Chapitre 2: Les classes et les objets

HANENE CHETTAOUI HAMROUNI

MAÎTRE ASSISTANTE EN INFORMATIQUE

Éléments de base

Commentaires

// commentaire sur une seule ligne

/ commentaire sur une ou plusieurs lignes */*

Point virgule, blocs et blancs

- Les instructions se terminent par un point virgule (*;*)
- Un bloc d'instructions est délimité par des accolades *{* et *}*
- Les blancs sont autorisés entre les éléments du code source (espaces, tabulations et sauts de ligne)

Déclaration d'une classe

```
[public] [abstract | final] class NomClasse [extends NomSuperClasse] [implements NomInterface1, Interface2,....., InterfaceN]
{
    // déclaration des attributs
    // déclaration des méthodes
}
```

Déclaration d'une classe

public : un modificateur d'accès précisant la possibilité d'utilisation de la classe: (**public** donne la possibilité d'utiliser la classe à toutes les autres classes.)

Par défaut(le mot class n'est pas précédé de mot particulier), toutes les classes du package peuvent utiliser la classe.

abstract et final : deux qualificateurs en relation avec le concept d'héritage (voir chapitre classes abstraites)

extends : annonce l'héritage de la classe NomSuperClasse (voir chapitre héritage)

implements : annonce l'implémentation d'une ou plusieurs interfaces par la classe (voir chapitre interface)

Exemple

```
public class Etudiant extends Personne
{
// déclaration des attributs
// déclaration des méthodes
}
```

Déclaration des attributs

[private/public/protected][static][final]type_attribut nom_attribut [= valeur];

Déclaration des attributs

private, public et protected : des modificateurs d'accès à l'attribut à partir d'autres classes :

- **private** indique que l'accès est réservé aux méthodes de la même classe
- **public** indique que l'accès est possible à toutes les méthodes des autres classes
- **protected** limite l'accès aux méthodes de la même classe, des classes dérivées et des classes du même package
- par défaut, l'accès est dit friendly protected (ou friendly), il correspond à un accès privé au package (limite l'accès aux méthodes de la même classe, des classes dérivées dans le même package et des classes du même package)

Déclaration des attributs

static : un attribut static est partagé par tous les objets de cette classe

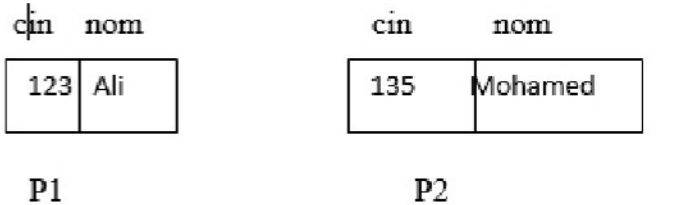
Déclaration des attributs

SANS STATIC

```
class Personne
{
int cin ;
String nom ;
}
```

P1 : objet de Personne (cin=123 et nom='Ali')

P2 : objet de Personne (cin=135 et nom='Mohamed')



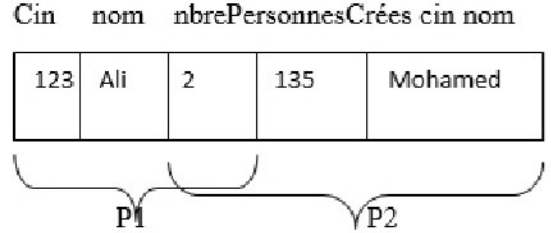
⇨ P1 et P2 sont indépendants

AVEC STATIC

```
class Personne
{
int cin ;
String nom ;
static int nbrePersonneCrées =0 ;
}
```

P1 : objet de Personne (cin=123, nom='Ali'et nbrePersonneCrées=2)

P2 : objet de Personne (cin=135 et nom='Mohamed' et nbrePersonneCrées=2)



P1 et P2 partagent l'attribut nbrePersonneCrées

Déclaration des attributs

final : un attribut constante

Exemple :

static final double pi=3.14;

type_attribut : int, float, String, char, boolean, double, ….

```
public class Personne
{
private int cin ;
protected String nom ;
public String prenom ;
int age ;
// déclaration des méthodes
}
```

Déclaration des méthodes

[private/public/protected] [static][final/abstract] type_de_retour nom_méthode ([type_paramètre1 paramètre1,…. , type_paramètreN paramètreN])

{

// déclaration des variables locales;

// instructions de la méthode

[return var_résultat;]

}

Déclaration des méthodes

private, public et protected : des modificateurs d'accès autorisant l'appel aux méthodes de la classe (voir déclaration des attributs)

static : précise que l'exécution de la méthode n'est pas liée aux objets (il s'agit d'une **méthode de classe**).

- Une méthode static est une méthode qui n'agit pas sur des variables d'instance mais uniquement sur des variables de classe.
- Les méthodes ainsi définies peuvent être appelée avec la notation **classe.nomMéthode** au lieu de **objet.nomMéthode**.
- Ces méthodes peuvent être utilisées sans instancier un objet de la classe.
- Il n'est pas possible d'appeler une méthode d'instance ou d'accéder à une variable d'instance à partir d'une méthode de classe statique.

Déclaration des méthodes

abstract et final : deux qualificateurs en relation avec le concept de classes abstraites (voir chapitre classes abstraites)

→ Le passage des paramètres est par valeur pour les arguments de type primitif (de base) et par référence pour les arguments de type classe (référence d'objet) et les tableaux

Exemple

```
class Personne
{
private int cin ;
protected String nom ;
public String prenom ;
int age;
public void affiche()
{
System.out.println(”le cin =”+cin+”le nom =”+nom+”le prénom =”+prenom+ ”et l’age =”+age) ;
}
}
```

La notion de constructeur

- Le constructeur d'une classe est une méthode appelée une seule fois par objet et ce au moment de sa création.
- Le constructeur porte le même nom que celui de la classe et n'a pas de type de retour
- Le rôle du constructeur est d'initialiser les attributs de l'objet
- En l'absence d'un constructeur défini par le programmeur, JAVA offre un constructeur non paramétré qui initialise les attributs de l'objet crée par les nulls correspondant à leurs types (0 pour les numériques, false pour les booléens et null pour les références aux objets,)
- Une classe peut avoir plusieurs constructeurs

```
public NomConstructeur ([typeArg1 Arg1, ….., typeArgN  ArgN])
{
//instructions
}
```

```
class Personne
{ private int cin;
private String nom;
private string prenom;
private int age ;
public Personne (int c, String n, String p, int a)
{cin = c;
nom=n ;
prenom=p ;
age=a ;
}
```

```
public Personne (int c, String n,)
{cin = c;
nom=n ;
}
}
```

Instanciation des objets

La création d'un objet se fait à l'aide de l'opérateur ***new*** qui se charge de :

- Allouer l'espace mémoire nécessaire pour stocker les attributs de l'objet en cours de construction
- Appeler le constructeur de l'objet adéquat pour initialiser les attributs. Le choix du constructeur se fait en fonction des paramètres passés

NomClasse nomObjet = new NomConstructeur ([Liste_paramètres]) ;

Instanciation des objets

```
class Personne
{ private int cin;
private String nom;
private string prenom;
private int age ;
public Personne (int c, String n, String p, int a)
{cin = c;
nom=n ;
prenom=p ;
age=a ;
}
```

```
public static void main (String [] args)
{
Personne p1=new Personne(1234, ''Ben Salah'', ''Mohamed'', 30) ;
}
}
```

Manipulation des objets

Les méthodes d'objets sont désignées selon la syntaxe suivante :

nomObjet.nomMéthode([Liste_paramètres]) ;

Les attributs d'objets sont désignés selon la syntaxe suivante :

nomObjet.nomAttribut

→ ***Si l'accès est autorisé***

```
class Personne
{ private int cin;
private String nom;
private String prenom;
public int age ;
public Personne (int c, String n, String p, int a)
{cin = c;
nom=n ;
prenom=p ;
age=a ;
}
public void affiche()
{
System.out.println("le cin ="+cin+"le nom ="+nom+"le prénom ="+prenom+ "et l'age ="+age) ;
}
protected void finalize()
{
System.out.println("objet détruit");
 }
}

public class testPersonne {

    public static void main(String[] args) {
        Personne P1=new Personne(1234, "Ben Salah", "Mohamed",30) ;
        P1.affiche() ;
        P1. age = 40 ; // accepté car age est public
        P1.cin =456 ;// n'est pas accepté car cin est private
        P1.finalize() ;


    }

}
```

Référence sur l'objet courant « this »

Le mot-clé *this* représente une référence sur l'objet courant, c.à.d., celui qui possède la méthode qui est en train de s'exécuter (sert simplement à référencer l'objet en cours).

Le mot clé this est implicitement présent dans chaque objet instancié et il contient la référence à l'objet actuel.

This est optionnel s'il n'y a pas d'ambigüité.

Exemple 1 :(this est optionnel)

```
class Personne
{ private int cin;
private String nom;
private String prenom;
public int age ;
public Personne (int c, String n, String p, int a)
{cin = c; // this.cin=c
nom=n ;//this.nom=n;
prenom=p ;//this.prenom=p;
age=a ;//this.age=a;
}
```

Exemple 2 : (this est obligatoire)

```
class Personne
{ private int cin;
private String nom;
private String prenom;
public int age ;
public  Personne (int cin, String nom, String prenom, int age)
{ this.cin = cin; // (obligatoire)
this.nom=nom ; // (obligatoire)
this.prenom=prenom ; // (obligatoire)
this.age =age;// (obligatoire)
}
```

Exemple 3 : (this est obligatoire)

```
class Personne
{ private int cin;
private String nom;
private String prenom;
public int age ;
Personne PlusGrand (Personne g)
{
if (age >g.age)
return this;
else
return g;
}

public  Personne (int cin, String nom, String prenom, int age)
public void affiche()
{
System.out.println("le cin ="+cin+"le nom ="+nom+"le prénom ="+prenom+ "et l'age ="+age) ;
}


public class testPersonne {

    public static void main(String[] args) {
        Personne P1=new Personne(1234, "Ben Salah", "Mohamed",30) ;
        Personne p2= new Personne (135, "Ali", "Medi", 20);
        Personne p3=P1.PlusGrand(p2);
        p3.affiche();



    }
```