# Advanced Classes

- Indexed Classes
- Classes as Objects
- Class Instance Variables and Methods
- Class Variables
- Pool Dictionaries

## Two Views on Classes

- Named or indexed instance variables
  - Named: 'addressee' of Packet
  - Indexed: Array

- Or looking at them in another way:
  - Objects with pointers to other objects
  - Objects with arrays of bytes (word, long)

- Difference for efficiency reasons: arrays of bytes (like C strings) are faster than storing an array of pointers, each pointing to a single byte.

## Types of Classes

| Indexed | Named | Definition Method | Examples |
|---|---|---|---|
| No | Yes | #subclass:... | Packet |
| Yes | Yes | #variableSubclass: | Array |
| Yes | No | #variableByteSubclass | String |

- Method related to class types: #isPointers, #isBits, #isBytes, #isFixed, #isVariable, #kindOfSubclass

## Constraints in VW

- Classes defined using #subclass: support any kind of subclasses
- Classes defined using #variableSubclass: can only have: variableSubclass: or variableByteSubclass: subclasses
- classes defined using #variableByteSubclass
- can only be defined if the superclass has no defined instance variable
- pointer classes and byte classes don't mix
- only byte subclasses

## Indexed Classes

- For classes that need a variable number of instance variables

- Example: the class Array

```
ArrayedCollection variableSubclass: #Array
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
  category: 'Collections-Arrayed'

  Array new: 4 -> #(nil nil nil nil)
  #(1 2 3 4) class isVariable -> true
```

## Indexed Classes / Instance Variables

- Indexed variable is implictly added to the list of instance variables
- Only one indexed instance variable per class
- Access with #at: and #at:put:
- (#at:put: answers the value, not the receiver)
- First access: anInstance at: 1
- #size returns the number of indexed instance variables
- Instantiated with #new: max

```
|t|
t := (Array new: 4).
t at: 2 put: 'lulu'.
t at: 1 -> nil
```

- Subclasses should also be indexed

## Advanced Classes

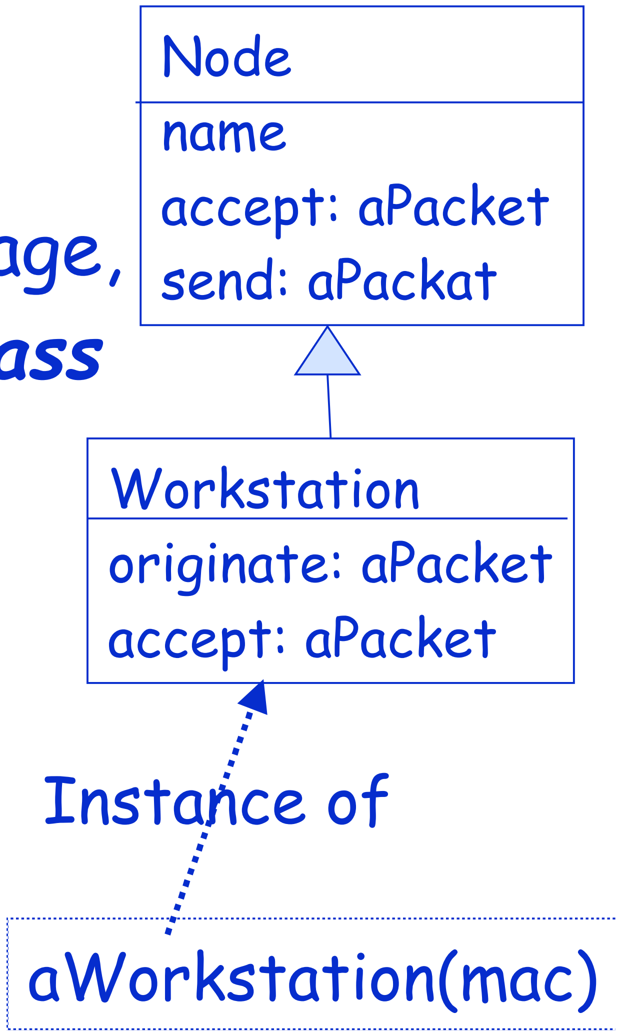
- Indexed Classes
- Classes as Objects
- Class Instance Variables and Methods
- Class Variables
- Pool Dictionaries

## The Meaning of is-a

- A class defines the structure and the behavior of all its instances.
- Each instance possesses its own set of values.
- Instances share the behavior defined in their class with other instances via the instance of link.

# The Meaning of Is-a

- Every object is an instance of a class.
- When anObject receives a message, the method is looked up in ***its class***
- And it continues possibly in its superclasses
- Every class is ultimately a subclass of Object (except Object).



## Remember: ...

- Example: macNode name
- macNode is an instance of Workstation

=> name is looked up in the class Workstation

- name is not defined in Workstation

=> lookup continues in Node

- name is defined in Node

=> lookup stops + method executed

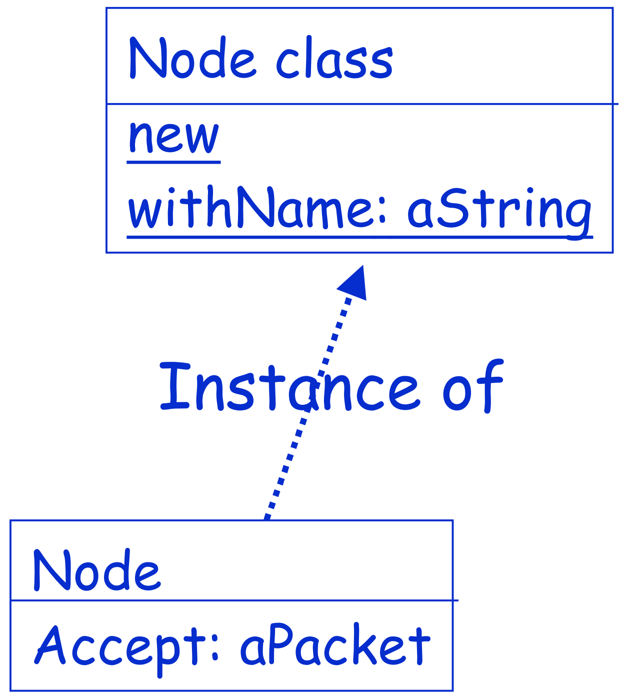
## Uniformity between classes and objects

- The same principle is true for objects and classes
- Same lookup strategy
- Everything that works at instance levels works at instance level

## A Class is an Object too…

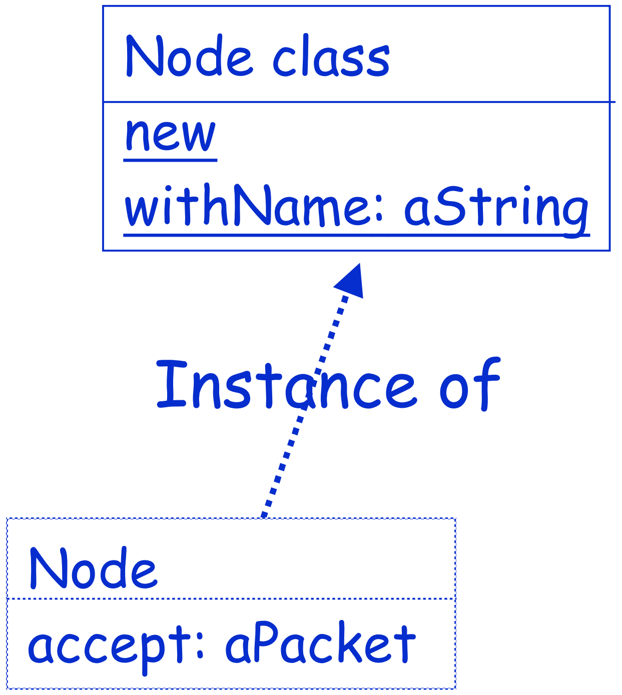
- Every class (X) is the unique instance of its associated metaclass named X class
- Example:

  Node is instance of
  "Node class"
  Point is the unique instance of
  "Point class"

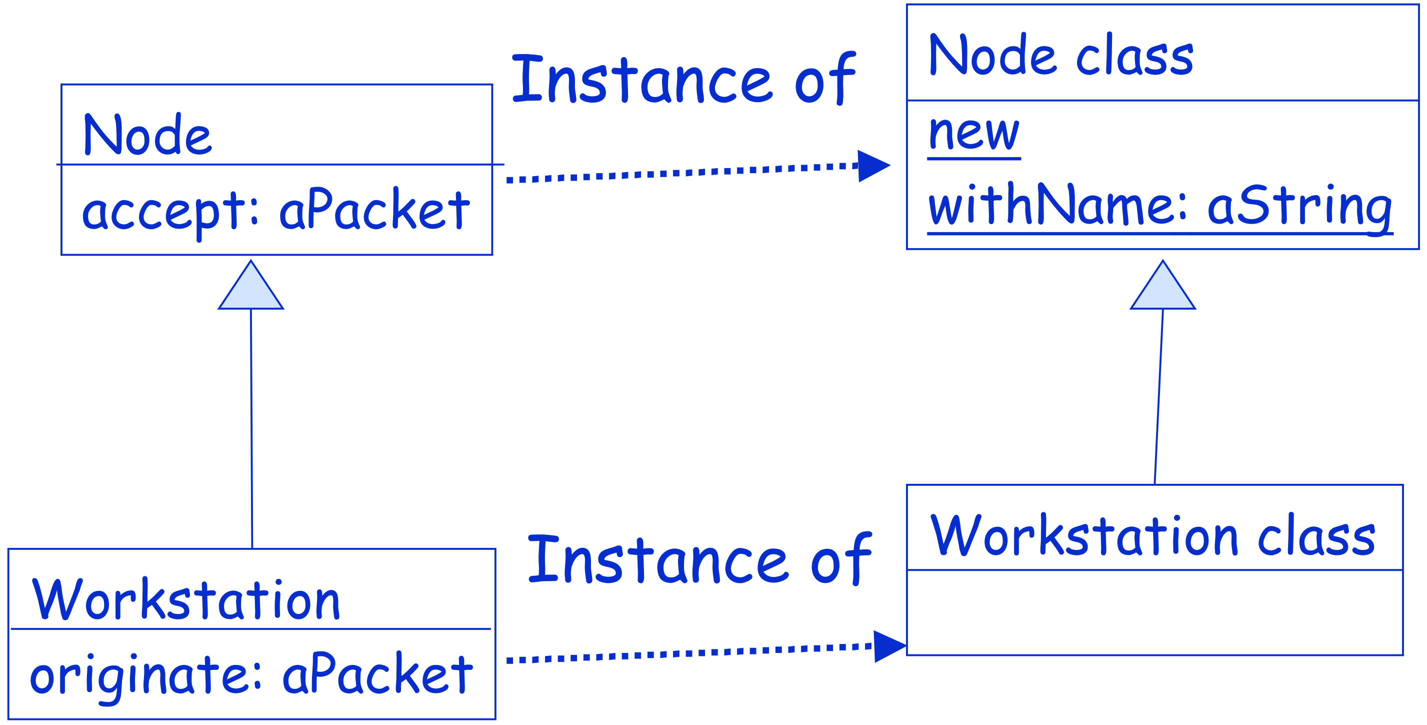


## A Class is an Object too…

- So messages sent to a class are looked up into the class of the class.
- Node withName: #node1
  - Node is an instance of "Node class"
  - withName: is looked up in the class "Node class"
  - withName: defined in "Node class"
  - lookup stops + method executed



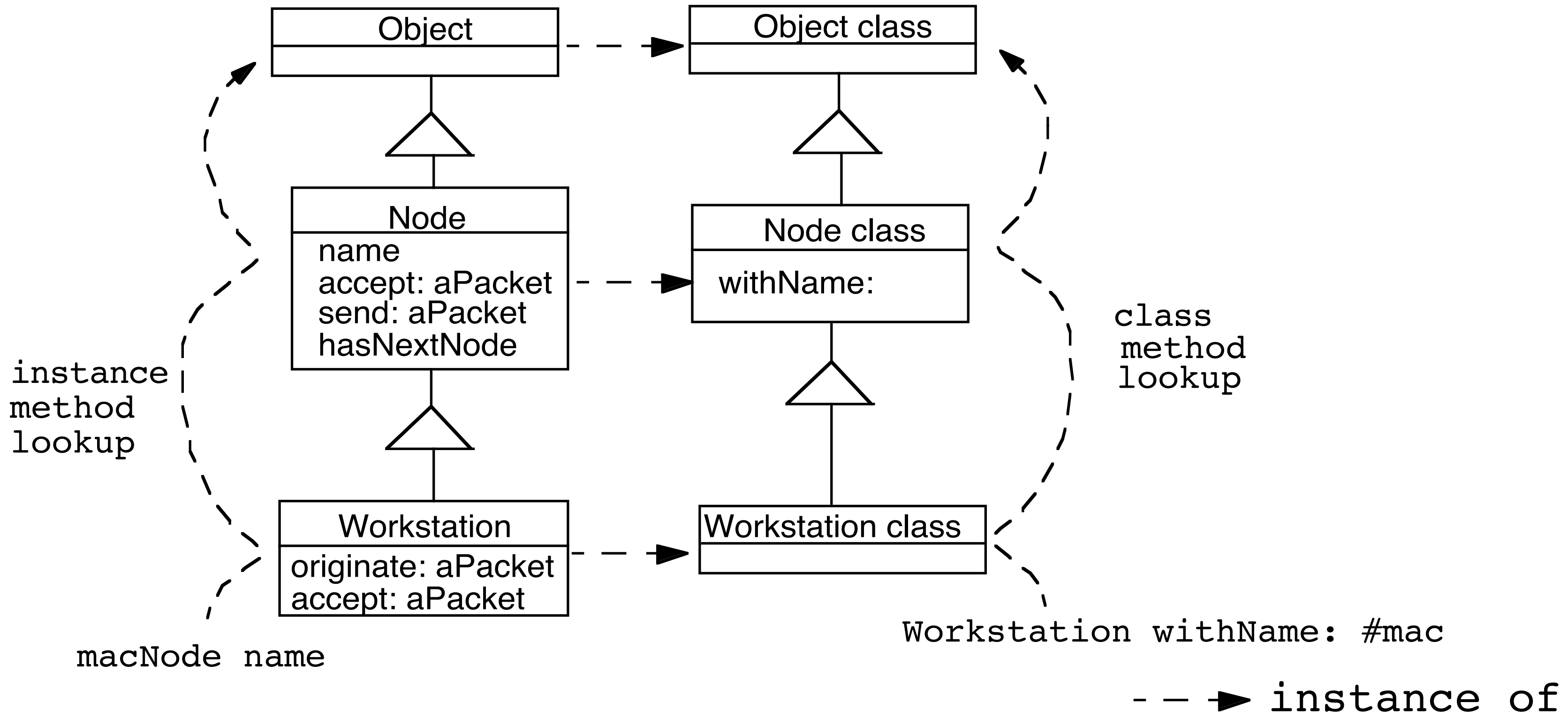
# Parallel Inheritance between classes



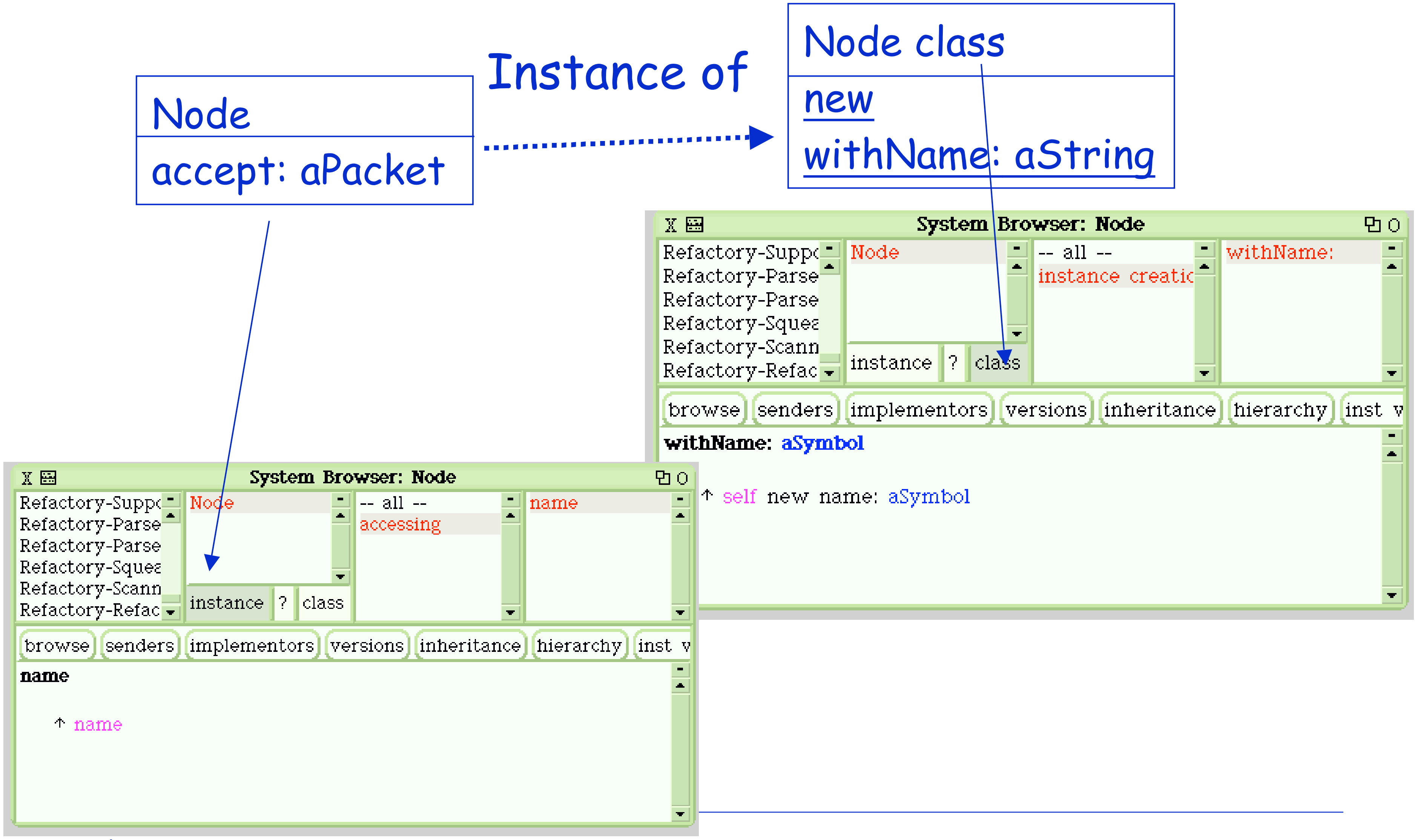
## Parallel inheritance between classes

- Workstation withName: #mac
  - Workstation is an instance of Workstation class fi withName: is looked up in the class Workstation class
  - withName: is not defined in Workstation class fi lookup continues in the superclass of Workstation class = Node class
  - withName: is defined in Node class fi lookup stops
    + method executed

# Lookup and Class Methods



# About the Buttons



```
withName: aSymbol

↑ self new name: aSymbol
```

```
name

    ↑ name
```

## Where is new defined?

- Node new: #node1
  - Node is an instance of Node class => new: is looked up in the class Node class
  - new: is not defined in Node class => lookup continues in the superclass of Node class = Object class
  - new: is not defined in Object class => lookup continues in the superclass of Node class ....Class, ClassDescription, Behavior
  - new: is defined in Behavior => lookup stops + method executed.
- This is the same for Array new: 4
  - new: is defined in Behavior (the ancestor of Array class)

Hint: Behavior is the essence of a class. ClassDescription represents the extra functionality for browsing the class. Class supports poolVariable and classVariable.

## Recap

- Everything is an object
- Each object is instance of one class
- A class (X) is also an object, the sole instance of its associated metaclass named X class
- An object is a class if and only if it can create instances of itself.
- A Metaclass is just a class whose instances are classes
  - Point class is a metaclass as its instance is the class Point

# Metaclass  Responsibilities

- instance creation
- class information (inheritance link, instance variables, method compilation...)
- Examples:

```
Node allSubclasses -> OrderedCollection (WorkStation
OutputServer Workstation File        Server PrintServer)
LanPrinter allInstances -> #()
Node instVarNames -> #('name' 'nextNode')
Workstation withName: #mac -> aWorkstation
Workstation selectors  -> IdentitySet (#accept: #originate:)
Workstation canUnderstand: #nextNode -> true
```

## Class Methods

- As any object a metaclass can have methods that represent the behavior of its instance: a class
- Uniformity => Same rules as for normal classes
- No constraint: just normal methods
- Can only access instance variable of the class:
- Example: NetworkManager class>>new can only access uniqueInstance class instance variable and not instance variables (like nodes).

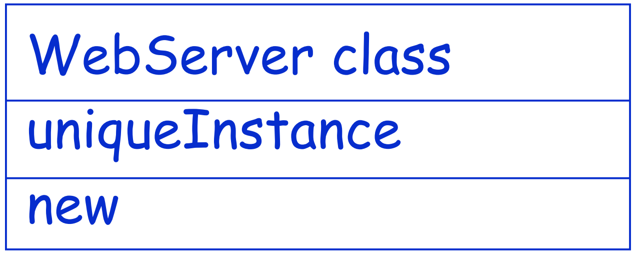
- Default Instance Creation class method:
  - new/new: and basicNew/basicNew: (see Direct Instance Creation)
    - Packet new
- Specific instance creation method
  - Packet send: 'Smalltalk is fun' to: #lpr

## Class Instance Variables

- Like any object, a class is an instance of a class that can have instance variables that represent the state of a class.
- When Point defines the new instance variable z, the instances of Point have 3 value (one for x, one for y, and one for z)
- When a metaclass defines a new instance variable, then its instance (a Class) gets a new value in addition to subclass, superclasses, methodDict…

## Example: the Singleton Pattern

- A class having only one instance
- We keep the instance created in an instance variable



```
WebServer class
  instanceVariableNames: 'uniqueInstance'

WebServer class>>new
  self error: 'You should use uniqueInstance to get the unique
  instance'

WebServer class>>uniqueInstance
   uniqueInstance isNil
     ifTrue: [ uniqueInstance := self basicNew initialize].
   ^ uniqueInstance
```

## Singleton (ii)

WebServer being an instance of WebServer class has an instance variable named uniqueInstance.

WebServer has a new value that is associated with uniqueInstance

## Design Implications

- An instance variable of a class can be used to represent information shared by all the instances of the class. However, you should use class instance variables to represent the state of the class (like the number of instances, ...) and use shared Variable instead (next Section).

## About Behavior

- Behavior is the first metaclass. All other metaclasses inherit from it
- Behavior describes the minimal structure of a class:
- superclass and subclasses
- method dictionary
- format (instance variable compressed description)

```
Object subclass: #Behavior
   instanceVariableNames: 'superclass methodDict format subclasses '
   classVariableNames: ''
   poolDictionaries: ''
   category: 'Kernel-Classes'
```

## Example of Queries

```
Packet superclass -> Object
Packet subclasses ->  #()
Packet selectors
-> IdentitySet (#originator: #addressee: #addressee
  #isOriginatedFrom: #printOn: #isAddressedTo:
  #originator #initialize #contents #contents:)
Packet allInstVarNames
-> OrderedCollection ('addressee' 'originator'
                        'contents' 'visitedNodes')
Packet isDirectSubclassOf: Object -> true
```

## Advanced Classes

- Indexed Classes
- Classes as Objects
- Class Instance Variables and Methods
- Class Variables
- Pool Dictionaries

## classVariable = Shared Variables

- How to share state between all the instances of a class: Use a classVariable
- a classVariable is shared and directly accessible by all the instances of the class and subclasses
- A pretty bad name: should have been called Shared Variables (now fixed in VW)
- Shared Variable => begins with an uppercase letter
- a classVariable can be directly accessed in instance methods and class methods

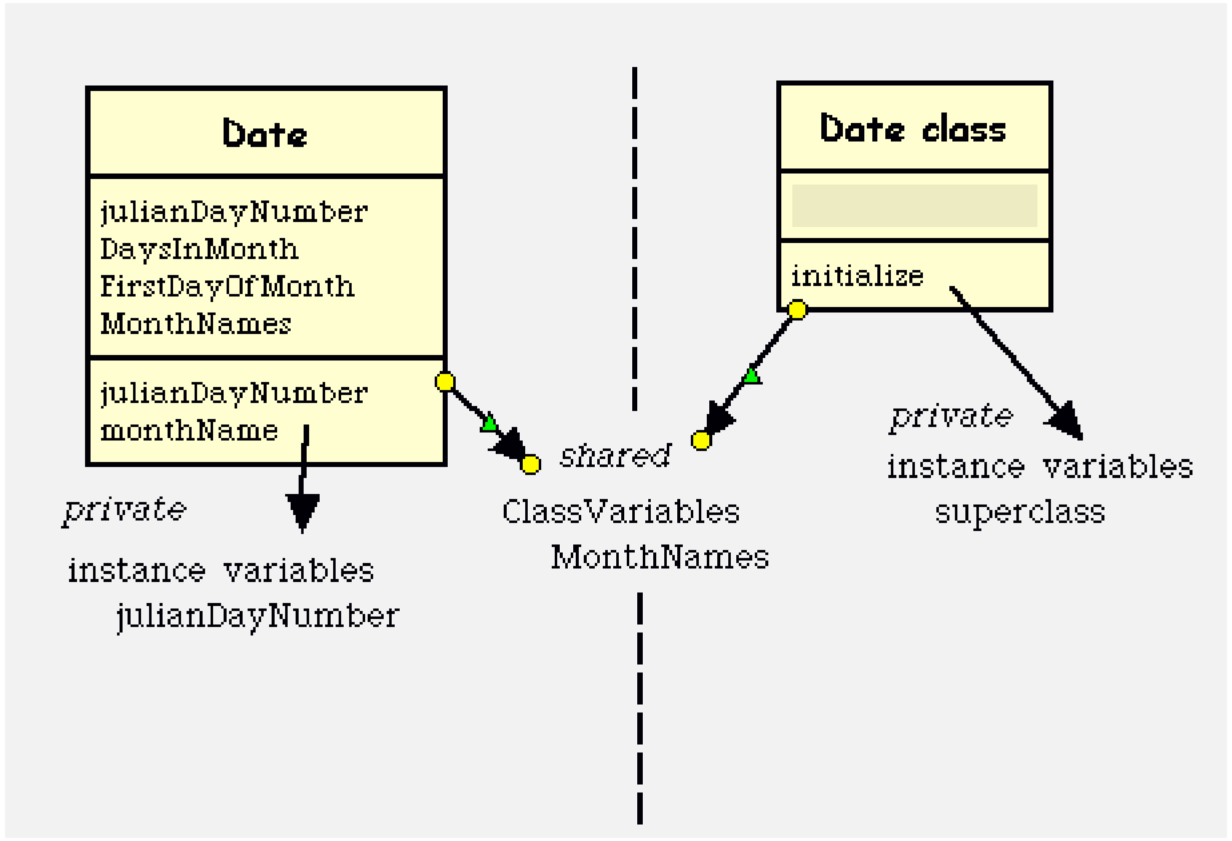
## classVariable = SharedVariable

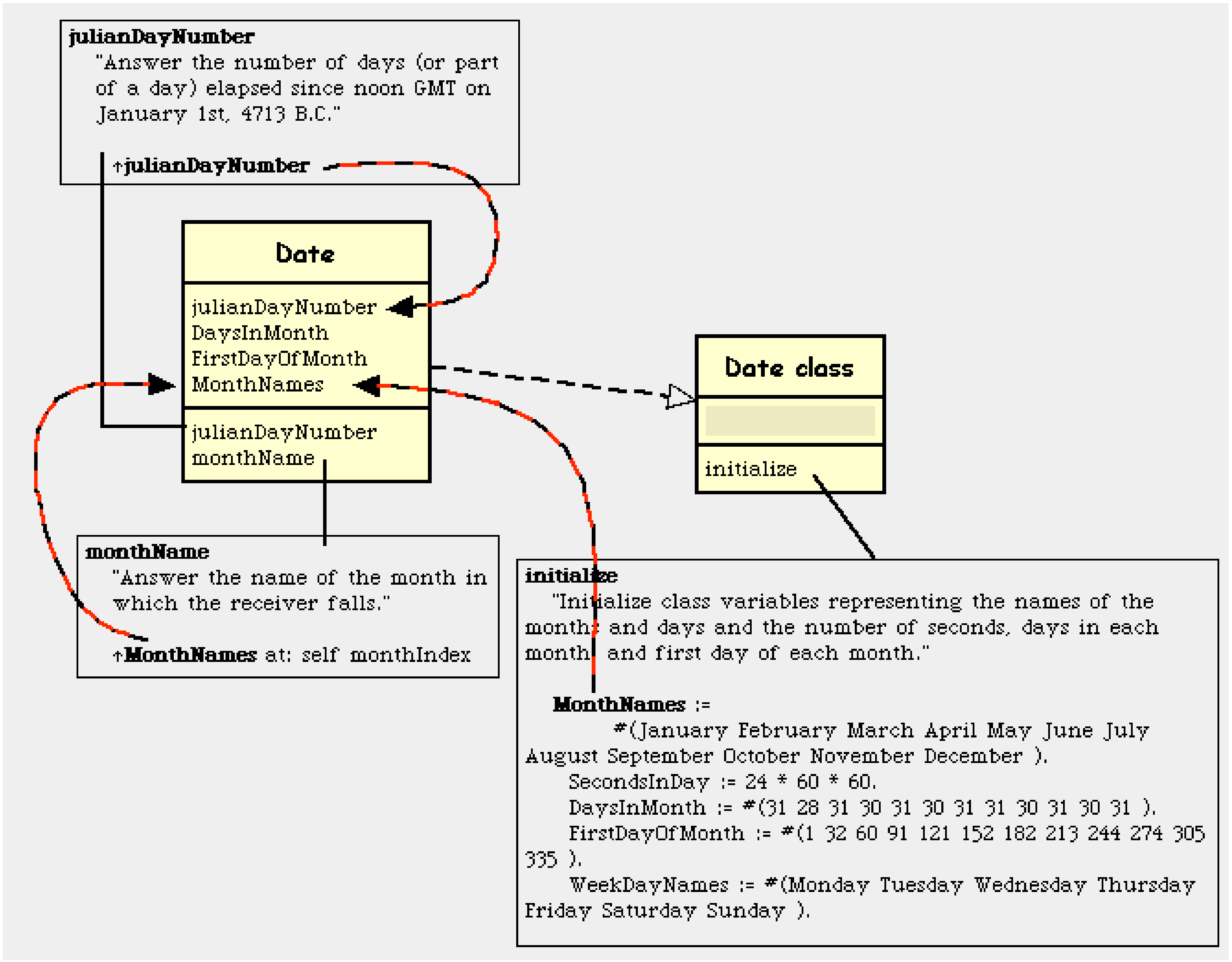
```
Magnitude subclass: #Date
   instanceVariableNames: 'julianDayNumber '
   classVariableNames: 'DaysInMonth FirstDayOfMonth
   MonthNames SecondsInDay WeekDayNames '
   poolDictionaries: ''
   category: 'Kernel-Magnitudes'
```

## Date class>>initialize

```
"Initialize class variables representing the names of the months and
days and the number of seconds, days in each month, and first day of
each month."

MonthNames  := #(January February March April May June July
August September October November December ).
SecondsInDay := 24 * 60 * 60.
DaysInMonth := #(31 28 31 30 31 30 31 31 30 31 30 31 ).
FirstDayOfMonth := #(1 32 60 91 121 152 182 213 244 274 305 335 ).
WeekDayNames := #(Monday Tuesday Wednesday Thursday Friday
Saturday Sunday ).
```

# ClassVariable vs. Instance Variables





```
julianDayNumber
    "Answer the number of days (or part
    of a day) elapsed since noon GMT on
    January 1st, 4713 B.C."

    ↑julianDayNumber
```

```
monthName
    "Answer the name of the month in
    which the receiver falls."

    ↑MonthNames at: self monthIndex
```

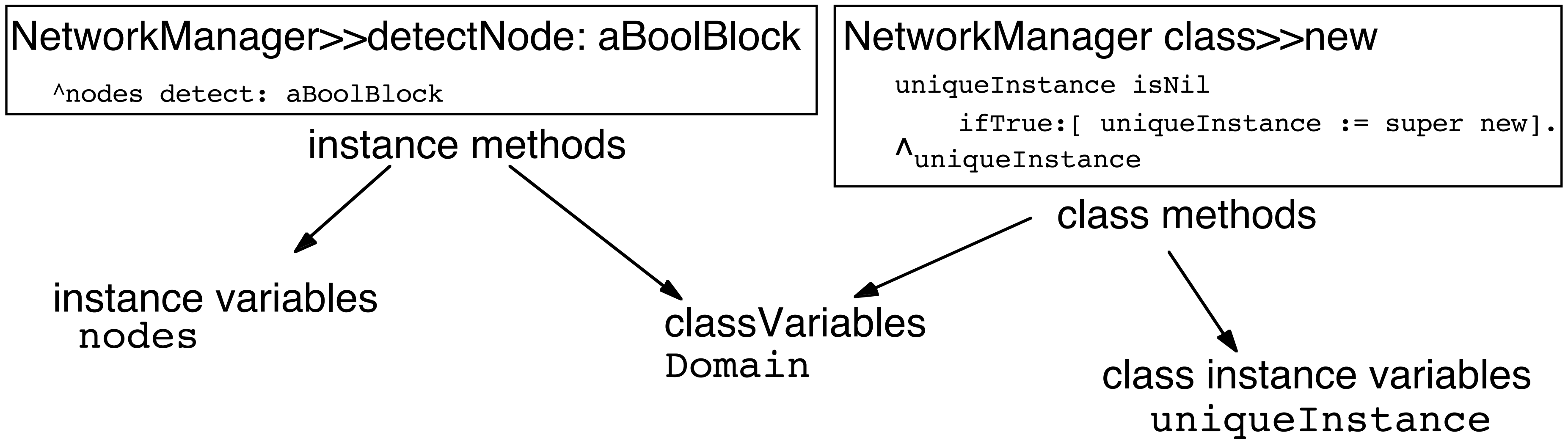
```
initialize
   "Initialize class variables representing the names of the
months and days and the number of seconds, days in each
month, and first day of each month."

   MonthNames :=
         #(January February March April May June July
August September October November December ).
     SecondsInDay := 24 * 60 * 60.
     DaysInMonth := #(31 28 31 30 31 30 31 31 30 31 30 31 ).
     FirstDayOfMonth := #(1 32 60 91 121 152 182 213 244 274 305
335 ).
     WeekDayNames := #(Monday Tuesday Wednesday Thursday
Friday Saturday Sunday ).
```

## Class Instance Variables vs classVariables

- a classVariable is shared and directly accessible by all the instances and subclasses

- Class *instance variables*, just like normal *instance variables*, can be accessed only via class message and accessors:
  - an instance variable of a class is private to this class.
- Take care: when you change the value of a classVariable the whole inheritance tree is impacted!

## Summary of Variable Visibility



```
NetworkManager>>detectNode: aBoolBlock
	^nodes detect: aBoolBlock
```

```
NetworkManager class>>new
	uniqueInstance isNil
		ifTrue:[ uniqueInstance := super new].
	^uniqueInstance
```

## Example From The System: Geometric Class

```
Object subclass: #Geometric
  instanceVariableNames: ''
  classVariableNames: 'InverseScale Scale '
  ...

Geometric class>>initialize
  "Reset the class variables."

  Scale := 4096.
  InverseScale := 1.0 / Scale
```

## Circle

```
Geometric subclass: #Circle
        instanceVariableNames: 'center radius'
        classVariableNames: ''
Circle>>center
        ^center
Circle>>setCenter: aPoint radius: aNumber
        center := aPoint.
        radius := aNumber
Circle>>area
        | r |
        r := self radius asLimitedPrecisionReal.
        ^r class pi * r * r

Circle>>diameter
        ^self radius * 2

Circle class>>center: aPoint radius: aNumber
        ^self basicNew setCenter: aPoint radius: aNumber
```

## Class Instance Variable vs. classVariable

- ClassVariables can be used in conjunction with instance variables to cache some common values that can be changed locally in the classes.

## Examples

in the Scanner class a table describes the types of the characters (strings, comments, binary....). The original table is stored into a classVariable, its value is loaded into the instance variable. It is then possible to change the value of the instance variable to have a different scanner.

```
Object subclass: #Scanner
                    instanceVariableNames: 'source mark prevEnd hereChar
  token tokenType buffer typeTable '
                    classVariableNames: 'TypeTable '
                    category: 'System-Compiler-Public Access'
```

## Advanced Classes

- Indexed Classes
- Classes as Objects
- Class Instance Variables and Methods
- Class Variables
- Pool Dictionaries

## poolVariables

- Shared variable => begins with a uppercase letter.
- Variable shared by a group of classes not linked by inheritance.
- Each class possesses its own pool dictionary (containing poolVariables).
- They are not inherited.
- DON'T USE THEM!

## Examples of PoolDictionaries

- from the System: the class Text

```
CharacterArray subclass: #Text
        instanceVariableNames: 'string runs '
        classVariableNames: ''
        poolDictionaries: 'TextConstants '
        category: 'Collections-Text'
```

- Elements stored into TextConstants like Ctrl, CR, ESC, Space can be directly accessed from all the classes like ParagraphEditor....
- On VW poolDictionary should not be an IdentityDictionary

## Example of PoolVariables

```
Smalltalk at: #NetworkConstant put: Dictionary new.
NetworkConstant at: #rates put: 9000.
Packet>>computeAverageSpeed
	...
	NetworkConstant at: #rates
```

- Equivalent to :

```
Object subclass: #Packet
  instanceVariableNames: 'contents addressee originator '
  classVariableNames: 'Domain'
  poolDictionaries: 'NetworkConstant'
Packet>>computeAverageSpeed
  ...
  ... rates
```

- rates is directly accessed in the global dictionary NetworkConstant.

## What you should know

- Classes are objects too
- Class methods are just methods on objects that are classes
- Classes are also represented by instance variables (class instance variables)
- (Shared Variables) ClassVariables are shared among subclasses and classes (metaclass)