Objects and Design in Smalltalk

3b. When (not) to inherit

Éric Tanter
etanter@dcc.uchile.cl



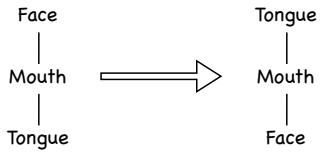


Inherit or Aggregate?

- It is not always obvious to choose between 'using' and 'extending'
- There is a design tension between both
  - potential for confusion...

Inverting Hierarchies

- Inverting an aggregation hierarchy can produce a legitimate inheritance hierarchy



  - instances of subclasses have more properties
    - a face does what a mouth does (eat) and more
- Nothing prevents you from doing this
  - write the code for both

Dilemma

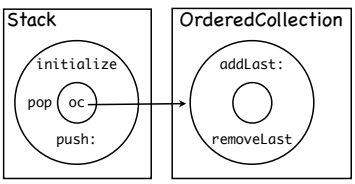
- Use or extend? (buy or inherit)
  - only two mechanisms for accessing a class's behavior or properties
  - trust your intuition
    - not intuitive to think of Face a special kind of Mouth
    - if not intuitive to you, won't be to others!
    - a good design reduces mental translations
- Not so simple if multiple inheritance: mixins
  - simple classes used as superclasses
  - use inheritance mechanism, but thinking about composition/aggregation

Exercise

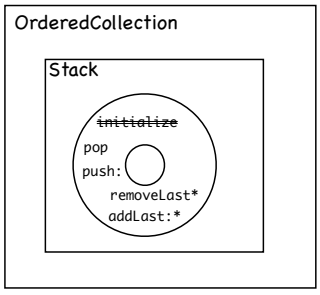
- In Smalltalk
  - Collection is an abstract class for containers
  - OrderedCollection is a concrete subclass
    - relative positions of elements
    - here, just need addLast: and removeLast
- Design a Stack class

Alternatives

- use or extend




```
push: anObject
  oc addLast: anObject
```




```
push: anObject
  self addLast: anObject
```

Conclusions

- Both techniques work
  - inheritance generally yields a (slightly) smaller solution
- Inheritance implies tight coupling
  - everything that applies to the superclass applies to the subclass
  - may or may not be what you want
  - Stack?
  - it is a long-term commitment (added methods)

Trade-offs

	Using (selectively access methods of another class)	Inheriting (access all methods and instance variables of another class)
Code size	More	Less (good!)
Subversive superclass messages	Preventable (good!)	Possible
Maintenance	Evolve independently (loose coupling)	Evolve together (tight coupling)
Familiarity with the superclass's internals	Less	More

To Sum Up

- kinds of reuse
  - using: black-box reuse
  - inheriting: white-box reuse
- powerful reason to inherit: polymorphism
  - next chapter
- in doubt, consider using
  - novices tend to overuse inheritance
  - using is less brittle than inheriting

On Multiple Inheritance

- Powerful facility (eg. C++, Eiffel)
  - without it, using is the only solution
  - but it's a poor substitute
- But it is complex
  - eg. repeated inheritance
  - requires complicated linguistic rules
    - which instance variables should be duplicated?
- Historical curiosity:
  - MI was introduced in Smalltalk-80, but then withdrawn (too much complications vs. benefits)

