Les types enregistrements

premier pas vers les classes

NFA032

Présentation n°1

Rappels

- Les types primitifs
- Les types non primitifs
 - Les type tableaux
 - statiques `T []`
 - dynamiques `ArrayList<T>`
 - Le type `String`
 - Les types énumérés
- En Java, tout type non primitif est décrit par une classe

Les types enregistrements

- La notion de type enregistrement n'a pas d'existence en Java :
 - il s'agit simplement d'une classe sans méthode
 - résurgence d'une notion ancienne, remise au goût du jour par F. Barthélémy, appelée *record* en anglais (enregistrement), héritée des langages Pascal ou Ada
 - notion appelée *structure* (**struct**) en C/C++

Les types enregistrements

- Dans un tableau statique :
 - Toutes les composantes ont un même type T
 - Toutes les composantes sont indépendantes
 - Chacune porte une valeur, sa manipulation (lecture ou affectation) n'ayant aucune incidence sur les autres
 - Le nombre de composantes reste fixe
 - Chaque composante est identifiée par un indice entier
- Dans un enregistrement :
 - Chaque composante (aussi appelée *variable non statique*, *champ*, *attribut*) a son propre type
 - Toutes les composantes sont indépendantes
 - Le nombre de composantes reste fixe
 - Chaque composante est identifiée par un nom

Exemple : les individus

- Tout individu est caractérisé par son nom, son prénom, son âge, sa taille, son poids et par le fait qu'il porte ou pas des lunettes :

Version où le type de chaque champ (attribut) est indiqué

```
class Individu {  
    String nom;  
    String prénom;  
    int âge;  
    float taille;  
    float poids;  
    boolean porteDesLunettes;  
}
```

Version où le type des champs ayant le même type est factorisé

```
class Individu {  
    String nom,  
        prénom;  
    int âge;  
    float taille,  
        poids;  
    boolean porteDesLunettes;  
}
```

Les types enregistrements

- Tableaux et enregistrements relèvent de la même famille de construction de type : le produit de types
 - type T = ensemble de valeurs
 - $A \times B$ = type produit des types A et B
 - Un tableau $t = (v_0, \dots, v_{n-1})$ est de type $\overbrace{T \times \dots \times T}^{n \text{ fois}}$
 - on écrit $t[i]$ pour accéder à la composante n° i
 - Un enregistrement $e = (v_1, \dots, v_n)$ est de type $T_1 \times \dots \times T_n$
 - on écrit $e.m_i$ pour accéder à la composante de nom m_i .

Principales propriétés des enregistrements

- Le nom du type est libre (doit se conformer à la règle d'écriture des identificateurs Java)
- Les noms des champs sont libres (idem : doivent être des identificateurs)
- Si m est le nom d'un champ d'un type enregistrement E et o un objet de type E , alors :
 - $o.m$ permet de lire la valeur de la composante m de o
 - $o.m = v$ est l'affectation qui permet de modifier la valeur de la composante m de o .
- L'opérateur **new** permet de créer un nouvel objet :
$$E \ o = \mathbf{new} \ E ();$$
- Tous les champs d'un objet créé sont initialisés au zéro de leur type respectif

Exemple : les individus

```
{  
    Individu I1 = new Individu();  
    I1.nom = "Dubois";  
    I1.prénom = "Charles";  
    I1.âge = 32;  
    I1.taille = 187;  
    I1.poids = 91;  
    I1.porteDesLunettes = false;  
    ...  
}
```

Exemple : les points du plan

- Ecrire le code Java permettant de :
 - Représenter des points du plan (coordonnées cartésiennes x et y)
 - Créer un nouveau point à partir d'informations diverses (des coordonnées, un point existant, ...)
 - Afficher les informations d'un point
 - Réinitialiser les coordonnées d'un point
 - Déplacer un point selon une translation (Δ_x, Δ_y)

Les points du plan

- Deux solutions différentes présentées
 - Point = objet tableau
 - Point = objet d'un type enregistrement

Un point = un tableau (1/2)

- Exemple d'utilisation

- Extrait du programme de test

```
Terminal.ecrireStringln("Petit jeu avec un point");
Terminal.ecrireStringln("=====");
Terminal.ecrireString("point P = ");
double[] P = créer_point(2, 3);
afficher(P);
Terminal.sautDeLigne();
double dx = -1, dy = 2;
Terminal.ecrireString("P est déplacé de (" + dx + ", " + dy + ") => point P = ");
déplacer(P, dx, dy);
afficher(P);
Terminal.sautDeLigne();
```

- Trace d'exécution

```
Petit jeu avec un point
=====
point P = ( 2.0 , 3.0 )
P est déplacé de (-1.0,2.0) => point P = ( 1.0 , 5.0 )
```

Un point = un tableau (2/2)

- Définitions des fonctions en Java

```
// 1ère fonction constructeur : des coordonnées
static double[] créer_point(double x, double y) {
    double[] p = new double[2];
    p[0] = x; p[1] = y;
    return p;
}
```

```
// 2ème fonction constructeur : un point existant (constructeur de copie/clonage)
static double[] créer_point(double[] p) { return créer_point(p[0], p[1]); }
```

```
static void initialiser(double[] p, double x, double y) { p[0] = x; p[1] = y; }
```

```
static void afficher(double[] p) {
    Terminal.ecrireString("( "+p[0]+" , "+p[1]+" )");
}
```

```
static void déplacer(double[] p, double dx, double dy) { p[0] += dx; p[1] += dy; }
```

```
static void déplacer(double[] p, double[] d) { déplacer(p, d[0], d[1]); }
```

Un point = un type enregistrement (1/2)

- Définitions en Java

```
class point {  
    double x,  
           y;  
}
```

```
// 1ère fonction constructeur : des coordonnées  
static point créer_point(double x, double y) {  
    point p = new point();  
    p.x = x; p.y = y;  
    return p;  
}
```

```
// 2ème fonction constructeur : un point existant (constructeur de copie/clonage)  
static point créer_point(point p) { return créer_point(p.x, p.y); }
```

```
static void initialiser(point p, double x, double y) { p.x = x; p.y = y; }
```

```
static void afficher(point p) {  
    Terminal.ecrireString("( "+p.x+" , "+p.y+" )");  
}
```

```
static void déplacer(point p, double dx, double dy) { p.x += dx; p.y += dy; }
```

```
static void déplacer(point p, point d) { déplacer(p, d.x, d.y); }
```

Un point = un type enregistrement (2/2)

- Extrait du programme de test

```
Terminal.ecrireStringln("Petit jeu avec un point");
Terminal.ecrireStringln("=====");
Terminal.ecrireString("point P = ");
point P = créer_point(2, 3);
afficher(P);
Terminal.sautDeLigne();
double dx = -1, dy = 2;
Terminal.ecrireString("P est déplacé de (" + dx + ", " + dy + ") => point P = ");
déplacer(P, dx, dy);
afficher(P);
Terminal.sautDeLigne();
```

Manipulation 1 : les fonctions sont intégrées au type enregistrement

```
class point {
    double x,
           y;

    // fonctions de manipulation des points
    // =====

    // 1ère fonction constructeur : des coordonnées
    static point créer_point(double x, double y) {
        point p = new point();
        p.x = x; p.y = y;
        return p;
    }

    // 2ème fonction constructeur : un point existant (constructeur de copie/clonage)
    static point créer_point(point p) { return créer_point(p.x, p.y); }

    static void initialiser(point p, double x, double y) { p.x = x; p.y = y; }

    static void afficher(point p) {
        Terminal.ecrireString("( "+p.x+" , "+p.y+" )");
    }

    static void déplacer(point p, double dx, double dy) { p.x += dx; p.y += dy; }

    static void déplacer(point p, point d) { déplacer(p, d.x, d.y); }
}
```

Manipulation 1 : transformation du code du test

```
Terminal.crireStringln("Petit jeu avec un point");
Terminal.crireStringln("=====");
Terminal.crireString("point P = ");
point P = point.créer_point(2, 3);
point.afficher(P);
Terminal.sautDeLigne();
double dx = -1, dy = 2;
Terminal.crireString("P est déplacé de (" + dx + ", " + dy + ") => point P = ");
point.déplacer(P, dx, dy);
point.afficher(P);
Terminal.sautDeLigne();
```

- Remarque :
 - On peut effacer le `_point` des fonctions constructeurs : `point.créer(...)`

Exercice : ajouter des segments

- Compléter le code pour permettre la manipulation de segments :
 - Un segment est caractérisé par ses deux points extrémités
 - Les services attendus :
 - Affichage des caractéristiques d'un segment
 - Déplacement d'un segment selon une translation (Δ_x , Δ_y)
 - Calcul de la longueur d'un segment

Un segment = un tableau (1/3)

```
// 1ère fonction constructeur : coordonnées des extrémités
static double[][] créer_segment(double x1, double y1, double x2, double y2) {
    double[][] s = new double[2][];
    s[0] = créer_point(x1, y1);
    s[1] = créer_point(x2, y2);
    return s;
}

// 2ème fonction constructeur : deux points extrémités
static double[][] créer_segment(double[] c1, double[] c2) {
    return créer_segment(c1[0], c1[1], c2[0], c2[1]);
}

// 3ème fonction constructeur : un segment existant (recopie/clonage)
static double[][] créer_segment(double[][] s) {
    return créer_segment(s[0], s[1]);
}

static void initialiser(double[][] s, double x1, double y1, double x2, double y2) {
    initialiser(s[0], x1, y1);
    initialiser(s[1], x2, y2);
}

static void afficher(double[][] s) {
    afficher(s[0]);
    Terminal.ecrireString("@");
    afficher(s[1]);
}
```

Un segment = un tableau (2/3)

```
static void déplacer(double[][] s, double dx, double dy) {
    déplacer(s[0], dx, dy);
    déplacer(s[1], dx, dy);
}

static void déplacer(double[][] s, double[] d) { déplacer(s, d[0], d[1]); }

static double longueur(double[][] s) {
    return Math.sqrt(Math.pow(s[0][0]-s[1][0], 2)+Math.pow(s[0][1]-s[1][1], 2)); }
}
```

Un segment = un tableau (3/3)

- Code du programme de test

```
Terminal.ecrireStringln("Petit jeu avec un segment");
Terminal.ecrireStringln("=====");
Terminal.ecrireString("segment S = ");
double[][] S = créer_segment(-1, 0, 3, 2);
afficher(S);
Terminal.ecrireStringln(" ; longueur = "+ longueur(S));
dx = -1; dy = 2;
Terminal.ecrireString("S est déplacé de (" + dx + ", " + dy + ") => segment S = ");
déplacer(S, dx, dy);
afficher(S);
Terminal.sautDeLigne();
```

- Trace d'exécution :

```
Petit jeu avec un segment
=====
segment S = ( -1.0 , 0.0 )@( 3.0 , 2.0 ) ; longueur = 4.47213595499958
S est déplacé de (-1.0,2.0) => segment S = ( -2.0 , 2.0 )@( 2.0 , 4.0 )
```

Un segment = un enregistrement (1/2)

```
// 1ère fonction constructeur : coordonnées des extrémités
static segment créer_segment(double x1, double y1, double x2, double y2) {
    segment s = new segment();
    s.e1 = créer_point(x1, y1);
    s.e2 = créer_point(x2, y2);
    return s;
}

// 2ème fonction constructeur : deux points extrémités
static segment créer_segment(point c1, point c2) {
    return créer_segment(c1.x, c1.y, c2.x, c2.y);
}

// 3ème fonction constructeur : un segment existant (recopie/clonage)
static segment créer_segment(segment s) {
    return créer_segment(s.e1, s.e2);
}

static void initialiser(segment s, double x1, double y1, double x2, double y2) {
    initialiser(s.e1, x1, y1);
    initialiser(s.e2, x2, y2);
}

static void afficher(segment s) {
    afficher(s.e1);
    Terminal.ecrireString("@");
    afficher(s.e2);
}
```

```
class segment {
    point e1, // 1ère extrémité
           e2; // 2ème extrémité
}
```

Un segment = un enregistrement (2/2)

- Suite des fonctions :

```
static void déplacer(segment s, double dx, double dy) {  
    déplacer(s.e1, dx, dy);  
    déplacer(s.e2, dx, dy);  
}
```

```
static void déplacer(segment s, point d) { déplacer(s, d.x, d.y); }
```

```
static double longueur(segment s) {  
    return Math.sqrt(Math.pow(s.e1.x-s.e2.x, 2)+Math.pow(s.e1.y-s.e2.y, 2)); }
```

- Programme de test :

```
Terminal.ecrireStringln("Petit jeu avec un segment");  
Terminal.ecrireStringln("=====");  
Terminal.ecrireString("segment S = ");  
segment S = créer_segment(-1, 0, 3, 2);  
afficher(S);  
Terminal.ecrireStringln(" ; longueur = "+ longueur(S));  
dx = -1; dy = 2;  
Terminal.ecrireString("S est déplacé de (" +dx+" , "+dy+" ) => segment S = ");  
déplacer(S, dx, dy);  
afficher(S);  
Terminal.sautDeLigne();
```

Composition des types – chemins d'accès aux variables

```
class A {  
    int    x;  
    double[] y;  
}
```

```
class B {  
    A[]    u;  
    boolean v;  
}
```

```
class C {  
    A    m;  
    B    n;  
}
```

```
class compositionTypes2 {  
    public static void main(String[] args) {  
        C o = new C();  
        o.m = new A();  
        o.n = new B();  
  
        o.m.x = 1;  
        o.m.y = new double[] { 7, 5 };  
        o.m.y[0] += o.m.y[1];  
  
        o.n.u = new A[] { o.m, new A() };  
        o.n.v = true;  
  
        o.n.u[0].x = 2;  
        o.n.u[1].y = new double[] { 9 };  
        o.n.u[1].y[0] += 1;  
    }  
}
```

VARIABLES STATIQUES ET NON STATIQUES

- Terminologie objet et Java
 - Attribut = champ = variable d'instance = variable non statique
 - Si m attribut d'une classe C , il y a autant de valeurs d'attributs $o.m$ qu'il y a d'objets o de type C
 - Variable de classe = variable statique
 - Si m variable de classe d'une classe C , il y a une seule valeur associée à cette variable : $C.m$
 - Attention : si o objet de type C , la notation $o.m$ est autorisée à la place de $C.m$
 - une variable de classe peut être vue comme une variable d'instance partagée : tous les objets l'ont en commun, et donc la modifier pour un objet la modifie pour tous les objets.
 - Dans le corps de toute méthode de la classe C , $C.m$ peut être abrégé en $m \Rightarrow$ attention aux ambiguïtés

VARIABLES STATIQUES ET NON STATIQUES

- La définition d'une variable (statique ou pas) peut inclure une valeur d'initialisation
 - à éviter pour les variables non statiques : leur initialisation est dévolue aux *constructeurs*
 - il existe une syntaxe particulière pour initialiser les variables statiques dont le code d'initialisation ne se limite pas une simple affectation
 - bloc d'initialisation **static**, exécuté avant que `main` ne soit exécutée.
 - il peut y en avoir plusieurs
 - ils sont exécutés dans l'ordre de leur écriture
 - aucune garantie sur l'ordre d'exécution si blocs d'initialisation dans des classes différentes

Exemple – variables statiques ou pas

```
public class statique {  
  
    static int vc1 = 1; // une variable de classe  
    int vi = 10;      // une variable d'instance  
  
    static double vc2; // une autre variable de classe  
    static { // bloc d'initialisation de variables de classe  
        // mise en place d'un code complexe d'initialisation ici  
        vc2 = 2;  
    }  
  
    public static void main(String[] args) {  
        System.out.println("vc1="+statique.vc1+" vc2="+statique.vc2);  
        statique.vc1++;  
        statique.vc2 = 3;  
        System.out.println("vc1="+vc1+" vc2="+vc2);  
  
        statique o = new statique();  
        o.vc1++;  
        System.out.println("o.vc1="+o.vc1+" o.vc2="+o.vc2+" o.vi="+o.vi);  
  
        statique p = new statique();  
        p.vc2++;  
        p.vi = 11;  
        System.out.println("vc1="+vc1+" vc2="+vc2+" p.vi="+p.vi);  
  
        System.out.println("vc1="+vc1+" vc2="+vc2);  
    }  
}
```

Exemples avancés

- Exercice 3 de l'examen NFA031 session 1 de 2012/13 (réels encadrés)
 - [sujet](#)
- Exercice 3 de l'examen NFA031 session 2 de 2012/13 (gestion de chambres d'hôtel)
 - [sujet](#)