

# Design Points - Subclassing vs Subtyping

Stéphane Ducasse  
stephane.ducasse@inria.fr  
<http://stephane.ducasse.free.fr/>

Stéphane Ducasse --- 2005

## How to Implement a Stack?

By subclassing OrderedCollection...

```
Stack>>pop
  ^ self removeFirst
Stack>>push: anObject
  self addFirst: anObject
Stack>>top
  ^ self first
```

Stack>>size, Stack>>includes:

## BUT BUT BUT!!!

- What do we do with all the rest of the interface of OrderedCollection?
- a Stack IS NOT an OrderedCollection!
- We cannot substitute an OrderedCollection by a Stack
- Some messages do not make sense on Stack
  - Stack new addLast: anObject
  - Stack new last
- So we have to block a lot of methods...

## Consequences...

```
Stack>>removeLast
  self shouldNotImplement
```

```
Stack>>pop
  ^ super removeLast
```



## The Problem



- There is not a clean simple relationship between Stack and OrderedCollection
- Stack interface is not an extension or subset of OrderedCollection interface
- Compare with CountingStack a subclass of Stack
- CountingStack is an extension

## Another Approach



By defining the class Stack that uses OrderedCollection

Object subclass: Stack  
iv: elements

```
Stack>>push: anElement  
elements addFirst: anElement  
Stack>>pop  
element isEmpty ifFalse: [^ element removeFirst]
```

## Inheritance and Polymorphism



- Polymorphism works best with standard interfaces
- Inheritance creates families of classes with similar interfaces
- Abstract class describes standard interfaces
- Inheritance helps software reuse by making polymorphism easier

## Specification Inheritance



- Subtyping
- Reuse of specification
  - A program that works with Numbers will work with Fractions.
  - A program that works with Collections will work with Arrays.
- A class is an abstract data type (Data + operations to manipulate it)

## Inheritance for Code Reuse



- Subclassing
  - Dictionary is a subclass of Set
  - Semaphore is a subclass of LinkedList
  - No relationship between the interfaces of the classes
- Subclass reuses code from superclass, but has a different specification. It cannot be used everywhere its superclass is used. Usually overrides a lot of code.
- ShouldNotImplement use is a bad smell...

## Inheritance for Code Reuse



- Inheritance for code reuse is good for
  - rapid prototyping
    - getting application done quickly.
- Bad for:
  - easy to understand systems
  - reusable software
  - application with long life-time.

## Subtyping Essence



- You reuse specification
  - You should be able to substitute an instance by one of its subclasses (more or less)
  - There is a relationship between the interfaces of the class and its superclass

## How to Choose?



- Favor subtyping
- When you are in a hurry, do what seems easiest.
- Clean up later, make sure classes use “is-a” relationship, not just “is-implemented-like”.
- Is-a is a design decision, the compiler only enforces is-implemented-like!!!

# Quizz



- Circle subclass of Point?
- Poem subclass of OrderedCollection?