Programmation Orientée Objet
Chapitre 1 : JAVA et la POO

Pr OMAR EL MIDAOUI

pr.oelmidaoui@gmail.com

Plan

- Présentation de Java
- Le paradigme objet
  - Classes
  - Objets
- L'essentiel du langage de Java

Présentation de Java

JAVA c'est quoi ?

- Une technologie développée par SUN Microsystem^{TM} lancé en 1995
- Un langage de programmation
- Une plateforme, environnement logiciel dans lequel les programmes java s'exécutent
- Présent dans de très nombreux domaines d'application : des serveurs d'applications aux téléphone portables et cartes à puces

Quelques caractéristiques de JAVA : Java est un langage simple, orienté objet, distribué, robuste, sûr, indépendant des architectures matérielles, portable, de haute performance, multithread et dynamique

Caractéristiques

- Simple & Robuste
  - Abandonner les éléments mal compris ou mal exploites dans d'autres langages
    - Pas de pointeur
    - Pas d'héritage multiple
  - Mécanisme d'exceptions pour la gestion des erreurs
- Portable
  - Java n'est pas compilé à destination d'un processeur particulier mais en "byte code" (code portable) qui pourra être ensuite interprété sur une machine virtuelle (JVM = Java Virtual Machine).
  - Le "byte code" généré est vérifié par les interpréteurs java spécifique à une machine donné avant exécution.
  -
  -

Caractéristiques

- Sûr
  - La sécurité fait partie intégrante du système d'exécution et du compilateur.
  - Il ne peut pas y avoir d'accès direct la mémoire.
- Multi thread
  - Une applications peut être décomposée en unités d'exécution (Threads) fonctionnant simultanément
- Dynamique
  - Les classes Java peuvent être modifiées sans avoir à modifier le programme qui les utilise.
- Politique
  - Java est actuellement totalement controlé par Oracle.

Plusieurs Java

- Java EE : “Enterprise Edition”. Rajoute certaines API et fonctionalités pour les entreprises.
- Java ME : “Micro Edition”. Édition qui sert à écrire des 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).

Les différentes version de java

- Java 1.0
  - 8 packages
  - 212 Classes et Interfaces
  - 1545 Méthodes
- Java 1.1
  - 23 packages
  - 504 Classes et Interfaces
  - 3 851 Méthodes
- Java 1.2
  - 60 packages
  - 1 781 Classes et Interfaces
  - 15 060 Méthodes
- Et bien plus encore dans l es versions s uivantes [$\rightarrow$ *Java* 17]

L'environnement de travail

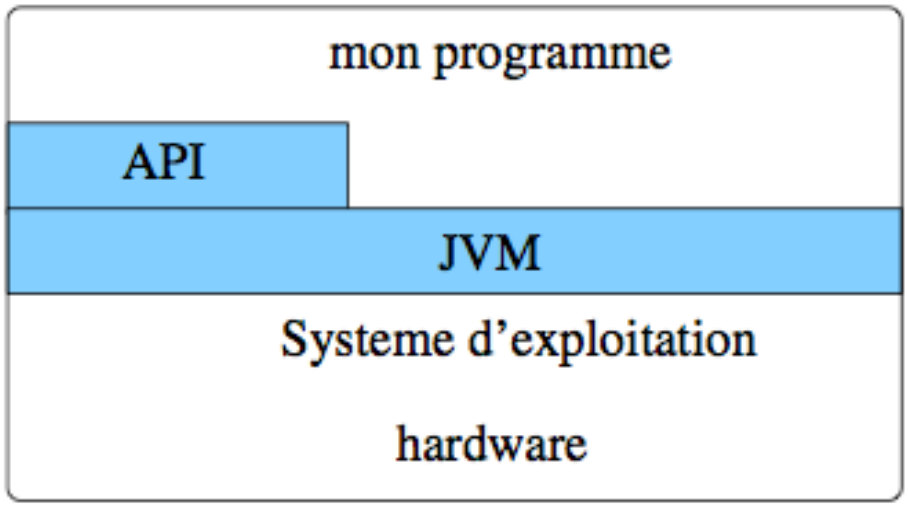
- Un éditeur de texte Ex. : Emacs, gedit, GVim, Sublime, NotePad++
- Le compilateur (javac)
- La JVM (java)
- Le générateur automatique de documentation (javadoc)
- Le debogueur (jdb)
- La documentation du JDK 13, disponible à : URL
- Dans La suite de ce cours on va utiliser l'IDE Eclipse 2019-09 R, disponible à : URL

Plate-forme Java

- La plate-forme est le matériel (“hardware”) et/ou l'environnement logiciel dans lequel un programme s'exécute.
- La plate-forme Java est un environnement logiciel, composée de deux parties :
  - API : (“Application Programming Interface “) grande collection de composants logiciels qui offrent de nombreuses fonctionnalités.
    Ex.: API de Paypal.
  - JVM : (“Java Virtual Machine”) le logiciel qui interprète le bytecode généré par le compilateur Java.

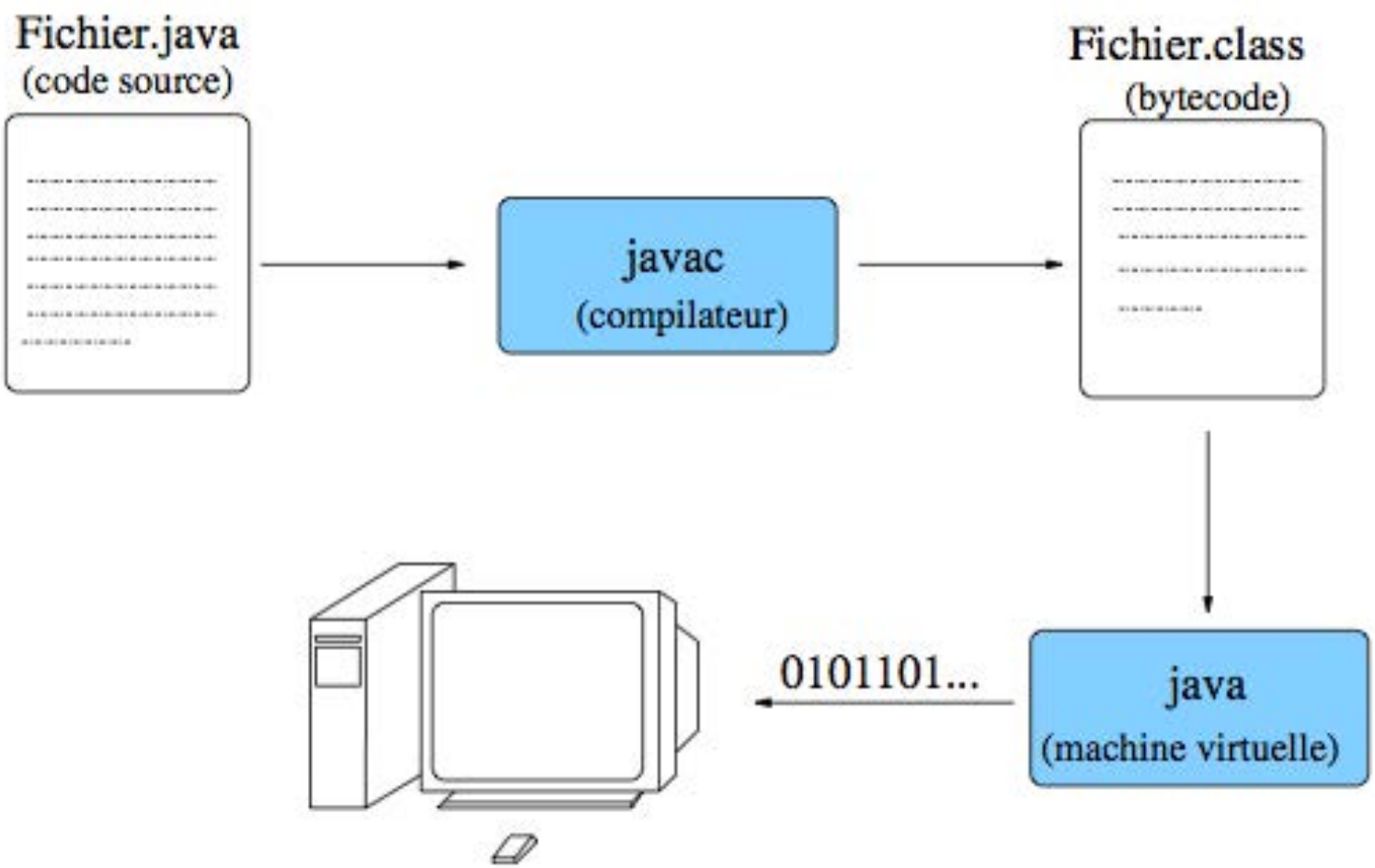
Plate-forme Java (2)

- Un programme Java est exécuté par la JVM, qui utilise les APIs.



Plate–forme Java

Compilation et exécution



La machine virtuelle

- Il y a des JVM pour la plupart des systèmes
  - Ex. : Windows, Linux, Mac OS, Solaris
- Si un système possède une JVM, il peut exécuter le bytecode généré sur n'importe quel autre système.
- Avantages de cet approche :
  - **Portabilité** : le bytecode peut être chargé depuis une machine distante sur Internet.
  - **Sécurité** : la JVM effectue des nombreuses vérifications sur le bytecode pour éviter les actions "dangereuses".

La machine virtuelle (2)

- Désavantage de cet approche : lenteur.
- Mais, des nouvelles techniques essayent de minimiser ce problème :
  - Ex., la traduction en code binaire des parties du bytecode qui sont utilisés très fréquemment.

Utilisation de JAVA

- Le Langage java peut générer
  - des applications
  - des applets
  - des servlets
  - etc.

Génération du code exécutable

- Le code est généré par un compilateur en plusieurs étapes :
  - Vérification syntaxique.
  - Vérification sémantique (typage).
  - Production de code dans un langage plus proche de la machine.

Production de code

- Avantages/inconvénients du code natif
  - Rapidité d'exécution
  - Nécessité de recompiler lors du portage d'un logiciel sur une autre architecture/système d'exploitation
- Solution Java, production de code intermédiaire: le bytecode

Un programme en Java

Code source (dans un fichier texte) :

HelloWorld.java

```
class HelloWorld{
public static void main(String[ ] args) {
System.out.println("Hello World!");
}
}
```

Un programme en Java (2) : Le main()

- Le point d'entrée pour l'exécution d'une application Java est la méthode statique **main** de la classe spécifiée à la machine virtuelle
- Profil de cette méthode
  - **public static void main(String [] args)**
- **String args** ???
  - args : tableau d'objets String (chaînes de caractères) contenant les arguments de la ligne de commande

Un programme en Java (3): Compilation et exécution

Compilation (dans la console) :

$ javac HelloWorld.java

Le compilateur recoit un nom d'un fichier ayant pour suffixe .java. Ensuite, il génère le bytecode dans un fichier ayant pour suffixe .class. Exécution (dans la console) :

$ java HelloWorld
Hello World!

Attention : La JVM recoit un nom d'une classe (donc, pas de suffixe .class). Le fichier contenant le bytecode de la classe doit être présent dans le même dossier.

Le paradigme objet

Paradigmes de programmation

- Un paradigme de programmation correspond à une manière de modéliser le monde.
- Il existent plusieurs paradigmes :
  - programmation **impérative** (ex. : Pascal, C, Fortran) ;
  - programmation **fonctionnelle** (ex. : Scheme, Lisp) ;
  - 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.

Qu'est-ce qu'un objet ?

- **Toute entité identifiable, concrète ou abstraite.**
  - Ex. : personne, stylo, table, ordinateur, vélo, logiciel.
- Deux caractéristiques importantes :
  - État
  - Comportement
- L'objet vélo :
  - États : vitesse, couleur, direction, etc.
  - Comportements : accélérer, s'arrêter, tourner à droite, etc.

Concept de classe

- Concept de base de la programmation orientée objet : la classe
- Une classe modélise :
  - La structure statique (données membres)
  - Le comportement dynamique (méthodes)

  des objets associés à cette classe.

Concept de classe

- Une classe est une implémentation d'un type abstrait de données.
- Une classe **encapsule** les attributs (propriétés) et méthodes (fonctions membres).
- Dans un fichier **.java** , une seule classe de visibilité **public** peut être codée. Le fichier prend le nom de la classe.
- Par convention, le nom d'une classe commence toujours par une **majuscule**.

Notion d'objet en programmation

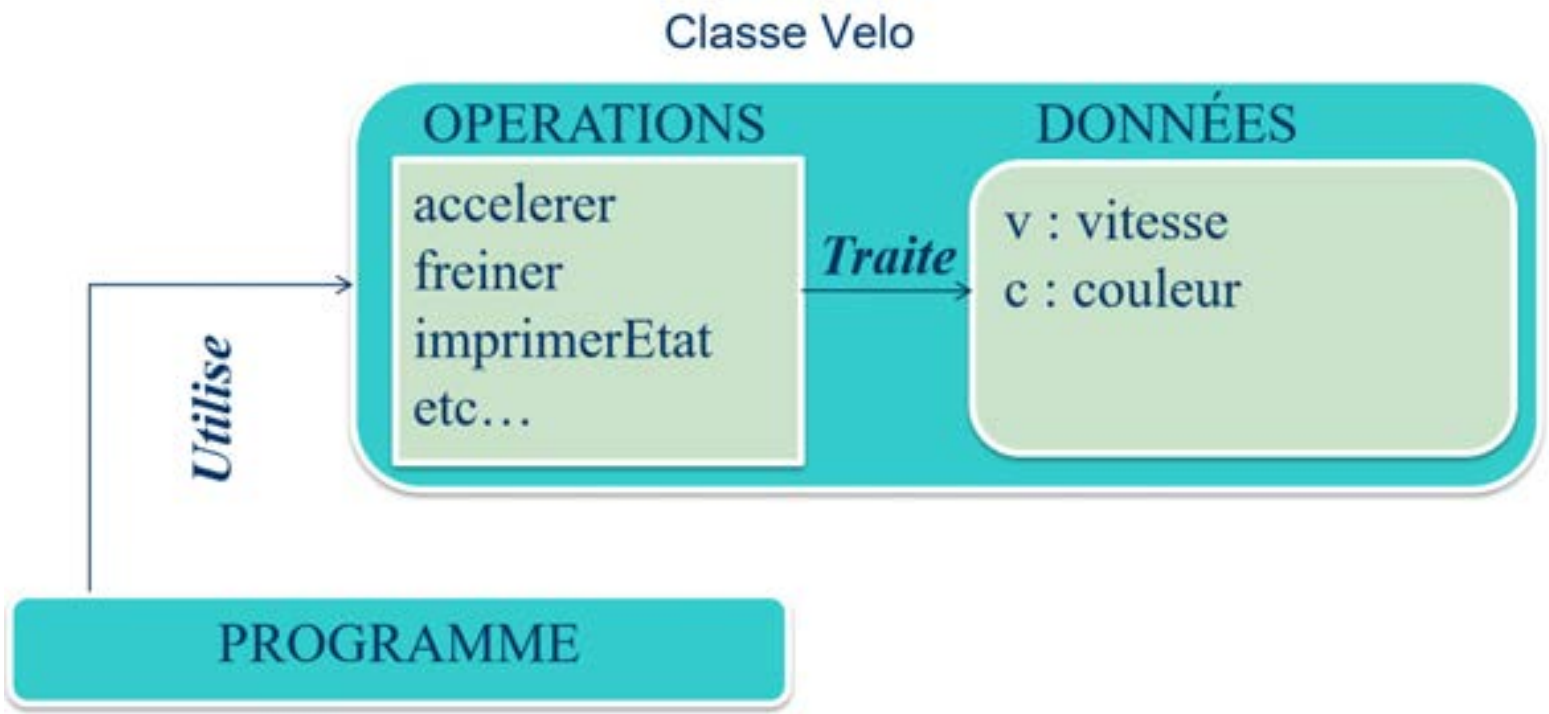
- Un objet d'une classe est appelé une instance.
- Une classe est la description d'un objet. Chaque objet est créé à partir d'une classe (avec l'opérateur **new**).

Un objet a :

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

Classes d'objets

- Une définition abstraite selon laquelle les objets sont crées (un type d'objet)
  - Ex. : la classe des vélos, la classe des stylos
- **Exemple** : Définir une classe Velo :
  - Attributs : vitesse, couleur
  - Méthodes : Accélérer, Freiner et ImprimeEtat



Définition d'une classe

Velo.java

```
class Velo {
int vitesse = 0;
String couleur;
void accelerer(int increment) {
vitesse = vitesse + increment; }
void freiner(int decrement) {
vitesse = vitesse - decrement; }
void imprimeEtat() {
System.out.println("vitesse: " + vitesse); }
}
```

Création des objets et messages

DemoVelo.java

```
class DemoVelo {
public static void main(String[] args) {
// Genere deux objets differents du type Velo
Velo velo1 = new Velo();
Velo velo2 = new Velo();
// Invoque les methodes
velo1.accelerer(10);
velo1.imprimeEtat();
velo2.accelerer(20);
velo2.imprimeEtat();
}
}
```

Compilation et exécution

- Compilation :

  $ javac Velo.java DemoVelo.java

- Exécution :

  $ java DemoVelo
  vitesse: 10
  vitesse: 20

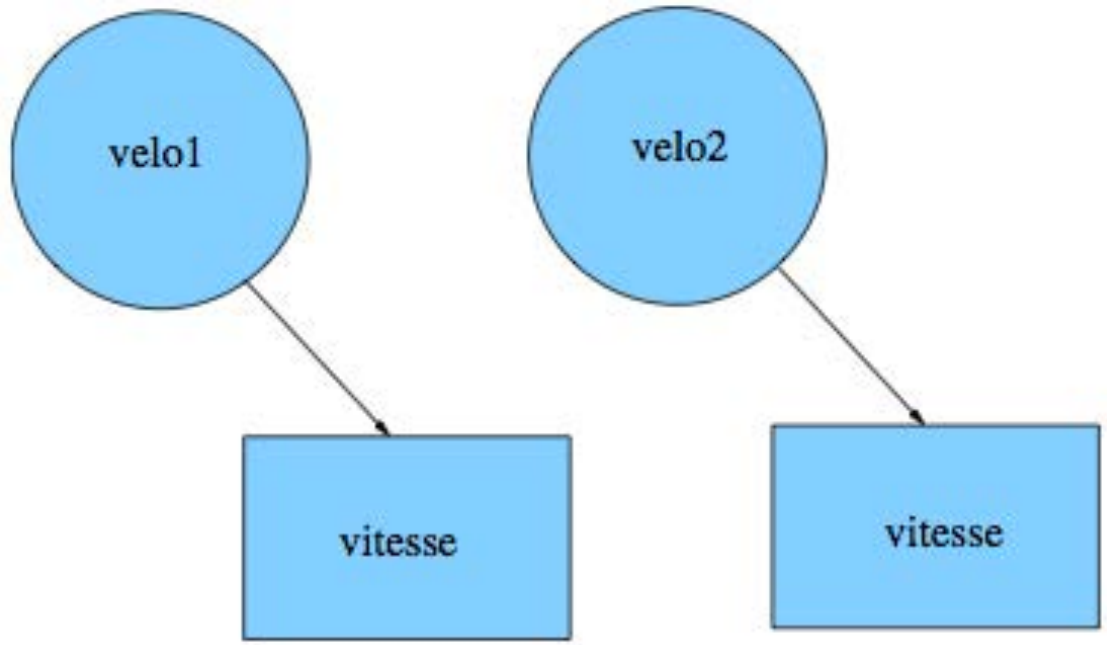
- Un programme source Java correspond à plusieurs fichiers .java.
- Chaque fichier .java peut contenir une ou plusieurs définitions de classes.

Objets

- 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 déclaré dans la classe,
  - ces valeurs caractérisent **l'état** de l'objet
  - il est possible de lui appliquer toute opération (**méthode**) définit 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.

Objets : exemple

Chaque objet vélo instance de la classe Velo possédera sa propre vitesse.



Créations d'Objets

- **new constructeur (liste de paramètres)**
- les constructeurs ont le même nom que la classe
- il existe un constructeur par défaut

```
Velo v1;
v1 =  new Velo();
Velo v2 = new Velo();
Velo v3 = v2;
```

Représentation mémoire

Représentation mémoire

- vélo1 et vélo2 contiennent l'adresse des zones mémoires allouées par l'opérateur new pour stocker les informations relatives à ces objets.
- vélo1 et vélo2 sont des références.
- 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**.

Représentation mémoire

- Une référence contenant la valeur null ne désigne aucun objet.
- Quand un objet n'est plus utilisé (aucune variable du programme ne contient une référence sur cet objet).
- L'objet non utilisé est automatiquement détruit et la mémoire est récupérée par le **garbage collector**.

- L'essentiel du langage de Java
  - Constructeurs et destructeur de classe
  - Déclaration : Classe, Attribut et Méthode
  - Visibilité et contôle d'accés

Constructeur de classe

- Un **constructeur** est une méthode automatiquement appelée au moment de la création de lóbjet.
- Un constructeur est utile pour procéder à toutes les initialisations nécessaires lors de la création de la classe.
- Le constructeur porte le même nom que le nom de la classe et n'a pas de valeur de retour.

Exemple de constructeur

```
class Velo{
double vitesse;
String couleur;
public Velo (double vit,String cl)
{
vitesse = vit ;
couleur = cl ;
}
}
Velo V1 = new Velo (20.5, rouge) ;
```

Lors de la création d'un autre constructeur outre que celui par défaut alors la definition de celui ci n'est plus fourni automatiquement dans la classe.

Constructeur de copie

Le constructeur de copie permet d'initialiser une instance en copiant l'état d'une autre instance du même type.

```
class Velo{
double vitesse;
String couleur;
public Velo (Velo vl)
{
vitesse = vl.vitesse ;
couleur = vl.couleur ;
}}
Velo V1 = new Velo (20.5, rouge) ;Velo V2 = new Velo (V1) ;
```

Lors de la création d'un autre constructeur outre que celui par d éfaut alors la définition de celui ci n'est plus fourni automatiquement dans la classe.

Destructeur de classe

- En Java, les destructeurs appelés finaliseurs (finalizers), sont automatiquement invoqués par le garbage collector.
- Un destructeur permet déxécuter du code lors de la libération de léspace mémoire occupé par l'objet.
- Pour créer un finaliseur, il faut redfinir la méthode: **void finalize ()** héritée de la classe Object.

```
Class Velo{
...
public void finalize()
{
System.out.println("Objet nettoyé de la mémoire");
}
}
```

Fin de vie d'un objet

- La fin de vie d'un objet a lieu lorsque la référence qui lui est associée n'est plus utilisée null part.

Cas pratique

```
Class DemoVelo{
public static void main(String[] args)
{
Début du code
System.out.println("Objet nettoyé de la mémoire");
//Suite du code }
static void AfficherVelo()
{ Velo v=new Velo(20.0d, rouge) //Création locale de l'objet
System.out.println(v); }
}
```

Déclaration : Classe, Attribut et Méthode

Classes

- Composants de la déclaration d'une classe (dans l'ordre) :
  1. Modificateurs (optionnels)
  2. Mot-clé **class** suivie du nom de la classe (obligatoire)
  3. Mot-clé **extends** suivie du nom de la superclasse (optionnel)
  4. Mot-clé **implements** suivie d'une liste de noms d'interfaces (optionnel, expliqué plus tard)
  5. Corps de déclaration entouré par { et }

```
public class Vtt extends Velo implements InterfaceVelo
{
// Declarations des attributs et methodes
}
```

Attributs

- Composants de la déclaration d'un attribut (dans l'ordre) :
  1. Modificateurs (optionnels)
  2. Type
  3. Nom

```
private int vitesse;
```

Les modificateurs d'accessibilité

A l'intérieur d'une classe, une variable ou une méthode peut être définie avec un modificateur d'accès. Les différents modificateurs d'accessibilité sont :

- **private** l'élément (variable ou méthode, d'instance ou de classe) est privé, il n'est accessible que depuis la classe elle-même (le code de la classe dans laquelle il est défini);
- **pas de modificateur d'accès** l'accès est dit *package*.
- **protected** l'accès est étendu (par rapport à `private`) au code des classes du même package **et** aux sous-classes de la classe. *Nous reviendrons plus tard sur cette notion, liée à l'héritage;*
- **public** accessible à partir de tout code qui a accès à la classe où l'élément est défini.

Contrôle d'accès aux classes

- modificateur **public** : la classe est visible à toute autre classe ;
- pas de modificateur : la classe est visible seulement dans son paquetage.

Modificateurs d'accès aux attributs

- Le premier modificateur (plus à gauche) permet de contrôler l'accès à l'attribut :

modificateur	**classe**	**paquetage**	**sous-classe**	**autre**
public	oui	oui	oui	oui
protected	oui	oui	oui	non
aucun	oui	oui	non	non
private	oui	non	non	non

Choix du niveau d'accès

- Pour respecter le principe **d'encapsulation**, il est préférable d'utiliser private.
- Si nécessaire, l'accès à l'attribut par une autre classe sera fait indirectement, par le biais des méthodes (comme ex. les méthodes accelerer et freiner de la classe Velo).
- Le passage travers une méthode permet une vérification de l'intégrité des données.

Attributs de classe

- Le modificateur static permet la création d'attributs de classe (aussi appelés attributs **statiques** et **variables de classe**).
- La valeur d'un attribut de classe est partagé par touts les objets de la classe.
- Tout objet de la classe peut changer sa valeur :

```
public class Velo {
...
private static int nbrVelos = 0;
...
public Velo() {
vitesse = 0;
++nbrVelos; }
}
```

Attributs de classe

- Les attributs de classe peuvent être manipulés sans la création d'un objet !
  - ++Velo.nbrVelo;
- Il est possible de se référer à un attribut de classe en utilisant le nom de l'objet. Mais cela n'est pas conseillé, car il n'est pas claire qu'il s'agit d'un tel type d'attribut.

Membres statiques

- Déclaration est précédée du modificateur static
- méthodes de classe : dont l'invocation peut être effectuée sans passer par l'envoi d'un message à l'une des instances de la classe.

Accès aux membres statiques :

- directement par leur nom dans le code de la classe où ils sont définis,
- en les préfixant du nom de la classe en dehors du code de la classe.
  - Exemple : **Math.PI Math.cos(x) Math.toRadians(90)** ...

Les méthodes

- Déclaration d'une méthode (dans l'ordre) :
  1. Modificateurs (optionnels)
  2. Type de retour
  3. Nom
  4. Liste de paramètres typés entre parenthèses
  5. Liste d'exceptions (optionnel, expliqué plus tard)
  6. Corps de déclaration entouré par { et }

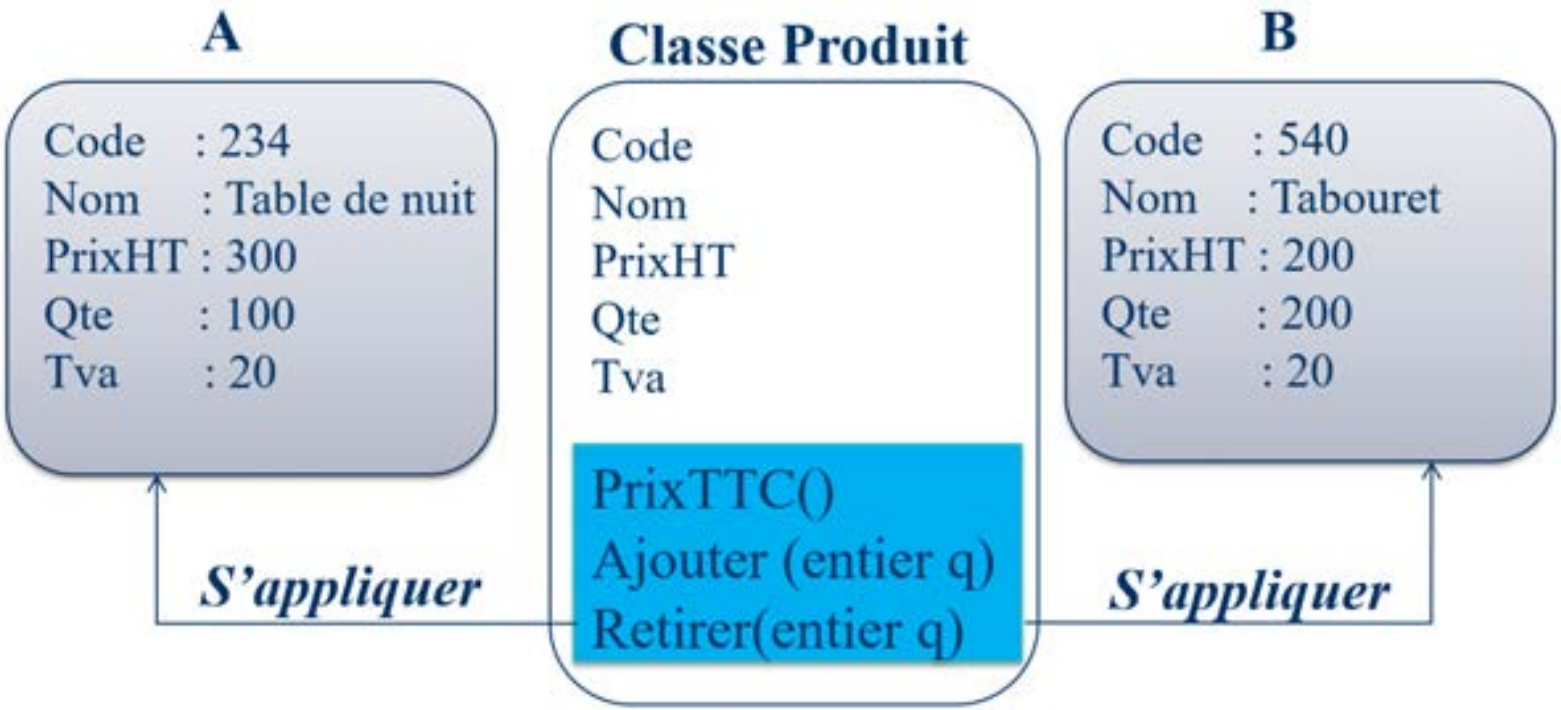
Exercice:

Créer la classe Produit

- Attributs :
  code
  nom
  prixHT
  qte
  tva
- Méthodes :
  PrixTTC () { return prixHT * (1+ tva/100); }
  AjouterAuStock (entier q) { qte = qte + q; }
  RetirerDuStock (entier q) { qte = qte - q; }

Exercice:

Créer deux objets (instances) A et B de la classe Produit, initialiser les avec les valeurs correspondantes et appliquer dessus les mthodes de la classe.



Surcharge de méthodes

Surcharge de méthodes

- Une méthode (y compris le constructeur) peut être définie plusieurs fois avec le même nom à condition de se différencier par le nombre et/ou le type des paramètres transmis (polymorphisme).
- Le compilateur décidera de la bonne méthode à utiliser en fonction des paramètres d'appel.

Exemples de surcharge de méthodes

Exemple de surcharge

```
class Velo
{
float vitesse ;

public void freiner (float decrement) { vitesse -= decrement ;}
public void freiner ()
{
vitesse = 0;
}

}
```

Types de variable

- Attributs : (ou **variables membres de classe**) variables définies dans une classe (à l'extérieur de ses méthodes).
  - Attribut d'instance : sa valeur est différente pour chaque instance de la classe (défaut).
  - Attribut de classe : sa valeur est la même pour toute instance de la classe.
- Variables locales : variables définies dans le corps de déclaration d'une méthode. (Visible seulement à l'intérieur de la méthode correspondante.)
- Paramètres : variables définies dans la liste des paramètres d'une méthode. (Visibles seulement à l'intérieur de la méthode correspondante.)

Accès aux attributs d'un objet

- pour accéder aux attributs d'un objet on utilise une notation pointée

  nomDeObjet.nomDeVariableDinstance

- similaire à celle utilisée en C pour l'accés aux champs d'une structure (Struct)

```
Velo v1;
v1 = new Velo();
Velo v2 = new Velo();
Velo v3 = v2;
v1.vitesse = 10;
v2.vitesse = 10;
v3.vitesse = v1.vitesse + v2.vitesse;
```

Envoi de messages (exemple)

- syntaxe :
  - **nomDeObjet.nomDeMethode(< paramètre effectifs >)**

```
class Velo {
double vitesse = 110.0;
void accelerer(double dx)
{
vitesse += dx;
}
void freiner(double dx){
vitesse -= dx;
}
} //Velo
```

dans le main

```
Velo v1 = new Velo();
Velo v2 = new Velo();

v1.accelerer(10.0);
v2.freiner(20.0);

System.out.println("vitesse de v1 est de "+v1.vitesse);
```

Exercice

- Ecrire une classe Compte contenant :
  - Attributs : Numéro du compte et solde
  - Méthodes : initialiser, deposer, retirer, consulterSolde et Afficher.
- Ecrire une classe Banque qui gére plusieurs objets de type Compte

Ecriture de la classe Compte

exercice2.java

```
class Compte {
int numero ;
float solde ;
void initialiser (int n, float s) { numero = n ; solde = s ; }
void deposer (float montant) { solde = solde + montant ; }
void retirer (float montant) { solde = solde - montant ; }
float consulterSolde ( ) { return solde ; }
void afficher()
{ System.out.println ("Compte : " + numero + " solde: " +
solde) ;
}
}
```

Utilisation de la classe Compte

Banque.java

```
public class Banque
{ static public void main (String args [])
{
Compte co1 = new Compte () ;
Compte co2 = new Compte () ;
co1.initialise (1234,1000f) ; co2.initialise (5678,500f) ;
co1.deposer (2100.95f) ; co1.afficher () ;
co2.retirer (1000.0f) ; co2.afficher () ;
}
}
```

Syntaxe de base

Syntaxe de base du langage Java

- Les commentaires existent sous plusieurs formes:
  - Commentaires multi lignes
    - /*
    - */
  - Commentaires sur une seule ou fraction de ligne
    - //
  - Commentaires destinés au générateurs de documentation javadoc
    - /**
    - *
    - *
    - */

Type de données prédéfinis

- **byte** -2^7, (2^7)-1 -128, 127
- **short** -2^{15}, (2^{15})-1 -32768, 32767
- **int** -2^{31}, (2^{31})-1 -2147483648, 2147483647
- **long** -2^{63}, (2^{63})-1 -9223372036854775808, 9223372036854775807
- Les entiers peuvent être exprimés en octal (0323), en décimale (311) ou en hexadécimal (0x137).

Type de données prédéfinis

- Nombres réels
  - float simple précision sur 32 bits 1.4032984e-45 3.40282347 e38
  - double précision sur 64 bits 4.9406564584124354 4 e-324 1.7976931348623157 0 e308
  - Repésentation des réels dans le standard IEEE 754. Un suffixe " **f** " ou " **d** " après une valeur numérique permet de spécifier le type.
    - Exemple

```
double x = 154.56d;
float y = 23.4f;
float f = 23.65; //Erreur
```

Type de données prédéfinis

- boolean
  - Valeurs true ou false
  - Un entier non nul est également assimilé à true
  - Un entier nul est assimilé à false
- char
  - Une variable de type char peut contenir un seul caractère codé sur 16 bits (jeu de caractères 16 bits Unicode contenant 34168 caractères)
  - Des caractères d'échappement existent
    - \b Backspace \t Tabulation
    - \n Line Feed \f Form Feed
    - \r Carriage Return \" Guillemet
    - \' Apostrophe \\ BackSlash
    - \u00xx Caractère Unicode (xx est compris entre 00 et FF)

Types primitifs

En Java, les types sont statiques (statically-typed language) et toute variable doit être déclarée avant son utilisation.

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 ne sont pas affectés aux variables locales !!

Déclaration et initialisation des variables

```
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;
```

Types de données prédéfinis

- Chaînes de caractères
  - Les chaînes de caractères sont manipulées par la classe **String** (ce n'est donc pas un type de données).
- Exemples :

```
String str = "exemple de chaîne de caractères" ;
String chaine = "Le soleil " + "brille" ;
// + est un Opérateur de concaténation
```

Chaînes de caractères

- Java fournit un support spécial aux chaînes des caractères.
- Exemple de déclaration :
  - **String s = "une chaîne de caractères";**
- La valeur initiale d'une variable du type String est null (ainsi que pour toutes les variables dont leur type est une classe d'objets).
- Mais attention!! Techniquement, String n'est pas un type primitif. Il s'agit d'une classe du paquetage java.lang.

Opérateurs

Par ordre de prioritée :

postfix	++ −−
unaires (prefixes)	++ −− + - ∼ !
multiplicatifs	* / %
additifs	+ -
décalage	>> << >>>
relationnels	< > <= >= instanceof
égalité	== !=
bitwise AND	&
bitwise excl. OR	^
bitwise incl. OR	|
conjonction	&&
disjonction	||
conditionnel	? :
affectation	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Opérateurs (2)

Quelques remarques :

- Opérateur $++$ (resp.$--$) préfixé évalue l'expression avant l'incrementation (resp. décrementation)
- Opérateurs && et || présentent le "short circuit behaviour" : le deuxième opérande est évalué seulement si nécessaire.
- Opérateur $+$ est utilisé aussi pour la concatenation des Strings.

Opérateurs (2)

Quelques remarques :

- Opérateur ++ (resp. -

  –

) préfixé évalue l'expression avant l'incrementation (resp. décrementation)
- Opérateurs && et || présentent le "short circuit behaviour" : le deuxième opérande est évalué seulement si nécessaire.
- Opérateur + est utilisé aussi pour la concatenation des Strings.

Commandes basiques

- Affectation :

  Variable = Expression ;

- Bloc de commande :

  { instruction ; instruction ; ... ; }

- Contrôle de flux :
  - if (Condition) instruction
  - if (Condition) instruction else instruction
  - switch (variable) {
    case valeur1 : instruction1
    case valeur2 : instruction2
    ... [default : instructionD] }
  - while (Condition) instruction
  - do instruction while (Condition)
  - for (instruction ; [Condition] ; [instruction]) instructions

Commandes basiques (2)

- Commandes de ramification :

  break [Identificateur] ;
  continue [Identificateur] ;
  return [Expression] ;

La classe de base

- Toute classe java hérite implicitement de la classe java.lang.Object.
- Quelques méthodes de la classe java.lang.Object:
  - public boolean **equals**(Object obj) ;
  - public String **toString()** ;

Comparaison d'objets

- On ne peut pas comparer 2 objets en comparant les variables d'instance.
  - Exemple 1 :
    - r1 = new Rectangle (10,20) ;
    - r2 = new Rectangle (30,40) ;
    - r3 = new Rectangle (10,20) ; Comparaison des variables d'instance:
    - r1 == r2 $\rightarrow$ false
    - r1 == r3 $\rightarrow$ false
- L'opérateur (==) appliqué á deux objets compare leurs références.

Comparaison d'objets

- Suite Exemple 1 :
  - Comparaison avec une méthode **equals** incluse dans la classe Rectangle
  - r1.equals (r2) $\rightarrow$ false
  - r1.equals (r3) $\rightarrow$ true

Redéfinition de la méthode **equals** dans la class Velo

```
Class Velo {
...
boolean equals( Velo v ){
if( v == null) return false;
else
return (vitesse == v.vitesse && couleur == v.couleur);
} }
```

Exemple 2:

Comparaison de chaînes de caractères:
String s1 = "Bonjour" ;
String s2 = "Bonjour" ;

if (s1.equals (s2)) *// Compare le contenu de s1 et s2.*

if (s1.equalsIgnoreCase (s2)) *// Compare le contenu de s1 et s2*
// sans tenir compte des majuscules
// et minuscules.

Affichage d'objets

- L'instruction : **System.out. println(velo1)** affiche la rfrence de l'objet vlo1.
- Afin d'afficher l'tat interne d'un objet on redfini la mthode **toString()**.

Exemple :

```
class Velo { ...
public String toString() {
return "La vitesse du vlo est: "+vitesse+"couleur du vlo est:
"+couleur;
}
...
}
```

Les tableaux

- Les tableaux peuvent être déclarés suivant la syntaxe suivante :
  - type [ ] nom ;
- Exemples :
  - int[ ] table;
  - double [ ] d1,d2 ;
- Pas de tableau statique.
- La taille d'un tableau est allouée dynamiquement par l'opérateur new

$$\text{table} = \text{new int } [10] ;$$
$$\text{int[ ] table2} = \text{new int } [20] ;$$
$$\text{int[ ] table3} = \{1,2,3,4,5\} ;$$

Les tableaux

- Tableau de deux dimensions :
  - int [ ] [ ] Matrice = new int [5][4] ; //5 lignes et 4 colonnes
- La taille n'est pas modifiable et peut être consultée par la propriété length
  - System.out.println (table3.length) ;
  - System.out.println (Matrice.length) ; // 1ère dimension
  - System.out.println (Matrice[0].length) ; // 2ème dimension

Tableaux (Arrays)

DemoTableau.java

```
class DemoTableau {
public static void main(String[ ] args) {
int[ ] unTableau; // declaration
unTableau = new int[3]; // allocation de memoire
unTableau[0] = 100; // initialisation
unTableau[1] = 200;
unTableau[2] = 300;
System.out.println(" Element 0 : " + unTableau[0]);
System.out.println(" Element 1 : " + unTableau[1]);
System.out.println(" Element 2 : " + unTableau[2]);
}
}
```

Tableaux multidimensionnels

DemoMultiTableau.java

```
class DemoMultiTableau {
public static void main(String[] args) {

String[][] noms = {{"Mr. ", "Mrs. ", "Ms. "},
                   {"Smith", "Jones"}};


System.out.println(noms[0][0] + noms[1][0]);
System.out.println(noms[0][2] + noms[1][1]);
}
}
```

Tableaux multidimensionnels

DemoMultiTableau.java

```
class DemoMultiTableau {
public static void main(String[] args) {

String[][] noms = {{"Mr. ", "Mrs. ", "Ms. "},
                   {"Smith", "Jones"}};


System.out.println(noms[0][0] + noms[1][0]);
System.out.println(noms[0][2] + noms[1][1]);
}
}
```

$ **javac** DemoMultiTableau.java
$ **java** DemoMultiTableau
Mr. Smith
Ms. Jones

Pour copier un tableau

Définition de la méthode (dans la classe System) :

public static void arraycopy (Object src, int posSrc, Object dest, int posDest int longueur)

Exemple d'utilisation :

DemoArrayCopy.java

```
class DemoArrayCopy {
public static void main(String[] args) {
char[] source = { 'd', 'e', 'c', 'a', 'f', 'e', 'i', 'n', 'e'};
char[] destin = new char[4];
System.arraycopy(source, 2, destin, 0, 4);
System.out.println(new String(destin));
}
}
```