Java

Licence Professionnelle CISII, 2009-2010

Cours 2 : Classes et Objets

Java

Classes et Objets

- Objectifs des LOO :
  - Manipuler des objets
  - Découper les programmes suivant les types des objets manipulés
  - Regrouper les données avec les traitements qui les utilisent
  - Exemple
    - Une classe **Compte_bancaire** regroupe tout ce que l'on peut faire avec un compte bancaire, avec toutes les données nécessaires à ses traitements

Java

Classes et Objets

- Qu'est ce qu'un objet ?
  - Toute entité identifiable, concrète ou abstraite, peut être considérée comme un objet
  - Un objet réagit à certains messages qu'on lui envoie de l'extérieur
    - la façon dont il réagit détermine le comportement de l'objet
  - Il ne réagit pas toujours de la même façon à un même événement
    - sa réaction dépend de l'état dans lequel il se trouve

Java

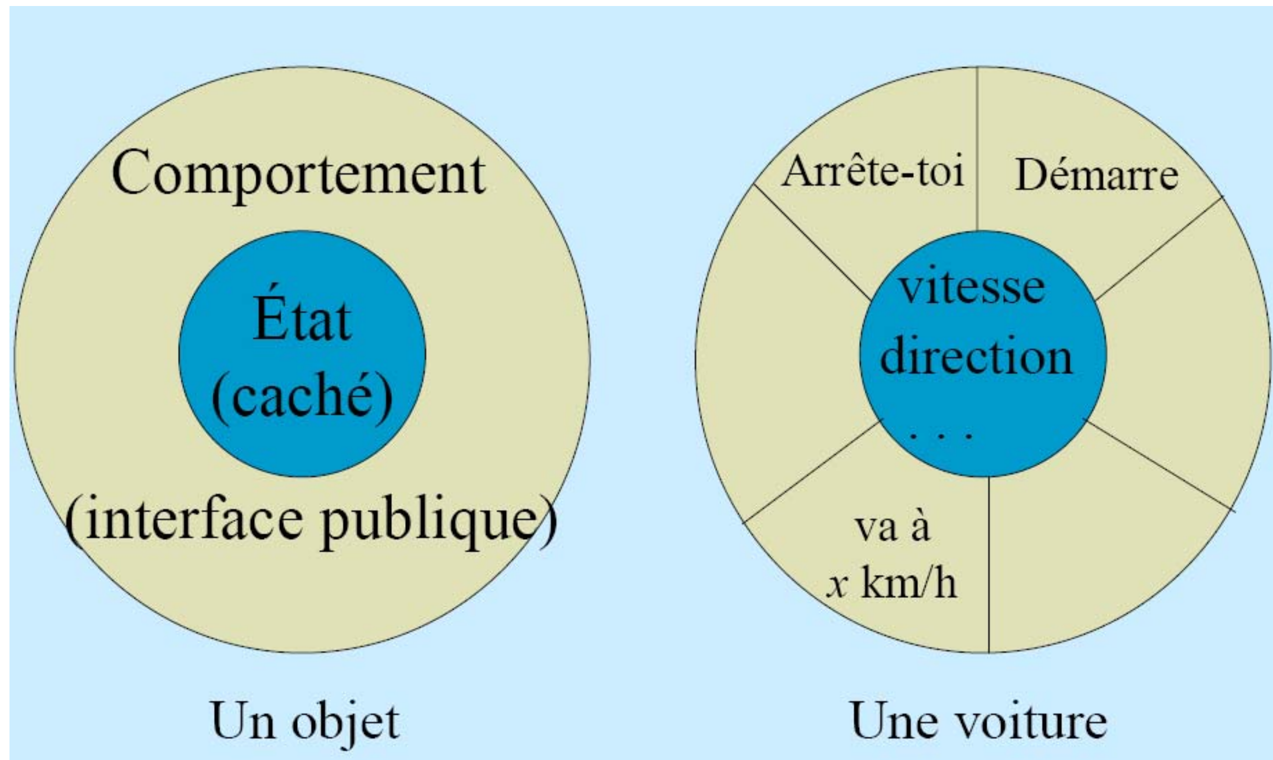
Classes et Objets

- Notion d'objet en Java
  - Un objet a :
    - une adresse en mémoire (identifie l'objet)
    - un comportement (ou interface)
    - un état interne
  - Le comportement est donné par des fonctions ou procédures, appelées méthodes
  - L'état interne est donné par des valeurs de variables d'instances

Java

Classes et Objets

- Exemple d'objet : voiture



Java

Classes et Objets

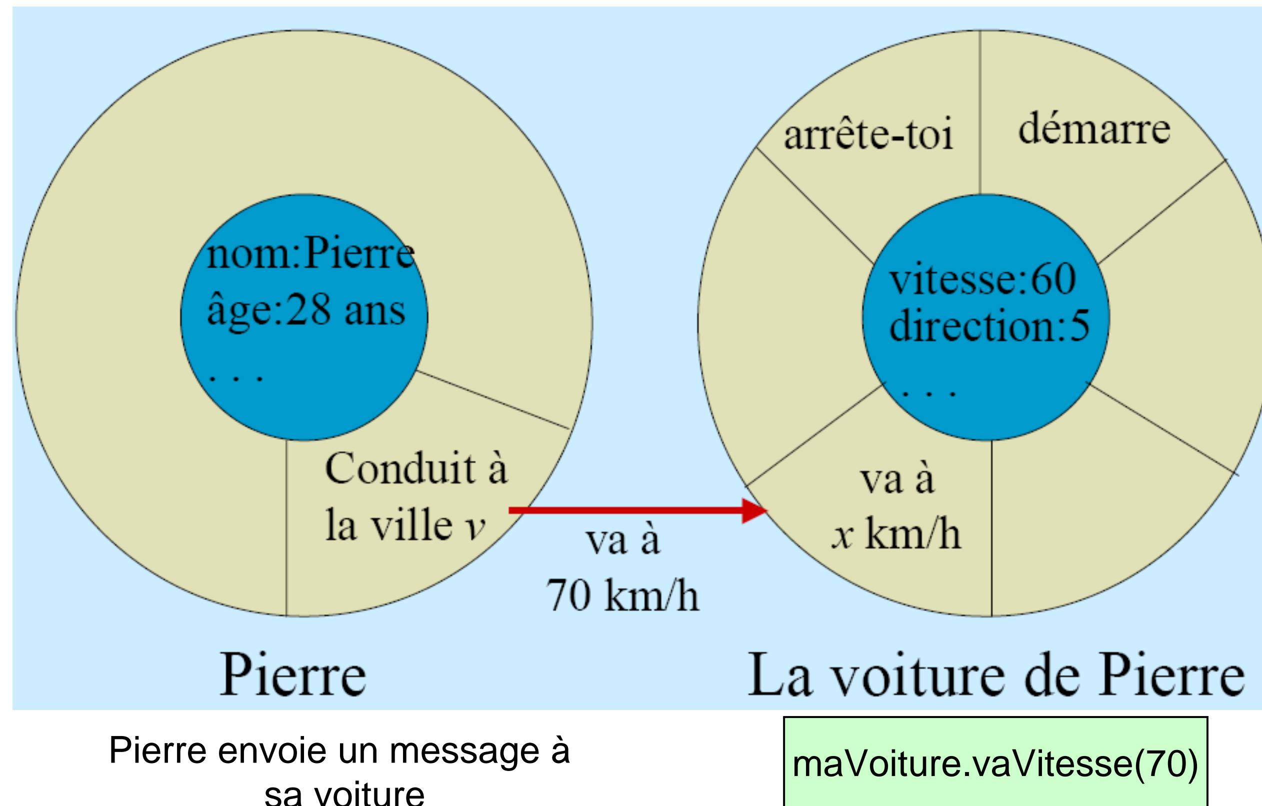
- Interactions entre objets
  - Les objets interagissent en s'envoyant des messages synchrones
  - Les méthodes d'un objet correspondent aux messages qu'on peut lui envoyer :
    - quand un objet reçoit un message, il exécute la méthode correspondante
  - Exemples :

```
objet1.decrisToi();
employe.reçoisSalaire(20000);
voiture.demarre();
voiture.vaAVitesse(50);
```

Java

Classes et Objets

- Messages entre objets



Pierre envoie un message à sa voiture

```
maVoiture.vaVitesse(70)
```

Java

Classes et Objets

- Messages entre objets
  - Exemple : la Voiture demande au Moteur de démarrer
  - Pour cela, la Voiture doit pouvoir accéder au Moteur !

```
class Voiture {
  private Moteur moteur;
  public void demarre() {
        moteur.demarre();
  }
}
```

Java

Classes et Objets

- Paradigme objet
  - La programmation objet est un paradigme, une manière de « modéliser le monde » :
    - des objets ayant un état interne et un comportement
    - collaborent en s'échangeant des messages (pour fournir les fonctionnalités que l'on demande à l'application)
  - D'autres paradigmes :
    - programmation impérative (Pascal, C)
    - programmation fonctionnelle (Scheme, Lisp)

Java

Classes et Objets

- Classe Java
  - Les objets qui collaborent dans une application sont souvent très nombreux
  - Mais on peut le plus souvent dégager des types d'objets :
    - des objets ont une structure et un comportement très proches, sinon identiques
  - Par exemple
    - tous les livres dans une application de gestion d'une bibliothèque
  - La notion de classe correspond à cette notion de types d'objets
    - Un objet correspond à une instanciation de classes

Java

Classes et Objets

- Classes
  - Les classes permettent de définir les nouveaux types
  - Un objet est une instance d'une classe
  - Une classe doit définir
    - les données ou attributs associés aux objets de la classe
    - les opérations ou méthodes qui peuvent être effectuées par les objets
  - Exemple :
    - Classe : Etudiant
    - Attributs : String nom, int numéro
    - Méthodes : créer, changer le nom, changer le numéro
    - Objet de la classe Etudiant : Bill Smith 4679

Java

Objets et classes

- Déclaration des classes : Exemple : Etudiant.java, Nom.java

```
public class Etudiant {
    private Nom nom;
    private int numero;

    public Etudiant(String p, String f, int n)
    {
      nom = new Nom(p,f);
      numero = n;
    }

    public void changerNomFamille (String
    s) {
      nom.changerNomFamille(s);
    }

    public void changerPrenom (String
    s) {
      nom.changerPrenom(s);
    }

    public void changerNumero (int n) {
      numero = n;
    }

    public String toString() {
      return (nom.toString() +
      "\nNumero : " + numero);
    }
}
```

```java
public class Nom{
    private String prenom;
    private String nomFamille;

    public Nom (String p, String n) {
      prenom = p;
      nomFamille = n;
    }

    public void changerPrenom (String p){
      prenom = p;
    }

    public void changerNomFamille (String n) {
      nomFamille = n;
    }

    public String toString () {
      return (prenom + " " + nomFamille);
    }
}
```

Java

Objets et classes

- Utilisation d'une classe : Exemple : TestEtudiant.java

```
public class TestEtudiant{
    public static void main(String args []) {
            Etudiant e; //e est une référence à la classe Etudiant
            e = new Etudiant ("Bil", "Jines", 0); //e est une instance de
                                                  // Etudiant
            e.changerNomFamille("Jones");
            e.changerPrenom("Bill");
            e.changerNumero(4679);
            System.out.println(e);
    }
}
```

Sortie :
Nom : Bill Jones
Numero : 4679

Java

Objets et classes

- Membres d'une classe
  - Les variables et les méthodes s'appellent les membres de la classe
  - Les constructeurs (il peut y en avoir plusieurs)
    - servent à créer des objets appelés instances de la classe
    - Quand une instance est créée, son état est conservé dans les variables d'instance
  - Les méthodes déterminent le comportement des instances de la classe quand elles reçoivent un message

Java

Objets et classes

- Rôles d'une classe
  - Une classe est
    - un type qui décrit une structure (variables d'état) et un comportement (méthodes)
    - un module pour décomposer une application en entités plus petites
    - un générateur d'objets (par ses constructeurs)
  - Une classe permet d'encapsuler les objets :
    - les membres **public** sont vus de l'extérieur mais les membres **private** sont cachés

Java

Objets et classes

- Conventions d'utilisation
  - Les noms de classes commencent par une majuscule (ce sont les seuls avec les constantes) :
    - Cercle, Object
  - Les mots contenus dans un identificateur commencent par une majuscule :
    - uneClasse, uneMethode, uneAutreVariable
  - Les constantes sont en majuscules et les mots sont séparés par le caractère souligné « _ » :
    - UNE_CONSTANTE
  - Si possible, des noms pour les classes et des verbes pour les méthodes

Java

Objets et classes

- La référence
  - Soit la classe :

```
public class Point {
    private int x = 0;
    private int y = 0;
}
```

  - Création d'une instance (i.e. un objet)
    - Pour utiliser une classe il faut définir une instance de cette classe en utilisant new
      - p = new Point();
  - Création d'une référence
    - Pour accéder à un objet, on utilise une référence à cet objet
      - Point p;
    - p fait maintenant référence à un objet qui contient un champ x et un champ y, tous les deux initialisés à 0

Java

Objets et classes

- Référence (suite)
  - On peut déclarer une référence et définir une instance en une seule ligne :
    - Point p = new Point();
    - Integer i = new Integer(5);

Java

Objets et classes

- Durée de vie
  - types primitifs :

{	aucun
int x = 12;	x
{	x
int q = 96;	x, q
}	x
}	aucun

  - objets :

{	aucun
String s;	s
s = new String("abc");	s, "abc"
}	"abc"

  - Les objets qui ne sont plus référencés sont détruits automatiquement par le "récupérateur de mémoire" qui fonctionne en arrière-plan

Java

Objets et classes

- Les opérateurs et les objets
  - Affectation : références vs attributs

```
public class Nombre {
    public int i; //attribut
}

Nombre n1 = new Nombre ();
Nombre n2 = new Nombre ();
n1.i = 9; // affecter une valeur à un attribut
n2.i = 47; // affecter une valeur à un attribut
n1 = n2; //affecter une référence à une référence
n1.i = 27;
System.out.println(n1.i); // 27
System.out.println(n2.i); // 27
```

Java

Objets et classes

- Autre exemple de références vs attributs

```
n1 = new Nombre ();
n2 = new Nombre ();
n1.i = 47;
n2.i = 47;
System.out.println(n1==n2); // false
System.out.println(n1.i == n2.i); // true
```

Java

Objets et classes

- La méthode equals()
  - sert à comparer les références
    - n1 = new Nombre ();
    - n2 = new Nombre ();
    - System.out.println(n1.equals(n2)); // false

Java

Objets et classes

- La méthode compareTo()
  - Sert à comparer deux objets
  - Si s1 et s2 sont deux String, s1.compareTo(s2) renvoie
    - 0 si s1 et s2 sont égaux
    - un nb<0 si s1<s2 selon l'ordre lexicographique (ordre du dico)
    - un nb>0 sinon

```
String s1 = new String("aardvark");
String s2 = new String("apple");
System.out.println(s1.compareTo(s2)); // -15
```

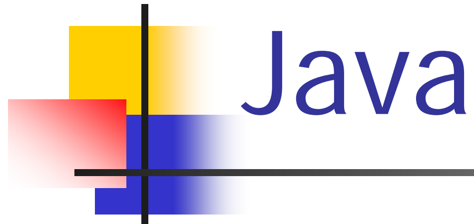
■ Exemple : methEquals.java

```
class MethodeChaine {
	public static void main(String[] arg) {
		String r = "essai";
		String s = "es" + "sai";
		String t = "essais";
		String u = "ESSAI".toLowerCase();
		System.out.println("\"" + r + "\" (r) identique avec \"" + t + "\" (t) : " + (r == t));
		System.out.println("\"" + r + "\" (r) identique avec \"" + s + "\" (s) : " + (r == s));
		System.out.println("\"" + r + "\" (r) identique avec \"" + u + "\" (u) : " + (r == u));
		System.out.println("\"" + r + "\" (r) identique avec \"" + u + "\" (u) : " + r.equals(u));
		System.out.println("\"" + r + "\".compareTo(\"" + u + "\") vaut : " + r.compareTo(u));
		System.out.println("\"" + r + "\".compareTo(\"" + t + "\") vaut : " + r.compareTo(t));
		System.out.println("\"" + t + "\".compareTo(\"" + r + "\") vaut : " + t.compareTo(r));
		}
}
```

```
"essai" (r) identique avec "essais" (t) : false
"essai" (r) identique avec "essai" (s) : true
"essai" (r) identique avec "essai" (u) : false
"essai" (r) identique avec "essai" (u) : true
"essai".compareTo("essai") vaut : 0
"essai".compareTo("essais") vaut : -1
"essais".compareTo("essai") vaut : 1
```

Java

- Classes et instances
  - Une instance d'une classe est créée par un des constructeurs de la classe
  - Une fois qu'elle est créée, l'instance
    - a son propre état interne (les valeurs des variables)
    - partage le code qui détermine son comportement (les méthodes) avec les autres instances de la classe



Java

- Constructeur
  - Chaque classe a un ou plusieurs constructeurs qui servent à
    - créer les instances
    - initialiser l'état de ces instances
  - Un constructeur
    - a le même nom que la classe
    - n'a pas de type retour

Java
Constructeurs

- Exemple

```
public class Employe {
      private String nom, prenom;
      private double salaire;
      // Constructeur
      public Employe(String n, String p) {
      nom = n;
      prenom = p;
      }
      public void setSalaire(double s){
      salaire=s;
      }
. . .
      public static void main(String[] args) {
      Employe e1;
      e1 = new Employe("Dupond", "Pierre");
      e1.setSalaire(12000);
      . . .
      }
}
```

variables d'instance

création d'une instance de Employe

Java

Constructeurs

- Plusieurs constructeurs (surcharge)

```
public class Employe {
private String nom, prenom;
private double salaire;
// Constructeur 1
    public Employe(String n, String p) {
    nom = n;
    prenom = p;
    }
// Constructeur 2
    public Employe(String n, String p, double s) {
    nom = n;
    prenom = p;
    salaire = s;
    }
. . .
e1 = new Employe("Dupond", "Pierre");
e2 = new Employe("Durand", "Jacques", 15000);
```

Java

Constructeurs

- Autre Exemple : TestPoint.java

```
public class Point {
    private int x;
    private int y;
    // Constructeur 1
    public Point() {
        x = 0;
        y = 0;
    }
    // Constructeur 2
    public Point(int coord1, int coord2) {
        x = coord1;
        y = coord2;
    }
}
```

```
public class TestPoint {
    public static void main (String []
    args) {
        Point p,q;
        p = new Point();
        q = new Point(2,8);
    }
}
```

Java

Constructeurs

- Désigner un constructeur par : this
  - Chaque objet a accès à une référence à lui-même
    - la référence this
  - Exemple :
    - si le constructeur a comme argument un nom de variable identique à une variable de la classe et qu'on souhaite l'initialiser avec la variable passée en paramètre, alors on écrit :

```
constructeur(int maVar){
   this.maVar=maVar
}
```

Java

Constructeurs

- this
  - Exemple :

```
public class Point {
    private int x;
    private int y;
    public Point() {
        x = 0;
        y = 0;
    }
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

Java

Constructeurs

- this : autre exemple : Voiture.java

```
public class Voiture
{
   private int roues;
   private String moteur;
   private boolean carrosserie;

   public Voiture (int roues, String moteur,
     boolean carrosserie)
   {
      this.roues = roues;
      this.moteur = moteur;
      this.carrosserie = carrosserie;
   }
```

```
public void avancer ()
{
   String moteur ="Renault";
    System.out.println("J'avance
   avec une porshe " +
   this.moteur);
}
```

Java

Constructeurs

- this
  - Autre exemple (suite) : testVoiture.java

```
public class Test
{
  public static void main (String [] args)
  {
    Voiture porsche = new Voiture(4,"V8",true);
    porsche.avancer();
  }
}
```

  - Résultat
    - System.*out*.println("J'avance avec une porshe " + this.moteur); donne "V8"
    - System.*out*.println("J'avance avec une porshe " + moteur); donne "Renault"

Java

Constructeurs

- Un constructeur peut appeler d'autres constructeurs

```
public class Arbre{
	private int hauteur = 0;
	private String espece = new String("null");

	public Arbre (int i) {
			hauteur = i;
	}
	public Arbre (String s) {
			espece = s;
	}
	public Arbre(String s, int i) {
			this(i); // première instruction
			espece = s;
	}
	public Arbre () {
			this("null", 0);
	}
}
```

Cours 2-TD 2

- Exercice 1
  - Créer une classe Point et une classe Rectangle
  - Les rectangles sont définis par le point inférieur gauche et supérieur droit
  - Proposer une méthode qui calcule la surface d'un rectangle
  - Tester l'utilisation

Java

Constructeurs

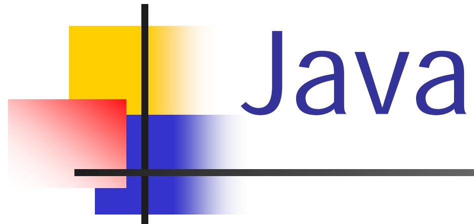
- Constructeur par défaut
  - Un constructeur sans paramètre est appelé constructeur par défaut explicite
  - Conseil :
    - Créer toujours un constructeur même s'il est sans paramètre : programmation propre

- Exemple de constructeur absent : Phrase.java et ConstructVide.java

```
class Phrase
{
  String phrase="";
}
```

```
public class ConstructVide {
  public static void main (String
    args[]) {
    Phrase p = new Phrase();
    System.out.println("phrase : "+
    p.phrase);
    p.phrase = "Du haut de ces
    pyramides, 40 siècles vous
    comtemplent";
    System.out.println("phrase : "+
    p.phrase);
  }
}
```

Résultat :
phrase :
phrase : Du haut de ces pyramides, 40 siècles vous
comtemplent



Java

- Méthodes
  - Deux types de méthodes servent à donner accès aux variables depuis l'extérieur de la classe :
    - les accesseurs en lecture pour lire les valeurs des variables
      - "accesseur en lecture" est souvent abrégé en "accesseur"
    - les modificateurs ou accesseurs en écriture, pour modifier leur valeur

Java

Méthodes

- Exemple

```
public class Employe {
private double salaire;
. . .
    public void setSalaire(double unSalaire) {
    if (unSalaire >= 0.0)
    salaire = unSalaire;
    }
    public double getSalaire() {
    return salaire;
    }
...
}
```

Modificateur

Accesseur

Java

Méthodes

- Sont définies par
  - le nom
  - les paramètres avec leur types (optionnel)
  - le type de retour
  - un qualificateur (public, protected, private)
  - et le corps de la méthode (instructions)
- Utilisation
  - référence.méthode(arguments)

```
p.setX(68);
```

Java

Méthodes

- Surcharge d'une méthode
  - En Java, on peut surcharger une méthode, c'est-à-dire, ajouter une méthode qui a le même nom mais pas la même signature qu'une autre méthode :
    - calculerSalaire(int) ← indice dans la grille des salaires
    - calculerSalaire(int, double) ← prime accordée aux commerciaux

Java

Méthodes

- Surcharge d'une méthode (suite)
  - En Java, il est interdit de surcharger une méthode en changeant le type de retour
  - Autrement dit, on ne peut différencier deux méthodes par leur type de retour
  - Par exemple
    - il est interdit d'avoir ces deux méthodes dans une classe
      - int calculerSalaire(int)
      - double calculerSalaire(int)

Java

Variables : 3 sortes

1. Locales
   - Déclarées dans une méthode, un constructeur ou un bloc
   - N'existent que localement, ne sont pas visibles de l'extérieur
   - Sorties de leur espace, elles n'existent plus, la mémoire correspondante est libérée
2. Variables de classe ou variable static
   - Déclarées avec le mot clé static dans une classe, mais en dehors d'une méthode
   - Définies pour l'ensemble du programme et sont visibles depuis toutes les méthodes

Java

Variables : 3 sortes

3. d'instance
   - Variables qui sont utilisées pour caractériser l'instance d'un objet créée par un constructeur
   - Déclarées dans une classe, mais en dehors des méthodes
   - Appelées aussi membres ou variables de champ
   - Créées quand un objet est créé et détruites quand l'objet est détruit

Exemple

```
import java.io.*;

class Employee{
  // cette variable d'instance est visible
    pour tout fils (héritier) de la classe
  public String name;

  // salary est visible dans la classe
    Employee seulement
  private double salary;

  // la variable name est initialisée
    dans le constructeur
  public Employee (String
    empName){
    name = empName;
  }

  // la variable salary est affectée
    d'une valeur
  public void setSalary(double
    empSal){
    salary = empSal;
  }

  // cette méthode imprime les détails
    concernant employee
  public void printEmp(){
    System.out.println("name  : " + name );
    System.out.println("salary :" + salary);
  }

  public static void main(String args[]){
    Employee empOne = new
    Employee("Ransika");
    empOne.setSalary(1000);
    empOne.printEmp();
  }
}
```

Java

Variables

- Variable locale ou variable d'instance ?
  - Il arrive d'hésiter entre référencer un objet
    - par une variable locale d'une méthode
    - ou par une variable d'instance de la classe
  - Si l'objet est utilisé par plusieurs méthodes de la classe
    - l'objet devra être référencé par une variable d'instance

Java

Variables

- Déclaration des variables
  - Toute variable doit être déclarée avant d'être utilisée
  - Déclaration d'une variable
    - On indique au compilateur que le programme va utiliser une variable de ce nom et de ce type
  - Exemple
    - double prime;
    - Employe e1;
    - Point centre;

Java

Variables

- Déclaration des variables : syntaxe

  modificateur type identificateur [= valeur];

  - Le nom d'une variable
    - est appelé un identificateur obéissant à certaines règles syntaxiques
  - Le type de la variable
    - correspond au type de données qu'elle devra contenir, en l'occurrence soit un type primitif (int, float, char, boolean, etc.), soit un type composite (String, File, Vector, Socket, Array, etc.)
  - La valeur de la variable
    - est appelée un littéral, et doit être conforme à son type de données

Java

Variables

- Le modificateur
  - Précise l'accès à la variable

  = private, protected, public ou rien (par défaut)
    - représente son type d'accessibilité au sein d'un programme
  - D'autres modificateurs peuvent être ajoutés, il s'agît de :
    - ***final*** pour définir une constante,
    - ***static*** rendant la variable accessible par la classe entière et même sans l'instanciation de cette classe
    - ***transient*** interdisant la sérialisation de la variable
    - ***volatile*** empêchant la modification asynchrone de la variable dans un environnement multi-threads

Java

Variables

- Le modificateur (suite)
  - Les modificateurs peuvent apparaître dans n'importe quel ordre
    - public int hauteur = 100;
    - static String unite = "mètres";
    - long population = 6000000000;
    - boolean reussite = false;
    - final char CR = '\r';
    - float = 10.87E5;
    - URL adresse = new URL("http://java.sun.com/");
    - private String[] tableau;
    - transient private long code_secret;
- Il est interdit de définir deux fois la même variable au sein d'une même portée

Java

Variables

- Affectation
  - L'affectation d'une valeur à une variable est effectuée par l'instruction

    variable = expression;
  - L'expression est calculée et ensuite la valeur calculée est affectée à la variable
  - Exemple :

```
x = 3;
x = x + 1;
```

Java

Variables

- Initialisation
  - Une variable doit être initialisée
    - i.e. recevoir une valeur avant d'être utilisée dans une expression
  - Si elles ne sont pas initialisées par le programmeur, les variables d'instance (et les variables de classe étudiées plus loin) reçoivent les valeurs par défaut de leur type
    - 0 pour les types numériques, par exemple
  - L'utilisation d'une variable locale non initialisée par le programmeur provoque une erreur
    - pas d'initialisation par défaut

Java

Variables

- Initialisation (suite)
  - On peut initialiser une variable en la déclarant
  - La formule d'initialisation peut être une expression complexe :
    - double prime = 2000.0;
    - Employe e1 = new Employe("Dupond", "Jean");
    - double salaire = prime + 5000.0;

Java

Variables

- Déclaration / création

```
public static void main(String[] args) {
Employe e1;
e1.setSalaire(12000);
```

- Il ne faut pas confondre
  - déclaration d'une variable et création d'un objet référencé par cette variable
    - « Employe e1; »
      - déclare que l'on va utiliser une variable **e1** qui référencera un objet de la classe **Employe**, mais aucun objet n'est créé

Java

Variables

- Déclaration / création (suite)
  - Il aurait fallu écrire :

```
public static void main(String[] args) {
    Employe e1;
    e1 = new Employe("Dupond", "Pierre");
    e1.setSalaire(12000);
    . . .
}
```

Java

Variables

- Désigner les variables d'une instance
  - Soit un objet **o1** ; la valeur d'une variable **v** de **o1** est désignée par
    - o1.v
  - Par exemple,

```
Cercle c1 = new Cercle(p1, 10);
System.out.println(c1.rayon); // affiche 10
```

Java

Variables

- Portée
  - La portée détermine la partie d'un programme, dans laquelle une variable peut être utilisée
  - La portée des variables diffère selon leur emplacement au sein du code Java
  - En fait, lorsqu'une variable est déclarée au sein d'un bloc de code comme une classe ou une méthode, elle ne peut être utilisée que dans ce même bloc

Java

Variables

- Portée : exemple

```
public class classe_comptage {
    /* i est utilisable dans toute la classe y compris dans ses
    méthodes */
    int i = 0;
    public void compteur(){
     /* pas est seulement utilisable dans la méthode
    compteur*/
            int pas = 5;
            while(i <= 100){
                    system.out.println(i + 'rn');
                    i += 5;
            }
     }
}
```

Java

Variables

- Portée
  - Par défaut, **les variables possèdent une portée essentiellement locale** dans leurs blocs respectifs délimités par des accolades

```
{
int x = 1;
// seul x est accessible à cet endroit
{
    // x est accessible à cet endroit,
    // y n'est pas accessible puisque la
    // variable n'est pas encore déclarée
    int y = 2;
    // x et y sont accessibles
}
    // y n'est pas accessible,
    // x est accessible à cet endroit
}
```

Java

Variables

- Portée
  - En se fondant sur cette règle, une variable définie dans une classe peut être accédée dans toute la classe
  - et une autre définie dans une méthode restera accessible dans cette seule méthode

```
public class calcul {
    int a = 10;
    public static void main(String[] args){
        int b = 4;
        System.out.println("resultat : " + a * b);
    }
}
```

Java

Variables

- Portée
  - Il est possible de déterminer la portée des variables plus précisément, par le truchement d'un modificateur d'accès
  - Les variables peuvent avoir une portée globale, soit une accessibilité entre toutes les classes de tous les paquetages, à condition d'utiliser le modificateur d'accès *public* lors de leur déclaration

```
public class affichage {
    public String texte = 'Un texte...';
    public void affiche(){
        system.out.printl(texte);
}
public class modification {
    public void ajout(){
        texte += 'Un second paragraphe...';
    }
}
```

Java

Variables

- Portée
  - Les variables déclarées avec le modificateur d'accès *protected ne sont accédées que par les classes d'un même paquetage*
  - Les variables déclarées avec le modificateur d'accès *private* ne peuvent être accédées que dans la classe elle-même
    - private string chaine = "Une chaîne de caractères...";
- Conseil
  - Un programme reposant sur des variables globales peut être à l'origine de bogues difficiles à détecter
  - C'est pourquoi, il est préférable d'utiliser des variables locales offrant plus de sûreté avec une durée de vie limitée

Java

Variables

- Portée : exemple

```
class calcul{
    int a = 1;
    void addition(){
     int b = 1;
     b += a;
     // a = 2
    }
    void soustraction(){
     int c = 1; c -= a + b;
     /* provoque une erreur car b n'est accessible que dans
     addition() */
    }
}
```

Java

Accès aux membres d'une classe

- Autre nom de portée : encapsulation
  - L'encapsulation est le fait de ne montrer et de ne permettre de modifier que ce qui est nécessaire à une bonne utilisation
  - Java permet plusieurs degrés d'encapsulation pour les membres (variables et méthodes) et les constructeurs d'une classe

Java

Désignation d'instance

- this
  - Le code d'une méthode d'instance désigne l'instance qui a reçu le message (l'instance courante), par le mot-clé **this**
  - et donc les membres de l'instance courante en les préfixant par « **this.** »

Java

Désignation d'instance

- Exemple de **this** implicite

```
public class Employe {
private double salaire;
. . .
public void setSalaire(double unSalaire) {
salaire = unSalaire;
}
public double getSalaire() {
return salaire;
}
. . .
}
```

Implicitement **this.salaire**

Java

Désignation d'instance

- **this** explicite
  - « this » est utilisé surtout dans deux cas :
    1. distinguer une variable d'instance et un paramètre qui ont le même nom :

```
public void setSalaire(double salaire) {
    this.salaire = salaire;
}
```

    2. passer une référence de lui-même à un autre objet :

```
salaire = comptable.calculeSalaire(this);
```

Comptable,
calcule **mon** salaire

Java

Désignation d'instance

- **this** explicite

2. passer une référence de lui-même à un autre objet : autre exemple

```
public Arbre pousser() {
    hauteur ++;
    return this;
}
Arbre a = new Arbre();
a.pousser().pousser().pousser();
```

Java

Désignation d'instance

- Interdit de modifier **this**
  - **this** se comporte comme une variable **final** (mot-clé étudié plus loin), c'est-à-dire qu'on ne peut le modifier
  - le code suivant est interdit :

```
this = valeur;
```

Java

Méthodes et variables de classe

- Variables de classe
  - Certaines variables sont partagées par toutes les instances d'une classe
    - Ce sont les variables de classe (modificateur **static**)
  - Si une variable de classe est initialisée dans sa déclaration, cette initialisation est exécutée une seule fois quand la classe est chargée en mémoire

Java

Méthodes et variables de classe

- Variables de classe
  - Exemple

```
public class Employe {
    private String nom, prenom;
    private double salaire;
    private static int nbEmployes = 0;
    // Constructeur
    public Employe(String n, String p) {
        nom = n;
        prenom = p;
        nbEmployes++;
    }
    . . .
}
```

Java

Méthodes et variables de classe

- Variables de classe
  - Une méthode de classe (modificateur **static** en Java) exécute une action indépendante d'une instance particulière de la classe
  - Une méthode de classe peut être considérée comme un message envoyé à une classe
  - Exemple :

```
public static int getNbEmployes() {
      return nbEmployes;
}
```

Java

Méthodes et variables de classe

- Désigner une méthode de classe
  - Pour désigner une méthode **static** depuis une autre classe, on la préfixe par le nom de la classe :

    ```
    int n = Employe.getNbEmploye();
    ```

  - On peut aussi la préfixer par une instance quelconque de la classe (à éviter car cela nuit à la lisibilité : on ne voit pas que la méthode est **static**) :

    ```
    int n = e1.getNbEmploye();
    ```

Java

Méthodes et variables de classe

- Méthodes de classes
  - Comme une méthode de classe exécute une action indépendante d'une instance particulière de la classe, elle ne peut utiliser de référence à une instance courante (**this**)
  - Il serait, par exemple, **interdit** d'écrire

```
static double tripleSalaire() {
    return salaire * 3;
}
```

Car ceci est équivalent à : return this.salaire*3;

Java

Méthodes et variables de classe

- La méthode static main
  - La méthode **main()** est nécessairement **static**
  - Pourquoi ?
    - La méthode **main()** est exécutée au début du programme
    - Aucune instance n'est donc déjà créée lorsque la méthode **main()** commence son exécution
    - Ça ne peut donc pas être une méthode d'instance

Java

Méthodes et variables de classe

- Blocs d'initialisation **static**
  - Ils permettent d'initialiser les variables **static** trop complexes à initialiser dans leur déclaration :

```
class UneClasse {
    private static int[] tab = new int[25];
    static {
        for (int i = 0; i < 25; i++) {
            tab[i] = -1;
        }
    }. . .
```

  - Ils sont exécutés une seule fois, quand la classe est chargée en mémoire
  - Pour désigner la variable static à partir d'une autre classe :

```
UneClasse.tab[i]
```

- Exercice 2