Approches Objet de la Programmation

Agrégation, Composition, Héritage

Didier Verna

didier@didierverna.net

didierverna.net @didierverna didier.verna in/didierverna

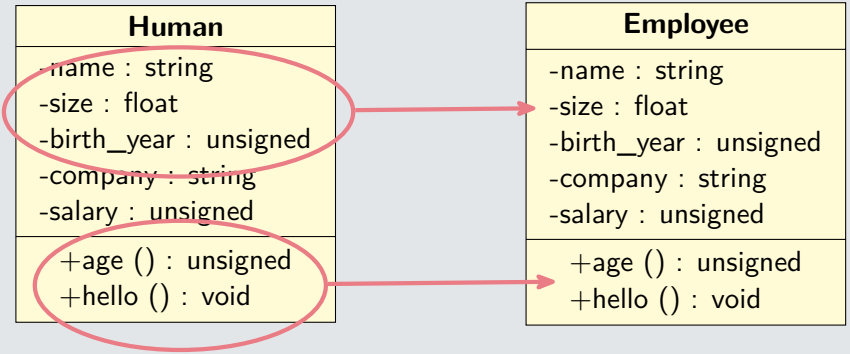
Build 2025-05-21 19:06:52+02:00

Plan

Plan

Copier / Coller

Exemple



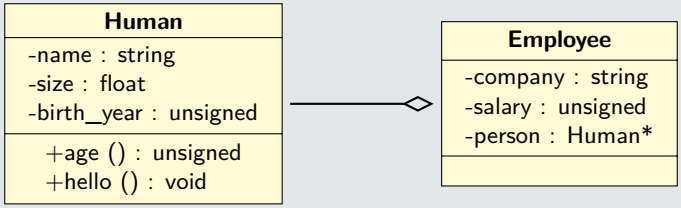
😠 Très mauvaise approche !

Plan

Agrégation

- Une forme d'inclusion
  - **Agrégat** : maintient de références/pointeurs vers des objets *agrégés*

Exemple



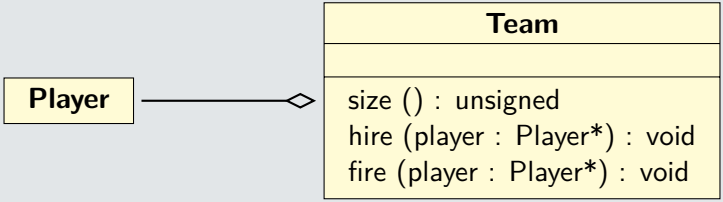
😕 Pas très adapté dans ce cas

- Relation « ensemble / parties » (transitive)
- L'agrégé survit à son agrégat

Agrégation

- Une forme d'inclusion
  - **Agrégat** : maintient de références/pointeurs vers des objets *agrégés*

Meilleur Exemple



Agrégation

- Une forme particulière
  - Agrégation

C++

```
class Player {};

using player_set = std::unordered_set<const Player*>;
class Team
{
public:
  player_set::size_type size () const;
  void hire (const Player& player);
  void fire (const Player& player);

private:
  player_set members;
};

player_set::size_type Team::size () const { return members.size (); }

void Team::hire (const Player& player) { members.insert (&player); }
void Team::fire (const Player& player) { members.erase (&player);  }
```

Agrégation

- Une forme d'inclusion
  - **Agrégat** : maintient de références/pointeurs vers des objets *agrégés*

Java

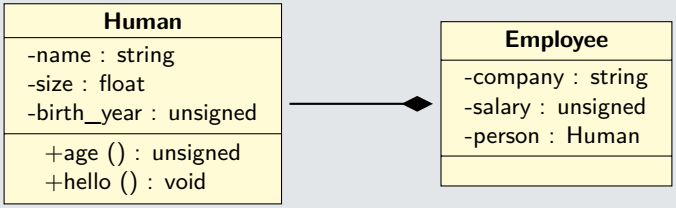
```java
public class Player {}

public class Team
{
  public Team () { members = new HashSet<Player> (); }

  public int size () { return members.size (); }

  public void hire (Player player) { members.add (player); }
  public void fire (Player player) { members.remove (player); }

  private HashSet<Player> members;
}
```

Plan

Composition (Agrégation Composite)

- Une forme d'inclusion plus stricte
  - L'agrégé ne survit pas à son (unique) agrégat

Exemple



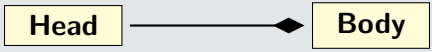
😕 Toujours pas très adapté dans ce cas

- L'accès à la partie `Human` est (toujours) indirect

Composition (Agrégation Composite)

- Une forme d'inclusion plus stricte
  - L'agrégé ne survit pas à son (unique) agrégat

Meilleur Exemple



Composition (Agrégation Composite)

- Une forme d'inclusion plus stricte
  - L'agrégé ne survit pas à son (unique) agrégat

Meilleur Exemple

C++

```
class Head {};
class Arm {};
class Leg {};

class Body
{
private:
  Head head;
  Arm arms[2];
  Leg legs[2];
};
```

Composition (Agrégation Composite)

- Une forme d'inclusion plus stricte
  - L'agrégé ne survit

Meilleur Exem

Java

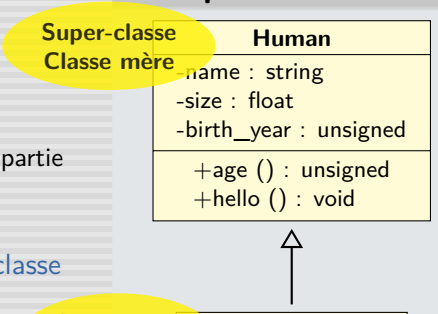
```java
public class Head {}
public class Arm {}
public class Leg {}

public class Body
{
  public Body ()
  {
    head = new Head ();
    arms = new Arm[2]; // ...
    legs = new Leg[2]; // ...
  }

  private Head head;
  private Arm[] arms;
  private Leg[] legs;
}
```

Plan

Héritage / Dérivation

- Une forme d'inclusion *encore* plus stricte
  - Contenu agrégé directement incorporé à la classe
- 🙂 Solution la plus adaptée ici
  - Pas de risque lié à la duplication manuelle
  - Pas d'objet intermédiaire (agrégat)
  - Le contenu de la classe `Human` fait *implicitement* partie de la classe `Employee` (sauf les constructeurs / destructeurs)
- La classe `Employee` « hérite » ou « dérive » de la classe `Human`
- 😉 Pas très loin d'un copier-coller (automatique)

Exemple



Héritage / Dérivation

- Une
  - Solu
  - La
  - Hum
- Pas

C++

```
class Employee : public Human
{
public:
  Employee (const std::string& name, float size, unsigned birth_year,
            const std::string& company, unsigned salary);
  ~Employee ();

private:
  std::string company_;
  unsigned salary_;
};

Employee::Employee (const std::string& name, float size, unsigned birth_year,
                    const std::string& company, unsigned salary)
  : Human (name, size, birth_year), company_ (company), salary_ (salary)
{}

Employee::~Employee ()
{}
```

Héritage / Dérivation

- Une forme d'inclusion *encore* plus stricte
  - Contenu agrégé directement incorporé à la classe

Solu

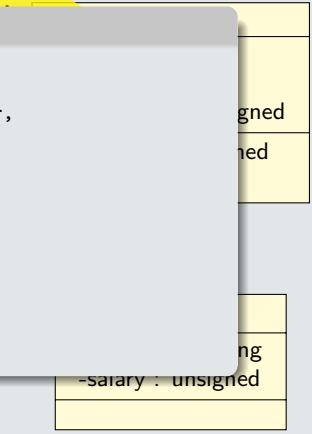
- La
  Hum

Pas

Java

```
public class Employee extends Human
{
  public Employee (String _name, float _size, int _birthYear,
                   String _company, int _salary)
  {
    super (_name, _size, _birthYear);
    company = _company;
    salary = _salary;
  }

  private String company;
  private int salary;
}
```

Exemple

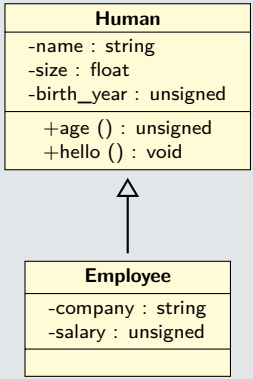


Plan

Caractéristiques de l'Héritage

- **Agrégation** : relation de type « a un »
  - Une équipe *a des* joueur, un corps *a une* tête *etc.*
- **Héritage** : relation de type « est un »
  - Un employé *est un* humain
- **Conséquence**
  - L'héritage *ressemble* à du sous-typage
    - Tout employé peut être vu comme un simple humain
    - La classe réelle d'un objet n'est plus nécessairement connue à la compilation
  - ⚠ Mais ressemblance seulement !
    Cf. principe de substitution de Liskov [Liskov, 1988]

Exemple



Caractéristiques de l'Héritage

- Agr… (partially hidden)
- Hér…
- Cor…

C++

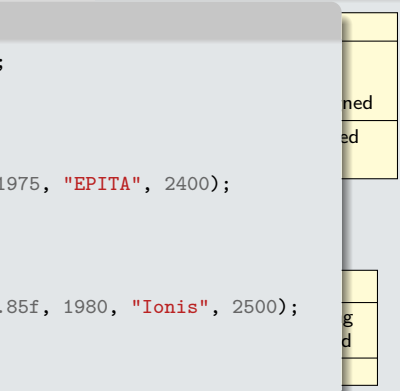
```
auto h = Human { "Alain Térieur", 1.80, 1970 };
h.census ();
h.hello ();
birth_control (h);
std::cout << std::endl;

auto e = Employee { "Alex Térieur", 1.78, 1975, "EPITA", 2400 };
e.census ();
e.hello ();
birth_control (e);
std::cout << std::endl;

Human* incognito = new Employee ("Vladimir Guez", 1.85, 1980, "Ionis", 2500);
incognito->census ();
incognito->hello ();
birth_control (*incognito);
std::cout << std::endl;
```


Build 2025-05-21 19:06:52+02:00

Caractéristiques de l'Héritage

- **Agrégation** : relation de type « a un »
  - [illegible]
- **Hér**
  - [illegible]
- **Cor**
  - [illegible]

Exemple



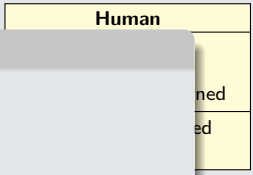
Java

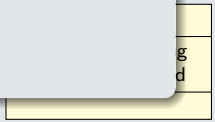
```java
Human h = new Human ("Alain Térieur", 1.80f, 1970);
h.census ();
h.hello ();
System.out.println ();

Employee e = new Employee ("Alex Térieur", 1.78f, 1975, "EPITA", 2400);
e.census ();
e.hello ();
System.out.println ();

Human incognito = new Employee ("Vladimir Guez", 1.85f, 1980, "Ionis", 2500);
incognito.census ();
incognito.hello ();
System.out.println ();
```

Caractéristiques de l'Héritage

- **Agrégation** : relation de type « a un »
  - Une équipe *a des* joueur, un corps *a une* tête *etc.*
- **Hér**
  -
- **Cor**
  -

Exemple





C++

```
const Human& unpredictable ()
{
  static const auto h
    = Human { "Corinne Titgoutte", 1.68, 1985 };
  static const auto e
    = Employee { "Justine Titgoutte", 1.83, 1990, "EPITA", 2600 };

  return (rand () % 2) ? e : h;
}

const Human& dont_know = unpredictable ();
```

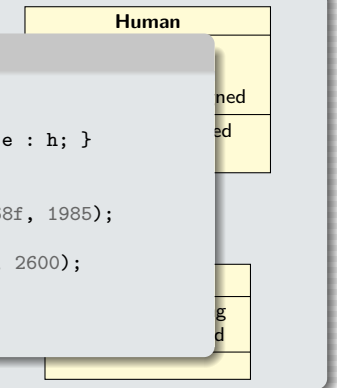
Caractéristiques de l'Héritage

- **Agrégation** : relation de type « a un »
  - Une équipe *a des* joueur, un corps *a une* tête *etc.*
- **Héri**
- **Cor**

Java

```
public class Unpredictable
{
  public static Human get () { return random.nextBoolean () ? e : h; }

  private static Random random = new Random ();
  private static Human h = new Human ("Corinne Titgoutte", 1.68f, 1985);
  private static Employee e
    = new Employee ("Justine Titgoutte", 1.83f, 1990, "EPITA", 2600);
}

Human dontKnow = Unpredictable.get ();
```

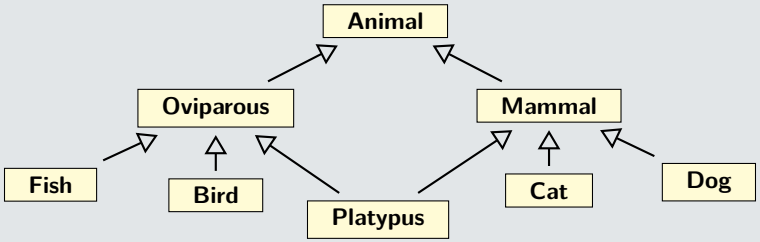
Exemple



Plan

Hiérarchies de Classes

Exemple



- L'héritage est une relation transitive
- Héritage multiple (pas toujours disponible) : plusieurs super-classes
- Hiérarchie de classes : arbre (ou graphe) orienté d'héritage

Hiérarchies de Classes

Exemple

C++

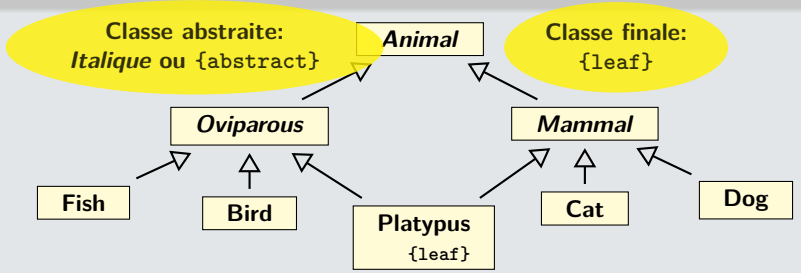
```
class Animal {};

class Oviparous : public Animal {};
class Bird : public Oviparous {};
class Fish : public Oviparous {};

class Mammal : public Animal {};
class Cat : public Mammal {};
class Dog : public Mammal {};

class Platypus : public Oviparous, public Mammal {};
```

- L'héritage es
- Héritage mul
- Hiérarchie de

Plan

Politiques d'Instanciation

Exemple



- **Classe abstraite :** non instanciable
- **Classe finale :** non dérivable

  Intérêts techniques supplémentaires : sûreté, performance

Politiques d'Instanciation

Exe

C++

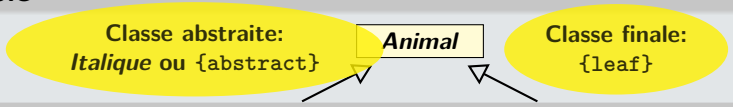
```cpp
class Animal
{
public:
  // Make this class abstract.
  virtual void cry () const = 0;
};

// Also abstract.
class Mammal : public Animal {};
class Cat : public Mammal
{
public:
  // Not abstract anymore.
  void cry () const override { std::cout << "Meow! Meow!\n"; };
};
```

- Cla
- Cla

Inté

Politiques d'Instanciation

Exemple



Java

```
public abstract class Animal {}
public abstract class Mammal extends Animal {}
public class Cat extends Mammal
{
  void cry () { System.out.println ("Meow! Meow!"); }
}
```

- **Cla**
- **Classe finale :** non dérivable
  Intérêts techniques supplémentaires : sûreté, performance

Politiques d'Instanciation

Exemple

C++

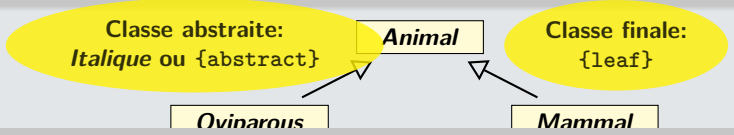
```cpp
// Final class (C++11).
class Platypus final : public Oviparous, public Mammal
{
public:
  // Not abstract anymore.
  void cry () const override { std::cout << "Platty! Platty!\n"; };
};
```

- **Cla**
- **Classe finale :** non dérivable

Intérêts techniques supplémentaires : sûreté, performance

Politiques d'Instanciation

Exemple



Java

```
public final class Platypus extends Animal
{
  void cry () { System.out.println ("Platty! Platty!"); }
}
```

- **Classe abstraite :** non instanciable
- **Classe finale :** non dérivable

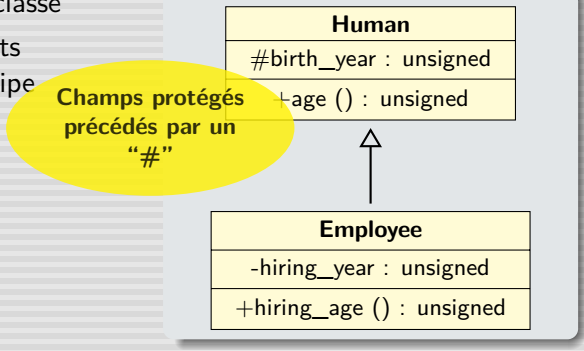
  Intérêts techniques supplémentaires : sûreté, performance

Plan

Accessibilité et Héritage

- **Attribut / Méthode Protégé(e) :**
  accès restreint à la sous-hiérarchie de la classe
- **Remarque :** Interface publique + attributs protégés = forme la plus proche du principe de sous-typage

Exemple

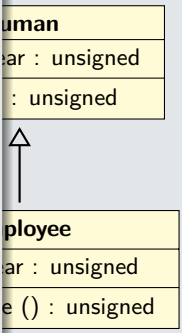


Accessibilité et Héritage

- **Attribut / M**
  accès restrein
- **Remarque :**
  protégés = fo
  de sous-typag

C++

```
class Human
{
protected:
  const unsigned birth_year_;
};

class Employee : public Human
{
public:
  unsigned hiring_age () const;

private:
  unsigned hiring_year_;
};

unsigned Employee::hiring_age () const
{
  return hiring_year_ - birth_year_;
}
```



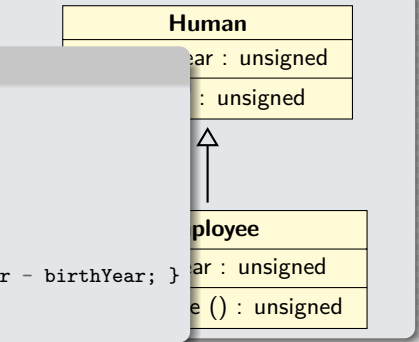
Accessibilité et Héritage

- **Attribut / Méthode Protégé(e) :**
  accès restreint à la sous-hiérarchie de la classe
- **Remarque :** Interface publique + attributs protégés = fo
  de sous-typag

Java

```
public class Human
{
  protected final int birthYear;
}

public class Employee extends Human
{
  public int hiringAge () { return hiringYear - birthYear; }
  private int hiringYear;
}
```

Exemple



Plan

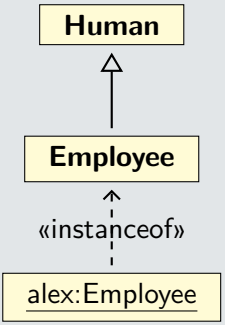
Plan

Héritage et Instanciation

⚠ Manipuler la relation « est un » avec précaution

- Un employé est un (type d')humain
- Alex est un employé (en particulier)

Exemple

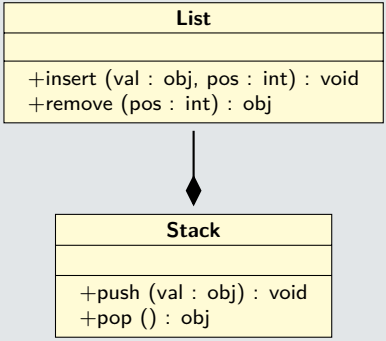


Plan

Ambivalence de l'Héritage

- Problèmes :
  1. Exposition de l'implémentation
  2. Héritage de l'interface de liste
- Deux effets du sous-classage :
  1. Héritage d'implémentation
     ré-utilisabilité du code
  2. Héritage d'interface
     sémantique, sous-typage
- ⚠ L'héritage d'implémentation entraîne celui de l'interface
- Préférer la composition à l'héritage
  Une pile n'est pas une liste

Exemple



Remarque : Héritage « Privé »

C++

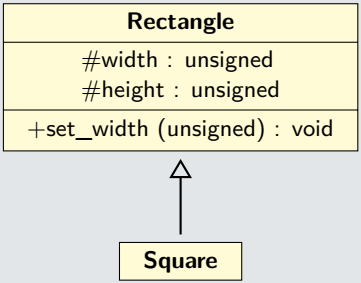
```
class List
{
public:
  void insert (obj val, int pos);
  obj remove (int pos);
};

class Stack : ~~public~~ private List
{
public:
  void push (obj val) { insert (val, 0); };
  obj pop () { return remove (0); }
};
```

- Relation de type « est implémenté en termes de »
- Privilégier la composition en général

Héritage par Restriction

- Le problème du carré/rectangle (cercle/elliptique)
  - Un carré est un rectangle...
  - ...mais avec des contraintes statiques...
  - ...et dynamiques
- Programmation Différentielle :
  - Hériter de manière additive et non restrictive
  - Problème surtout lié à la mutation
- Cf. principe de substitution de Liskov [Liskov, 1988]

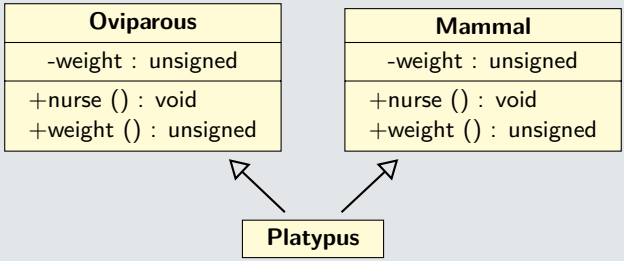
Exemple



Plan

Ambiguïtés de l'Héritage Multiple

Exemple



- Quelle(s) méthode(s) choisir (`nurse`) ?
- Pourquoi y en aurait-il plusieurs (`weight`) ?
- Remarques tout aussi valables pour les attributs

Ambiguïtés de l'Héritage Multiple

Exe

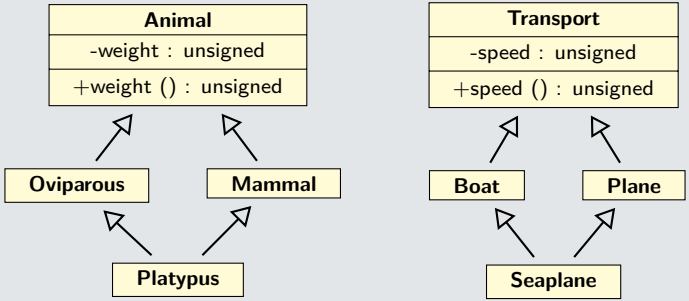
C++

```
class Oviparous
{
public:
  void nurse () const { std::cout << "I brood my eggs.\n"; }
};

class Mammal
{
public:
  void nurse () const { std::cout << "I suckle my offsprings.\n"; }
};

class Platypus final : public Oviparous, public Mammal {};

Platypus platty;
platty.Oviparous::nurse ();
platty.Mammal::nurse ();
```

- Quelle(s
- Pourquo
- Remarqu

Héritage en Diamant / Losange

Exemple



- Combien de copies de la classe de base veut-on ?
- Pourquoi raisonner au niveau classe ?
- ⚠ Chaque langage a son propre point de vue...

Héritage en Diamant / Losange

Exemp

Héritage « partagé »

```
class Animal
{
public:
  Animal (unsigned weight) : weight_ (weight) {};

private:
  unsigned weight_;
};

class Oviparous : public virtual Animal {};
class Mammal    : public virtual Animal {};

class Platypus final : public Oviparous, public Mammal
{
public:
  Platypus (unsigned weight) : Animal (weight) {};
};
```

- Combien de
- Pourquoi rais
- ⚠ Chaque langa

Héritage « répété »

```
class Transport
{
public:
  Transport (unsigned speed) : speed_ (speed) {};
private:
  unsigned speed_;
};
class Boat : public Transport
{
public:
  Boat (unsigned speed) : Transport (speed) {};
};
class Plane : public Transport
{
public:
  Plane (unsigned speed) : Transport (speed) {};
};
class Seaplane : public Boat, public Plane
{
public:
  Seaplane (unsigned boat_speed, unsigned plane_speed)
    : Boat (boat_speed), Plane (plane_speed)
  {};
};
```

- Combien de
- Pourquoi rais
- ⚠ Chaque langa

Plan

Bibliographie

Bibliographie

Barbara Liskov.
Data Abstraction and Hierarchy.
OOPSLA'87 Keynote, 1988.