

Info 2 SCP

Structures de données avancées

Damien Sauveron

<http://damien.sauveron.fr/>

Rappels sur les types de base et les déclarations

- Types numériques : `int` ou `float`
- Types alphanumériques : `char` ou `String`
- Type logique (booléen) : `boolean` (valeur : `true` ou `false`)
- Déclaration :
`type nom_var;`
- exemple :
`int toto;`

De nouveaux types : les types composés

- Fréquemment, le concepteur d'algorithmes a besoin de types plus adaptés aux données qu'il veut manipuler.
- Par exemple, s'il veut faire des manipulations sur des informations concernant des relevés de température, idéalement, il souhaiterait disposer d'un type approprié pour faciliter l'accès aux différentes parties les constituant. Prenons d'un relevé de température « simplifié » : il contient le lieu, la date, la température.

Les types composés

- Malheureusement, un tel type n'existe pas de base dans les langages de programmation (c'est aussi le cas en Processing). D'ailleurs on constate que les informations qu'il définit sont complexes car de natures différentes : certaines textuelles (le lieu), certaines numériques (la température) et même d'autres qui sont de nature composés (la date)
- Afin de permettre au développeur de manipuler ces données complexes en cohérence, il existe dans la plupart des langages la possibilité de définir des types composés.
 - En Processing, ça sera au travers de la **notion de classe**.

Les types composés – La classe

- Une classe consistera (pour nous) en un nombre fixe de champs de différents types.
- Le nom de la classe servira à nommer le nouveau type.
- Dans la classe, chaque champ est défini par :
 - un type (simple ou composé)
 - un identificateur (le nom du champ)

Les types composés – La classe

- La syntaxe est :

```
class Non_du_type_compose
{
    type1 nom_du_champ1;
    type2 nom_du_champ2;
    ...
    typen nom_du_champn;
}
```

```
class Releve
{
    String lieu;
    Date date;
    float temperature;
}
```

- Exemple :

```
class Date
{
    int jour;
    int mois;
    int annee;
}
```

Quelques bonnes pratiques

- Attention il s'agit de bonnes pratiques et pas de contrainte syntaxique !
- Le nom d'une classe commence toujours par une majuscule
 - `class Ami` suit les recommandations
 - `class cestmal` ne les suit pas !
- Le nom d'un champ d'une classe commence toujours par une minuscule

```
class BonnesPratiques
{
    int cEstBien;
    int CEstMal;
    boolean superJoli;
}
```

Définition et utilisation

- Une fois, la classe (un type composé) définie, il est possible :
 - de déclarer des variables de ce nouveau type
 - de créer l'espace mémoire nécessaire en utilisant le mot clé « new » (on dit aussi allouer)
 - de les utiliser.
- L'accès aux champs se fait au travers de l'opérateur « . » comme nous l'illustrons ci-dessous.

```
void setup() {  
    Date d = new Date();  
    d.jour = 23;  
    d.mois = 1;  
    d.annee = 2017;  
}
```

Ca fonctionne mais
c'est pas très joli !

Pour une meilleure utilisation : le constructeur

- Il est possible d'ajouter un « constructeur » qui explique comment la variable du type déclaré doit être initialisée. Cela permet d'avoir du code plus lisible ensuite.

- Exemple :

```
class Date
{
    int jour;
    int mois;
    int annee;
    Date(int j, int m, int a)
    {
        jour = j;
        mois = m;
        annee = a;
    }
}

void setup() {
    Date d = new Date(23, 1, 2017);
}
```

- **Obligation syntaxique :**

- Le constructeur porte toujours le nom de la classe !
- Exemple
 - Ici nous avons la classe Date
 - Le constructeur s'appelle donc Date

Constructeur et conflit

- S'il y a un conflit de nommage entre les noms des champs et ceux des paramètres du constructeur, il faut utiliser le mot clé « `this` » pour désigner les champs de la classe et lever l'ambiguïté.

```
class Date
{
    int jour;
    int mois;
    int annee;
    Date(int jour, int mois, int annee)
    {
        this.jour = jour;
        this.mois = mois;
        this.annee = annee;
    }
}

void setup() {
    Date d = new Date(23, 1, 2017);
}
```

Retour sur notre exemple

- Ainsi pour définir le type Releve:

```
class Date
{
    int jour;
    int mois;
    int annee;

    Date(int j, int m, int a)
    {
        jour = j;  mois = m; annee = a;
    }
}

class Releve
{
    String lieu;
    Date date;
    float temperature;

    Releve(String l, int j, int m, int a, float t)
    {
        lieu = l;
        date = new Date(j, m, a);
        temperature = t;
    }
}

void setup() {
    Releve r = new Releve("Limoges", 23, 1, 2017, 35.52);
}
```

Affichage une variable de type composé

- Il faut définir une fonction pouvant afficher ce type d'information
 - Par exemple pour Releve:

```
void afficheReleve(Releve r) {  
    println("Le " + r.date.jour + "/" + r.date.mois + "/"  
+ r.date.annee + " à " + r.lieu + "la température était de "  
+ r.temperature+"°C");  
}
```

```
void setup() {  
    Releve r = new Releve("Limoges", 23, 1, 2017, 35.52);  
    afficheReleve(r);  
}
```

```
// Le 23/1/2017 à Limoges la température était de 35.52°C
```

Affectation de variables de type composé

- Attention à la façon de copier !!

```
class Date
{
    int jour;
    int mois;
    int annee;
    Date(int j, int m, int a)
    {
        jour = j;  mois = m; annee = a;
    }
}

void afficheDate(Date d) {
    println(d.jour + "/" + d.mois + "/" + d.annee);
}

void setup() {
    Date d1 = new Date(23, 1, 2017);
    Date d2 = d1;
    afficheDate(d1); // affiche 23/1/2017
    afficheDate(d2); // affiche 23/1/2017
    d2.jour = 31;
    afficheDate(d1); // affiche 31/1/2017
    afficheDate(d2); // affiche 31/1/2017
}
```

C'est un problème !

Affectation de variables de type composé

- Méthode 1 : Il faut faire la copie des champs à la main !

```
class Date
{
    int jour;
    int mois;
    int annee;
    Date(int j, int m, int a)
    {
        jour = j;  mois = m; annee = a;
    }
}

void afficheDate(Date d) {
    println(d.jour + "/" + d.mois + "/" + d.annee);
}

void setup() {
    Date d1 = new Date(23, 1, 2017);
    Date d2 = new Date();
    d2.jour = d1.jour;
    d2.mois = d1.mois;
    d2.annee = d1.annee;
    afficheDate(d1); // affiche 23/1/2017
    afficheDate(d2); // affiche 23/1/2017
    d2.jour = 31;
    afficheDate(d1); // affiche 23/1/2017
    afficheDate(d2); // affiche 31/1/2017
}
```

← `Date d2 = new Date(d1.jour, d1.mois, d1.annee);`

OK, mais c'est pas
pratique ! ☹️

Affectation de variables de type composé

- Méthode 2 : Ajouter un second constructeur spécifique !

```
class Date
{
    int jour;
    int mois;
    int annee;
    Date(int j, int m, int a)
    {
        jour = j;  mois = m; annee = a;
    }

    Date(Date d)
    {
        jour = d.jour;  mois = d.mois; annee = d.annee;
    }
}

void afficheDate(Date d) {
    println(d.jour + "/" + d.mois + "/" + d.annee);
}

void setup() {
    Date d1 = new Date(23, 1, 2017);
    Date d2 = new Date(d1);
    afficheDate(d1); // affiche 23/1/2017
    afficheDate(d2); // affiche 23/1/2017
    d2.jour = 31;
    afficheDate(d1); // affiche 23/1/2017
    afficheDate(d2); // affiche 31/1/2017
}
```

C'est bien mieux ! 😊

Types composés et fonction

- Comme pour les autres types, une fonction peut recevoir en paramètre un type composé.
 - Elle peut lire les valeurs comme nous l'avons vu avec `afficheDate`

```
void afficheDate(Date d) {  
    println(d.jour + "/" + d.mois + "/" + d.annee);  
}
```

- Elle peut aussi modifier les valeurs de telle sorte que la variable de l'appelant est modifiée

```
void modifierDate(Date d) {  
    if (d.jour == 14 && d.mois == 2) {  
        d.jour = 15;  
    }  
}  
  
void setup() {  
    Date d1 = new Date(14, 2, 2017);  
    afficheDate(d1); // affiche 14/2/2017  
    modifierDate(d1);  
    afficheDate(d1); // affiche 15/2/2017  
}
```

Attention à bien utiliser
cette possibilité !

Types composés et fonction

- Attention à comment sont réalisées les modifications des paramètres d'entrée de type composé

- Par exemple:

```
void modifierDate2(Date d) {  
    // j'aime bien Noël !  
    d = new Date(25,12,d.annee);  
}
```

Ne fonctionnera pas !

```
void setup() {  
    Date d1 = new Date(23, 1, 2017);  
    afficheDate(d1); // affiche 23/1/2017  
    modifierDate2(d1);  
    afficheDate(d1); // affiche 23/1/2017  
}
```

Types composés et fonction

- Une fonction peut aussi retourner une donnée de type composé

```
Date saisirDate() {  
    Date d;  
    int j, m, a;  
    j = askInteger("Veuillez saisir le jour");  
    m = askInteger("Veuillez saisir le mois");  
    a = askInteger("Veuillez saisir l'annee");  
    d = new Date(j, m, a);  
    return d;  
}  
void setup() {  
    Date x = saisirDate();  
    afficheDate(x);  
}
```

Types composés et tableaux

```
int TAILLE = 10;
```

```
void setup() {  
    Date[] dates = new Date[TAILLE];  
    for(int i=0;i<dates.length; i++) {  
        dates[i] = saisirDate();  
    }  
  
    for(int i=0;i<dates.length; i++) {  
        afficheDate(dates[i]);  
    }  
}
```

Attention si ici la fonction `saisirDate()` ne retournerait pas une date, il faudrait nous même les allouer avec `new Date()`

Types composés et tableaux

- Une autre syntaxe existe pour itérer **sur la totalité** des éléments d' un tableau

```
int TAILLE = 10;
```

```
void setup() {  
    Date[] dates = new Date[TAILLE];  
    for(int i=0;i<dates.length; i++) {  
        dates[i] = saisirDate();  
    }  
  
    for(Date d : dates) {  
        afficheDate(d);  
    }  
}
```

Recherche du maximum

- On considère le premier élément comme le maximum (ici le maximum sera suivant l'ordre chronologique)

```
int TAILLE = 10;
```

```
void setup() {  
    Date[] dates = new Date[TAILLE];  
    for(int i=0;i<dates.length; i++) {  
        dates[i] = saisirDate();  
    }  
  
    // on suppose qu'il y a au moins 1 élément  
    Date drecente = new Date(dates[0]);  
  
    for(Date d : dates) {  
        if (d.annee > drecente.annee){  
            drecente = new Date(d);  
        } else if (d.annee == drecente.annee && d.mois > drecente.mois){  
            drecente = new Date(d);  
        } else if (d.annee == drecente.annee && d.mois == drecente.mois && d.jour > drecente.jour){  
            drecente = new Date(d);  
        }  
    }  
  
    afficheDate(drecente);  
}
```

Trier un tableau

- C'est finalement définir une relation d'ordre entre les éléments
- En Processing, il est possible de trier un tableau de types de base en appelant la fonction `sort()`

```
float[] a = { 3.4, 3.6, 2, 0, 7.1 };  
a = sort(a);  
println(a)  
// Affiche le contenu du tableau trié  
// [0] 0.0  
// [1] 2.0  
// [2] 3.4  
// [3] 3.6  
// [4] 7.1
```

Super! 😊

Et sur des types composés ?

```
String[] s = { "deer", "elephant", "bear", "aardvark", "cat" };  
s = sort(s);  
println(s);  
// Affiche le contenu du tableau trié  
// [0] "aardvark"  
// [1] "bear"  
// [2] "cat"  
// [3] "deer"  
// [4] "elephant"
```

```
String[] s = { "deer", "elephant", "bear", "aardvark", "cat" };  
s = sort(s, 3);  
println(s);  
// Affiche le contenu du tableau trié des 3 premiers éléments  
// [0] "bear"  
// [1] "deer"  
// [2] "elephant"  
// [3] "aardvark"  
// [4] "cat"
```

Trier un tableau de type composé

- Pour trier un tel tableau, on peut implémenter un algorithme de tri.
 - Avantage : une maîtrise total de l'algorithme de tri (tri par recherche du maximum, tri rapide, tri par insertion, etc.)
 - Inconvénient : il faut l'implémenter ... et vous ne ferez pas toujours cela efficacement.
- Prenons un exemple :
 - Définir une classe `ReleveSimple` contenant le lieu, la température
 - Saisir les relevés de température et stocker les dans un tableau
 - Trier les relevés selon la température croissante
 - Afficher les relevés en donnant le même rang dans le classement à des lieux ayant la même température

Trier un tableau de type composé

```
class ReleveSimple
{
    String lieu;
    float temperature;
    ReleveSimple(String l, float t)
    {
        lieu = l;
        temperature = t;
    }
    ReleveSimple(ReleveSimple r){
        lieu = r.lieu;
        temperature = r.temperature;
    }
}

ReleveSimple saisirReleveSimple() {
    ReleveSimple r;
    String l;
    float t;
    l = askString("Veuillez saisir le lieu");
    t = askFloat("Veuillez saisir la température");
    r = new ReleveSimple(l, t);
    return r;
}

int TAILLE = 10;
void setup() {
    float temp_min;
    int indicemin, i, j, rang;
    ReleveSimple tmp;

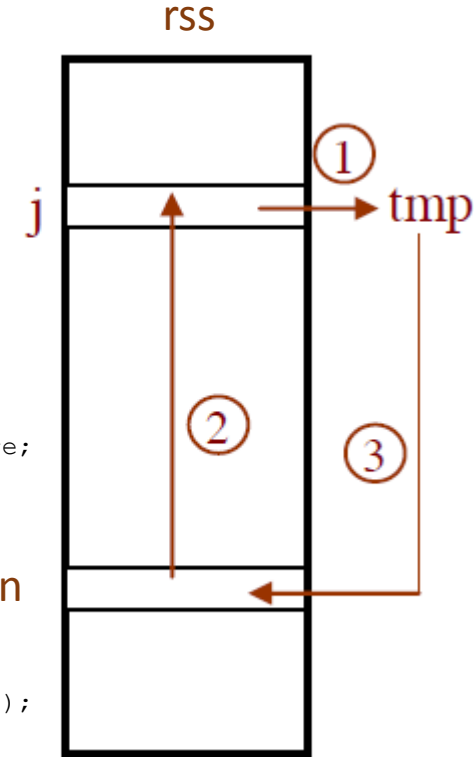
    ReleveSimple[] rss = new ReleveSimple[TAILLE];
    for(i=0;i<rss.length; i++) {
        rss[i] = saisirReleveSimple();
    }
}
```

```
// tri du tableau (tri par recherche du minimum)
for (j=0;j<TAILLE-1;j++) {
    temp_min = rss[j].temperature;
    indicemin = j;

    for(i = j+1; i<TAILLE; i++) {
        if (rss[i].temperature < temp_min) {
            temp_min = rss[i].temperature;
            indicemin = i;
        }
    }

    if (indicemin != j)
    {
        tmp = new ReleveSimple(rss[j]);
        rss[j] = new ReleveSimple(rss[indicemin]);
        rss[indicemin] = new ReleveSimple(tmp);
    }
}

// affichage
rang = 1;
for(int i=0;i<TAILLE; i++) {
    if(i!=0 && rss[i].temperature > rss[i-1].temperature)
        rang = i+1; // pour les températures égales
    println(rang + " " + rss[i].temperature + " " + rss[i].lieu);
}
}
```



Trier un tableau de type composé

- Pour trier un tel tableau, il faut définir une relation d'ordre (utilisée par `sort()`) entre les éléments de ce type.
- Pour cela il faudra :
 - Soit importer `java.util.Comparable`, l'implémenter et écrire une fonction `int compareTo(Object o)` dans la classe du type composé
 - Soit importer `java.util.Comparator` et écrire la relation d'ordre sous la forme d'une classe implémentant `Comparator`

Trier un tableau de type composé – Méthode 1

```
import java.util.*;
class Date implements Comparable
{
    int jour;
    int mois;
    int annee;
    Date(int j, int m, int a)
    {
        jour = j;  mois = m; annee = a;
    }
    int compareTo(Object o) { // pour un tri chronologique
        Date d = (Date)o;
        if (d.annee > annee){
            return -1;
        } else if (d.annee == annee && d.mois > mois){
            return -1;
        } else if (d.annee == annee && d.mois == mois && d.jour > jour){
            return -1;
        } else if (d.annee == annee && d.mois == mois && d.jour == jour){
            return 0;
        } else {
            return 1;
        }
    }
}

int TAILLE = 10;

void setup() {
    Date[] dates = new Date[TAILLE];
    for(int i=0;i<dates.length; i++) {
        dates[i] = saisirDate();
    }
    Arrays.sort(dates);
    for(Date d : dates) {
        afficheDate(d);
    }
}
```

Trier un tableau de type composé – Méthode 2

```
import java.util.*;
class Compareur1Date implements Comparator
{
    public int compare(Object o1, Object o2) { // selon le jour et à jour équivalent selon le mois et enfin selon l'année
        Date d1 = (Date)o1;
        Date d2 = (Date)o2;

        if (d2.jour > d1.jour){
            return -1;
        } else if (d2.jour == d1.jour && d2.mois > d1.mois){
            return -1;
        } else if (d2.jour == d1.jour && d2.mois == d1.mois && d2.annee > d1.annee){
            return -1;
        } else if (d2.jour == d1.jour && d2.mois == d1.mois && d2.annee == d1.annee){
            return 0;
        } else {
            return 1;
        }
    }
}

int TAILLE = 10;

void setup() {
    Date[] dates = new Date[TAILLE];
    for(int i=0;i<dates.length; i++) {
        dates[i] = saisirDate();
    }
    Arrays.sort(dates, new Compareur1Date()); // tri selon la méthode définie
    for(Date d : dates) {
        afficheDate(d);
    }
}
```

Trier un tableau de type composé – Méthode 2

```
import java.util.*;
class Compareur2Date implements Comparator
{
    public int compare(Object o1, Object o2) { // selon l'ordre chronologique inverse
        Date d1 = (Date)o1;
        Date d2 = (Date)o2;

        if (d1.annee > d2.annee){
            return -1;
        } else if (d1.annee == d2.annee && d1.mois > d2.mois){
            return -1;
        } else if (d1.annee == d2.annee && d1.mois == d2.mois && d1.jour > d2.jour){
            return -1;
        } else if (d1.annee == d2.annee && d1.mois == d2.mois && d1.jour == d2.jour){
            return 0;
        } else {
            return 1;
        }
    }
}

int TAILLE = 10;

void setup() {
    Date[] dates = new Date[TAILLE];
    for(int i=0;i<dates.length; i++) {
        dates[i] = saisirDate();
    }
    Arrays.sort(dates, new Compareur2Date()); // tri selon la méthode définie
    for(Date d : dates) {
        afficheDate(d);
    }
}
```