Cours Programmation Orientée Objet en Java

Cours 1

Anne Parrain

UFR des Sciences - Université d'Artois
Licence Informatique
Semestre 4

Année universitaire 2018–2019

Section 1

Introduction

Le langage Java

- créé en 1995 (James Gosling, Patrich Naughton - SUN Microsystems)
- les cinq objectifs lors de sa création
  - simple, orienté objet et familier
  - robuste et sûr
  - indépendant de la machine employée pour l'exécution
  - très performant
  - compilé, multi-tâches et dynamique.
- Présent dans de très nombreux domaines d'application : des serveurs d'applications aux téléphone portables et cartes à puces (JME).
- Java est un langage couramment utilisé du fait de ses qualités de fiabilité, de robustesse et de maintenabilité.

Le langage Java (2)

- familier : syntaxe à la C
- robuste et sûr : langage fortement typé
- orienté objet : différent de Python !
- indépendant de la machine employée pour l'exécution et performant : interprété mais aussi compilé !
- Java est fourni dans un éco-système :
  - différents outils pour le compiler, générer la documentation, ...
  - Java Virtual Machine : l'interprète Java
  - les API Java : un ensemble de librairies

Les différent environnements Java

- Java EE : "Enterprise Edition". Rajoute certaines API et fonctionnalités pour les entreprises.
- Java ME : “Micro Edition”. Édition pour les applications embarquées Ex. : téléphone portable, carte à puce
- Java SE : “Standard Edition” :
  - JRE : “Java Runtime Environment”. Contient la plate-forme Java (JVM + API).
  - JDK : (“Java Development Kit”). Contient le langage de programmation et la plate-forme (compilateur + JVM + API).

Compilation et interprétation

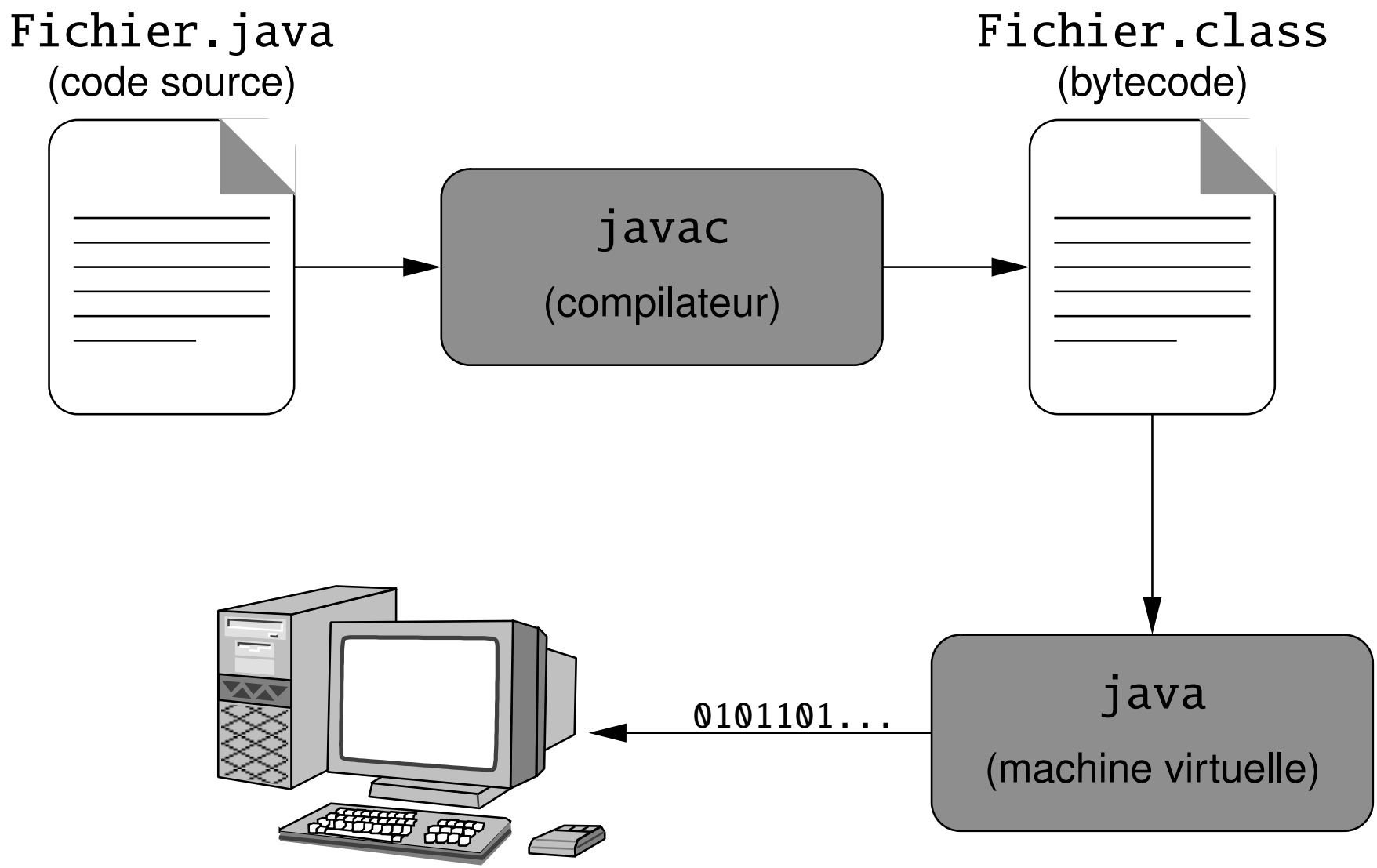
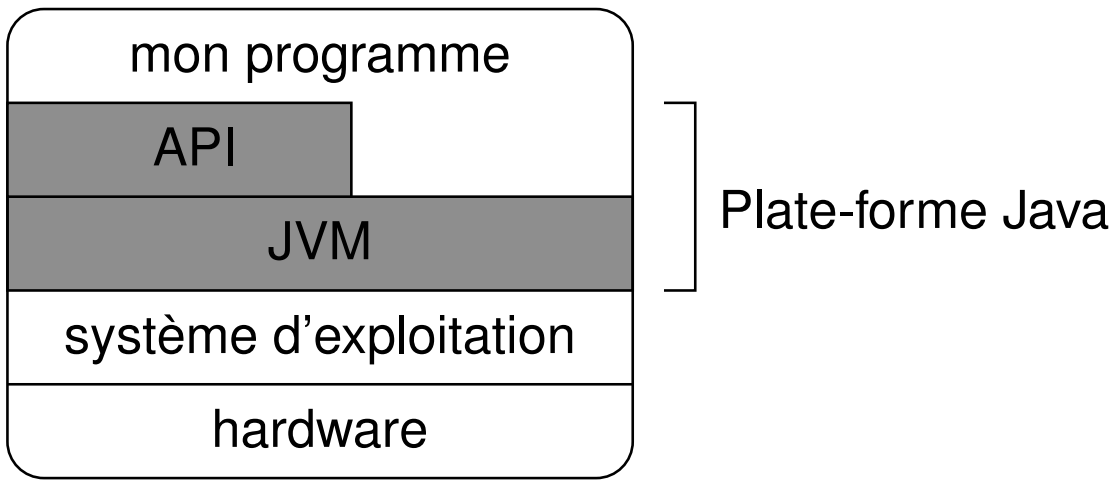


Plate-forme Java

- Un programme Java est exécuté par la JVM, en utilisant les API.



La variable d'environnement `CLASSPATH` contient le chemin d'accès à l'API.

Premier programme en Java

- en Java, on ne peut écrire aucune instruction en dehors d'une classe
- le programme principal s'appelle obligatoirement `main`

Fichier `PremierExemple.java`

```java
/** Classe qui lance une application pour afficher un message.
Ceci est le premier exemple du cours.
*/
public class PremierExemple {

    /** Methode pour lancer l'application.
    Cette methode contient juste une instruction
    d'affichage d'un message.
    @param args : parametre de la fonction principale
    */
    public static void main(String[] args){
        System.out.println("Bonjour les L2 !");
    }
}
```

Premier programme en Java (2)

- Première étape : la compilation

```
chezmoi:~/poo$ ls
PremierExemple.java
chezmoi:~/poo$ javac *.java
chezmoi:~/poo$ ls
PremierExemple.class PremierExemple.java
```

  - On doit compiler tous les fichiers `.java` nécessaires à l'application.
  - On obtient autant de fichiers `.class` en bytecode que de classes

- Deuxième étape : l'interprétation

```
chezmoi:~/poo$ java PremierExemple
Bonjour les L2 !
```

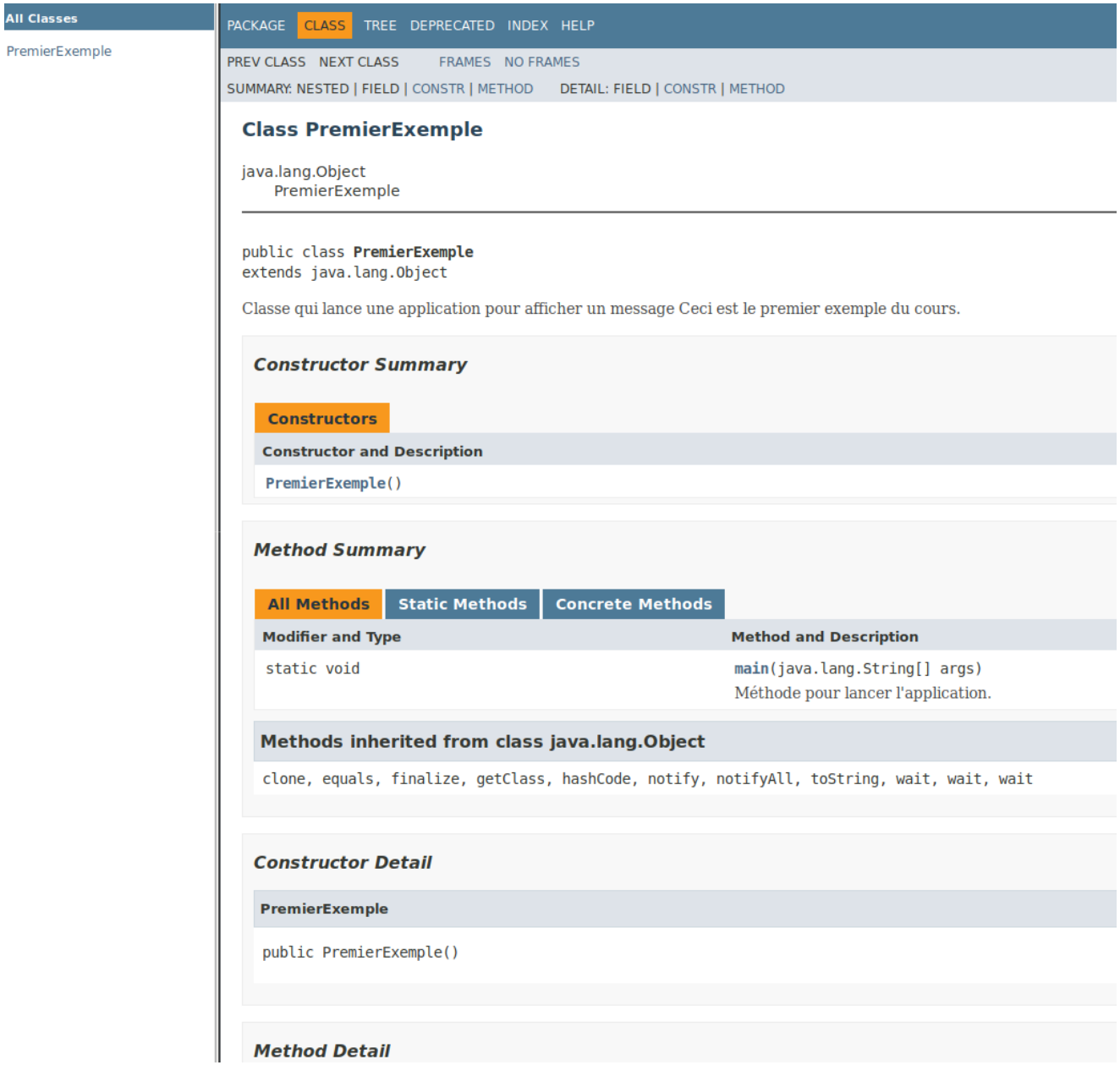
  - **Attention !** l'interprète `java` prend en paramètre le nom de la classe qui contient la méthode `main` (ce n'est pas un nom de fichier, donc pas de suffixe `.class`)
  - le fichier bytecode correspondant doit se trouver dans le répertoire courant.

Premier programme en Java (3)

- Troisième étape : on génère la documentation

```
chezmoi:~/poo$ mkdir doc
chezmoi:~/poo$ ls
doc  PremierExemple.class PremierExemple.java
chezmoi:~/poo$ javadoc -charset utf8 *.java -d doc
chezmoi:~/poo$ ls doc/
allclasses-frame.html index-all.html   package-summary.html
allclasses-noframe.html index.html     package-tree.html
constant-values.html  overview-tree.html PremierExemple.html
deprecated-list.html  package-frame.html script.js
help-doc.html         package-list     stylesheet.css
```

Premier programme en Java (4)



All Classes

PremierExemple

PACKAGE CLASS TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class PremierExemple

java.lang.Object
    PremierExemple

public class **PremierExemple**
extends java.lang.Object

Classe qui lance une application pour afficher un message Ceci est le premier exemple du cours.

Constructor Summary

Constructors

Constructor and Description
PremierExemple()

Method Summary

All Methods | **Static Methods** | **Concrete Methods**

Modifier and Type	Method and Description
static void	main(java.lang.String[] args) Méthode pour lancer l'application.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

PremierExemple

public PremierExemple()

Method Detail

Paradigme Objet

- Un paradigme de programmation correspond à une manière de modéliser le monde.
- Il existe plusieurs paradigmes :
  - programmation impérative (ex. : Pascal, C, Fortran)
  - programmation fonctionnelle (ex. : Scheme, Lisp, Haskell)
  - programmation logique (ex. : Prolog)
  - programmation orientée objet (ex. : C++, Java).
- Dans le paradigme objet :
  - Le monde est modélisé comme un ensemble d'objets.
  - Les objets ont un état interne et un comportement.
  - Ils collaborent en s'échangeant des messages.

Paradigme objet en Java

- en Java, tout est classe et objets (sauf types simples prédéfinis)
- on trouve dans une classe :
  - des attributs : des variables qui contiennent l'état de l'objet
  - des méthodes : qui définissent le comportement de l'objet
  - une méthode particulière : le constructeur
- mais aussi des attributs de classe et des méthodes de classe

Différences avec Python - éléments de base

- les types simples prédéfinis ne sont pas des objets
- contrôle strict et modulable de l'accès aux attributs et aux méthodes
- typage fort des méthodes
- surcharge des méthodes

Exemple des étudiants

- un étudiant est caractérisé par son nom et son prénom
- il possède des notes (au maximum 10)
- on ne peut pas changer son nom ni son prénom
- on peut lui ajouter une note (dans la limite de 10 notes)

Classe Etudiant

Création de la classe et déclaration des attributs

```
/** classe qui modelise la gestion d'un etudiant dans une
    universite.
*/
public class Etudiant {
    public final static int MAXNOTES = 10;
    public final static int MINVALUE = 0;
    public final static int MAXVALUE = 20;

    private String nom;
    private String prenom;

    private float[] lesNotes = new float[MAXNOTES];
    private int nbNotes; // nombre actuel de notes de l'etudiant
```

Classe Etudiant (2)

Constructeurs et accesseurs

```
/** Constructeur d'un etudiant.
@param nom : le nom de l'etudiant
@param prenom : le prenom de l'etudiant
*/
public Etudiant(String nom, String prenom) {
    this.nom = nom;
    this.prenom = prenom;
}
/** accesseur pour le nom de l'etudiant
@return le nom de l'etudiant */
public String getNom(){
    return nom;
}
/** accesseur pour le prenom de l'etudiant
@return le prenom de l'etudiant */
public String getPrenom(){
    return prenom;
}
```

Classe Etudiant (3)

S'afficher, tester la validité d'une note. Déclaration de méthodes.

```
/** teste si une note est bien dans une echelle de 0 a 20.
    */
public static boolean valide(float note){
    return !((MINVALUE > note) || (note > MAXVALUE));
}

/** retourne une presentation textuelle d'un etudiant :
son nom, son prenom, son nombre de notes */
public String toString(){
    return prenom+" "+nom+", "+nbNotes+" note(s).";
}
```

Classe Etudiant (4)

S'afficher, ajouter une note. Déclaration de méthodes. Documentation.

```
    /** ajoute une note a un etudiant
    dans la limite de la capacite de l'etudiant
    @param note : la note qu'on veut ajouter a l'etudiant
    @return vrai si l'ajout a pu etre fait */
    public boolean ajouteNote(float note){
        if (! valide(note))
            return false;
        if (nbNotes < lesNotes.length){
            lesNotes[nbNotes] = note;
            nbNotes++;
            return true;
        }
        return false;
    }
} // fin class Etudiant
```

Attributs

- les attributs sont déclarés à l'intérieur de la classe, mais en dehors du constructeur
- ils peuvent être initialisés à la déclaration
- ils sont spécifiés par un quantificateur d'accès
  - `public` : accessible de l'intérieur ou de l'extérieur de la classe, sans contrôle, en lecture et en écriture
  - `private` : accessible uniquement depuis l'intérieur de la classe
- par défaut un attribut est un attribut d'instance
- si on précise `static`, c'est un attribut de classe
  - c'est une variable partagée par toutes les instances de la classe
- le mot-clé `final` est accolé à un attribut dont la valeur ne peut changer
- les constantes ont vocation à être déclarées `public static final`

Méthodes

- comme les attributs, elles sont spécifiées par un quantificateur d'accès
  - `public` : accessible de l'intérieur ou de l'extérieur de la classe, sans contrôle, en lecture et en écriture
  - `private` : accessible uniquement depuis l'intérieur de la classe
- par défaut, une méthode est une méthode d'instance
  - elle manipule des attributs d'instance
- si on précise `static`, c'est une méthode de classe
  - c'est une méthode partagée par toutes les instances de la classe
  - elle ne peut pas accéder à un attribut ou une méthode d'instance

```
public final static MINVALUE = 0;
public final static MAXVALUE = 20;

/** teste si une note est bien dans une echelle de 0 a 20.
    */
public static boolean valide(float note){
   return !((MINVALUE > note) || (note > MAXVALUE));
}
```

Tableaux

- Java propose des tableaux comme en C
  - ils sont créés pour un nombre fixe d'éléments
  - il est (presque toujours) nécessaire de mémoriser le nombre réel d'éléments du tableau
  - l'attribut `length` des tableaux retourne la taille réservée pour le tableau

```
private float[] lesNotes = new float[MAXNOTES];
private int nbNotes; // nombre actuel de notes de l'etudiant

Etudiant[] promo = new Etudiant[10];
```

Tableaux (2)

Pour parcourir un tableau :

```
/** Calcule la moyenne de l'etudiant
@return retourne la moyenne des notes de l'etudiant s'il en
    a, -1 sinon. */
public float moyenne(){
    if (nbNotes == 0)
        return -1;
    float som = 0;
    for (int i = 0; i < nbNotes; i++)
        som += lesNotes[i];
    return som/nbNotes;
}
```

Classe TestEtudiant

Une classe pour tester la classe `Etudiant`

```
/** Classe qui permet de lancer une application qui teste la
    classe Etudiant.
 */
public class TestEtudiant {
    public static void main(String[] args){
        Etudiant etud1 = new Etudiant("Dupont","Jean");
        Etudiant etud2 = new Etudiant("Durand","Paul");
        System.out.println(etud1);
        System.out.println(etud2);
        etud1.ajouteNote(15);
        etud2.ajouteNote(17);
        etud1.ajouteNote(13);
        etud2.ajouteNote(8);
        etud1.ajouteNote(-4);
        etud1.ajouteNote(23);
        System.out.println(etud1);
        System.out.println(etud2);
    }
}
```

Classe TestEtudiant (2)

- un fichier = une classe java (pas tout à fait vrai, on reviendra plus tard)
- `etud1` et `etud2` sont des variables locales à la méthode `main`
- leur portée est réduite au bloc dans lequel elles ont été déclarées
- le mot-clé `new` est l'instruction qui provoque la création d'un objet :

```
float[] lesNotes = new float[MAXNOTES];
Etudiant etud1 = new Etudiant("Dupont","Jean");
```

- seulement pour les chaînes de caractères on omettra l'instruction `new` :

```
String nom = "Dupont";
```

- il n'est pas utilisé pour créer un `int` ou un `float` qui ne sont pas des objets
- la méthode `System.out.println` fait appel (de manière implicite) à la méthode `toString` des objets `etud1` et `etud2`

Test de la classe Etudiant

```
chezmoi:~/poo$ ls
doc           TestEtudiant.java
Etudiant.java PremierExemple.java
chezmoi:~/poo$ javac TestEtudiant.java
chezmoi:~/poo$ ls
doc            Etudiant.java PremierExemple.java
    TestEtudiant.java
Etudiant.class TestEtudiant.class
chezmoi:~/poo$ java TestEtudiant
Jean Dupont, 0 note(s).
Paul Durand, 0 note(s).
Jean Dupont, 2 note(s).
Paul Durand, 2 note(s).
```

Test de la classe Etudiant (2)

- demander la compilation de `TestEtudiant.java`
  - provoque la compilation de `Etudiant.java`
  - mais ne provoque pas la compilation de `PremierExemple.java`
- la compilation d'un fichier entraîne la (re-)compilation de toutes les classes utilisées dans la classe correspondante (si nécessaire)

Classes et Objets

- Concept de base de la programmation orientée objet : la classe
- Une classe modélise la structure statique (données membres) et le comportement dynamique (méthodes) des objets associés à cette classe.
- Un objet d'une classe est appelé une instance.
- Une classe est la description d'un objet.
- Chaque objet est créé à partir d'une classe (avec l'opérateur `new`).

Objet

Un objet a :

- une identité : adresse en mémoire
- un état : la valeur de ses attributs
- un comportement : ses méthodes

Objet (2)

- Un objet est une instance d'une seule classe :
  - il se conforme à la description que celle-ci fournit,
  - il admet une valeur (qui lui est propre) pour chaque attribut d'instance déclaré dans la classe,
  - ces valeurs caractérisent l'état de l'objet
  - il est possible de lui appliquer toute opération (méthode) définie dans la classe
- Tout objet admet une identité qui le distingue pleinement des autres objets :
  - il peut être nommé et être référencé par un nom (la variable qui le désigne)

Références

- Pour désigner des objets dans une classe (attributs ou variables dans le corps d'une méthode) on utilise des variables d'un type particulier : les références
- Une référence contient l'adresse d'un objet pointeur vers la structure de données correspondant aux attributs (variables d'instance) propres à l'objet.
- Une référence peut posséder la valeur `null`
  - aucun objet n'est accessible par cette référence
- Déclarer une référence ne crée pas d'objet
  - une référence n'est pas un objet, c'est un nom pour accéder à un objet
  - on créé un objet avec l'opérateur `new`

Références (2)

```
Etudiant etud1 = new Etudiant("Dupont","Jean");
Etudiant etud2 = new Etudiant("Durand","Paul");
Etudiant etud3;
etud3 = etud1;
```

- `etud1`, `etud2` et `etud3` sont des références.
- Comme un pointeur une référence contient l'adresse d'une structure
- Mais à la différence des pointeurs la seule opération autorisée sur les références est l'affectation d'une référence de même type
- La référence d'un objet est utilisée pour accéder aux données et fonctions membres de l'objet.
- Un objet peut accéder à sa propre référence grâce à la valeur `this` (variable en lecture seule).
- Quand un objet n'est plus utilisé (aucune variable du programme ne contient une référence sur cet objet), il est automatiquement détruit et la mémoire récupérée (garbage collector).

Section 2

Éléments syntaxiques

Variables et attributs

- toute variable (variable locale ou attribut) doit être déclarée avant son utilisation
- une variable ne peut pas changer de type (différent de Python)
- un attribut peut être initialisé à sa déclaration, sinon une valeur par défaut lui est affectée
- Les types primitifs sont les seuls éléments qui ne sont pas des objets en Java :

```
byte short int long float double boolean char
```

- pour affecter une valeur à une variable d'un type primitif, on n'utilise jamais de `new`

Types primitifs

Type	valeurs possibles	valeur par défaut
byte	entiers 8-bit	`0`
short	entiers 16-bit	`0`
int	entiers 32-bit	`0`
long	entiers 64-bit	`0L`
float	virgule flottante 32-bit	`0.0f`
double	virgule flottante 64-bit	`0.0d`
boolean	`true`, `false`	`false`
char	16-bit (caractère unicode)	`'\u0000'`

Attention : les valeurs par défaut sont affectées uniquement aux attributs en l'absence d'initialisation à la déclaration.

Types primitifs (2)

```
boolean result = true;
char capitalC = 'C';
byte b = 100;
short s = 10000;
int i = 100000;

int decVal = 26;  // Le numero 26, en decimal
int octVal = 032;  // Le numero 26, en octal
int hexVal = 0x1a; // Le numero 26, en hexadecimal

double d1 = 123.4;
double d2 = 1.234e2; // notation scientifique
float f1 = 123.4f;
```

Opérateurs

Par ordre de priorité :

unaires postfixés	++ - -
unaires (préfixés)	++ - - - !
multiplicatifs	* / %
additifs	+ -
décalage	>> <<
relationnels	< > <= >=
égalité	== !=
et bit à bit	&
ou exclusif bit à bit	^
ou bit à bit	|
conjonction coupe-circuit	&&
disjonction coupe-circuit	||
conditionnel	? :
affectation	= += -= *= /= %= &= ^= |= <<= >>=

Opérateurs (2)

Quelques remarques :

- Opérateur `++` (resp. `--`) préfixé évalue l'expression avant l'incrementation (resp. décrémentation).
- Opérateurs `&&` et `||` présentent le « short circuit behaviour » : le deuxième opérande est évalué seulement si nécessaire.
- Opérateur `+` est utilisé aussi pour la concaténation des chaînes de caractères (`String`).

Structures de contrôle

- Affectation :
  `Variable = Expression ;`
- Bloc de commande :
  `{ Commande ; Commande ; ... ; }`
- Contrôle de flux :

```
if ( Expression ) Commande

if ( Expression ) Commande if Commande

switch ( Expression ) { case ExpConstante : Commande
... [default : Commande] }

while ( Expression ) Commande

do Commande while ( Expression ) ;

for (Commande ; [Expression] ; [Commande]) Commande
```

Commentaires

- Documentation :
  - entre /** et */
  - juste avant l'élément qu'elle documente (classe, attribut, méthode, ...)
- Commentaires de code :
  - après // et jusqu'à la fin de la ligne
  - entre /* et */ et sur plusieurs lignes.