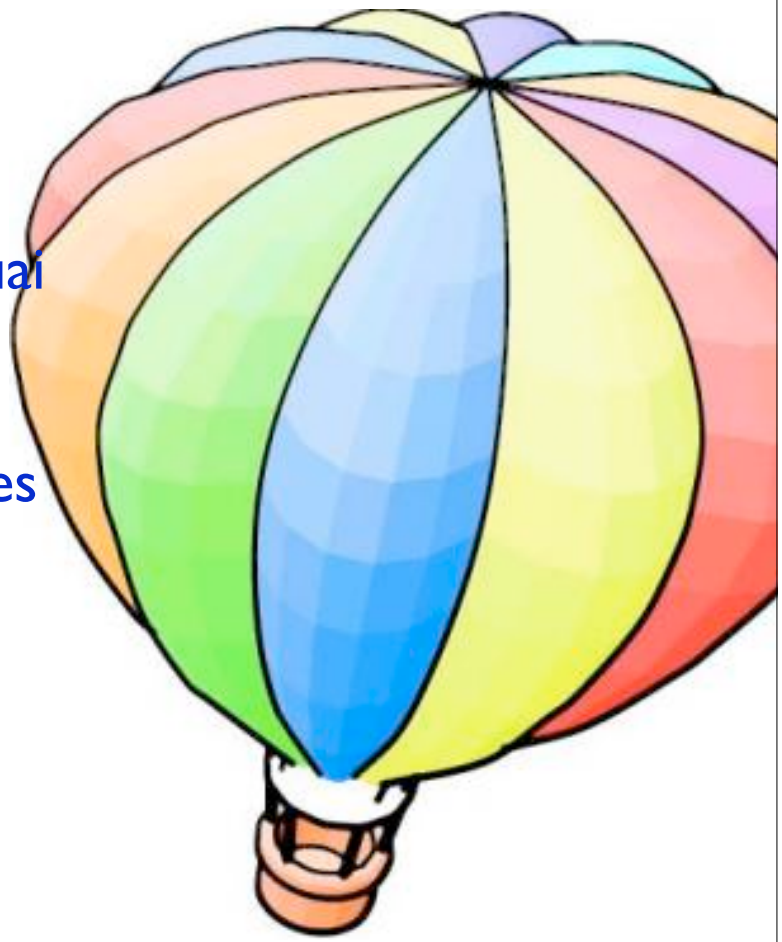
Why using Smalltalk for Teaching Object-Oriented Design

N. Bouraqadi - Ecole des Mines de Douai
S. Ducasse - University of Berne
S. Stinckwich - University of Caen
R. Wuyts - Université Libres de Bruxelles



Outline

What do we want to teach?
Smalltalk model
Smalltalk syntax
Smalltalk advantages
Smalltalk drawbacks
Experiences
Testimonies

Teaching Goals

- OOP and not UML!
- UML is a notation not a paradigm
- Object-Oriented Prog. vs Procedural
- Polymorphism, self, super semantics
- Dynamicity and late-binding
- Responsibility driven design
- Design
  - Design Patterns
  - Inheritance vs. delegation based reuse
  - Law of Demeter
- Subtyping vs. Subclassing
- Unit testing and refactoring

What are objects?

Objects are not abstract data-types

Objects are

- Unit of responsibility
- Unit of behavior

Furniture vs. Tamagashi classes

- Furniture is a dead object

- A tamagashi is a living object
- It gets hungry, sleeps...
- It reacts to messages

Smalltalk

- Everything is an object
- Only references
- Objects, messages and closures
- High-level iterators

- No primitive types
- No static methods
- No inextensible operators
- No public fields

Lot of ready to use material

- Free books online
  http://www.iam.unibe.ch/~ducasse/FreeBooks.html

  - Free online lectures:
  - www.iam.unibe.ch/~ducasse/
  - st-www.cs.uiuc.edu/users/cs497/
  - http://www.eli.sdsu.edu/courses/index.html
  - http://prog.vub.ac.be/POOL

- Mark Guzdial's Squeak Textbook
- Design patterns companion
- K. Beck's Best Smalltalk Practices

Smalltalk Object Model

- Everything is an object
- Only message sends and closures
- Public methods
- Protected attributes
- Single Inheritance
- Nothing special for static

Smalltalk Syntax in a Postcard!

```
exampleWithNumber: x
 "A method that illustrates every part of Smalltalk method syntax except
primitives.  It has unary, binary, and keyword messages, declares arguments
and temporaries,  accesses a global variable (but not and instance variable),
uses literals (array, character, symbol, string, integer, float), uses the pseudo
variable true false, nil, self, and super, and has sequence, assignment, return
and cascade. It has both zero argument and one argument blocks."

        |y|
        true & false not & (nil isNil) ifFalse: [self halt].
        y := self size + super size.
        #($a #a "a" 1 1.0)
                        do: [:each | Transcript show: (each class name);
                                                show: ' '].
        ^ x < y
```

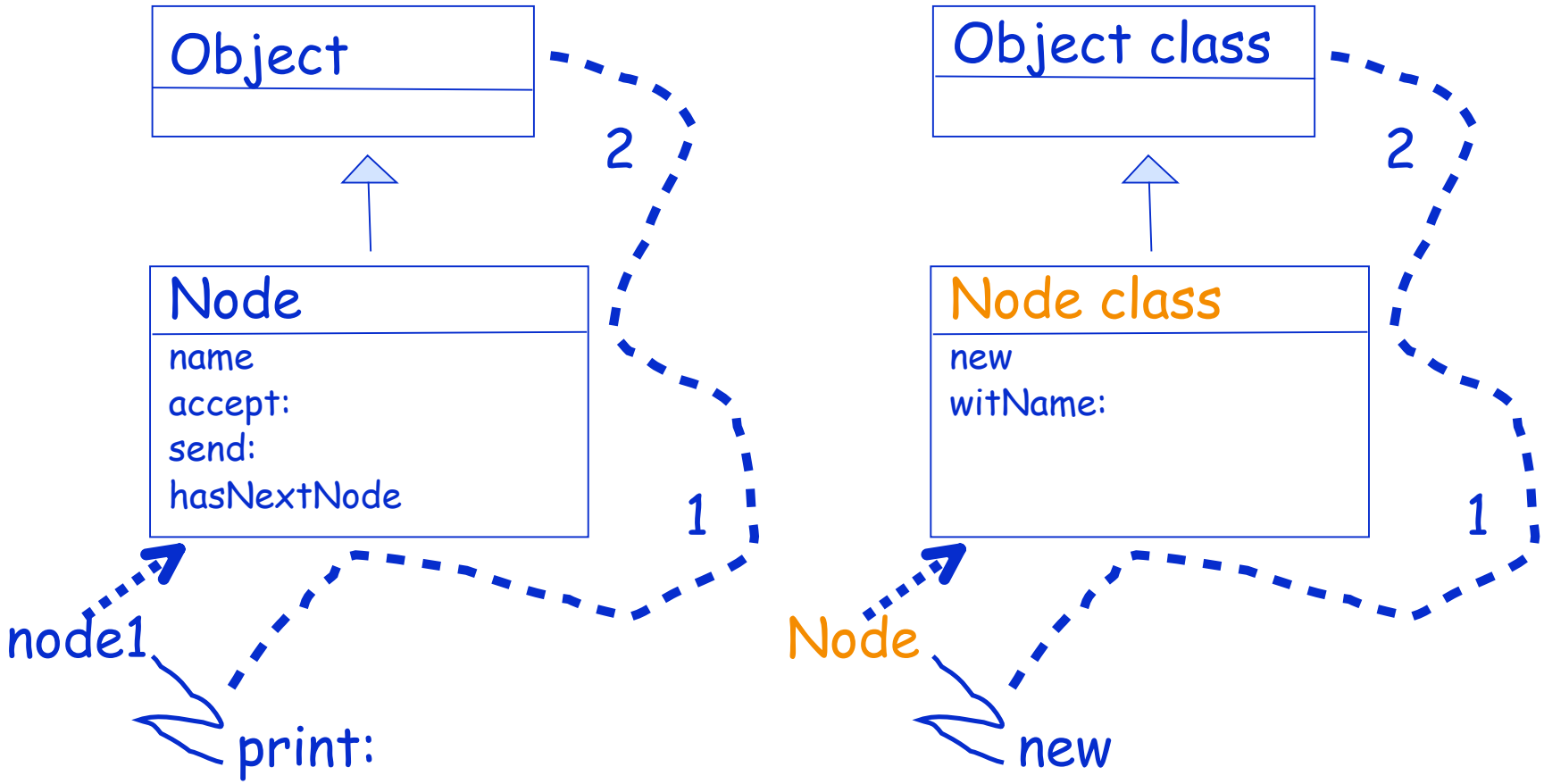

Simplcitiy Force you to Think

What is the meaning the instance-of?

Class/instance: one ***single*** lookup

Sending a message: looking in ***class*** of receiver and in class ***ancestors***

One lookup: look in *class* of receiver



What is the Essence of ...

Object creation
Super semantics

```
Window class>>new
        |instance|
        instance := super new.
        instance initialize.
        ^ instance
```

```
xxx
    ^ self class == super class
    "return this.getClass.equalsTo(super.getClass)
```


Antropomorphism

- Talk with your objects
- Direct manipulation
- Incremental compilation
- Inspect: the power of living entities
- Transcript inspect

Antropomorphism (II)

Vocabulary

sending messages vs. calling a function

-- sending message is about communication and behavior

-- sending a message conveys late-binding

stress responsibility

Syntax: we talk to objects

```
bunny turn: #left times: 3
```

is equivalent to

```
bunny.turntimes(#left,3);
```


Exact Numbers and Automatic Coercion

```
(1/3) + (2/3)
  > 1
1000 factorial / 999 factorial
  > 1000
(1/3) + 1
   > 4/3
15 sin
1 class
  > SmallInteger
The biggest small int: 1 class maxVal
The smallest large int: 1 class maxVal + 1
```


Iterators: Closures at Work

-Simple
-Elegant
-Powerful

```
#(1 2 3 4) do:
        [:each| Transcript show: each printString]

'abcd' do: [:each| Transcript show: each]

#(1 2 3 4) collect: [:each | each isOdd]

#(1 2 3 4) select: [:each | each isOdd]
```


The Design is in the System

Boolean class is abstract

false, true are objects

not, and, or, & are polymorphic methods
defined on True and False

Numberous Libraries

Collections (influenced Java 1.1)
Network
Full compiler available
All kinds of packages (XML, SOAP,....)
Processes, access to stack

Powerful IDE

Incremental compilation
Compiler everywhere
Powerful debugger with hot recompilation
Several code browsers (Refactoring browser)
Cross referencers, navigations
Unit testing
Full log of changes
Team repository (www.squeaksource.com)
Do not require a 256 mb PIII

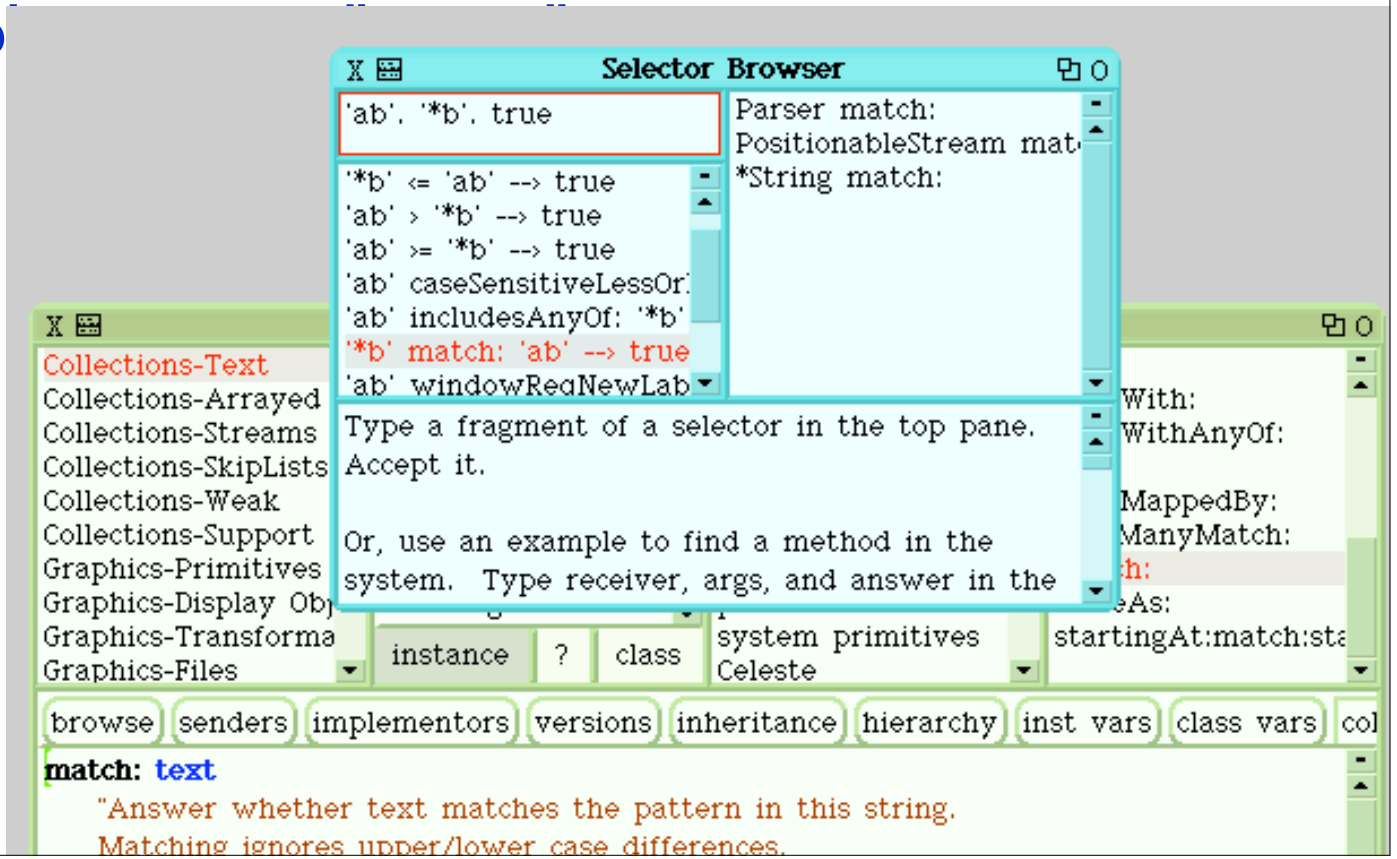
Squeak Powerful Tools

Message Finder

Method Finder

give methods arguments and obtain the methods

'ab' . '*b



Full Access to Everything

-Learning by reading
-The system is written in itself
-All the code is there in one click!
--Compiler
--Bytecode
--Execution stack access (continuations)

Smalltalk Drawbacks

Multiple dialects: Multiple UIs frameworks
- use seaside for the UI

No buzz

No hype

Syntax seems difficult to {} thinkers

Mathematical syntax not supported

Other can think that you are an idiot

Experiences: Teaching Design

- Berne
- Java before
- 5th year
- 13 * 2h
- 3 lectures only on Smalltalk
- Small groups with projects
- Mentor
- Design reviews + presentations
- Code critics

Experiences: Teaching Smalltalk

- Berne
- Java before
- 5th year
- 13 * 2h
- Use seaside as UI
- Reflection: building a code metric tools
- Building an abstract interpreter

Experiences: Teaching basic OOP

- Neuchatel
- 13 * 2 hours lectures
- 13 * 2 hours lab
- OOP
- Semantics of self - super
- Subclassing supertyping
- Composition inheritance vs. delegation
- object = responsibility + behavior
- Some design patterns
- Law of Demeter
- Problems with Morph (blurring model and UI)

Testimony

I have tried teaching software engineering using C++. After many attempts, I finally gave up using C++. I concluded that there was so much overhead using C++. The language obscured what I was trying to convey. I also tried Java which was better but not good enough. There were still too many language issues. Smalltalk was by far the most successful. For the first time ever, language was never an issue. It made everything easier to discuss because I didn't need to wade through a load of syntax. Object oriented design is much easier when everything is an object.

Rick Zaccone zaccone@bucknell.edu

I used Smalltalk as the basis for my object-oriented programming course in the years 1992 -1998. I read my course mostly at Vytautas Magnus University in Kaunas, Lithuania. I don't know any other programming language which would demonstrate the essence of object-oriented programming so naturally as Smalltalk. I used many programming languages in my life, but none of them equals Smalltalk in ease and pleasure of programming. Some of my students were able to develop quite interesting programs in a one-semester course.

Raimundas Vaitkevièius

Stéphane Ducasse

I worked as a teacher in OO at Enator (large consulting firm in Sweden) where we especially used Smalltalk in a 2-day crash course called "Practical exercise in OO". I developed this course as a response to the problem that all the short courses had very small trivial examples and exercises and didn't really show how OO can make complex systems much more approachable. This course had a small library system instead, in which the students (working in pairs) ultimately should add the class Loan to make it work. That was the second day, the first day was spent learning enough Smalltalk to do it! :-)

The course was a success, almost everyone that took it had an "Aha-experience". But it also demanded some courage and quite some Smalltalk experience from the teacher.

goran.hultgren@bluefish.se

Summary

- Simple Model - Uniformity
- Simple Syntax
- Reinforce antropomorphism
- Focus on concepts
- Powerful environments
- Free and everywhere
- Excellent teaching material available

Refs

- M. Guzdial, E. Soloway, Teaching the Nintendo Generation to Program, CACM, 2001

- T. Kuehne, Smalltalk for Students: A Giant Leap for Studentkind, Journal of Object-Oriented Programming, 2001

- S. Ducasse and R. Wuyts, Supporting Objects as An Anthropomorphic View at Computation or Why Smalltalk for Teaching Objects?, In Proceedings of the Ecoop'02 International Educator Symposium, 2002