Object-Oriented Approaches to Programming

Overloading, Masking, Overriding

Didier Verna

didier@didierverna.net


didierverna.net


@didierverna


didier.verna


in/didierverna

Build 2024-11-11 11:29:31+01:00

Outline

Static Polymorphism
- Overloading
- Masking

Dynamic Polymorphism
- Overriding
- Overloading, Masking, Overriding
- Abstract Methods
- Corollary: Java Interfaces

(Sub)class / (Sub)Type Relation
- Reminder
- Covariance / Contravariance
- Liskov Substitution Principle

 Build 2024-11-11 11:29:31+01:00

Plan

Static Polymorphism
- Overloading
- Masking

Dynamic Polymorphism
- Overriding
- Overloading, Masking, Overriding
- Abstract Methods
- Corollary: Java Interfaces

(Sub)class / (Sub)Type Relation
- Reminder
- Covariance / Contravariance
- Liskov Substitution Principle

Plan

Static Polymorphism
- Overloading
- Masking

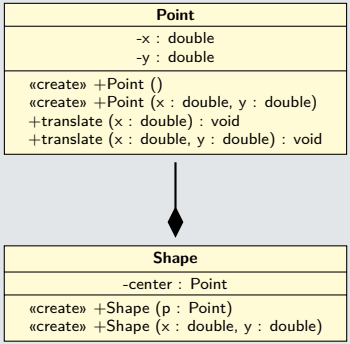
Dynamic Polymorphism
- Overriding
- Overloading, Masking, Overriding
- Abstract Methods
- Corollary: Java Interfaces

(Sub)class / (Sub)Type Relation
- Reminder
- Covariance / Contravariance
- Liskov Substitution Principle

Overloading

- Same name, different signature
  - Type / number of parameters
  - Possibly, return type
- Constructors, methods
  - including static methods
  - functions, operators
- Usage:
  - do slightly different things
  - do something in slightly different ways
- "*Ad hoc*" polymorphism (intra-class)
  [Strachey, 1967]
- Advantage: static dispatch
  compilation, performance

Example



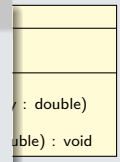
Overloading

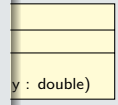
- Same name, different signature
  - Typ
  - Pos
- Construc
  - incl
  - func
- Usage:
  - do s
  - do s
- "*Ad hoc*
  [Strachey,
- Advanta
  compilation, performance

Example

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_;
};
```

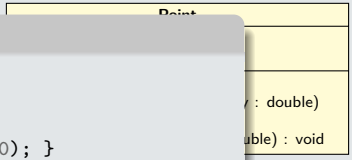




Overloading

- Same name, different signature
  - Type / number of parameters
  - Pos
- Construc
  - incl
  - func
- Usage:
  - do s
  - do s
- "*Ad hoc*
  [Strachey,
- Advantage: static dispatch
  compilation, performance

Example



Java

```java
public 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;
}
```

Plan

Static Polymorphism
- Overloading
- Masking

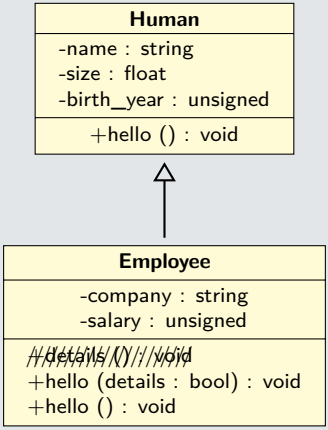
Dynamic Polymorphism
- Overriding
- Overloading, Masking, Overriding
- Abstract Methods
- Corollary: Java Interfaces

(Sub)class / (Sub)Type Relation
- Reminder
- Covariance / Contravariance
- Liskov Substitution Principle

Masking

- Inter-class polymorphism
- Usage: identical to overloading
- Same name, signature different or identical
  - Class = distinction factor
- Including static methods
- Advantage: static dispatch
  compilation, performance
- Inconvenient: static dispatch…😉

Example



Masking

- Inter-class polymorphism
- Usa
- San
- Incl
- Adv
- *con*
- Incc

C++

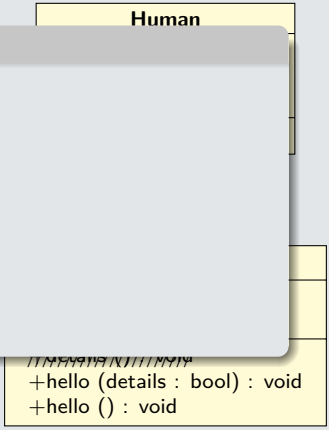
```
void Human::hello () const
{
  std::cout << "Hello! I'm " << name_ <<  ", "
            << size_ << "m, "
            << age () << "yo.\n";
}

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); }
```

Masking

- Inter-class polymorphism
- Usage: identical to overloading
- San
  - C++
- Incl
- Adv
  con
- Incc

```cpp
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 !
```

Example



Plan

Static Polymorphism
- Overloading
- Masking

Dynamic Polymorphism
- Overriding
- Overloading, Masking, Overriding
- Abstract Methods
- Corollary: Java Interfaces

(Sub)class / (Sub)Type Relation
- Reminder
- Covariance / Contravariance
- Liskov Substitution Principle

Plan

Static Polymorphism
- Overloading
- Masking

Dynamic Polymorphism
- Overriding
- Overloading, Masking, Overriding
- Abstract Methods
- Corollary: Java Interfaces

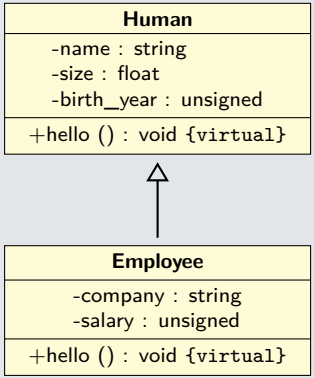
(Sub)class / (Sub)Type Relation
- Reminder
- Covariance / Contravariance
- Liskov Substitution Principle

 Build 2024-11-11 11:29:31+01:00

Overriding

- *"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]
- "Inclusion" polymorphism (intra-hierarchy)
- Usage: *cf.* overloading / masking
- Same name, same signature
- Advantage: dynamic dispatch
- Inconvenient: performance cost
- "Virtual" methods

Example



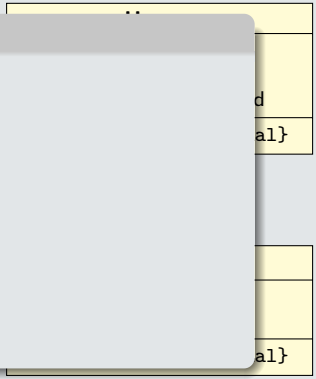
Overriding

- *"We also needed to group together common process properties in such a way that they could be applied later, in a va*
- *adva*
- "Inc
- Usa
- San
- Adv
- Incc
- "Vi

C++

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

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

Example



Overriding

- "We
  prop
  a va
  adva
- "Inc
- Usa
- San
- Adv
- Incc
- "Vi

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); }
}
```


Plan

Static Polymorphism
- Overloading
- Masking

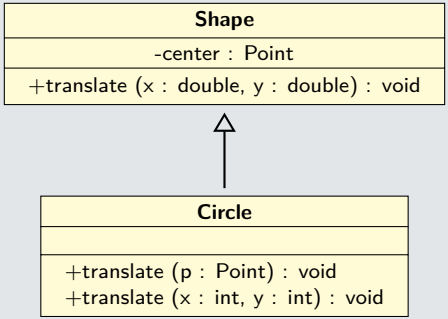
Dynamic Polymorphism
- Overriding
- Overloading, Masking, Overriding
- Abstract Methods
- Corollary: Java Interfaces

(Sub)class / (Sub)Type Relation
- Reminder
- Covariance / Contravariance
- Liskov Substitution Principle

Overloading, Masking, Overriding

- Propagation of overloading: langage-dependent
  - Java: yes
  - C++: no by default
    integral masking, cf. `using`
  - Independent from the static or virtual method status
- Pay attention to type conversions!
- Masking $\neq$ overriding
  - Static *vs.* dynamic
  - Java: static methods only
    same signature
  - C++: static *and* non-virtual methods
    identical or different signatures

Example



Overloading, Masking, Overriding

- Propagation of overloading: langage-dependent
  - C++
  - Pay
  - Mas
  - identical or different signatures

Example

```
) : void
```

```
void
```

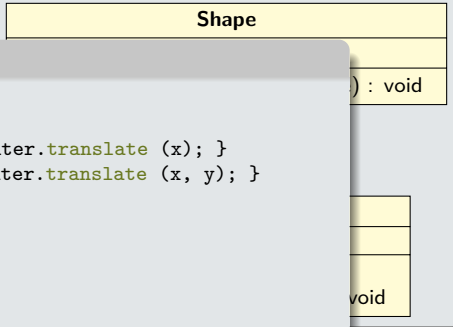
```
C++
class Shape
{
public:
  void translate (double x)           { center_.translate (x); };
  void translate (double x, double y) { center_.translate (x, y); };
};

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

Overloading, Masking, Overriding

- Propagation of overloading: langage-dependent
  - Java: yes
  - Java
  - Pay
  - Mas
  - same signature
  - C++: static *and* non-virtual methods
    identical or different signatures

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

public class Circle extends Shape
{
  public void translate (Point p) { center = p; }
}
```

Example



Plan

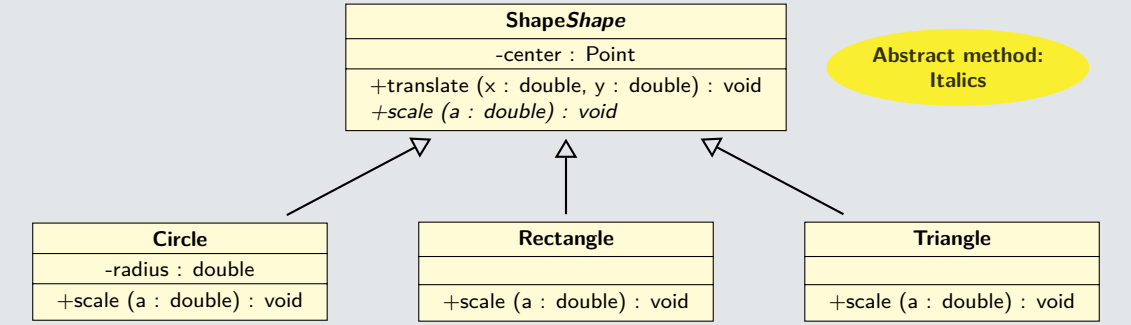
Static Polymorphism
- Overloading
- Masking

Dynamic Polymorphism
- Overriding
- Overloading, Masking, Overriding
- Abstract Methods
- Corollary: Java Interfaces

(Sub)class / (Sub)Type Relation
- Reminder
- Covariance / Contravariance
- Liskov Substitution Principle

Abstract Methods

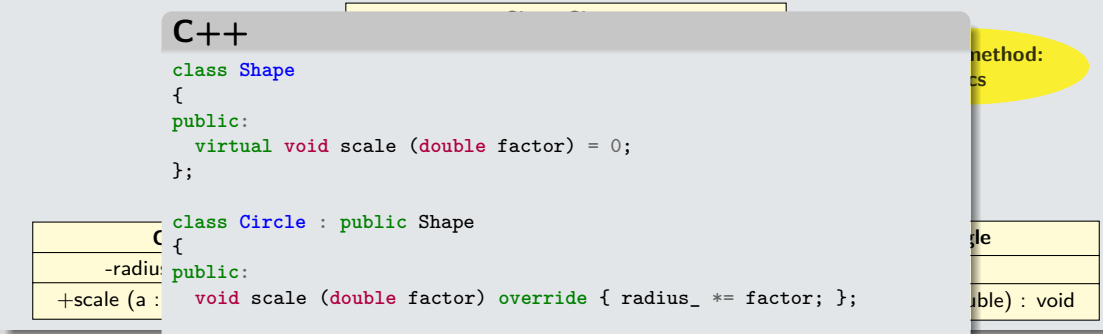
Example



- Methods *declared*, but without initial implementation (*a.k.a. "purely virtual"*)
- Implementation required in sub-classes
- Abstract Method(s) $\Longrightarrow$ Abstract Class

Abstract Methods

Example



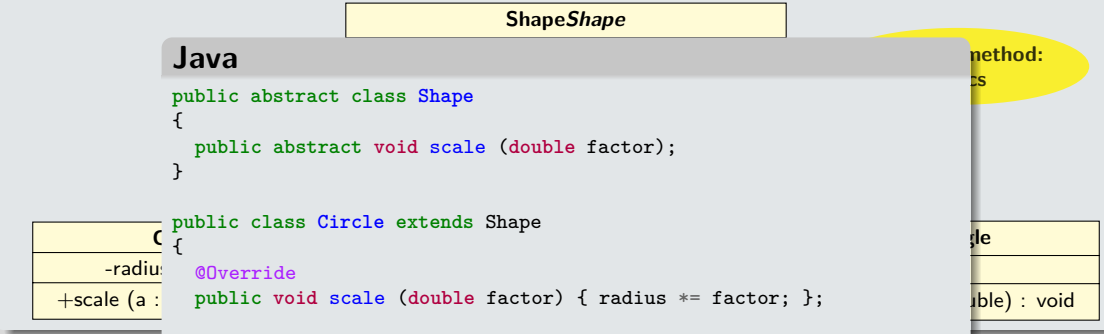
C++

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

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

private:
  double radius_;
};
```

- Methods
- Impleme
- Abstract Method(s) $\Longrightarrow$ Abstract Class

Abstract Methods

Example



Java

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

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

  private double radius;
}
```

- Methods [illegible] a/")
- Implementation required in sub-classes
- Abstract Method(s) $\Longrightarrow$ Abstract Class

Plan

Static Polymorphism
- Overloading
- Masking

Dynamic Polymorphism
- Overriding
- Overloading, Masking, Overriding
- Abstract Methods
- Corollary: Java Interfaces

(Sub)class / (Sub)Type Relation
- Reminder
- Covariance / Contravariance
- Liskov Substitution Principle

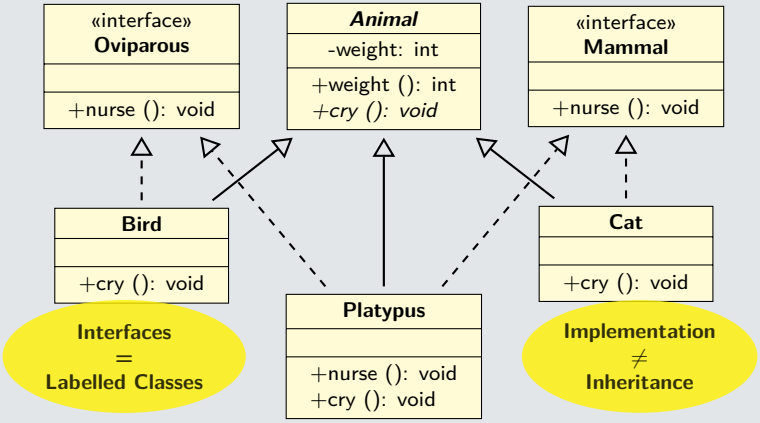
 Build 2024-11-11 11:29:31+01:00

Corollary: Java Interfaces

- Constants and abstract methods only
  - Everything is public (`abstract`, `public`, and `final` keywords are optional)
  - A class `implements` one or several interfaces
  - An interface `extends` one or several interfaces (diamond inheritance available)
- Since Java 8
  - Static Methods *not inheritable* (implementation required)
  - Default Methods *inheritable* (idem)
- Depuis Java 9
  - Private Methods *not inheritable*, static or not

Application

Example



Application

Exa

Java

```java
public interface Nurse
{
  public abstract void nurse ();
}
public interface Oviparous extends Nurse
{
  default void nurse ()
  {
    System.out.println ("I brood my eggs.");
  }
}
public interface Mammal extends Nurse
{
  default void nurse ()
  {
    System.out.println ("I suckle my offsprings.");
  }
}
```

Application

Example

«interface» *Animal* «interface»

Java

```
public class Bird extends Animal implements Oviparous
{
  @Override
  public void cry () { System.out.println ("Tweet! Tweet!"); }
}

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

= Labelled Classes

+nurse (): void
+cry (): void

≠ Inheritance

Application

Example

Java

```java
public final class Platypus extends Animal
  implements Oviparous, Mammal
{
  @Override
  public void cry () { System.out.println ("Platty! Platty!"); }

  // Required for disambiguation, even if never called (!= C++)
  @Override public void nurse ()
  {
    // Default method calls
    Oviparous.super.nurse ();
    Mammal.super.nurse ();
  }
}
```

+cry (): void

Plan

Static Polymorphism
- Overloading
- Masking

Dynamic Polymorphism
- Overriding
- Overloading, Masking, Overriding
- Abstract Methods
- Corollary: Java Interfaces

(Sub)class / (Sub)Type Relation
- Reminder
- Covariance / Contravariance
- Liskov Substitution Principle

Plan

Static Polymorphism
- Overloading
- Masking

Dynamic Polymorphism
- Overriding
- Overloading, Masking, Overriding
- Abstract Methods
- Corollary: Java Interfaces

(Sub)class / (Sub)Type Relation
- Reminder
- Covariance / Contravariance
- Liskov Substitution Principle

Reminder on the Concept of Inheritance

- "is a" relationship
  - Any object of a subclass can be seen as an object of a superclass
- Ambivalence
  - Implementation inheritance
  - Interface inheritance (consequence of implementation inheritance)
- Substitutability Principle
  - If $S <: T$, then every term of type S may be use safely in a context in which a term of type T is expected
  - 😕 For some definition of "safely" and "context"...
- Strong relation between inheritance (subclassing) et subtyping

Plan

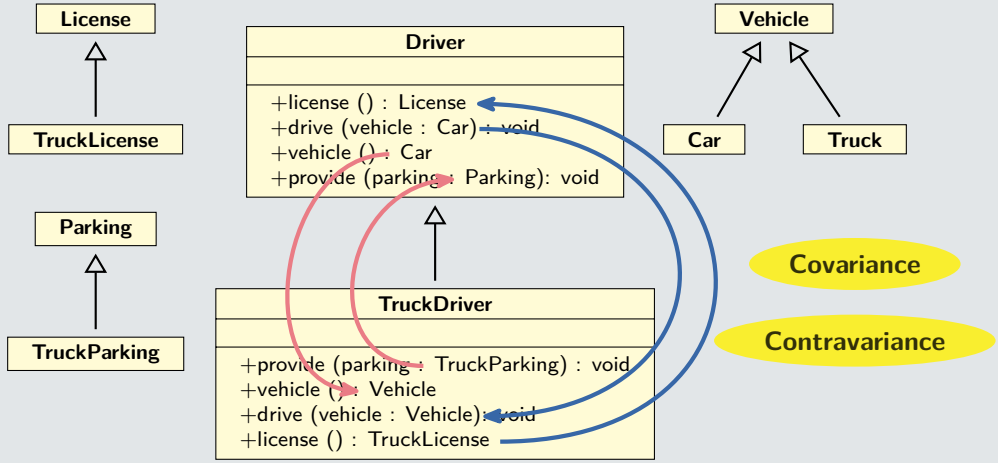
Static Polymorphism
- Overloading
- Masking

Dynamic Polymorphism
- Overriding
- Overloading, Masking, Overriding
- Abstract Methods
- Corollary: Java Interfaces

(Sub)class / (Sub)Type Relation
- Reminder
- Covariance / Contravariance
- Liskov Substitution Principle

Application to Methods

Example



Plan

Static Polymorphism
- Overloading
- Masking

Dynamic Polymorphism
- Overriding
- Overloading, Masking, Overriding
- Abstract Methods
- Corollary: Java Interfaces

(Sub)class / (Sub)Type Relation
- Reminder
- Covariance / Contravariance
- Liskov Substitution Principle

Liskov Substitution Principle

- Substitutability Principle applied to objects
- *Strong behavioral* subtyping

Let $\phi(x)$ be a provable property on every object x of type T.
Then, $\phi(y)$ must hold for every object y of type S such that $S <: T$.

- Covariance of return types in S
- Contravariance of argument types in S
- The exceptions raised in S must be subtypes of those raised in T
- The invariants of T must be preserved in S
- Pre-conditions cannot be stronger in S
- Post-conditions cannot be weaker in S
- *Historical constraint:* mutation can only occur through an interface. No method in S must allow mutations that are prohibited in T.

Plan

Bibliography

Bibliography

Christopher Strachey.
Fundamental Concepts in Programming Languages.
International Summer School in Computer Programming, Copenhagen, 1967.
Higher-Order and Symbolic Computation, 13(1–2): 11-–49, 2000.

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.