

- Attributs et méthodes de classes
  - Attributs de classe ou variables de classe
  - Retour sur les données constantes
  - Méthodes de classe
  - Exemples de méthodes de classes
- Introduction à l'héritage (1ère partie)

## Déclaration des attributs de classe en Java

- En Java les attributs de classe sont définis avec le mot-clé `static`

```
private static type_retour NomAttribut ;
```
- Exemple : Dans une application musicale, je veux associer à une plage d'un CD un style, alors la cohérence veut que la liste des styles de musique soit partagée par tous les objets de type Plage de CD. De plus, on veut pouvoir faire évoluer la liste (ajouter de nouveaux styles).

```
private static String[] Styles =
    {"classique", "reggae", "jazz", "celte"} ;
```

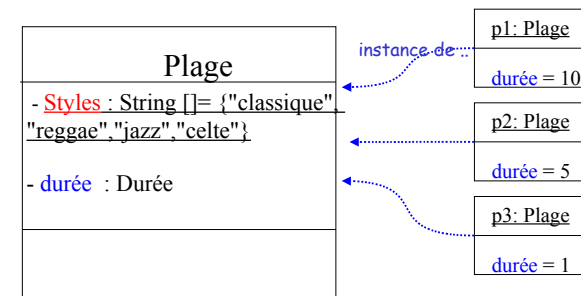
Lisibilité : les noms des attributs de classe commencent par une Majuscule

## Attributs de classe ou variables de classe

- Un **attribut d'instance** est associé à une **instance** d'une classe
  - le rayon est un attribut d'un cercle
  - chaque instance de la classe Cercle aura sa propre valeur de rayon
- Quand la valeur d'un attribut est la même pour tous les objets de la classe :
  - Tous les canaris sont jaunes, signifie que la caractéristique de couleur jaune doit être associée à la classe des canaris.
  - Elle doit être partagée par tous les objets de la classe, la variable commune est à associer à la classe : il s'agit d'une *variable de classe* ou *attribut de classe*.
  - Un **attribut de classe** est associé à la **classe** elle-même

## Utilisation des attributs de classe

- Un **attribut de classe** a une valeur qui est stockée dans la classe elle-même et non dans les objets instances.
- Ici **Styles** est un attribut de classe privé accessible dans toute la classe.



## Attention à l'utilisation de *this*


- Dans la définition des méthodes d'instance d'une classe, le mot clé **this** est utilisé :
  - Soit pour désigner un attribut d'instance,
  - Soit pour faire référence à un objet instance de la classe : l'objet sur lequel la méthode sera invoquée.
- Un attribut de classe ne peut donc PAS être désigné avec this.
- On utilise directement le nom de l'attribut ou précédé du nom de la classe.
- Le nom d'un attribut de classe doit commencer par une majuscule pour améliorer la lisibilité du code de la classe.

## Accès à un attribut de classe publique

- Quand un attribut de classe est public c'est qu'il correspond à un objet très particulier du langage.
- Un exemple classique est celui de la référence **out** au canal de sortie standard défini comme une variable **static** dans la classe `java.lang.System`

```
public static final PrintStream out
```
- L'accès à cet attribut de classe utilise la notation pointée, mais pour la classe :
  - **System.out** // accès à travers la classe System

```
System.out.println("classique");
```



## Les attributs de classe de la classe System

Field Summary		
static <code>PrintStream</code>	<code>err</code>	The "standard" error output stream.
static <code>InputStream</code>	<code>in</code>	The "standard" input stream.
static <code>PrintStream</code>	<code>out</code>	The "standard" output stream.

**System.out** est un objet de type **PrintStream**.

La classe **PrintStream** définit les méthodes **print** et **println** pour des paramètres de type String et primitifs.

## Exemple des Comptes Bancaires

- Nous voulons assigner automatiquement un numéro à un compte bancaire lors de sa création.
- On conserve le dernier numéro dans la variable de classe.

```
public class CompteBancaire {  
    private int numero;  
    private double solde;  
    private static int DernierNumero= 1000;  
  
    public CompteBancaire () {  
        // genere un nouveau numero  
        DernierNumero = DernierNumero + 1;  
        this.setNumero(DernierNumero);  
        this.setSolde(0);  
    }  
}
```

## Retour sur les données constantes

- Donnée constante : une fois initialisée sa valeur ne change plus à l'exécution.
- Un attribut d'instance ou variable d'instance avec une valeur constante est déclaré **final**.
- Les noms des constantes sont en majuscules.
- On n'utilise la notion de constante que pour les primitifs
- Que cela signifie-t-il pour les objets ?

## Constante de classe = static + final

- Un attribut de classe peut être une donnée constante.
- La valeur de PI est constante et commune à tous les objets géométriques, donc on ne veut qu'un seul exemplaire, partagé par toutes les instances de Cercle.

```
public class Cercle {  
    private double r; //le rayon  
    private static final double PI = 3.1416;  
    ...  
    public double circonference() {  
        double res;  
        res = 2* PI * this.getRayon();  
        return res;  
    }  
}
```

Annotations: "constante" points to `static final`; "attribut de classe" points to `PI`.

## Les attributs de la classe Math

- La classe Math possède deux constantes de classe publiques dont l'une est le nombre PI :
- ```
public static final double PI
```
- La constante est déclarée publique et peut donc être utilisée à l'extérieur de la classe Math par toute classe qui en a besoin.  
L'accès à cet attribut de classe se fait par :  
**Math.PI**

## Utilisation de la constante Math.PI

```
public class Cercle {  
    private double r; //le rayon  
    ...  
    public double circonference() {  
        double res;  
        res = 2* Math.PI * this.getRayon();  
        return res;  
    }  
}
```

## Méthode d'instance /Méthode de classe

- Les méthodes définies dans une classe et destinées aux instances sont des [méthodes d'instance](#).
- Elles sont utilisées par des envois de messages aux objets instances.
- Parfois une classe définit des méthodes qui ne sont pas destinées à être invoquées sur un objet de la classe.
- Il s'agit alors de [méthodes de classe](#).
- Elles sont utilisées par des envois de messages aux classes elles-mêmes.

## Utilisation des attributs et méthodes de classe

Méthodes statiques : les éléments statiques (méthodes, variables, constantes) sont associés à la classe elle-même, et non à ses instances, et ne peuvent être utilisés qu'avec d'autres éléments statiques.

- La classe Math ne possède que des méthodes statiques :

```
public static double abs(double a)
public static int max(int a, int b)
public static double min(double a, double b)
public static double sin(double a)
public static double random()
Etc..
```

## Limitations des méthodes de classe

- Les méthodes de classe sont associées à une classe et non à une instance.
- Une méthode de classe ne peut pas accéder aux attributs d'instance définis dans la classe.
- Une méthode de classe ne peut pas appeler une méthode d'instance de la classe (cf méthodes de test).

Une méthode de classe :

- ne peut accéder qu'aux [variables de classes](#) de sa classe
- ne peut invoquer que d'autres [méthodes de classes](#) définies dans sa classe.

## Des méthodes de classes pour des conversions

- La classe String possède des méthodes de classe pour convertir des données en chaînes de caractères :

```
public static String valueOf(Type)
```

où Type peut être tout type primitif : boolean, char, int, double etc.

- La méthode retourne une [nouvelle chaîne](#) de caractère, elle peut être utilisée pour instancier de nouveaux objets de type String quand c'est nécessaire :

```
String number = String.valueOf(50+3);    //"53"
String truth = String.valueOf(true);      //"true"
String number = String.valueOf(Math.PI);  //"3.14159"
```

## Dans les classes Wrapper

- Les classes Wrapper ont elles aussi des méthodes de classe

```
Float : public static String toString(float f)
        public static float parseFloat(String s)
```

```
Integer : public static String toString(int i)
```

Etc.

- `Math`
  - `public static int round(float a)` : retourne le int le plus proche de a
  - `public static long round(double a)`

## Les classes de test et les méthodes statiques

```
public class TestEtudiant{
    public static void main (String[] args) {
        testMoyenne() ;
    }

    public void testMoyenne() {
        // ...
    }
}
```

Erreur à la compilation :

```
../TestEtudiant.java:27: non-static method testMoyenne()
cannot be referenced from a static context
```

## Les méthodes appelées doivent être des méthodes statiques

```
public class TestEtudiant{
    public static void main (String[] args) {
        testMoyenne() ; // envoi de message sans receveur
    }

    public static void testMoyenne() {
        // ...
    }
}
```

## Introduction à l'héritage

## Principe de l'héritage

- Développer en objets c'est aller du simple au complexe.
- Pour construire une nouvelle classe on ajoute des attributs et des méthodes à une ou plusieurs classes déjà existantes.
- L'héritage accroît la modularité des applications
  - facilite la mise au point
  - facilite la maintenance
  - permet un gain de place par mise en commun des informations

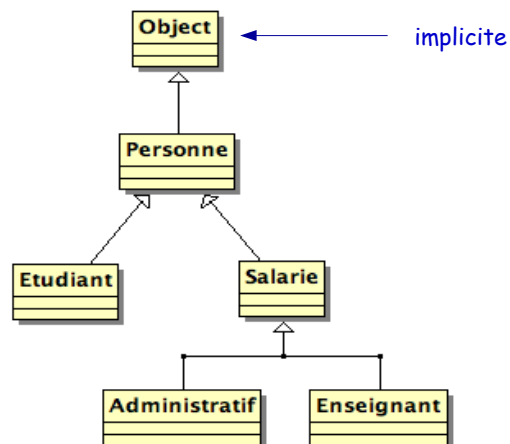
## Structure hiérarchique des classes

Toute classe est sous-classe d'une autre classe.

La classe racine est la classe *Object*.

- La classe *Object* décrit le comportement par défaut de tous les objets du système.
- Plusieurs classes peuvent partager la même super-classe.
- La structure des classes est arborescente.

## Exemple de spécialisation

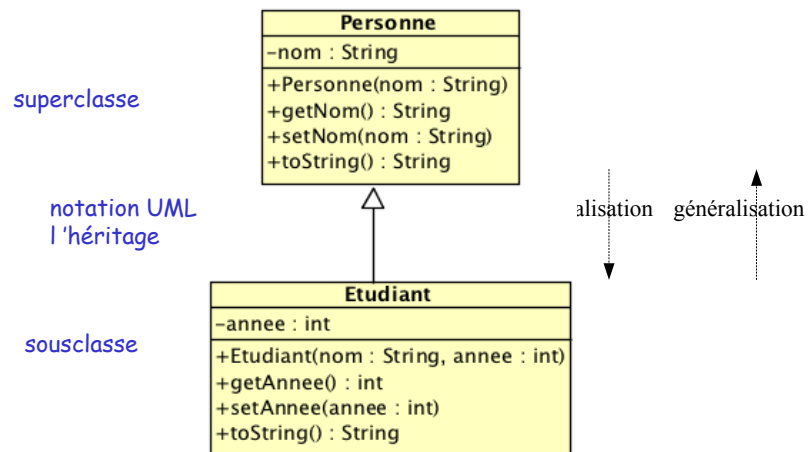


## Exemple : la classe Personne

```
public class Personne {  
    private String nom;  
  
    public Personne (String unNom) {  
        this.setNom(unNom);  
    }  
    public String getNom() {  
        return this.nom;  
    }  
    public void setNom(String unNom) {  
        this.nom = unNom;  
    }  
    public String toString() {  
        return("Nom : " + this.getNom());  
    }  
}
```

| Personne                |
|-------------------------|
| -nom : String           |
| +Personne(nom : String) |
| +getNom() : String      |
| +setNom(nom : String)   |
| +toString() : String    |

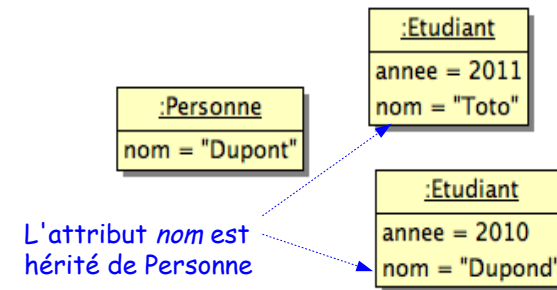
## La classe Etudiant hérite de la classe Personne



IB - R2.01

25/33

## Héritage statique des attributs



IB - R2.01

26/33

## Déclaration de l'héritage en Java

- Pour déclarer que la classe Etudiant hérite de la classe Personne on écrit :

```
public class Etudiant extends Personne{
    ...
}
```

IB - R2.01

27/33

## Le code Java de la classe Etudiant

```
public class Etudiant extends Personne {
    private int annee;

    public Etudiant (String unNom, int uneAnnee) {
        super(unNom);
        this.setAnnee(uneAnnee);
    }
    public int getAnnee() {
        return this.annee;
    }
    public void setAnnee(int annee) {
        this.annee = annee;
    }
    public String toString() {
        String resultat = "Nom : " + this.getNom() + "\n" +
            "Année : " + this.getAnnee();

        return resultat;
    }
}
```

IB - R2.01

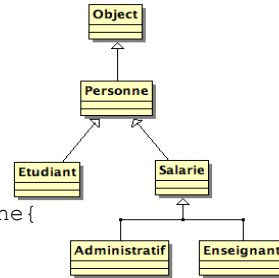
28/33

## Exemple

de même

```
public class Salarie extends Personne{  
    ...  
}
```

```
public class Enseignant extends Salarie{  
    ...  
}
```



## Vocabulaire de l'héritage

- Une classe B hérite d'une classe A :  

```
public class B extends A {  
    ...  
}
```

B hérite de A

A est la superclasse de B

B est une sous-classe de A

## Exemple : utilisation des méthodes héritées

```
public class ScenarioEtudiant {  
  
    public static void main(String[] args) {  
        Etudiant etudiant1 = new Etudiant("Toto", 2011);  
  
        etudiant1.setNom("Dupond"); // méthode héritée de Personne  
        etudiant1.setAnnee(4); // méthode définie dans Etudiant  
    }  
}
```

## Factorisation des structures d'information

- Le mécanisme d'héritage permet une factorisation des programmes.
- Chaque classe hérite des attributs et des méthodes de sa super-classe.
- Un seul exemplaire du code.
- Les noms des attributs d'instance d'une classe doivent être différents de ceux de ses superclasses.



# Attributs d'instance privés

- Les attributs d'instance sont déclarés privés, donc non accessibles des autres classes, même si on en hérite.
- Exemple : classe [Etudiant](#)

Quel est le problème ?

```
public String toString() {  
    String resultat = "Nom : " + this.nom + "\n" +  
                      "Année : " + this.annee();  
    return resultat;  
}
```

```
public String toString() {  
    String resultat = "Nom : " + this.getNom() + "\n" +  
                      "Année : " + this.annee();  
    return resultat;  
}
```