المدرسة العليا للتكنولوجيا - الداخلة
ⵜⵜⵉⵛⵏ ⵜⴰⵙⴷⴰⵡⵜ ⵏ ⵜⵜⴽⵏⵓⵍⵓⵊⵢⴰ - ⴷⴰⵅⵍⴰ
ÉCOLE SUPÉRIEURE DE TECHNOLOGIE - DAKHLA

Programmation objet avancée

2ème année GI

Pr. Naoual Nassiri

N.Nassiri@uiz.ac.ma

Année universitaire 2023-2024

Les classes Java

- En Java, la déclaration et la définition d'une classe sont faites en même temps contrairement à C++ où la déclaration de la classe se fait dans un «fichier.h » alors que la définition de la classe se fait dans un « fichier.cpp ».
- Java est un langage purement orienté objet. En effet :
  - Tout est objet, sauf les types primitifs (entiers, flottants, booléens, ...).
  - L'entité de base de tout code Java est la classe : C'est-à-dire qu'en Java, tout se trouve dans une classe.
  - Il ne peut y avoir de déclarations ou de code en dehors du corps d'une classe, le code se trouve dans le corps des méthodes.
  - La classe contient : des attributs et des méthodes qui peuvent aussi contenir des déclarations de variables locales (visibles uniquement dans le corps de la méthode).
  - Une classe permet de créer les objets : Classe = Modèle d'objets

Les classes Java

Exemple: Considérons la classe « ***Rectangle*** » qui est définie pour modéliser les objets représentant des rectangles. Un objet de type «***Rectangle***» est caractérisé par:

- Son état interne (valeurs des attributs)
    - La longueur et la largeur de ses cotés.
    - Sa position dans le plan: par la position de son centre dans le plan.
- Son comportement (les opération qu'on peut effectuer sur un objet de type rectangle) :
    - Calcul de la surface
    - Déplacement dans le plan

Déclaration d'une classe Java

- Pour déclarer une classe (un nouveau type d'objets) on utilise le mot clé **class** suivi du nom de la classe.
- La syntaxe pour créer une classe de nom « **ClasseTest** » est la suivante:

```
class ClasseTest{ /* ClasseTest est le nom de la classe à créer */
/* Le corps de la classe comprend :
- la définition des attributs (données membres)
- la définition des méthodes et des constructeurs. */
}
```

Déclaration d'une classe Java

Pour les commentaire on utilise:

// pour un commentaire sur une seule ligne

/* pour un commentaire étalé sur
plusieurs lignes. Il doit terminer avec */

Attributs d'une classe

- Les attributs (les variables), appelés aussi champs ou données membres, définissent l'état interne d'un objet (les données propre à chaque objet*)*.
- Tous les attributs doivent être déclarés avant d'être utilisés. La déclaration des attributs se fait en début ou en fin de classe (mais de préférence en début de classe).

Syntaxe:

modificateur type nomAttribut;

- modificateur : un spécificateur d'accès.
- nomAttribut : le nom de l'attribut (le nom de la variable).
- type : le type de l'attribut. Il est de type primitif (char, byte, short, int, long, float, double, boolean) ou un nom d'une classe.

Attributs d'une classe

Exemple :

private double z ; // nom de l'attribut : z, le type : double ; le modificateur : private

public Rectangle r ; // nom de l'attribut : r, le type : Rectangle ; le modificateur : public

Attributs d'une classe

Nous supposons qu'un étudiant est caractérisé par son nom, son prénom, son cne et sa moyenne.

```
class Etudiant {
    private String nom;
    private String prenom;
    private String cne;
    private double moyenne
}
```

Par convention, les noms des attributs et des méthodes doivent être en minuscule. Le premier mot doit commencer en minuscule et les autres mots doivent commencer en majuscule (**ceciEstUneVariable**, **ceciEstUneMethode**).

Les méthodes d'une classe

- Les méthodes (appelées aussi fonctions membres ou comportement) définissent l'ensemble d'opérations applicables à l'objet : on manipule les objets par des appels de ces méthodes.
- La syntaxe pour définir une méthode est identique à celle utilisée en C pour définir les fonctions.

Syntaxe:

```
modificateur typeRetour nomMethode ( listeParamètres ) {
// corps de la méthode: les instructions décrivant la méthode
}
```

Les méthodes d'une classe

- modificateur : un spécificateur d'accès.
- nomMethode : le nom de la méthode
- listeParamètres : la liste (éventuellement vide) des paramètres (arguments). On spécifie les types et les noms des informations qu'on souhaite passer à la méthode lors de son appel.
- typeRetour : c'est le type de la valeur qui sera retournée par la méthode après son appel. Si la méthode ne fournit aucun résultat, alors typeRetour est remplacé par le mot clé void. Dans le cas où la méthode retourne une valeur, la valeur retournée par la fonction doit être spécifiée dans le corps de la méthode par l'instruction de retour:

```
return expression;
```

ou

```
return; // possible uniquement pour une fonction de type void
```

Les méthodes d'une classe

Définition : L'ensemble des méthodes « public » est appelé l'interface de l'objet. Une interface définit toutes les opérations qu'on peut appliquer à l'objet.

Signature d'une méthode en java

- La signature d'une méthode permet d'identifier de manière unique une méthode au sein d'une classe.
- En java, la signature d'une méthode est l'ensemble composé de :
  - nom de la méthode
  - l'ensemble des types de ses paramètres.
- Mais attention, en java, le type de retour ne fait pas partie de la signature de la méthode.

Les méthodes d'une classe

Passage des paramètres

Le mode de passage des paramètres dans les méthodes dépend de la nature des paramètres :

- **par valeur** pour les types primitifs.
- **par valeur des références** pour les objets: la référence est passée par valeur (i.e. le paramètre est une copie de la référence), mais le contenu de l'objet référencé peut être modifié par la méthode (car la copie de référence pointe vers le même objet).

Surcharge des méthodes

- On parle de surcharge (en anglais overloading) lorsqu'un même symbole possède plusieurs significations. Le choix entre ces symboles se fait en fonction du contexte. Par exemple le symbole + dans l'instruction «a+b» dépend du type des variables a et b. En Java :
  - On ne peut pas surcharger les opérateurs.
  - On peut surcharger une méthode, c'est-à-dire on peut définir, dans la même classe, plusieurs méthodes qui ont le même nom mais pas la même signature. Elles sont différenciées par le nombre et/ou les type des arguments.

Surcharge des méthodes

Exemple : Considérons l'exemple suivant ou la classe « **ClasseTest** » est dotée de 2 méthodes « calcMoyenne() »:
- La première a deux arguments de type double. Elle calcule la moyenne de deux valeurs.
- La deuxième a trois arguments de type double. Elle calcule la moyenne de trois valeurs.

```
class ClasseTest{

public double calcMoyenne(double m1, float m2) {
return (m1+m2)/2;
}
```

Surcharge des méthodes

```
public float calcMoyenne(double m1, double m2, double m3) { /* méthode surchargée*/
return (m1+m2+m3)/3;
}

}

public class TestSurcharge {
public static void main(String[] args) {
ClasseTest objA;
…
System.out.println(“Moy1="+objA.calcMoyenne(10.1, 12.25));
System.out.println(“Moy2="+objA.calcMoyenne(13.5, 12.6, 8.5));
} }
```

Création et manipulation d'Objets

- Une fois la classe est définie, on peut l'utiliser pour créer des objets (variables). Donc chaque objet est une variable d'une classe qui a son espace mémoire pour stocker les valeurs de ses attributs.
- Les valeurs des attributs sont propres à l'objet. L'ensemble des valeurs des attributs d'un objet à un instant donné est appelé état interne de l'objet.
- L'opération de création de l'objet est appelée une instanciation**.** Un objet (appelé aussi une instance d'une classe) est référencé par une variable ayant un état (une valeur). On parle indifféremment d'instance, de référence ou d'objet.
- Une classe permet d'instancier plusieurs objets:
  - les noms des attributs et les méthodes sont les mêmes pour toutes les instances (objets) de la classe.
  - les valeurs des attributs sont propres à chaque objet.

Etapes de création des Objets

Contrairement aux types primitifs, la création d'objets se passe en deux étapes:

1. déclaration de l'objet.

2. création de l'objet.

Déclaration d'un objet

- Chaque objet doit être déclaré avec son type.

Syntaxe de déclaration:

NomDeClasse idObjet1, idObjet2, … ;

- **NomDeClasse** : désigne le nom de la class.
- **idObjet1, idObjet2, …** : désignent les noms des variables

Attention:

- La déclaration d'une variable de type primitif réserve un emplacement mémoire pour stocker la valeur de la variable.
- Par contre, la déclaration d'un objet, ne réserve pas un emplacement mémoire pour l'objet, mais seulement un emplacement pour mémoriser une référence à cet objet.

➔ les objets sont manipulés avec des références

Déclaration d'un objet

Exemple: Considérons la classe « **ClasseTest** »

```
class ClasseTest {
// Corps de la classe ClasseTest
}
```

l'instruction: **ClasseTest obj;**

- déclare que « **obj** » est une variable de type « **ClasseTest** ».
- **« obj »** est une variable pour référencer un objet de type « **ClasseTest ».**

Autrement dit, **obj** est une référence à un objet de la classe « **ClasseTest ».**

Aucun objet n'est créé, « **obj** » est un objet seulement déclaré, il vaut «null».

Important:

- la déclaration seule d'un objet, ne nous permet pas de l'utiliser.
- avant d'utiliser un objet on doit le créer.

Création d'un objet

- Après la déclaration d'un objet, on doit le créer: réserver un emplacement mémoire pour stocker l'état interne de l'objet correspondant.
- La création doit être demandée explicitement dans le programme en faisant appel à l'opérateur « new » qui permet de réserver de la place mémoire pour stocker l'objet et initialiser les attributs.

Syntaxe :

```
NomDeClasse idObjet;
idObjet = new nomDeClasse();
```

- Les deux expressions peuvent être remplacées par :

```
NomDeClasse idObjet = new NomDeClasse();
```

- l'expression « ***new*** **NomDeClasse()** » crée un emplacement mémoire pour stocker l'état interne de l'objet ***idObjet***.

Création d'un objet

Remarque:

- Chaque objet met ses données membres dans sa propre zone mémoire.
- En général, les données membres ne sont partagées entre les objets de la même classe.

Création d'un objet

Exemple 1 :

```
class ClasseTest {
/* Corps de la classe ClasseTest */
}
```

- ClassTest objA ; /* déclare une référence pour l'objet objA */
- objA= new ClasseTest(); /* créé un emplacement pour stocker l'objet objA */
- On peu regrouper les deux expressions en une seule expression

```
ClassTest objA= new ClasseTest();
```

Création d'un objet

Exemple2 : Considérons la classe rectangle

```
public class Rectangle{
int longueur;
int largeur;
int x;
int y;
public static void main (String args[]) {
Rectangle rectangleR1; /* déclare une référence sur l'objet rectangleR1 */
rectangleR1 = new Rectangle(); /* création de l'objet rectangleR1 */
// rectangle R1 est une instance de la classe Rectangle
// Les deux expressions peuvent être remplacées par :
// Rectangle rectangleR1=new Rectangle();
}
}
```

Création d'un objet

Lors de la création d'un objet, tous les attributs sont initialisés par défaut. Dans les sections suivantes on verra comment initialiser les attributs lors de la création d'un objet.

Type	boolean	char	byte	short	int	long
valeur par défaut	false	`'\u0000'`	(byte)0	(short)0	0	0L

Type	float	double	objet
valeur par défaut	0.0f	0.0	null

Accès aux attributs

- Une fois l'objet est créé, on peut accéder à ses attributs (données membres) en indiquant le nom de l'objet suivi par un point, suivi par le nom de l'attribut comme suit:

  nomObjet.nomAttibut

- nomObjet = nom de la référence à l'objet (nom de l'objet)
- nomAttribut =nom de la donnée membre (nom de l'attribut).

Accès aux attributs

Exemple: Considérons la classe « **ClassTest** » suivante:

```
class ClasseTest {
double x;
boolean b;
}
```

- ClasseTest obj = new ClasseTest(); /* crée un objet de référence obj (crée un objet : obj) */
- Les instructions :
  - obj.b=true; // affecte true à l'attribut « b » de l'objet obj.
  - obj.x=2.3; // affecte le réel 2.3 à l'attribut « x » de l'objet obj.

Accès aux méthodes: appels des méthodes

- Une méthode ne peut être appelée que pour un objet.
- L'appel d'une méthode pour un objet se réalise en indiquant le nom de l'objet suivi d'un point, suivi du nom de la méthode et de sa liste d'arguments:

  nomObjet.nomMethode(listeArguments).
- nomObjet: nom de l'objet
- nomMethode: nom de la méthode.
- listeArguments : liste des arguments

Accès aux méthodes: appels des méthodes

Exemple: Considérons la classe **Rectangle** dotée de la méthode «initialise()» qui permet d'affecter des valeurs à l'origine (aux attributs x et y).

```
public class Rectangle{
int longueur, largeur;
int x, y;
void initialise_origine(int x0, int y0) {
x=x0;
y=y0;
}
```

Accès aux méthodes: appels des méthodes

```
public static void main (String args[]) {
Rectangle r1;
r1.longueur=4; r1.largeur=2;
r1. initialise_origine(0,0); // affecte 0 aux attribut x et y de l'objet r1
}
}
```

- Le programme ne compile pas. En effet l'objet r1 est seulement déclaré. Il n'est pas encore créé. Avant de l'utiliser, il faut tout d'abord le créer. On doit rajouter l'instruction de création de l'objet r1.

Accès aux méthodes: appels des méthodes

```
public static void main (String args[]) {
Rectangle r1=new Rectangle ();
r1.longueur=4; r1.largeur=2;
r1. initialise_origine(0,0); // affecte 0 aux attribut x et y de l'objet r1
}
```

Concept d'encapsulation

- En Java, l'encapsulation est assurée par les modificateurs d'accès (spécificateurs d'accès) permettant de préciser 4 niveaux de visibilité des membres de la classe. Ces 4 niveaux de visibilité sont : **public, private, par défaut et protected.** Ils définissent les droits d'accès aux données suivant le type d'accès:
  - Membres publiques (membres précédés par le modificateur «**public**»): correspond à la partie visible de l'objet depuis l'extérieur : c'est-à-dire que cette partie est directement utilisable depuis l'extérieure (depuis une classe quelconque).
  - Membres privés (membres précédés par le modificateur «**private**»): correspond à la partie non visible de l'objet : c'est à dire que cette partie n'est pas directement accessible depuis l'extérieur (depuis une autre classe), elle n'est accessible que depuis l'intérieur de la classe elle même.

Concept d'encapsulation

- Accès par défaut: lorsqu'aucun spécificateur d'accès n'est mentionné devant un membre d'une classe C, on dit que ce membre a l'accessibilité par défaut. Ce membre n'est accessible que depuis la classe C et depuis les autres classes du même paquetage que la classe C.
- Membres protégés: un membre d'une classe C dont la déclaration est précédé par le modificateur **protected** se comporte comme **private** avec moins de restriction: il n'est accessible que depuis:
  - ✓ la classe C,
  - ✓ les classes du même paquetage que la classe C
  - ✓ et les sous-classes, directes ou indirectes, de C.

Attention: une sous-classe a un accès direct aux membres **protected** mais pas aux membres **private**.

Concept d'encapsulation

Exemple 1: Accès pour modification d'un attribut depuis l'extérieur.

```
class ClasseTest {
public int x;
private int y;
public void initialise (int i, int j){
x=i;
y=j; // Valide: accès de puis l'intérieur à l'attribut private y
}
}
```

Concept d'encapsulation

```
public class TestA{ // fichier source de nom TestA.java
public static void main (String args[]) {
ClasseTest objA = new ClasseTest();
objA.initialise(1,3); /* valide : accès direct depuis une autre classe à une méthode public */
objA.x=2; /* Valide: accès direct depuis l'extérieur à un attribut public x. la valeur de x de l'objet objA devienne 2 */
objA.y=3; /* Non valide : accès direct depuis une classe externe à un attribut déclaré private */
}
}
```

Concept d'encapsulation

Exemple2: Accès pour consultation d'un attribut depuis l'extérieur: affichage du contenu d'un attribut.

```
class ClasseTest {
public int x;
private int y;
public void initialise (int i, int j){
x=i;
y=j; // Valide: accès de puis l'intérieur à l'attribut private y
}
}
```

Concept d'encapsulation

```
public class TestA{ // fichier source de nom TestA.java
public static void main (String args[]) {
ClasseTest objA = new ClasseTest();
objA.initialise(1,3); /* Valide : car la méthode « initialise() » est déclarée
public*/
System.out.println(" x= "+objA.x); // Valide : l'attribut « x » déclaré public.
System.out.println(" y= "+objA.y); /* Non valide : accès depuis l'extérieur
à l'attribut private y*/
}
}
```

Méthodes d'accès aux valeurs des variables depuis une classes externe

Comment peut-on accéder à la valeur d'une variable protégée ?

Exemple: Considérons la classe « Etudiant » qui a les champs « nom » et «prenom » privés:

```
class Etudiant {
private String nom, prenom;
public initialise(String st1, String st2){
nom=st1; prenom=st2;
}
}
```

Méthodes d'accès aux valeurs des variables depuis une classes externe

```
public class TestEtudiant{
public static void main(String[] argv) {
Etudiant e= new Etudiant();
e. initialise("Mohammed","Ali");
System.out.println("Nom = "+e.nom);
// ne compile pas car l'attribut nom est privé
}
}
```

Méthodes d'accès aux valeurs des variables depuis une classes externe

- Accès en lecture à la valeur d'un attribut protégé (private) :
  - Pour accéder en lecture (par exemple afficher) la valeur d'un attribut protégé (private), on définit ***Un accesseur*** (***un getter***): c'est une méthode qui permet d'accéder en lecture au contenu de l'attribut, c'est à-dire, lire depuis une classe externe, le contenu d'une donnée d'un attribut protégée.
  - Généralement on lui donne comme nom : getNomAttribut() ;

Méthodes d'accès aux valeurs des variables depuis une classes externe

Exemple:

```
class Etudiant {
private String nom, prenom;
public initiale(String nom, String Prenom){
this.nom=nom; this.prenom=prenom;
}
public String getNom (){ // accesseur qui retourne le contenu de l'attribut « nom »
return nom;
}
public String getPrenom (){ /* accesseur qui retourne le contenu de l'attribut
«prenom»*/
return prenom;
}
}
```

Méthodes d'accès aux valeurs des variables depuis une classes externe

```
public class TestEtudiant{
public static void main(String[] argv) {
Etudiant e= new Etudiant();
e.initialise("Mohammed","Ali");
System.out.println("Nom = "+e.getNom());
// e.getNom() retourne la valeur de l'attribut « nom » de l'objet « e ».
System.out.println("Prenom = "+e.getPrenom());
// e.getNom() retourne la valeur de l'attribut « prenom » de l'objet « e ».
}
}
```

Méthodes d'accès aux valeurs des variables depuis une classes externe

Accès en écriture à un attribut protégé (private) : Comment peut-on modifier (depuis l'extérieur) le contenu d'un attribut privé ?

Exemple:

```
class Etudiant {  private int cne;  }

public class TestEtudiant{
public static void main(String[] argv) {
Etudiant e= new Etudiant();
// Modifier le champs cne (attribut private)
e.cne=23541654;
// ne compile pas car le champ cne est un champ protégé (private)
} }
```

Méthodes d'accès aux valeurs des variables depuis une classes externe

- Pour accéder en écriture (modifier) la valeur d'un attribut protégé (private), on définie ***Un modificateur (mutateur ou setter)***: c'est une méthode qui permet d'accéder en écriture à l'attribut, c'est-à-dire modifier, depuis une classe externe, le contenu d'un attribut privé.
- Généralement on lui donne comme nom : setNomAttribut() ;

Méthodes d'accès aux valeurs des variables depuis une classes externe

Exemple:

```
class Etudiant {
private int cne;
public void setCNE (int cne){
this.cne=cne;
}
}

public class TestEtudiant{
public static void main(String[] args) {
Etudiant e= new Etudiant();
e.setCNT(23541654);
}
}
```

Autoréférences: emploi de this

- Au sein d'une méthode, on peut utiliser le mot clé « **this** », pour désigner l'instance courante (l'objet courant) c'est pour faire référence à l'objet qui a appelé cette méthode.

```
public void f(….) {
// l'emploi de this , dans ce bloc, désigne la référence à
l'objet ayant appelé la méthode f()
}
```

Exemple:

```
class Etudiant {
private String nom, prenom;
public initialise(String st1, String st2) {
nom = st1;
prenom = st2;
} }
```

Autoréférences: emploi de this

- Comme les identificateurs st1 et st2 sont des arguments muets pour la méthode « initialise() », alors on peut les noter nom et prenom qui n'ont aucune relation avec les attributs private « nom » et « prenom ».
- Dans ce cas la classe « Etudiant » peut s'écrire comme suit:

```
class Etudiant {
private String nom, prenom;
public initialise(String nom, String prenom){
this.nom=nom;
this.prenom=prenom;
}
}
```

Autoréférences: emploi de this

- « this.nom » désigne l'attribut « nom » de l'objet qui appelle la méthode « initialise() »
- « this.prenom » désigne l'attribut «prenom» de l'objet qui appelle la méthode « initialise() »

Champs de classe (variable de classe)

- Considérons la classe suivante :

```
class ClasseTest {
int n;
double x;
}
```

- Chaque objet de type « **ClasseTest** » possède son propre état interne (ses propres données) pour les champs n et x.

➔ Les valeurs des champs ne sont pas partagées entre les objets.

Champs de classe (variable de classe)

- Si on créé deux objets différents instances de la classe « **ClassTest** », leurs états internes sont différents

  ClasseTest objA1=new **ClasseTest**();

  ClasseTest objA2= new **ClasseTest**();
- objA1.n et objA2.n désignent deux données différentes.
- objA1.x et objA2.x désignent aussi deux données différentes.
- Certains attributs peuvent être partagés par toutes les instances d'une classe. C'est-à-dire ils peuvent être définis indépendamment des instances (objets).
- Par exemple: le nombre d'étudiants = le nombre d'objets étudiants créés.

Champs de classe (variable de classe)

- Les attributs qui peuvent être partagés entre tous les objets sont nommés champs de classe ou variables de classe. Ils sont comparables aux «variables globales ». Ces variables n'existent qu'en un seul exemplaire.
- Elles sont définies comme les attributs mais précédés du mot-clé **static**.

- **Exemple:** Considérons la classe:

```
class ClasseTest {
static int n; // la valeur de n est partagée par toutes les instances
double y;
}

ClasseTest objA1=new ClasseTest(),
ClasseTest objA2=new ClasseTest();
```

Champs de classe (variable de classe)

- Ces déclarations créées deux objets (instances) différents: objA1 et objA2.
- Puisque « n » est un attribut précédé du modificateur **static**, alors il est partagé par tous les objets en particulier, les objets objA1 et objA2 partagent la même variable n.
  - objA1.n et objA2.n désignent la même donnée.
  - La valeur de l'attribut « n » est indépendante de l'instance (l'objet).
- Pour accéder au champ statique « n », on utilise le nom de la classe : **ClasseTest**.n // champs (statique) de la classe **ClasseTest**

Champs de classe (variable de classe)

Exemple1:

```
class ClasseTest {
int n;
double y;
}
public class MethodeStatic{
public static void main(String[] argv) {
ClasseTest objA1=new ClasseTest();
objA1.n+=4;
ClasseTest objA2=new ClasseTest();
// objA2.n = ?
/* la valeur de l'attribut n de l'objet objA2 vaut 0.
initialisation par défauts des attributs. */
}
}
```

Champs de classe (variable de classe)

Exemple2:

```
class ClasseTest {
static int n; // la valeur de n est partagée par toutes les instances
double y;
}
public class MethodeStatic{
public static void main(String[] argv) {
ClasseTest objA1=new ClasseTest();
objA1.n+=4; // objA1.n vaut 4;
// équivalent à ClasseTest.n=4;
ClasseTest objA2=new ClasseTest();
// objA2.n vaut 4 car « n » est champs de classe (champ statique).
}
}
```

Méthodes de classe

- Ce sont des méthodes qui ont un rôle indépendant d'un objet spécifique. Elles exécutent une action indépendante d'une instance particulière de la classe La déclaration d'une méthode de classe se fait à l'aide du mot clé static. L'appelle d'une telle méthode ne nécessite que le nom de la classe correspondante.

Important: Une méthode de classe ne pourra pas accéder à des champs usuels (champs non statiques).

Méthodes de classe

Exemple:

```
class ClasseTest{
private static int n; // champs de classe
private float x; // champs usuel
public static void f() { // méthode de clase
/* ici (dans le corps de la méthode f(), on ne pas accéder au champ x,
champs usuel. Par contre on peut accéder au champ statique n.
*/
}
}
```

Le mot clé final

- L'attribut ***final*** indique que la valeur de la variable ne peut être modifiée : on pourra lui donner une valeur une seule fois dans le programme.

Variable d'instance final

- Une variable *d'instance* ***final*** (attribut déclaré final) est une constante pour chaque objet.

Remarques:

- une variable *d'instance* ***final*** peut avoir 2 valeurs différentes pour 2 objets différents.
- une variable d'instance ***final*** peut ne pas être initialisée à sa déclaration mais elle doit avoir une valeur à la sortie de tous les constructeurs.

Le mot clé final

Variable locale final

- Considérons une variable locale précédé par le mot clé *final*.
  - Si la variable est d'un type primitif, sa valeur ne pourra pas changer.
  - Si la variable référence un objet, elle ne pourra pas référencer un autre objet mais l'état de l'objet peut être modifié

Le mot clé final

Exemple:

```
final Etudiant e = new Etudiant();
e.initialise("Mohammed", "Ali");
e.nom = "Ahmed"; // ok, changement de l'état de l'objet e
e.setCNE(32211452); // ok, changement de l'état de l'objet e
e = new Etudiant(); /* Non valide car ceci indique que « e » référence un autre objet */
```

Constantes de classe

Usage

- Ce sont des variables de classes déclarées avec le mot-clé final, ce sont des constantes liées à une classe.
- Elles sont écrites en MAJUSCULES.
- Pour y accéder, il faut utiliser le nom de la classe et non un identificateur d'objet.

Exemple:

```
public class Etudiant {
public static final int TB= 15; // une moyenne supérieur 15 donne mention très bien
…..
}
if (e.getMoy() >= Etudiant.TB ) // e est un objet de type « Etudiant ».
System.out.println(“ Mention Très bien”); }
```

Lecture à partir du clavier

Pour lire à partir du clavier, il existe la classe **Scanner**. Pour utiliser cette classe, il faut la rendre visible au compilateur en l'important en ajoutant la ligne :

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

nextShort()	permet de lire un **short**
nextByte()	permet de lire un **byte**
nextInt()	permet de lire un **int**
nextLong()	permet de lire un **long**. Il n'est pas nécessaire d'ajouter L après l'entier saisi.
nextFloat()	permet de lire un **float**. Il n'est pas nécessaire d'ajouter F après le réel saisi.
nextDouble()	permet de lire un **double**
nextLine()	permet de lire une ligne et la retourne comme un **String**
next()	permet de lire la donnée suivante comme **String**

Exemple :

```
import java.util.Scanner;
public class TestScanner
{
    public static void main(String[] args)
    {
        String nom;
        int age;
        double note1,note2,moyenne;

        /*clavier est un objet qui va permettre la
           saisie clavier
        vous pouvez utiliser un autre nom (keyb,
           input, ...)*/
        Scanner clavier = new Scanner(System.in);
```

Exemple :

```
        System.out.print("Saisir votre nom : ");
        nom = clavier.nextLine();
        System.out.print("Saisir votre age : ");
        age = clavier.nextInt();
        System.out.print("Saisir vos notes : ");
        note1 = clavier.nextDouble();
        note2 = clavier.nextDouble();
        moyenne = (note1+note2)/2;
        System.out.println("Votre nom est " + nom +
            ", vous avez " + age + " ans et vous avez
             obtenu "+ moyenne);
        clavier.close(); //fermer le Scanner
    }
}
```

Problèmes liées à nextLine()

Reprenons l'exemple précédent et au lieu de commencer par la saisie du nom, on commence par la saisie de l'age.

```
Scanner clavier = new Scanner(System.in);
System.out.print("Saisir votre age : ");
age = clavier.nextInt();
System.out.print("Saisir votre nom : ");
nom = clavier.nextLine();
System.out.print("Saisir vos notes : ");
note1 = clavier.nextDouble();
note2 = clavier.nextDouble();
moyenne = (note1+note2)/2;
System.out.println("Votre nom est " + nom +
    ", vous avez " + age + " ans et vous avez
     obtenu "+ moyenne);
```

Problèmes liées à nextLine()

L'exécution du précédent programme est la suivante :

Saisir votre age : 23

Saisir votre nom : Saisir vos notes : 12

13

Votre nom est , vous avez 23 ans et vous avez obtenu

12.5

Lors de la saisie de l'age, on a validé par Entrée . La touche Entrée a été stocké dans nom!

Problèmes liées à nextLine()

Pour éviter ce problème, il faut mettre clavier.nextLine() avant nom = clavier.nextLine() ;.

```
public class TestScanner
{
    public static void main(String[] args)
    {
        ...
        age = clavier.nextInt();
        System.out.print("Saisir votre nom : ");
        clavier.nextLine();
        nom = clavier.nextLine();
        ...
    }
}
```