

Approches Objet de la Programmation

~ Surcharge, Masquage, Ré-écriture ~

Didier Verna

EPITA / LRDE

didier@lrde.epita.fr



lrde/~didier



@didierverna



didier.verna



in/didierverna

Plan

Polymorphisme Statique

Surcharge
Masquage

Polymorphisme Dynamique

Ré-écriture
Surcharge, Masquage, Ré-écriture
Méthodes Abstraites

Relation (sous-)Classe / (sous-)Type

Rappels
Covariance / Contravariance
Principe de Substitution de Liskov

Plan

Polymorphisme Statique

Surcharge
Masquage

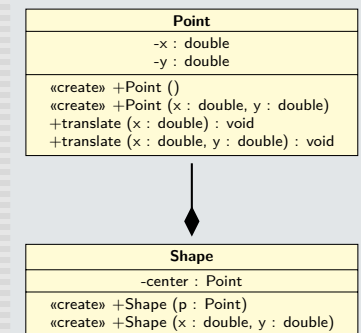
Polymorphisme Dynamique

Relation (sous-)Classe / (sous-)Type

Surcharge

- ▶ Même nom, signature différente
 - ▶ Type / nombre de paramètres
 - ▶ Éventuellement, type de retour
- ▶ Constructeurs, méthodes
 - ▶ Y compris méthodes statiques
 - ▶ fonctions, opérateurs
- ▶ Utilité : faire...
 - ▶ ...des choses un peu différentes
 - ▶ ...la même chose différemment
- ▶ Polymorphisme intra-classe (ad-hoc)
- ▶ Avantage : dispatch statique
compilation, performance

Exemple UML



Polymorphisme Statique Polymorphisme Dynamique Relation Classe / Type

Surcharge

- ▶ Même nom, signature différente
 - ▶ Type / nombre de paramètres

Exemple UML

C++

```
class Point
{
public:
    // Constructor overloading
    Point () : Point (0, 0) {};
    Point (double x, double y) : x_ (x), y_ (y) {};

    // Method overloading
    void translate (double x) { x_ += x; };
    void translate (double x, double y) { x_ += x; y_ += y; };

private:
    double x_, y_;
};
```

compilation, performance

AOP / Surcharge, Masquage, Ré-écriture – Didier Verna 4/15

Polymorphisme Statique Polymorphisme Dynamique Relation Classe / Type

Surcharge

- ▶ Même nom, signature différente
 - ▶ Type / nombre de paramètres

Exemple UML

Java

```
public static class Point
{
    // Constructor overloading
    public Point () { this (0, 0); }
    public Point (double _x, double _y) { x = _x; y = _y; }

    // Method overloading
    public void translate (double _x) { x += _x; }
    public void translate (double _x, double _y) { x += _x; y += _y; }

    private double x, y;
}
```

compilation, performance

AOP / Surcharge, Masquage, Ré-écriture – Didier Verna 4/15

Polymorphisme Statique Polymorphisme Dynamique Relation Classe / Type

Masquage

- ▶ Polymorphisme inter-classe
- ▶ Utilité : identique à la surcharge
- ▶ Même nom, signature différente ou identique
 - ▶ Classe = facteur de distinction
- ▶ Y compris méthodes statiques
- ▶ Avantage : dispatch statique
compilation, performance
- ▶ Inconvénient : dispatch statique...

Exemple UML

```
classDiagram
    class Human {
        -name : string
        -size : float
        -birth_year : unsigned
        +hello () : void
    }
    class Employee {
        -company : string
        -salary : unsigned
        +hello (details : bool) : void
        +hello () : void
    }
    Human <|-- Employee
```

AOP / Surcharge, Masquage, Ré-écriture – Didier Verna 5/15

Polymorphisme Statique Polymorphisme Dynamique Relation Classe / Type

Masquage

- ▶ Polymorphisme inter-classe

Exemple UML

C++

```
void Human::hello () const
{
    std::cout << "Hello! I'm " << name_ << ", "
               << size_ << "m, "
               << age () << "yo.\n";
}

void Employee::hello (bool details) const
{
    Human::hello ();
    if (details)
        std::cout << "Working at " << company_ << " for " << salary_ << "€,"
                  << "started at the age of " << hiring_age () << ".\n";
}

void Employee::hello () const { hello (true); }
```

AOP / Surcharge, Masquage, Ré-écriture – Didier Verna 5/15

Polymorphisme Statique Polymorphisme Dynamique Relation Classe / Type

Masquage

- Polymorphisme inter-classe
- Utilité : identique à la surcharge

Exemple UML

```

classDiagram
    class Human {
        +hello (details : bool) : void
        +hello () : void
    }

```

C++

```

auto h = Human { ... };
h.hello (); // Human::hello

auto e = Employee { ... };
e.hello (); // Employee::hello

Human* incognito = new Employee (...);
incognito->hello (); // Human::hello !

const Human& dont_know = unpredictable ();
dont_know.hello (); // Human::hello !

```

AOP / Surcharge, Masquage, Ré-écriture – Didier Verna 5/15

Polymorphisme Statique Polymorphisme Dynamique Relation Classe / Type

Plan

Polymorphisme Statique

Polymorphisme Dynamique

- Ré-écriture
- Surcharge, Masquage, Ré-écriture
- Méthodes Abstraites

Relation [surc] / Classe / [surc] / Type

AOP / Surcharge, Masquage, Ré-écriture – Didier Verna 6/15

Polymorphisme Statique Polymorphisme Dynamique Relation Classe / Type

Ré-écriture

- « We also needed to group together common process properties in such a way that they could be applied later, in a variety of different situations not necessarily known in advance. » [Dahl, 1978]
- Polymorphisme intra-hiérarchie (d'inclusion)
- Utilité : Cf. surcharge / masquage
- Même nom, même signature
- Avantage : dispatch dynamique
- Inconvénient : coût en performance
- Méthodes « virtuelles »

Exemple UML

```

classDiagram
    class Human {
        -name : string
        -size : float
        -birth_year : unsigned
        +hello () : void {virtual}
    }
    class Employee {
        -company : string
        -salary : unsigned
        +hello () : void {virtual}
    }
    Human <|-- Employee

```

AOP / Surcharge, Masquage, Ré-écriture – Didier Verna 7/15

Polymorphisme Statique Polymorphisme Dynamique Relation Classe / Type

Ré-écriture

- « We also needed to group together common process properties in such a way that they

Exemple UML

```

classDiagram
    class Human {
        +hello () : void {virtual}
    }
    class Employee {
        +hello () : void {virtual}
    }
    Human <|-- Employee

```

C++

```

class Human
{
public:
    virtual void hello () const { ... };
};

class Employee : public Human
{
public:
    void hello (bool details) const { ... };
    void hello () const override { ... };
};

```

- Méthodes « virtuelles »

AOP / Surcharge, Masquage, Ré-écriture – Didier Verna 7/15

Polymorphisme Statique Polymorphisme Dynamique Relation Classe / Type

Ré-écriture

Java

```
public class Human
{
    public void hello ()
    {
        System.out.println ("Hello! I'm " + name + ", " + size + "m, "
            + age () + "yo.");
    }
}

public class Employee extends Human
{
    public void hello (boolean details)
    {
        super.hello ();
        if (details)
            System.out.println ("Working at " + company + " for " + salary + "€, "
                + "started at the age of " + hiringAge () + ".");
    }

    @Override public void hello () { hello (true); }
}
```

AOP / Surcharge, Masquage, Ré-écriture – Didier Verna 7/15

Polymorphisme Statique Polymorphisme Dynamique Relation Classe / Type

Surcharge, Masquage, Ré-écriture

- ▶ Propagation de la surcharge : langage-dépendant
 - ▶ Java : oui
 - ▶ C++ : non par défaut *masquage intégral*, Cf. *using*
 - ▶ Indépendant du caractère virtuel ou statique
- ▶ Attention aux conversions de type !
- ▶ Masquage ≠ ré-écriture
 - ▶ Statique vs. dynamique
 - ▶ Java : méthodes statiques uniquement même signature
 - ▶ C++ : méthodes statiques et non virtuelles signatures identiques ou différentes

Exemple UML

```

classDiagram
    class Shape {
        -center : Point
        +translate (x : double, y : double) : void
    }
    class Circle {
        +translate (p : Point) : void
        +translate (x : int, y : int) : void
    }
    Shape <|-- Circle
    
```

AOP / Surcharge, Masquage, Ré-écriture – Didier Verna 8/15

Polymorphisme Statique Polymorphisme Dynamique Relation Classe / Type

Surcharge, Masquage, Ré-écriture

- ▶ Propagation de la surcharge : langage-dépendant

Exemple UML

```

classDiagram
    class Shape {
        +translate (double x) : void
        +translate (double x, double y) : void
    }
    class Circle {
        +translate (Point p) : void
    }
    Shape <|-- Circle
    
```

C++

```
class Shape
{
public:
    void translate (double x) { center_.translate (x); };
    void translate (double x, double y) { center_.translate (x, y); };
};

class Circle : public Shape
{
public:
    using Shape::translate;
    void translate (Point p) { center_ = p; }
};
```

signatures identiques ou différentes

AOP / Surcharge, Masquage, Ré-écriture – Didier Verna 8/15

Polymorphisme Statique Polymorphisme Dynamique Relation Classe / Type

Surcharge, Masquage, Ré-écriture

- ▶ Propagation de la surcharge : langage-dépendant
 - ▶ Java : oui

Exemple UML

```

classDiagram
    class Shape {
    }
    class Circle {
    }
    Shape <|-- Circle
    
```

Java

```
public class Shape
{
    public void translate (double x) { center.translate (x); }
    public void translate (double x, double y) { center.translate (x, y); }

    public static class Circle extends Shape
    {
        public void translate (Point p) { center = p; } // Not a copy!
    }
}
```

- ▶ C++ : méthodes statiques et non virtuelles signatures identiques ou différentes

AOP / Surcharge, Masquage, Ré-écriture – Didier Verna 8/15

Polymorphisme Statique Polymorphisme Dynamique Relation Classe / Type

Méthodes abstraites

Exemple UML

```

classDiagram
    class ShapeShape {
        -center : Point
        +translate(x : double, y : double) : void
        +scale(a : double) : void
    }
    class Circle {
        -radius : double
        +scale(a : double) : void
    }
    class Rectangle {
        +scale(a : double) : void
    }
    class Triangle {
        +scale(a : double) : void
    }
    ShapeShape <|-- Circle
    ShapeShape <|-- Rectangle
    ShapeShape <|-- Triangle
  
```

Méthode abstraite: Italique

- ▶ Méthodes *déclarées*, mais sans implémentation initiale
a.k.a. « virtuelle pure »
- ▶ Implémentation nécessaire dans les sous-classes
- ▶ Méthode(s) abstraite(s) $\Rightarrow$ Classe abstraite

AOP / Surcharge, Masquage, Ré-écriture – Didier Verna 9/15

Polymorphisme Statique Polymorphisme Dynamique Relation Classe / Type

Méthodes abstraites

Exemple UML

C++

```

class Shape
{
public:
    virtual void scale (double factor) = 0;
};

class Circle : public Shape
{
public:
    void scale (double factor) override { radius_ *= factor; };
private:
    double radius_;
};
  
```

- ▶ Méthode(s) abstraite(s) $\Rightarrow$ Classe abstraite

AOP / Surcharge, Masquage, Ré-écriture – Didier Verna 9/15

Polymorphisme Statique Polymorphisme Dynamique Relation Classe / Type

Méthodes abstraites

Exemple UML

Java

```

public abstract class Shape
{
    public abstract void scale (double factor);
}

public static class Circle extends Shape
{
    @Override
    public void scale (double factor) { radius *= factor; };
private double radius;
}
  
```

- ▶ Implémentation nécessaire dans les sous-classes
- ▶ Méthode(s) abstraite(s) $\Rightarrow$ Classe abstraite

AOP / Surcharge, Masquage, Ré-écriture – Didier Verna 9/15

Polymorphisme Statique Polymorphisme Dynamique Relation Classe / Type

Plan

Polymorphisme Statique

Polymorphisme Dynamique

Relation (sous-)Classe / (sous-)Type

- Rappels
- Covariance / Contravariance
- Principe de Substitution de Liskov

AOP / Surcharge, Masquage, Ré-écriture – Didier Verna 10/15

Polymorphisme Statique Polymorphisme Dynamique Relation Classe / Type

Rappels sur la Notion d'Héritage

- Relation de type « est un »
 - Un objet d'une sous-classe peut être vu comme d'un objet d'une super-classe
- Ambivalence
 - Héritage d'implémentation
 - Héritage d'interface (entraîné par l'héritage d'implémentation)
- Principe de substituabilité
 - Si $S <: T$, alors tout terme de type S peut être utilisé en toute sécurité dans un contexte où un terme de type T est attendu
 - Pour une certaine définition de « en toute sécurité » et « contexte » ...
- Relation forte entre héritage (sous-classage) et sous-typage

AOP / Surcharge, Masquage, Ré-écriture – Didier Verna 11/15

Polymorphisme Statique Polymorphisme Dynamique Relation Classe / Type

Application aux Méthodes

Exemple UML

```

classDiagram
    License <|-- TruckLicense
    Driver <|-- TruckDriver
    Vehicle <|-- Car
    Vehicle <|-- Truck
    Parking <|-- TruckParking
    Driver --> License : +license() : License
    Driver --> Car : +drive(vehicle : Car) : void
    Driver --> Parking : +provide(parking : Parking) : void
    TruckDriver --> License : +license() : TruckLicense
    TruckDriver --> Vehicle : +drive(vehicle : Vehicle) : void
    TruckDriver --> TruckParking : +provide(parking : TruckParking) : void
  
```

The diagram illustrates the relationship between classes and their methods, highlighting covariance and contravariance. Covariance is shown where a subclass method returns a more specific type (e.g., `TruckLicense` instead of `License`). Contravariance is shown where a subclass method accepts a more general type (e.g., `Vehicle` instead of `Car`).

AOP / Surcharge, Masquage, Ré-écriture – Didier Verna 12/15

Polymorphisme Statique Polymorphisme Dynamique Relation Classe / Type

Principe de Substitution de Liskov

- Principe de substituabilité appliqué aux objets
- Sous-typage *comportemental fort*
 - Soit $\phi(x)$ une propriété prouvable sur les objets x de type T . Alors, $\phi(y)$ doit être vraie pour tout objet y de type S tel que $S <: T$.
- Covariance des types de retour dans S
- Contravariance des types d'arguments dans S
- Les exceptions levées dans S doivent être des sous-types de celles levées dans T
- Les invariants de T doivent être préservés dans S
- Les pré-conditions ne peuvent pas être renforcées dans S
- Les post-conditions ne peuvent pas être affaiblies dans S
- Contrainte historique* : la mutation n'a lieu qu'à travers l'interface. Aucune méthode de S ne doit permettre des mutations interdites dans T .

AOP / Surcharge, Masquage, Ré-écriture – Didier Verna 13/15

Bibliographie

Plan

Bibliographie

AOP / Surcharge, Masquage, Ré-écriture – Didier Verna 14/15

Bibliographie

-  Ole-Johan Dahl and Kristen Nygaard.
The Development of the SIMULA Languages.
History of Programming Languages Conference, 1978.
-  Barbara Liskov.
Data Abstraction and Hierarchy.
OOPSLA'87 Keynote, 1988.

