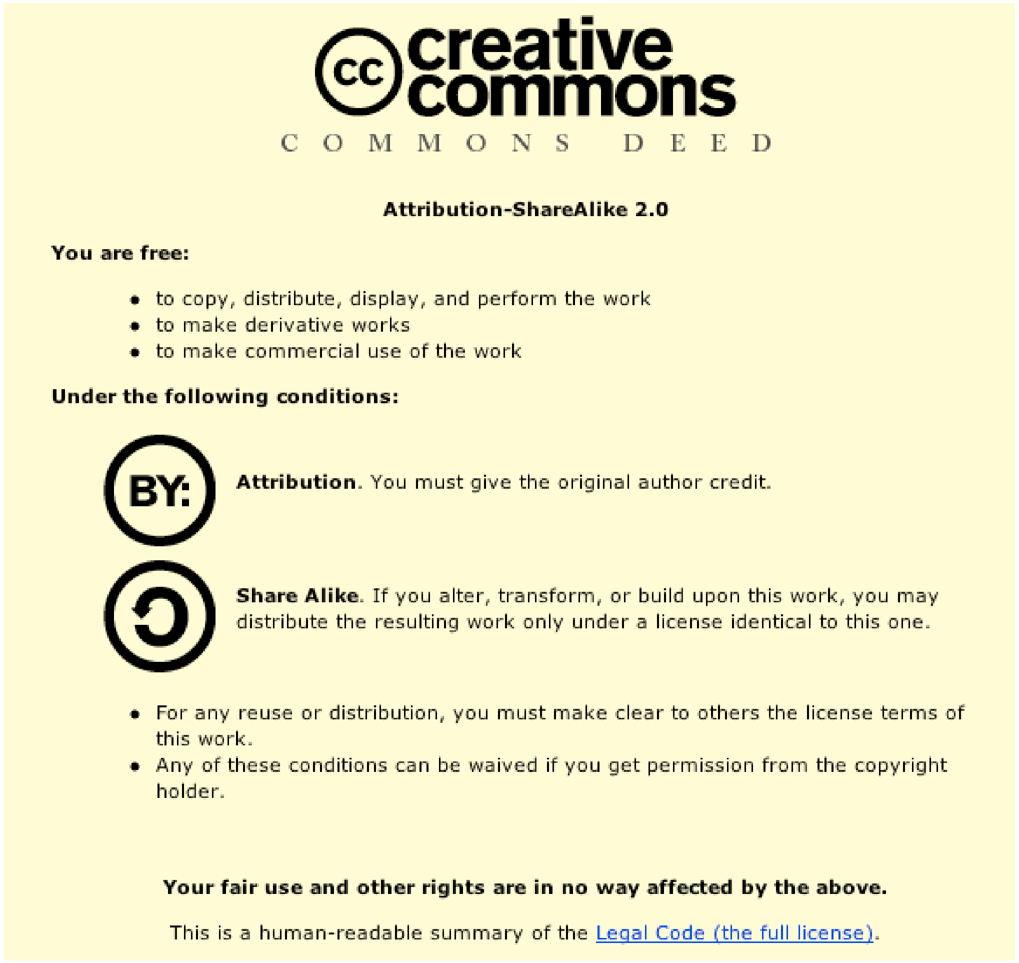
A Taste of Smalltalk

Stéphane Ducasse
Stephane.Ducasse@univ-savoie.fr
http://www.listic.univ-savoie.fr/~ducasse/

License: CC-Attribution-ShareAlike 2.0

http://creativecommons.org/licenses/by-sa/2.0/



Goals

- Two examples:
  - ‘hello world’
  - A tamagotchi
- To give you an idea of:
  - the syntax
  - the elementary objects and classes
  - the environment

An Advice

You do not have to know everything!!!

- “Try not to care - Beginning Smalltalk programmers often have trouble because they think they need to understand all the details of how a thing works before they can use it. This means it takes quite a while before they can master Transcript show: ‘Hello World’. One of the great leaps in OO is to be able to answer the question "How does this work?" with "I don’t care"“. Alan Knight. Smalltalk Guru

- We will show you how to learn and find your way

Some Conventions

- Return Values

```
1 + 3 -> 4
Node new -> aNode
```

- Method selector #add:
- Method scope conventions
- Instance Method defined in class Node:

```
Node>>accept: aPacket
```

- Class Method defined in class Node (in the class of the the class Node)

```
Node class>>withName: aSymbol
```

- aSomething is an instance of the class Something

Roadmap

- “hello world”
- Syntax
- a tamagotchi

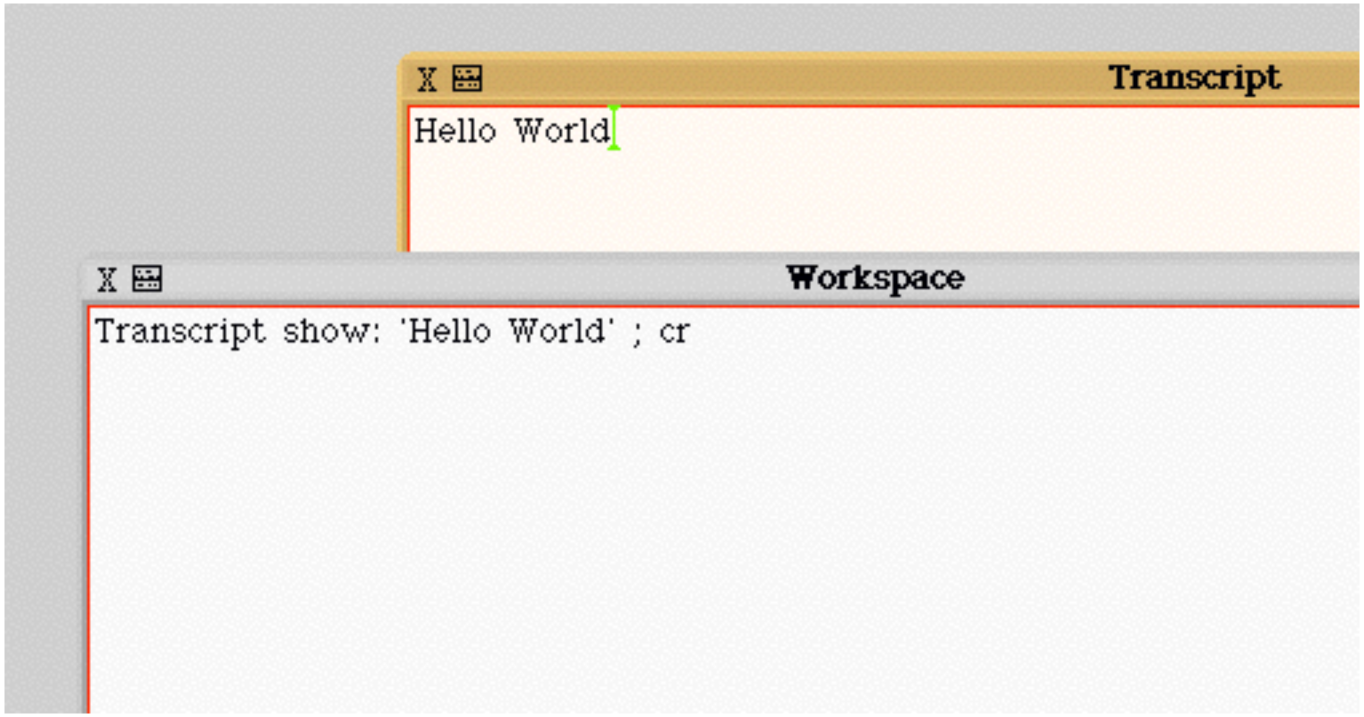
Hello World

- Transcript show: 'hello world'

- At anytime we can dynamically ask the system to evaluate an expression. To evaluate an expression, select it and with the middle mouse button apply doIt.
- **Transcript** is a special object that is a kind of standard output.
- It refers to a TextCollector instance associated with the launcher.

- In Squeak Transcript is dead slow...

Transcript show: ‘hello world’



```
Hello World
```

```
Transcript show: 'Hello World' ; cr
```

Everything is an Object

- The workspace is an object.
- The window is an object: it is an instance of SystemWindow.
- The text editor is an object: it is an instance of ParagraphEditor.
- The scrollbars are objects too.
- ‘hello word’ is an object: it is aString instance of String.
- #show: is a Symbol that is also an object.
- The mouse is an object.
- The parser is an object: instance of Parser.
- The compiler is also an object: instance of Compiler.
- The process scheduler is also an object.
- The garbage collector is an object: instance of ObjectMemory.
- Smalltalk is a consistent, uniform world written in itself. You can learn how it is implemented, you can extend it or even modify it. All the code is available and readable

Smalltalk Object Model

- ****Everything**** is an object
  - ⇒ Only message passing
  - ⇒ Only late binding
- Instance variables are private to the object
- Methods are public
- Everything is a pointer

- Garbage collector
- Single inheritance between classes
- Only message passing between objects

Roadmap

- Hello World
- First look at the syntax
- a Tamagotchi

Power & Simplicity: The Syntax on a PostCard

```
exampleWithNumber: x
"A method that illustrates every part of Smalltalk method syntax except primitives. It
has unary, binary, and key word messages, declares arguments and temporaries (but
not block 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. It doesn't do anything useful, though"
    |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: (each printString);
                                                show: ' '].
      ^ x < y
```

Yes ifTrue: is sent to a boolean

```
Weather isRaining
    ifTrue: [self takeMyUmbrella]
    ifFalse: [self takeMySunglasses]
```

ifTrue:ifFalse is sent to an object: a boolean!

Yes a collection is iterating on itself

```
#(1 2 -4 -86)
	do: [:each | Transcript show: each abs
printString ;cr ]

> 1
> 2
> 4
> 86
```

Yes we ask the collection object to perform the loop on itself

DoIt, PrintIt, InspectIt and Accept

- Accept = Compile: Accept a method or a class definition
- DoIt = send a message to an object
- PrintIt = send a message to an object + print the result (#printOn:)
- InspectIt = send a message to an object + inspect the result (#inspect)

Objects send messages

- Transcript show: 'hello world'
- The above expression is a message
  - the object Transcript is the ***receiver*** of the message
  - the ***selector*** of the message is #show:
  - one ***argument***: a string 'hello world'
  - Transcript is a global variable (starts with an uppercase letter) that refers to the Launcher's report part.

Vocabulary Point

Message passing or sending a message is equivalent to

- invoking a method in Java or C++
- calling a procedure in procedural languages
- applying a function in functional languages
- of course the last two points must be considered under the light of polymorphism

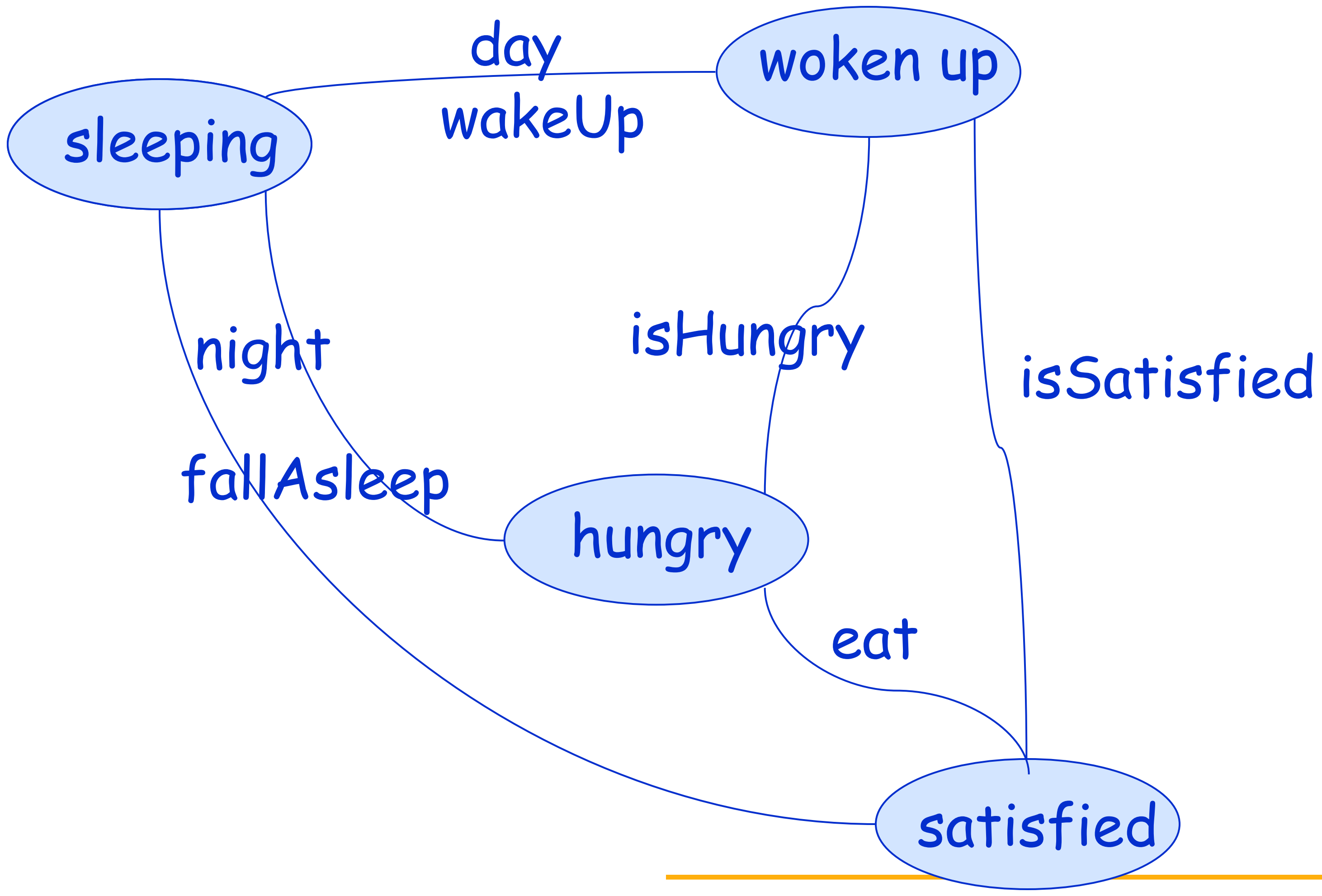
Roadmap

- Hello World
- First look at the syntax
- A tamagotchi

Tamagotchi

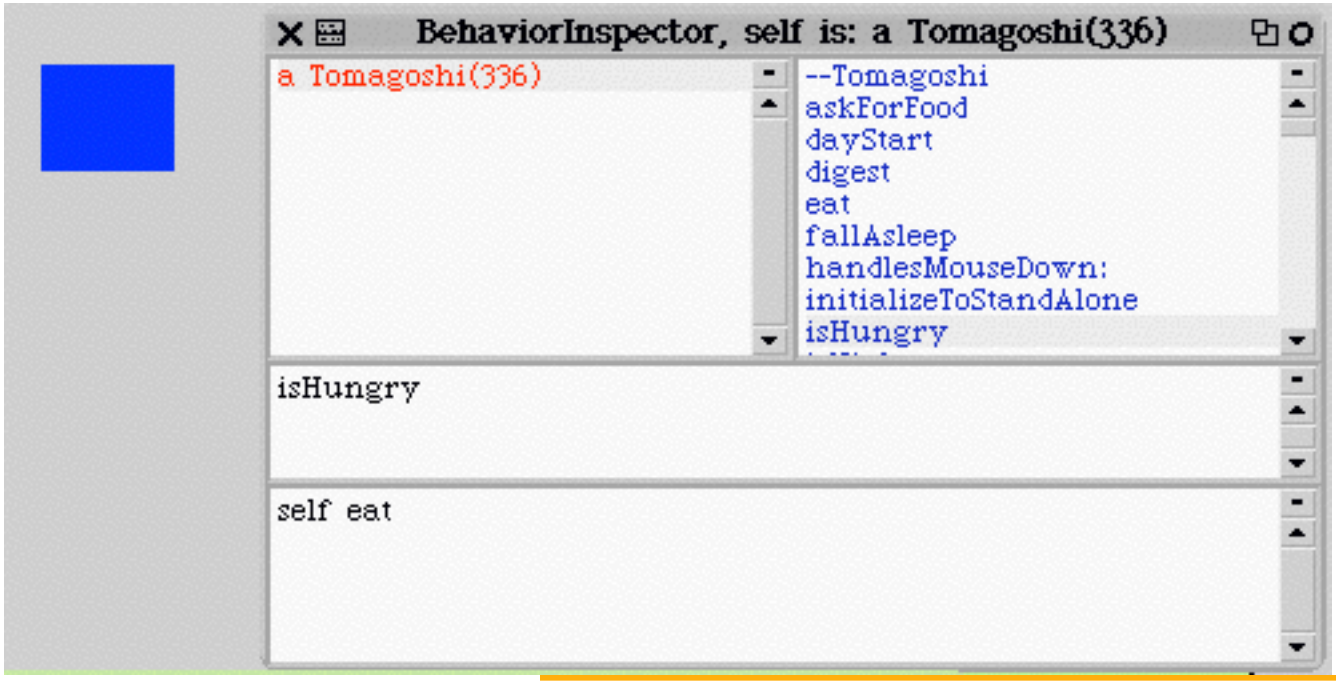
- Small entity
  - Its own night and day cycle
  - Eating, sleeping, been hungry, been satisfied
  - Changing color to indicate its mood

Tomagotchi


day
woken up
sleeping
wakeUp
night
isHungry
isSatisfied
fallAsleep
hungry
eat
satisfied

Instantiating…

- To create a tomagoshi:
- Tomagoshi newStandAlone openInWorld



```
a Tomagoshi(336)
```

```
--Tomagoshi
askForFood
dayStart
digest
eat
fallAsleep
handlesMouseDown:
initializeToStandAlone
isHungry
```

```
isHungry
```

```
self eat
```

S.Ducasse

How to Define a Class (Sq)

- Fill the template:

```
NameOfSuperclass subclass: #NameOfClass
    instanceVariableNames: 'instVarName1'
    classVariableNames: 'ClassVarName1 ClassVarName2'
    poolDictionaries: ''
    category: ’category name'
```

Tomagoshi (Sq)

- For example to create the class Tomagoshi

Morph subclass: #Tomagoshi
instanceVariableNames: '**tummy hunger dayCount isNight**'
classVariableNames: ''
poolDictionaries: ''
category: 'TOMA'

Class Comment!

- I represent a tomagoshi. A small virtual animal that have its own life.

- dayCount <Number> represents the number of hour (or tick) in my day and night.
- isNight <Boolean> represents the fact that this is the night.
- tummy <Number> represents the number of times you feed me by clicking on me.
- hunger <Number> represents my appetite power.
- I will be hungry if you do not feed me enough, but I'm selfish so as soon as I' satisfied I fall asleep because I do not have a lot to say.

How to define a method?

```
message selector and argument names
    "comment stating purpose of message"

    | temporary variable names |
    statements
```

```
Tomagoshi>>initializeToStandAlone
 “Initialize the internal state of a newly created tomagoshi”

 super initializeToStandAlone.
  tummy := 0.
  hunger := 2 atRandom + 1.
  self dayStart.
  self wakeUp
```

Initializing

```
Tomagoshi>>initializeToStandAlone
  "Initialize the internal state of a newly created
tomagoshi"

  super initializeToStandAlone.
  tummy := 0.
  hunger := 2 atRandom + 1.
  self dayStart.
  self wakeUp
```

dayStart

```
Tomagoshi>>dayStart

    night := false.
    dayCount := 10
```

Step

```
step
  "This method is called by the system at regurlar time interval. It defines
the tomagoshi behavior."
      self timePass.
      self isHungry
                  ifTrue: [self color: Color red].
      self isSatisfied
                  ifTrue:
                              [self color: Color blue.
                              self fallAsleep].
      self isNight
                  ifTrue:
                              [self color: Color black.
                              self fallAsleep]
```

Time Pass

```
Tomagoshi>>timePass
 "Manage the night and day alternance"

     Smalltalk beep.
      dayCount := dayCount -1.
      dayCount isZero
          ifTrue:[ self nightOrDayEnd.
                                          dayCount := 10].
      self digest

Tomagoshi>>nightOrDayEnd
  "alternate night and day"
     night := night not
```

Digest

```
Tomagoshi>>digest
"Digest slowly: every two cycle, remove one from the tummy”

 (dayCount isDivisibleBy: 2)
          ifTrue: [ tummy := tummy -1]
```

Testing

```
Tomagoshi>>isHungry
	^ hunger > tummy

Tomagoshi>>isSatisfied
	^self isHungry not

Tomagoshi>>isNight
	^ night
```

State

```
Tomagoshi>>wakeUp
    self color: Color green.
    state := self wakeUpState

Tomagoshi>>wakeUpState
  "Return how we codify the fact that I sleep"
    ^ #sleep

Tomagoshi>> isSleeping
    ^ state = self wakeUpState
```

Eating

```
Tomagoshi>>eat
        tummy := tummy + 1
```

S.Ducasse

Time and Events

```
Tomagoshi>>stepTime
     "The step method is executed every steppingTime ms"
     ^ 500

Tomagoshi>>handlesMouseDown: evt
    "true means that the morph can react when the mouse
    down over it"
    ^ true

Tomagoshi>>mouseDown: evt
  self eat
```

Summary

What is a message?

What is the message receiver?

What is the method selector?

How to create a class?

How to define a method?