Objects and Design in Smalltalk

2. Classes & Inheritance

Éric Tanter
etanter@dcc.uchile.cl





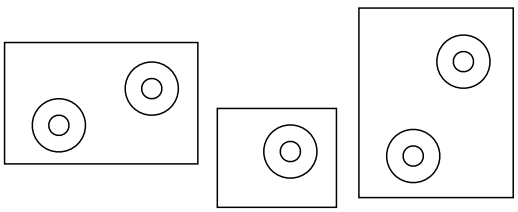
Classes

- How do you create objects?
  - Ex-nihilo in prototype-based languages
  - With a class in class-based languages
- A class can be seen as a factory that can produce objects

```
smalltalk syntax
  Dictionary new            -> create object
  X := Dictionary new       -> establish a “handle”
  X add: ...                -> use it
  Y := Dictionary new       -> create another
```


Classes

- First structuring principle
- In a language with classes
  - objects are not individualistic
  - objects from the same class have the same operations (same behavior)

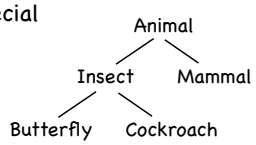


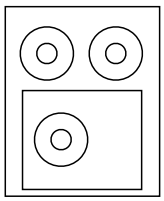
The word "class"

- Misleading: its usual meaning is that of a set
- A class is a factory rather than a collection
  - it is not a container!
  - e.g. onions on the grocer's shelf vs. shopping basket
- A class is not a type
  - although it's close (more on this later)

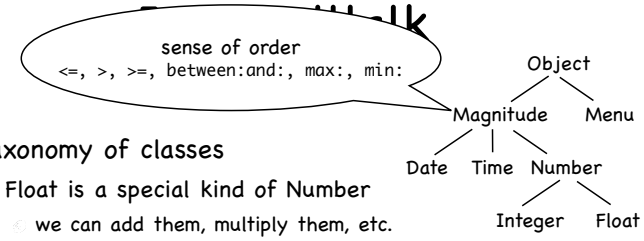
Inheritance

- 2nd structuring principle
  - specifying that a class is a special kind of another class
    - insect is a special kind ("specialization") of animal
  - taxonomy
    - reuse of knowledge





- In OOP
  - Butterfly is a subclass of Insect
  - Insect is the superclass of Butterfly
  - Butterfly inherits from Insect[3]

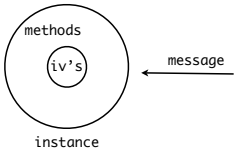


- Taxonomy of classes
  - Float is a special kind of Number
    - we can add them, multiply them, etc.
  - "everything is an object"
    - Object is very abstract (so is Animal!)
- What's Magnitude?
  - subclasses inherit operations: code savings!
    - (more on specializing inherited operations later)
  - design: factoring out commonalities in a superclass

Use of Inheritance

- Factoring out commonalities
- Incremental programming
  - extend existing class with what is lacking
  - beware: can lead to obscure/arbitrary designs
- Recall
  - objects are not inside their classes
  - objects are created from their class (factory)
  - objects of a class have more qualities than objects of its superclass: subclass = enrich

Terminology



- Commonly used
  - an object is instance of a class
  - a message activates a method (operation)
  - data inside an object are instance variables (iv's)
- Alternatives
  - outside: protocol, behavior, interface, services
  - inside: attributes, characteristics, memory, state

Examples

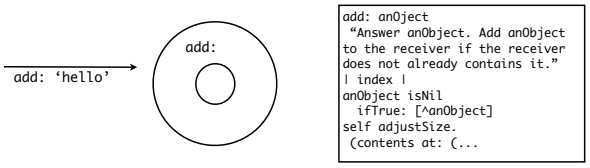
class	instance	message	effect or return
Account	MySavings	MySavings withdraw: 230	Processes a withdrawal
Integer	134	134 - 95	Returns 39
String	'hello'	'hello' size	Returns 5
Set	MySet	MySet add: 'hello'	Puts one more object in a set

Messages

- Difference with function calls?
  - a message has a distinguished "argument": the receiver
  - a function treats all its arguments as peers
- The receiver defines the execution context
  - locality of state changes and behavior customization

Terminology

- message, selector, method

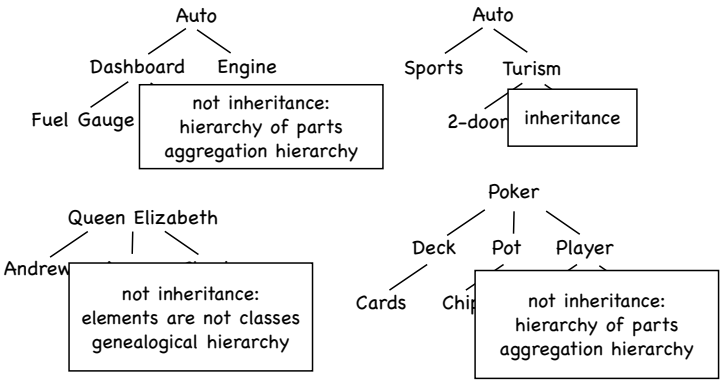



```
add: anOject
 "Answer anObject. Add anObject
to the receiver if the receiver
does not already contains it."
| index |
anObject isNil
  ifTrue: [^anObject]
self adjustSize.
 (contents at: (...
```

- variable, object



Exercise: Hierarchies

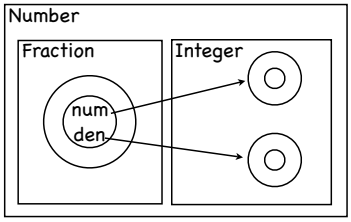


A Note on Aggregation

- Also called: part-of, assembly, whole-part, composite, has-a
- Aggregation is even more fundamental
  - essential in (life and) programming much before abstraction
- A node may appear more than once
- Can be tight or loose
  - car-engine (tight) deck-cards (loose)
  - loose aggregation is often called a container

Aggregation+Inheritance

```
Number subclass: #Fraction
instanceVariableNames: 'num den'
...
```



Without both aggregation and inheritance, you don't get far in OO!

Self

- Every OO language has a special 'context' variable called `self` (or `this`)
  - it refers to the current instance for which the method is executing
  - think anthropomorphically! self is me

```
talk
  self answer: 'I have nothing to say'
```

Super

- Every OO language has a special 'context' variable called super
  - it tells the system to start looking for a method in the parent of this (superclass)

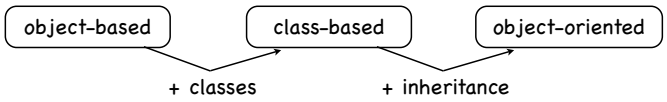
```
Point subclass: #ColorPoint
instanceVariableNames: 'color'
...
```

```
printOn: aStream
  super printOn: aStream.
  aStream nextPut: '('.
  color printOn: aStream.
  aStream nextPut: ')'.
```

- both self *and* super refer to the same object!
  - this is 'late binding of self'

What is OOP?

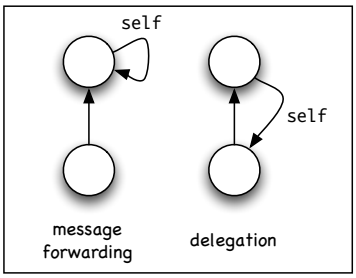
- Peter Wegner (1987) proposed a definition for OO languages



- That's just one classification
  - late binding, garbage collection, multiple inheritance, etc.
  - doesn't include prototype-based languages with true delegation

Delegation vs. Message Forwarding

- Late binding of self



???

- the `class` message returns the class of an object (`getClass()` in Java)

```
      self class = super class

this.getClass() == super.getClass()
```


19