MetaclassTalk
a Testbed for Exploring Programming Paradigms

Noury Bouraqadi
Computer Science Laboratory (CSL)
Ecole des Mines de Douai
France

Starting Point

- ***Goal: Experiment Various Paradigms***
  - *Multiple-Inheritance*
  - *Aspect-Oriented Programming*
  - *Mixin-based Inheritance*
  - *Types*
  - *...*
- ***Requirements: a "Ball of Mud" (© R. Fateman ;-)***
  - *OO Programming Language*
  - *API + "framework" to ease changes*

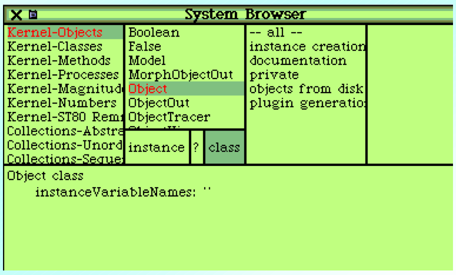
Why Extend Smalltalk?

✓ ***Powerfull (Reflective Facilities)***
✓ ***Simple & Uniform***

✗ ***Difficult to define new kinds of classes***
  ✗ *Metaclasses are Implicit*
  ✗ *Parallel Inheritance Hierarchies*
✗ ***Complex to change the evaluation process***
✗ ***No API or framework easing extensions***

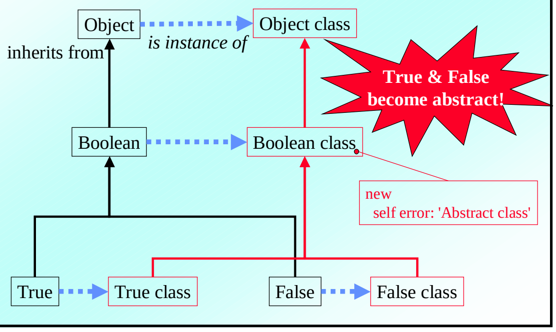
Smalltalk Metaclasses are Implicit

Metaclasses: Classes which instances are also classes



```
Object class
    instanceVariableNames: ''
```

Smalltalk Parallel Inheritance Hierarchies

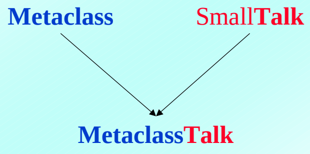


Outline

- ***What is MetaclassTalk?***
- ***Class Properties Reuse Using Mixins***
- ***Aspect-Oriented Programming Using MetaclassTalk***
- ***Conclusion***

What is MetaclasTalk?

Meta… what?



Metaclass **SmallTalk**

MetaclassTalk

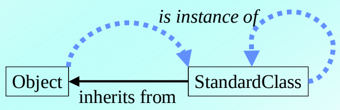
Programming with Explicit Metaclasses

What to do with Metaclasses?

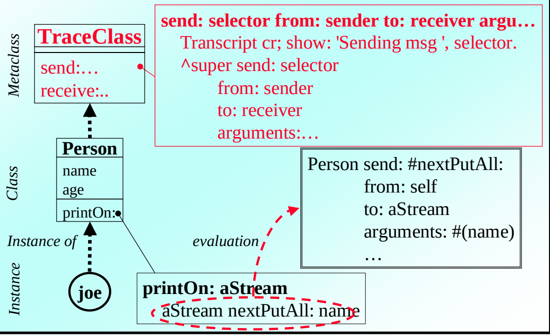
- ***Class Properties***
  - *Definition of new kinds of classes*
- ***Control of Program Execution***
  - *instance creation*
  - *IV reads & writes*
  - *message sends & reception*
  - *method lookup & evaluation*

MetaclassTalk "Conceptual" Kernel

The ObjVLisp Model [Cointe 87]
= The Smalltalk-76 Kernel + ability to subclass the root metaclass



Example of Message Sending Control



TraceClass
send:…
receive:..

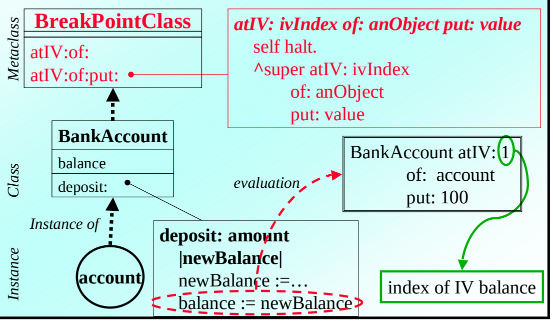
```
send: selector from: sender to: receiver argu…
    Transcript cr; show: 'Sending msg ', selector.
    ^super send: selector
         from: sender
         to: receiver
         arguments:…
```

Person
name
age
printOn:

```
Person send: #nextPutAll:
        from: self
        to: aStream
        arguments: #(name)
        …
```

```
printOn: aStream
  aStream nextPutAll: name
```

Example of IV Read-Write Control



BreakPointClass
atIV:of:
atIV:of:put:

```
atIV: ivIndex of: anObject put: value
   self halt.
   ^super atIV: ivIndex
         of: anObject
         put: value
```

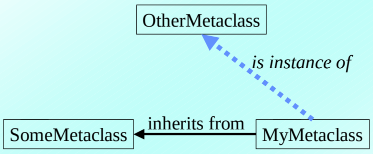
BankAccount
balance
deposit:

```
BankAccount atIV: 1
        of:  account
        put: 100
```

index of IV balance

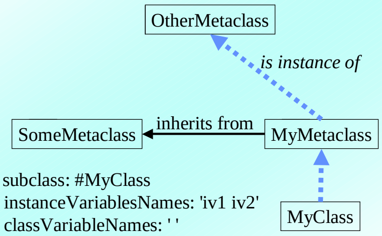
```
deposit: amount
   |newBalance|
   newBalance :=…
   balance := newBalance
```

How to Use MetaclassTalk?



```
SomeMetaclass subclass: #MyMetaclass
        instanceVariablesNames: 'x y'
        classVariableNames: ' '
        poolDictionaries: ' '
        category: 'MetaclassTalk-Example'
        metaclass: OtherMetaclass
```

How to Use MetaclassTalk?



```
Object subclass: #MyClass
        instanceVariablesNames: 'iv1 iv2'
        classVariableNames: ' '
        poolDictionaries: ' '
        category: 'MetaclassTalk-Example'
        metaclass: MyMetaclass
```

Some Possible Uses of the MetaclassTalk's MOP

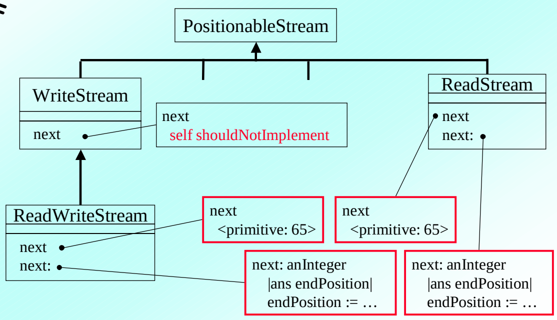
	Allocate	New	Read	Write	Send	Receive	Lookup	Apply
Abstract Class		X					X	
Asynchronous Communication					X	X		
Break Point			X	X		X		
Lazy Memory Allocation	X		X	X				
Message Caching						X		
Multiple Inheritance							X	
Persistent Objects (Data Base)	X		X	X				
Sole Instance (Pattern)		X						
Subject Class (Observer Pattern)				X				X
Synchronisation			X	X				X
Tracing			X	X	X	X		
Type Checking							X	

Class Properties Reuse Using Mixins

What are Mixins?

- **Context:**
  - *Single inheritance*
  - *Unreleated class hierarchies*
- **Problem:**
  - *Reuse shared properties*
  - *Avoid code duplication*
- **Solution:**
  - *Mixin = Subclass builder [Bracha 90]*

Example of Code Duplication



PositionableStream

WriteStream
next

next
self shouldNotImplement

ReadStream
next
next:

ReadWriteStream
next
next:

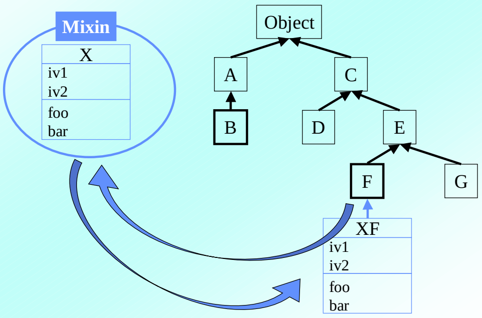
next
<primitive: 65>

next
<primitive: 65>

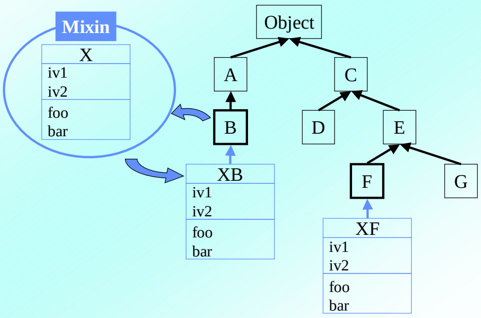
next: anInteger
|ans endPosition|
endPosition := …

next: anInteger
|ans endPosition|
endPosition := …

How Do Mixins Work?



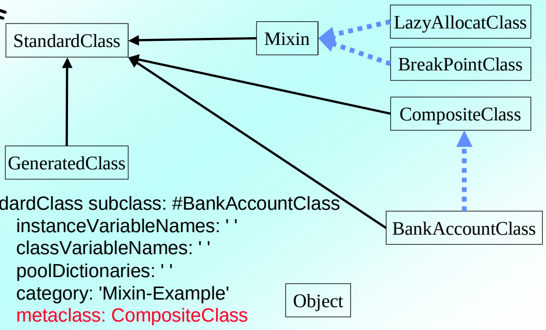
How Do Mixins Work?



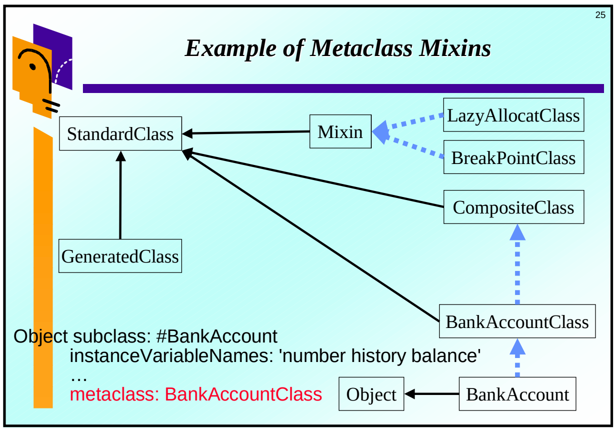
Mixins Implementation in MetaclassTalk

- ***3 metaclasses (i.e. 3 class properties):***
  - *Mixins*
  - *Classes generated by mixins*
  - *Classes that make use of mixins*
- ***Generated classes are updated on mixin change***
- ***Use of Smalltalk reflective facilities:***
  - *Method Compilation*
  - *Class Building*
  - *Adds & Removals of Methods and Instance Variables*

Example of Metaclass Mixins



StandardClass subclass: #BankAccountClass
instanceVariableNames: ' '
classVariableNames: ' '
poolDictionaries: ' '
category: 'Mixin-Example'
metaclass: CompositeClass

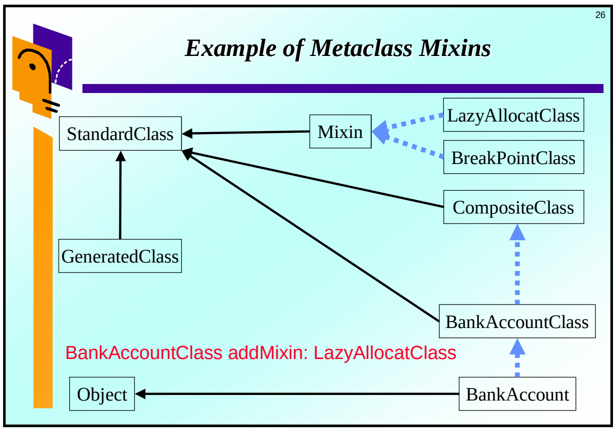


Example of Metaclass Mixins

StandardClass
Mixin
LazyAllocatClass
BreakPointClass
CompositeClass
GeneratedClass
BankAccountClass

```
Object subclass: #BankAccount
         instanceVariableNames: 'number history balance'
         ...
         metaclass: BankAccountClass
```

Object
BankAccount

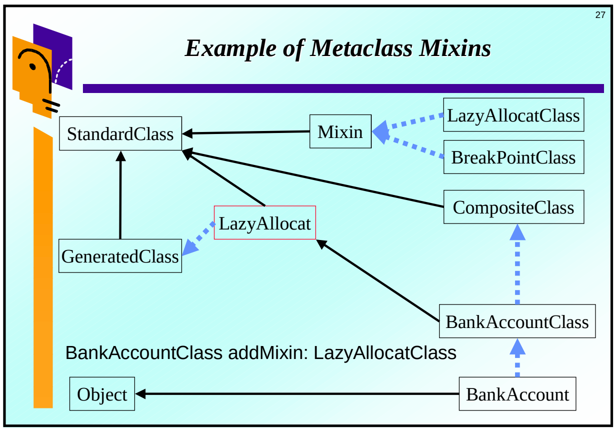


Example of Metaclass Mixins

StandardClass
Mixin
LazyAllocatClass
BreakPointClass
CompositeClass
GeneratedClass
BankAccountClass

```
BankAccountClass addMixin: LazyAllocatClass
```

Object
BankAccount

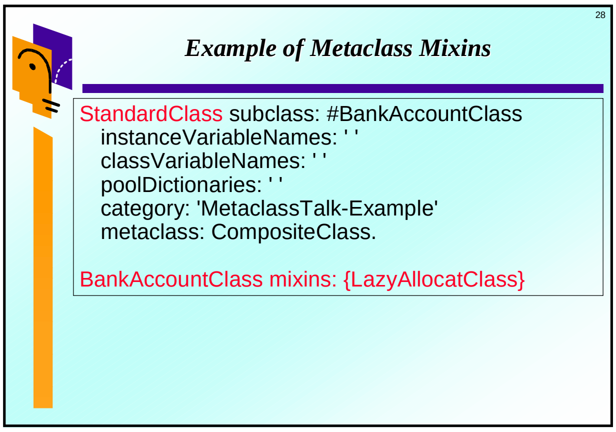


Example of Metaclass Mixins

StandardClass
Mixin
LazyAllocatClass
BreakPointClass
LazyAllocat
CompositeClass
GeneratedClass
BankAccountClass

```
BankAccountClass addMixin: LazyAllocatClass
```

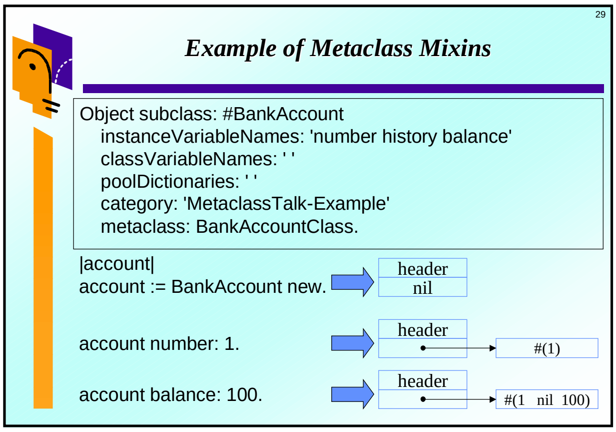
Object
BankAccount



Example of Metaclass Mixins

```
StandardClass subclass: #BankAccountClass
    instanceVariableNames: ' '
    classVariableNames: ' '
    poolDictionaries: ' '
    category: 'MetaclassTalk-Example'
    metaclass: CompositeClass.

BankAccountClass mixins: {LazyAllocatClass}
```



Example of Metaclass Mixins

```
Object subclass: #BankAccount
    instanceVariableNames: 'number history balance'
    classVariableNames: ' '
    poolDictionaries: ' '
    category: 'MetaclassTalk-Example'
    metaclass: BankAccountClass.
```

```
|account|
account := BankAccount new.
```

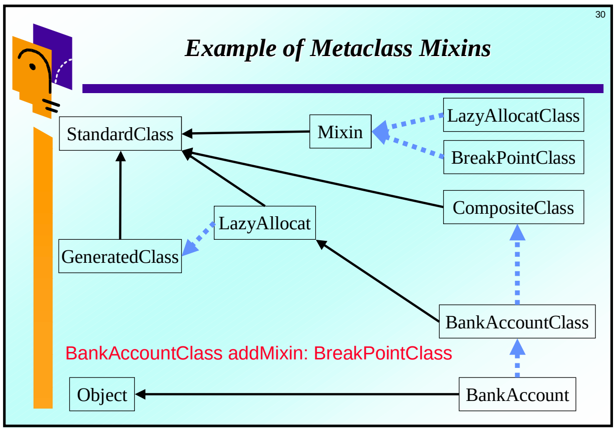
header
nil

```
account number: 1.
```

header
#(1)

```
account balance: 100.
```

header
#(1 nil 100)



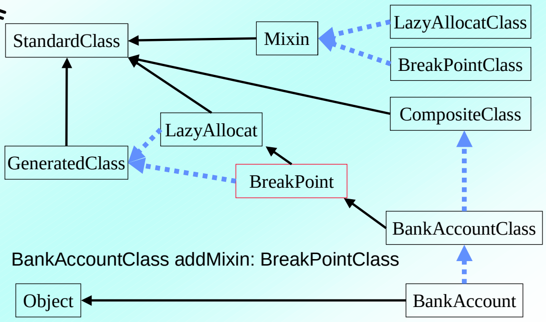
Example of Metaclass Mixins

StandardClass
Mixin
LazyAllocatClass
BreakPointClass
LazyAllocat
CompositeClass
GeneratedClass
BankAccountClass

```
BankAccountClass addMixin: BreakPointClass
```

Object
BankAccount

Example of Metaclass Mixins



BankAccountClass addMixin: BreakPointClass

Example of Metaclass Mixins

StandardClass subclass: #BankAccountClass
…
metaclass: CompositeClass.

BankAccountClass mixins:
{LazyAllocatClass, BreakPointClass}

Object subclass: #BankAccount
instanceVariableNames: 'number history balance'
…
metaclass: BankAccountClass.

BankAccount breakOnSelectors: #( ) andIVs: #( )

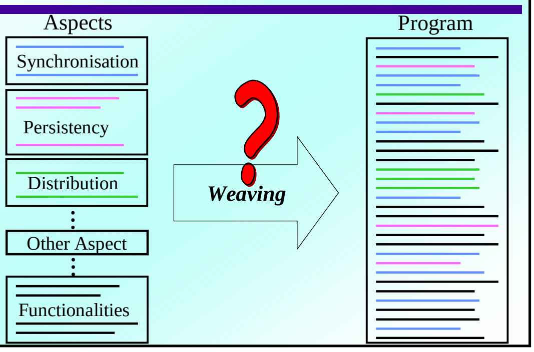
Mixins: a Usefull Extension

- ***Experiment of a New Paradigm***
- ***A Tool for Composing class Properties***

Aspect-Oriented Programming Using MetaclassTalk

Experimenting
a New Programming Paradigm

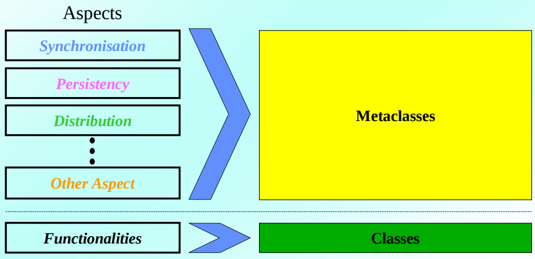
AOP



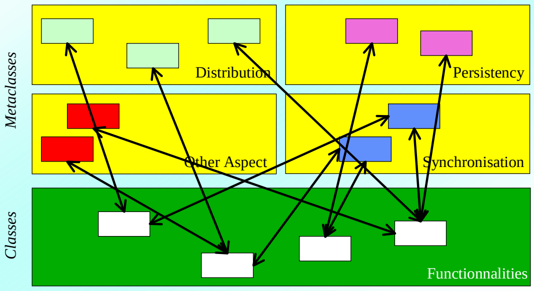
Issues & Solutions

- ***How to define isolated aspects?***
  - *Metaclasses*
- ***How to weave aspects to build an application?***
  - *The "instance-of" link*
  - *Mixin-based inheritence*
- ***What about reuse?***
  - *Generic Metaclasses*

A First Natural Separation



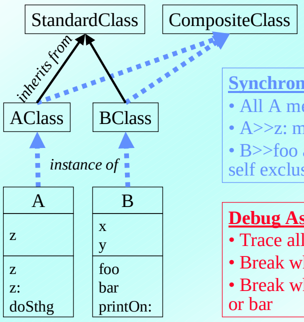
How to Isolate & Weave Aspects?



Aspects in MetaclassTalk

1. **Set of Class Properties: Metaclasses**
2. **Set of Classes**
3. **Class-Metaclass relationships**
4. **Class initialization (Properties setup)**

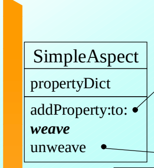
Example - Starting Point



Synchronisation Aspect
- All A methods are mutually exclusive
- A>>z: method is self exclusive
- B>>foo and B>>bar are both self exclusives and mutually exclusives

Debug Aspect
- Trace all behavior of A
- Break when accessing to x in aB
- Break when aB receives messages foo or bar

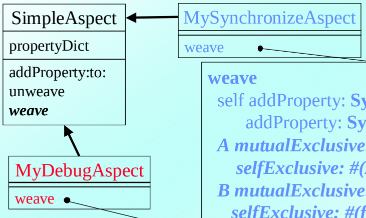
Easy Implementation



```
addProperty: aMixin to: aClass
  |properties|
  aClass class addMixin: aMixin.
  properties := propertyDict at: aClass
          ifAbsentPut: [OrderedCollection new].
  properties add: aMixin
```

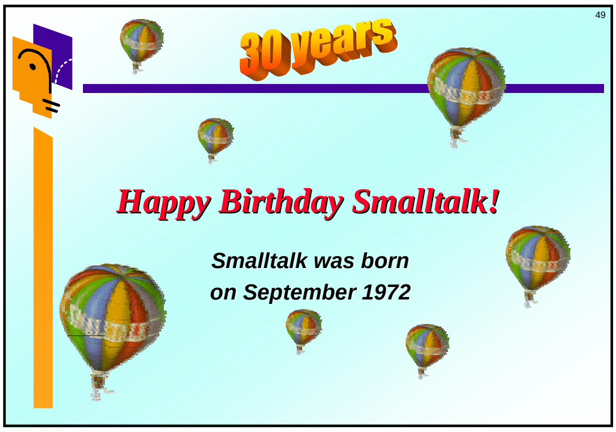
```
unweave
  |properties|
  propertyDict keysAndValuesDo: [:aClass :mixins|
        mixins do: [:mix| aClass class removeMixin: mix]].
  properties := Dictionary new
```

Example - Aspects Implementation



```
weave
  self addProperty: SynchronizeClass to: A;
        addProperty: SynchronizeClass to: B.
  A mutualExclusive: #(z z: doSthg);
     selfExclusive: #(z:).
  B mutualExclusive: #(foo bar);
     selfExclusive: #(foo bar).
```

```
weave
  self addProperty: TraceClass to: A;
        addProperty: BreakPointClass to: B.
  B breakOnSelectors: #(foo bar) andIvs: #(x)
```



30 years

Happy Birthday Smalltalk!

Smalltalk was born
on September 1972

Questions? Comments?

Download
http://csl.ensm-douai.fr/MetaclassTalk