UML 2.0

Interfaces

UML 2.0

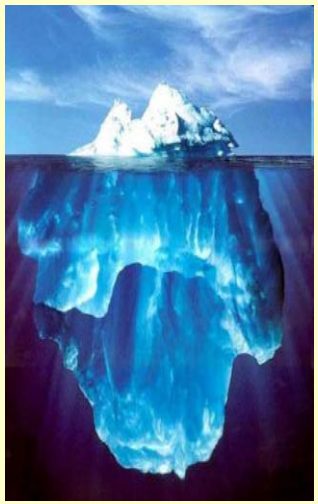
- Class diagrams (+ OCL constraints)
- Package diagrams
- Component diagrams
- Deployment diagrams
- Use case diagrams
- State diagrams
- Activity diagrams
- Interaction diagrams

UML 2.0

- ✓ Class diagrams (+ OCL constraints): back here!
- Package diagrams
- Component diagrams
- Deployment diagrams
- ✓ Use case diagrams
- ✓ State diagrams
- Activity diagrams
- ✓ Interaction diagrams

Encapsulation

- [Parnas72] Information hiding
  - The *secrets* of a module: design decisions that can be changed without affecting any other module
  - The assembly of a system made of module is usually best done if the interfaces are simple and well understood by all involved

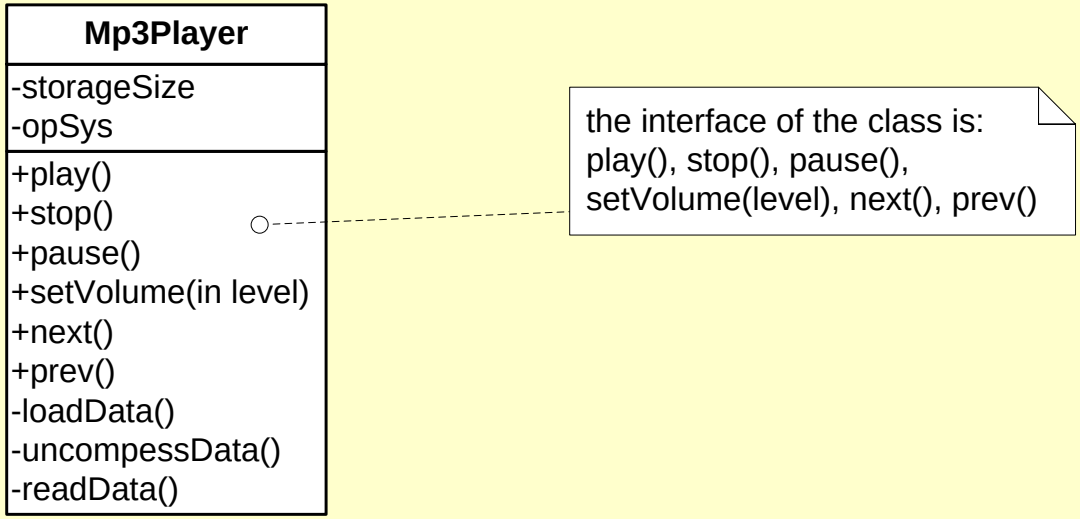


Interfaces

- figuratively: the emerged part of the iceberg!
- of an object (execution): the set of messages an object can receive
- of a class (design): the external view of a class, i.e. the set of its public methods and properties. Abstraction is increased by hiding inner implementation details
- UML: *an interface is a kind of classifier that represents a declaration of a set of coherent public features and obligations. An interface specifies a contract; any instance of a classifier that realizes the interface must fulfill that contract*

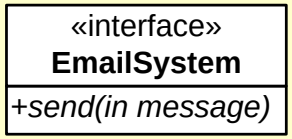
Interface of a class

- The interface of an object is the set of messages it can receive, i.e. the set of its public methods

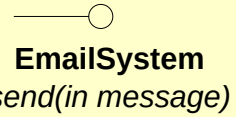


Interfaces in UML

- An interface is a kind of classifier that represents a declaration of a set of coherent public features and obligations.
- An interface specifies a contract: any instance of a classifier that realizes the interface must fulfill that contract



stereotype notation



« ball » or « lollipop » notation

Interfaces in UML (2)

- Interfaces declare (i.e. they do not implement!) a set of public operations that a class must offer
- They cannot be instantiated! ~~new Interface()~~
- Can be seen as pure abstract classes: **all** operations are abstract
- A class which inherits from an interface must implement **all** the operations declared by the interface
- A class can inherit from several interfaces
- Reminder: a class can inherit from only one (abstract) class

Interfaces vs. Abstract Classes

Interfaces:

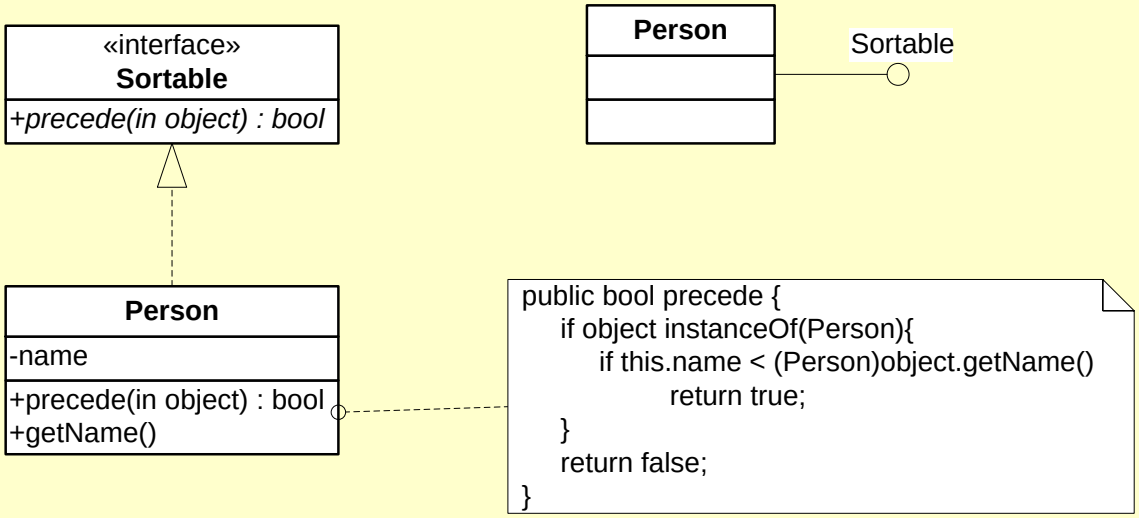
- Cannot be instantiated
- **All** methods are abstract
- A class can inherit from **several** interfaces
- Only declare (i.e. they do not implement!) a set of public operations

Abstract Classes:

- Cannot be instantiated
- **At least one** method is abstract
- A class can inherit from **only one** abstract class
- Can both declare abstract methods and implement concrete methods

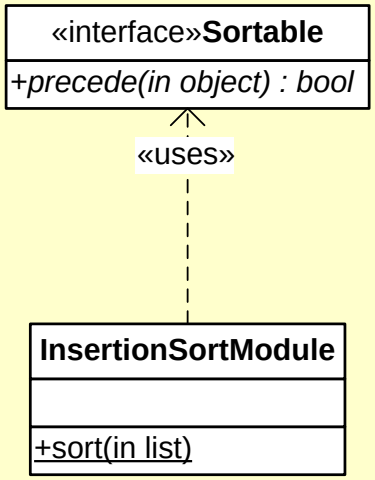
Interface Provided

- Exposes a set of services available to client classes
- Loose coupling (*couplage faible*) between the client and the class which implements the required service
- Classes can be easily replaced as long as they implement the same interface

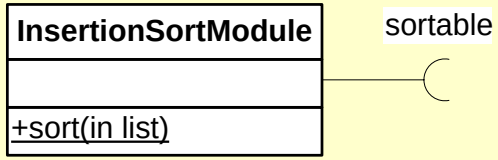


Interface required

- Complementary to the notion of interface
- Explicitly declares the functional dependencies which are required by a class to work
- Favor cohesion of a class for future reuse

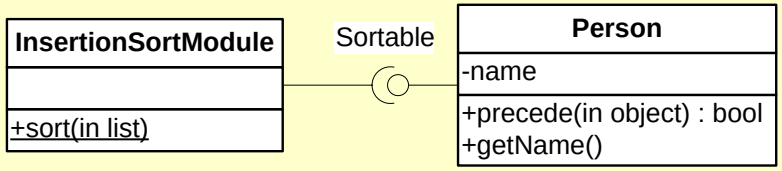


usage notation

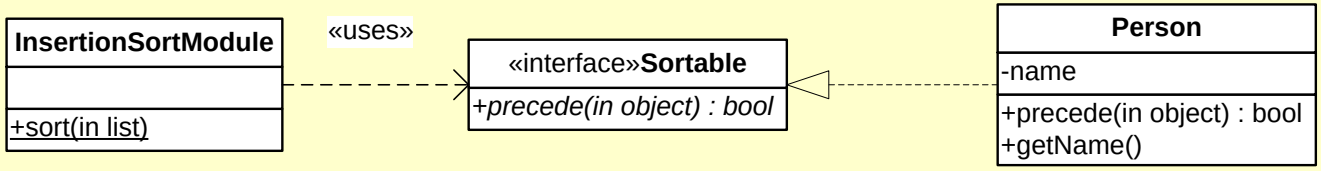


socket notation

Composing Interfaces

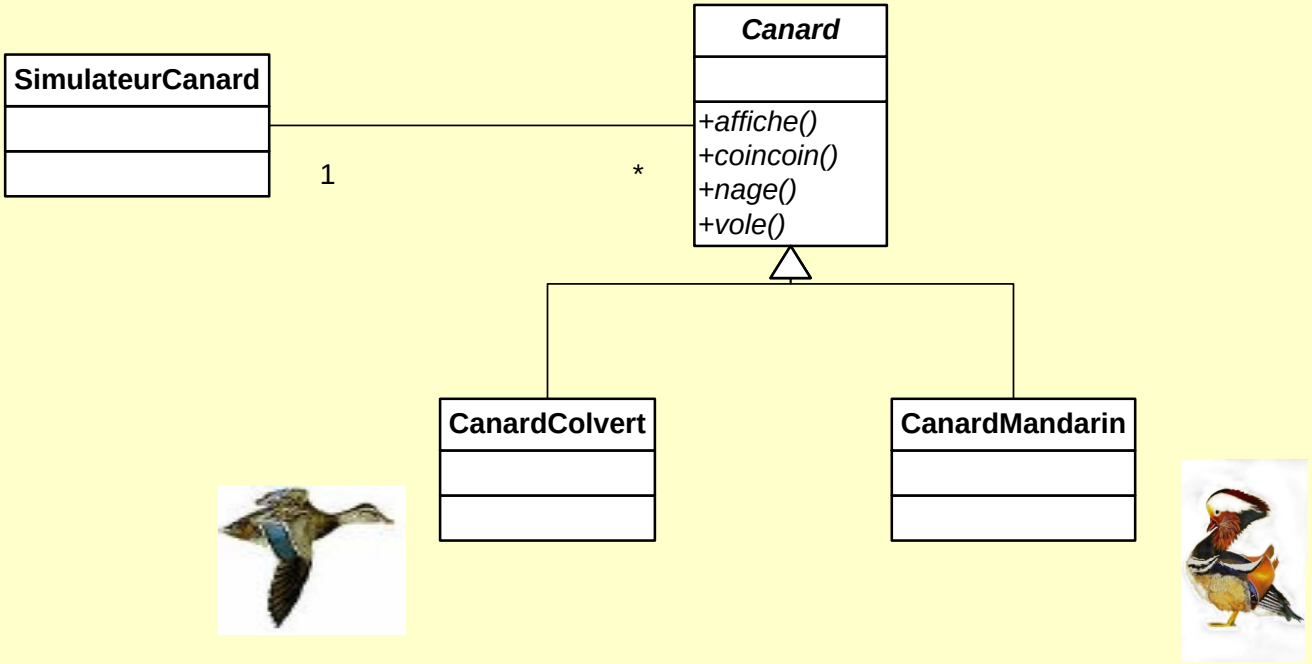


socket notation

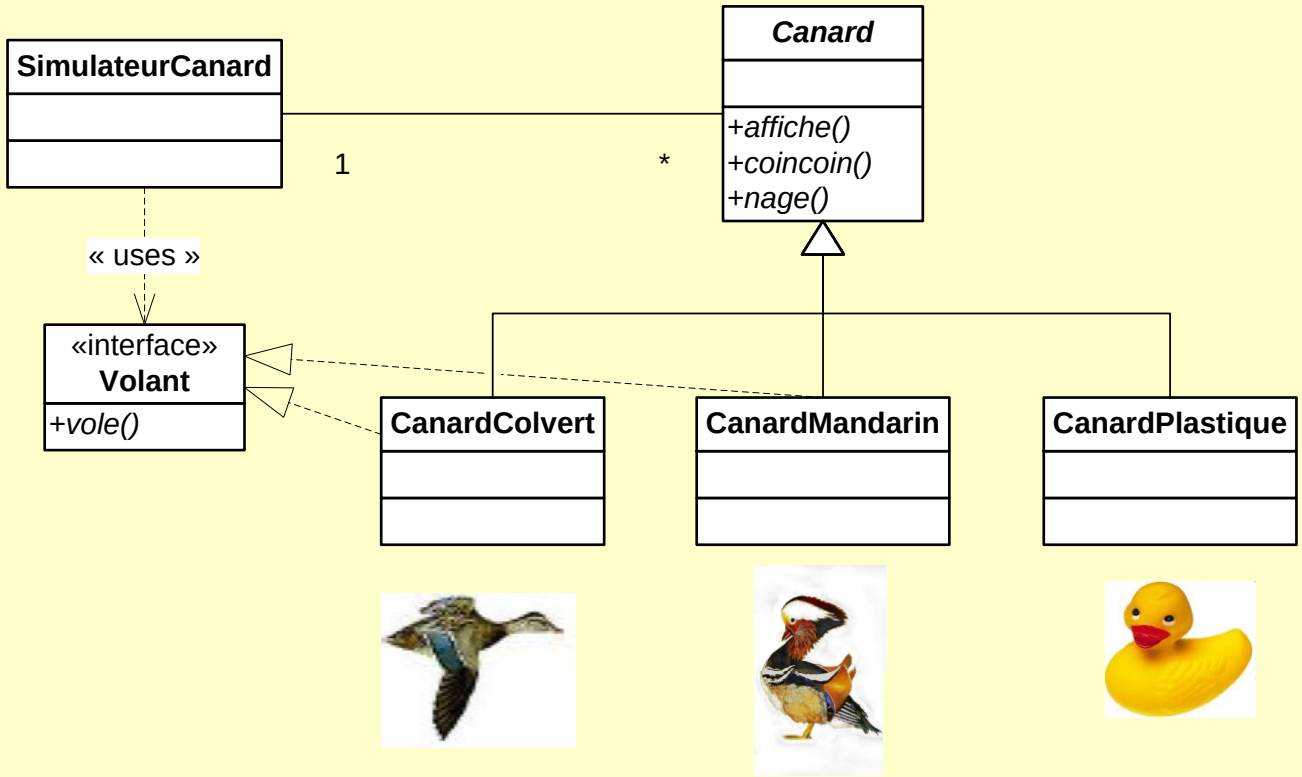


explicit notation

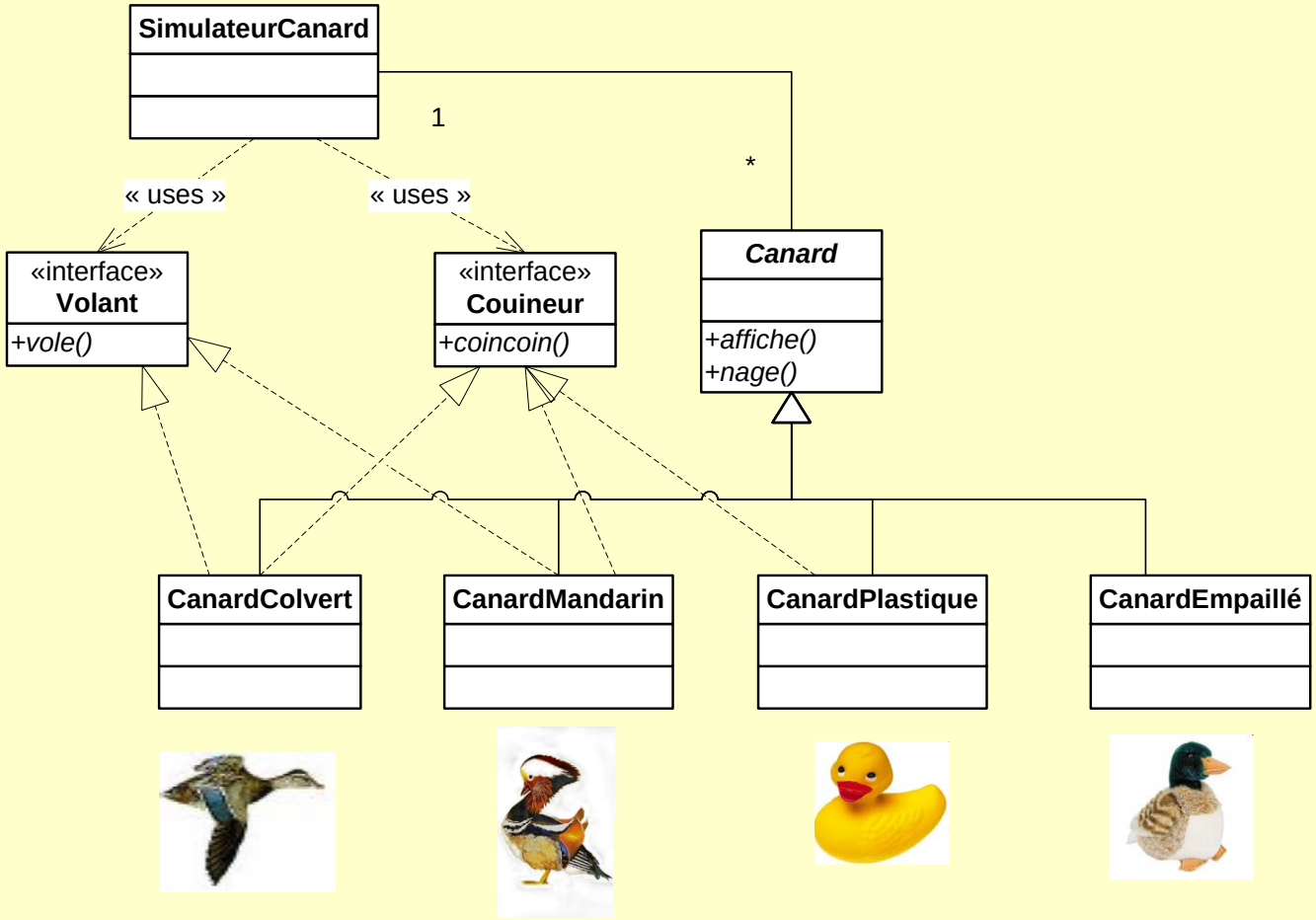
Example: Simulateur de Canards



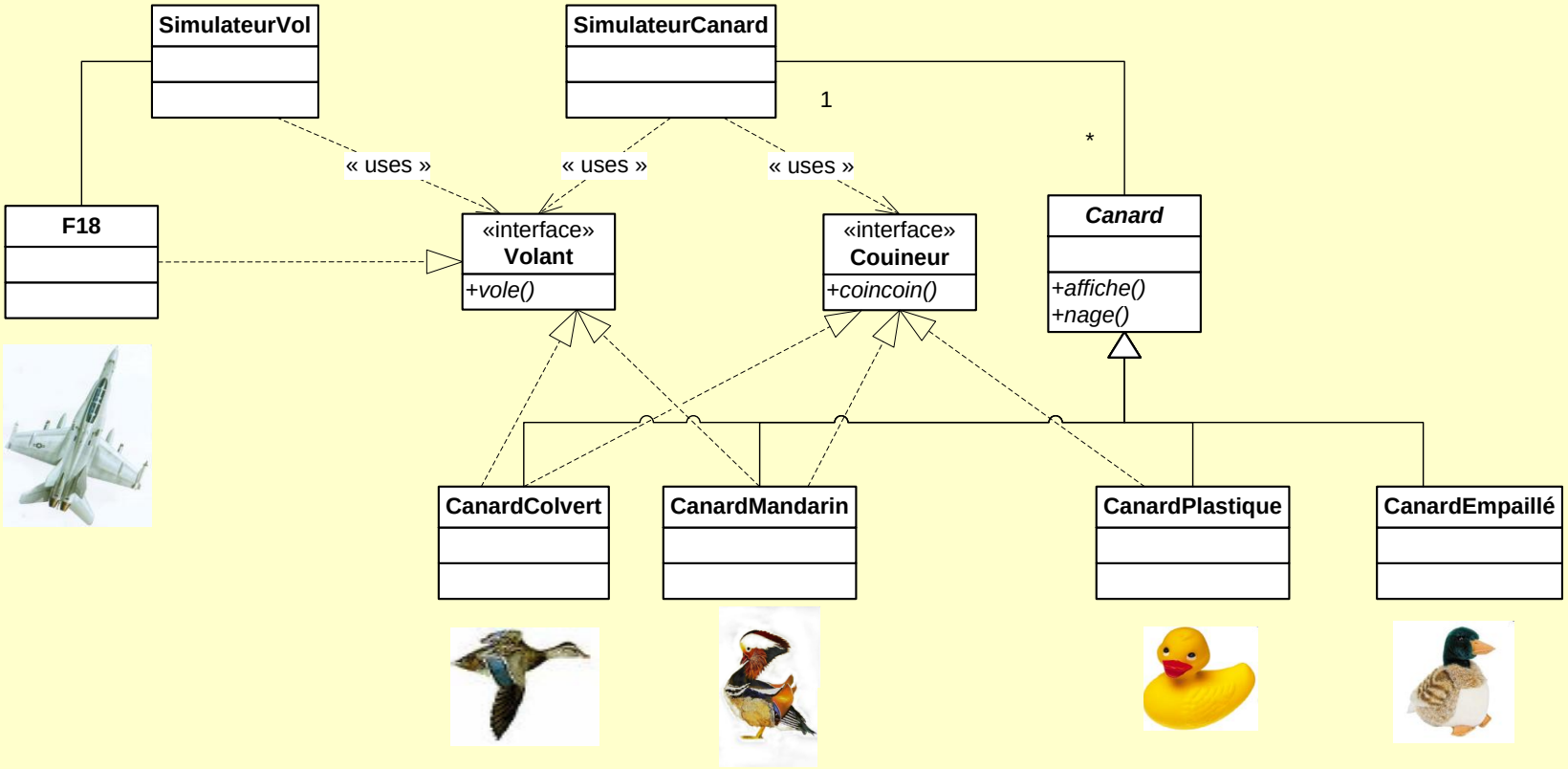
Simulateur de Canards: une interface



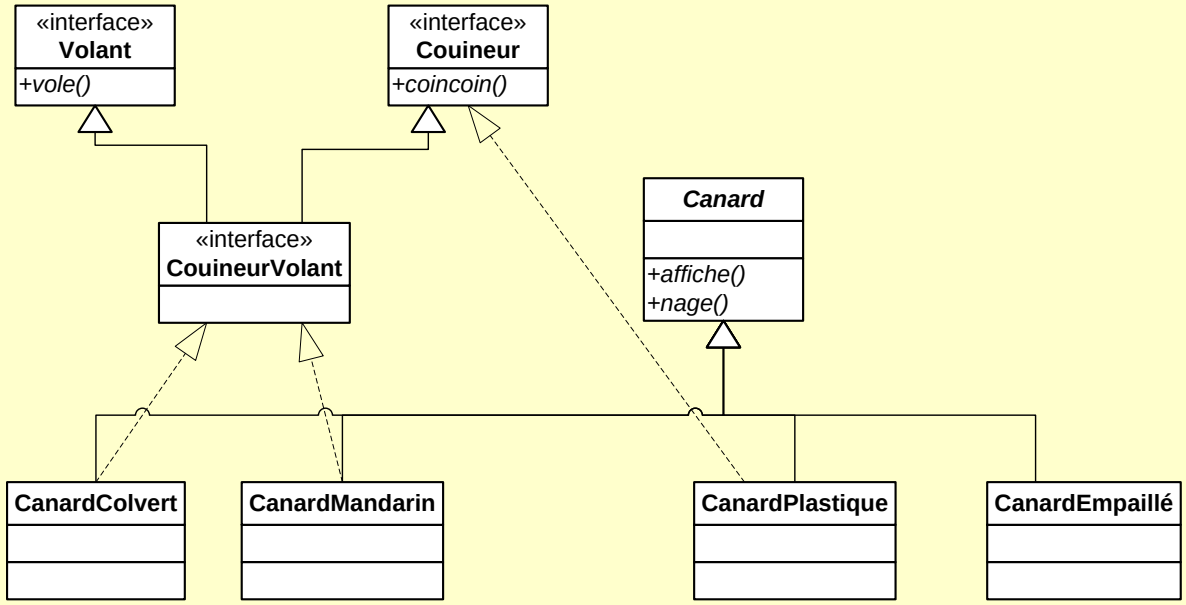
Simulateur de Canards: deux interfaces



Substitution de classes



Héritage d'interfaces



First Principle of OO Design

- [GoF95] *Programming to an interface, not an implementation*
- Two advantages:
  - Client classes do not need to know the concrete type of the classes they are manipulating as long as they respect a given interface
  - Underlying classes can be changed and evolve: as long as they expose the same interface, the application keeps working