Modélisation et Programmation par Objets (niveau 1)

Moodle

https ://moodle.umontpellier.fr/course/view.php ?id=25108

Modalités de contrôle des connaissances

- Session 1 :
  - Contrôle continu (1h, 30%)
  - Contrôle terminal (1h30 organisé par la FDS, 70%)
- Session 2 : Contrôle terminal (1h30 organisé par la FDS, 100%)

Enseignants 2021-2022

- Marianne Huchard (cours, TD/TP groupe PEIP A2-1)
- Thomas Georges (TD/TP groupe PEIP A2-3)
- Marie-Laure Mugnier (TD/TP groupe PEIP A2-2)
- Bachar Rima (TD/TP groupe PEIP A2-4)

Pourquoi étudier et utiliser les approches à objets ?

- un mode de pensée et de conception qui s'est étendu à de multiples domaines (BD, logiciel, prog. système, prog. IHM, ...)
- en se combinant avec d'autres paradigmes : par ex. impératif, fonctionnel, événementiel, logique, distribué
- des parentés avec des approches de représentation des connaissances (en particulier les ontologies)
- par sa capacité à monter en abstraction (la programmation s'est nourrie au travers du temps de différentes montées en abstraction)
- comme base de construction et d'inspiration pour de nouveaux paradigmes plus récents, comme les composants, les services et les micro-services

Avantages attendus en termes de construction de logiciel

- stabilité des artefacts construits (données "abstraites", plus stables que les fonctions)
- compréhensibilité des programmes (représentation des entités d'un domaine)
- structuration modulaire favorisant la maintenabilité
- extensibilité des programmes
- économie de développement par la réutilisation

Objectifs du module

Concepts essentiels de l'approche objet en s'appuyant sur :

- un langage de modélisation, UML, normalisé par l'OMG (Object Management Group)
  https://www.omg.org/spec/UML/2.5.1/About-UML/
- un langage de programmation, Java, très répandu, typage statique
  https://docs.oracle.com/javase/tutorial/

UML, un langage de modélisation

UML (Unified Modeling Language)

- langage de modélisation graphique
- issu en 1995 de la fusion de plusieurs méthodes à objets incluant OOSE (Jacobson), OOD (Booch), OMT (Rumbaugh)
- assurant une continuité des concepts depuis les phases amonts jusqu'à l'implémentation

incluant :

- les concepts des approches par objets : classe, instance, classification, etc.
- intégrant d'autres aspects : associations, fonctionnalités, événements, états, séquences, composants, patrons de collaboration, etc.

Modèle

Représentation abstraite d'une réalité
Image simplifiée du monde réel selon un point de vue utile pour :

- comprendre et visualiser (en réduisant la complexité),
- communiquer (langage commun, nombre restreint de concepts),
- mémoriser les choix effectués,
- valider (contrôler la cohérence, simuler, tester).

Modèles et diagrammes UML

Modèles et diagrammes

- Modèle d'utilisation : logique des fonctionnalités
  - Diagramme de cas d'utilisation
- Modèle structurel : concepts, relations, objets/instances
  - Diagramme de classes
  - Diagramme d'instances
- Modèle dynamique : stimuli des objets et leurs réponses
  - Diagrammes de collaboration
  - Diagrammes de séquences
  - Diagrammes d'états
  - Diagrammes d'activités
- Modèle d'implémentation : composants, projection sur le matériel
  - Diagramme de composants
  - Diagrammes de déploiement

Concrétisation en Java

Langage de programmation par objets à classes et à typage statique

- 1991 (Oak), 1995 (Java)
- emprunt de syntaxe à C++ ;
- plus grande simplicité que C++ ;
- abstraction des problèmes de gestion de la mémoire ;
- pas « tout objet », un peu d'introspection ;
- compilateur (programme qui traduit le code Java en code exécutable) et interprète (machine virtuelle, un programme qui exécute le programme Java) généralement inclus dans les navigateurs internet.

Objectifs du premier cours

- introduire les notions de types, de classes, d'instance, d'attributs
- réaliser des premiers modèles et programmes

La notion de classe

Une classe correspond à un concept du domaine modélisé

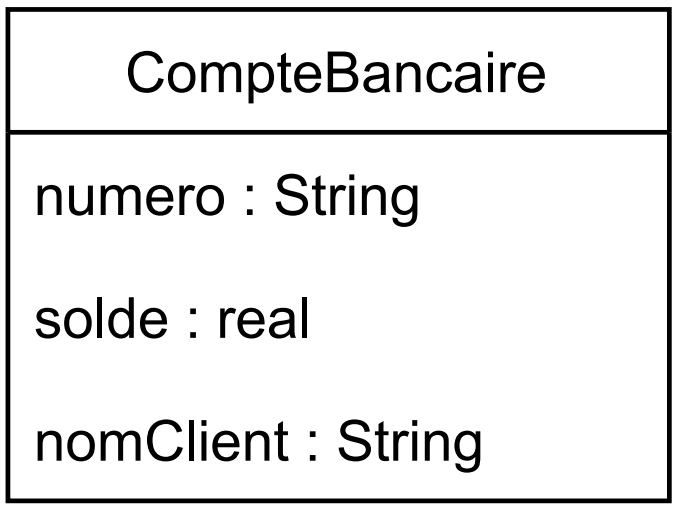
Trois points de vue :

- *extensionnel* objets partageant des caractéristiques
- *intensionnel* caractéristiques partagées
- *générationnel* elle est le modèle des objets engendrés

Un exemple de classe : `CompteBancaire`

- *extensionnel :* des comptes bancaires
- *intensionnel (attributs) :* numéro de compte, solde, client
- *intensionnel (opérations) :* créditer, débiter
- *générationnel :* un compte bancaire sera formé comme une structure dotée des trois attributs

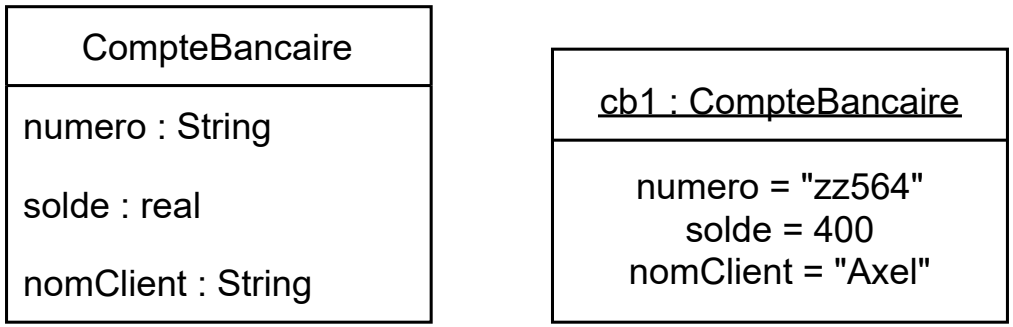
UML : La classe `CompteBancaire`, première version



Notation graphique pour une classe

- classe boîte rectangulaire, compartiments (nom, attributs)
- attribut *identificateur*:*type*

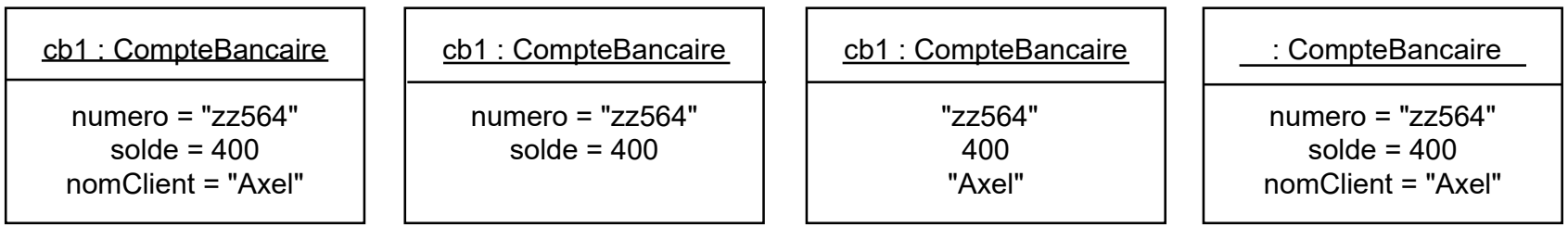
UML : La classe `CompteBancaire`, première version et un objet (instance)



Notation graphique pour un objet (instance à droite)

- objet (instance) boîte rectangulaire, compartiments (identification, attributs et leur valeur)
- nom de l'instance souligné *identification instance* : *classe*
- attribut *identificateur* = *valeur*
  - valeurs `String` (chaîne de caractères) : entre guillemets "Axel"
  - valeurs nombre : notation mathématique usuelle

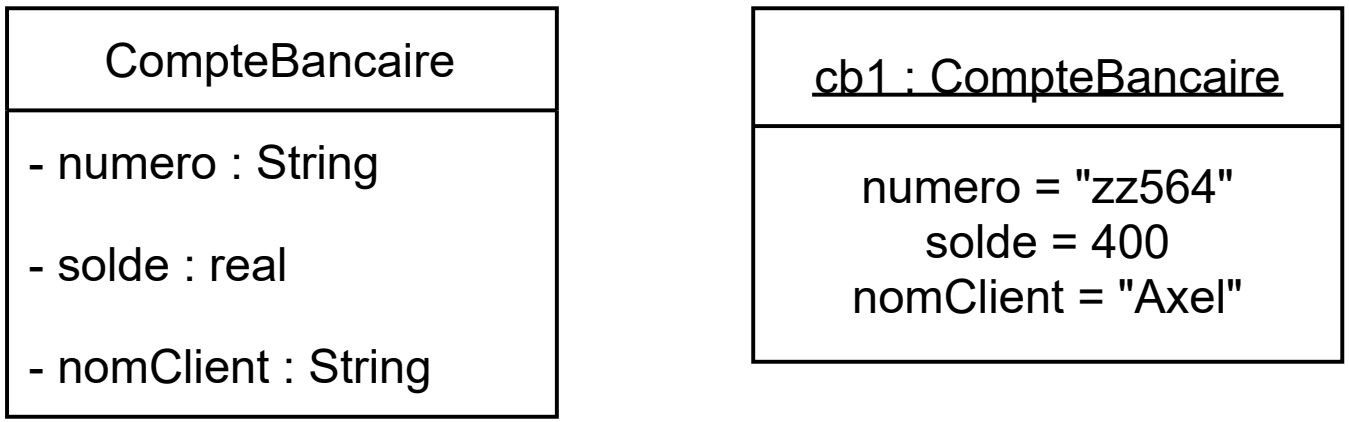
UML : La classe `CompteBancaire`, première version et un objet (instance)



L'objet doit être conforme à la classe et il y a des variantes :

- certains attributs peuvent être omis
- si tous les attributs ont une valeur, les valeurs sont données dans l'ordre de déclaration dans la classe et on peut omettre les identificateurs
- l'identification de l'objet (`cb1`) peut être omise

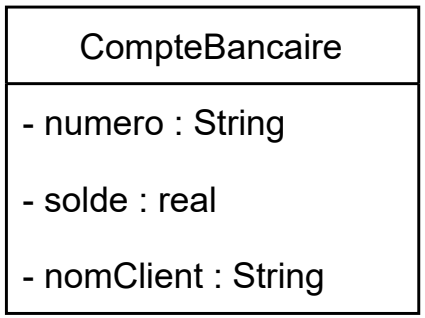
UML : La classe `CompteBancaire`, rafinée, et un objet (instance)

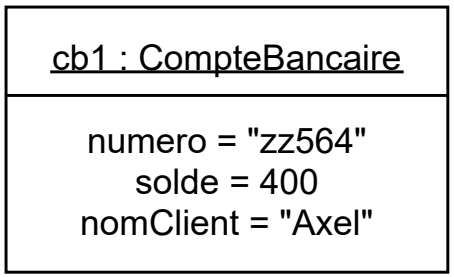


Lorsque l'on raffine le diagramme avant le codage en Java :

- Les attributs sont indiqués comme étant privés dans la classe
- La notation consiste à ajouter, devant l'identificateur, le signe -
- On peut l'indiquer dans l'objet, mais ce n'est pas une obligation (ici ce n'est pas fait)

Java : Classe CompteBancaire






```
public class CompteBancaire {
        private String numero;
        private double solde;
        private String nomClient;
...}
```

- `public` : la classe sera visible "partout"
- `private` : on ne pourra manipuler les attributs d'un compte que dans les opérations de la classe `CompteBancaire` elle-même

Création de comptes

On peut déjà :

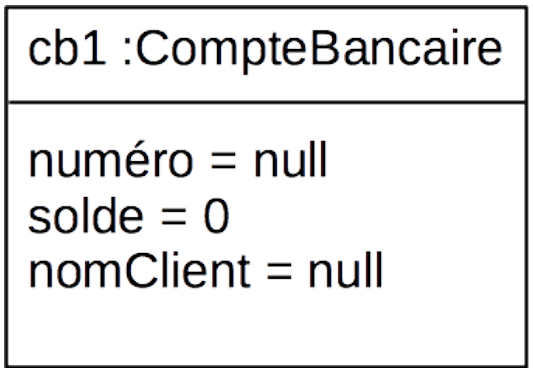
- créer des objets dans une méthode `main` interne à la classe grâce à l'opérateur new

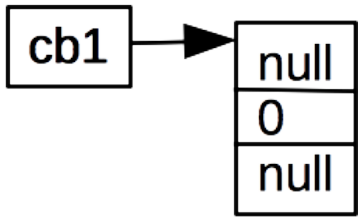
```
public class CompteBancaire {
  private String numero;
  private double solde;
  private String nomClient;

  public static void main(String[] a) {

    CompteBancaire cb1 = new CompteBancaire();  // creation

    // ici on peut acceder à l'attribut car cette methode "main"
    // est dans la classe CompteBancaire
    // ce ne sera pas toujours le cas des methodes "main"
    ....
  }
...
}
```

Instance de CompteBancaire

```
CompteBancaire cb1 = new CompteBancaire();  // creation
```

A gauche représentation UML, à droite schéma qui peut vous servir à vous représenter ce qui se passe dans la mémoire pendant l'exécution du programme :





- les attributs contiennent des valeurs par défaut correspondant à leur type :
  - 0 pour les nombres
  - null pour les classes
  - false pour les booléens

Accès aux attributs

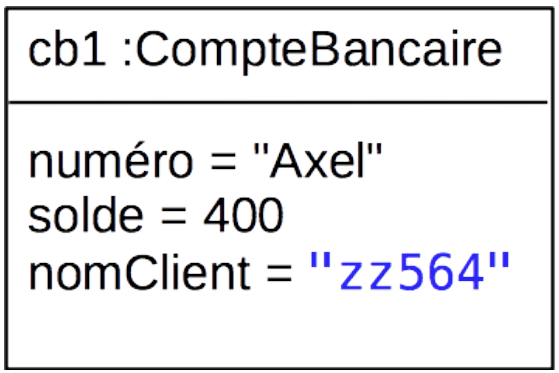
On peut aussi :

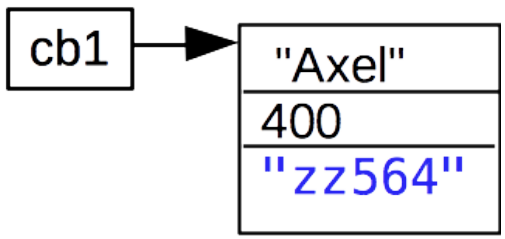
- accéder à leurs attributs (dans la même classe) : opérateur . (point)

```
public class CompteBancaire {
  private String numero;
  private double solde;
  private String nomClient;

  public static void main(String[] a) {

    CompteBancaire cb1 = new CompteBancaire();  // creation

    cb1.nomClient = "Axel";           // acces par l'operateur .
    cb1.solde = 400;
    cb1.numero = "zz564";
  }
...
}
```

Instance de CompteBancaire après modification des attributs

```
cb1.nomClient = "Axel";          // acces par l'operateur .
cb1.solde = 400;
cb1.numero = "zz564";
```

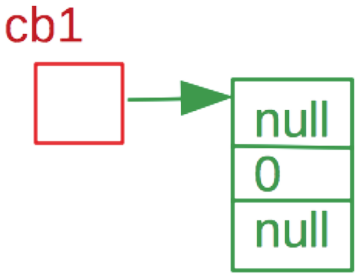




- les valeurs des attributs ont été mises à jour

Détails sur la déclaration et la création

```
CompteBancaire cb1 = new CompteBancaire();
```



Une autre écriture :

```
CompteBancaire cb1 ; // déclaration variable cb1
cb1 = new CompteBancaire(); // création objet
```

Pourquoi les attributs sont-ils privés ?

Règle métier : le découvert d'un compte ne peut dépasser 500 euros ... Mais on peut écrire :

```
cb1.solde = -10000;
```

Dans la classe elle-même, on peut y faire attention, mais dans d'autres programmes utilisateurs de la classe, ce sera plus difficile, donc on contrôlera ces affectations pour avoir des données cohérentes dans un accesseur (voir au prochain cours).

Description détaillée des attributs

`[visibilité][/]nom[:type][[multiplicité]][= valeurParDéfaut]`

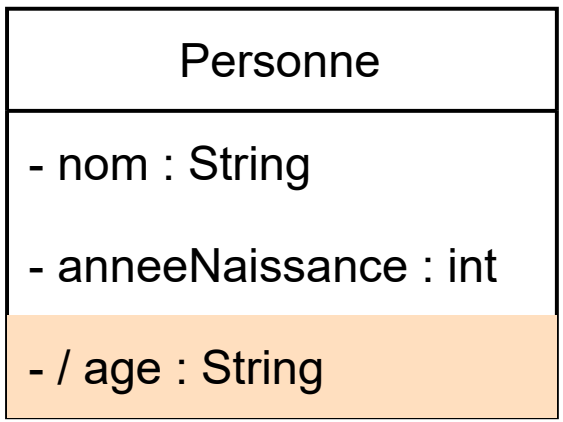
- `visibilité` $\in \{+, -, \#, \sim\}$, et `multiplicité` définit une valeur (1, 2, n, ...) ou une plage de valeurs (1..*, 1..6, ...).
- visibilité : d'où est visible l'attribut ?
  - Public. $+$ est la marque d'un attribut accessible partout (public)
  - Privé. - est la marque d'un attribut accessible uniquement par sa propre classe (privé)
  - Package. $\sim$ est la marque d'un attribut accessible par tout le paquetage
  - Protected. $\#$ est la marque d'un attribut accessible par les sous-classes de la classe
- Le symbole $/$ indique que l'attribut est dérivé
- le nom est la seule partie obligatoire de la description

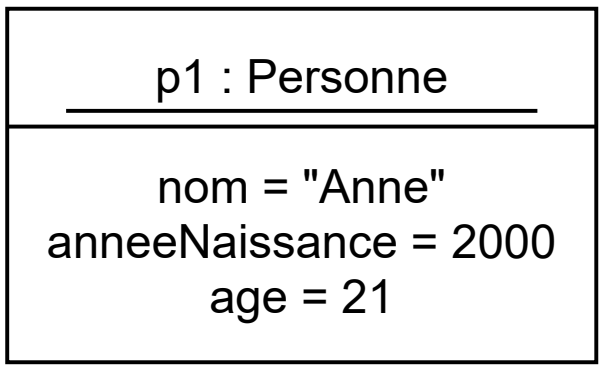
Description détaillée des attributs

`[visibilité][/]nom[:type][[multiplicité]][= valeurParDéfaut]`

- la multiplicité décrit le nombre de valeurs que peut prendre l'attribut (à un même moment). La multiplicité est 1 par défaut (l'attribut a une valeur)
- le type décrit le domaine de valeurs
- la valeur initiale décrit la valeur que possède l'attribut à l'origine
- des propriétés peuvent préciser si l'attribut est constant (`{readOnly}`), si on peut seulement ajouter des valeurs dans le cas où il est multi-valué (`{addOnly}`), etc.

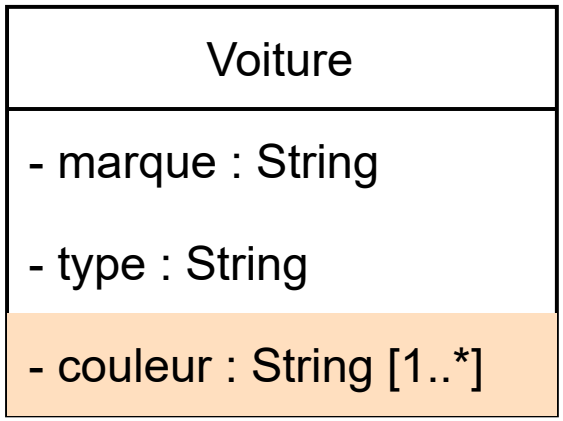
Attribut dérivé

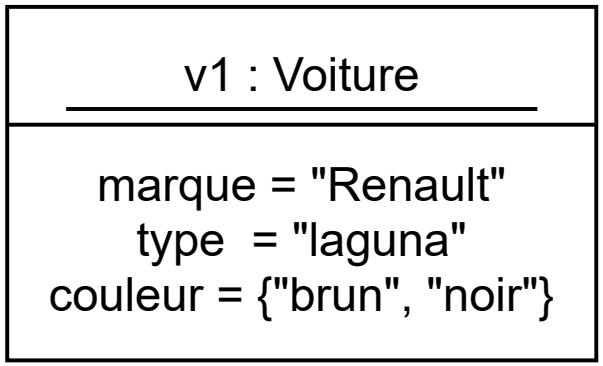




Une formule de calcul permet d'indiquer la manière dont la valeur dérivée est obtenue.
Ici l'âge est la différence entre l'année courante et l'année de naissance.
En Java nous pourrons utiliser une méthode de calcul ou un attribut, il n'y a pas de syntaxe particulière.

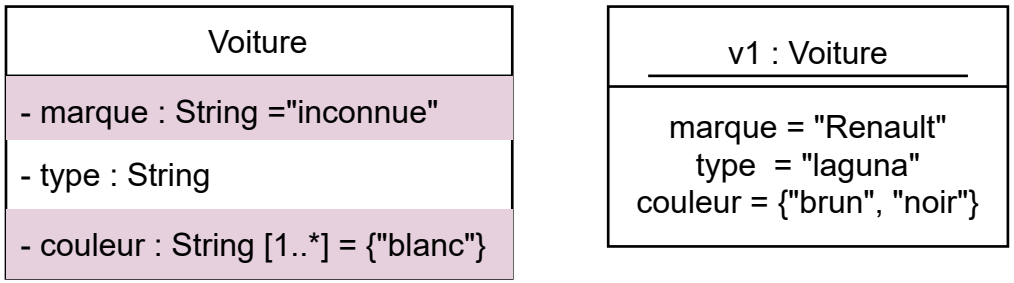
Multiplicité





Ici, une voiture a plusieurs couleurs, et au moins une couleur [1..*].
$v1$ a les couleurs {"brun","noir"}.
En Java, nous verrons dans un prochain cours comment traiter ces attributs.

Valeur initiale



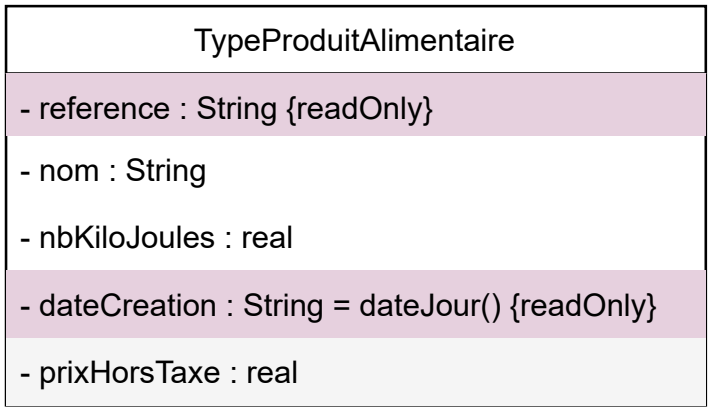
Elle est indiquée derrière le signe $=$.
Ici une voiture est :

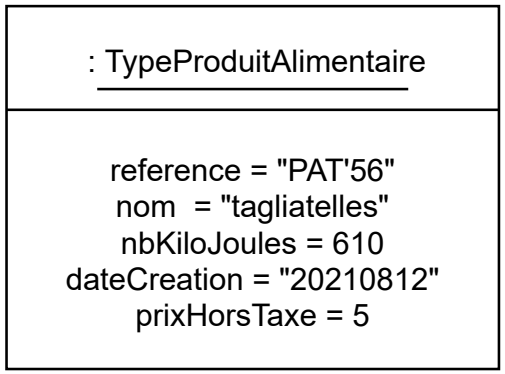
- de marque "inconnue" par défaut
- de couleur blanche par défaut

En Java, on peut le réaliser au moment de la déclaration des attributs. C'est recommandé de le faire pour les `String`.

```
public class Voiture
   private String marque = "inconnue";
   ...
}
```

Attribut de valeur constante (`readOnly`)



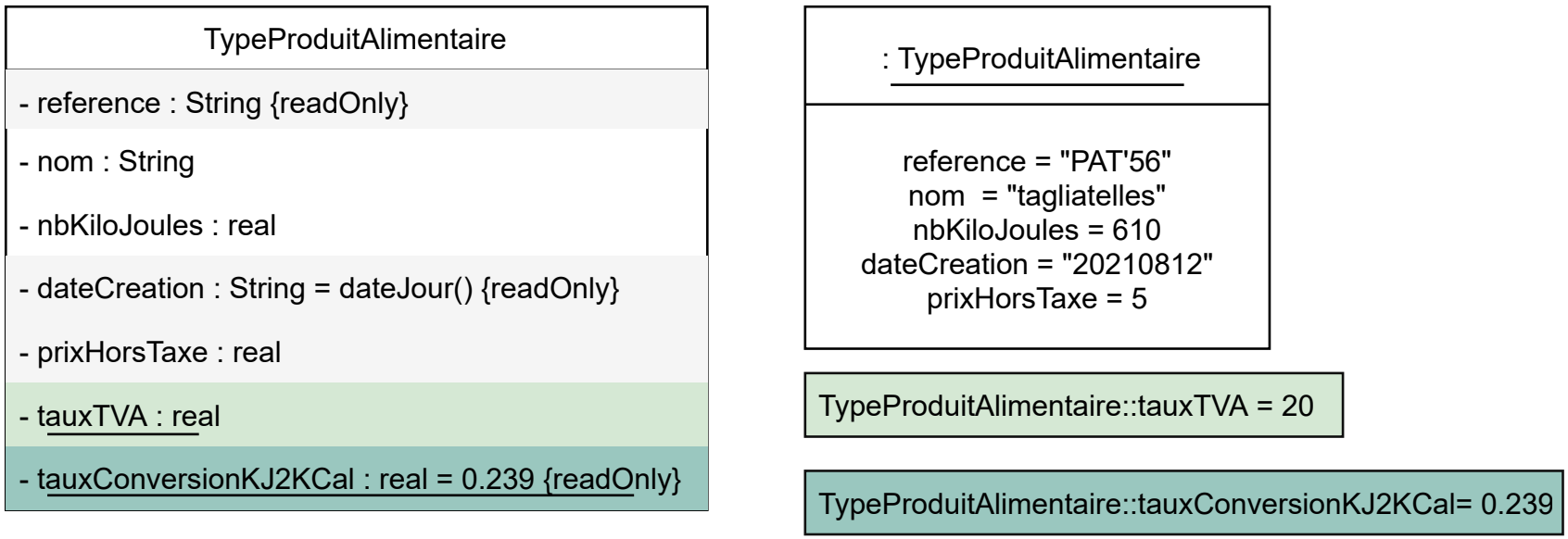


En UML, elle est indiquée par la contrainte readOnly.
La valeur d'un attribut `readOnly` ne peut pas être modifiée après qu'une première valeur ait été affectée. La valeur peut être donnée lors de la description (déclaration) ou plus tard mais une seule fois.

En Java, cela se traduit avec le mot-clef final.

```
public class TypeProduitAlimentaire
   private final String reference;
   private String nom;
   private double nbKiloJoules;
   private final String dateCreation = dateJour(); // operation appelee
   ...
}
```

Attribut dont la portée est la classe et non un objet particulier

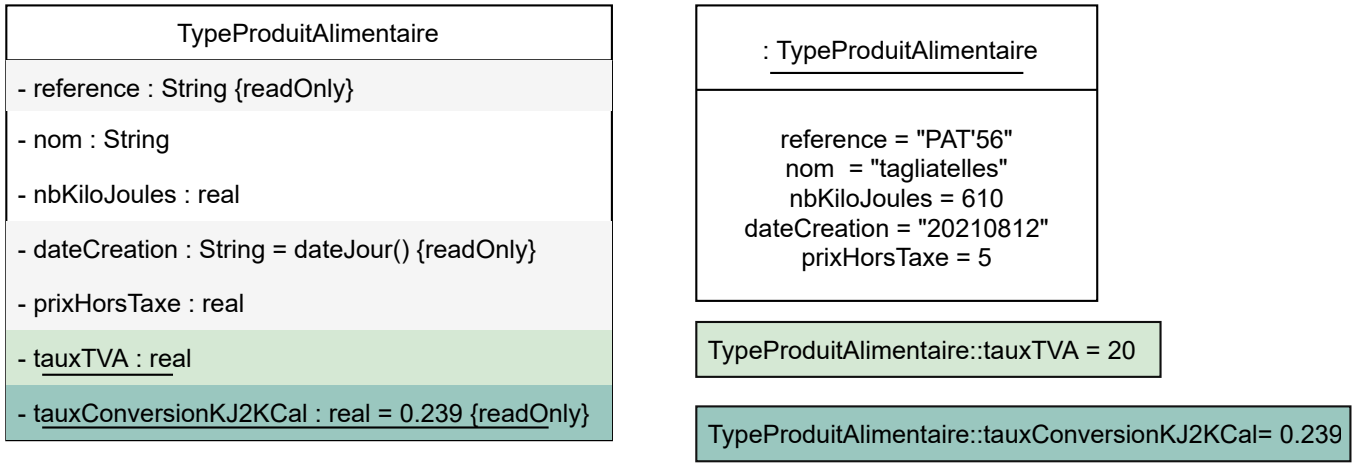


La notation UML consiste à souligner la description de l'attribut dans la classe.

- `tauxTVA` se rapporte à la classe et non à un objet
- `tauxConversionKJ2KCal` se rapporte à la classe et non à un objet ; il est de plus constant

On observe que ces deux attributs n'apparaissent pas dans l'objet (à droite du diagramme), mais comme des variable globales, avec leur nom préfixé par `TypeProduitAlimentaire::`

Attribut dont la portée est la classe et non un objet particulier



En Java, cela se traduit avec le mot-clef static.

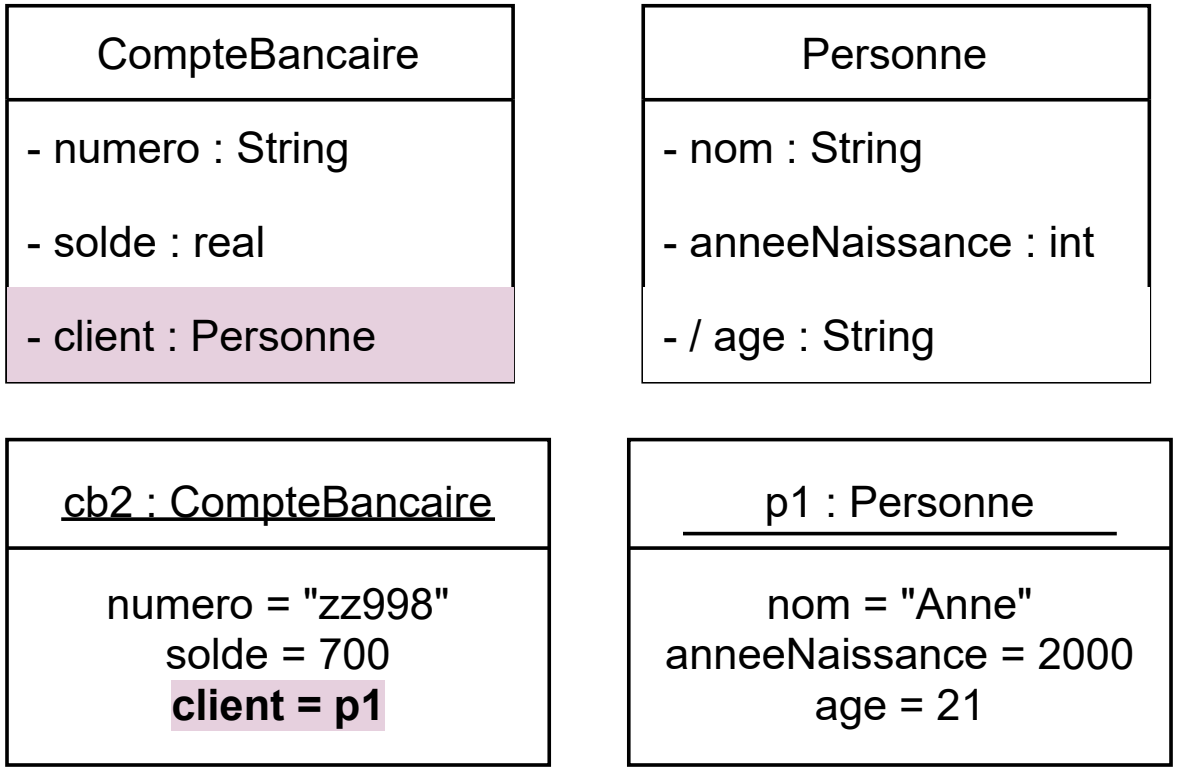
```
public class TypeProduitAlimentaire
   private final String reference;
   private String nom;
   private double nbKiloJoules;
   private final String dateCreation = dateJour();
   private static double tauxTVA;
   private static final double tauxConversionKJ2KCal = 0.239;
}
```

Attribut dont la portée est la classe et non un objet particulier

On accède à un attribut static en utilisant le nom de la classe et non le nom d'un objet avec leur nom préfixé par TypeProduitAlimentaire.

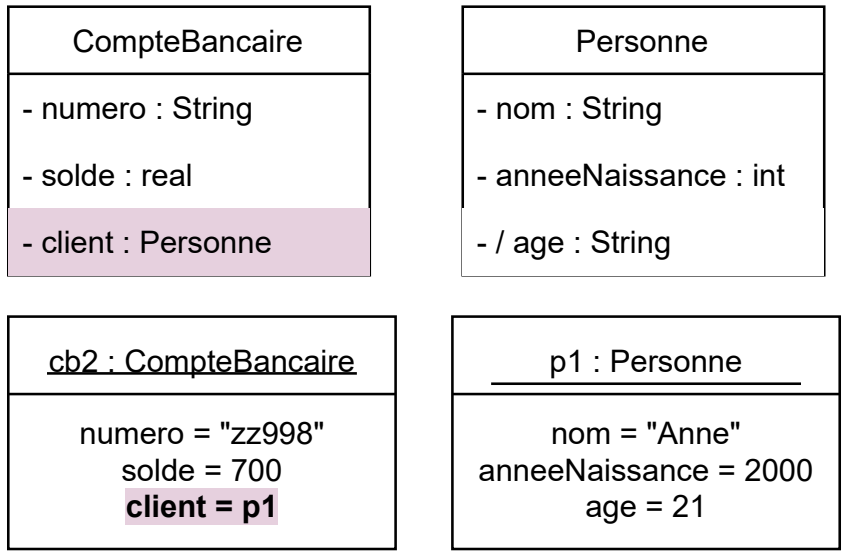
```
public class TypeProduitAlimentaire
   private final String reference;
   private String nom;
   private double nbKiloJoules;
   private final String dateCreation = dateJour();
   private static double tauxTVA;
   private static final double tauxConversionKJ2KCal = 0.239;

   public static void main(String[] a) {
     ....
       TypeProduitAlimentaire.tauxTVA = 20;
     ....
  }
}
```

Attribut dont le type est une classe



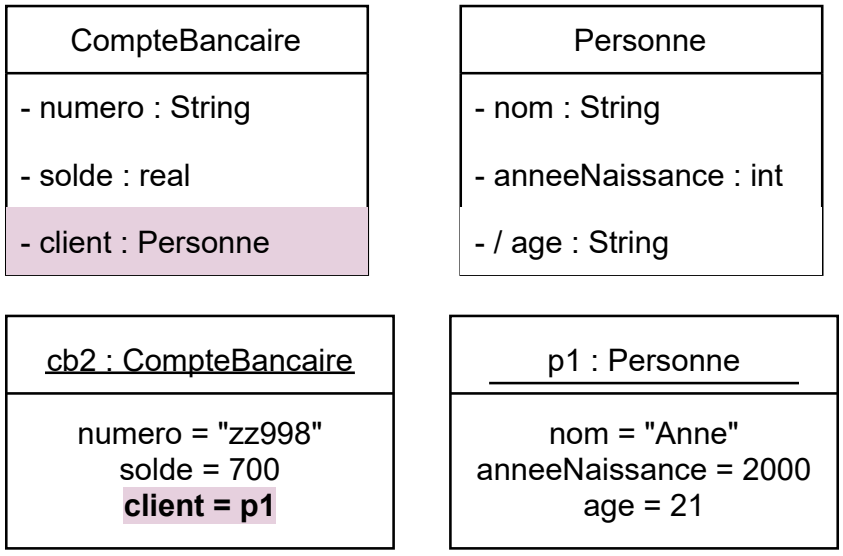
Le type d'un attribut n'est pas forcément un type primitif, ce peut être une classe. C'est déjà le cas pour les String, et on peut aussi utiliser une autre classe du modèle.
Ici plutôt que de stocker le nom du client comme attribut dans le compte, on stocke un client. Cela permet d'accéder à toutes les informations du client (pas seulement à son nom).

Attribut dont le type est une classe



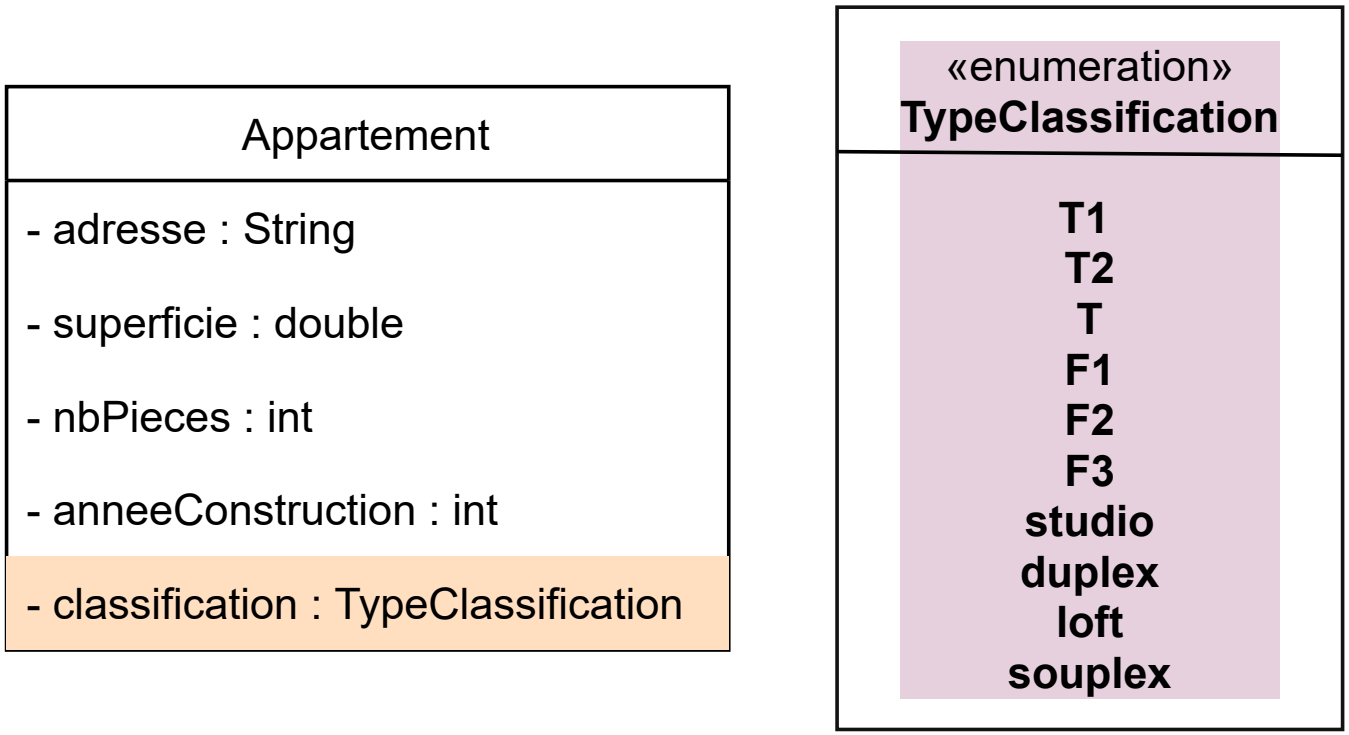

```
public class Personne {
  ...
}

public class CompteBancaire {
  private String numero;
  private double solde;
  private Personne client;
  ...
```

Attribut dont le type est une classe



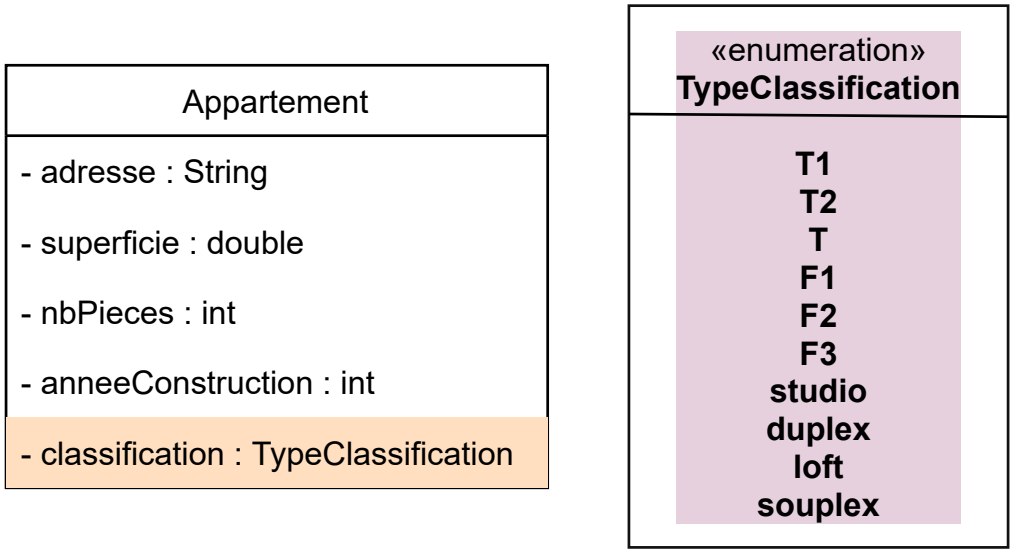

```
public class Personne {  ... }

public class CompteBancaire {
...
  private Personne client;
  ...
     public static void main(String[] a) {
      Personne p1 = ...;
      CompteBancaire cb2 = ...
      cb2.client = p1;
    ....
  }
```

Enumération



Une énumération est un type décrit en extension, c'est-à-dire par la liste de ses valeurs. Un appartement a une classification. Cette classification est décrite par une liste possible de valeurs, rangées dans le type énuméré `TypeClassification`.

Enumération




```
public enum TypeClassification {
    T1, T2, T3, F1, F2, F3, studio, duplex, loft, souplex
}
public class Appartement {.  .....
    private TypeClassification classificaton = TypeClassification.T1;
...
}
```

Notez que la valeur d'un type énuméré est précédée du nom du type.
`TypeClassification.T1`

Structure générale d'un programme Java

```
/*
* Voici un programme simple qui affiche "Bonjour"
*/
package Cours1NotesExpress;
import java.util.Scanner;

public class cours1 { // debut de la classe

  public static void main(String[] args) { // debut du main
            System.out.println("Bonjour");
   } // fin du main

} // fin de la classe
```

Structure générale d'un programme Java

```
package Cours1NotesExpress;
```

- Le fichier se place dans un répertoire dont le nom figure derrière la directive "package".
- Vous pouvez assimiler dans un premier temps : répertoire = package.
- Les répertoires peuvent contenir des sous-répertoires, sans limitation de niveau d'imbrication.

Structure générale d'un programme Java

/*Voici un programme simple qui affiche "Bonjour"*/

//Voici un programme simple qui affiche "Bonjour"

- Dans un texte de programme, on peut placer des commentaires :
  - ceux entre / * et * / peuvent faire plusieurs lignes
  - s'ils ne font qu'une ligne, on peut les introduire par //
  - les outils comme javadoc auront des commentaires particuliers

```
/**
 * Classe exemple pour javadoc
 * @author marianne
 * @version 1.0
 */
public class cours1 { ......

// commentaire sur une ligne

/* commentaire sur
   plusieurs lignes
*/

}
```

Imports

```
import java.util.Scanner;
```

- Des ressources extérieures pourront être nécessaires
  - Dans ce cas on les importe grâce à une directive import
  - Importe ici une classe Scanner (représentation du flux d'entrée)
  - La classe Scanner est rangée dans le répertoire (package) "java" et son sous-répertoire "util" d'où l'écriture java.util

Classe et fichier

public class cours1 { . . . . . }

- Le fichier porte le nom d'une structure de haut-niveau appelée une classe
- Ici le fichier s'appelle cours1.java, et la classe aussi.
- Les accolades { .. } sont des éléments de structuration et peuvent se lire
  - { = début
  - } = fin

Programme Main

```
public class cours1 {

      public static void main(String[] args) {
          System.out.println("Bonjour"); // affiche Bonjour
      }

}
```

- "public" indique que la méthode est visible depuis d'autres programmes y compris dans d'autres packages
- "static" sera la marque de certaines méthodes, dites "statiques"
- "void" indique qu'il s'agit d'une procédure (rien n'est retourné)

Programme Main

```
public class cours1 {

      public static void main(String[] args) {
          ....
      }

}
```

- "main" est le nom de la méthode (de l'algorithme principal)
- entre les parenthèses se trouvent les paramètres, il n'y en a qu'un ici, il est de type "tableau de chaînes (suites) de caractères" et s'appelle "args", il sert à récupérer des informations qui seraient données lors du lancement du programme (nous n'utiliserons pas cette possibilité au début)
- c'est à la première instruction que commencera l'exécution du programme quand on le lancera.

Programme Main et instruction

```
public class cours1 {

      public static void main(String[] args) {
          System.out.println("Bonjour"); // affiche Bonjour
      }

}
```

- Dans cet espace on peut mettre les instructions, par exemple, nous affichons sur la console la suite de caractères "Bonjour"
- "System.out" est le nom d'un objet qui représente la console
- "println" est un algorithme qui écrit sur la console
- "("Bonjour")" est sa liste d'arguments

Types

- Comme en algorithmique, les programmes Java manipulent des valeurs qui appartiennent à des types.
- De manière simplifiée un type est en informatique :
  - un ensemble de valeurs (appelées aussi littéraux, valeurs littérales)
  - un ensemble d'opérations admises sur ces valeurs
- En Java, on trouvera des types :
  - primitifs, simples
  - construits : tableaux, énumérations, classes

Type Entier

- Entier : int
- Ecriture des valeurs littérales : 2 3 12 -2 -3 -12
- Principales opérations : + - * / % < <= > >= == !=
- Quelques expressions (affichées)
  - System.out.println(2) ; // affiche 2
  - System.out.println(-2) ; // affiche -2
  - System.out.println(2+4) ; // affiche 6
  - System.out.println(2 == 4) ; // affiche false (voir ci-dessous !)

Type booléen

- Booléen : boolean
- Ecriture des valeurs littérales : true false
- Principales opérations : ! && & | || == !=
- Utilisez plutôt && (et logique) et || (ou logique) car elles n'évaluent que ce qu'il faut pour conclure
- Quelques expressions (affichées)
  - System.out.println(true) ; // affiche true
  - System.out.println(!true) ; // affiche false
  - System.out.println(false && true) ; // affiche false
  - System.out.println(false || true) ; // affiche true
  - System.out.println(2 >=4 && 2 >= 1) ; // affiche false

Type réel

Il en existe plusieurs, ici nous considérons les réels en double précision

- Réel : double
- Ecriture des valeurs littérales : 2 -2 2.01 2.01E-3 -2.01E+3
- Principales opérations : + - * / < <= > >= == !=
- Quelques expressions
  - System.out.println(2.01) ; // affiche 2.01
  - System.out.println(2.01E-3) ; // affiche 0.00201
  - System.out.println(-2.01E+3) ; // affiche -2010.0

Type Caractère

- Caractère : char
- Ecriture des valeurs littérales : 'a' 'z' '\t' '\n'
- Principales opérations : + < <= > >= == !=
- Quelques expressions :
  - System.out.println('a') ; // affiche a
  - System.out.println('2') ; // affiche 2
  - System.out.println(""+'a'+'2') ; // affiche a2
  - System.out.println('a'+'2') ; // affiche 147 (code ascii de 'a' + code ascii de '2')
  - System.out.println('a'<'2') ; // affiche false (rang dans les codes ascii)
  - System.out.println('A'<'a') ; // affiche true (rang dans les codes ascii)

Le type construit "chaîne de caractères"

- Chaîne de caractères : String
- Ecriture des valeurs littérales : "abricot" "orange" "Ligne1 \n Ligne2"
- Principales opérations : + equals compareTo charAt
- Quelques expressions, pour certaines la syntaxe sera clarifiée lorsque nous développerons la programmation par objets en Java
  - System.out.println("abricot") ; // affiche abricot
  - System.out.println("abricot"+"ier") ; // affiche abricotier
  - System.out.println("abricot"+'s') ; // affiche abricots
  - System.out.println("Ligne1 \n Ligne2") ; // affiche
    Ligne1
    Ligne2
  - System.out.println("abricot".compareTo("Abricot")>0) ; // affiche true
  - System.out.println("abricot".compareTo("orange")>0) ; // affiche false
  - System.out.println("abricot".equals("abricot")) ; // affiche true

Le type construit "chaîne de caractères"

Deux points d'attention

- ne pas comparer les chaînes avec l'opérateur d'égalité == utiliser equals et compareTo
- lorsqu'une chaîne est attendue, par exemple comme paramètre de System.out.println, elle est transformée en chaîne

  Ex. System.out.println("L'article "+ 5 + "est vendu") ;

  Les chaînes et l'entier 5 sont concaténées

Déclarer une variable

- < Type > <nomVariable> ;
- Comprenez pour le moment une variable comme une petite boîte dont le nom est le nom de la variable
- Son format (sa dimension) est proportionné au format du type de la variable. Quelques déclarations de variables :
  - int i ;
  - double d ;
  - String s ;

Affecter une valeur à une variable

- on utilise le symbole =
- attention aux étourderies consistant à le confondre avec la comparaison ==
  - Avec `int i;` i est une petite boîte dans laquelle on met l'entier 4
    `i = 4;`
  - on ne peut pas mettre une valeur booléenne dans i
  - Avec `double d;` d est une petite boîte dans laquelle on met le réel 3.7E+1
    `d = 3.7E+1;`
  - Avec `String s;` s est une petite boîte dans laquelle on met le mot "fraise"
    `s = "fraises";`

On peut combiner ....

```
i = i+2;
```

- utilise le contenu de i (4), lui ajoute 2 et remet le tout dans i
- i vaut 6 après

Affichage d'une expression

- Sans passer à la ligne

```
System.out.print(s+" à la chantilly - ");
```

- En passant à la ligne

```
System.out.println("charlotte aux "+s);
```

Saisie d'une valeur

Elle va nous demander plus d'efforts et l'ajout d'un nouvel objet représentant le "clavier" ou plus exactement le flux d'entrée du programme.
On se rappelle que l'on a importé cette ressource tout en haut de notre programme !

```
Scanner clavier = new Scanner(System.in);

// on veut saisir une valeur et la mettre
// dans la boîte (variable) i

i = clavier.nextInt();
// il y a une variété d'opérations de type "next"

// quand on a fini, on "ferme" la connexion avec le clavier

clavier.close();
```

Synthèse

- Approches à objets : conceptualiser le domaine
- Classes, objets (instances), attributs
- Attributs : visibilité, valeur initiale, dérivé, final, static
- Enumération
- Structure d'un programme Java

Pour travailler chez vous

- Installer eclipse sur son propre ordinateur
- A défaut, utiliser https ://replit.com/
  - codage en ligne par le navigateur, pas d'installation
  - un environnement moins riche, un peu lent, mais facile à prendre en main
  - créez un compte personnel
  - vous pouvez créer un dépôt avec New repl (en Java) et remplir le `main`
  - vous pouvez exécuter en appuyant sur triangle vert (Run), l'exécution apparaîtra dans la fenêtre à fond noir à droite