Les classes (suite)

NFA032

Présentation n°3

Membres statiques ou pas

- Membre = variable ou méthode
- Un membre peut être :
  - Statique
    - Variable de classe
    - Fonction = méthode statique
  - Non statique (variable implicite **this**)
    - Variable d'instance
    - Méthode = méthode non statique

Exemple – définitions (1/4)

```
class A {
    int m1;  // m1 = variable non statique
        // Il y a une variable o.m1 pour chaque objet o de classe A
    static int m2;  // m2 = variable statique
        // Il n'y a qu'une seule variable A.m2
    void m3() {};  // m3 = méthode non statique
        // appel via o.m3() ; le paramètre implicite this, de type A,
        référence l'objet receveur o qui a servi à appeler la méthode
    static void m4() {} ;  // m4 = méthode statique
        // appel via A.m4()
    … }
```

Exemple – exploitation (2/4)

- **class** A {

  …

```
static void méthode_statique_utilisations_depuis_A() {
  A x = new A(),  // x est un objet de classe A
    y = new A();  // y est un autre objet de classe A

  x.m1 = 1;
  y.m1 = 2;

  A.m2 = 3;
  m2   = 4; // équivalent à A.m2 : on peut omettre 'A.' dans le code des méthodes de A

  x.m3();
  y.m3();

  A.m4();
  m4();   // équivalent à A.m4()
}
```

Exemple – exploitation (3/4)

- **class** A {

  …

```
void méthode_non_statique_utilisations_depuis_A() { // il y a un paramètre implicite : this
  A x = new A(),  // x est un objet de classe A
    y = new A();  // y est un autre objet de classe A

  x.m1 = 1;
  y.m1 = 2;
  m1   = 3; // équivalent à this.m1 : on peut omettre 'this.' dans le code des méthodes de A

  A.m2 = 3;
  m2   = 4; // équivalent à A.m2 : on peut omettre 'A.' dans le code des méthodes de A

  x.m3();
  y.m3();
  m3();   // équivalent à this.m3()

  A.m4();
  m4();   // équivalent à A.m4()
}
```

Exemple – exploitation (4/4)

```
class test {
  public static void main(String[] args) {
    A x = new A(),  // x est un objet de classe A
      y = new A();  // y est un autre objet de classe A

    // accès à une variable non statique (terminologie Java)
    // Vocabulaire en terminologie objet : variable d'instance, attribut, champ
    // Nécessite de passer par un objet pour y accéder : chaque objet de classe A poss
    //   => autrement dit, il y a autant de variables m1 qu'il y a d'objets de classe
    x.m1 = 1;
    y.m1 = 2;

    // accès à une variable statique (terminologie Java)
    // Vocabulaire en terminologie objet : variable de classe
    // Nécessite de spécifier un chemin d'accès (liste de classes) pour y accéder : il
    //      pour m2 de la classe A
    //  => autrement dit, il y a une seule variable m1 de la classe A dans l'environne
    A.m2 = 3;

    // appel d'une méthode non statique (terminologie Java)
    // Vocabulaire en terminologie objet : méthode, méthode d'instance
    // Nécessite de passer par un objet pour l'appeler : cet objet est appelé le recev
    //      est accessible via la variable prédéfinie 'this' dans le corps de la métho
    x.m3();
    y.m3();

    // appel d'une méthode statique (terminologie Java)
    // Vocabulaire en terminologie objet : méthode de classe
    // Nécessite de spécifier un chemin d'accès (liste de classes) pour l'appeler
    A.m4();
  }
}
```

Omission du préfixe X. devant un membre : pas toujours adapté

```
class B {
  static int y = 0; // variable de classe, dite "statique" en Java

  int x; // variable d'instance, dite "non statique" en Java

  B() { x = 0; y++; } // ici x est en fait this.x, et y est B.y

  void f(            ) { x++; y++; } // ici x est en fait this.x, et y est B.y
  void g(int x       ) { x++; y++; } // ici x est le paramètre de g, et y est B.y
  void h(int y       ) { x++; y++; } // ici x est en fait this.x, et y est le paramètre de h
  void i(int x, int y) { x++; y++; } // ici x et y sont les paramètres de i

  public static void main(String[] args) {
    B o = new B();
    o.f(); o.g(1); o.h(2); o.i(3, 4);
    new B(); // <= création d'un objet anonyme : aucune variable ne le référence
    // point 1 ; variables ayant un x ou y dans leur description : o.x B.y
    //           il y a aussi le .x de l'objet anonyme, mais qui ne peut être désigné
    Terminal.ecrireStringln("o.x = "+o.x+", B.y = "+B.y);
  }
}
```

Un membre peut aussi être une classe…

- Ce n'est pas au programme, mais jetons un œil :
  - Fichier classes.java :

```
class classes {
  class classe_C {}
  static class classe_D {}

  static void méthode_statique() {
    class classe_E {};

    classe_A a = new classe_A();

    classe_B b = new classe_B();

    classes xx = new classes();
    classe_C c = xx.new classe_C(); // il faut nécessairement passer par

    classe_D d = new classe_D();

    classe_E e = new classe_E();
    //classe_F f = new classe_F(); // classe non accessible depuis cette
  }

  void méthode_non_statique() {
    class classe_F {};

    classe_A a = new classe_A();

    classe_B b = new classe_B();

    classe_C c1 = new classe_C(); // version simplifiée de : this.new c
    classes  xx = new classes();
    classe_C c2 = xx.new classe_C(); // l'instance c2 est associée à xx

    classe_D d = new classe_D();

    //classe_E e = new classe_E(); // classe non accessible depuis cette
    classe_F f = new classe_F();
  }
}

class classe_B {}
```

  - Fichier classe_A.java :

```
class classe_A {
}
```

Classe membre : exploitation extérieure

```
public class classe_utilisatrice {
  public static void main(String[] args) {
    classe_A a = new classe_A();

    classe_B b = new classe_B();

    classes.classe_C c = new classes().new classe_C();

    classes.classe_D d = new classes.classe_D();

    //classe_E e = ... // classe non accessible depuis celle-ci
    //classe_F f = ... // classe non accessible depuis celle-ci
  }
}
```

Constructeurs et destructeurs

- **Constructeur** = méthode appelée à la création d'un objet (opérateur **new**)
- **Destructeur** = méthode appelée à la destruction d'un objet (retrait de la mémoire)
- En Java
  - Un destructeur n'est jamais appelé par le programmeur
  - Le programmeur peut toutefois définir le corps du destructeur :
    - méthode `protected void finalize();`
  - C'est le ramasse-miettes (*gc* = *garbage collector* = récupérateur mémoire) qui s'en charge
  - Le ramasse-miettes est un processus qui fonctionne en parallèle du programme principal

Les objets composés (composites)

- Relation abstraite = relation tout / partie
- Un objet composé a des attributs dont les valeurs sont des références vers d'autres objets
  - Objet composé = tout
  - Objet composant = partie

  Exemple : une voiture est composée de quatre roues
- Une partie ne peut pas exister sans son tout
- Supprimer le tout, c'est supprimer ses parties

Exemple simplifié (1/8)

```
class B {
   int m;
   // 1er constructeur
   B(int M) {
      this.m = M;
      Terminal.ecrireStringln("appel B("+M+")");
   }
   // destructeur
   @Override protected void finalize() throws Throwable {
      super.finalize();
      Terminal.ecrireStringln("finalize B("+this.m+")");
   }
}
```

Exemple simplifié (2/8)

```
class A {
  int x;
  B   b;
  // 1er constructeur
  A(int X) {
    this.x = X;
    this.b = null;
    Terminal.ecrireStringln("appel A("+X+")");
  }
  // 2ème constructeur
  A(int X, int M) {
    this(X);
    this.b = new B(M);
    Terminal.ecrireStringln("appel A("+X+","+M+")");
  }
  // 3ème constructeur
  A(int X, B oB) {
    this.x = X;
    this.b = oB;
    Terminal.ecrireStringln("appel A("+X+","+(oB==null ? "null" : "B("+oB.m+")")+")");
  }
  // destructeur
  @Override protected void finalize() throws Throwable {
    super.finalize();
    Terminal.ecrireStringln("finalize A("+this.x+","+(this.b==null ? "null" : "B("+this.b.m+")'
  }
}
```

Exemple simplifié (3/8)

- Question : qu'est-il affiché ?

```
class AB {
  public static void main(String[] args) {
    new A(1);
    Terminal.ecrireStringln("-------");
    B o = new B(2);
    Terminal.ecrireStringln("-------");
    new A(3, 4);
    Terminal.ecrireStringln("-------");
    new A(5, o);
    Terminal.ecrireStringln("-------");
    { // décommenter ce sous-bloc une fois l'e
    }
  }
}
```

Exemple simplifié (4/8)

- Réponse :

```
appel A(1)
-------
appel B(2)
-------
appel A(3)
appel B(4)
appel A(3,4)
-------
appel A(5,B(2))
```

Exemple simplifié (5/8)

- Appel explicite du ramasse-miettes
  - Question : qu'est-il affiché ?

```
class AB {
  public static void main(String[] args) {
    new A(1);
    Terminal.ecrireStringln("-------");
    B o = new B(2);
    Terminal.ecrireStringln("-------");
    new A(3, 4);
    Terminal.ecrireStringln("-------");
    new A(5, o);
    Terminal.ecrireStringln("-------");
    { // décommenter ce sous-bloc une fois l'exercice tr
      o = null;
      System.gc(); // appel explicite au récupérateur mé
      System.runFinalization(); // appel explicite à fir
    }
  }
}
```

Exemple simplifié (6/8)

- Réponse :

```
appel A(1)
-------
appel B(2)
-------
appel A(3)
appel B(4)
appel A(3,4)
-------
appel A(5,B(2))
-------
finalize A(1,null)
finalize A(5,B(2))
finalize B(4)
finalize A(3,B(4))
finalize B(2)
```

Exemple simplifié (7/8)

- On se contente de déplacer une affectation :

```
class AB {
  public static void main(String[] args) {
    new A(1);
    Terminal.ecrireStringln("-------");
    B o = new B(2);
    Terminal.ecrireStringln("-------");
    new A(3, 4);
    Terminal.ecrireStringln("-------");
    new A(5, o);
    Terminal.ecrireStringln("-------");
    { // décommenter ce sous-bloc une fois l'
      System.gc(); // appel explicite au récu
      System.runFinalization(); // appel expl
      o = null;
    }
  }
}
```

Exemple simplifié (8/8)

- Réponse :

```
appel A(1)
-------
appel B(2)
-------
appel A(3)
appel B(4)
appel A(3,4)
-------
appel A(5,B(2))
-------
finalize A(1,null)
finalize A(5,B(2))
finalize B(4)
finalize A(3,B(4))
finalize B(2)
```

Exemple : voiture et roues

- Une **voiture** est caractérisée par :
  - Son poids sans les roues (kg)
  - Ses quatre roues
- Une **roue** est caractérisée par :
  - Son diamètre (cm)
  - Son poids (kg)
- On veut pouvoir :
  - Définir une voiture dont :
    - toutes les roues sont identiques
    - les roues peuvent être différentes
  - Afficher les informations d'une voiture :
    - Poids total
    - Informations sur ses roues

Exemple : programme de test

```
class test_voitures {
  public static void main(String[] args) {
    {
      // exploitation du 3ème constructeur des voitures
      uneVoiture v = new uneVoiture(
        500,                                  // poids propre
        new double[] {50, 49, 51, 50.5}, // diamètres des roues
        new double [] {5, 10, 15, 20});  // poids des roues
      v.afficher("1ère voiture de l'exemple (roues toutes différentes) :
    }
    {
      // exploitation du 1er constructeur des voitures
      uneVoiture v = new uneVoiture(500, 50, 10);
      v.afficher("2ème voiture de l'exemple (roues toutes identiques) :"
    }
  }
}
```

Exemple : trace d'exécution

```
1ère voiture de l'exemple (roues toutes différentes) :
  poids sans les roues = 500.0
  roue n°1 : diamètre = 50.0, poids = 5.0
  roue n°2 : diamètre = 49.0, poids = 10.0
  roue n°3 : diamètre = 51.0, poids = 15.0
  roue n°4 : diamètre = 50.5, poids = 20.0
  poids total = 550.0
2ème voiture de l'exemple (roues toutes identiques) :
  poids sans les roues = 500.0
  roue n°1 : diamètre = 50.0, poids = 10.0
  roue n°2 : diamètre = 50.0, poids = 10.0
  roue n°3 : diamètre = 50.0, poids = 10.0
  roue n°4 : diamètre = 50.0, poids = 10.0
  poids total = 540.0
```

Solution : les roues

```
class uneRoue {
  // variables d'instances (attributs)
  double sonDiamètre,
         sonPoids;

  // constructeur
  uneRoue(double diamètre, double poids) {
    this.sonDiamètre = diamètre ;
    this.sonPoids = poids ;
  }

  // méthode affichant les caractéristiques d'une roue
  void afficher(String entête) {
    if (entête != null) Terminal.ecrireString(entête);
    Terminal.ecrireString("diamètre = "+this.sonDiamètre);
    Terminal.ecrireStringln(", poids = "+this.sonPoids);
  }
}
```

Solution : les voitures (1/4)

```
class uneVoiture {
  // variables d'instances (attributs)
  double    sonPoidsSansRoues ;
  uneRoue[] sesRoues;
    // on aurait pu faire 4 initialisation de uneRoue r1,r2,r3,r4 ;

  // 1er constructeur : toutes les roues ont les mêmes caractéristiques
  uneVoiture(double poidsSansRoues, double diamètreRoue, double poidsRoue) {
    this.sonPoidsSansRoues = poidsSansRoues;
    this.sesRoues = new uneRoue[4];    // création du tableau des roues (valait null jusq
    for (int i=0 ; i<4 ;i++) // création des 4 roues
      this.sesRoues[i] = new uneRoue(diamètreRoue, poidsRoue);
  }
```

Solution : les voitures (2/4)

```
// 2ème constructeur : le tableau des roues est passé explicitement
uneVoiture(double poidsSansRoues, uneRoue[] roues) {
  this.sonPoidsSansRoues = poidsSansRoues;
  this.sesRoues = roues ; // solution la plus simple : reprise du tableau tel quel
  /* noter qu'aucun contrôle n'est fait sur le paramètre 'roues', ce qui est risqué :
     - roues peut valoir 'null'
     - roues peut être un tableau de longueur quelconque (pas forcément 4)
     - des composantes du tableau peuvent être nulles

     Autrement dit, pour être sûr que 'roues' est bien un tableau de 4 roues,
     il faut que, à la fin des tests qui suivent, OK ait pour valeur 'true'

     boolean OK = roues != null && roues.length == 4;
     for (int i=0; OK && i < 4; i++) if (roues[i] == null) OK = false;
     // si ici OK vaut true, on a bien un tableau de 4 roues
   */
   /* on pourrait créer de nouvelles roues pour cette voiture, dont les
      caractéristiques seraient identiques à celles passées en paramètre :

     this.sesRoues = new uneRoue[4];
     for (int i=0 ; i<4 ;i++)
       this.sesRoues[i] = new uneRoue(roues[i].sonDiamètre, roues[i].sonPoids);
    */
}
```

Solution : les voitures (3/4)

```
// 3ème constructeur : deux tableaux précisent les diamètres et les poids de chaque roue
uneVoiture(double poidsSansRoues, double[] diamètres, double[] poids) {
  this.sonPoidsSansRoues = poidsSansRoues;
  this.sesRoues = new uneRoue[4];
  /* aucune vérification faite sur 'diamètres' et 'poids' : en principe, ils
     devraient être des tableaux d'exactement 4 composantes */
  for (int i=0; i<4; i++)
    this.sesRoues[i] = new uneRoue(diamètres[i], poids[i]);
}
```

Solution : les voitures (4/4)

```
// méthode calculant le poids total de la voiture avec les roues
double poidsTotal() {
  double Total = this.sonPoidsSansRoues ;
  for (int i=0 ; i<4 ; i++)
    Total += this.sesRoues[i].sonPoids ;
  return Total ;
}

// méthode affichant les caractéristiques d'une voiture
void afficher(String entête) {
  String décalage = "";
  if (entête != null) {
    Terminal.ecrireStringln(entête);
    décalage = "  ";
  };
  Terminal.ecrireStringln(décalage+"poids sans les roues = "+this.sonPoidsSansRoues);
  for (int i = 0; i < 4; i++)
    this.sesRoues[i].afficher(décalage+"roue n°"+(i+1)+" : ");
  Terminal.ecrireStringln(décalage+"poids total = "+this.poidsTotal());
}
```