LU2IN002
HÉRITAGE

Vincent Guigue & Christophe Marsala

SCIENCES SORBONNE UNIVERSITÉ

LIP6

MLIA Machine Learning & Deep Learning for Information Access

Principe 1: Encapsulation

- Rapprochement données (attributs) et traitements (méthodes)
- Protection de l'information (private/public)

Principe 2: Agrégation/Association

- Classe A **A UN** Classe B
- Classe A **UTILISE** Classe B

Principe 3: Héritage

- Class B **EST UN** Classe A

Idée de l'héritage

Spécialiser une classe, ajouter des fonctionnalités dans une classe
Hériter tout le comportement d'une classe existante

- **1 classe de base = plusieurs spécialisations possibles**
  - Animaux → vache, chien, mouton...
  - Hiérarchisation possible: Animaux → AnimauxAiles → Papillon
- **Ne pas modifier le code existant**
  - Point → PointNomme: un point avec un nom
  - Ne pas modifier la classe de base
- **Mais ne pas faire de copier coller**
  - Hériter le comportement d'une classe

Pour les cas suivants: dire si les relations sont des relations de type **Agrégation/Association** ou **Héritage**:

- Salle de Bains et Baignoire
- Piano et JoueurPiano
- Personne, Enseignant et Etudiant
- Animal, Chien et Labrador
- Cercle et Ellipse
- Entier et Réel

Pour les cas suivants: dire si les relations sont des relations de type **Agrégation/Association** ou **Héritage**:

- Salle de Bains et Baignoire
- Piano et JoueurPiano
- Personne, Enseignant et Etudiant
- Animal, Chien et Labrador
- Cercle et Ellipse
- Entier et Réel

Pour les cas suivants: dire si les relations sont des relations de type **Agrégation/Association** ou **Héritage**:

- Salle de Bains et Baignoire
- Piano et JoueurPiano
- Personne, Enseignant et Etudiant
- Animal, Chien et Labrador
- Cercle et Ellipse
- Entier et Réel

Pour les cas suivants: dire si les relations sont des relations de type **Agrégation/Association** ou **Héritage**:

- Salle de Bains et Baignoire
- Piano et JoueurPiano
- Personne, Enseignant et Etudiant
- Animal, Chien et Labrador
- Cercle et Ellipse
- Entier et Réel

Pour les cas suivants: dire si les relations sont des relations de type **Agrégation/Association** ou **Héritage**:

- Salle de Bains et Baignoire
- Piano et JoueurPiano
- Personne, Enseignant et Etudiant
- Animal, Chien et Labrador
- Cercle et Ellipse
- Entier et Réel

$\Rightarrow$ Qu'est ce qu'une SalleDeClasse?

Réponse détaillée un peu plus tard

Problème

On veut implémenter deux classes:

1. Point en 2 dimensions
2. Point en 2 dimensions qui possède un nom

```
1  public class Point { // (1) Type / nom de classe
2    private double x; private double y;   // (2)  Attributs
3
4    public Point(double x, double y) { this.x = x;  this.y = y; } // (3) Constructeurs
5    public double getX() {   return x; }
6    public double getY() {   return y; }
7
8    public double calculeDistance(Point p2) {   // (4) methodes (traitements)
9       double dx = Math.abs(p2.getX() - getX()); double dy = Math.abs(p2.getY() - getY
10      return Math.sqrt(dx*dx+dy*dy);
11   }
12   public void move(double tx, double ty) { deplace(x+tx,y+ty); }
13   private void deplace(double x, double y) {   this.x=x; this.y=y; }// (5) methodes
14
15   public String toString() {  return "("+x+","+y+")"; } // (6) methodes standards
```

Héritage : syntaxe

```java
public class PointNomme extends Point {
  private String name;

  public PointNomme(double x, double y, String name) {
      super(x, y);
      this.name = name;
  }

  public String toString() {
    return "PointNomme [name=" + name + " " +
          super.toString() +  "]";
  }
}
```

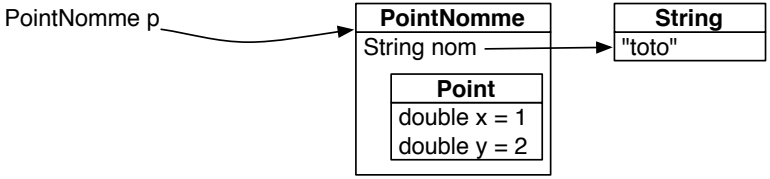
- Mot clé `extends` dans la signature de la classe
- Mot clé `super`

Erreur courante

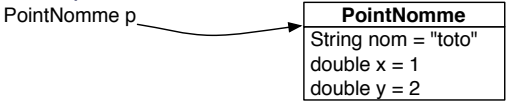
Attention à ne pas dupliquer les attributs: la classe fille étend la super-classe, elle *contient* la super-classe

```
1 PointNomme p = new PointNomme(1, 2, "toto");
```

1. Représentation complète: tous les objets sont séparés, on représente explicitement la séparation entre attributs de la classe et de la super-classe



2. Représentation usuelle simplifiée:



Attention, la représentation de la String est limite...

Idée:

1. construire une instance de la classe mère
2. initialiser les attributs relatifs à la classe fille

```java
public class PointNomme extends Point {
    private String name;
    public PointNomme(double x, double y, String name) {
        super(x, y); // premiere instruction, obligatoirement
        this.name = name; // init. attributs de la classe fille
    }
```

- **Règle générale:** choisir un constructeur dans la super-classe et l'appeler avec `super(...)`
- **Exception:** si la super-classe a un constructeur sans argument, l'appel à `super()` est implicite

Cas particulier : `super()`

S'il existe un constructeur accessible & sans argument dans la super-classe:

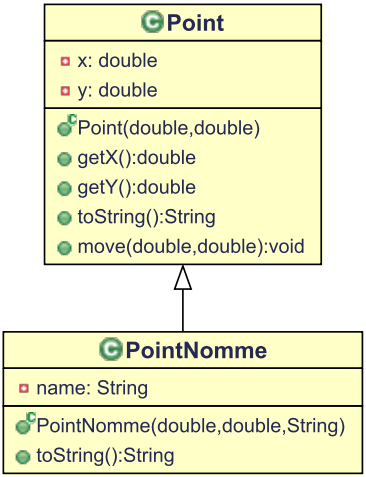
```
1 public class Point {
2     private double x,y;
3     public Point(){
4         x=0; y=0;
5     }
6     ...
```

Alors, les deux écritures suivantes sont équivalentes:

```
1 public class PointNomme
2                 extends Point {
3 private String name;
4
5 public PointNomme(String name) {
6         super();
7         this.name = name;
8     }
9     ...
```

```
1 public class PointNomme
2                 extends Point {
3 private String name;
4
5 public PointNomme(String name) {
6           this.name = name;
7     }
8     ...
```

Représentation des liens UML



- Extension des capacités/propriétés d'un objet
- `PointNomme p = new PointNomme(1,2,"totot");`
- `p` est un PointNomme
- `p` est un Point (accès à `getX()`, `getY()`...)

Si B hérite de A, implicitement, B hérite de:

- des **méthodes publiques** de A
- des **méthodes protégées** de A
- d'un attribut **super** du type de la super-classe (A)
  - **super** référence la *partie de B* qui correspond à A

En revanche, B n'hérite pas:

- des **attributs privés** de A
- des **méthodes privées** de A
- des contructeurs **publiques, privés ou protégés** de A
  - invocation spécifique via `super()`

PointNomme
PointNomme(x, y, nom)
getX() : double getY() : double move(tx :double, ty:double) : void toString() : String **getNom() : String**

```java
PointNomme p =
    new PointNomme(1,2,"p1");

System.out.println(p.getX());
// methode definie dans Point
// heritee dans PointNomme
// utilisee de maniere transparente
```

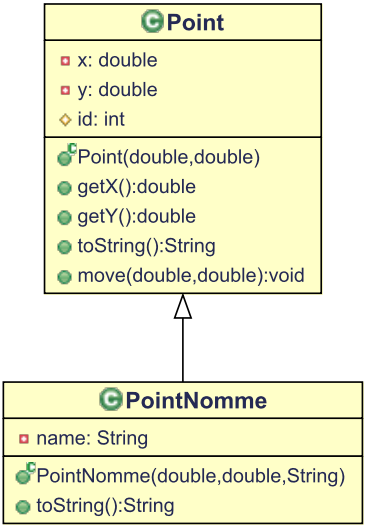
- Méthodes publiques de Point (mais pas les constructeurs)
- Pas de vision sur les données `private`

protected

- Niveau intermédiaire
- Les attributs et méthodes protected ne sont pas visibles de l'extérieur mais sont visibles dans les classes filles

- public: visible partout, dans la classe et chez le client (main, autres classes...)
- protected: visible dans la classe, dans les classes filles mais nulle part ailleurs.
- private: visible dans la classe seulement

Exemple d'accès protected



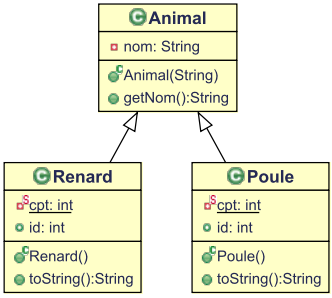

```java
public class Point {
  private double x,y;
  // les classes derivees y ont acces
  protected int id;
  ...
}
public class PointNomme {
  ...
  void methode(double d){
    int toto = id; // ou super.id;
    ...
  }
}
```

Variable/Méthode `protected`

- Accès depuis la classe
- Accès depuis les classes filles
- Pas d'accès depuis l'extérieur

Une classe $\Rightarrow$ 3 visions possibles: développeur, héritier, client

Cas particulier : arguments par défaut



- Le constructeur de la super-classe a des arguments
- Les constructeurs des classes filles non...
  - On donne des arguments par défaut

```
1  public class Animal {
2    private String nom;
3    public Animal( String nom) {
4      this.nom = nom;
5    }
6    public String getNom(){
7      return nom;
8    }
9  }
10
11 public class Poule extends Animal{
12   private static int cpt = 0;
13   public int id;
14
15   public Poule() {
16     super("poule");
17     id = cpt++;
18   }
19   public String toString(){
20     return String.format("%s%02d",
21                 getNom(), id);
22   }
23 }
```

- Un nouveau paradigme de réflexion, au coeur de la POO
- Des éléments de syntaxe à maitriser
- ... To be continuated (**subsomption, surcharge & redéfinition**)