            result = result * base;
            exponent = exponent - 1;
        }
        return result;
    }
```

— La première ligne qui déclare la fonction est appelée **en-tête** ou **prototype** de la fonction et précise le type de sa valeur de retour, son nom et les types et les noms de chacun de ses paramètres.
— Le sens précis de `public` et `static` seront traités dans le cours de programmation orientée objet en Java.
— Le choix des noms de la fonction et de ses paramètres fournit une documentation minimale du comportement de la fonction, il est donc souhaitable qu'ils soient parlant. Par exemple, le programme est moins compréhensible si on remplace l'en-tête de `power` par « `public static int function1 (int x, int y)` ».

— Dans le corps de la fonction se trouvent les instructions qui détaillent la manière dont la valeur de sortie est calculée. Ces instructions utilisent les paramètres de la fonction (`base` et `exponent`) et, éventuellement, de nouvelles variables déclarées localement dans le corps (dans notre exemple, on utilise la variable `result`).

Zoom sur `return` [Cours]

— La valeur retournée par la fonction est indiquée par l'instruction

```
return expression ;
```

où `expression` a le même type que celui de la valeur de sortie déclaré dans l'en-tête de la fonction.

— L'instruction `return` fait deux choses :

1. elle précise la valeur qui sera fournie par la fonction en résultat,
2. elle met fin à l'exécution des instructions de la fonction.

— Il est possible d'avoir plusieurs occurrences de `return` dans le même corps, comme dans la fonction suivante :

```
public static int minimum (int a, int b) {
    if (a < b) {
        return a;
    } else {
        return b;
    }
}
```

— On doit s'assurer que tout chemin d'exécution possible du corps d'une fonction mène à un `return`, sinon Java rejette le programme. Par exemple :

```
public static int minimum (int a, int b) {
    if (a < b) {
        return a;
    }
}
```

produit l'erreur suivante à la compilation :

```
ExoInvalidMinimum.java:10: missing return statement
```

Exercice 24 (Utiliser une fonction, ⋆)
Étant donnée la fonction suivante, que vaut l'expression `twice(3)` *?*

```
1 public static int twice (int x) {
2     return (2 * x);
3 }
```

□

Exercice 25 (Utiliser une fonction, ⋆⋆)
On considère la fonction suivante.

```
1 public static int cube (int x) {
2     return (x * x * x);
3 }
```

Quelle est la valeur de la variable `a` *à la suite des instructions suivantes ?*

```
1 int b = 5;
2 int a = 3;
3 a = b - a;
4 a = cube(a);
```

□

Exercice 26 (Écrire une fonction, ⋆⋆)
Voici une fonction

```
1 public static int f(int x) {
```

```
    return (x * x + 5);
}
```

Quel est son nom ? Que calcule-t-elle ? Modifier son corps pour qu'elle calcule la division entière par 3. Modifier son nom pour qu'elle se nomme third.

□

Exercice 27 (Écrire une fonction, ⋆⋆)
Écrire des fonctions effectuant les calculs suivants :

1. *La puissance 5 d'un entier donné en paramètre.*
2. *Le produit de deux entiers donnés en paramètre moins leur somme.*
3. *Le produit au carré de trois entiers donnés en paramètre.*

□

Exercice 28 (Évaluer un appel, ⋆)
Soit `power` *la fonction définie dans le programme* `ExoExecution` *ci-dessus, quelle est la valeur de la variable* `x` *après exécution de la suite d'instructions qui suit :*

```
int x = 2;
int y = 2;
x = power (x, y);
x = power (x, y);
x = power (x - 8, y);
```

□

Exercice 29 (Appelle-moi bien !, ⋆)
Soient « `public static int power(int base, int exponent)` *» un prototype de fonction et* `x, y` *deux variables de type* `int`*. Trouver les appels incorrects parmi ceux de la liste suivante. Motiver votre réponse.*

1. *«* `y = power (2, x);` *»*
2. *«* `int a = power ("2", 3);` *»*
3. *«* `String a = power (2, x);` *»*
4. *«* `if (power (2, x) > 5) { ... }` *»*
5. *«* `while (power (2, x)) { ... }` *»*
6. *«* `x = power (x, power (x, x));` *»*
7. *«* `x = power(x, System.out.print (y));` *»*

□

Exercice 30 (Mauvais en-tête, ⋆⋆)
Dire pourquoi aucun des en-têtes suivants n'est correct.

1. *«* `public static int fonc1 (int a; int b)` *»*
2. *«* `public static fonc2 (boolean c)` *»*
3. *«* `public static int, fonc3 (String w)` *»*
4. *«* `public static boolean fonc4 (int)` *»*
5. *«* `public static String fonc5 (String t, int d, int t)` *»*
6. *«* `public static int 6fonc (int x)` *»*

□

Exercice 31 (En-tête déduite d'une spécification, ⋆)
Donner l'en-tête d'une fonction qui prend en paramètre une chaîne de caractères et renvoie sa longueur. □

Exercice 32 (En-tête déduite d'un corps de fonction, ⋆⋆)
Donner l'en-tête de la fonction dont le corps est défini comme suit. (La fonction « `public static int intArrayLength (int [] t)` *» prend en paramètre un tableau d'entiers et renvoie sa longueur.) :*

```
... {
    int r = 0;
    int i = 0;

    if (intArrayLength (t) != intArrayLength (u)) {
        return -1;
    }

    while (i < intArrayLength (t)) {
```

```
        r = r + t[i] * u[i];
        i = i + 1;
    }

    return r;
}
```

□

Les procédures [Cours]

— Les **procédures** sont des suites d'instructions qui ne renvoient pas de valeur mais qui produisent des *effets* (affichage, modification des valeurs en mémoire, ...) sur le système qui les exécute. Par exemple, la procédure « `void System.out.print(String s)` » affiche un message à l'écran.

— Les procédures ont un en-tête et un corps, mais le type de leur valeur de retour est `void` (*vide* en français) et le corps ne contient donc pas obligatoirement l'instruction `return`. Par exemple :

```
public static void printRange (int from, int to) {
    for (int i = from; i <= to; i++) {
        System.out.print (i);
    }
}
```

— La procédure `main` a un rôle particulier car elle contient les premières instructions à exécuter lorsque le programme est lancé : on dit que `main` est le *point d'entrée* du programme.

— Le prototype de `main` est obligatoirement « `public static void main(String [] args)` ». On verra par la suite à quoi sert le tableau de chaînes de caractères `args`.

Exercice 33 (Première procédure, ⋆)
Que fait la procédure suivante ?

```
public static void showProductAndSum (int x, int y) {
  System.out.println (x + y);
  System.out.println (x * y);
}
```

□

Exercice 34 (Différences syntaxiques, ⋆⋆)

1. *Quelles différences trouvez-vous entre la syntaxe de déclaration des fonctions et des procédures ?*
2. *Même question pour l'appel de fonctions et l'appel de procédures.*

□

DIY

Exercice 35 (Utiliser une fonction, ⋆)
On considère la fonction suivante.

```
public static int sumsquare (int x, int y) {
    return (x * x + y * y);
}
```

Quelle est la valeur des variables `a`, `b` *et* `c` *à la suite des instructions suivantes :*

```
int a = sumsquare(2, 3);
int b = sumsquare(3, 1);
int c = sumsquare(4, 3);
```

□

8 Caractères et chaînes de caractères : les types `char` et `String`

Les caractères et chaînes de caractères [Cours]

— Le type `char` correspond à un caractère. Ceux-ci sont entourés par des guillemets simples (`'`). Un caractère peut être une lettre (par ex. `'a'`, `'b'`), un chiffre (par ex. `'1'`), un signe de ponctuation (par ex. `'?'`) ou encore un caractère spécial (par ex. `'\n'`, `' '`) .
— On utilisera `'\''` pour ' et `'\\'` pour \. On pourra aussi utiliser `'\u273F'` (code unicode sur 4 bits).
— Deux caractères peuvent être comparés avec ==.
— Attention, par définition une donnée de type `char` ne contient qu'un seul caractère ! On ne peut pas écrire `'sy'` par exemple.
— Le type `String` correspond aux chaînes de caractères.
— Une chaîne de caractère peut être :
(a) du texte entre guillemets doubles (`"Mr Robot"`, `""`, `"###\n# #"`)
(b) deux chaînes de caractères séparées par + (concaténation)
— La **longueur** d'une chaîne de caractère est son nombre de caractères (0 pour la chaîne `""`). La longueur d'une chaîne de caractères stockée dans une variable `s` est donnée par la fonction `s.length()`.
— Les caractères d'une chaîne de caractères sont numérotés de 0 à n-1 où n est la longueur de la chaîne. Pour obtenir le (i + 1)-ème caractère d'une chaîne de caractères stockée dans une variable `st`, on peut utiliser la fonction `st.charAt(i)`. Attention il faut que i soit dans l'intervalle $[0;$`st.length`$[$.
— Par exemple, après la suite d'instructions suivantes :

```
String ch = "Hello !" ;
char c1 = ch . charAt (0) ;
char c2 = ch . charAt (2) ;
char c3 = ch . charAt ( ch . length () -1) ;
```

le caractère stocké dans la variable c1 est `'H'`, celui stocké dans la variable c2 est `'l'` et celui stocké dans la variable c3 est `'!'`.
— On peut ajouter un caractère à la suite ou au début d'une chaîne de caractères en utilisant le symbole de concaténation +. Par exemple `"Salu"+'t'` donnera la chaîne `"Salut"`. De même `'S'+"alut"` donnera aussi la chaîne `"Salut"`.
— Mais on NE peut PAS concaténer deux caractères. Ainsi, l'instruction `'a'+'b'` ne produira pas la chaîne `"ab"`.
— Pour introduire un guillemet dans une chaîne, on écrit `\"`.
— Dans une chaîne, la séquence `\n` représente le passage à une nouvelle ligne.
— Attention, on ne peut pas tester l'égalité de deux chaînes de caractères avec ==. Pour cela, on utilise dans les tests la fonction "`s1.equals(s2)`".

Exercice 36 (De premiers exemples, ⋆)
Qu'est-ce qui est affiché après l'exécution des instructions suivantes ? Que valent les variables en mémoire ?

```
String s = " rentre chez lui.";
String s2 = "Paul";
String s3 = "Michel";
boolean b1=s2.equals("Paul");
boolean b2=s3.equals("Paul");
System.out.println(s3 + s);
String s4 = s2 + " et " + s3 + " rentrent chez eux.\n";
int n=s2.length()
System.out.println(s4.length())
```

□

Exercice 37 (Manipulation de chaînes, ⋆)
1. Quelle est la valeur de la variable a après les instructions suivantes ?

```
String ch = " Ceci est " ;
String ch2 = " une chaine " ;
ch = ch + ch2 ;
int a = ch . length () ;
```

2. Que vaut la variable cha dans l'exemple suivant ?

```
String ch = " Avec consonnes " ;
String cha = " " + ch . charAt (0) ;
cha = cha + ch . charAt (2) ;
cha = cha + ch . charAt (6) ;
cha = cha + ch . charAt (9) ;
cha = cha + ch . charAt (12) ;
```

□

Différence entre affichage et calcul [Cours]

— Il ne faut pas confondre l'affichage d'une chaîne de caractères sur le terminal et le calcul d'une chaîne.

```
System.out.println ("Omar");
```

affiche la ligne `Omar` sur le terminal tandis que le programme suivant n'affiche rien :

```
String  a = "Omar" + "Roma";
```

— mais initialise une variable a avec `"OmarRoma"`, le résultat de la concaténation des deux chaînes `"Omar"` et `"Roma"`.

Exercice 38 (Des exemples simples avec des variables de différents type, ⋆)

1. *Quelle est la valeur de la variable* `ch` *après l'exécution des instructions suivantes ?*

```
int x = 3;
String ch = "Salut";
if (x <= 2) {
    ch = "Hallo";
} else {
    System.out.println("Hi");
}
```

2. *Que fait la suite d'instructions suivantes ?*

```
int x = 3;
String st = "Ca va ?";
String ch = "";
x = x / 2;
if (x == 1) {
    ch = "How are you ?";
} else {
    ch = "Comment allez-vous ?";
}
ch = ch + "\n";
System.out.print (ch);
```

□

Exercice 39 (Deux représentations très différentes de la même chose, ⋆⋆)

— *Que vaut l'expression* `"41" + "1"` *?*
— *Que vaut l'expression* `41 + 1` *?*
— *Quelle différence faites-vous donc entre la chaîne de caractères* `"42"` *et l'entier* 42 *? Dans quelles situations utiliser l'une ou l'autre de ces représentations de l'entier* 42 *?*

□

DIY

Exercice 40 (Concaténation et somme, ⋆⋆)
Que valent les variables `x` *et* `s` *après les instructions suivantes ?*

```
int x = 0;
String s = "";
for (int n = 0; n < 10; n++) {
    x = x + 1;
    s = s + "1";
}
```

□

9 Fonctions avec le type `String`

Les fonctions peuvent utiliser le type `String` [Cours]

— Une fonction peut prendre en paramètres des valeurs de type `String` et en renvoyer. On considère les deux fonctions données ci-dessous :

```
public static String sandwich (String s, String s2) {
    return s + s2 + s;
}

public static String voiePoudlard (int x) {
    String st = "";
    if (x <=9) {
        st = "La voie du train est après.";
    } else {
        st = "La voie du train est avant.";
    }
    return st;
}
```

Elles peuvent être utilisées de la façon suivante :

```
String s = sandwich ("Hello", "World!\n");
String s2 = sandwich (s, "Ca va ?");
String s3 = voiePoudlard(1);
```

— La fonction `String.valueOf(x)` prend en argument un élément de type `int` et renvoie un élément de type `String`. Par exemple, `System.out.println(String.valueOf(123));` renverra `"123"`.

— De même on peut transformer une chaîne de caractères représentant un entier en un entier grâce à la fonction `int Integer.valueOf(String ch)`. ATTENTION : si l'on donne en argument à cette fonction une chaîne de caractères ne correspondant pas à la représentation d'un entier alors il y aura une erreur à l'exécution du programme ! C'est ce qui se passera par exemple avec l'instruction `Integer.valueOf("Boum")`.

Exercice 41 (Utilisation de fonctions données, ⋆)
On considère la fonction suivante :

```
public static String addA (String ch) {
    return (ch + "A");
}
```

Que vaut la variable `st` *après exécution du code suivant ?*

```
String s = "Ma lettre finale est ";
String st = addA (s);
```

□

Exercice 42 (Modification d'une fonction, ⋆⋆)

```
public static String message (int x) {
    return String.valueOf(x);
}
```

Et on considère la liste d'instructions suivante :

```
int a = 4;
int b = a * a;
a = b - a;
String s = message (a);
System.out.println (s);
```

1. *Quelles sont les valeurs des variables de ce programme à la fin de son exécution ?*
2. *Qu'affichent ces instructions ?*

3. *Est-il possible de modifier la fonction "*`String message (int x)`*" de telle sorte que le programme affiche à la fin "Le résultat du calcul est n." (où n est la valeur contenue dans l'argument transmis à* `message`*) ?*
4. *Définissez une fonction "*`int square (int n)`*" qui calcule le carré du nombre donné en paramètre et utilisez-la pour afficher une ligne du type $n * n = n^2$ (où n sera remplacé par sa valeur).*

□

Exercice 43 (Trouver le bon type, ⋆⋆⋆)
Dans les fonctions suivantes, remplacez les `[...]` *par les bons types :*

```java

public static [...] f ([...] x) {
    [...] y = x % 2 ;
    if (y == 0) {
        System.out.println ("Argument pair");
    } else {
        System.out.println ("Argument impair");
    }
}

public static [...] g ([...] x) {
    [...] y = x % 2 ;
    if (y == 0) {
        return "Argument pair";
    } else {
        return "Argument impair";
    }
}

public static [...] h ( [...] x, [...] y) {
    return (x + y);
}

public static [...] id ( [...] x) {
    return x;
}
```

code/ExoInference.java

□

DIY

Exercice 44 (Concaténation, ⋆)
Écrire une fonction qui prend en paramètre une chaîne de caractères `toto` *et affiche une ligne commençant par* `bonjour,` *suivi du contenu de* `toto`*. Si* `toto` *a la valeur* `"toi"`*, l'appel à la fonction doit afficher "* `bonjour, toi`*".* □

Exercice 45 (Pyramide, ⋆⋆⋆)

Écrire une fonction qui prend en entrée un entier n et affiche une pyramide d'étoiles de hauteur n. Si n est négatif, la pyramide devra être inversée. Par exemple, sur entrée $n = 4$, afficher :

```
   *
  * *
 * * *
* * * *
```

Sur entrée $n = -3$, afficher :

```
* * *
 * *
  *
```

□

10 Instructions composées

Instructions composées [Cours]

— Certaines des instructions présentées sont des instructions *composées* : elles peuvent contenir d'autres instructions. C'est le cas des instructions conditionnelles et des boucles.
— On a le droit d'imbriquer des instructions dans d'autres instructions à volonté : des conditionnelles dans des boucles dans des conditionnelles, etc ...
— Avec des instructions imbriquées sur plusieurs niveaux, il est absolument indispensable (dans l'intérêt des lecteurs humains) de suivre strictement une discipline d'indentation du code source.

Exercice 46 (Premières imbrications, ⋆)
Quelle est la valeur de z à la fin de l'exécution des instructions suivantes ?

```
int x = 1;
int y = 2;
int z = 0;
if (x == 1) {
    if (y == 2) {
        z = 3;
    } else {
        z = 5;
    }
} else {
    z = 7;
}
```

□

Exercice 47 (Utilisation des opérateurs booléens, ⋆⋆)
Réécrire les suites d'instructions suivantes en utilisant moins de tests `if`.

1.

```
public static void f (int l) {
    int x = 0;
    if (l != 0) {
      if (l <= 10) {
        x = 1;
      } else {
        x  = 2;
      }
    } else {
        x = 2;
    }
}
```

Quelle est la valeur finale de `x` *pour des valeurs de* `l` *dans* $\{0, 5, 15\}$ *? Vérifier que votre réécriture du code est compatible.*

2.

```
public static void g (int a) {
    int b = 2 * a;
    b = a - a;
    if (b == a) {
        b = 0;
    } else {
      if (b > 43) {
        b  = 0;
      } else {
        b = 1;
      }
    }
}
```

Quelle est la valeur finale de `b` *pour des valeurs de* `a` *dans* $\{0, 25, 50\}$ *? Vérifier que votre réécriture du code est compatible.*

3.

```
public static void h (int c, int d) {
    int e = 0;
    if (d == 6) {
        e = 3;
```

```
    } else {
       if (c >= 2){
        e = 2;
      } else {
        e = 3;
      }
    }
}
```

Quelle est la valeur finale de `e` *pour de valeurs de* (`c`,`d`) *dans* $\{(0,0),(0,6),(6,0),(6,6)\}$ *? Vérifier que votre réécriture du code est compatible.*

□

Exercice 48 (Boucles et conditionnelles, ⋆)
Que fait le programme suivant ?

```
for(int i=0;i<10;i++){
    if (i%2==0){
        System.out.println("Pair");
    }else{
        System.out.println("Impair");
    }
}
```

□

Exercice 49 (Pendu à l'envers, ⋆⋆)

1. *Écrire une fonction qui prend en argument une chaîne de caractères et affiche une suite de tirets (-) de même longueur. Par exemple, elle renverra* `----------` *si son argument est* `"amphitheatre"`.
2. *Écrire une fonction qui prend en argument une chaîne de caractères et l'affiche à l'envers. Par exemple, elle affichera* `ertaehtihpma` *si son argument est* `"amphitheatre"`.

□

Exercice 50 (Effacement des espaces, ⋆)
Ecrire une fonction qui prend en argument une chaîne de caractères et affiche cette même chaîne sans les espaces. Si son argument est "Bonjour monde !", elle affichera "Bonjourmonde !". □

DIY

Exercice 51 (Simplification de code, ⋆)

1. *Dire, en justifiant, ce que fait la fonction suivante selon les valeurs de* x *et* y.

```
public static int f (int x, int y) {
    if (((x <= y) || (x >= y + 1))) {
        if ((x <= y) && (x < y)) {
            if (x < -x) {
                int a = -x;
                return a;
            } else {
                return x;
            }
        } else {
            if (y * y * y < 0) {
                return -y;
            }
        }
    }
    return y;
}
```

2. *Écrire la même fonction de manière plus simple.*

□

Exercice 52 (Mention, ⋆)
Écrire une conditionnelle afin de stocker dans une chaîne de caractères `s` *la mention correspondant à la note (entière) contenue dans la variable* `n` *de type* `int`.

Rappel : entre 10 inclus et 12 exclu, la mention est passable *; entre 12 inclus et 14 exclu ;* bien *entre 14 inclus et 16 exclu, et* très bien *au-delà de 16.* □

Exercice 53 (Nombres parfaits, ⋆⋆⋆)
Un entier $n > 1$ est dit parfait *s'il est égal à la somme de ses diviseurs propres (c'est-à-dire autres que lui-même). Par exemple, 6 est parfait car ses diviseurs propres sont 1, 2 et 3, et on a $6 = 1 + 2 + 3$.*

1. *Écrire une fonction* `divisors` *qui prend en entrée un entier* `n` *et affiche tous les entiers strictement inférieurs à* `n` *qui divisent* `n`.

   Contrat:
   n = 1 → affichage :
   n = 5 → affichage : 1
   n = 60 → affichage : 1 2 3 4 5 6 10 12 15 20 30

2. *Écrire une procédure* `void isPerfect(int n)` *qui prend en entrée un entier* `n` *et qui affiche* `"n est parfait"` *si* `n` *est parfait.*
3. *Écrire une procédure* `void enumPerfect(int m)` *qui prend en paramètre un entier* `m` *et affiche pour tous les nombres parfaits tels que* `n`≤`m`, `"n est parfait"` *(en remplaçant avec la valeur de* `n`*).*

□

Exercice 54 (Ordinaux anglais, ⋆⋆⋆)
On s'intéresse aux ordinaux anglais abrégés, *où le nombre (le cardinal) est écrit en chiffres :*

1st, 2nd, 3rd, 4th,..., 9th, 10th, 11th, 12th,..., 19th, 20th, 21st, 22nd, 23rd,...

Pour déterminer le suffixe, on regarde le dernier chiffre du nombre : si c'est 1, on ajoute le suffixe "st" ; si c'est 2, le suffixe est "nd" ; si c'est 3, le suffixe est "rd" ; sinon le suffixe est "th". Il y a cependant une exception : si l'avant-dernier chiffre du nombre est 1, le suffixe est toujours "th".

Écrire une fonction `printOrdinal` *qui prend un entier de type* `int` *en paramètre et affiche l'ordinal anglais abrégé correspondant.*

Contrat:
1 → 1st
12 → 12th
23 → 23rd
32 → 32nd
44 → 44th

□

Exercice 55 (conversion, ⋆⋆⋆)
Ecrire une fonction `int string_to_int(String s)` *qui prend un* `String` *et renvoie un entier de la façon suivante : cette fonction ignore tous les caractères qui ne sont pas des chiffres et construit un entier avec ce qui reste en lisant de gauche à droite. Par exemple* `string_to_int("a234-5")` *renvoie l'entier* `2345`. *On pourra pensera au fait que par exemple 1234 est égal à 123*10 + 4.* □

Exercice 56 (Lignes et pointillés, ⋆⋆)

1. *Écrivez une fonction qui prend en paramètre un entier supposé positif n et qui affiche une ligne d'astérisques « * » de longueur n puis va à la ligne. Par exemple, si n vaut 7, votre fonction affichera :*
   ****
2. *Modifiez votre fonction pour qu'elle affiche une ligne pointillée alternant astérisques et espaces. Par exemple, si n vaut 7, votre fonction affichera :*
   * * * *
   Qu'affiche votre fonction si n est pair ?

□

Exercice 57 (Occurrences d'un caractère, ⋆⋆)
Écrire une fonction `inString` *qui prend en entrée une chaîne de caractères* `s` *et un caractère* `c` *(c'est-à-dire* `c` *est une chaîne de caractères qu'on suppose de longueur 1), et affiche toutes les positions de* `s` *où* `c` *apparaît.*

Contrat:
(s,c) = ("abracadabra","a") → affichage : 0 3 5 7 10

□

11 Retour conditionnel

Expressions conditionnelles [Cours]

— En utilisant l'opérateur ternaire ?, on peut créer des expressions dont la valeur dépend d'une condition.
— La syntaxe est alors la suivante : condition ? expression1 : expression2 et la valeur de l'expression est expression1 si condition est évaluée à vraie et expression2 si condition est évaluée à fausse.
— Par exemple, le code suivant :

```
int n = 25;
String ch = " " ;
if ( n %2 == 0) {
ch = " Le nombre est pair " ;
}
else {
ch = " Le nombre est impair " ;
}
```

peut se réécrire avec une expression conditionnelle de la façon suivante :

```
int n = 25;
String ch = ( n %2 == 0 ) ? " Le nombre est pair " : " Le nombre est
impair " ;
```

Exercice 58 (Fonctions avec expression conditionnelle, ⋆)

1. *On considère la fonction suivante :*

```
public static int fonc ( int x ) {
    int res = (x >=0) ? x : -x ;
    return res ;
}
```

Que renvoient `fonc(-5)` *et* `fonc(3)` *? Que calcule donc la fonction* `fonc` *?*

2. *On considère la fonction suivante :*

```
public static String fonc2 ( String st ) {
    String res =(st.equals("Alice") || st.equals("Bob"))?
      "Utilisateur connu" : "Utilisateur inconnu";
    return res ;
}
```

Que renvoient `fonc2("John")` *et* `fonc2("Alice")` *?*

□

12 Accumuler des valeurs grâce aux boucles

Utilisation d'accumulateur dans les boucles [Cours]

— Si une variable est déclarée avant une boucle, sa valeur persiste d'une itération de la boucle à la suivante. On peut donc modifier sa valeur à chaque itération de boucle, et donc *accumuler* une valeur dans cette variable.
— Par exemple, le fragment de code suivant calcule la somme des carrés des entiers de 1 à 100 :

```
int s = 0;
for(int i = 1; i <= 100; i++) {
    s = s + i * i;
}
```

Exercice 59 (Accumulation, ⋆)
Modifier la boucle donnée ci-dessus pour qu'elle calcule la somme des cubes des entiers de 1 à 42. □

Exercice 60 (Comprendre et modifier une boucle, ⋆)

```java
String st = "";
for (int i = 1; i <= 50; i++) {
    st = st + "ab";
}
```

1. *Que contient la variable* `st` *après la suite d'instructions donnée ci-dessus ?*
2. *Modifier les instructions ci-dessus pour qu'à la fin de leur exécution la variable* `st` *contienne la chaîne de caractère "aaaaaa ... aaaa" avec la lettre "a" répétée 110 fois.*
3. *Modifier de nouveau la suite d'instructions pour imprimer à l'écran à chaque tour de boucle le contenu de la variable* `st`*. Qu'affichera l'exécution de votre code ?*

□

Exercice 61 (Accumulation booléenne, ⋆⋆)
On suppose donnée la fonction suivante qui lit un entier au clavier :

```java
public static int readInt() {
    Scanner sc = new Scanner (System.in);
    return sc.nextInt ();
}
```

Écrire un fragment de code qui lit 5 entiers au clavier, puis affiche « Gagné » si l'entier 42 se trouvait parmi ceux-là, et sinon « Perdu ». Attention, il faut lire les 5 entiers et seulement ensuite afficher le résultat. □

DIY

Exercice 62 (Puissance, ⋆)
Écrire une fonction "`int power (int x, int n)`*" qui renvoie la valeur* x^n*.* □

Exercice 63 (Factorielle, ⋆)
Écrire une fonction "`int fact (int n)`*" qui renvoie la factorielle de* n*.* □

Exercice 64 (Ça use, ⋆⋆)

1. *Écrire un fragment de code qui affiche les paroles de la chanson que vos neveux chantaient durant les vacances d'été :*
   ```
   1 kilomètre à pied, ça use les souliers.
   2 kilomètres à pied, ça use les souliers.
   ...
   100 kilomètres à pied, ça use les souliers.
   ```
   Attention, le premier vers est différent (le mot kilomètre *est au singulier).*
2. *Modifier votre code pour qu'il affiche les vers dans l'ordre inverse.*

□

13 Boucles imbriquées

Boucles imbriquées [Cours]

Le corps d'une boucle peut lui-même contenir une boucle. On parle alors de *boucles imbriquées* :

```java
for (int i = 0; i < 10; i++) {
    for (int j = 1; j <= 5; j++) {
        System.out.println(i);
        System.out.println(j);
    }
}
```

— À chaque tour de la boucle externe, la boucle interne fait un tour complet. La boucle externe est donc lente, la boucle interne est rapide.
— Chaque boucle a son propre compteur.
— Pour le lecteur humain, il est essentiel d'utiliser l'indentation pour indiquer l'imbrication des boucles. Comme d'habitude, le compilateur ignore l'indentation et ne considère que les accolades.
— La variable de la boucle externe peut être utilisée dans la boucle interne. Par contre, la variable de la boucle interne (j) n'est pas accessible en dehors de celle-ci.

Exercice 65 (Cent, ⋆)
Écrire un fragment de code qui affiche les nombres de 0 *à* 99*, à raison de* 10 *nombres par ligne :*

```
0 1 2 3 4 5 6 7 8 9
10 11 12 13...
...
90 91 92 93 94 95 96 97 98 99
```

□

Exercice 66 (Triangle, ⋆⋆)
Écrire une fonction qui prend en paramètre un entier n *et affiche, pour* i *allant de* 1 *à* n*,* i *étoiles sur la* i*-ième ligne. Par exemple, pour* $n = 5$*, afficher :*

```
*
* *
* * *
* * * *
* * * * *
```

□

DIY

Exercice 67 (Nombres premiers, ⋆⋆)

1. *Écrire une fonction "*`int isPrime (int n)`*" qui renvoie* 1 *si* n *est un nombre premier,* 0 *sinon. (On rappelle que* 1 *n'est* pas *un nombre premier ; le plus petit nombre premier est donc* 2*.)*
2. *Écrire une fonction "*`int sumPrime (int n)`*" qui renvoie la somme des nombres premiers compris (au sens large) entre* 1 *et* n*.*

□

Exercice 68 (Table de multiplication, ⋆⋆)

1. *Écrire un fragment de code qui affiche les tables de multiplication pour les entiers de* 1 *à* 10*.*
   ```
   1 2 3 4 ... 10
   2 4 6 8 ... 20
   ...
   10 20 30 ... 100
   ```
2. *Écrire une fonction qui prend un entier* n *en paramètre et renvoie* 1 *si* n *apparaît dans la table de multiplication, et 0 sinon.*

□

Exercice 69 (Carrés, ⋆⋆)

1. *Écrire une fonction qui prend en paramètre un entier positif* n *et affiche un carré d'étoiles plein de côté* n*. Par exemple, si* n *vaut* 4*, votre fonction affichera*
   ```
   ****
   ****
   ****
   ****
   ```
2. *Modifier votre fonction pour qu'elle affiche un carré creux, par exemple pour* n *valant* 4*,*
   ```
   ****
   *  *
   *  *
   ****
   ```

□

Exercice 70 (Séries, ⋆ – ⋆⋆⋆)
Pour chacune des séries suivantes, trouver le plus petit programme (en nombre d'appels de fonctions) qui affiche les séries de 20 *nombres suivants à l'écran :*

— *0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 (⋆)*
— *1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 (⋆)*
— *0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 (⋆⋆⋆)*
— *0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 (⋆⋆)*
— *0 2 4 6 8 9 11 13 15 17 18 20 22 24 26 27 29 31 33 35 (⋆⋆⋆)*
— *0 4 2 2 1 2 3 3 4 12 5 17 9 2 1 3 4 19 3 8 (⋆)*

□

Exercice 71 (Sous-chaîne, ⋆⋆)
Écrire une procédure `sousChaine` qui prend en paramètre deux chaînes de caractères `s` et `ss` non vides, et affiche toutes les positions de `s` à partir desquelles `ss` apparaît comme sous-chaîne.

Contrat:
(s,ss) = ("abracadabra","abra") → affichage : 0 7

□

14 Boucle `while`

Boucle non bornée [Cours]

La boucle non bornée permet de répéter des instructions tant qu'une expression booléenne est vraie. Elle est utile lorsqu'on ne connaît pas *a priori* le nombre d'itérations nécessaires.

```
while (expression booléenne) {
    instructions à répéter
}
```

— L'expression booléenne est appelée condition ou encore test d'arrêt de la boucle.
— Par exemple, la boucle suivante s'arrêtera quand la valeur de la variable entière a sera 0 après avoir affiché 50 lignes contenant la chaîne « Hello ».

```
int a = 50;
while (a > 0) {
    System.out.println("Hello");
    a = a - 1;
}
```

— Une boucle non bornée peut ne jamais terminer si sa condition est toujours vraie. La plupart du temps, la non terminaison du programme n'est pas le comportement escompté car on souhaite obtenir un résultat ! [1]
— Par exemple, la boucle suivante ne termine jamais et l'instruction « `System.out.println ("Bonjour");` » n'est donc jamais exécutée :

```
int b = 0;
while (b == 0) {
    b = b / 2;
}
System.out.println ("Bonjour\n");
```

— On peut aussi faire en sorte que la boucle soit exécutée au moins une fois et l'expression booléenne est alors évaluer après la première itération. Pour cela on utilise la syntaxe suivante :

```
do{
  instructions à répéter
}
while (expression booléenne);
```

Exercice 72 (Première boucle, ⋆)
Quelle est la valeur de la variable `r` à la fin de la suite des instructions suivantes ?

```
int n = 49;
int r = 0;
while (r * r < n) {
    r = r + 1;
}
```

□

Exercice 73 (Différence entre while et do...while, ⋆)

1. *Quelle est la valeur de la variable `n` à la fin de la suite d'instructions suivantes ?*

```
int n=10;
while (n<10){
  n=n*2;
}
```

2. *Même question avec la suite d'instructions suivantes ?*

```
do{
  n=n*2;
} while (n<10);
```

□

Exercice 74 (Les "for"s vus comme des "while"s, ⋆)

Réécrivez la suite d'instructions suivante pour obtenir le même comportement en utilisant un "`while`" à la place du "`for`".

```
int a = 0;
for (int i = 0; i < 32; i++) {
    a = a + i;
}
System.out.println (a);
```

□

Exercice 75 (Affichage maîtrisé, ⋆)

Écrire, à l'aide d'une boucle, un programme qui affiche la chaîne `"bla"` *plusieurs fois de suite autant que possible, sans dépasser les* 80 *caractères (longueur standard d'une ligne dans un terminal).*

□

Exercice 76 (Terminaison, ⋆)

Qu'affichent les deux séquences d'instructions suivantes ? Est-ce que leur exécution se termine ?

```
int n = 2;
while (n > 0) {
    System.out.println (n);
}
```

```
int n = 2;
while (n > 0) {
    n = n * 2;
    System.out.println(n);
}
```

□

Exemples d'utilisation de variables booléennes [Cours]

— Les variables booléennes peuvent être utilisées pour signaler qu'une condition a été vérifiée, ce qui peut être utile pour arrêter une boucle.
— On peut aussi utiliser les variables booléennes comme paramètres de fonction pour faire varier le comportement selon des conditions.
— Les variables booléennes sont parfois appelées drapeaux (*flag* en anglais).

Exercice 77 (Drapeau et recherche, ⋆)

1. *On considère la suite d'instructions suivante qui parcourt un tableau* `t` *et affecte* `true` *à la variable* `found` *si une des cases de* `t` *vaut* 3 *:*

```
int[] t = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int i = 0;
boolean found = false;  // flag
```

```
while (i < t.length) {
    if (t[i] == 3) {
        found = true;
    }
    i = i + 1;
}
```

Quelle est la valeur contenue dans la variable `i` *après ces instructions ?*

2. *Même question sur la suite d'instructions suivante :*

```
int[] t = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int i = 0;
boolean found = false;  // flag
while (i < t.length && !found) {
    if (t[i] == 3) {
        found = true;
    }
    i = i + 1;
}
```

3. *Qu'en déduisez-vous ?*
4. *Écrire une fonction « `boolean findValueTab(int[] t, int v)` » qui teste si la valeur contenue dans* `v` *est présente dans une case du tableau d'entiers* `t`.

□

Exercice 78 (Paramètre booléen, ⋆)
Écrire une fonction qui énumère les entiers entre 1 *et* n*. La fonction prend deux paramètres : l'entier* n*, et un booléen* `up`*. Si le paramètre booléen est vrai, on compte de 1 à n, sinon on compte de* n *à* 1*.* □

DIY

Exercice 79 (Comptine, ⋆)
Modifier les instructions suivantes pour que l'accord de kilomètre(s) soit correct.

```
int n = 10;
while (n > 0) {
    System.out.print (n);
    System.out.println (" kilometre a pied ca use les souliers");
    n = n - 1;
}
```

□

Exercice 80 (Palindrome, ⋆⋆)
Un mot est un palindrome si on obtient le même mot en le lisant à l'envers. Par exemple "kayak", "ressasser", ou "laval" sont des palindromes. Écrire une fonction qui renvoie le booléen `true` *si le mot* `s` *est un palindrome et* `false` *sinon.* □

Exercice 81 (Logarithme, ⋆)
Écrire une fonction qui prend en paramètre un entier n, et renvoie le nombre de fois qu'il faut diviser celui-ci par deux avant que le résultat soit inférieur ou égal à 1. □

Exercice 82 (Itérations, ⋆)
Combien y a-t-il d'itérations dans la boucle suivante :

```
int n = 15;
while (n >= 0) {
    n = n - 1;
}
```

Même question pour la boucle suivante :

```
static void nbIterations (int n) {
    while (n >= 0) {
        n = n - 1;
    }
}
```

□

Exercice 83 (n-ème chiffre, ⋆⋆)
Écrire une fonction qui prend en paramètres deux entiers k et n et renvoie le n-ème chiffre de k, ou 0 si k n'est pas aussi long (par exemple le 2ème chiffre de 1789 est 8). □

Exercice 84 (Binaire, ⋆⋆⋆)
Écrire une fonction qui prend en paramètre un entier `n` *et qui renvoie la représentation binaire de* `n` *sous la forme d'une chaîne de caractères formée de caractères* 0 *et* 1. □

Exercice 85 (lastOcc, ⋆⋆)
Écrire une fonction « `int lastOcc(int[] tab, int x)` » qui prend en paramètre un tableau et une valeur entière. Si le tableau contient cette valeur, la fonction renvoie l'indice de la dernière occurrence de la valeur. Si le tableau ne contient pas cette valeur, la fonction renvoie -1.

Cet exercice a déjà été traité au TD 4, mais on demande de remplacer la boucle `for` *par une boucle* `while`. □

Exercice 86 (Primalité, ⋆⋆)

1. *Écrire une fonction* `prime` *qui prend en paramètre un entier* `n` *et renvoie* `true` *si* `n` *est premier ou* `false` *sinon.*
2. *Écrire une fonction* `next` *qui prend en entrée un entier x et qui renvoie le plus petit nombre premier $p \geq x$. On pourra bien sûr se servir de la fonction* `prime` *précédente.*
3. *Ècrire une fonction* `number` *qui prend en entrée un entier y et qui renvoie le nombre de nombres premiers $p \leq y$. On pourra bien sûr se servir de la fonction* `prime`.

□

Exercice 87 (Logarithme (itéré), ⋆⋆)

1. *Le logarithme en base 2 d'un entier $n \geq 1$ (noté $\log_2 n$) est le réel x tel que $2^x = n$. On se propose de calculer la partie entière de $\log_2 n$, c'est-à-dire le plus grand entier m tel que $2^m \leq n$. On notera l la partie entière du logarithme en base 2, c'est-à-dire que $m = l(n)$. Par exemple, $l(8) = l(9) = 3$ car $2^3 = 8 \leq 9 < 2^4$.*
   Écrire une fonction `l` *qui prend en paramètre un entier n et renvoie la partie entière $l(n)$ de son logarithme en base 2. On calculera $l(n)$ en effectuant des divisions successives par 2.*
2. *À partir de la fonction l précédente, on peut définir une fonction l^* ainsi : $l^*(n)$ est le plus petit entier i tel que la i-ème itération de l sur l'entrée n vaille 0. Par exemple, $l^*(1) = 1$ car dès la première itération, $l(1) = 0$; ou encore $l^*(2) = 2$ car $l(2) = 1$ et $l(1) = 0$. Pour prendre un exemple plus grand, on a $l^*(1500) = 4$ car $l(1500) = 10$, $l(10) = 3$, $l(3) = 1$ et $l(1) = 0$: la quatrième itération de l sur 1500 vaut 0 (en d'autres termes, $l \circ l \circ l \circ l(1500) = 0$).*
   Écrire une fonction `lstar` *qui prend en paramètre un entier n et renvoie $l^*(n)$.*

□

Exercice 88 (Racine cubique, ⋆⋆)
Écrire une fonction qui renvoie la racine cubique d'un entier x, c'est-à-dire le plus petit entier a tel que $a^3 \geq x$. On peut s'inspirer de l'exercice 72. □

Exercice 89 (Racine $n$ième, ⋆⋆⋆)
Maintenant, écrire une fonction qui renvoie la racine $n$ième d'un entier x, c'est-à-dire le plus petit entier a tel que $a^n \geq x$. □

Exercice 90 (Syracuse, ⋆⋆⋆)
La suite de Syracuse de premier terme p (entier strictement positif) est définie par $a_0 = p$ et

$$a_{n+1} = \begin{cases} \frac{a_n}{2} & \text{si } a_n \text{ est pair} \\ 3 \cdot a_n + 1 & \text{si } a_n \text{ est impair} \end{cases} \quad \text{pour } n \geq 0$$

Une conjecture (jamais prouvée à ce jour !) est que toute suite de Syracuse contient un terme $a_m = 1$. Trouver le premier m pour lequel cela arrive, étant donné p. □

Exercice 91 (Ecrire du Proust à l'écran, ⋆⋆⋆)
Écrire une fonction qui prend en paramètre une chaîne de caractères contenant une phrase de Proust et l'affiche à l'écran. La phrase peut être longue, et sur une ligne on ne peut afficher que 80 caractères, il faut donc revenir à la ligne quand le prochain mot à afficher ne peut tenir sur la ligne courante. Les mots sont délimités par les caractères espace.

□

15 Instruction `switch`

L'instruction `switch` pour distinguer des valeurs [Cours]

— L'instruction switch prend en paramètre une expression ayant pour type int ou String et réalise une instruction selon la valeur exacte de cette expression.
— C'est donc une façon facile de faire une énumération de cas.
— Par exemple :

```
int n = 3;
String jour = " " ;
switch ( n ) {
    case 1 : jour = "Lundi";
        break ;
    case 2 : jour = "Mardi";
        break ;
    case 3 : jour = "Mercredi";
        break ;
    case 4 : jour = "Jeudi";
        break ;
    case 5 : jour = "Vendredi";
        break ;
    case 6 : jour = "Samedi";
        break ;
    case 7 : jour = "Dimanche";
        break ;
    default : jour = "Invalide";
}
```

Dans la partie de code précédent, la variable jour prendra après l'exécution la valeur "Mercredi" car l'entier stocké dans n vaut 3.
— Pour utiliser l'instruction switch, on liste donc avec les case les différentes valeurs pour lesquelles on veut tester l'égalité et le mot clef default est utilisé pour traiter les cas non présents. Ainsi dans l'exemple précédent, si n avait stocké par exemple l'entier 10, alors la variable jour aurait valu "Invalide".
— Lorsque l'instruction trouve un cas correspondant, elle exécute toutes les instructions suivantes jusqu'à rencontrer l'instruction break. Il est donc important de ne pas oublier cette instruction.
— Par exemple, si on prend les instructions suivantes :

```
String prenom= "Paul";
String nom= "";
switch(prenom){
   case "John" : nom= "Lennon";
   case "Paul" : nom= "McCartney";
   case "Ringo" : nom= "Starr";
   case "George" : nom= "Harrison";
      break;
   default : nom= "Inconnu";
}
```

à la fin la variable nom contiendra la chaîne de caractères "Harrison". Si on veut qu'elle contienne la valeur "McCartney", alors il faut procéder de la façon suivante :

```
String prenom= "Paul";
String nom= "";
switch(prenom){
   case "John" : nom= "Lennon";
       break;
   case "Paul" : nom= "McCartney";
       break;
   case "Ringo" : nom= "Starr";
       break;
   case "George" : nom= "Harrison";
       break;
   default : nom= "Inconnu";
}
```

Exercice 92 (Simplification de conditions, ⋆)
Réécrivez la suite d'instructions suivantes en utilisant l'instruction `switch`.

```
int nombre =1024;
String dernierChiffre = " " ;
```

```
if ( nombre %10 == 0) {
  dernierChiffre = " zero " ;
} else if (1024%10 == 1) {
  dernierChiffre = " un " ;
} else if (1024%10 == 2) {
  dernierChiffre = " deux " ;
} else if (1024%10 == 3) {
  dernierChiffre = " trois " ;
} else if (1024%10 == 4) {
  dernierChiffre = " quatre " ;
} else if (1024%10 == 5) {
  dernierChiffre = " cinq " ;
} else if (1024%10 == 6) {
  dernierChiffre = " six " ;
} else if (1024%10 == 7) {
  dernierChiffre = " sept " ;
} else if (1024%10 == 8) {
  dernierChiffre = " huit " ;
} else {
  dernierChiffre = " neuf " ;
}
```

□

Exercice 93 (Utilisation de switch, $\star$)
Ecrire une fonction `numeroMois` *qui prend en argument une chaîne de caractères et qui renvoie un entier correspondant au numéro de mois (compris entre 1 et 12) si cette chaîne est un mois compris dans l'ensemble* { `Janvier`, `Fevrier`, `Mars`, `Avril`, `Mai`, `Juin`, `Juillet`, `Aout`, `Septembre`, `Octobre`, `Novembre`, `Decembre`} *et renvoie* 0 *sinon.* □