Introduction à la programmation

Java sous Processing

Jean Diraison – ISN

Table des matières

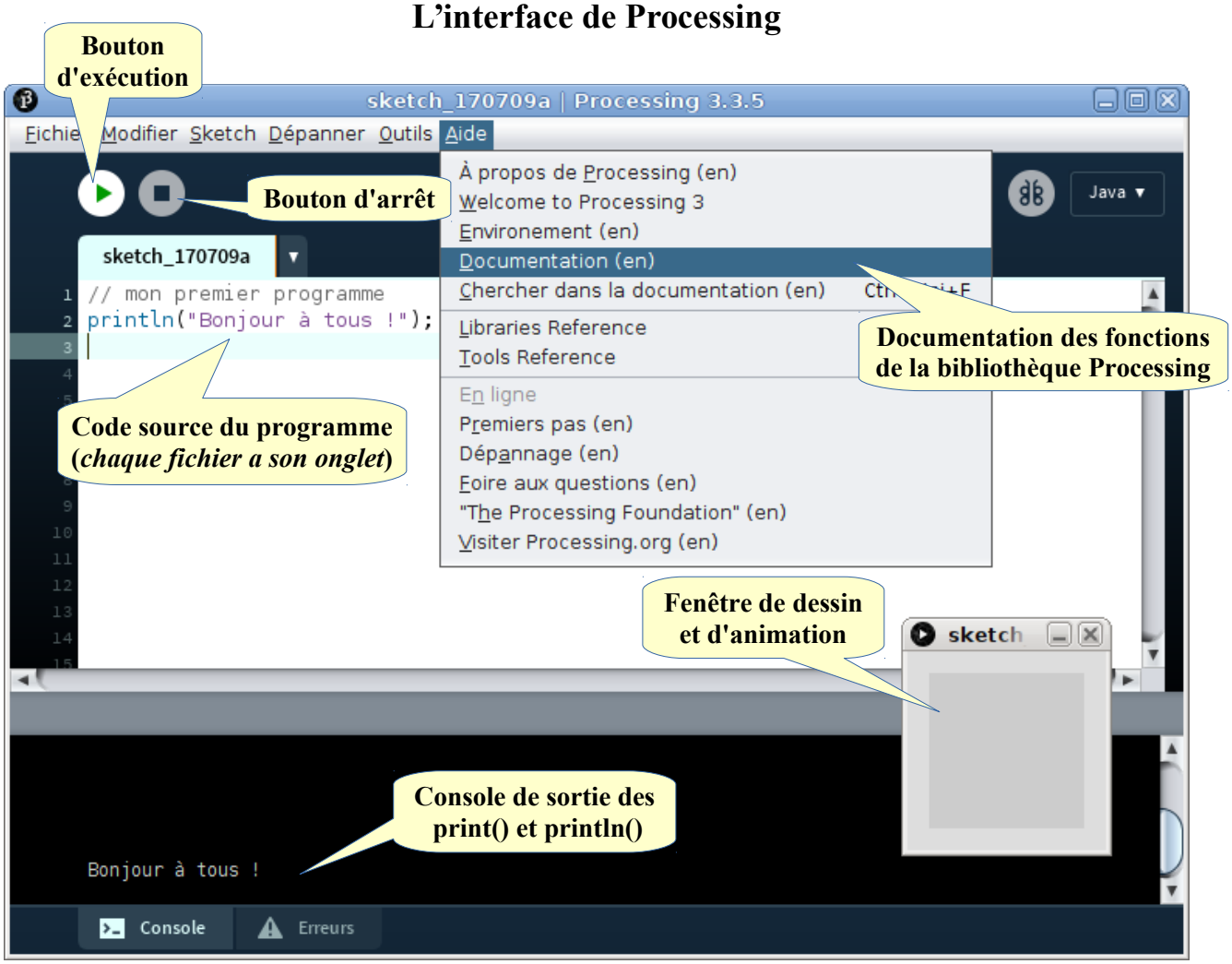
1. Découverte de Java sous Processing

Java est un langage de programmation :

- impératif : on explicite la manière de résoudre un problème
- procédural : on rassemble au sein de fonctions des suites d'instructions
- orienté objet : on regroupe dans des classes, les données et les fonctions qui s'y rapportent

Processing est un environnement de programmation Java (mais pas que), adapté à l'apprentissage et qui offre des fonctionnalités graphiques avancées. Il est librement téléchargeable à l'adresse https://processing.org/.

L'interface de Processing



Activité, un premier programme sous Processing

- Démarrez Processing.
- Tapez ce code source en respectant bien la syntaxe avec le point-virgule de fin d'instruction.

```
// mon premier programme
println("Bonjour à tous !");
```

- Cliquez enfin sur le bouton d'exécution.

Explications du programme

La première ligne est un commentaire au programme.

```
// mon premier programme
```

Les commentaires sont utilisés pour aider à la compréhension du programme. Ils peuvent aussi servir à désactiver une partie du programme lors de sa rédaction.

Il y a deux façons d'écrire des commentaires :

- En commencent par une double barre oblique `//` (*un double slash*). Le commentaire prend automatiquement fin en bout de ligne.
- En encadrant le commentaire entre `/*` et `*/`. Ceci facilite l'écriture de commentaires tenant sur plusieurs lignes.

```
/* Ce programme est écrit en Java
   Il a été codé en juillet 2017
   Il peut être copié librement
*/
println("Bonjour!");  // dire bonjour
```

La seconde ligne est une sortie d'affichage.

```
println("Bonjour à tous !");
```

Il s'agit d'une instruction, elle se termine par un point-virgule « `;` ». Cette instruction est un appel à la fonction `println()` avec un unique argument : la chaîne de caractères `"Bonjour à tous !"`.

Deux fonctions permettent d'afficher un message dans la console : `print()` et `println()`. Cette dernière rajoute automatiquement à la fin de l'affichage, un saut de ligne très utile pour éviter que tout ne soit écrit à la suite, sur une seule et même ligne sans coupure. Les sorties réalisées avec ces deux fonctions seront très utiles pour afficher des informations et déboguer les programmes…

On peut passer plusieurs paramètres en argument aux fonctions `print()` et `println()`, il suffit de les séparer par une virgule.

```
/* Un seul argument :
   - Une chaîne de caractères (type : String)   */
println("une chaine de caractères");

/* Quatre arguments :
   - L'entier 125       (type : int   )
   - La chaîne "cm ="   (type : String)
   - Le réel 1.25       (type : float )
   - La chaîne "m"      (type : String)        */
println(125, "cm =", 1.25, "m");
```

2. Types de données primitifs

Java est un langage typé, cela veut dire que les données, stockées dans des variables, possèdent un type bien défini afin d'être exploitées correctement. Nous en avons rencontré quelques-uns dans l'introduction à Processing. Nous allons commencer par les types de base (ou primitifs).

Les entiers : byte, short, int et long.

- **byte** est un type permettant de représenter des entiers de -2^7 à 2^7-1, soit -128 à 127 [*1 octet*]

```
// définition d'une variable de type byte
byte octet = 123;
println(octet);
```

```
// définition d'une variable contenant le code ascii de A
byte code_ascii_a_majuscule = 'A';
println(code_ascii_a_majuscule);     // affiche 65
```

- **short** est un type permettant de représenter des entiers de -2^{15} à $2^{15}-1$ [*2 octets*]

```
// valeur décimale du symbole Unicode € (euro)
short euro = '€';
println(euro);       // affiche 8364   (hexa: 20AC)
```

- **int** « *integer* » est le type associé aux nombres de -2^{31} à $2^{31}-1$ (≈ 2 milliards) [*4 octets*]

```
// déclaration d'une variable age de type int
int age;
// initialisation de la variable age à 23
age = 23;
println(age);
```

```
// définition d'une variable x de type int
int x = 0x7ffffffe;       // 2147483646 en hexadécimal
println(x);
// incrémentation de x (on ajoute 1 à x)
x = x + 1;
println(x);
// incrémentation de x, observez bien la valeur affichée
x = x + 1;
println(x);
```

- **long** est un type permettant de représenter des entiers de -2^{63} à $2^{63}-1$ ($\approx 9\times10^{18}$) [*8 octets*]

```
// définition d'une variable de type long
long grand_nombre = 1000000000000000000L;  // notez le L
println(grand_nombre);
```

Le plus souvent on préférera le type `int` au type `long`, car il requiert moins d'espace mémoire et il conduit à des calculs plus rapides. Les types `byte` et `short` sont, quant à eux, souvent trop limités.

Les flottants (nombre à virgule flottante) : float, double.

- **`float`** est le type associé aux réels en simple précision

```
// définition de deux variables de type float
float pi = 3.1415926;
float million = 1e6;
println(pi, million);
```

```
// limites de représentation
println("plus grand nombre positif", Float.MAX_VALUE);
println("plus petit nombre positif", Float.MIN_NORMAL);
```

- **`double`** permet de représenter des nombres plus « grands » en double précision

```
// définition d'une variable de type double et limites
double pi = Math.PI;
println(pi);
println("plus grand nombre positif", Double.MAX_VALUE);
println("plus petit nombre positif", Double.MIN_NORMAL);
```

Les caractères : char.

- **`char`** permet de représenter des caractères

```
// définition d'une variable de type char
char lettre = 'z';      // notez les guillemets simples
println(lettre);
```

Les booléens : boolean.

- **`boolean`** est le type associé aux résultats logiques `true` et `false` (vrai/faux)

```
boolean comparaison = 5 > 8;
// 5 est inférieur à 8 donc comparaison contient false
println(comparaison);
```

```
// négation logique avec l'opérateur !
boolean le_jeu_est_fini = !false;
println(le_jeu_est_fini);
```

Compléments

Certaines conversions de types sont automatiques mais peuvent générer une perte sur le nombre de chiffres significatifs, comme : `int` → `float`.

Enfin, les types primitifs possèdent une classe dite enveloppe :

`int : Integer`	`long : Long`	`short : Short`
`float : Float`	`double : Double`	`boolean : Boolean`

Activités

Un chic type

Vous allez devoir choisir les types de vos variables, faites-le soigneusement en gardant à l'esprit leurs limites intrinsèques.

Complétez les définitions et les déclarations suivantes en choisissant le type qui convient.

	Type	Variable	Valeur (optionnelle)
a)		`nombre_de_dents`	`= 32;`
b)		`lumiere_rouge`	`= 700e-9;`
c)		`interrupteur_ouvert;`	
d)		`nombre_de_pixels;`	
e)		`initiale_du_prenom;`	
f)		`nombre_d_adresses_IPv4_possibles;`	
g)		`nombre_d_atomes_dans_l_univers;`	
h)		`abscisse_boulet;`	
i)		`est_mere_de_famille;`	
j)		`moyenne_de_math;`	

Trop c'est trop-trop-trop

```
// calcul du cube
int x = 129 ;
int y = x * x * x;
println("Le cube de", x, "est", y);
```

Ce programme calcule le cube de 129.

À partir de quel entier `x` supérieur à 129, ce programme calcule-t-il faux ? 

À ski

Donner les codes ASCII des lettres et caractères suivants :

`0 :` 	`A :` 	`a :` 	`é :` 
`9 :` 	`Z :` 	`z :` 	`ç :` 
`(espace) :` 	`\n :`  (*new line / linefeed*)	`\r :`  (*carriage return*)	`\t :`  (*tabular*)

3. Opérations et calculs

Les quatre opérations arithmétiques

Les règles de priorités habituelles sont suivies par les 4 opérations.

```
// calcul de 2.3*(6.9-5.4)+8.7
float x0 = 5.4, x = 6.9;            // on définit x0 et x
float y = 2.3 * ( x - x0 ) + 8.7;   // on définit y
println(y);                         // on affiche la valeur de y
```

```
// division entière de 31 par 4
int n = 31;
int quotient = n / 4;
int reste    = n % 4;      // notez le symbole modulo %
println(n, "÷", 4, "=", quotient, "reste", reste);
```

Les fonctions mathématiques

Certaines fonctions mathématiques sont définies dans la bibliothèque de Processing et peuvent être utilisées directement et simplement.

```
// fonctions mathématiques
float x = pow(1.07, 20);         // 1.07 puissance 20
float phi = (1 + sqrt(5)) / 2;   // racine carrée...
float n = round(15.831);         // arrondi à l'unité
println("x =", x, "phi =", phi, "n =", n);
```

La colonne « *Calculation* » dans l'aide de Processing montre les fonctions disponibles.

Incrémentation et décrémentation

Lorsqu'on ajoute (ou soustrait) 1 à un entier, on dit qu'on l'incrémente (ou le décrémente). Ce sont des opérations courantes, très utilisées dans les boucles.

Il existe deux types d'incrémentation (et de décrémentation) :

- la pré-incrémentation ++n : on incrémente n avant d'utiliser la variable n s'il y a lieu
- la post-incrémentation n++ : on utilise la variable n, puis on l'incrémente juste avant de passer à l'instruction suivante

```
// incrémentation et décrémentation
int n = 5;
int a = n++;    // post-incrémentation: a = n puis n = n+1
println("a =", a, "et n =", n);
n--;            // post-décrémentation simple
println("n =", n);
a = ++n;        // pré-incrémentation: n = n+1 puis a = n
println("a =", a, "et n =", n);
a += 7;         // on additionne 7 à la variable a
n -= 2;         // on soustrait 2 à la variable n
println("a =", a, "et n =", n);
```

Autres opérations arithmétiques

Les décalages à droite et à gauche

Les microprocesseurs travaillent en binaire, c'est-à-dire en base 2. De fait il existe des opérations de décalage des chiffres pour les nombres entiers, semblables au décalage de la virgule des nombres décimaux.

Il existe deux orientations de décalage :

<< Les décalages à gauche :

- décaler un entier de 1 bit vers la gauche revient à le multiplier par 2
- décaler un entier de 2 bits vers la gauche revient à le multiplier par 4 (2^2).
- décaler un entier de 3 bits vers la gauche revient à le multiplier par 8 (2^3).

>> Les décalages à droite :

- décaler un entier de 1 bit vers la droite revient à le diviser par 2
- décaler un entier de 5 bits vers la droite revient à le diviser par 32 (2^5).

```
// décalages à droite et à gauche
int n = 10;
println(n << 3);      // 3 décalages à gauche (*8)
int m = 9;
println(m >> 1);      // 1 décalage à droite (÷2)
```

ET (AND) , OU (OR) et OU Exclusif (EOR/XOR) arithmétiques

Ces opérations permettent de manipuler l'écriture binaire des nombres entiers :

& ET arithmétique : il permet de mettre à zéro les bits qui ne sont pas sélectionnés

| OU arithmétique : il permet de mettre à un les bits sélectionnés sans toucher aux autres

^ OU Exclusif arithmétique : il permet d'inverser les bits choisis sans toucher aux autres

```
// opérations binaires : ET , OU , OU Exclusif
int n = 0b1100;
int m = 0b0101;
// ET : 0b0100 = 4
int n_et_m = n & m;
println(n, "ET", m, "=", n_et_m);
// OU : 0b1101 = 13
int n_ou_m = n | m;
println(n, "OU", m, "=", n_ou_m);
// OE : 0b1001 = 9
int n_oe_m = n ^ m;
println(n, "OE", m, "=", n_oe_m);
```

Activités

Facile ou pas…

Trouver la valeur de `x` pour que le résultat affiché soit 0.

```
// opération à trou 1
int x =      ;
int y = ( 2 * x - 10 ) / 2 + 1 ;
println(y);
```

```
// opération à trou 2
int x =      ;
int y = x - 27 / 4;
println(y);
```

```
// opération à trou 3
float x =      ;
float y = x / 2 - 6.05;
println(y);
```

```
// opération à trou 4
float x =      ;
float y = (x - 0.2) - 0.1;
println(y);
```

Tombée du ciel

Un objet en chute libre, sans vitesse initiale, tombe d'une hauteur h en un temps t suivant la formule de physique bien connue $h = \frac{1}{2} \times g \times t^2$ dans laquelle : $g \approx 9{,}8$ m/s².

Complétez le programme suivant pour qu'il calcule la durée de la chute `t`, en seconde, en fonction de la hauteur `h`, donnée en mètre.

```
final float g = 9.8;          // constante flotante g
float h = 1000;               // chute de 1000 m
float t =                   ; // formule du temps de chute
println("Une chute de", h, "m dure", t, "s");
```

Hexa-ordinaire

Que fait le programme suivant et que représentent les variables `a`, `b`, `c` et `d` par rapport à `x` ?

```
int x = 1412567177;
println(x, "s'écrit", hex(x), "en hexadécimal");
int a = (x >> 24) & 0xff;
int b = (x >> 16) & 0xff;
int c = (x >>  8) & 0xff;
int d =  x        & 0xff;
println(hex(a,2), hex(b,2), hex(c,2), hex(d,2));
```

4. Chaînes de caractères

La classe String

Nous avons déjà utilisé des chaînes de caractères en arguments de la fonction `println()`. Celles-ci débutent et finissent par des guillemets `"`.

Le type `String` va nous permettre de déclarer et définir des variables chaînes de caractères.

```
// une variable de type String
String nom = "James Bond";
println("Mon nom est", nom);
```

Remarques

- En Java les chaînes de caractères ne sont pas modifiables, on dit qu'elles sont non-mutables. Si on souhaite « modifier » une chaîne, il faut en créer une nouvelle en se servant des méthodes mises à disposition par la classe `String`.
- Les variables ne stockent que les références vers ces objets de la classe `String` que sont les chaînes de caractères.

La concaténation

Concaténer des chaînes c'est tout simplement les joindre pour n'en former qu'une seule. Ainsi la concaténation des trois chaînes `"tic-"`, `"tac-"` et `"toc"` donne `"tic-tac-toc"`.

Il y a deux façons de concaténer des chaînes :

- En utilisant simplement l'opérateur d'addition +

```
// concaténation à l'aide de l'opérateur +
String debut  = "Je suis venu, ";
String milieu = "j'ai vu, ";
String fin    = "j'ai vaincu.";
println(debut + milieu + fin);
```

- En utilisant la méthode `concat()` avec la notation *`chaine1`*`.concat(`*`chaine2`*`)`

```
// concaténation immédiate avec la méthode concat()
println("Être attentif ".concat("tu dois."));
```

```
// méthode concat() avec des variables
String debut  = "Les dés";
String fin    = " sont jetés.";
String phrase = debut.concat(fin);
println(phrase);
```

```
// méthode concat() multiple
String mot1="humanum ", mot2="diabolicum.\n";
String mot3="Erare ", mot4="perseverare ", mot5="est, ";
print(mot3.concat(mot1).concat(mot5));
print(mot4.concat(mot2));
```

Les conversions

Contrairement à la méthode `concat()`, l'opérateur + convertit implicitement le second argument en un type `String`.

```
// concaténation avec conversion implicite d'un float
float moyenne = 17.2;
String matiere = "mathématiques:";
println(matiere + moyenne);
```

On obtient le même résultat à l'aide de la méthode `toString()`, ce qui ne s'applique pas aux types primitifs comme `float`, voilà pourquoi on utilise la classe `Float`.

```
// concaténation avec conversion explicite d'un Float
Float moyenne = 15.3;         // ne marche pas avec float
String matiere = "français:";
println(matiere.concat(moyenne.toString()));
```

La conversion inverse d'une chaîne en un flottant ou un entier s'effectue à l'aide des fonctions `float()` et `int()` spécifiques à Processing (voir « *Conversion* » dans l'aide).

```
// conversion en flotant et en entier
int   taille = int("184");
float poids  = float("73");
println("taille:", taille, "cm - poids:", poids, "kg");
```

Quelques méthodes

La classe `String` dispose de nombreuses méthodes, nous allons en voir quelques-unes.

`length()`

Elle donne la taille d'une chaîne, c'est-à-dire le nombre de caractères qu'elle contient.

```
// taille d'une chaîne
String ville = "Le Havre";
int n = ville.length();
println("Il y a", n, "lettres dans :", ville);
```

`charAt()`

Cette méthode donne le caractère situé à la position indiquée, en commençant à 0.

```
// Caractère
String ville = "Paris";
char lettre = ville.charAt(3);  // index 3 : 4ème lettre
println("La 4ème lettre de", ville, "est un", lettre);
```

`toUpperCase()`

Renvoie la chaîne convertie en majuscule.

```
// Caractère
String ville = "la ville lumière";
println(ville.toUpperCase());
```

Activités

Recherche de méthode, méthode de recherche

En vous aidant de la documentation officielle Java de la classe `String` disponible à l'adresse https://docs.oracle.com/javase/8/docs/api/java/lang/String.html déterminer le nom des méthodes permettant :

- de convertir une chaîne en minuscule :

  exemple : "*La Grande Évasion*" est convertie en "*la grande évasion*"

- de trouver la position de la 1ère occurrence d'une sous-chaîne :

  exemple : "*Boss*" est à la position 3 dans la chaîne "*Le Bossu*"

- d'extraire une sous-chaîne suivant les positions de début et de fin :

  exemple : la sous-chaîne de "*Peur sur la ville*" allant de 5 à 8 est "*sur*"

- de remplacer une sous-chaîne (comme un mot) par une autre :

  exemple : remplacer "*courte*" par "*belle*" dans la chaîne "*La vie est courte.*" donne "*La vie est belle.*"

À la lettre

Que fait le programme suivant ?

```
String titre = "le livre de la jungle";
String lettre = "" + titre.charAt(0);  // double guillemets
String chaine = lettre.toUpperCase() + titre.substring(1);
println(chaine);
```

Réécriture et calcul

Que fait le programme suivant ?

```
String chaine = "Pommes : 2.15 €/kg";
int n = chaine.indexOf(":");
float prix = float(chaine.substring(n+2,n+7));
println(prix);
```

En vous inspirant du programme précédent, écrire un programme qui :

- à partir de la chaîne "Poires : 2.36 €/kg"
- calcule le prix pour 5 kg
- affiche la réponse sous la forme : "5 kg de … coûtent … €"
- en complétant automatiquement les … par ce qui convient
- et qui soit aussi capable de donner la bonne réponse pour "Bananes : 1.78 €/kg"

5. Tableaux

Qu'est-ce qu'un tableau ?

Un tableau d'entiers, de réels, de chaînes… est un espace mémoire contenant ces éléments rangés les uns à la suite des autres, et rendus accessibles par leur rang ou leur indice qui débute à 0, en utilisation la notation avec crochets *nomTableau*[*indice*]

```
// définition d'un tableau de flottants
float[] notes = { 14.5, 13.0, 16.75, 10.5, 12.4 };
println(notes[0], notes[1], notes[4]);
```

```
// autre définition d'un tableau
float prix[] = { 2.64, 1.83, 5.17, 0.90 }; // notez les []
float total = prix[0] + prix[1] + prix[2] + prix[3];
println(total, "€");
```

On peut aussi définir des tableaux d'objets et manipuler le contenu du tableau.

```
// tableau de chaîne et modification d'un élément
String[] articles = { "livre", "cahier", "stylo" };
articles[2] = "classeur";      // on redéfinit l'élément 2
println(articles[0], articles[1], articles[2]);
```

Attribut : `length`

Les tableaux Java ont chacun une taille fixée, indiquée par l'attribut `length`.
On accède à cet attribut par la notation *tableau*`.length`

```
// taille d'un tableau
String[] felins = { "tigre", "guépard", "panthère" };
int n = felins.length;          // attribut length
println("C'est un tableau de taille", n );
```

Référence

On peut employer plusieurs variables pour se référer à un même tableau.

```
// deux variables pour un même tableau
String[] hobbits = { "Frodon", "Sam", "Merry", "Pippin" };
String[] prenoms = hobbits;
println(prenoms[3]);
```

Création d'un tableau

Pour créer un tableau de taille prédéfinie on utilise le mot clef `new` suivi du type et de sa taille entre deux crochets, c'est-à-dire en respectant la notation `new` *type*[*taille*]

```
// création puis initialisation d'un tableau de 5 entiers
int[] tailles = new int[5];
tailles[0] = 35;
tailles[1] = 26;
tailles[2] = tailles[3] = 17;
println(tailles[1], tailles[2], tailles[4]);
```

Remarque

Dans le cas des types primitifs, les éléments du tableau nouvellement créé sont initialisés à 0 (entier, flottant, caractère…).
Pour un tableau d'objets, les éléments sont initialisés avec la référence `null`.

Recopie d'un tableau

Si on souhaite agrandir un tableau, pour y stocker davantage d'éléments, on n'a pas d'autre choix que d'en créer un nouveau et d'y copier tous les éléments du tableau initial.

```
// création d'un tableau plus grand
String[] mechants = { "Sauron", "Nazgul" };
String[] nouveau = new String[3];
nouveau[0] = mechants[0];
nouveau[1] = mechants[1];
nouveau[2] = "Balrog";
// on change la référence du tableau
mechants = nouveau;
println(mechants[0], mechants[1], mechants[2]);
```

Cette opération de recopie, utilise généralement la boucle `for` ou `arrayCopy()` (Processing)

Tableau multi-dimensionnel

Il est possible de créer des tableaux à plusieurs dimensions. Un tableau à deux dimensions peut s'avérer très utile pour représenter la grille d'un jeu de bataille navale par exemple.

```
// création d'une grille de bataille navale
boolean[][] grille = new boolean[10][10];  // tout à false
// positionnement d'un navire de longueur 2
grille[3][7] = true;       // position (3,7)
grille[3][8] = true;       // position (3,8)
// résultats des tirs en (2,8) et (3,8)
println(grille[2][8]);     // false : dans l'eau
println(grille[3][8]);     // true  : touché
```

Un tableau d'entiers à deux dimensions comme : `int[][] tab = new int[7][5]` est un tableau contenant 7 éléments : `tab[0], tab[1], ... tab[7]` qui sont tous des tableaux de 5 entiers : `tab[0][0] ... tab[0][4]` puis `tab[1][0] ...`

```
// taille d'un tableau à deux dimensions
int[][] table = new int[7][5];
println("Taille du tableau tab  : ", table.length);
println("Taille du tableau tab[0] : ", table[0].length);
println("Taille du tableau tab[6] : ", table[6].length);
```

Exception

Toute tentative d'accès à un élément dont l'indice dépasse la longueur du tableau se solde par une erreur d'exécution, appelée exception, qui si elle n'est pas gérée provoque l'arrêt du programme.

```
// déclenchement de l'exception ArrayIndexOutOfBounds
String[] liste = { "Legolas", "Elrond", "Arwen" };
println(liste[2]);   // s'exécute normalement jusqu'ici
println(liste[3]);   // sort du tableau de taille 3
```

Activités

Le bon indice

1) On considère le tableau uni-dimensionnel suivant :

```
int[] tab = { 56, 91, 768, 83, 24, 512, 49, 21 };
```

a) Quelle est sa taille ?

b) Donner les valeurs des termes suivants :

`tab[0]` : `tab[1]` : `tab[5]` : `tab[10]` :

2) On considère le tableau bidimensionnel suivant :

```
String[][] tab = { { "livre" , "stylo", "papier", "crayon", "gomme"  },
                   { "bateau", "rame" , "ancre" , "voile" , "safran" },
                   { "maison", "tente", "hutte" , "igloo" , "cabane" },
                   { "auto"  , "frein", "volant", "feux"  , "klaxon" } };
```

a) Donner les tailles des deux tableaux :

`tab` : `tab[0]` :

b) Donner les valeurs des termes suivants :

`tab[0][1]` : `tab[1][0]` : `tab[2][3]` :

Le plein de couleurs

Compléter le programme pour rajouter au tableau les chaînes `"jaune"` et `"cyan"` et obtenir l'affichage final : `rouge vert bleu jaune cyan magenta`

```
String[] a = { "rouge", "vert", "bleu", "magenta" };
// plusieurs lignes à écrire entre les deux commentaires


// affichage final à ne pas modifier
println(a[0], a[1], a[2], a[3], a[4], a[5]);
```

Renversante

Compléter le programme suivant pour que l'affichage final donne la réponse inversée : `géant grand moyen petit mini`

```
String[] a = {"mini", "petit", "moyen", "grand", "géant"};
// plusieurs lignes à écrire entre les deux commentaires


// affichage final à ne pas modifier
println(a[0], a[1], a[2], a[3], a[4]);
```

6. Fonctions

Nous allons définir nos propres fonctions à l'image des fonctions mathématiques que nous avons déjà utilisées sous Processing.

Utilité

Une fonction informatique sert à déporter un bloc d'instructions, pour :

- Un usage spécifique : cela permet de structurer le programme, en regroupant les instructions destinées à accomplir une tâche bien précise.

  exemple : on peut définir une fonction `ecrireTitre()` pour écrire un titre paramétrable, au centre de la fenêtre.

- Un usage répété : plutôt que de réécrire plusieurs fois la même série d'instructions (avec le risque d'erreur que cela peut entraîner) il suffit d'invoquer la fonction qui rassemble cette série d'instructions.

  exemple : on peut définir une fonction `dessinerEtoile()` pour tracer un point jaune au hasard à l'écran, et appeler régulièrement cette fonction.

Syntaxe

Pour définir une fonction, on commence par établir ses caractéristiques, en précisant :

- le type du résultat qu'elle renvoie, ou le mot clef `void` quand il n'y a aucun résultat
- le nom de la fonction ; *pour lequel l'emploi d'un verbe est vivement recommandé…*
- la liste de ses paramètres précédés de leur type, s'il y a lieu, entre deux parenthèses

On écrit ensuite le corps de la fonction entre deux accolades, en utilisant le mot clef `return` suivi de la valeur de retour pour quitter la fonction.

```
type nomDeLaFonction(type param1, type param2, ...) {
      // bloc d'instructions à exécuter
      return resultat;
}
```

```
// Cette fonction carre() a un paramètre x de type
// float et elle renvoie un résultat de type float
float carre(float x) {
    return x*x;       // renvoie le résultat de x * x
}
```

Remarque

Pour pouvoir utiliser cette fonction dans Processing, il faut passer en mode actif, en définissant une fonction d'initialisation nommée `setup()` qui est appelée une seule et unique fois automatiquement.

```
float carre(float x) { return x*x; }
// fonction d'initialisation, spécifique à Processing
void setup() {
    println("Le carré de", 3.2, "est", carre(3.2));
}
```

Variable locale

Une fonction peut posséder ses propres variables, qui ne sont pas accessibles au reste du programme, on parle de variables locales. Elles ne sont utilisables que dans la fonction, on dit que leur portée est restreinte à la fonction, c'est le cas des variables en paramètre.

```
String creerEntete(String nom, String prenom) {
    String nom_complet = prenom + " " + nom;
    return "M. ou Mme " + nom_complet;
}
// fonction d'initialisation, spécifique à Processing
void setup() {
    println(creerEntete("Melmou", "Cara"));
}
```

Procédure

Une fonction peut ne rien renvoyer, on parle alors de procédure et on précise le "type" `void`

```
// afficher un message au milieu de la petite fenêtre
void afficher(String message) {        // mot clef void
    text(message, 20, 50);   // fonction Processing text()
}
void setup() {
    afficher("Vive l'ISN !");
}
```

Processing

La bibliothèque Processing possède de nombreuses fonctions graphiques. Voici un petit exemple qui intègre les deux fonctions `ecrireTitre()` et `dessinerEtoile()` présentées en introduction. À cela il faut rajouter la fonction `draw()` qui est exécutée 60 fois par seconde.

```
// écrire un titre en bleu au milieu de l'écran
void ecrireTitre(String titre) {
    fill(#8888ff);          // remplissage en bleu
    textSize(32);           // police de taille 32 pixels
    text(titre, 180, 240); // écriture du titre au centre
}
// dessiner un point jaune au hasard à l'écran
void dessinerEtoile() {
    float x = random(640);  // coordonnées aléatoires
    float y = random(480);  // du point à tracer
    stroke(#ffffcc);        // contour de couleur jaune
    point(x, y);            // tracé du point jaune
}
// fonction d'initialisation, spécifique à Processing
void setup() {
    size(640, 480);         // taille de la fenêtre
    background(0);          // fond en noir (#000000)
    ecrireTitre("La Nuit des Etoiles");
}
// fonction périodique de dessin, spécifique à Processing
void draw() {
    dessinerEtoile();
}
```

Activités

Très moyenne

Écrire une fonction `moyenne()` qui calcule la moyenne de deux réels passés en argument. Testez-la en vérifiant que `moyenne(15.3, 19.1)` renvoie bien `17.2`.

Oui, Jean-Pierre

Écrire une fonction `premierMot()` qui renvoie le premier mot de la chaîne passée en argument. Testez-la en vérifiant que `premierMot("chose truc bidule machin")` renvoie bien la chaîne `"chose"`.

Indication : on recherchera la position du premier caractère espace…

Colorée

```
color fonction() {
    int r = int(random(256));
    int v = int(random(256));
    int b = int(random(256));
    return color(r, v, b);
}
```

a) Comment s'appelle cette fonction ?

b) Combien de paramètres possède-t-elle ?

c) Quel est le type renvoyé par cette fonction ?

d) Possède-t-elle des variables locales, si oui donnez leur nom ?

e) Que fait cette fonction ?

f) Quel nom serait plus adapté à cette fonction ?

g) Reprenez l'exemple précédant sous Processing en y intégrant cette fonction et en l'exploitant dans la fonction `dessinerEtoile()` pour obtenir un plus bel effet.

French-touch

Créer une fonction `dessinerDrapeauTricolore()` qui dessine dans la fenêtre un drapeau français à une position aléatoire, avec une hauteur elle aussi aléatoire.

Vous respecterez les contraintes de couleurs et de rapport largeur/hauteur indiquées sur la page Drapeau français de *Wikipedia*. La fenêtre aura pour dimensions 800 × 600, et la hauteur sera choisie aléatoirement entre 10 et 40 à l'aide de l'instruction `random(10,40)`.

Vous emploierez pour cela la fonction `rect()` de Processing qui dessine un rectangle avec les paramètres : `x` et `y` (coordonnées du sommet supérieur gauche), `w` (largeur) et `h` (hauteur).

7. Instruction conditionnelle if-then-else

Utilité

L'instruction conditionnelle permet de modifier le déroulement d'un programme suivant le résultat d'un test. Elle fait partie de ce qu'on nomme les instructions de contrôle.

Syntaxe

Elle prend la forme suivante dans l'ordre :

- le mot clef `if` suivi d'une condition (simple ou complexe) écrite entre parenthèses
- le bloc d'instructions à exécuter lorsque la condition est remplie, entre 2 accolades
- le mot clef `else`, suivi entre accolades du bloc à exécuter dans le cas contraire.

```
if (condition) {
      // bloc d'instructions exécuté lorsque
      // la condition est vérifiée (true)
} else {
      // bloc d'instructions exécuté lorsque
      // la condition n'est pas vérifiée (false)
}
```

```
float age = 23.5;     // testez en changeant cette valeur
// première instruction conditionnelle
if (age < 18) {
    println("Vous êtes mineur.");
} else {
    println("Vous êtes majeur.");
}
// seconde instruction conditionnelle
if (age >= 12) {
    println("Vous pouvez passer.");
} else {
    println("Vous êtes trop jeune pour passer.");
}
```

On peut faire suivre le mot clef `else` d'une seconde instruction conditionnelle.

```
float taille = 196;   // testez en changeant cette valeur
if (taille >= 190) {
    println("grande taille");
} else if (taille >= 170) {
    println("taille normale");
} else {
    println("petite taille");
}
```

La partie `else` suivie du second bloc est facultative et peut être omise s'il y a lieu.

```
float age = 7;        // testez en changeant cette valeur
String salut = "Bonjour";
if (age < 10) {
    salut += " terreur";
}
println(salut);
```

Remarque

Si un bloc d'instructions ne contient qu'une seule et unique instruction alors les accolades qui l'encadrent sont facultatives. C'est cependant une pratique risquée, car l'indentation fait qu'on oublie facilement de remettre des accolades lorsqu'on ajoute une seconde instruction.

Condition

Voici des exemples courants d'écriture de conditions simples :

- le nombre `m` est-il égal à `37` ? `m == 37`
- le nombre `x` est-il différent de `46` ? `x != 46`
- le nombre `y` est-il inférieur où égal à `12.5` ? `y <= 12.5`
- la chaîne `mdp` est-elle égale à `"secret"` ? `mdp.equals("secret")`
- la chaîne `mot1` est-elle après `mot2` dans le dico ? `mot1.compareTo(mot2) > 0`

On peut aussi écrire des conditions plus complexes à l'aide des opérateurs : `&&` (et logique) et `||` (ou logique) :

- le nombre `n` est-il situé entre `1` inclus et `10` exclus ? `1 <= n && n < 10`
- la touche `t` est-elle le `'a'` ou le `'z'` ? `t == 'a' || t == 'z'`

La fonction suivante indique si la chaîne transmise en argument débute par une majuscule.

```
boolean debuteParUneMajuscule(String phrase) {
    if (phrase.length() == 0)     // si la chaîne est vide
        return false;
    char c = phrase.charAt(0);    // c = première lettre
    return 'A' <= c && c <= 'Z';  // c entre A et Z : true
}
```

Les variables de type `boolean` permettent de simplifier l'écriture des conditions. Ainsi pour tester si une variable booléenne `etat` est à `true` ou à `false` on fait :

- `(etat)` plutôt que : `(etat == true)`
- `(!etat)` plutôt que : `(etat == false)`

```
boolean monstreVivant = true;
if (!monstreVivant) { // utilise l'opérateur ! de négation
    println("Le monstre a été vaincu.");
}
```

Opérateur ternaire

Dans de nombre cas on désire affecter à une variable des valeurs différents suivant qu'une condition est remplie ou pas. Plutôt que d'utiliser l'instruction conditionnelle, on peut employer l'opérateur ternaire dont la syntaxe, plus concise, est la suivante :

condition ? *résultat_si_vrai* : *résultat_si_faux*

```
float nbPoints = 145;
String remarque = nbPoints >= 100 ? "super" : "nul";
println("C'est", remarque);
```

On peut employer des parenthèses pour augmenter la lisibilité d'une telle expression.

Activités

Zoologique

Écrire une fonction `crier()` qui prend en paramètre la chaîne `animal` et qui affiche :

- `"ouaf !"` lorsque `animal` est égal à `"chien"`
- `"miaou"` lorsque `animal` est égal à `"chat"`
- `"cui-cui"` lorsque `animal` est égal à `"oiseau"`
- `"hein ?"` enfin dans les autres cas

Au bord du précipice

Modifier la fonction `dessinerCercle()` pour que le disque devienne rouge lorsqu'on s'approche à moins de 50 pixels d'un des bords de la fenêtre.

```
void dessinerCercle() {
    fill(#ffffff);      // couleur de remplissage : blanc
    // dessine une ellipse aux coordonnées de la souris
    // (mouseX,mouseY) et de diamètres 40 pixels
    ellipse(mouseX, mouseY, 40, 40);
}
void setup() {
    size(640,480);      // dimensions de la fenêtre
}
void draw() {
    background(0);      // redessine un fond noir
    dessinerCercle();
}
```

Tarif réduit

En boutique de photocopies propose le tarif suivant :

- plein tarif : 0,08 € / photocopie, les 50 premières
- tarif réduit : 0,06 € / photocopie, les suivantes

Écrire une fonction `calculerPrix()` qui prend en paramètre le nombre `n` de photocopies à réaliser et renvoie le prix à payer en euro pour toutes ces photocopies.

Récursive

Que fait la fonction `miroir()` suivante ?

```
String miroir(String s) {
    if (s.length() <= 1)
        return s;
    return miroir(s.substring(1)) + s.charAt(0);
}
void setup() {
    println(miroir("C'est renversant !"));
}
```

8. Boucle for

Utilité

La boucle for permet de répéter l'exécution d'un bloc d'instructions, un nombre déterminé de fois en se servant d'une variable pour compteur. Cette instruction de boucle trouve toute son utilité dans le parcours des éléments d'un tableau.

Syntaxe

Elle prend la forme suivante dans l'ordre :

- le mot clef `for` est suivi de trois expressions séparées par un point virgule
- la première expression est la définition de la variable choisie comme compteur
- la seconde est la condition à respecter pour répéter l'exécution du bloc d'instructions
- la troisième expression modifie la valeur du compteur (incrémentation, ou autre)
- enfin on trouve le bloc d'instructions, devant être répété (itéré), entre deux accolades.

```
for (déclaration; condition; modification) {
      // bloc d'instructions à répéter
}
```

```
String[] pays = { "France", "Suisse", "Canada", "Gabon" };
// affichage dans l'ordre de déclaration
//  - compteur : un entier i initialisé à 0
//  - condition : i doit être inférieur à la taille du tableau
//  - modification : i est incrémenté après chaque itération
for (int i = 0; i < pays.length; i++) {
    println("Pays francophone :", pays[i]);
}
// dans l'ordre inverse
//    comme le tableau a une taille de 4, on initialise le
//    compteur au rang du dernier élément, c'est à dire 3
for (int i = pays.length-1; i >= 0; i--) {
    println(pays[i]);
}
```

Explications

1. On commence par initialiser l'entier `i` qui sert de compteur à 0.
2. On teste la condition « `i` *est inférieur à la taille du tableau* », si elle vaut `true` on passe au point 3, sinon la boucle s'arrête et le programme se poursuit après le bloc.
3. On exécute une fois le bloc d'instructions entre accolades.
4. On modifie le compteur : `i` est incrémenté (augmenté de 1).
5. On revient au point 2.

Remarques

- Une exécution du bloc d'instructions s'appelle une itération.
- Comme la condition est vérifiée avant l'exécution du bloc, il est possible que ce bloc d'instructions ne soit jamais exécuté.
- La variable déclarée comme compteur, est une variable locale au bloc d'instructions.

- Les trois paramètres de la boucle `for` sont en fait optionnels. On peut tout à fait omettre l'un d'eux, voire deux ou tous les trois. Ce dernier cas permet de créer des boucles infinies, dont on peut sortir avec l'instruction `break`.

Instruction : `break`

On utilise l'instruction `break` pour sortir immédiatement d'une boucle.

```
int[] nombres = { 5, 32, 41, 67, 89, 103, 173, 208 };
// Affiche les premiers entiers inférieurs à 100
for (int i = 0; i < nombres.length; i++) {
    if (nombres[i] >= 100) // dès qu'un entier est >= 100
        break;             // on quitte la boucle
    println(nombres[i]);
}
```

Instruction : `continue`

On utilise l'instruction `continue` pour passer immédiatement à l'itération suivante.

```
int[] nombres = { 15, -4, -18, 7, 23, -56, 30, -2 };
// Affiche les entiers positifs uniquement
for (int i = 0; i < nombres.length; i++) {
    if (nombres[i] < 0)     // passe les nombres négatifs
        continue;
    println(nombres[i]);
}
```

Boucles imbriquées

On utilise des boucles imbriquées (une boucle dans une boucle) pour parcourir un tableau multi-dimensionnel.

```
String[][] mots = {
    { "Abricot", "Amande", "Avocat",  "Avoine"         },
    { "Brocoli", "Blette", "Betterave"                 },
    { "Carotte", "Courge", "Cresson", "Céleri", "Chou" }
};
for (int i = 0; i < mots.length; i++) {
    for (int j = 0; j < mots[i].length; j++) {
        print(mots[i][j] + " ");
    }
}
```

Boucle for-each

Comme le parcours des éléments d'un tableau est une opération très courante, il existe une variante Java adaptée à une telle opération, c'est la boucle for-each, dont la syntaxe est :

```
for (type element : tableau) {
      // bloc d'instructions à répéter
}
```

```
String[] liste = { "arme", "balle", "couteau", "dague" };
for (String mot : liste)
    println(mot);
```

Activités

Étoilée

Compléter le programme suivant pour qu'il affiche `n` étoiles (`n` pouvant varier).

```
int n = 5;
for (          ;          ;          )
    print("*");
```

Graphique

Dans une fenêtre de taille 480×480, et en utilisant les fonctions graphiques de Processing :

1) Créez, sur toute la hauteur, un dégradé horizontal allant du noir vers le blanc.
2) Créez un damier de jeu d'échec, en utilisant la fonction `rect()`.
3) Créez une cible, de tir à l'arc, formée de dix couronnes concentriques blanches, de même largeur et numérotées de 10 au centre à 1 pour la couronne extérieure.

Conseils : Pour fixer la couleur du remplissage et celle d'écriture du texte vous utiliserez la fonction `fill()`. Pour le contour, vous utiliserez la fonction `stroke()`. L'écriture du texte se fera avec `text()`, sa taille sera définie à 12 avec la fonction `textSize()`. Enfin le positionnement sera cadré à l'aide de l'appel `textAlign(RIGHT,CENTER)` au préalable.

Copier-coller

Compléter le programme suivant qui copie le contenu du tableau `source` dans un tableau référencé par la variable `destination` à l'aide d'une boucle for.

```
String[] source = { "vélo", "roue", "frein", "guidon" };
String[] destination =          ;
for (int i =          ;          ;          ) {
    ;
}
```

Existe-t-il une fonction Processing capable de faire le même travail ?

Sommes

Vous devez écrire un programme contenant les trois parties suivantes :

- Une fonction `somme()` qui prend en paramètre un entier naturel `n` et qui renvoie la somme des entiers naturels (positifs) inférieurs ou égaux à `n`.
- Écrire une fonction `somme3()` qui prend en paramètre un entier naturel `n` et qui renvoie la somme des cubes des entiers naturels inférieurs ou égaux à `n`.
- Écrire une boucle d'entiers `n` allant de 100 à 110 inclus et affichant côte à côte le carré de `somme(n)` et la valeur de `somme3(n)`.

Quelle conjecture pouvez-vous formuler sur : $1^3 + 2^3 + 3^3 + \cdots + n^3 =$?

9. Boucle while

Utilité

La boucle `while` permet de répéter l'exécution d'un bloc d'instructions, tant qu'une condition est respectée. Elle vérifie ainsi avant chaque exécution du bloc que le test donne `true`.

Syntaxe

Elle prend la forme suivante dans l'ordre :

- le mot clef `while` est suivi du test de contrôle, écrit entre parenthèses.
- on trouve ensuite le bloc d'instructions à répéter, écrit entre deux accolades.

```
while (condition) {
      // bloc d'instructions à répéter
}
```

```
// Ecriture d'un nombre entier positif en base 2
int nombre = 151;
// divise nombre par 2 en notant à chaque fois le reste
String s = "";            // s contient l'écriture binaire
while (nombre != 0) {
    int c = nombre % 2;   // reste de la division par 2
    s = "" + c + s;
    nombre /= 2;          // divise nombre par 2
}
println(s);
```

Explications

1. On initialise la chaîne `s` destinée à recueillir la représentation de `nombre` en binaire.
2. On teste si `nombre` n'est pas nul, si `true` on passe à l'étape 3 sinon on quitte.
3. On définit la variable entière `c` avec le reste (0 ou 1) de la division de nombre par 2.
4. On rajoute le chiffre `c` au début de la chaîne `s`, et non à la fin car l'ordre est inversé.
5. On divise `nombre` par 2.
6. On revient enfin à l'étape 2.

Remarque

Bien que les boucles `while` et `for` soient interchangeables on préfère utiliser une boucle `while` lorsque le nombre d'itérations n'est pas connu ou lorsqu'il n'y a pas besoin d'utiliser un compteur.

```
size(800,600);
noStroke();
color blanc = color(#ffffff);
color bleu  = color(#4040ff);
float d = 400;   // parties replaçables par la boucle
while (d > 1) {  // for (float d = 400; d > 1; d *= 0.99)
    fill(lerpColor(blanc, bleu, d/400));
    ellipse(400, 300, d, d);
    d *= 0.99;   // mais d n'est pas un compteur...
}
```

Instructions : `break` et `continue`

Les instructions `break` et `continue` jouent le même rôle que dans les boucles `for` :

- `break` permet de sortir immédiatement de la boucle
- `continue` permet de revenir au test débutant la boucle, sans exécuter le reste du bloc

```
// fait patienter l'utilisateur le temps voulu...
int nbSecondes = 10; // nombre de secondes à attendre
print("veuillez patienter");
int t0 = millis();   // temps écoulé depuis le début en ms
while (true) {       // boucle infinie semblable à for(;;)
    int t = millis();
    if (t-t0 < 1000) // recommence si moins de 1s écoulée
        continue;
    print(".");    // affiche un point toutes les secondes
    t0 = t;
    if (--nbSecondes <= 0)   // décompte les secondes
        break;
}
println("\nmerci");  // s'affiche une fois le temps écoulé
```

Boucle `do-while`

Dans un certain nombre de cas, on désire que le bloc d'instructions soit exécuté au moins une fois. On peut réaliser ceci à l'aide d'une boucle `do-while`, qui suit la syntaxe :

```
do {
      // bloc d'instructions à répéter
} while (condition);
```

```
// Tire un nombre entier au hasard entre 0 et 100, et
// qui ne se termine pas par 0
int n;
do {
    n = int(random(101)); // tire un entier au hasard
} while (n % 10 == 0);   // recommence tant qu'il finit par 0
println(n);
```

Algorithme de dichotomie

Une méthode de recherche efficace du rang d'un élément présent dans un tableau trié (ou encore du zéro d'une fonction monotone sur un intervalle), consiste à observer l'élément situé au milieu du tableau, pour établir dans quelle moitié se trouve l'élément que l'on recherche. Il suffit de répéter la même démarche pour trouver, en quelques étapes seulement, le rang de l'élément recherché, puisqu'à chaque itération on divise l'ensemble de recherche par deux. Ainsi avec dix étapes on divise la taille du tableau de recherche par $2^{10} = 1024$.

```
rechercher ( chaine , debut , fin ) :  # intervalle [debut;fin[
      tant que debut + 1 < fin         # reste plus d'une chaîne
      faire
            milieu = (debut + fin) / 2   # rang du milieu
            si chaine < tableau[milieu]  # compare des chaînes
            alors
                  fin = milieu           # prendre [debut;milieu[
            sinon
                  debut = milieu         # prendre [milieu;fin[
      renvoyer debut
```

Activités

Hexadécimale

Reprenez le premier exemple de la leçon et adaptez-le pour obtenir l'écriture d'un entier positif en base 16.

Indication : On pourra utiliser un tableau { "0", …, "9", "a", ...,"f" } pour faire la correspondance des représentations de chiffres de types `int` et `String`.

Collatz

Compléter le programme suivant pour tester la validité de la conjecture de Syracuse.

```
// Conjecture de Syracuse (à voir sur Wikipedia)
long m = 31;           // essayez avec différents nombres
print(m);
long n =        ;
while (         ) {
    n = (n%2==0 ? n/2 : 3*n+1);  // teste si n est pair...
    print(",", n);
}
println("\nLa conjecture est vérifiée pour", m);
```

Trop for

Remplacer la boucle `for` par une boucle `while` dans le programme suivant.

```
String[] mammipheres = { "baleine", "dauphin", "lamentin",
"loutre", "morse", "narval", "orc", "otarie", "phoques" };
for (int i = 0; i < mammipheres.length; i++)
    println(mammipheres[i]);
```

Dichotomie

Implémenter l'algorithme de recherche dichotomique en complétant la fonction rechercher. Elle prend pour arguments la référence du tableau et la chaîne à rechercher, et elle renvoie le rang de la chaîne dans le tableau. La comparaison des chaînes se fera avec `compareTo()`.

```
int rechercher(String[] tableau ,String chaine) {
    int debut = 0;
    int fin = tableau.length;

}
String[] syllables = { "ba", "be", "bi", "bo", "ca", "co",
     "ki", "ma", "me", "mo", "mu", "pa", "pe", "pi", "po",
     "ri", "ta", "te", "ti", "to", "tu", "za", "zo" };
void setup() {     // mode actif de Processing obligatoire
    String clef = "po";
    int index = rechercher(syllables, clef);
    println(clef, syllables[index], ": index", index);
}
```

10. Instruction switch-case

Utilité

On est parfois amené à vérifier à la suite, plusieurs valeurs pour une même variable. Plutôt que de répéter les instructions conditionnelles `if-then` on peut faire appel à l'instruction `switch-case` dès lors que cette variable est de type primitif. Ceci facilite grandement la lecture et la compréhension du code.

Syntaxe

Elle prend la forme suivante :

- on utilise le mot clef `switch` suivi de la variable écrite entre parenthèses
- on ouvre le bloc d'instructions par une accolade
- on écrit le mot clef `case` suivi de la valeur à tester et du caractère deux-points
- on écrit à la suite les instructions à exécuter si la variable prend cette valeur
- on termine la suite d'instructions par le mot clef `break` pour sortir du bloc
- on recommence ainsi un case pour chacune des valeurs restantes
- on peut rajouter le mot clef `default` suivi du caractère deux-points et d'une suite d'instructions pour traiter le cas de toutes les autres valeurs de la variable
- on termine par refermer le bloc avec une accolade

```
switch (variable) {
case valeur1:
     // suites d'instructions à exécuter si variable = valeur1
     break;
case valeur2:
     // suites d'instructions à exécuter si variable = valeur2
     break;
...
default:
     // suites d'instructions à exécuter sinon (facultatif)
}
```

```
// Déplace une ligne en appuyant sur les touches flêchées
int y = 300;                // ordonnée de la ligne
void setup() {           // appelée une fois au tout début
    size(800,600);
    stroke(#ffffff);
}
void draw() {            // appelée 60 fois par seconde
    background(0);          // efface tout (fond noir)
    line(0, y, 800, y);   // trace la ligne à l'ordonnée y
}
void keyPressed() {     // appelée à chaque touche pressée
    switch (keyCode) {     // code de la touche pressée
    case UP:   y -= 8;     // UP = 38   (touche flêchée ↑)
               break;
    case DOWN: y += 8;     // DOWN = 40 (touche flêchée ↓)
               break;
    default:   y = 300;    // autre touche : réinitialise y
    }
}
```

Remarques

Si le mot clef `break` est absent alors les instructions du `case` suivant sont exécutées.
Il n'est pas nécessaire après la dernière instruction du `switch`.

```
String mot = "blancheur";
for (int i = 0; i < mot.length(); i++) {
    char lettre = mot.charAt(i);
    switch (lettre) {
    case 'a': case 'e':      // un case par voyelle
    case 'i': case 'o':      // sans utiliser break
    case 'u':
        println(lettre, "est une voyelle");
        break;
    default:                 // toutes les autres lettres
        println(lettre, "est une consonne");
    }
}
```

La partie `default` est facultative. Elle peut aussi être déplacée ailleurs qu'à la fin du `switch`, mais dans ce cas elle doit se terminer par un `break` afin d'éviter l'exécution des instructions du `case` suivant.

```
// Fait varier la teinte suivant les touches fléchées
int teinte = 0;          // teinte variant entre 0 et 100
void setup() {
    size(400,300);
    colorMode(HSB, 100); // mode teinte-saturation-lumière
}
void draw() {
    background(color(teinte, 100, 100));
    fill(0);
    textSize(32);
    text("Teinte: "+teinte, 130, 150);
}
void keyPressed() {    // réagit à la pression des touches
    switch (keyCode) {
    case UP:   if (teinte < 100) // valeur maximale de 100
                   teinte++;
               break;
    case DOWN: if (teinte > 0)   // valeur minimale de 0
                   teinte--;
    }                            // break final facultatif
}
```

L'instruction `switch-case`, n'est applicable qu'aux types primitifs. L'utilisation des objets conduit à une comparaison de leurs références, ce qui n'est généralement pas le but recherché.

Les constantes `UP`, `DOWN`, `LEFT`, `RIGHT` se réfèrent aux touches fléchées du clavier.
La variable `keyCode` peut prendre d'autres valeurs aisément identifiables sur le clavier par un `println(keyCode)`.

Voir la documentation Processing de `key` et de `keyCode` pour plus de détails.

Activités

Disque coloré

En vous inspirant du dernier exemple, créez un programme affichant un disque dont la taille sera réglable par les touches fléchées haut/bas et la couleur par les touches droite/gauche.

ADN de dragon

Complétez les déplacements avec leur tracé, pour voir apparaître la Courbe du Dragon.

```
String adn = "H";  // déplacement H:haut B:bas D:droite G…
// Initialisation
size(800,600);
background(0);
stroke(#aaccff);
// Evolution de l'ADN… renverse et tourne d'1/4 de tour
for (int i = 0; i < 13; i++) {
    String s = "";
    for (int k = adn.length()-1; k >= 0; k--) {
        switch (adn.charAt(k)) {
            case 'D': s += 'B'; break;    // D devient B
            case 'H': s += 'D'; break;    // H devient D
            case 'G': s += 'H'; break;    // G devient H
            case 'B': s += 'G'; break;    // B devient G
        }
    }
    adn = s + adn;   // concatène l'ADN modifié au début
}
// Représentation de l'ADN...
float x = 630, y = 150;           // coordonnées initiales
for (int i = 0; i < adn.length(); i++) {
    char commande = adn.charAt(i);
    switch (commande) {
    case 'D': line(x, y, x+5, y); // avance à droite
              x += 5;             // de 5 pixels
              break;
    case 'G':                     ; // avance à gauche
                                  ;
              break;
    case   :                      ; // avance en haut
                                  ;
              break;
    case   :                      ; // avance en bas
                                  ;
    }
}
```

Au doigt et à l'œil

Reprenez le programme précédent et créez une fonction de dessin qui réagit aux touches fléchées et complète en même temps une chaîne de caractères qui permettra de reproduire automatiquement le dessin tracé.

11. Programmation Orientée Objet (POO)

Les objets

Un objet est une structure qui rassemble (encapsule) les données (attributs) et les fonctions (méthodes) servant à manipuler ces données. Des mécanismes de protection améliorent la fiabilité des programmes en forçant le programmeur à utiliser les méthodes dédiées et l'empêchent d'accomplir toute « mauvaise » manipulation directe de ces données internes. Il est ainsi plus aisé de réutiliser ces objets dans d'autres programmes.

Les classes

Pour créer des objets, on commence par créer une classe, qui est une structure de référence (une sorte de moule) et qui constitue un type de données à part entière. On génère ensuite des objets à partir de cette classe, on dit qu'on les instancie.

Dans une classe on définit les différentes variables servant à stocker les informations qui caractérisent chaque objet, ce sont les attributs. On définit aussi les méthodes (fonctions) qui servent à manipuler les données internes à l'objet. Enfin on définit au moins un constructeur, une méthode particulière qui sert à initialiser les objets à leur création. Les constructeurs qui se distinguent par leurs paramètres, n'ont pas de type de retour et portent comme nom celui de la classe.

La syntaxe pour définir une classe est la suivante :

```
class MaClasse {          // nom de la classe (initiales en majuscule)
      type attribut1;     // variables internes propre à chaque objet
      …
      MaClasse() {…}                  // constructeur sans paramètre
      MaClasse(type param1, …) {…}    // constructeurs avec paramètres
      …
      type methode1(type param1, …) {…}  // méthodes
      …
}
```

Pour modéliser une balle jaune qui rebondira sur les bords de la fenêtre de taille 600×400, nous allons créer une classe `Balle`.

Les coordonnées du centre de la balle et son rayon correspondront aux trois attributs : `x`, `y` et `r` de type `float`. Les déplacements à appliquer en `x` et `y` seront notés `dx` et `dy`.

La balle sera positionnée initialement au hasard sur l'écran, son rayon `r` sera fixé à 20, et les déplacements `dx` et `dy` seront eux aussi fixés au hasard dans le constructeur `Balle()`.

Deux méthodes seront disponibles :

- `void avancer()` : chargée d'appliquer les déplacements en `x` et `y` et gérant les rebonds sur les bords de la fenêtre suivant `x` en 0 et 600 et suivant `y` en 0 et 400 (en tenant compte du rayon de la balle), les déplacements seront remplacés par leur opposé.

- `void dessiner()` : chargée de tracer le disque jaune de rayon `r` en `(x;y)`.

```
class Balle {
    protected float x, y, r;    // coordonnées et rayon
    private   float dx, dy;     // déplacements en x et y

    // constructeur
    Balle() {
        x = random(600);      // coordonnées aléatoires
        y = random(400);
        r = 20;               // rayon de 20
        float angle = random(TWO_PI);  // angle aléatoire
        dx = 5 * cos(angle);  // déplacements en x et y
        dy = 5 * sin(angle);  // aléatoires
    }

    // avance la balle en gérant les rebonds
    void avancer() {
        if (x + dx < r || x + dx >= 600-r) { // si dépasse
            dx = -dx;      // inverse le déplacement en x
        } else {
            x += dx;       // applique le déplacement en x
        }
        if (y + dy < r || y + dy >= 400-r) { // si dépasse
            dy = -dy;      // inverse le déplacement en y
        } else {
            y += dy;       // applique le déplacement en y
        }
    }

    // dessine la balle en jaune de rayon r en (x,y)
    void dessiner() {
        fill(#ffff00);
        ellipse(x, y, 2*r, 2*r);
    }
}

Balle balle;    // déclare la variable balle de type Balle

void setup() {
    size(600,400);
    balle = new Balle();  // crée une nouvelle Balle
}

void draw() {
    background(0);        // vide la fenêtre
    balle.avancer();      // fait avancer la balle
    balle.dessiner();     // dessine la balle
}
```

Remarques

Une bonne pratique consiste à précéder le type des attributs du mot clef `private`, ou `protected` afin de masquer leur visibilité et empêcher tout accès aux attributs autrement que par les méthodes mises en place par le programmeur de la classe ou des classes filles.

Dans une classe, le mot clef `this` se réfère à l'objet lui-même ; ceci permet de distinguer un attribut d'une variable locale portant le même nom dans une méthode, comme `this.x` et `x`.

Activités

Constructeur

Compléter l'exemple précédent, en rajoutant à la classe `Balle` un constructeur `Balle()` permettant de paramétrer le rayon `r` de la balle. Puis créez une balle de rayon 30.

Méthode

Compléter l'exemple précédent, en ajoutant une méthode pour accélérer ou ralentir la vitesse de la balle. Vous créerez dans le corps du programme la fonction `keyPressed()` afin d'y gérer l'accélération et le ralentissement de la balle au moyen des touches du clavier.

Réutilisation

Modifiez le programme en exemple pour afficher simultanément 5 balles en mouvement.

Héritage et polymorphisme

Reprenez l'exemple de la leçon. Dans le menu Sketch, rajoutez un fichier image au format PNG (avec transparence) représentant une balle de tennis avec le nom `balle-tennis.png`.

Complétez l'exemple en insérant la classe `BalleTennis` ci-dessous. Elle hérite de la classe `Balle`. Son constructeur appelle celui de sa classe mère et charge un fichier image.

La méthode `dessiner()` est aussi redéfinie pour afficher l'image de la balle de tennis à la bonne échelle et centrée sur les coordonnées `(x,y)`. Les attributs `x` et `y` sont accessibles du fait de l'utilisation de mot clef `protected` au lieu de `private`…

```
class BalleTennis extends Balle {   // hérite de Balle
    private PImage img;       // attribut
    BalleTennis() {
        super();       // appelle le constructeur de Balle
        img = loadImage("balle-tennis.png");
    }
    void dessiner() {
        imageMode(CENTER);     // positionnement au centre
        image(img, x, y, 2*r, 2*r);  // dessin à l'échelle
    }
}
```

Modifiez enfin la fonction d'initialisation de Processing `setup()` pour créer un objet de la classe `BalleTennis` en conservant le type `Balle` pour la variable `balle`.

Bien que `balle` soit de type `Balle`, elle se comporte comme un objet de type `BalleTennis`. Ce comportement s'appelle le polymorphisme.

```
void setup() {
    size(600,400);
    balle = new BalleTennis();  // crée une nouvelle Balle
}                               // mais de tennis...
```

12. Dessin et animation sur Processing

Les couleurs

- Un niveau de gris allant du noir (0) jusqu'au blanc (255) s'obtient avec la fonction `color()`, en utilisant un seul paramètre, le niveau qui doit être un entier de 0 à 255. La valeur de retour un du type `color` (en fait un entier).

```
color gris = color(192);   // gris de niveau 192 sur 255
```

- Une couleur à 3 composantes rouge, vert, bleu, s'obtient encore avec la fonction `color()`, en passant chaque composante (de 0 à 255) en argument et dans cet ordre.

```
color jaune = color(255,255,0); // R = 255, V = 255, B = 0
```

- Une couleur peut être directement définie à partir de sa représentation HTML hexadécimale.

```
color rose = #ffc0cb;     // R = 0xff, V = 0xc0, B = 0xcb
```

- La couleur `#123456` correspond à l'entier `0xff123456` où l'octet de poids fort `0xff` est la valeur du canal alpha mesurant la transparence (ou l'opacité).

Les formes de base

Processing met en œuvre le repère suivant : l'origine `0` est le sommet supérieur gauche de la fenêtre, l'axe des abscisses `X` est horizontal et orienté vers la droite, tandis que l'axe des ordonnées `Y` est vertical et orienté vers le bas, enfin l'unité de graduation est le pixel.

- Un point se trace avec la fonction `point()` avec en argument les coordonnées `(x,y)`.

```
// point au coordonnées (200,100)
point(200,100);            // paramètres de types float
```

- Un segment se trace avec la fonction `line()` avec en argument les coordonnées successives des deux extrémités.

```
// segment d'extrémités (200,100) et (400,300)
line(200,100,400,300);     // paramètres de types float
```

- Un rectangle plein se trace avec la fonction `rect()` avec en argument le coin supérieur gauche, sa largeur et sa hauteur.

```
// rectangle : coin (200,100), largeur 80 et hauteur 60
rect(200,100,80,60);       // paramètres de types float
```

- Un disque plein se trace avec la fonction `ellipse()` avec en argument son centre, puis son diamètre horizontal et son diamètre vertical de même valeur.

```
// cercle de centre (200,100) et de rayon 40
ellipse(200,100,2*40,2*40);   // paramètres de types float
```

- Un triangle et un quadrilatère se tracent avec les fonctions `triangle()` et `quad()` avec en argument, les coordonnées des sommets.

```
// triangle de sommets (200,100) (300,100) et (250,20)
triangle(200,100,300,100,250,20);     // paramètres float
// quadrilatere de sommets (10,20) (30,90) (70,80) (60,40)
quad(10,20,30,90,70,80,60,40);        // paramètres float
```

Remplissage et contour

- La couleur des traits des contours se définit avec la fonction `stroke()` qui utilise un argument de type `color`.

```
stroke(#ffaa66);   // contour orange
```

- La couleur de remplissage d'une figure (rectangle, ellipse, triangle, quadrilatère, …) se définit avec la fonction `fill()` qui utilise un argument de type `color`.

```
fill(#44aaff);   // remplissage bleu ciel
```

- Le tracé du contour se désactive avec la fonction `noStroke()` et le remplissage par la fonction `noFill()` qui ne prennent aucun argument. Ces fonctions doivent être appelées avant un dessin ou un tracé. Les fonctions `stroke()` et `fill()` rétablissent le comportement par défaut.

```
noStroke();    // supprime le tracé du contour
noFill();      // supprime le remplissage
```

Fenêtre et animation

- La taille de la fenêtre d'affichage se règle par un appel à la fonction `size()` dans le corps de la fonction d'initialisation `setup()`, avec pour arguments sa largeur et sa hauteur.
Les dimensions de la fenêtre sont par ailleurs enregistrées dans les variables globales `width` (*largeur*) et `height` (*hauteur*).

```
void setup() {
    size(600,400);    // ouvre une fenêtre de 600 par 400
    println(width, height);  // et affiche ses dimensions
}
```

- La vitesse de rafraîchissement du dessin, c'est-à-dire le nombre d'appels par seconde à la fonction `draw()`, se règle avec la fonction `frameRate()`. Cette vitesse, qui est initialement de 60, est indiquée par la variable `frameRate`.

```
void setup() {
    frameRate(30);       // 30 appels par seconde à draw()
}
void draw() {
    println(frameRate)); // affiche la valeur du frameRate
}
```

- Les fonctions `noLoop()` et `loop()` permettent de suspendre et de relancer les appels à la fonction `draw()` et donc l'animation.

Activités

Couleurs

Réaliser un programme dessinant une représentation des couleurs primaires et secondaires à l'image du cercle chromatique.

Étoiles

Exécutez le programme suivant après avoir créé une classe `Etoile` ayant :

- Deux attributs `x` et `y` de type `float` représentant les coordonnées du centre de l'étoile
- Un attribut `couleur` de type `color`, représentant la couleur de l'étoile
- Un constructeur `Etoile()` initialisant une étoile à une position et avec une couleur aléatoires.
- Une méthode `dessiner()`, qui tracera une étoile pleine à 6 branches, formée de deux triangles équilatéraux inversés.

La fenêtre de rendu est de 800×600 et vous avez libre choix concernant la taille des étoiles. Vous veillerez à supprimer les contours des triangles pour obtenir une étoile bien pleine.

```
void setup() {
    size(800,600);
    background(#000044);      // fond bleu nuit
    frameRate(10);            // 10 draw() par seconde
}
void draw() {
    Etoile etoile = new Etoile();  // crée une étoile
    etoile.dessiner();             // et la dessine
}
```

Pluie d'étoiles

Reprenez la classe `Etoile` et modifiez-la, comme suit :

- Le constructeur doit faire apparaître l'étoile en haut de la fenêtre.
- Un attribut vitesse doit être défini aléatoirement dans le constructeur.
- Une méthode `chuter()` doit modifier l'attribut `y` suivant la valeur `vitesse`.

Vous créerez ensuite un tableau nommé `etoiles` d'éléments de type `Etoile` que vous initialiserez dans la fonction `setup()`. Vous conviendrez de la taille du tableau.

Enfin vous compléterez la fonction `draw()` pour qu'elle invoque pour chaque étoile du tableau, à la fois les méthodes `chuter()` et `dessiner()`.

13. Fichier image sur Processing

Lecture

- La lecture d'un fichier image s'effectue à l'aide de la fonction `loadImage()`. Elle prend en argument le chemin d'accès au fichier et renvoie un objet de type `PImage`. Le chemin est relatif au sous-répertoire `data` du code source du programme.

```
PImage imgballe = loadImage("chat.jpg");  // lit une image
```

- Les attributs `width` et `height` de l'objet `PImage` donnent la largeur et la hauteur de l'image.

```
println("La largeur de l'image est", imgballe.width);
println("La hauteur de l'image est", imgballe.height);
```

Affichage

- L'affichage d'une image aux coordonnées `(x,y)` s'effectue avec la fonction `image()` qui prend en argument l'image précédemment lue (objet de type `PImage`) et les coordonnées. Une variante de la fonction `image()` prend en plus la largeur et la hauteur à imposer à l'image ; ce qui peut engendrer une déformation.

```
// affiche l'image imgballe aux coordonnées (300,200)
image(imgballe, 300, 200);      // dimensions conservées
image(imgballe,300,200,60,40);  // redimensionnée en 60×40
```

- Le positionnement relatif de l'image se règle par la fonction `imageMode()` qui prend comme argument une des constantes :
  - `CORNER` : les coordonnées d'affichage sont celles du coin supérieur gauche
  - `CENTER` : les coordonnées d'affichage sont celles du centre de l'image
  - `CORNERS` : réinterprète la largeur et la hauteur comme les coordonnées du sommet opposé du rectangle image

```
imageMode(CENTER);   // positionnement centré de l'image
```

- On peut aussi définir une image de fond dans la fenêtre avec la fonction `background()`.

```
background(image);   // image de fond
```

Sauvegarde

Il est possible de sauvegarder l'image ou le dessin de la fenêtre, il suffit d'invoquer la fonction `save()` avec en argument le nom du fichier.

```
save("creation.jpg");     // sauvegarde de la fenêtre
```

Ainsi pour sauvegarder un objet `PImage`, il faudra au préalable l'afficher dans la fenêtre en la redimensionnant, puis utiliser la fonction `save()`.

Création

La création d'une image s'effectue avec la fonction `createImage()` en précisant en arguments sa largeur, sa hauteur et le mode `RGB`.

```
PImage img = new createImage(640, 480, RGB);
```

Modification

- La couleur du point situé en `(x,y)` s'obtient avec la méthode `get()`.

```
color c = img.get(x,y);    // couleur du point (x,y)
```

- La couleur de ce point peut être modifiée par la méthode `set()`.

```
img.set(x, y, c & 0xff);   // applique ici un filtre bleu
```

Exemple de traitement d'image

```
// Affiche l'image "chat.jpg" puis effectue un traitement
PImage img;
// lit le fichier image et redimensionne la fenêtre
void setup() {
    surface.setResizable(true);
    img = loadImage("chat.jpg");      // nom de l'image
    surface.setSize(img.width, img.height);
    println("Largeur de l'image:", img.width);
    println("Hauteur de l'image:", img.height);
}
// affiche l'image 60 fois par seconde
void draw() {
    image(img, 0, 0);           // affiche l'image
    if (frameCount == 4 * 60)  // 4 secondes écoulées
        filtrerVert();          // applique le filtre vert
}
// conserve uniquement la composante verte des points
void filtrerVert() {
    for (int y = 0; y < img.height; y++) {
        for (int x = 0; x < img.width; x++) {
            color c = img.get(x, y);
            img.set(x, y, c & 0xff00);  // masque binaire
        }
    }
}
// sauvegarde l'image au clic de souris
void mouseClicked() {
    save("chat-vert.jpg");     // image créée
}
```

Activités

Gris

Réalisez un programme qui convertie une image couleur en niveau de luminosité suivant la norme de conversion `R'G'B'-Y'CbCr` utilisée par le format d'image JPEG.

Inversion des couleurs

Réalisez un programme qui renverse chaque composante rouge, verte et bleue de l'ensemble des points d'une image. Ainsi le rouge 0 devient 255, le vert 37 devient 218, …

Miroir

Que fait le programme suivant ?

```
PImage imgsrc, imgdst;
void setup() {
    surface.setResizable(true);
    imgsrc = loadImage("chat.jpg");   // image source
    surface.setSize(imgsrc.width, imgsrc.height);
    imgdst = miroir(imgsrc);          // image transformée
}
void draw() {
    image(imgdst, 0, 0);
}
PImage miroir(PImage imgsrc) {
    int largeur = imgsrc.width;
    int hauteur = imgsrc.height;
    PImage img = createImage(largeur, hauteur, RGB);
    for (int y = 0; y < hauteur; y++) {
        for (int x = 0; x < largeur; x++) {
            color c = imgsrc.get(x, y);
            img.set(largeur - x - 1, y, c);
        }
    }
    return img;
}
```

Symétrie

Réalisez un programme qui effectue la symétrie d'une image, suivant un axe centré horizontal.

Quadrants

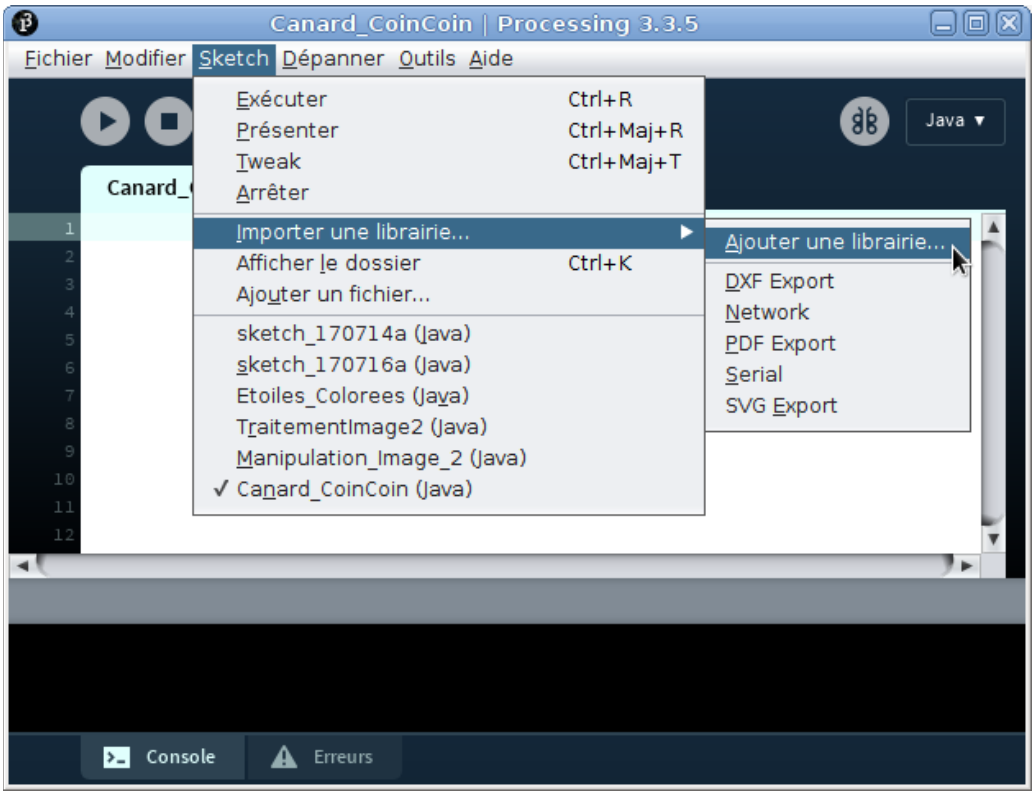
Reprenez le programme précédent divisant la taille d'une image par quatre (largeur et hauteur seront divisées par deux) et en l'affichant dans le coin supérieur gauche.

Vous convertirez ensuite cette image réduite en chacune des trois composantes `Y'`, `Cb` et `Cr` pour les afficher dans le coin inférieur droit pour `Y'`, inférieur gauche pour `Cb` et supérieur droit pour `Cr`.

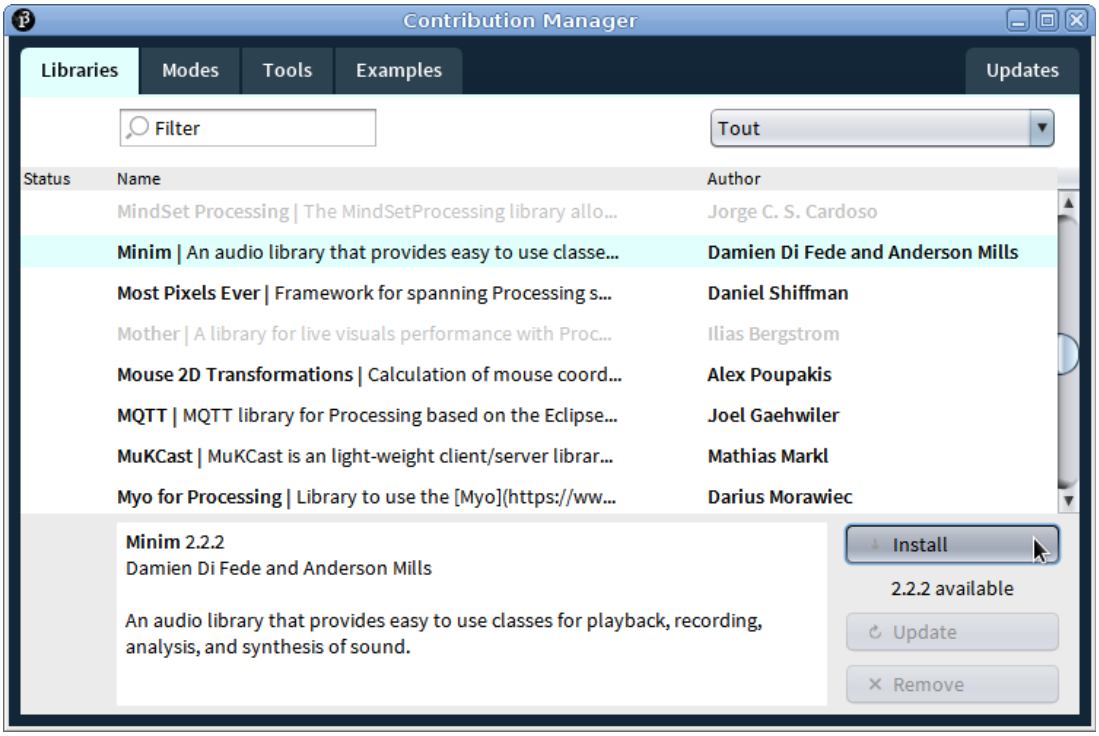
14. Fichier son sur Processing

Installation de la bibliothèque sonore

Les fonctionnalités de gestion du son, sont accessibles à travers la bibliothèque `Minim` qu'il faut commencer par installer dans Processing, en suivant les deux étapes suivantes :



On choisit `Minim` et clique sur `Install` pour procéder au téléchargement et à l'installation.



Importation

La bibliothèque `Minim` rassemble plusieurs classes dans un même paquetage (*package*). Ces classes utilisent des noms hiérarchisés de la forme `ddf.minim.Minim` afin d'empêcher les conflits avec d'autres classes qui pourraient porter le même nom.

On peut éviter d'avoir à taper ces noms complets et adapter l'espace de nommage aux noms relatifs, en important toutes les classes de `ddf.minim` avec la commande `import`.

```
import ddf.minim.*;     // importe les classes de ddf.minim
```

Ainsi le constructeur `ddf.minim.Minim()` peut dorénavant s'appeler `Minim()`.

Ceci permet, par la même occasion, de préciser au compilateur Java où trouver ces classes.

Initialisation

Avant de charger un fichier son, on doit créer et initialiser un objet de la classe `Minim` en invoquant son constructeur auquel on donne comme argument `this`, qui se réfère à l'objet que représente notre programme. Ceci va permettre de jouer le fichier son en parallèle à l'exécution du programme.

```
Minim minim = new Minim(this);      // crée un objet Minim
```

Chargement

Le chargement d'un fichier son se fait en appelant la méthode `loadFile()` avec le chemin d'accès au fichier son (relatif au sous-répertoire `data`) en argument. Cette fonction renvoie un objet de type `AudioPlayer`.

```
AudioPlayer sonCanard = minim.loadFile("coincoin.wav");
```

Lecture

L'objet `AudioPlayer` précédent peut maintenant être joué avec la méthode `play()`.

```
sonCanard.play();   // joue sonCanard de type AudioPlayer
```

Lecture répétée

Le son peut être répété de deux façons distinctes :

- La lecture est reprise automatiquement à la fin de la lecture. Dans ce cas on utilise la méthode `loop()` qui prend en argument un entier positif qui indique le nombre de répétitions à effectuer à la suite, ou rien pour une répétition perpétuelle.

```
sonCanard.loop(1);   // joue 2 fois de suite sonCanard
```

- La lecture est reprise seulement à la demande. Dans ce cas on utilise la méthode `rewind()` pour se replacer au début du son et pouvoir ré-invoquer `play()`.

```
sonCanard.rewind();   // «rembobine» sonCanard
sonCanard.play();     // joue sonCanard
```

Activités

Formats

Quels sont les formats audio pris en charge par `Minim` ?

Que fait le programme suivant ?

```
import ddf.minim.*;
AudioSample sonCanard;
float diametre = 0;
void setup() {
    size(400,300);
    Minim minim = new Minim(this);
    AudioPlayer ambiance = minim.loadFile("nature.mp3");
    ambiance.play();
    ambiance.loop();
    sonCanard = minim.loadSample("coincoin.wav",32);
}
void draw() {
    background(#cc8800);
    ellipse(width/2, height/2, diametre, diametre);
    diametre = (diametre > 3 ? diametre/1.1 : 0);
}
void mouseClicked() {
    sonCanard.trigger();
    diametre = 150;
}
```

Effet sonore

Reprenez le programme de la balle qui rebondit sur les bords de la fenêtre et rajoutez un effet sonore audible à chaque rebond.

Vous utiliserez la classe `AudioSample` et la méthode `trigger()` plus adaptées pour cet effet.

Lecteur

Réalisez un programme affichant les commandes de lecture, pause, rembobinage, réglage du volume, pour jouer un son ou une musique.

Vous consulterez pour cela la documentation officielle de `Minim` à l'adresse : `http://code.compartmental.net/minim/` et utiliserez la leçon suivante sur la gestion des événements souris et clavier.

FFT

- Que signifie FFT ?
- À quoi sert la FFT ?
- Mettez en œuvre le programme donné en exemple dans la documentation de `Minim` à l'adresse `http://code.compartmental.net/minim/fft_method_forward.html`

15. Clavier et souris sur Processing

Gestion des événements Souris

Position du pointeur

Les coordonnées du pointeur de souris, par rapport à l'angle supérieur gauche de la fenêtre, sont données par les variables globales `mouseX` et `mouseY`.

```
void setup() {
    size(640,480);
}
// dessine un disque blanc centré sur le pointeur
void draw() {
    background(0);
    ellipse(mouseX, mouseY, 20, 20);
}
```

Clic de souris

L'appui sur un des boutons de la souris est indiqué par la variable booléenne globale `mousePressed`. Le numéro du bouton qui est donné par la variable entière globale `mouseButton` peut être comparé aux constantes `LEFT`, `CENTER`, `RIGHT` plus explicites.

```
void setup() {
    size(640,480);
    textAlign(CENTER);
    textSize(60);
}
// indique au centre de la fenêtre le bouton cliqué
void draw() {
    background(0);
    if (mousePressed) {
        String bouton = "Bouton ";
        switch(mouseButton) {
        case LEFT:
            bouton += "gauche";
            break;
        case CENTER:
            bouton += "du milieu";
            break;
        case RIGHT:
            bouton += "droit";
            break;
        }
        text(bouton, width/2, height/2);
    }
}
```

Fonctions de rappel

Deux fonctions de rappel (*callback*) `void mousePressed()` et `void mouseClicked()` peuvent être définies pour réagir à tout appui ou relâchement d'un bouton de la souris.

Gestion des événements Clavier

Appui sur une touche

L'appui d'une touche est indiqué par la variable booléenne globale `keyPressed`.

La variable globale `key` contient alors le caractère associé à la touche pressée, ou la constante particulière `CODED` si ce caractère n'est pas défini.

Ceci permet de traiter le cas des touches spéciales : *Fléchées*, *Shift*, *Alt*, *Ctrl* en affectant à la variable globale `keyCode` une des constantes : `UP/DOWN/LEFT/RIGHT`, `SHIFT`, `ALT`, `CONTROL`

```
void setup() {
    size(640,480);  textAlign(CENTER);  textSize(40);
}
void draw() {
    background(#3366aa);
    String msg = "Appuie sur une flèche";
    if (keyPressed) {
        msg = "Perdu !";
        if (key == CODED) {
            switch (keyCode) {
            case UP: case DOWN: case LEFT: case RIGHT:
                      msg = "Gagné !";  break;
            default:  msg = "Toujours pas...";
            }
        }
    }
    text(msg, width/2, height/2);
}
```

Remarque

Suivant le système d'exploitation, la constante `ENTER` (caractère ASCII 10 – LF) associée à la touche *Entrée*, peut être remplacée par la constante `RETURN` (caractère ASCII 13 – CR). Il est donc recommandé de tester aussi cette seconde valeur de la variable `key`.

Fonctions de rappel

Les deux fonctions de rappel `void keyPressed()` et `void keyReleased()` peuvent être définies pour réagir à tout appui ou relâchement d'une touche du clavier.

```
void setup() {
    size(640,480);
    background(#33aa66);
}
// dessine un disque blanc à la position du pointeur
// quand le bouton gauche de la souris est cliqué
void draw() {
    if (mousePressed && mouseButton == LEFT)
        ellipse(mouseX, mouseY, 40, 40);
}
// vide la fenêtre à chaque appui sur la touche 'z'
void keyPressed() {
    if (key == 'z')
        background(#33aa66);
}
```

Activités

Souris pressée

Quel problème rencontre-t-on à lire directement la valeur de `mouseButton` sans avoir testé au préalable l'état de `mousePressed` ?

Une souris et des boutons

Que fait le programme suivant ?

```
color[] couleurs = { #3388aa, #3388aa, #3388aa };
void setup() {
    size(300,480);
    background(#11aa66);
    stroke(0);
}
void draw() {
    for (int i = 0; i < 3; i++) {
        fill(couleurs[i]);
        rect(30, 30 + 150*i, 240, 100);
    }
}
void mouseClicked() {
    for (int i = 0; i < 3; i++)       // réinitialise
        couleurs[i] = #3388aa;        // les couleurs
    if (mouseX < 30 || mouseX > 270)      // contrôle
        return;                           // horizontal
    for (int i = 0; i < 3; i++)
        if (30 + 150*i < mouseY && mouseY < 130 + 150*i) {
            couleurs[i] = #22aaff;
            break;
        }
}
```

Dessin à la souris

Vous allez créer un programme de dessin rudimentaire à la souris.

Il affichera une fenêtre au fond noir, avec un bandeau sur le côté gauche. Celui-ci sera formé de carrés empilés verticalement et affichant en leur centre une icône représentant les différentes fonctions du logiciel :

- Plusieurs choix d'épaisseurs de trait.
- Plusieurs choix de gommes : traçant en noir.
- Plusieurs choix de couleurs : primaires, secondaires et blanc.
- Un bouton pour effacer l'image entière.
- Un bouton pour enregistrer l'image de la fenêtre sous le nom « *dessin.png* ».

Indication : La position fixe du bandeau et des carrés qui le composent, permettront de déterminer facilement à quelle fonctionnalité associer un clic de souris dans ce bandeau.

16. Fichier texte sur Processing

Lecture d'un fichier

La fonction `loadStrings()` permet de lire les lignes d'un fichier texte au format UTF8. Elle prend en argument une chaîne correspondant au chemin d'accès au fichier (qui est relatif au répertoire du programme) et renvoie un tableau `String[]`, dont les éléments sont les lignes du fichier.

```
// lecture et affichage du contenu de "fichier.txt"
String[] lignes = loadStrings("fichier.txt");
if (lignes == null)  return;     // si problème de lecture
for (int i = 0; i < lignes.length; i++)
    println(lignes[i]);
```

Remarques

- En cas d'échec la fonction `loadStrings()` renvoie la référence `null` qu'il convient de tester au risque de provoquer une erreur d'exécution par la suite en voulant utiliser le tableau
- La fonction `loadStrings()` accepte aussi les URL en argument, ceci permet de charger le contenu d'une page web en mémoire.

Écriture d'un fichier

De la même façon, la fonction `saveStrings()` permet d'enregistrer le contenu d'un tableau de chaînes de caractères dans un fichier texte au format UTF8. Cette fonction prend deux arguments : en premier la chaîne correspondant au chemin d'accès au fichier à créer et en second le tableau de chaînes de caractères à enregistrer.

```
// écriture dans le fichier "nouveau.txt"
String[] lignes = { "Une 1ère ligne", "Une 2nde" };
saveStrings("nouveau.txt", lignes);
```

Remarques

- Si le fichier existe déjà, son contenu sera préalablement supprimé. Avec cette fonction, pour rajouter des lignes à un fichier existant, il faut donc commencer par lire son contenu dans le tableau…
- La fonction `saveStrings()` crée les répertoires indiqués dans le chemin si ceux-ci n'existent pas encore.
- En cas d'échec de création du fichier ou d'enregistrement des données, la fonction `saveStrings()` génère une Exception qui, si elle n'est pas attrapée, cause l'arrêt du programme. On utilise pour cela la structure de contrôle `try-catch`.

```
// gestion d'une erreur d'écriture
String[] lignes = { "ligne1", "ligne2", "ligne3" };
try {                                  // exécution contrôlée
    saveStrings("/acces-interdit/fichier.txt", lignes);
} catch (NullPointerException e) {  // gestion de l'erreur
    println("Erreur de sauvegarde");
}
```

Fenêtre de sélection de fichier

La demande d'un nom de fichier à travers une boite de dialogue de sélection de fichier, peut être réalisée avec la fonction `selectInput()`. Il s'agit d'une fonction non bloquante, elle ne suspend pas l'exécution du programme le temps que l'utilisateur sélectionne un fichier. En conséquence, elle utilise une fonction de rappel (*callback*) qui sera appelée une fois la sélection effectuée.

La fonction `selectInput()` a ainsi deux arguments : une chaîne qui apparaît en titre de la fenêtre de sélection, et une seconde chaîne portant le nom de la fonction de rappel.

La fonction de rappel prend pour seul argument, un objet de type `File` qui avec la méthode `getAbsolutePath()` permet d'obtenir le chemin du fichier sélectionné. En cas d'annulation de l'utilisateur c'est la référence `null` qui est transmise.

Voici une mise en œuvre de la fonction `selectInput()` avec une animation en parallèle :

```
// boite de dialogue de sélection de fichier
String nomFichier;          // initialisé à null par défaut
// initialisations et appel à selectInput()
void setup() {
    size(800,200);
    textSize(24);
    textAlign(CENTER);
    stroke(255);
    noFill();
    selectInput("Choisir un fichier", "choisirFichier");
}
// joue une animation jusqu'à ce qu'un fichier soit choisi
void draw() {
    background(#5555bb);
    if (nomFichier == null)
        animation(frameCount);
    else
        text(nomFichier, width/2, height/2 + 10);
}
// affiche le temps écoulé dans un rectangle mouvant
void animation(int k) {
    int r = (k & 64) == 0 ? 80-(k&63) : 16+(k&63);
    rect(r, r, width-2*r, height-2*r);
    text(frameCount / frameRate, width/2, height/2 + 10);
}
// fonction de rappel utilisée par selectInput()
void choisirFichier(File fichier)
{
    nomFichier = "(annulé)";
    if (fichier != null)
        nomFichier = fichier.getAbsolutePath();
}
```

Remarque

On distingue en fait deux fonctions de sélection :

- `selectInput()` destinée à choisir un fichier en lecture
- `selectOutput()` destinée à choisir un fichier en écriture

Activités

Affichage

Réalisez un programme qui lit un fichier texte puis l'affiche dans la fenêtre principale. Vous veillerez à ce que le texte ne sorte pas de la fenêtre et à ce que les sauts de lignes soient conservés.

Inversion

Réalisez un programme qui lit un fichier texte puis échange les lignes du tableau de sorte que la première chaîne devienne la dernière et inversement ; puis sauvegarde le tableau dans un nouveau fichier.

Statistiques

Réalisez un programme qui lit un fichier texte et qui donne les fréquences d'apparition des lettres de l'alphabet ; en ne faisant aucune distinction entre majuscules et minuscules, et en ignorant les autres caractères.

Question-Réponses

```
String[] donnees = { "id,question,reponse,propos1,propos2,propos3\n" +
  "0,Quelle est la capitale de la France ?,2,Londres,Paris,Madrid\n" +
  "1,Qu'est ce qu'un chat ?,3,Un minéral,Un végétal,Un animal\n"     +
  "2,Quelle couleur est aussi un fruit ?,1,Orange,Jaune,Bleu" };
saveStrings("data/jeu.csv", donnees);       // crée le fichier jeu.csv
```

```
Table table;                  // voir la documentation de Processing
TableRow ligne;               // sur le type Table et les fichiers csv
String question, propos1, propos2, propos3;
char reponse;
int score = 0, index = 0;
void setup() {
    size(800,600);  textSize(24);  textAlign(CENTER);
    table = loadTable("jeu.csv", "header");   // lit le fichier jeu.csv
    recupererInfos(index);
}
void draw() {
    background(#3366dd);
    text(question, width/2, 150);     // affiche la question et
    text(propos1,  width/2, 250);     // les 3 propositions de réponse
    text(propos2,  width/2, 350);
    text(propos3,  width/2, 450);
}
void recupererInfos(int i) {          // lit les infos de la question i
    if (i >= table.getRowCount()) {   // pas d'autre question
        question = "Score final : " + score + " point(s)"; // vide tout
        propos1 = propos2 = propos3 = "";        // et affiche le score
        reponse = ESC;  return;       // désactivation
    }
    ligne = table.getRow(i);
    question = ligne.getString("question");
    reponse = ligne.getString("reponse").charAt(0);
    propos1 = "1. " + ligne.getString("propos1");
    propos2 = "2. " + ligne.getString("propos2");
    propos3 = "3. " + ligne.getString("propos3");
}
void keyReleased() {
    if ('1' <= key && key <= '3') {
        if (key == reponse)  score++;  // vérifie avec la réponse
        recupererInfos(++index);       // passe à la question suivante
    }
}
```

17. Quelques conseils

Nommage

Choisissez des noms parlants pour vos variables, classes, fonctions et méthodes en respectant la notation CamelCase habituelle en Java. Ceci facilite la lecture et la compréhension de votre code. Elle consiste à juxtaposer les mots qui composent le nom et mettant leur première lettre en majuscule. Avec quelques variantes suivant le rôle à jouer.

- Une classe commence par une majuscule :

```
class Voilier { … }          class MoteurAvion { … }
```

- Une variable ou un attribut commence par une minuscule :

```
int somme;                   float tarifReduit;
```

- Une fonction ou méthode commence par un verbe et une minuscule :

```
void manger()                String direBonjour()
```

- Une constante est entièrement en majuscules avec ses mots séparés d'un underscore :

```
final float PI = 3.1415;     final int NB_ROUES = 4;
```

Bien entendu, on peut aussi utiliser les noms courts, surtout avec des variables locales :

- `i`, `j`, `k` pour les indices dans un tableau
- `x`, `y`, `z` pour des flottants
- `n`, `m`, `p`, `q` pour des entiers
- `r` pour un rayon
- […]

Lisibilité

Aérez votre code, rajoutez des espaces, des lignes blanches... Alignez les opérations et les codes répétitifs sur plusieurs lignes…

Ainsi `x = 3 * a + 7;` vaut mieux que `x=3*a+7;`

Chaque fonction se doit de ne faire qu'une chose, aussi n'hésitez pas à scinder vos fonctions.

Commentaires

Tâchez de commenter votre code, mais avec modération. Votre commentaire doit être constructif. Il est inutile d'écrire : `i = i + 1;  // incrémenter i`

Précisez la signification de vos variables globales et/ou dans vos classes.

Commentez enfin vos fonctions et vos méthodes :

- rédigez une phrase d'explication sur ce qu'elles font
- indiquez les différents paramètres (type et nom) et leur signification
- indiquez la valeur de retour (type) et l'interprétation qu'on doit lui donner

À cet égard, la syntaxe Javadoc est une piste intéressante à suivre.