



Cours java Objet

Présenté par : Mr Djamel MOUCHENE

1er octobre
2019

1



Classe

- ▶ Un programme en Java est défini comme une classe
- ▶ Dans une classe:
 - attributs, méthodes
- ▶ L'en-tête de la classe
 - public class NomDeClasse**
 - *public* = tout le monde peut utiliser cette classe
 - *class* = unité de base des programmes OO
- ▶ Une classe par fichier
- ▶ La classe NomDeClasse doit être dans le fichier NomDeClasse.java
- ▶ Si plus d'une classe dans un fichier *.java*, *javac* génère des fichiers *.class* séparés pour chaque classe

Mouchene Djamel – HTC 1er octobre
2019



Classe

► Le corps

```
{
    ...
}
```

► Contient les attributs et les méthodes

- Attributs: pour stocker les informations de la classe
- Méthodes: pour définir ses comportement, ses traitements, ...

► Conventions et habitudes

- nom de classe: **NomDeClasse**
- indentation de { }
- indentation de ...
- Les indentations correctes ne seront pas toujours suivies dans ces notes pour des raisons de contraintes d'espace par PowerPoint...



Méthode Main : en-tête

► L'en-tête:

```
public static void main(String[] args)
```

- *main*: nom de méthode
- *void*: aucune sortie (ne retourne rien)
- *String[] args*: le paramètre (entrée)
 - *String[]*: le type du paramètre
 - *args*: le nom du paramètre

► Conventions

- **nomDeParametre**
- **nomDeMethode**
- **nomDAttributs**
- **nomDObjet**



Méthode: corps

► Le corps:

```
{
    // afficher une salutation
    System.out.println("Hello, World!");
}
```

► contient une séquence d'instructions, délimitée par { }

- // afficher une salutation : commentaire
- System.out.println("Hello, World!"); appel de méthode

► les instructions sont terminées par le caractère ;



Méthode: corps

► En général:

nomDObjet.nomDeMethode(<liste des paramètres>)

- System.out: l'objet qui représente le terminal (l'écran)
- println: la méthode qui imprime son paramètre (+ une fin de ligne) sur un *stream* (écran)
- **System.out.println("Hello, World!");**
- "Hello, World!": le paramètre de println

► La méthode **main**

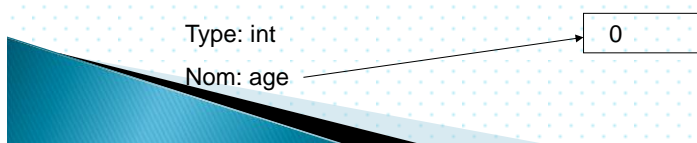
- "java Hello" exécute la méthode *main* dans la classe *Hello*
- *main* est la méthode exécutée automatiquement à l'invocation du programme (avec le nom de la classe) qui la contient



Variable

- ▶ Variable: contient une valeur
 - Nom de variable
 - Valeur contenue dans la variable
 - Type de valeur contenue
 - *int*: entier, *long*: entier avec plus de capacité
 - *Integer*: classe entier, avec des méthodes
 - *float*: nombre réel avec point flottant, *double*: double précision
 - *String*: chaîne de caractères ("Hello, World!")
 - *char*: un caractère en Unicode ('a', '\$', 'é', ...)
 - *boolean*: true/false
- ▶ Définition générale
`Type nomDeVariable;`

Exemple: `int age;`



Classe et Objet

- ▶ Classe: moule pour fabriquer des objets
- ▶ Objet: élément concret produit par le moule
- ▶ Définition de classe:

<pre>class NomClasse { Attributs; Méthodes; }</pre>	<pre>class Personne { String nom; int AnneeNaissance; public int age() {...} }</pre>
---	--
- ▶ Une classe regroupe un ensemble d'objets (instances)



Objet

- ▶ Structure à deux parties:
 - Référence
 - Corps
- ▶ Les étapes
 - Déclaration de la classe (ex. Personne)
 - À l'endroit où on utilise:
 - Déclarer une référence du type de la classe
 - Créer une instance d'objet (*new*)
 - Manipuler l'objet

Exemple



```
public class Personne {
    public String nom;
    public int anneeNaissance;
    public int age() {return 2019 - anneeNaissance; }
}

class Utilisation {
    public static void main(String[] args) {
        Personne qui;
        qui = new Personne();
        qui.nom = "Pierre";
        qui.anneeNaissance = 1980;
        System.out.println(qui.age());
    }
}
```

Déclaration de référence

Création d'une instance

Manipulation de l'instance
référé par la référence

qui

Personne:
age()

nom: "Pierre"
anneeNaissance: 1983



Un autre exemple

```
class Circle {
    public double x, y; // coordonnées du centre
    private double r; // rayon du cercle
    public Circle(double r) {
        this.r = r;
    }
    public double area() {
        return 3.14159 * r * r;
    }
}

public class MonPremierProgramme {
    public static void main(String[] args) {
        Circle c; // c est une référence sur un objet de type Circle, pas encore
        un objet
        c = new Circle(5.0); // c référence maintenant un objet alloué en
        mémoire
        c.x = c.y = 10; // ces valeurs sont stockées dans le corps de l'objet
        System.out.println("Aire de c : " + c.area());
    }
}
```

'r' est inaccessible de l'extérieur de la classe

constructeur

Math.PI



Constructeurs d'une classe

- ▶ Un constructeur est une façon de fabriquer une instance
- ▶ Une classe peut posséder plusieurs constructeurs
- ▶ Si aucun constructeur n'est déclaré par le programmeur, alors on a la version par défaut: `NomClasse()`
- ▶ Plusieurs versions d'une méthode: surcharge (*overloading*)

```
class Circle {
    public double x, y; // coordonnées du centre
    private double r; // rayon du cercle
    public Circle(double r) {
        this.r = r;
    }
    public Circle(double a, double b, double c) {
        x = a; y = b; r = c;
    }
}

public class Personne {
    public String nom;
    public int anneeNaissance;
    public int age() { return 2008 - anneeNaissance; }
}
```

this: réfère à l'objet courant

Le constructeur `Personne()` est déclaré par défaut

Manipulation des références

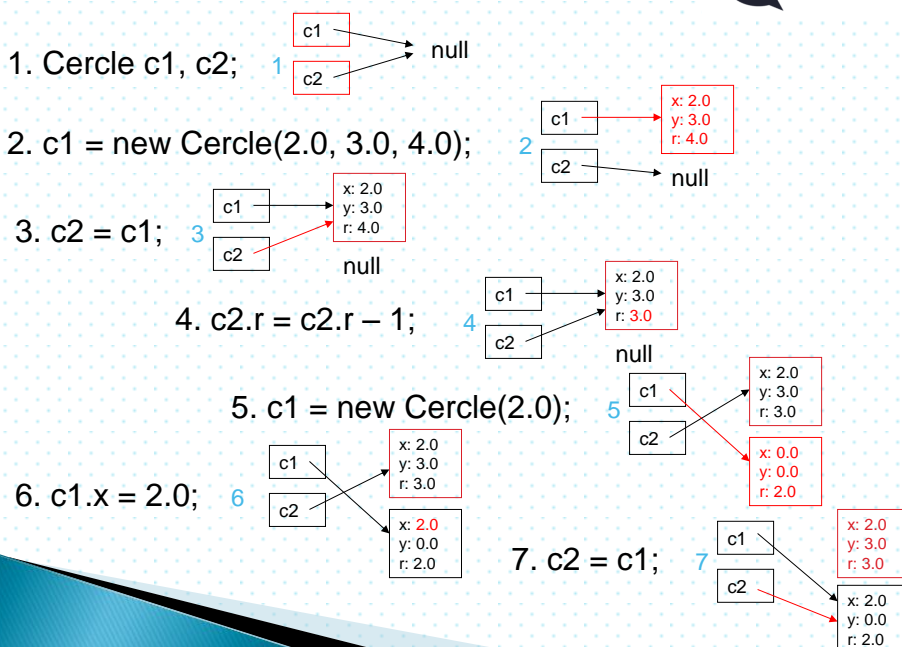


```
class Circle {
    public double x, y; // coordonnées du centre
    private double r; // rayon du cercle
    public Circle(double r) {
        this.r = r;
    }
    public Circle(double a, double b, double c) {
        x = a; y = b; r = c;
    }
}

// Dans une méthode, par exemple, main:
Circle c1, c2;
c1 = new Circle(2.0, 3.0, 4.0);
c2 = c1; // c2 et c1 pointent vers le même objet
c2.r = c2.r - 1; // l'objet a le rayon réduit
c1 = new Circle(2.0); // c1 point vers un autre objet, mais c2 ne change pas
c1.x = 2.0; // on modifie le deuxième objet
c2 = c1; // maintenant, c2 pointe vers le 2ième objet aussi
```

- ▶ Que faire du premier objet?
 - Aucune référence ne pointe vers lui
 - L'objet est perdu et inutilisable
 - Ramasse miette (*garbage collector*) va récupérer l'espace occupé par l'objet
- Comparaison des références
(`c1 == c2`): est-ce que `c1` et `c2` pointent vers le même objet?

Illustration





Manipulation des objets

► Forme générale

référence.attribut: réfère à un attribut de l'objet

référence.méthode(): réfère à une méthode de l'objet

► **static**: associé à une classe

- Attribut (variable) statique: si on le change, ça change la valeur pour tous les objets de la classe
- Méthode statique: on la réfère à partir de la classe

• *Classe.méthode*

• E.g. *Math.sqrt(2.6)*: Appel à la méthode *sqrt* de la classe *Math*

• Constante: *Math.PI*

- Dans une classe: **static final float PI = 3.14159265358979;**
- Une constante n'est pas modifiable

Classes et Héritage



► Héritage

- Les enfants héritent les propriétés du parent
- Classe enfant (sous-classe) possède systématiquement les attributs et les méthodes de la classe parent (super-classe)
- Héritage simple (une seule super-classe au plus)

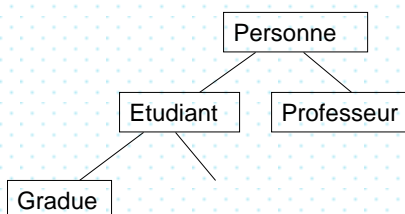
► E.g.

```
class Personne {
    String nom;
    int anneeNaissance;
    public int age() { return 2008 - anneeNaissance; }
}
```

```
class Etudiant extends Personne {
    String [] cours;
    String niveau;
    String ecole;
    ...
}
```

► Ce qui est disponible dans Etudiant:

- nom, anneeNaissance, age(),
- cours, niveau, ecole, ...





Principe

- Définir les propriétés communes dans la classe supérieure
- Définir les propriétés spécifiques dans la sous-classe
- Regrouper les objets le plus possible
- Les objets d'une sous-classe sont aussi des objets de la super-classe
- La classe dont tous les objets appartiennent: *Object*
- Tester l'appartenance d'un objet dans une classe: *instanceof* (e.g. *qui instanceof Etudiant*)

Exemple



```
public class Ellipse {
    public double r1, r2;
    public Ellipse(double r1, double r2) { this.r1 = r1; this.r2 = r2; }
    public double area() { return r1 * r2 * Math.PI; }
}

final class Circle extends Ellipse {
    public Circle(double r) { super(r, r); }
    public double getRadius() { return r1; }
}

// Dans une méthode
Ellipse e = new Ellipse(2.0, 4.0);
Circle c = new Circle(2.0);
System.out.println("Aire de e: " + e.area() + ", Aire de c: " + c.area());
System.out.println((e instanceof Circle)); // false
System.out.println((e instanceof Ellipse)); // true
System.out.println((c instanceof Circle)); // true
System.out.println((c instanceof Ellipse)); // true (car Circle dérive de Ellipse)
e = c;
System.out.println((e instanceof Circle)); // true
System.out.println((e instanceof Ellipse)); // true
int r = e.getRadius(); // erreur: méthode getRadius n'est pas trouvée dans la
                        // classe Ellipse
c = e; // erreur: type incompatible pour = Doit utiliser un cast explicite
```

super(r,r): constructeur de la super-classe

final assure qu'aucune autre classe n'hériterait de Circle



Surcharge de méthode

```
class A {
    public void meth() {System.out.println("Salut"); }
}
class B extends A {
    public void meth(String nom) {
        System.out.println("Salut" + nom);
    }
}
```

- ▶ Dans la sous-classe: une version additionnelle
 - Signature de méthode: nom+type de paramètres
 - Surcharge: créer une méthode ayant une autre signature



Overriding: écrasement

- ▶ Par défaut, une méthode est héritée par une sous-classe
- ▶ Mais on peut redéfinir la méthode dans la sous-classe (avec la même signature)
- ▶ Les objets de la sous-classe ne possèdent que la nouvelle version de la méthode
- ▶ E.g.

```
class A {
    public void meth() {System.out.println("Salut");}
}
class B extends A {
    public void meth() {System.out.println("Hello");}
}
```

```
A a = new A();
B b = new B();
a.meth(); // Salut
b.meth(); // Hello
a = b; // a réfère à un objet de classe B
a.meth(); // Hello. Même si la référence est de classe A, l'objet est de
classe B
```





Classe abstraite

- ▶ Certains éléments peuvent être manquants dans une classe, ou la classe peut être trop abstraite pour correspondre à un objet concret
- ▶ Classe abstraite
 - Une classe non complétée ou une classe conceptuellement trop abstraite
 - Classe *Shape*
 - on ne connaît pas la forme exacte, donc impossible de créer un objet
 - cependant, on peut savoir que chaque *Shape* peut être dessinée

```
abstract class Shape {
  abstract void draw();
}
```



Interface

- ▶ Interface
 - Un ensemble de méthodes (comportements) exigées
 - Une classe peut se déclarer conforme à (implanter) une interface: dans ce cas, elle doit implanter toutes les méthodes exigées

▶ E.g.

```
public interface Inter {
  public abstract int carre(int a);
  public abstract void imprimer();
}

class X implements Inter {
  public int carre(int a) { return a*a; }
  public void imprimer() {System.out.println("des informations"); }
}
```



Utilité de l'interface

- ▶ Permet de savoir qu'une classe contient les implantations de certaines méthodes
- ▶ On peut utiliser ces méthodes sans connaître les détails de leur implantation
- ▶ Souvent utilisée pour des types abstraits de données (e.g. pile, queue, ...)