Chapitre 4 Orienté Objet

Mr IGHIL Mohamed

INSIM Boumerdes

1. Présentation classe

Pour savoir comment concevoir une **classe**, on a l'exemple du **livre**.

Nous avons l'échantillon d'informations qui peuvent décrire un livre donné :

- titre;
- auteur;
- nombre de pages ;
- éditeur.

Ce sont les **attributs** de tout livre dans la vie réelle. Dans le contexte des classes, ces attributs sont appelés attributs de classe : les variables

Pour déclarer une classe en Java, utilisez le mot-clé class suivi d'un personnalisé. Ensuite, nom terminez avec des accolades ({}).

Ceci inclut la liste complète de ses attributs :

```
class Book {
     String titre;
     String auteur;
     int nPages;
     String éditeur;
```

2. Les instances

Chaque champ de l'objet
 créé doit avoir une valeur.

Ces valeurs peuvent être

fournies de plusieurs façons.

 Une manière de procéder consiste à fournir une valeur dans l'affectation qui crée la classe, c'est le constructeur. Il permet à la fois de créer une instance de la classe et de **spécifier** la **valeur** des attributs de l'objet.

```
class Book {
    String titre;
     String auteur;
     int nPages;
     String éditeur;
Book(String titre, String auteur, int nPages, String éditeur) {
       this.title = titre;
       this.author = auteur;
       this. nPages = nPages;
       this.publisher = éditeur;
```

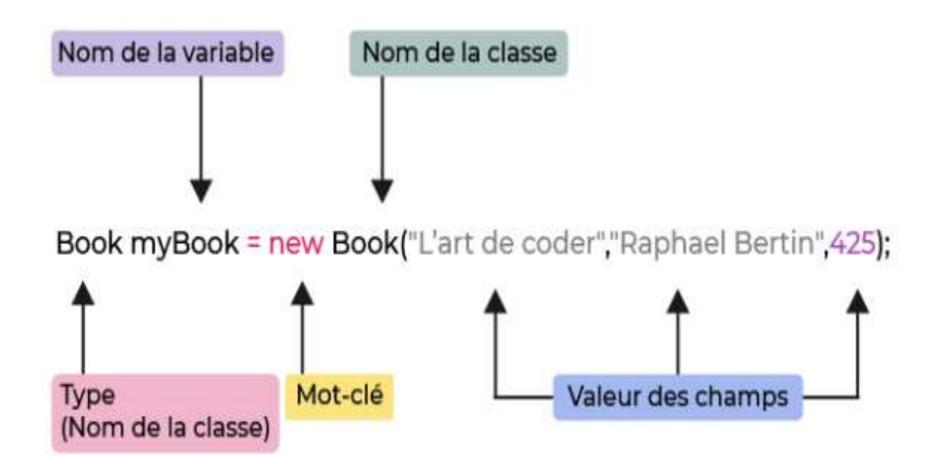
En Java, le **constructeur** est une **fonction** spéciale du **même nom** que la **classe** avec les **arguments** passés en **paramètres**.

À l'intérieur de la fonction, nous utilisons les paramètres pour initialiser les attributs de notre objet avec le mot clé **this**. Il est également possible de déclarer **plusieurs**

constructeurs différents pour la même classe.

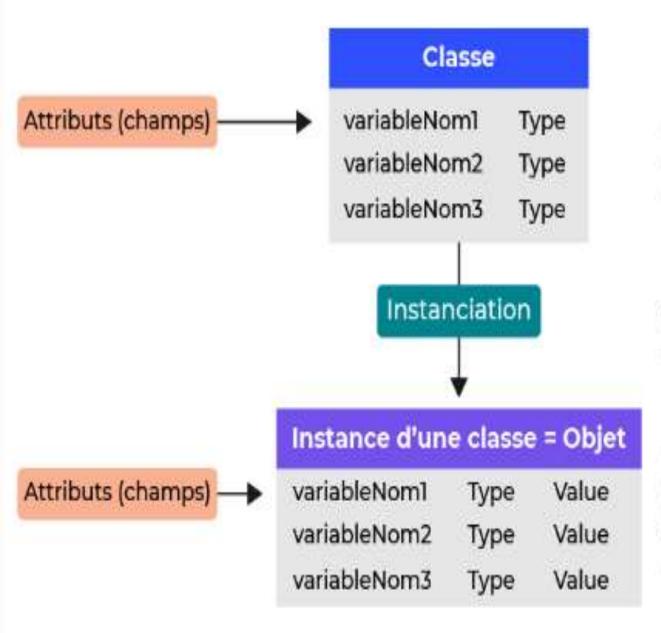
```
class Book {
    String titre;
     String auteur;
     int nPages;
     String éditeur;
Book(String titre, String auteur, int nPages, String éditeur) {
       this.titre = titre;
       this. auteur = auteur;
       this. nPages = nPages;
       this.publisher = éditeur;
```

Voici un exemple de code pour créer un livre :



Récapitulation avec un schéma

rapide:



Les attributs (aussi appelés champs) en Java sont les variables que vous définissez quand vous créez une classe.

Pour instancier un objet, vous déclarez une variable de cette classe.

Quand vous instanciez un objet de cette classe, vous définissez également la valeur de chaque champ de cet objet.

```
public class Booke {
public static void main(String[] args) {
Book mybook = new Book("IGHIL","Mohamed",100, "INSIM");
System.out.println("Titre est " + mybook.titre);
System.out.println("Auteur est : " + mybook.auteur);
System.out.println("N° Pages est : " + mybook.nPages);
System. out. println ("Editeur est : " + mybook. éditeur);
```

3. Héritage

On la classe mère:

```
public class Figure {
  private int x;
  private int y;
  public void move(int newX, int newY) {
    this.x = newX;
    this.y = newY;
```

```
Et on a la classe fille:
public class Carre extends Figure{
  private int cote;
  public int getCote() {
    return cote;
  public int getPerimetre(){
    return 4*cote;
```

classe Nous une avons mère **Figure** que nous allons spécialiser en Carre . Le mot clé est extends. On peut dire que la classe étend Carre classe Figure

Avec la classe Carre, nous récupérons automatiquement tous les attributs de la classe de mère Figure.

Et nous lui avons **ajouté un** nouvel **attribut** de classe et **2 nouvelles méthodes** participant ainsi à la **spécialisation**.

- Un champ défini comme private ne peut pas être hérité
- Une classe fille ne peut hériter que d'une seule classe mère.
- Par contre, rien n'empêche cette classe
 mère d'être la classe fille d'une autre
 classe mère

```
public class Test {
  public static void main(String[] args) {
    Figure fig = new Figure();
    fig.move(1, 1);
    Carre car = new Carre();
    car.move(2, 2);
```

3.1 Initialisez les attributs hérités

Création de constructeur de la classe mère:

```
class Figure {
  private int x;
  private int y;
  Figure( int x, int y) {
     this.x = x;
     this.y = y;
```

```
class Carre extends Figure {
 int cote;
 Carre(int cote, int x, int y){
   super(x, y);
   this.cote = cote;
```

Pour appeler le constructeur de la mère depuis classe constructeur de la classe fille, on utilise la méthode super.

```
package test;
public class Test {
public static void main(String[] args) {
Figure fig = new Figure(1,1);
fig.affiche();
Carre car = new Carre(10,2,2);
car.affiche();
```

```
public class Figure {
public int x ;
public int y ;
Figure(int x, int y) {
this.x = x;
this.y = y;
public void affiche() {
System. out. println("La valeur de X est : " + x);
System.out.println("La valeur de Y est : " + y);
```

```
package test;
public class Carre extends Figure {
int cote;
Carre(int cote, int x, int y){
super(x,y);
this.cote = cote;
System.out.println("La valeur de cote est : " + cote);
```

4. Le polymorphisme

 Lorsque vous construisez une classe **héritant** d'une autre classe, vous avez la possibilité de redéfinir certaines méthodes de la classe **mère**. Il s'agit de **remplacer** le comportement de la fonction qui a été définie par la classe mère.

 C'est le concept de polymorphisme. L'idée étant de pouvoir utiliser le même nom de méthode sur des objets **différents**. Et bien sûr, cela n'a de sens que si le comportement des méthodes est différent.

La classe mere:

```
class Animal {
  void deplacer() {
  System.out.println("Je me déplace");
```

```
class Chien extends Animal {
 void deplacer() {
      System.out.println("Je marche");
```

```
class Oiseau extends Animal {
         void deplacer(){
            System.out.println("Je
 vole");
```

```
class Pigeon extends Oiseau {
  void deplacer() {
  System.out.println("Je vole en ville");
```

Sur toutes ces classes, On peut appeler deplacer() .

Le polymorphisme permet alors d'appeler la méthode **adéquate** selon le **type d'objet** :

```
public class Test {
public static void main(String[] args) {
  Animal a1 = new Animal();
  Animal a2 = new Chien();
  Animal a3 = new Pigeon();
  a1.deplacer();
  a2.deplacer();
  a3.deplacer();
```

On accéde à l'implémentation parente grâce au mot clé super, et appeler la méthode déplacer avant d'ajouter l'aboiement du chien:

```
class Chien extends Animal {
 void deplacer() {
   super.deplacer();
   System.out.println("ouaf ouaf");
```