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 class Node):
    - Node class>>withName: aSymbol
  - aSomething is an instance of the class Something

Roadmap

- "hello world"
- Syntax
- a LAN simulator

Goi ls

- Two examples:
  - "hello world"
  - a LAN simulator
- To give you an idea of:
  - the syntax
  - the elementary objects and classes
  - the environment
- To provide the basis for all the lectures:
  - all the code examples,
  - constructs,
  - design decisions, ...

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

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.

Transcript show: 'hello world'

- The workspace is an object.
  - The window is an object; it is an instance of ApplicationWindow
  - The text editor is an object; it is an instance of ParagraphEditor.
  - The scrollbars are objects too.
- hello world is an object; it is a String 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 an object; instance of Compiler.
  - The process scheduler is also an object.
- The garbage collector is an object; instance of MemoryObject.
  - Smalltalk is a cons, non-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.

Everything is an Object

- ***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

Smalltalk Object Model
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.

Yes a collection is iterating on itself

```ruby
#(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.

Vocabulary Point

A LAN contains nodes, workstations, printers, file servers. Packets are sent in a LAN and each node treats them differently.

Roadmap

- Hello World
- First look at the syntax
- LAN Simulator

Complete Syntax on a PostCard

```ruby
exampleWithNumber: x
    "Illustrates every part of Smalltalk method syntax. It has unary literals, key word messages, declares arguments and temporaries, accesses a global variable (not a and instance variable), uses literals (self 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."
| b | true & false & & (pd w) (false p (self halt)) |
| y | self size = super size. |
| x | ^ x > y |
| do: [:each | Transcript show: each abs printString ;cr]
| > 1 |
| > 2 |
| > 4 |
| > 86 |
```

Dolt, PrintIt, InspectIt and Accept

- Accept = Compile: Accept a method or a class definition
- Dolt: send a message to an object
- PrintIt: send a message to an object and print the result (printOn)
- InspectIt: send a message to an object and 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.

A LAN Simulator

A LAN contains nodes, workstations, printers, file servers. Packets are sent in a LAN and each node treats them differently.
Three Kinds of Objects
Node and its subclasses represent the entities that are connected to form a LAN. Packet represents the information that flows between Nodes. NetworkManager manages how the nodes are connected.

How to Define a Class?
- Fill the template:
  ```plaintext
  NameOfSuperclass subclass: #NameOfClass
  !instanceVariableNames: 'instVarName1'
  !classVariableNames: 'ClassVarName1 ClassVarName2'
  !poolDictionaries: ''
  !category: 'LAN'
  ```

Node and Packet Creation
```plaintext
<table>
<thead>
<tr>
<th>macNode</th>
<th>pcNode</th>
<th>node1</th>
<th>printerNode</th>
<th>node2</th>
<th>node3</th>
<th>packet</th>
</tr>
</thead>
</table>
macNode := Workstation withName: #mac.
pcNode := Workstation withName: #pc.
nod1 := Node withName: #node1.
nod2 := Node withName: #node2.
nod3 := Node withName: #node3.
printerNode := Printer withName: #lpr.
node1 nextNode: pcNode.
nod2 nextNode: pcNode.
nod3 nextNode: printerNode.
lpr nextNode: macNode.
packet := Packet send: 'This packet travelled to' to: #lpr.
```

Node and Packet Creation
```plaintext
<table>
<thead>
<tr>
<th></th>
<th>macNode</th>
<th>pcNode</th>
<th>node1</th>
<th>printerNode</th>
<th>node2</th>
<th>node3</th>
<th>packet</th>
</tr>
</thead>
</table>
macNode := Workstation withName: #mac.
pcNode := Workstation withName: #pc.
nod1 := Node withName: #node1.
nod2 := Node withName: #node2.
nod3 := Node withName: #node3.
printerNode := Printer withName: #lpr.
node1 nextNode: pcNode.
nod2 nextNode: pcNode.
nod3 nextNode: printerNode.
lpr nextNode: macNode.
packet := Packet send: 'This packet travelled to' to: #lpr.
```

Objects Send Messages
```plaintext
Message: 1 + 2
receiver: 1 (an instance of SmallInteger)
selector: #+
arguments: 2
```
```plaintext
Message: #nextNode: macNode
receiver: lpr (an instance of LanPrinter)
selector: #send:to:
arguments: 'This packet travelled to' and #lpr
```

Transmitting a Packet
```plaintext
| aLan packet macNode |
|---|---|
macNode := aLan findNodeWithAddress: #mac.
packet := Packet send: 'This packet travelled to the printer' to: #lpr.
```

Objects Send Messages
```plaintext
Message: 1 + 2
receiver: 1 (an instance of SmallInteger)
selector: #+
arguments: 2
```
```plaintext
Message: #nextNode: macNode
receiver: lpr (an instance of LanPrinter)
selector: #send:to:
arguments: 'This packet travelled to' and #lpr
```

Transmitting a Packet
```plaintext
| aLan packet macNode |
|---|---|
macNode := aLan findNodeWithAddress: #mac.
packet := Packet send: 'This packet travelled to the printer' to: #lpr.
```

How to Define a Method?
```plaintext
message selector and argument names
"comment stating purpose of message"
| temporary variable names | statements |
accept: thePacket
"If the packet is addressed to me, print it. Otherwise just behave like a normal node."
```
```plaintext
(message (thePacket isAddressedTo: self)
ifTrue: [self print: thePacket]
ifFalse: [super accept: thePacket])
```
In Java

- In Java we would write
  void accept(Packet thePacket)
  
  /* If the packet is addressed to me, print it. Otherwise just 
  behave like a normal node. */

  if (thePacket.isAddressedTo(this)) {
    this.print
    (thePacket)
  } else super.accept(thePacket)