The Programming Language Ruby

```
[1,2,3,4,5].each { |x| puts x }
```

Michael Stal, Senior Principal Engineer

Siemens Corporate Technology, CT SE 2

E-mail: Michael.Stal@siemens.com

Software & Engineering Architecture

Agenda

- **Ruby Background**
- **Ruby Type System**
- **Statements**
- **A Small Tour through the Libraries**
- **Tools and More**
- **Resources**

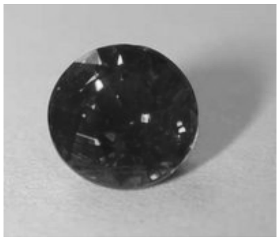
Software & Engineering Architecture

History and Motivation of Ruby

- Ruby is a programming language developed by Yukihiro Matsumoto (a.k.a. Matz) in 1993
- It was originally designed to be a better Perl than Perl (that‘s the reason for its name)
- It is available on multiple platforms such as Linux, MacOS X, Windows
- According to Matz its primary application domains are Text processing, CGI-, Network-, GUI-, XML-programming, Prototyping, Programming education
- Ruby has adopted features from languages such as Perl, Lisp, Smalltalk
- It is very popular in Asia, especially in Japan

What the hell is Ruby?

- Paradigm: Pure OO language
- Simple and without surprises: Easy to learn and understand
- Potential: Powerful and expressive
- Add Ons: Rich library support
- Productive: Rapid development
- Non commercial: Open Source
- Robust: Garbage Collector on Board
- Flexible: Untyped, dynamic language
- And of course: It‘s cool!



A Small Example (1)

Comments in Ruby start with #

```
#This is just to show you an example

class MyFirstRubyProgram
  def SayHello(name=“Micha“)
    puts "Hello, #{name}!"
  end
end

MyFirstRubyProgram.new.SayHello("OOP 2005")
```

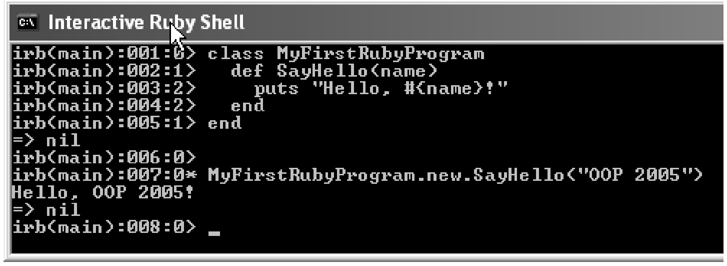
Here we define a class

Extremely impressing instance method with default parameter☺

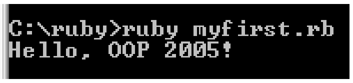
Here we instantiate the class and call a method

A Small Example (2)

- **Running the example in the interactive Ruby Shell (IRB) and with the command line interpreter (Windows version):**



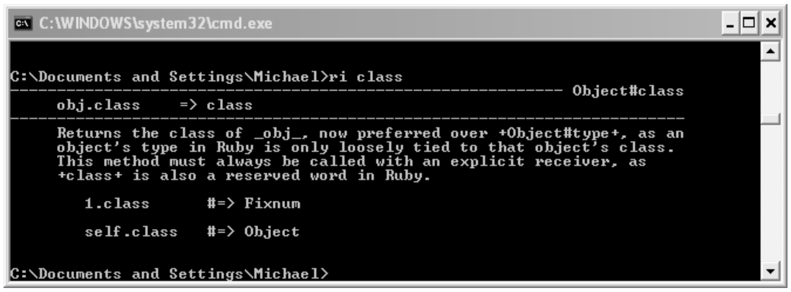
```
irb(main):001:0> class MyFirstRubyProgram
irb(main):002:1>   def SayHello(name)
irb(main):003:2>     puts "Hello, #{name}!"
irb(main):004:2>   end
irb(main):005:1> end
=> nil
irb(main):006:0>
irb(main):007:0* MyFirstRubyProgram.new.SayHello("OOP 2005")
Hello, OOP 2005!
=> nil
irb(main):008:0> _
```



```
C:\ruby>ruby myfirst.rb
Hello, OOP 2005!
```

A Small Example (3)

- **Using ri to obtain help:**



```
C:\Documents and Settings\Michael>ri class
---------------------------------------------------------- Object#class
     obj.class    => class
-----------------------------------------------------------------------
     Returns the class of _obj_, now preferred over +Object#type+, as an
     object's type in Ruby is only loosely tied to that object's class.
     This method must always be called with an explicit receiver, as
     +class+ is also a reserved word in Ruby.

        1.class      #=> Fixnum

        self.class   #=> Object

C:\Documents and Settings\Michael>
```

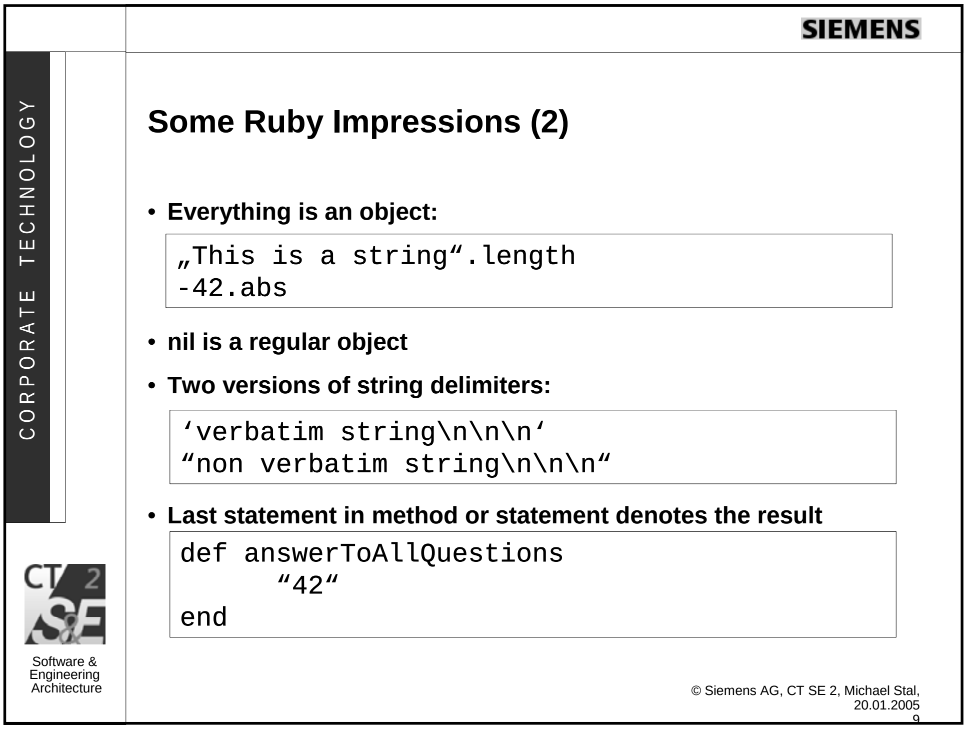
Some Ruby Impressions (1)

- **Ruby is untyped. Special syntax used to define variable scope:**

```
a = 42       # local variable w.r.t. scope
$b = 42      # global variable
@c = 42      # instance variable
@@d = 42     # class variable
PI = 3.14159 # constant
```

- **Different method call notations possible (beware of precedence)**

```
def max(a, b)
      (a > b) ? a : b
end
max(5,6) # call using brackets
max 5,6  # possible but deprecated
```



SIEMENS

CORPORATE TECHNOLOGY

Some Ruby Impressions (2)

- **Everything is an object:**

```
„This is a string“.length
-42.abs
```

- **nil is a regular object**
- **Two versions of string delimiters:**

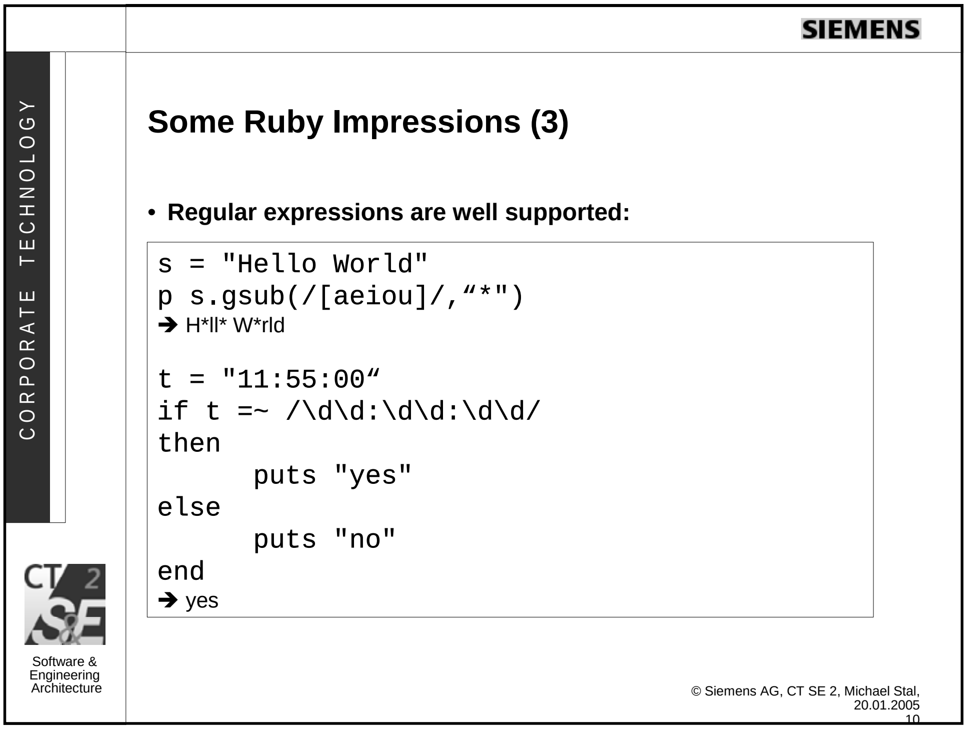
```
‘verbatim string\n\n\n‘
“non verbatim string\n\n\n“
```

- **Last statement in method or statement denotes the result**

```
def answerToAllQuestions
      “42“
end
```

Software & Engineering Architecture

© Siemens AG, CT SE 2, Michael Stal, 20.01.2005 9



SIEMENS

CORPORATE TECHNOLOGY

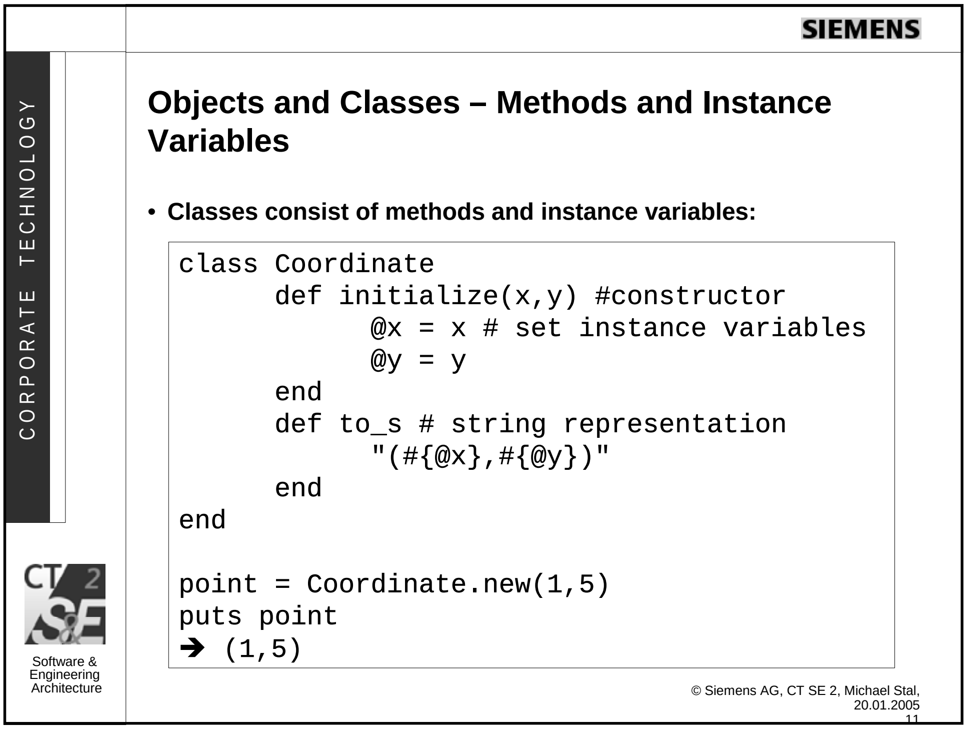
Some Ruby Impressions (3)

- **Regular expressions are well supported:**

```
s = "Hello World"
p s.gsub(/[aeiou]/,“*")
➔ H*ll* W*rld

t = "11:55:00“
if t =~ /\d\d:\d\d:\d\d/
then
      puts "yes"
else
      puts "no"
end
➔ yes
```

Software & Engineering Architecture

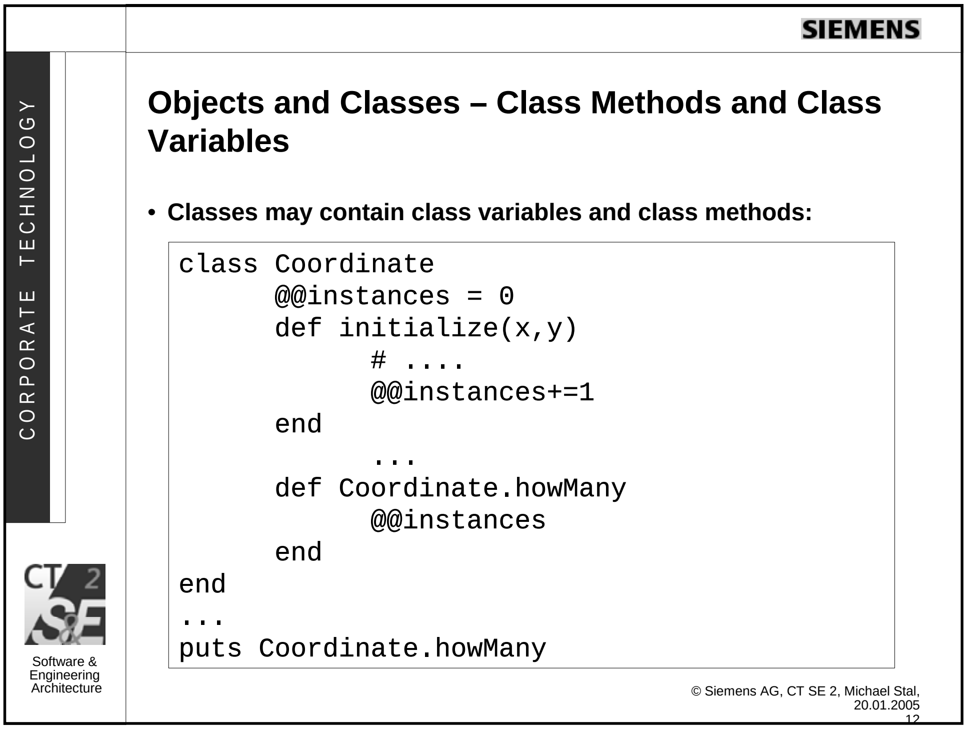
© Siemens AG, CT SE 2, Michael Stal, 20.01.2005 10



Objects and Classes – Methods and Instance Variables

- **Classes consist of methods and instance variables:**

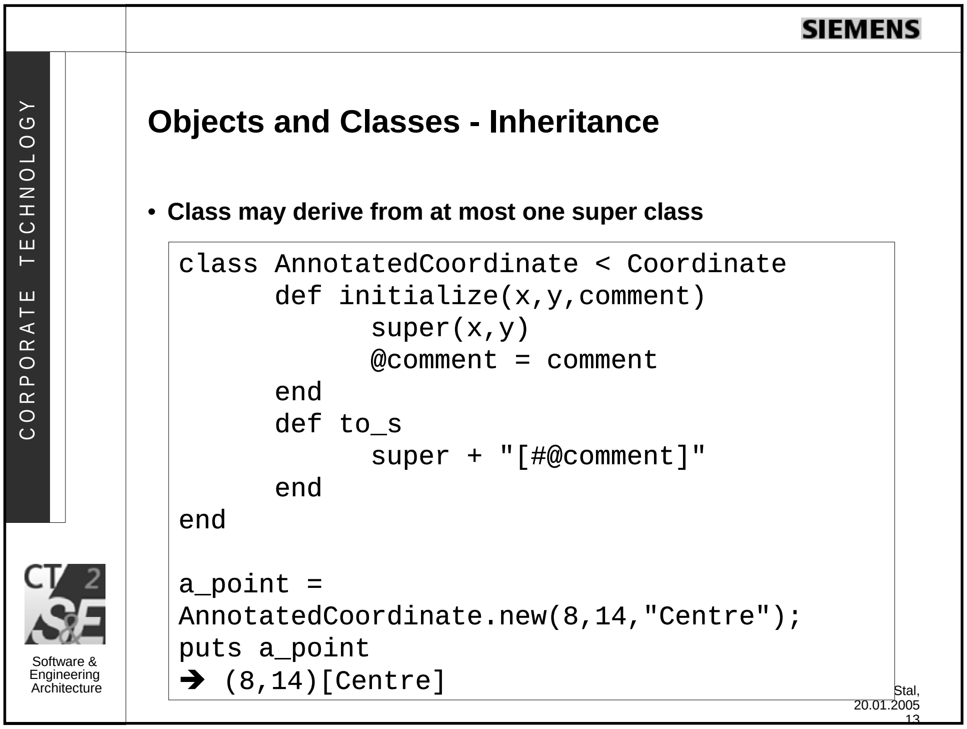
```
class Coordinate
      def initialize(x,y) #constructor
            @x = x # set instance variables
            @y = y
      end
      def to_s # string representation
            "(#{@x},#{@y})"
      end
end

point = Coordinate.new(1,5)
puts point
➔ (1,5)
```



Objects and Classes – Class Methods and Class Variables

- **Classes may contain class variables and class methods:**

```
class Coordinate
      @@instances = 0
      def initialize(x,y)
            # ....
            @@instances+=1
      end
            ...
      def Coordinate.howMany
            @@instances
      end
end
...
puts Coordinate.howMany
```



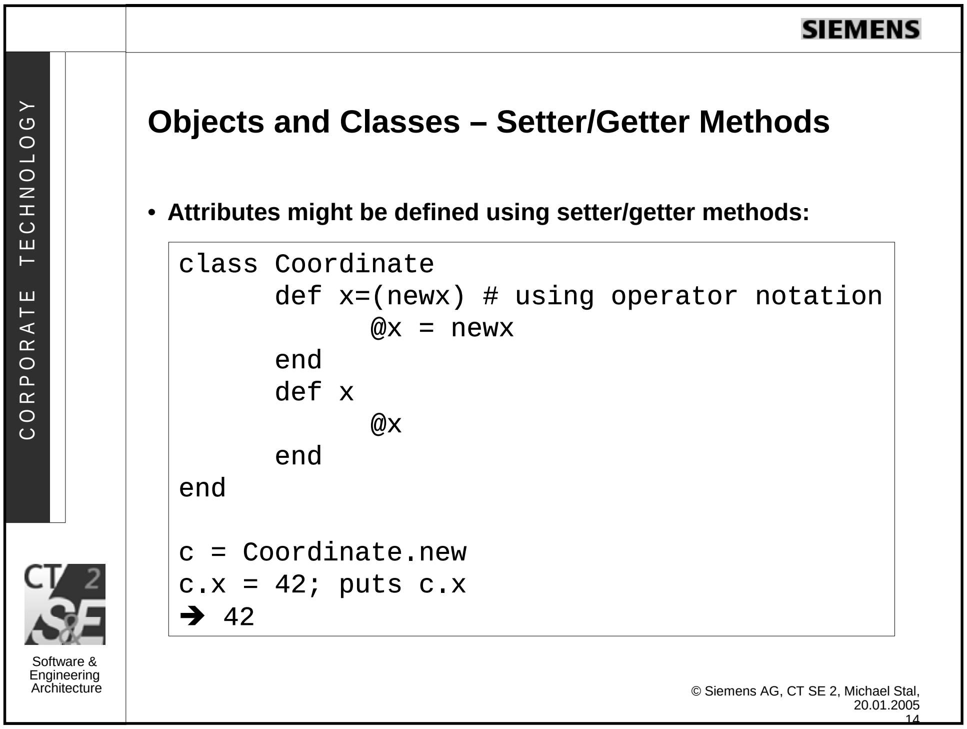
CORPORATE TECHNOLOGY

Objects and Classes - Inheritance

- **Class may derive from at most one super class**

```
class AnnotatedCoordinate < Coordinate
      def initialize(x,y,comment)
            super(x,y)
            @comment = comment
      end
      def to_s
            super + "[#@comment]"
      end
end

a_point =
AnnotatedCoordinate.new(8,14,"Centre");
puts a_point
➔ (8,14)[Centre]
```

Software & Engineering Architecture



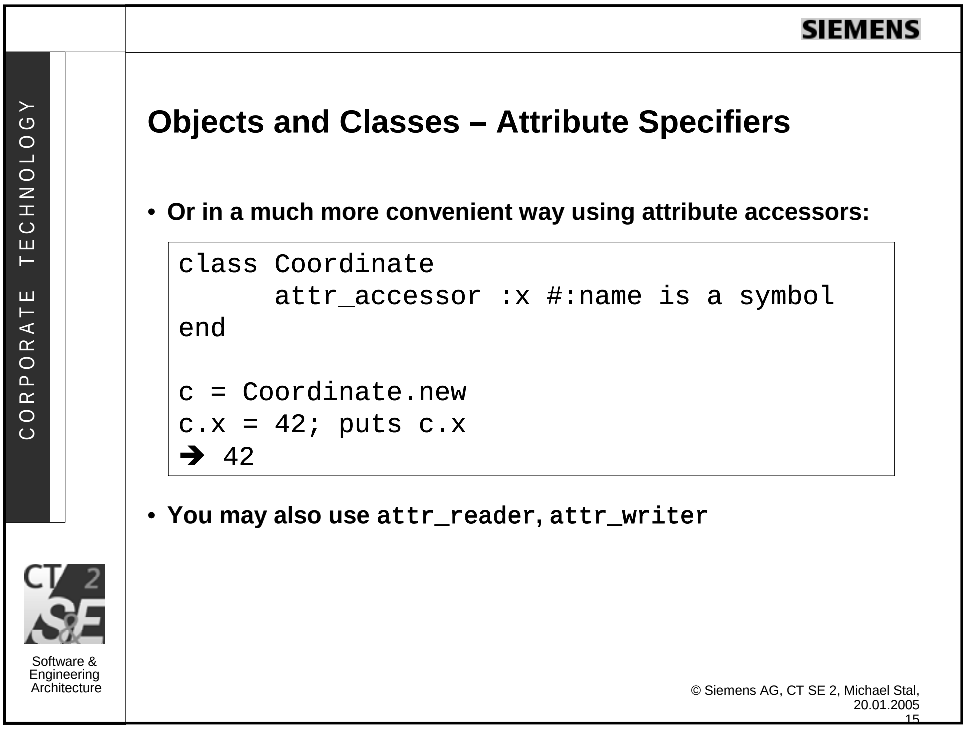
CORPORATE TECHNOLOGY

Objects and Classes – Setter/Getter Methods

- **Attributes might be defined using setter/getter methods:**

```
class Coordinate
      def x=(newx) # using operator notation
            @x = newx
      end
      def x
            @x
      end
end

c = Coordinate.new
c.x = 42; puts c.x
➔ 42
```

Software & Engineering Architecture

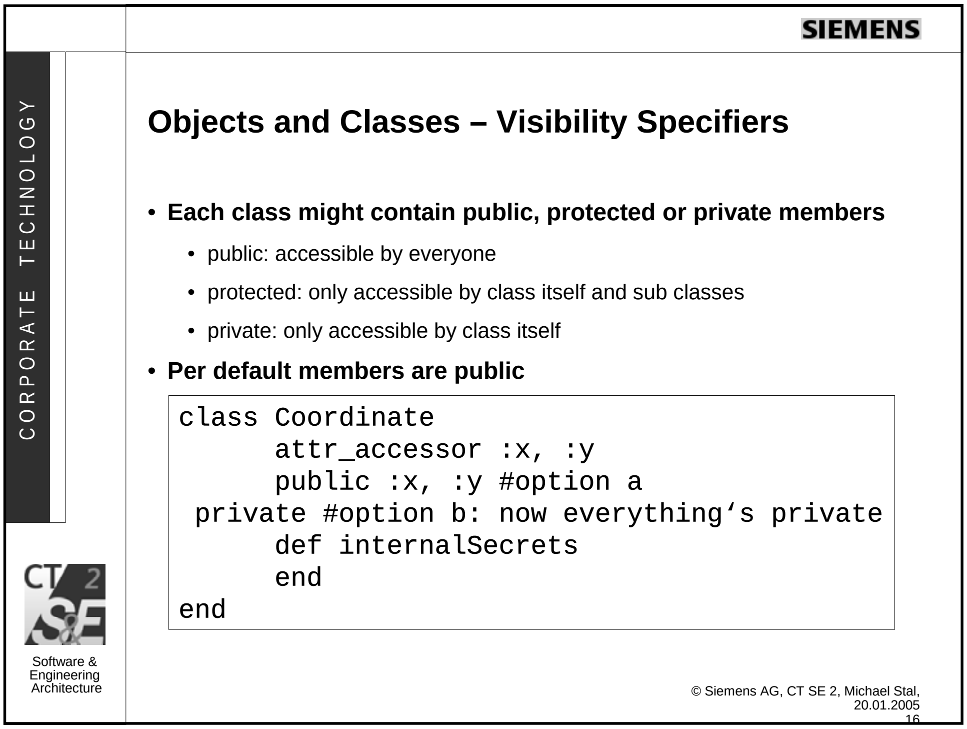


Objects and Classes – Attribute Specifiers

- **Or in a much more convenient way using attribute accessors:**

```
class Coordinate
      attr_accessor :x #:name is a symbol
end

c = Coordinate.new
c.x = 42; puts c.x
➔ 42
```

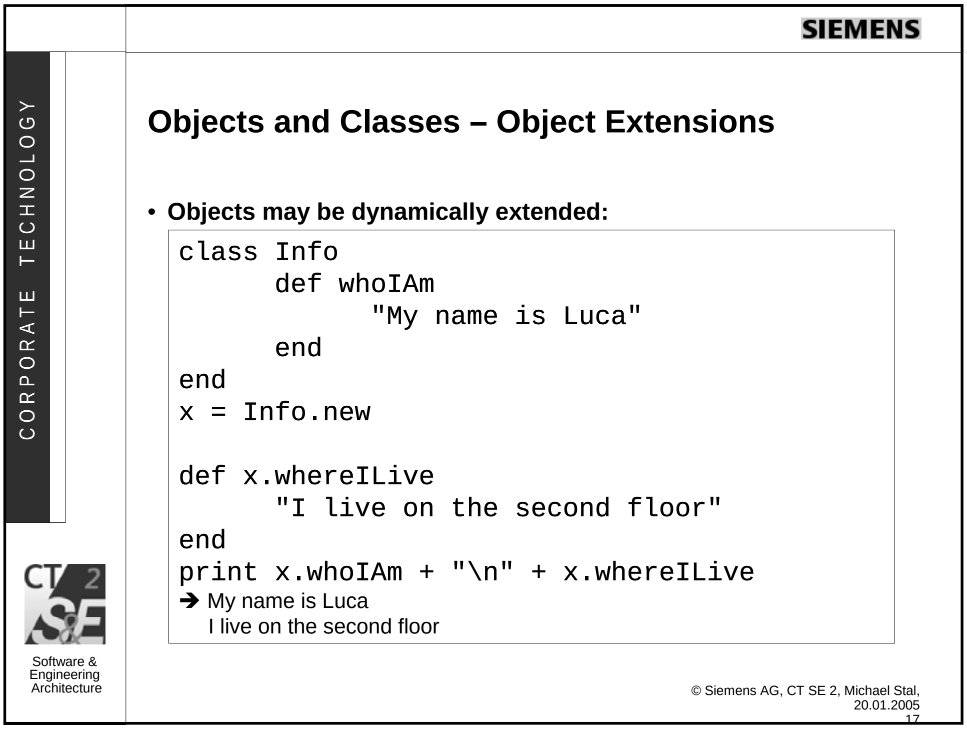
- **You may also use `attr_reader`, `attr_writer`**



Objects and Classes – Visibility Specifiers

- **Each class might contain public, protected or private members**
  - public: accessible by everyone
  - protected: only accessible by class itself and sub classes
  - private: only accessible by class itself
- **Per default members are public**

```
class Coordinate
      attr_accessor :x, :y
      public :x, :y #option a
 private #option b: now everything‘s private
      def internalSecrets
      end
end
```



SIEMENS

CORPORATE TECHNOLOGY

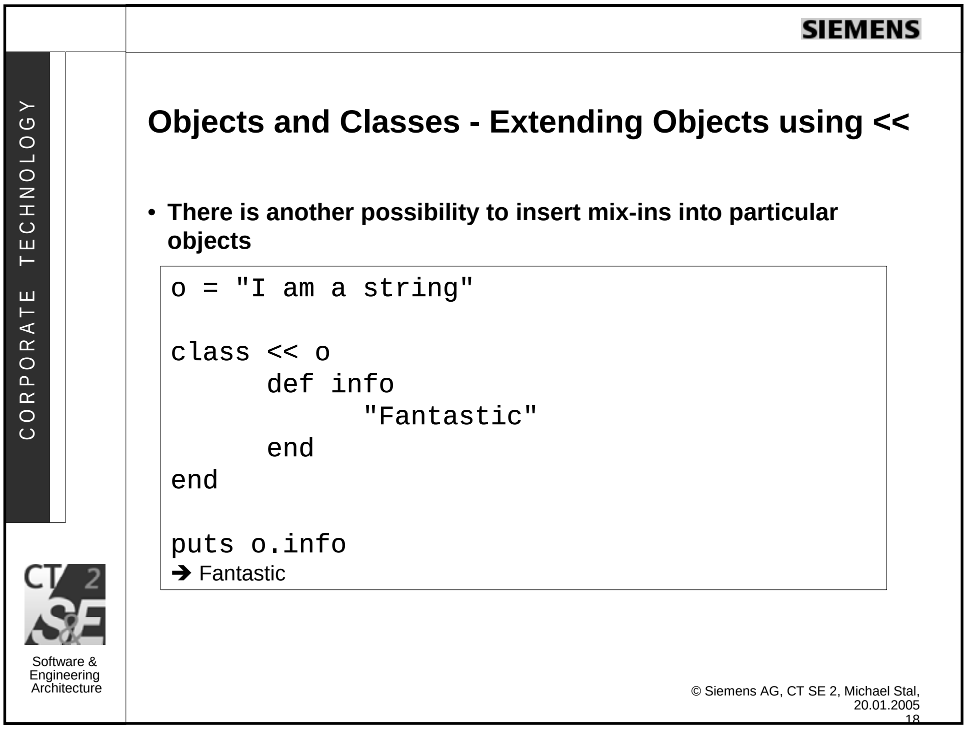
Objects and Classes – Object Extensions

- **Objects may be dynamically extended:**

```
class Info
      def whoIAm
            "My name is Luca"
      end
end
x = Info.new

def x.whereILive
      "I live on the second floor"
end
print x.whoIAm + "\n" + x.whereILive
```

➔ My name is Luca
I live on the second floor

Software &
Engineering
Architecture



SIEMENS

CORPORATE TECHNOLOGY

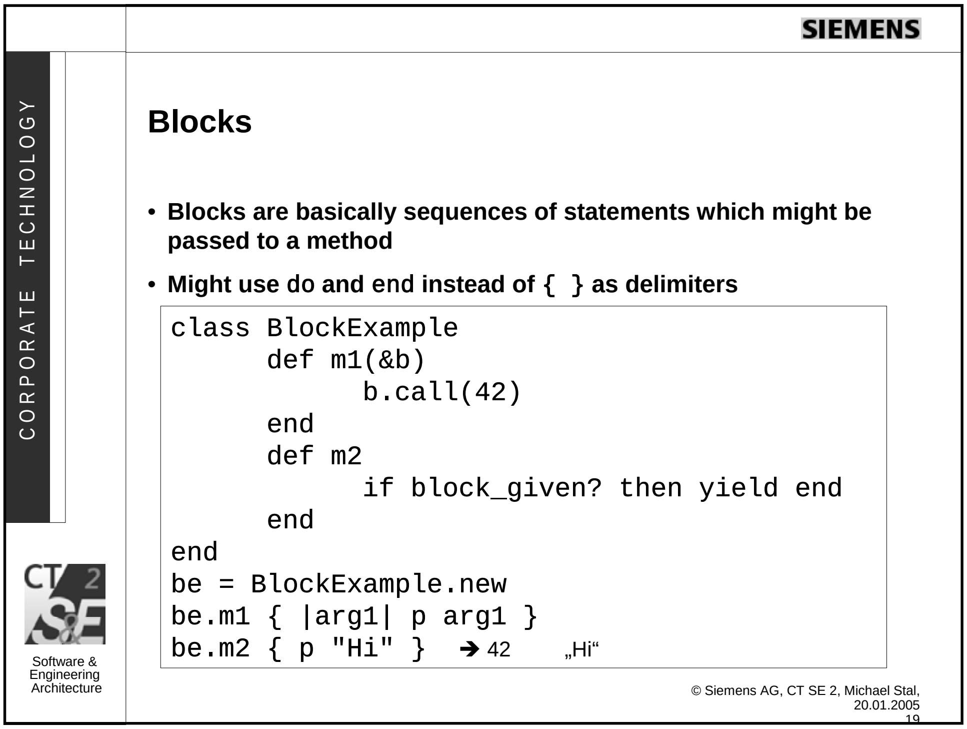
Objects and Classes - Extending Objects using <<

- **There is another possibility to insert mix-ins into particular objects**

```
o = "I am a string"

class << o
      def info
            "Fantastic"
      end
end

puts o.info
```

➔ Fantastic

Software &
Engineering
Architecture



SIEMENS

CORPORATE TECHNOLOGY

Blocks

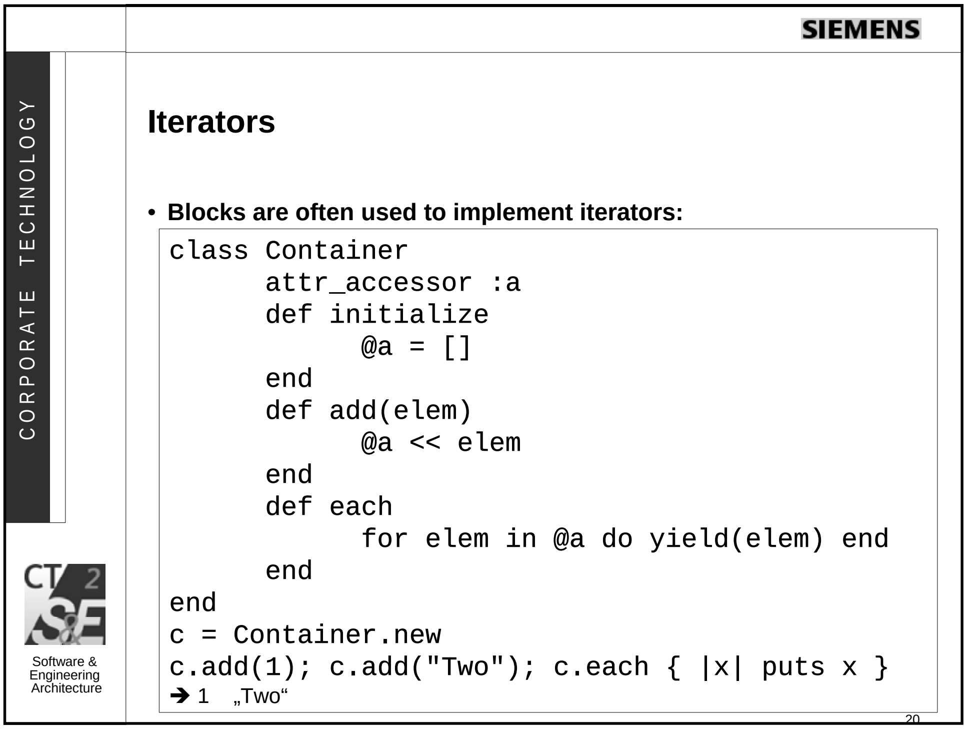
- **Blocks are basically sequences of statements which might be passed to a method**
- **Might use do and end instead of { } as delimiters**

```
class BlockExample
      def m1(&b)
            b.call(42)
      end
      def m2
            if block_given? then yield end
      end
end
be = BlockExample.new
be.m1 { |arg1| p arg1 }
be.m2 { p "Hi" }  ➔ 42     „Hi“
```

Software & Engineering Architecture

© Siemens AG, CT SE 2, Michael Stal, 20.01.2005

19



SIEMENS

CORPORATE TECHNOLOGY

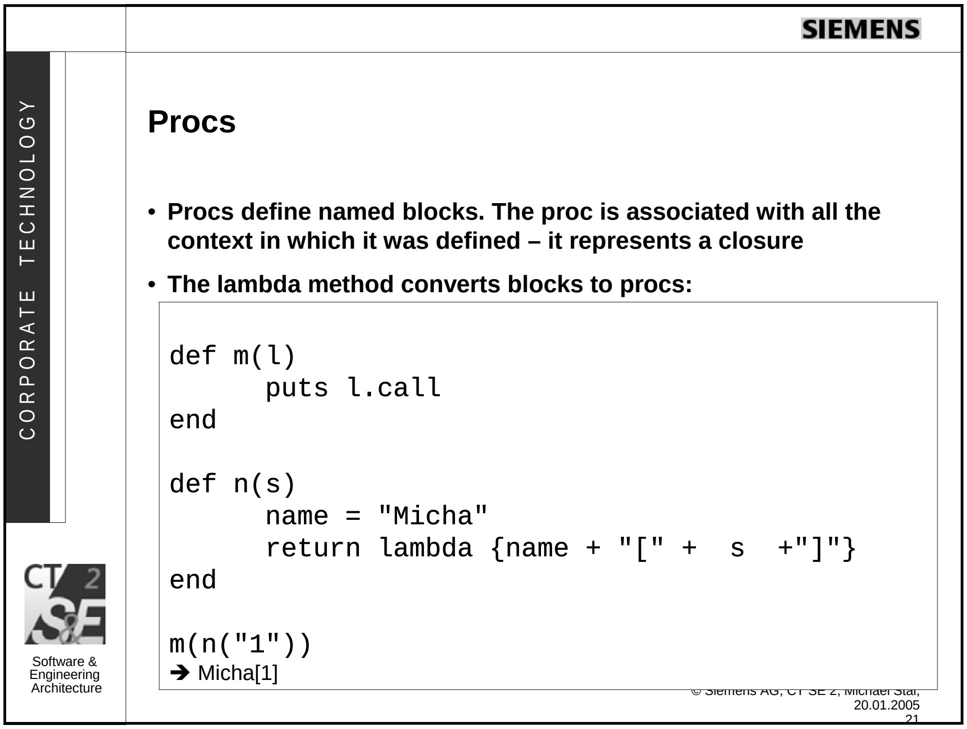
Iterators

- **Blocks are often used to implement iterators:**

```
class Container
      attr_accessor :a
      def initialize
            @a = []
      end
      def add(elem)
            @a << elem
      end
      def each
            for elem in @a do yield(elem) end
      end
end
c = Container.new
c.add(1); c.add("Two"); c.each { |x| puts x }
➔ 1   „Two“
```

Software & Engineering Architecture

20



CORPORATE TECHNOLOGY

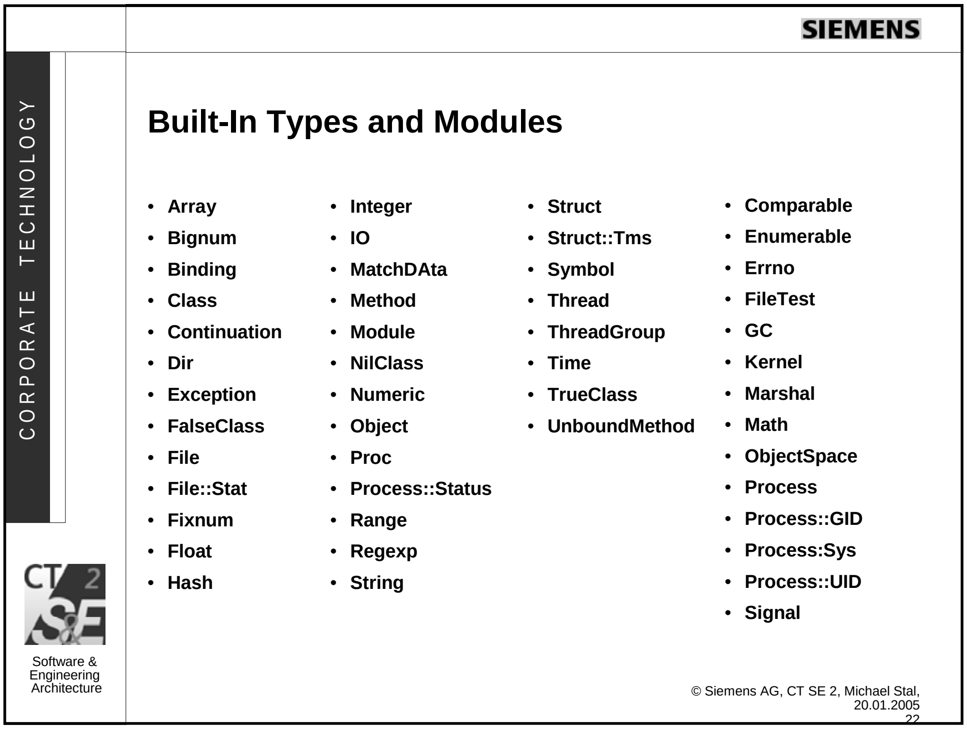
Procs

- Procs define named blocks. The proc is associated with all the context in which it was defined – it represents a closure
- The lambda method converts blocks to procs:

```
def m(l)
      puts l.call
end

def n(s)
      name = "Micha"
      return lambda {name + "[" +  s  +"]"}
end

m(n("1"))
```

➔ Micha[1]

Software & Engineering Architecture

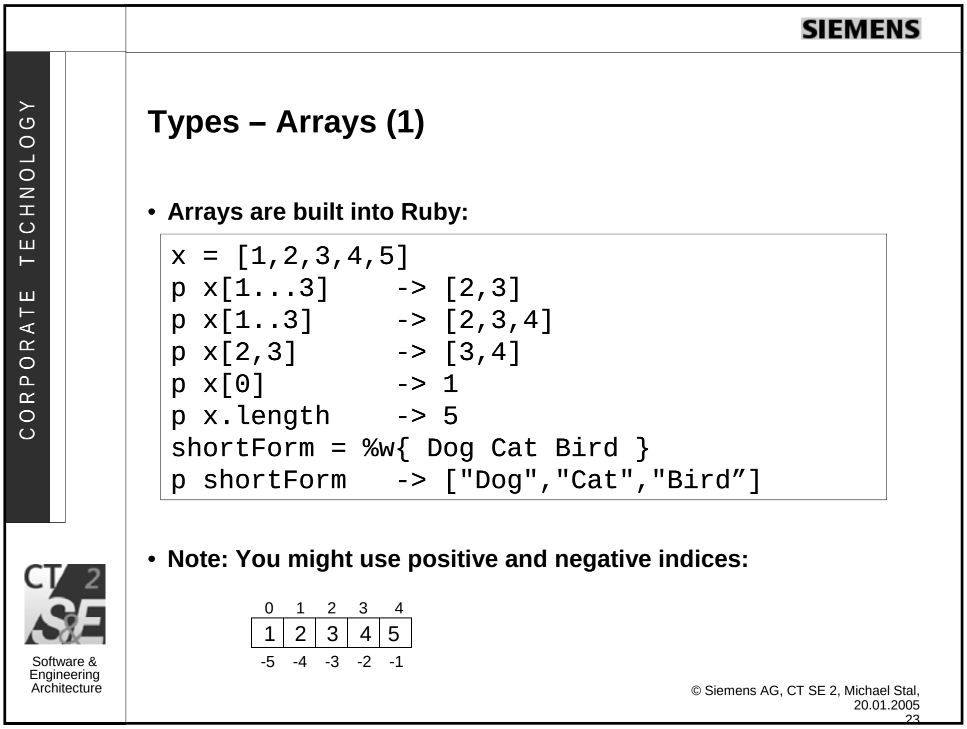


CORPORATE TECHNOLOGY

Built-In Types and Modules

- Array
- Bignum
- Binding
- Class
- Continuation
- Dir
- Exception
- FalseClass
- File
- File::Stat
- Fixnum
- Float
- Hash
- Integer
- IO
- MatchDAta
- Method
- Module
- NilClass
- Numeric
- Object
- Proc
- Process::Status
- Range
- Regexp
- String
- Struct
- Struct::Tms
- Symbol
- Thread
- ThreadGroup
- Time
- TrueClass
- UnboundMethod
- Comparable
- Enumerable
- Errno
- FileTest
- GC
- Kernel
- Marshal
- Math
- ObjectSpace
- Process
- Process::GID
- Process:Sys
- Process::UID
- Signal

Software & Engineering Architecture



CORPORATE TECHNOLOGY

Types – Arrays (1)

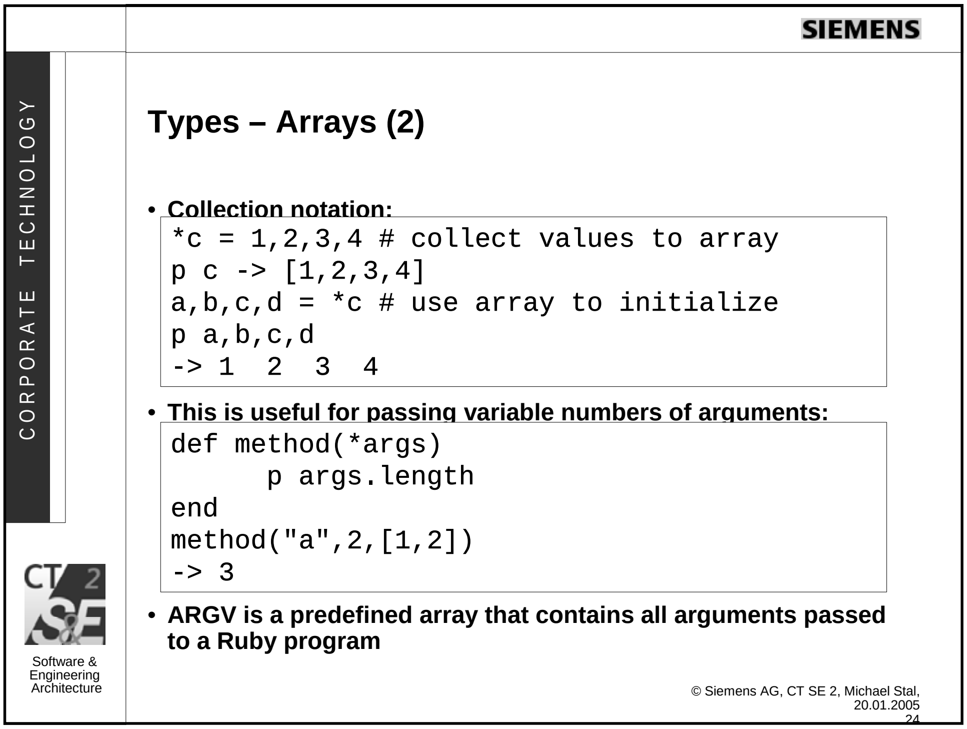
- **Arrays are built into Ruby:**

```
x = [1,2,3,4,5]
p x[1...3]    -> [2,3]
p x[1..3]     -> [2,3,4]
p x[2,3]      -> [3,4]
p x[0]        -> 1
p x.length    -> 5
shortForm = %w{ Dog Cat Bird }
p shortForm   -> ["Dog","Cat","Bird”]
```

- **Note: You might use positive and negative indices:**

0	1	2	3	4
1	2	3	4	5
-5	-4	-3	-2	-1

Software & Engineering Architecture



CORPORATE TECHNOLOGY

Types – Arrays (2)

- **Collection notation:**

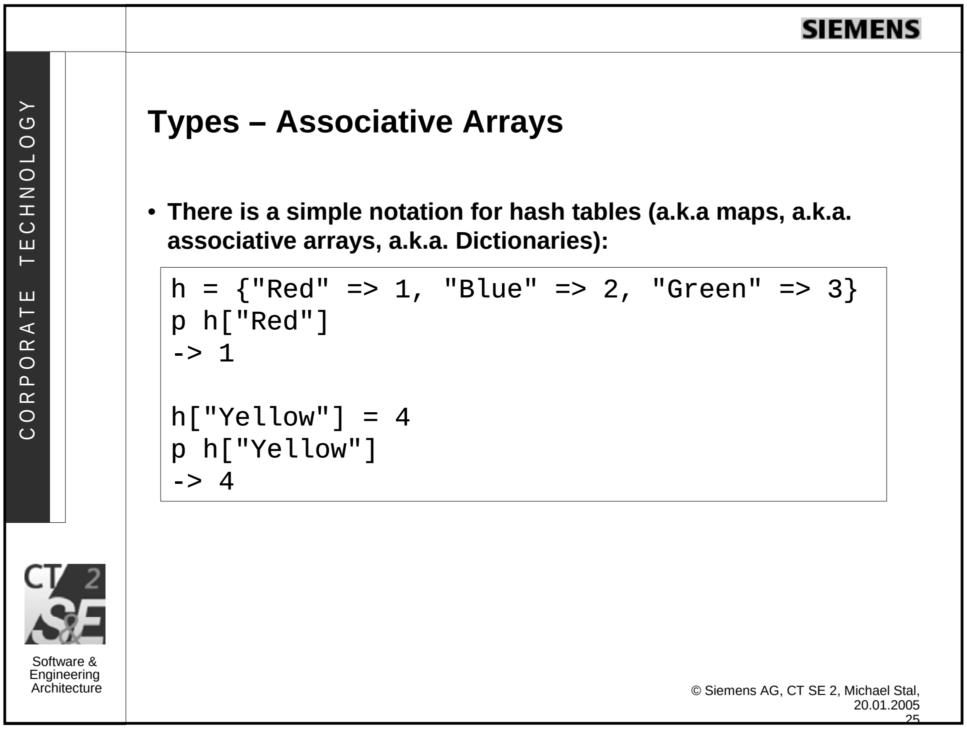
```
*c = 1,2,3,4 # collect values to array
p c -> [1,2,3,4]
a,b,c,d = *c # use array to initialize
p a,b,c,d
-> 1  2  3  4
```

- **This is useful for passing variable numbers of arguments:**

```
def method(*args)
      p args.length
end
method("a",2,[1,2])
-> 3
```

- **ARGV is a predefined array that contains all arguments passed to a Ruby program**

Software & Engineering Architecture



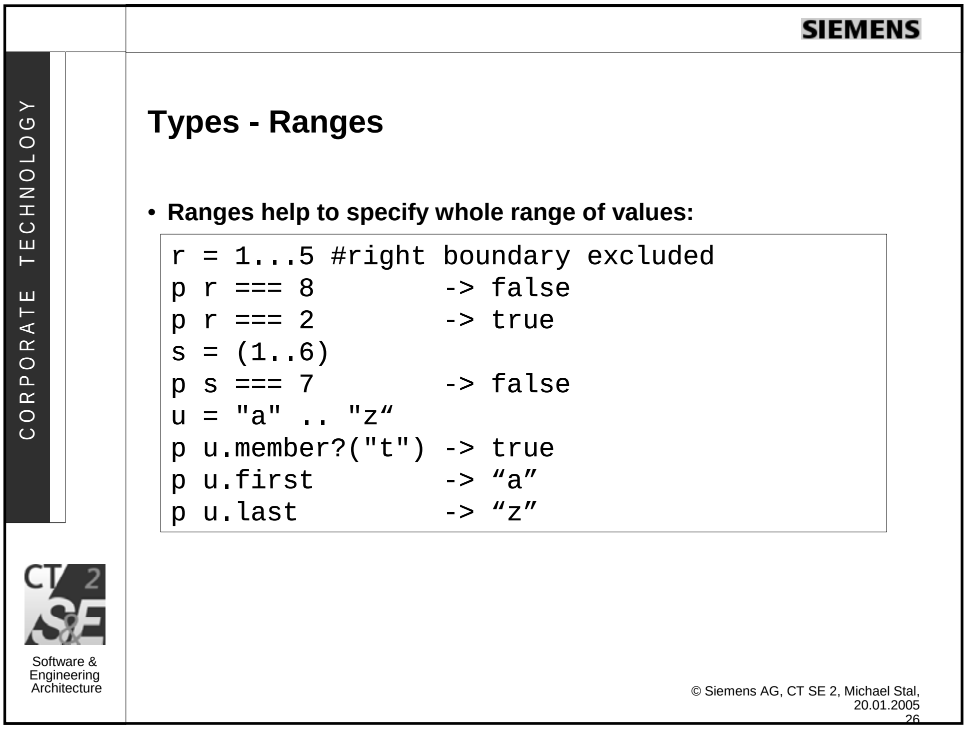
CORPORATE TECHNOLOGY

Types – Associative Arrays

- There is a simple notation for hash tables (a.k.a maps, a.k.a. associative arrays, a.k.a. Dictionaries):

```
h = {"Red" => 1, "Blue" => 2, "Green" => 3}
p h["Red"]
-> 1

h["Yellow"] = 4
p h["Yellow"]
-> 4
```

Software & Engineering Architecture



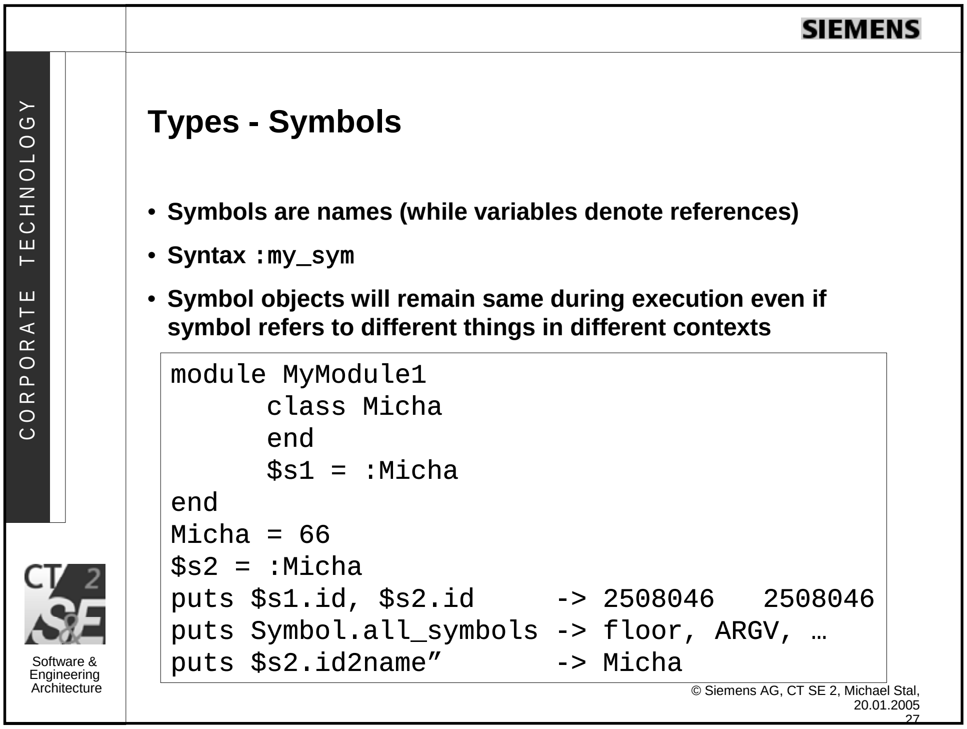
CORPORATE TECHNOLOGY

Types - Ranges

- Ranges help to specify whole range of values:

```
r = 1...5 #right boundary excluded
p r === 8        -> false
p r === 2        -> true
s = (1..6)
p s === 7        -> false
u = "a" .. "z“
p u.member?("t") -> true
p u.first        -> “a”
p u.last         -> “z”
```

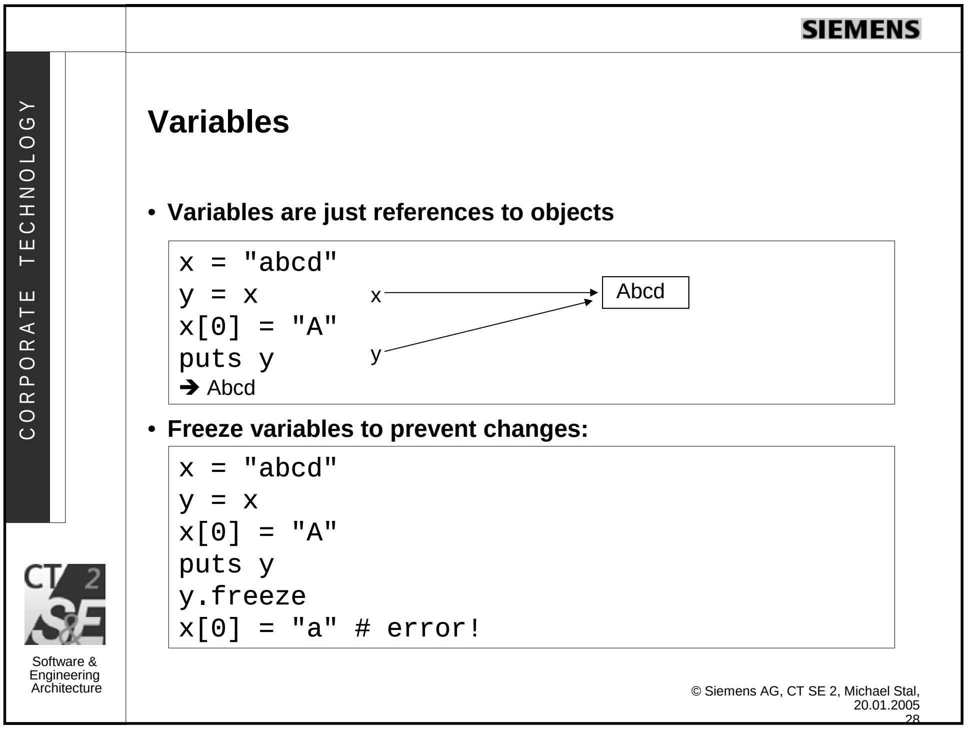
Software & Engineering Architecture



Types - Symbols

- Symbols are names (while variables denote references)
- Syntax `:my_sym`
- Symbol objects will remain same during execution even if symbol refers to different things in different contexts

```
module MyModule1
      class Micha
      end
      $s1 = :Micha
end
Micha = 66
$s2 = :Micha
puts $s1.id, $s2.id      -> 2508046   2508046
puts Symbol.all_symbols -> floor, ARGV, …
puts $s2.id2name”       -> Micha
```



Variables

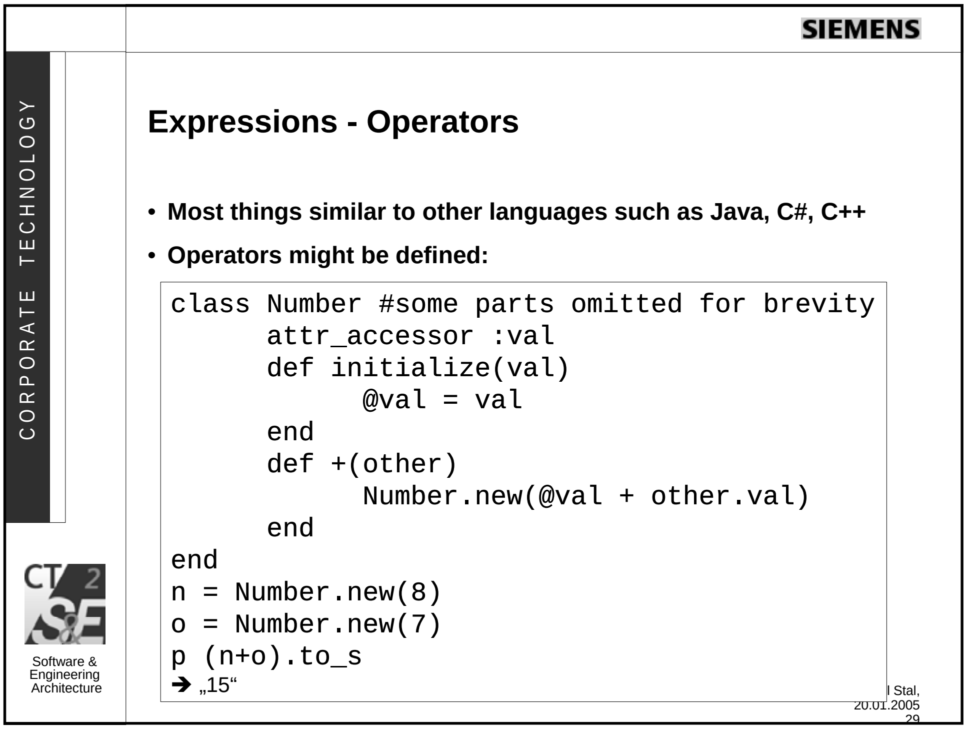
- Variables are just references to objects

```
x = "abcd"
y = x
x[0] = "A"
puts y
```

➔ Abcd

- Freeze variables to prevent changes:

```
x = "abcd"
y = x
x[0] = "A"
puts y
y.freeze
x[0] = "a" # error!
```



CORPORATE TECHNOLOGY

Expressions - Operators

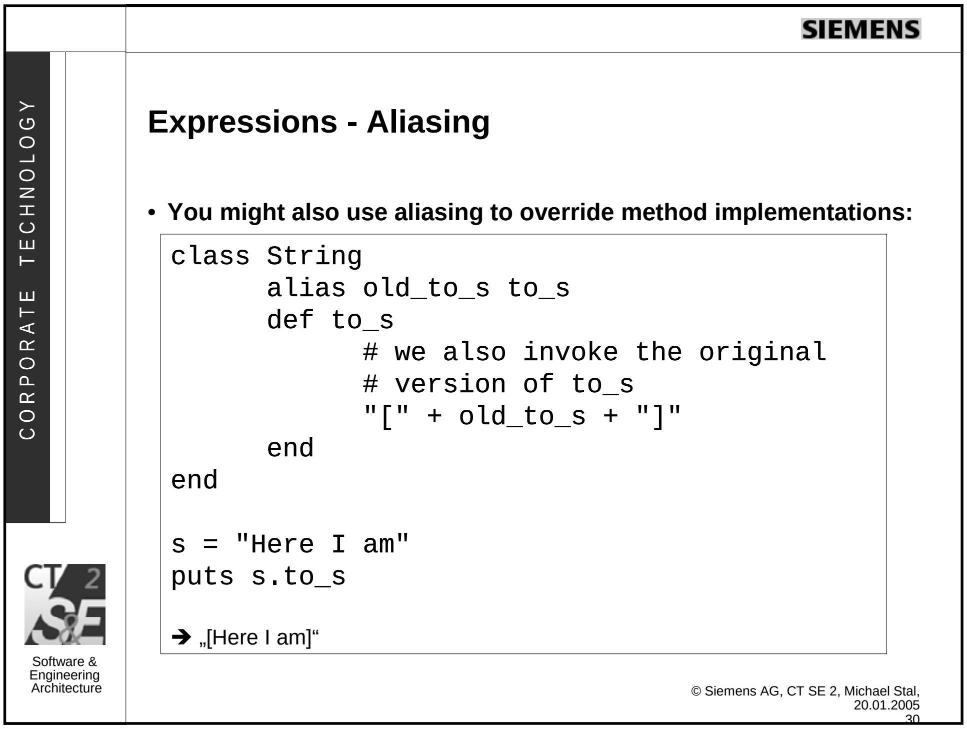
- Most things similar to other languages such as Java, C#, C++
- Operators might be defined:

```
class Number #some parts omitted for brevity
      attr_accessor :val
      def initialize(val)
            @val = val
      end
      def +(other)
            Number.new(@val + other.val)
      end
end
n = Number.new(8)
o = Number.new(7)
p (n+o).to_s
```

➔ „15“

Software & Engineering Architecture

l Stal,
20.01.2005
29



CORPORATE TECHNOLOGY

Expressions - Aliasing

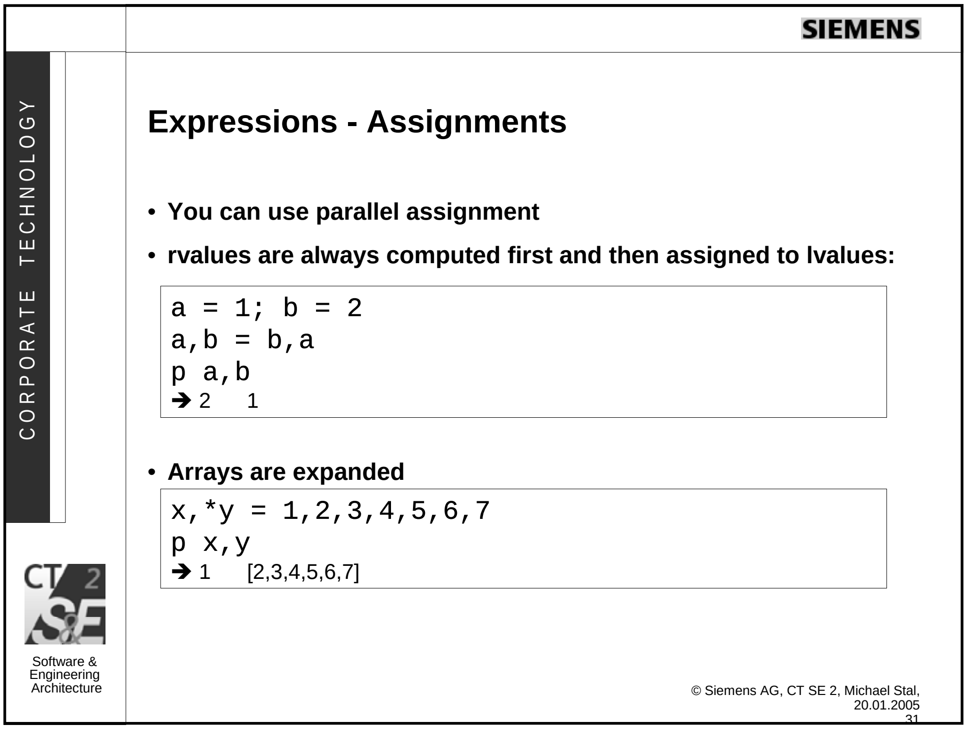
- You might also use aliasing to override method implementations:

```
class String
      alias old_to_s to_s
      def to_s
            # we also invoke the original
            # version of to_s
            "[" + old_to_s + "]"
      end
end

s = "Here I am"
puts s.to_s
```

➔ „[Here I am]“

Software & Engineering Architecture

© Siemens AG, CT SE 2, Michael Stal,
20.01.2005
30



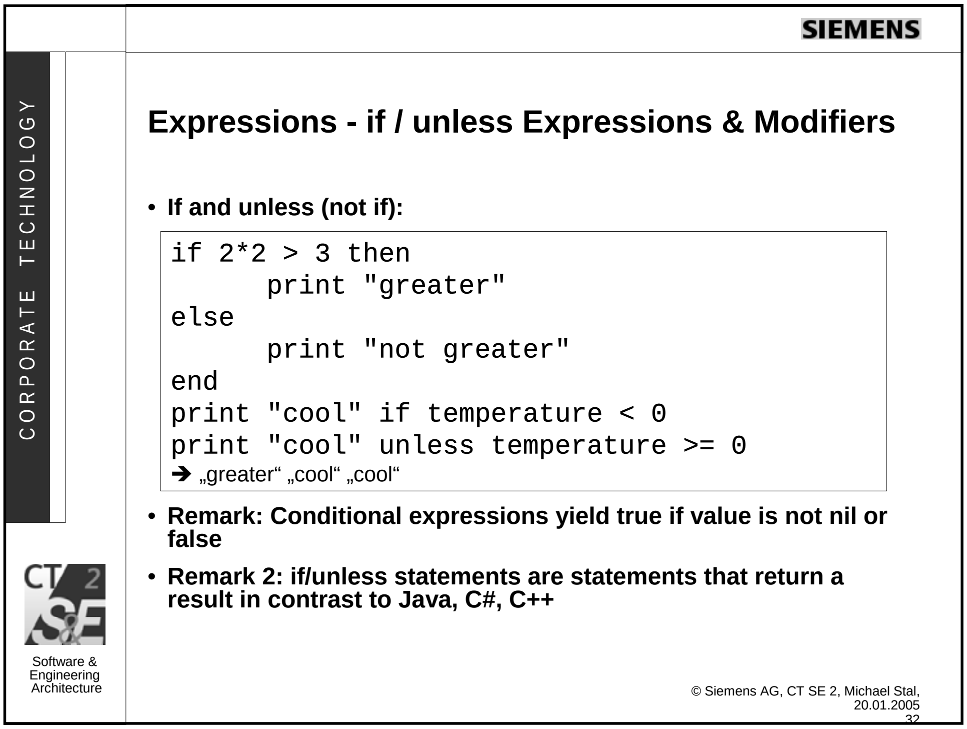
Expressions - Assignments

- **You can use parallel assignment**
- **rvalues are always computed first and then assigned to lvalues:**

```
a = 1; b = 2
a,b = b,a
p a,b
➔ 2   1
```

- **Arrays are expanded**

```
x,*y = 1,2,3,4,5,6,7
p x,y
➔ 1     [2,3,4,5,6,7]
```

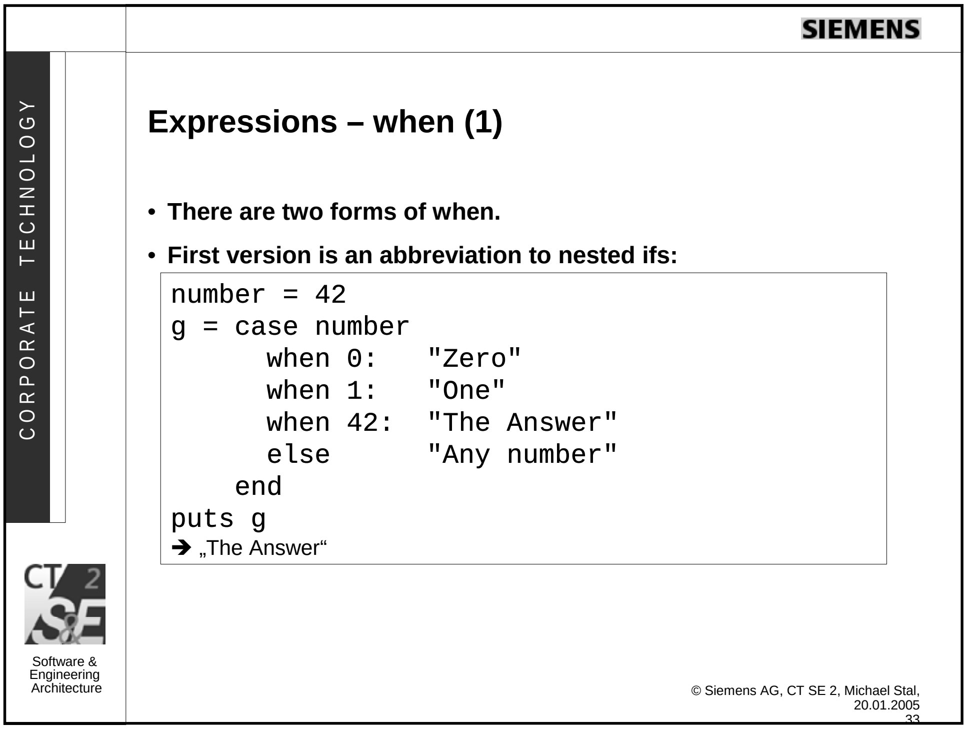


Expressions - if / unless Expressions & Modifiers

- **If and unless (not if):**

```
if 2*2 > 3 then
      print "greater"
else
      print "not greater"
end
print "cool" if temperature < 0
print "cool" unless temperature >= 0
➔ „greater“ „cool“ „cool“
```

- **Remark: Conditional expressions yield true if value is not nil or false**
- **Remark 2: if/unless statements are statements that return a result in contrast to Java, C#, C++**



SIEMENS

CORPORATE TECHNOLOGY

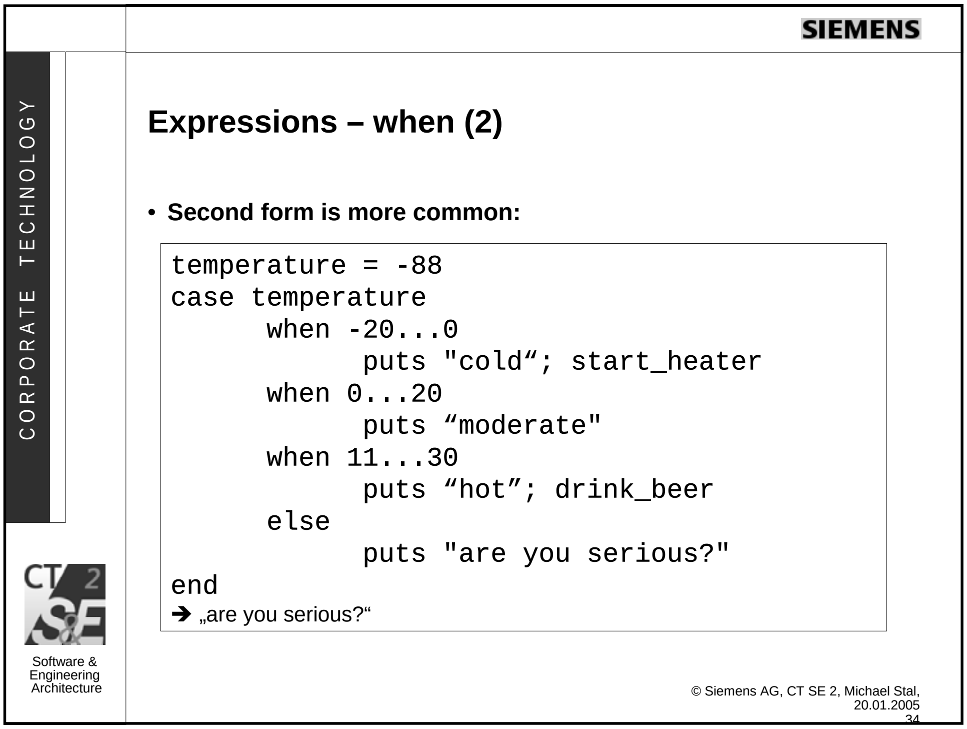
Expressions – when (1)

- There are two forms of when.
- First version is an abbreviation to nested ifs:

```
number = 42
g = case number
      when 0:   "Zero"
      when 1:   "One"
      when 42:  "The Answer"
      else      "Any number"
    end
puts g
```

➔ „The Answer“

Software & Engineering Architecture



SIEMENS

CORPORATE TECHNOLOGY

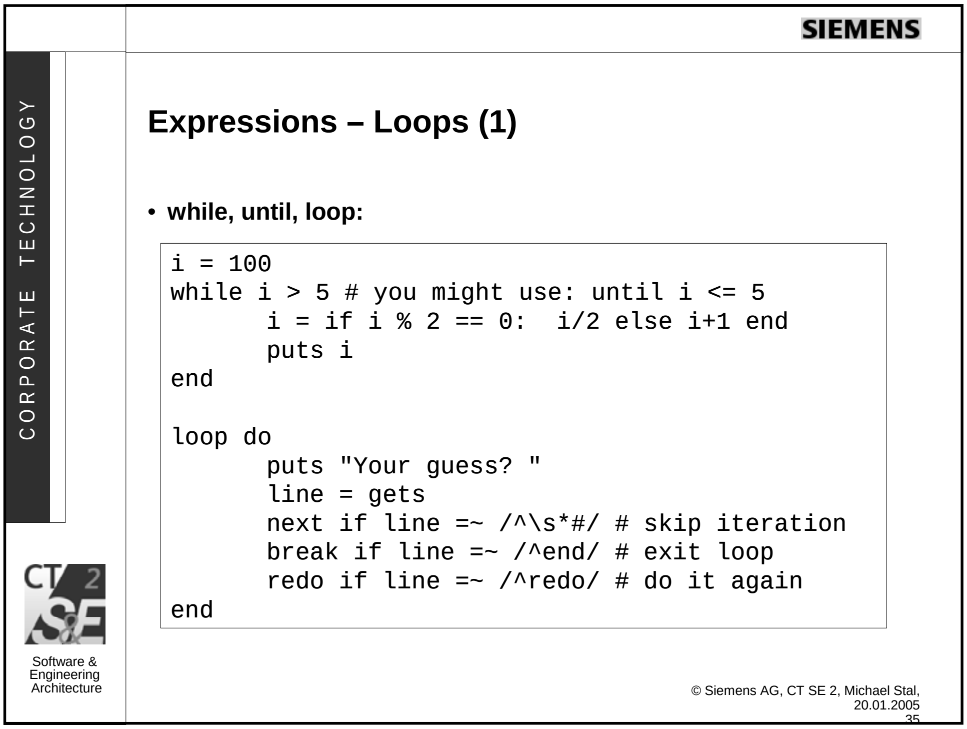
Expressions – when (2)

- Second form is more common:

```
temperature = -88
case temperature
      when -20...0
            puts "cold“; start_heater
      when 0...20
            puts “moderate"
      when 11...30
            puts “hot”; drink_beer
      else
            puts "are you serious?"
end
```

➔ „are you serious?“

Software & Engineering Architecture



SIEMENS

CORPORATE TECHNOLOGY

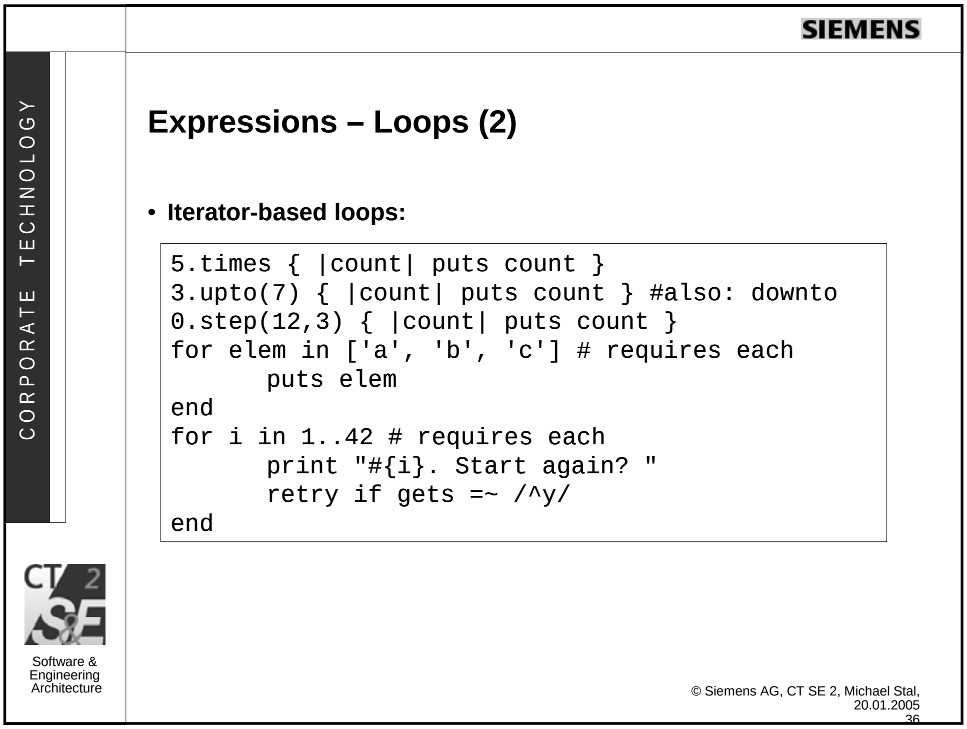
Expressions – Loops (1)

- **while, until, loop:**

```
i = 100
while i > 5 # you might use: until i <= 5
       i = if i % 2 == 0:  i/2 else i+1 end
       puts i
end

loop do
       puts "Your guess? "
       line = gets
       next if line =~ /^\s*#/ # skip iteration
       break if line =~ /^end/ # exit loop
       redo if line =~ /^redo/ # do it again
end
```

Software & Engineering Architecture

© Siemens AG, CT SE 2, Michael Stal, 20.01.2005
35



SIEMENS

CORPORATE TECHNOLOGY

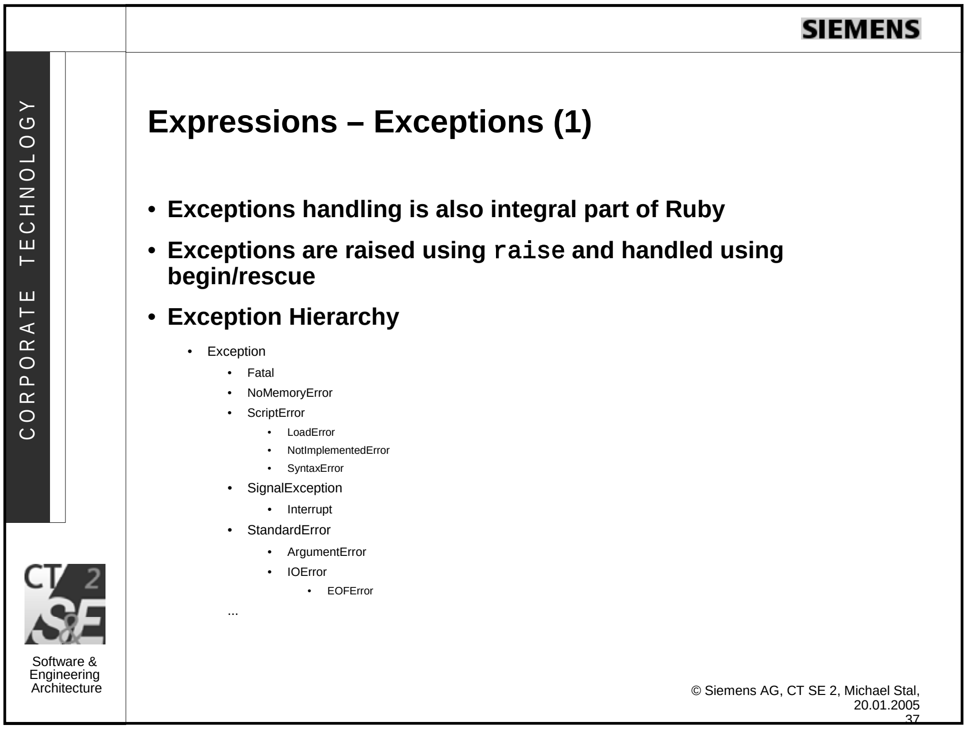
Expressions – Loops (2)

- **Iterator-based loops:**

```
5.times { |count| puts count }
3.upto(7) { |count| puts count } #also: downto
0.step(12,3) { |count| puts count }
for elem in ['a', 'b', 'c'] # requires each
       puts elem
end
for i in 1..42 # requires each
       print "#{i}. Start again? "
       retry if gets =~ /^y/
end
```

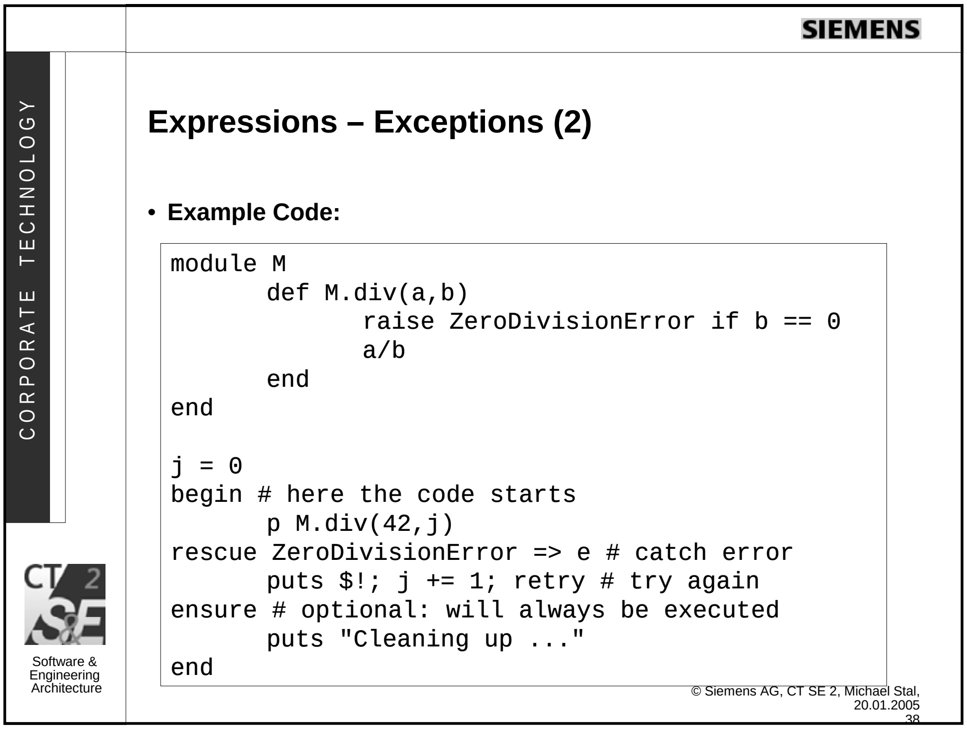
Software & Engineering Architecture

© Siemens AG, CT SE 2, Michael Stal, 20.01.2005
36



Expressions – Exceptions (1)

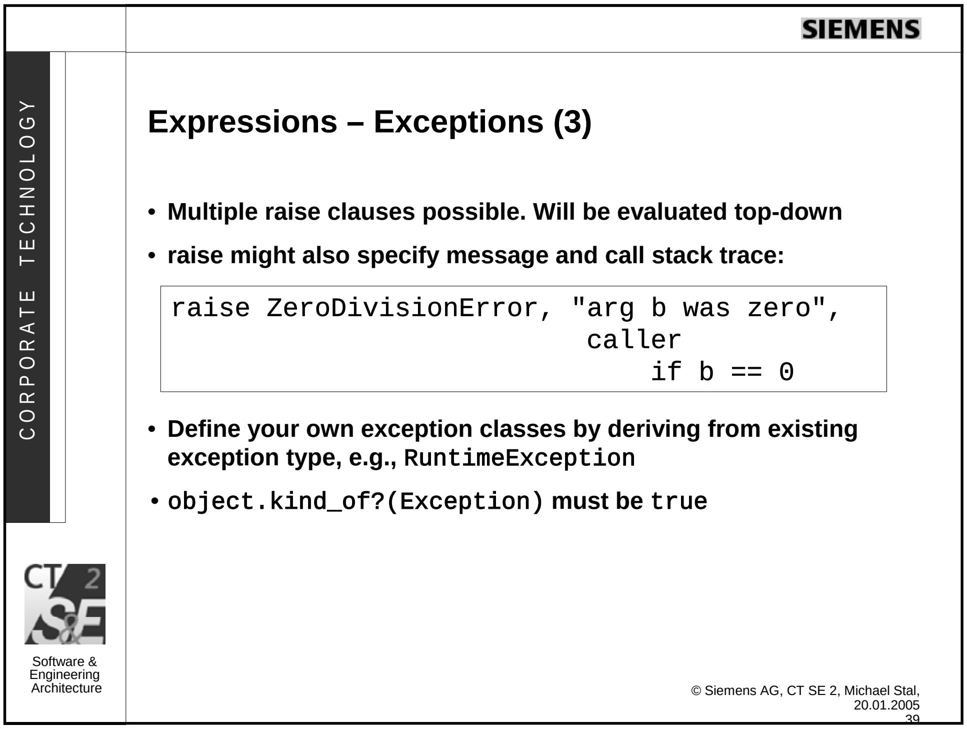
- **Exceptions handling is also integral part of Ruby**
- **Exceptions are raised using `raise` and handled using begin/rescue**
- **Exception Hierarchy**
  - Exception
    - Fatal
    - NoMemoryError
    - ScriptError
      - LoadError
      - NotImplementedError
      - SyntaxError
    - SignalException
      - Interrupt
    - StandardError
      - ArgumentError
      - IOError
        - EOFError
    - ...



Expressions – Exceptions (2)

- **Example Code:**

```
module M
       def M.div(a,b)
              raise ZeroDivisionError if b == 0
              a/b
       end
end

j = 0
begin # here the code starts
       p M.div(42,j)
rescue ZeroDivisionError => e # catch error
       puts $!; j += 1; retry # try again
ensure # optional: will always be executed
       puts "Cleaning up ..."
end
```



SIEMENS

CORPORATE TECHNOLOGY

Expressions – Exceptions (3)

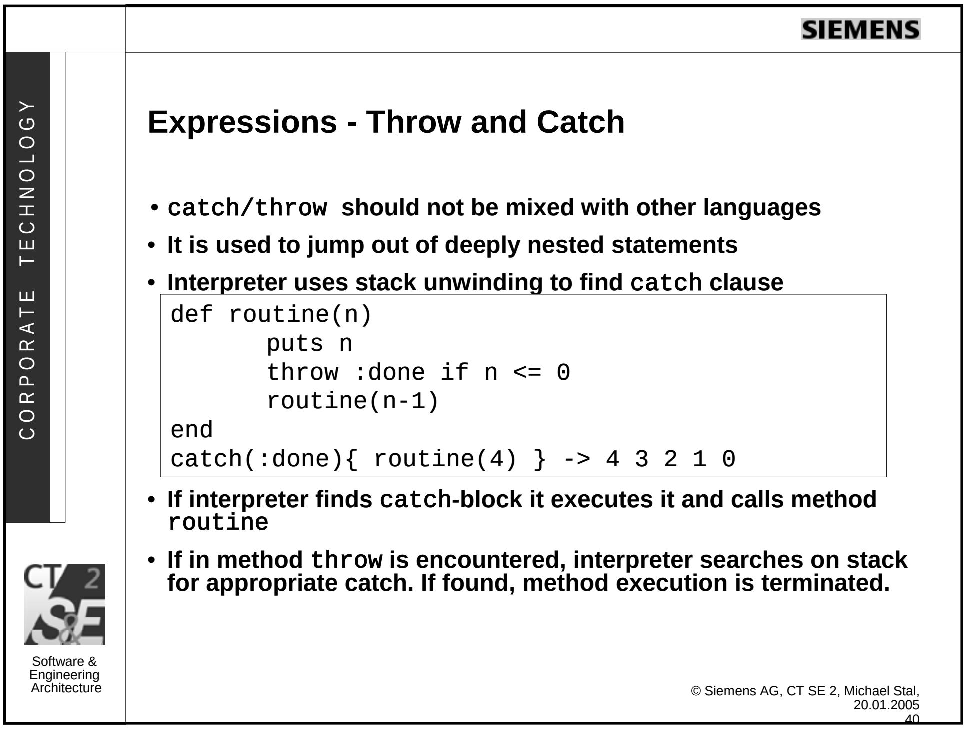
- Multiple raise clauses possible. Will be evaluated top-down
- raise might also specify message and call stack trace:

```
raise ZeroDivisionError, "arg b was zero",
                          caller
                              if b == 0
```

- Define your own exception classes by deriving from existing exception type, e.g., `RuntimeException`
- `object.kind_of?(Exception)` must be `true`

Software & Engineering Architecture

© Siemens AG, CT SE 2, Michael Stal, 20.01.2005 39



SIEMENS

CORPORATE TECHNOLOGY

Expressions - Throw and Catch

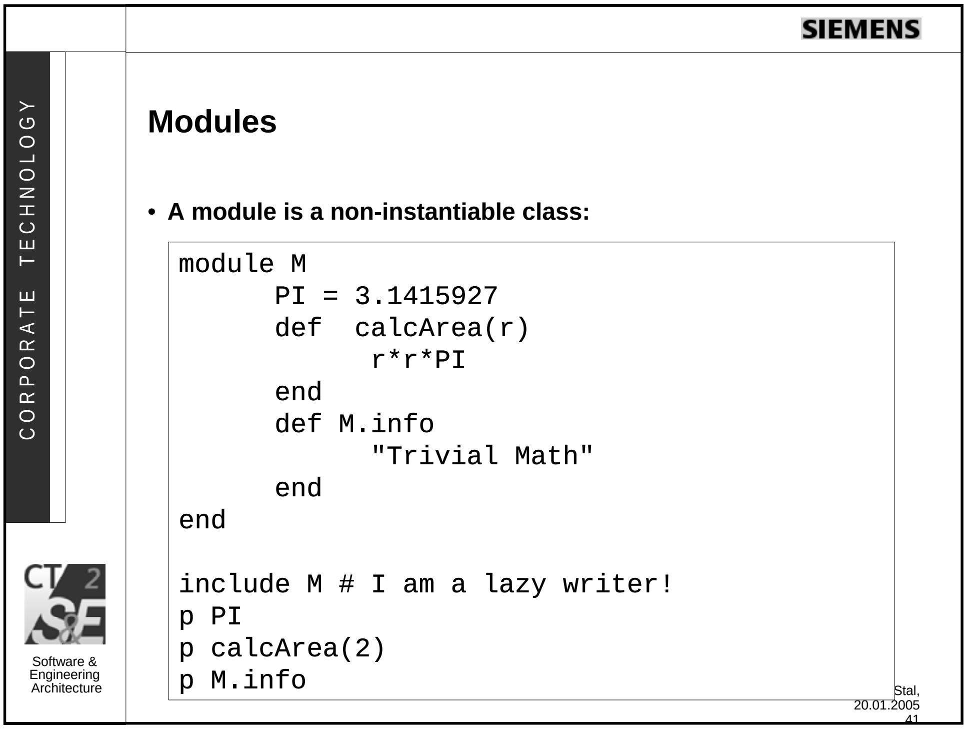
- `catch/throw` should not be mixed with other languages
- It is used to jump out of deeply nested statements
- Interpreter uses stack unwinding to find `catch` clause

```
def routine(n)
      puts n
      throw :done if n <= 0
      routine(n-1)
end
catch(:done){ routine(4) } -> 4 3 2 1 0
```

- If interpreter finds `catch`-block it executes it and calls method `routine`
- If in method `throw` is encountered, interpreter searches on stack for appropriate catch. If found, method execution is terminated.

Software & Engineering Architecture

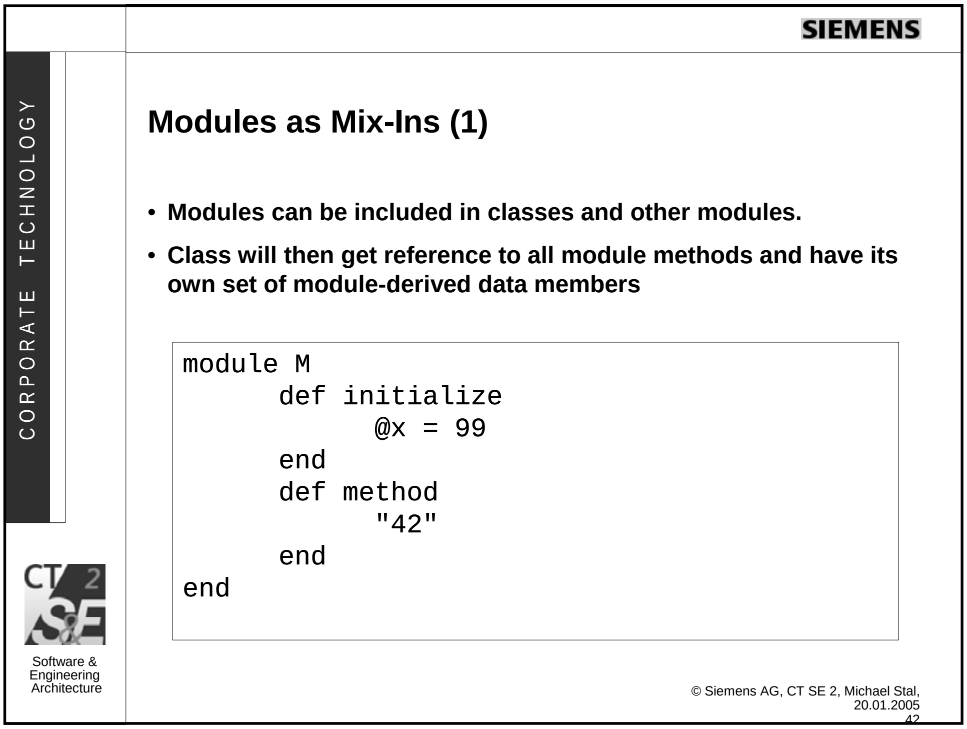
© Siemens AG, CT SE 2, Michael Stal, 20.01.2005 40



Modules

- A module is a non-instantiable class:

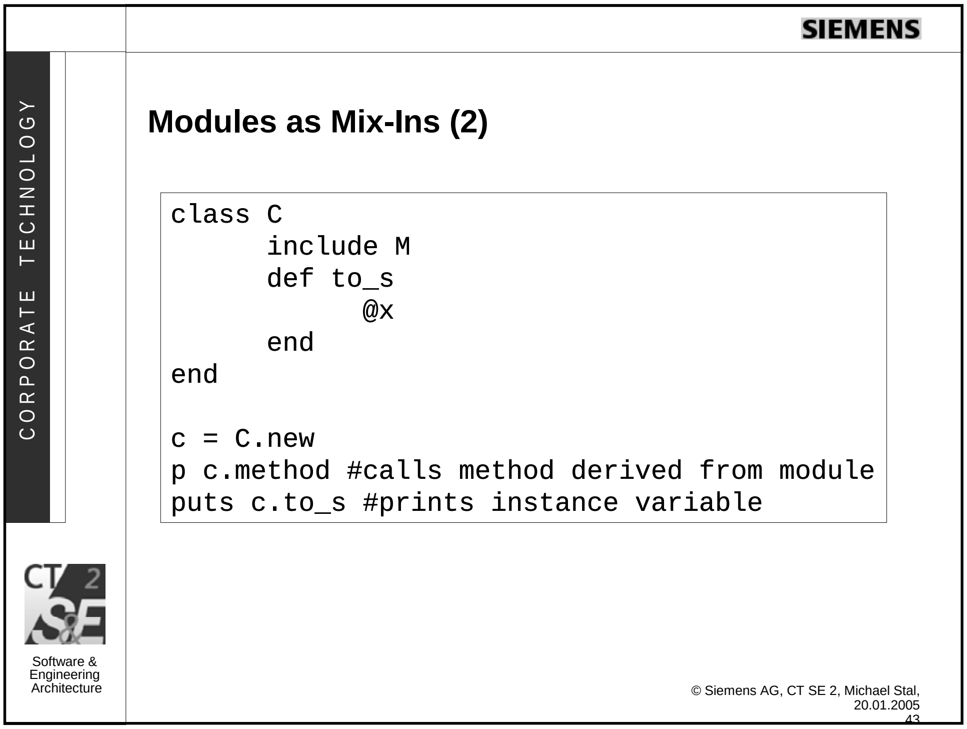
```
module M
      PI = 3.1415927
      def  calcArea(r)
            r*r*PI
      end
      def M.info
            "Trivial Math"
      end
end

include M # I am a lazy writer!
p PI
p calcArea(2)
p M.info
```



Modules as Mix-Ins (1)

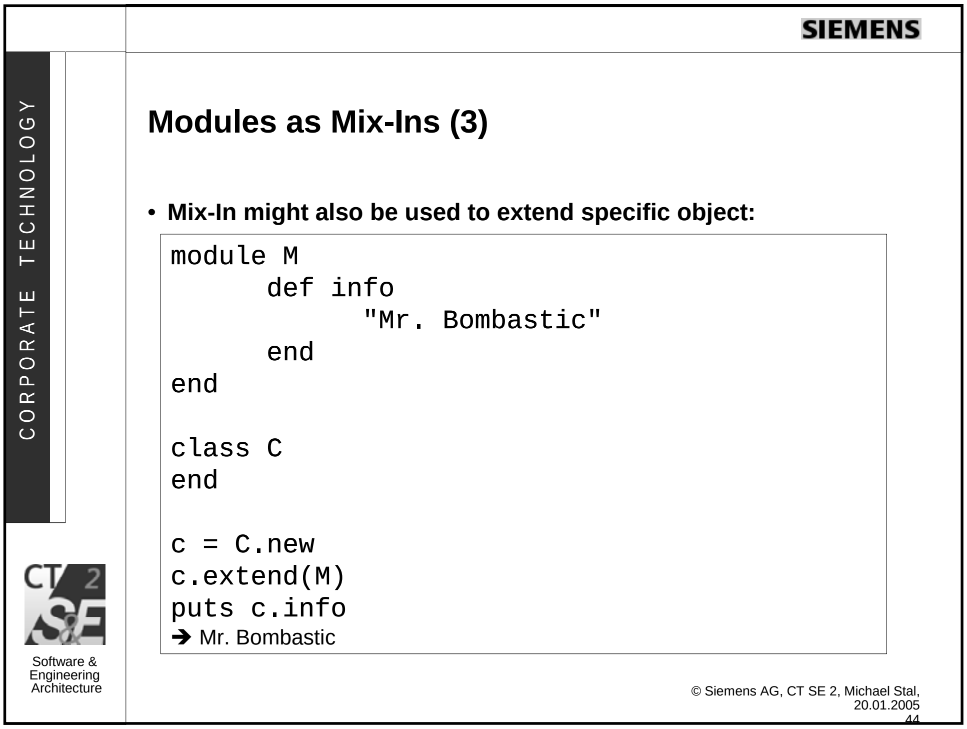
- Modules can be included in classes and other modules.
- Class will then get reference to all module methods and have its own set of module-derived data members

```
module M
      def initialize
            @x = 99
      end
      def method
            "42"
      end
end
```



Modules as Mix-Ins (2)

```
class C
      include M
      def to_s
            @x
      end
end

c = C.new
p c.method #calls method derived from module
puts c.to_s #prints instance variable
```



Modules as Mix-Ins (3)

- **Mix-In might also be used to extend specific object:**

```
module M
      def info
            "Mr. Bombastic"
      end
end

class C
end

c = C.new
c.extend(M)
puts c.info
```

➔ Mr. Bombastic

Modules as Mix-Ins (4)

- A predefined example for using a mix-in is the Singleton module

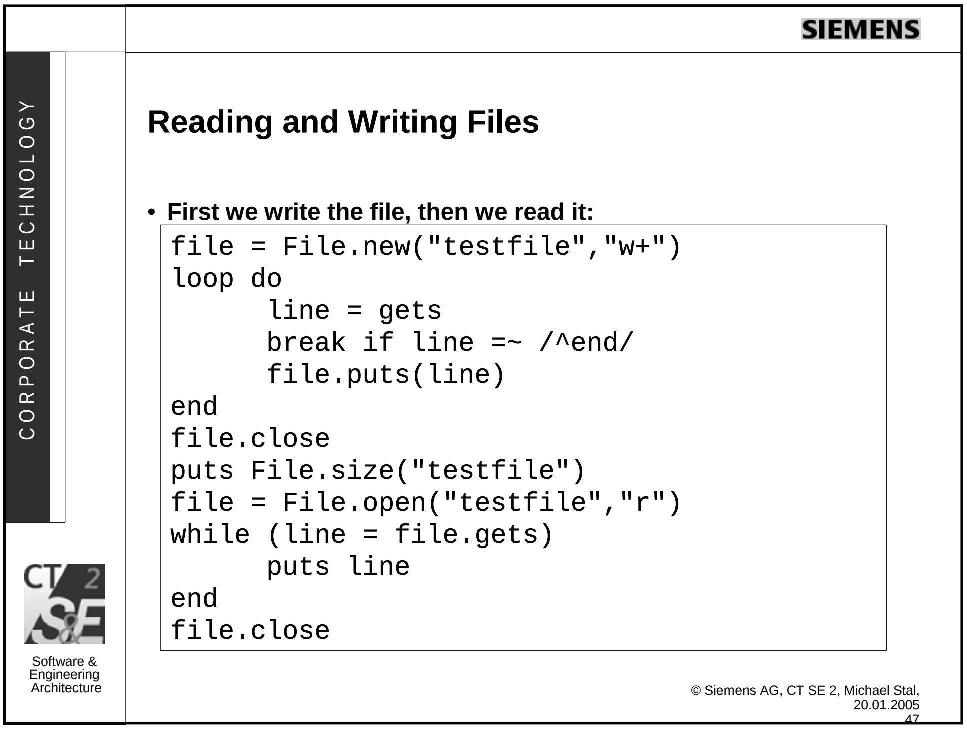
```
require "singleton"

class ValueObject
      include Singleton
      attr_accessor :val
end

a = ValueObject.instance
b = ValueObject.instance
a.val = "test it!"
puts b.val
```

➔ “test it”

I/O

- For I/O we might either use the Kernel primitves or leverage IO Objects
- `IO` is the base class from which `File` and others derive
- Standard IO, e.g.: `puts, gets, putc, print`:

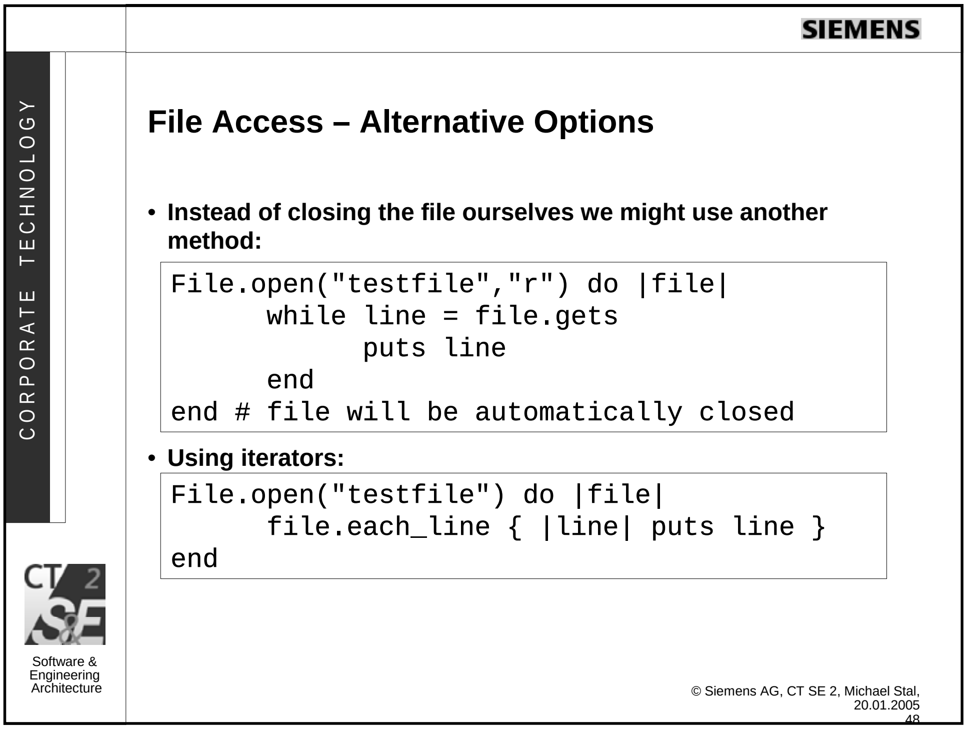
```
puts "Enter your name"
line = gets
putc line
printf("%x",42)
```



Reading and Writing Files

- **First we write the file, then we read it:**

```
file = File.new("testfile","w+")
loop do
      line = gets
      break if line =~ /^end/
      file.puts(line)
end
file.close
puts File.size("testfile")
file = File.open("testfile","r")
while (line = file.gets)
      puts line
end
file.close
```



File Access – Alternative Options

- **Instead of closing the file ourselves we might use another method:**

```
File.open("testfile","r") do |file|
      while line = file.gets
            puts line
      end
end # file will be automatically closed
```

- **Using iterators:**

```
File.open("testfile") do |file|
      file.each_line { |line| puts line }
end
```

Network IO Using Sockets (1)

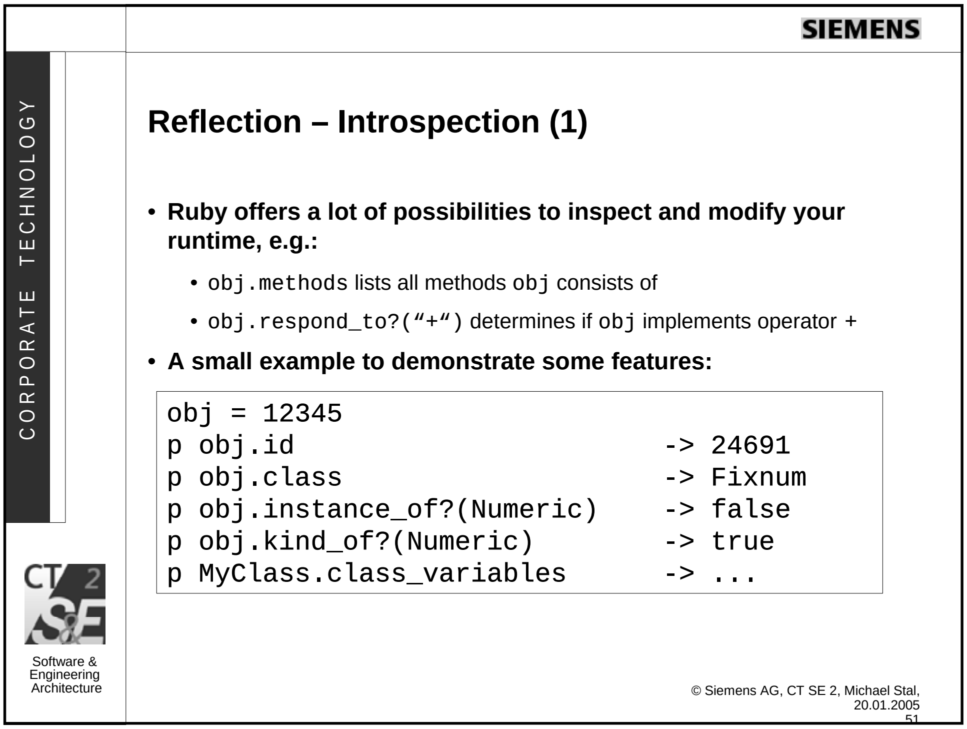
- **We are using a socket to implement a small web server:**

```
require "socket"
port = 9999
server = TCPServer.new("localhost", port)
while (session = server.accept)
       Thread.new do
              puts "Incoming request is
#{session.gets}"
              session.print"HTTP/1.1
200/OK\r\nContent-type: text/html\r\n\r\n"
       session.print"<html><body><h1>#{Time.now}<
/h1></body></html>\r\n"
              session.close
       end
end
```

Network IO using Sockets (2)

- **Let us access the aforementioned web server using a socket-based client:**

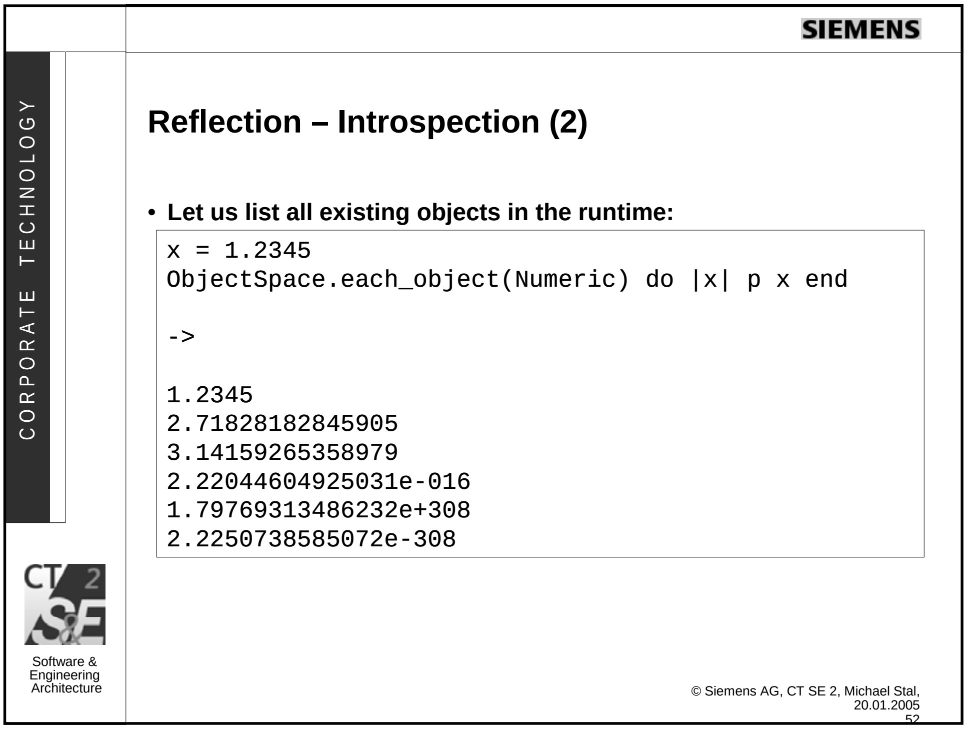
```
require 'socket‘
a = TCPSocket.new('localhost',9999)
a.puts("Get index.html")
while result = a.gets
       puts result
end
a.close
->
HTTP/1.1 200/OK
Content-type: text/html
<html><body><h1>Mon Dec 27 08:53:55 W. Europe Standard Time
2004</h1></body></html>
```



Reflection – Introspection (1)

- **Ruby offers a lot of possibilities to inspect and modify your runtime, e.g.:**
  - `obj.methods` lists all methods `obj` consists of
  - `obj.respond_to?(“+“)` determines if `obj` implements operator +
- **A small example to demonstrate some features:**

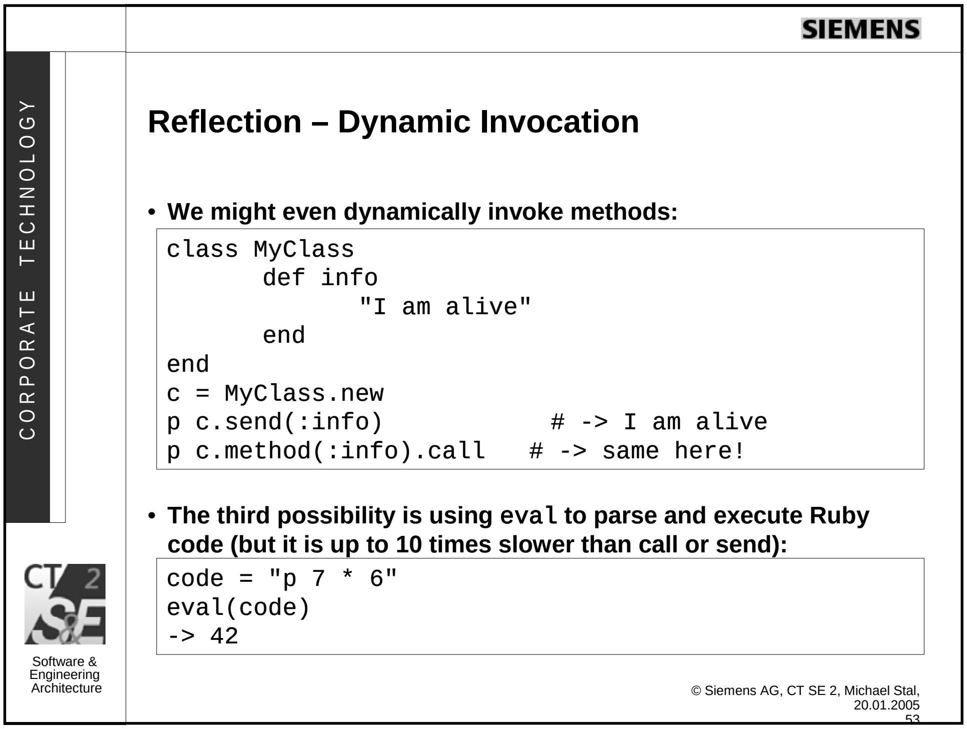
```
obj = 12345
p obj.id                        -> 24691
p obj.class                     -> Fixnum
p obj.instance_of?(Numeric)     -> false
p obj.kind_of?(Numeric)         -> true
p MyClass.class_variables       -> ...
```



Reflection – Introspection (2)

- **Let us list all existing objects in the runtime:**

```
x = 1.2345
ObjectSpace.each_object(Numeric) do |x| p x end

->

1.2345
2.71828182845905
3.14159265358979
2.22044604925031e-016
1.79769313486232e+308
2.2250738585072e-308
```



CORPORATE TECHNOLOGY

Reflection – Dynamic Invocation

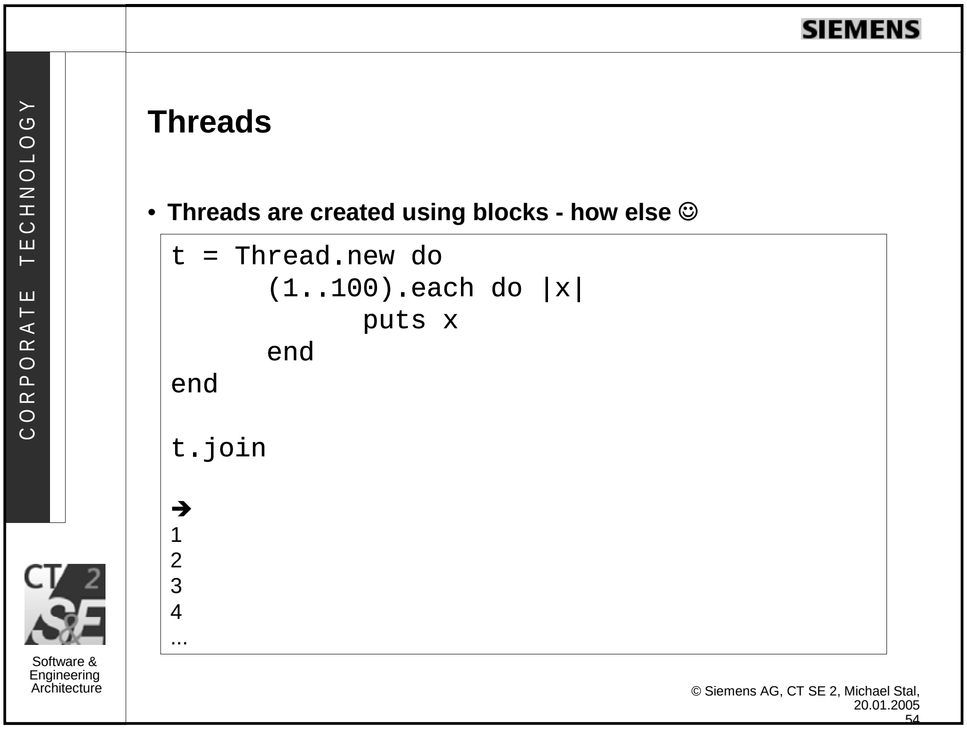
- **We might even dynamically invoke methods:**

```
class MyClass
       def info
              "I am alive"
       end
end
c = MyClass.new
p c.send(:info)            # -> I am alive
p c.method(:info).call   # -> same here!
```

- **The third possibility is using `eval` to parse and execute Ruby code (but it is up to 10 times slower than call or send):**

```
code = "p 7 * 6"
eval(code)
-> 42
```

Software & Engineering Architecture



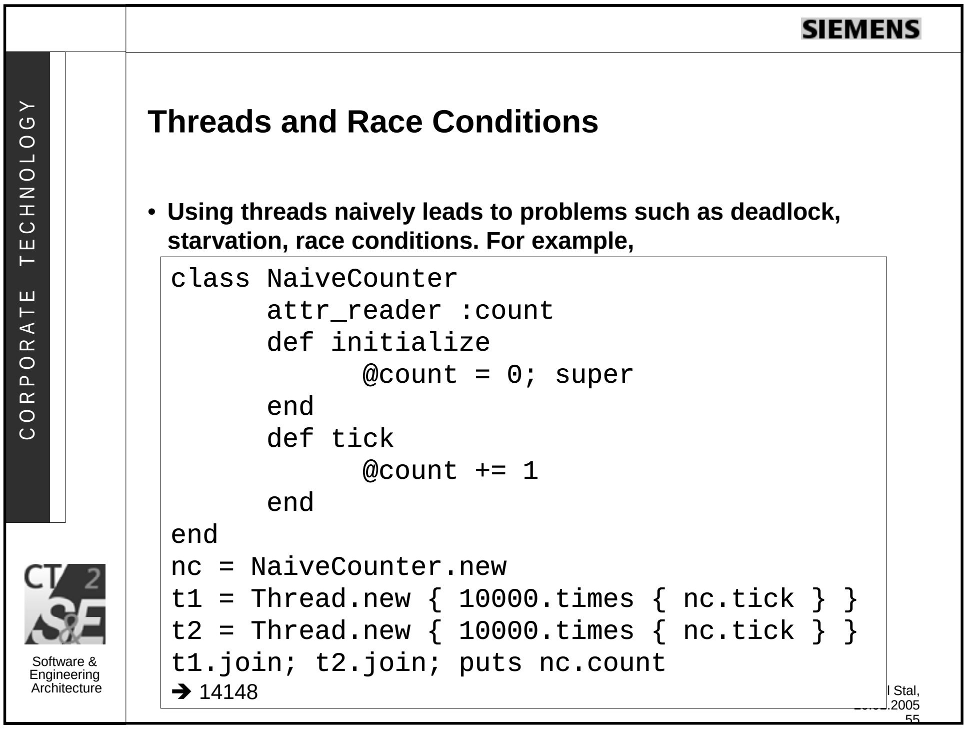
CORPORATE TECHNOLOGY

Threads

- **Threads are created using blocks - how else ☺**

```
t = Thread.new do
      (1..100).each do |x|
             puts x
      end
end

t.join

➔
1
2
3
4
…
```

Software & Engineering Architecture



SIEMENS

CORPORATE TECHNOLOGY

Threads and Race Conditions

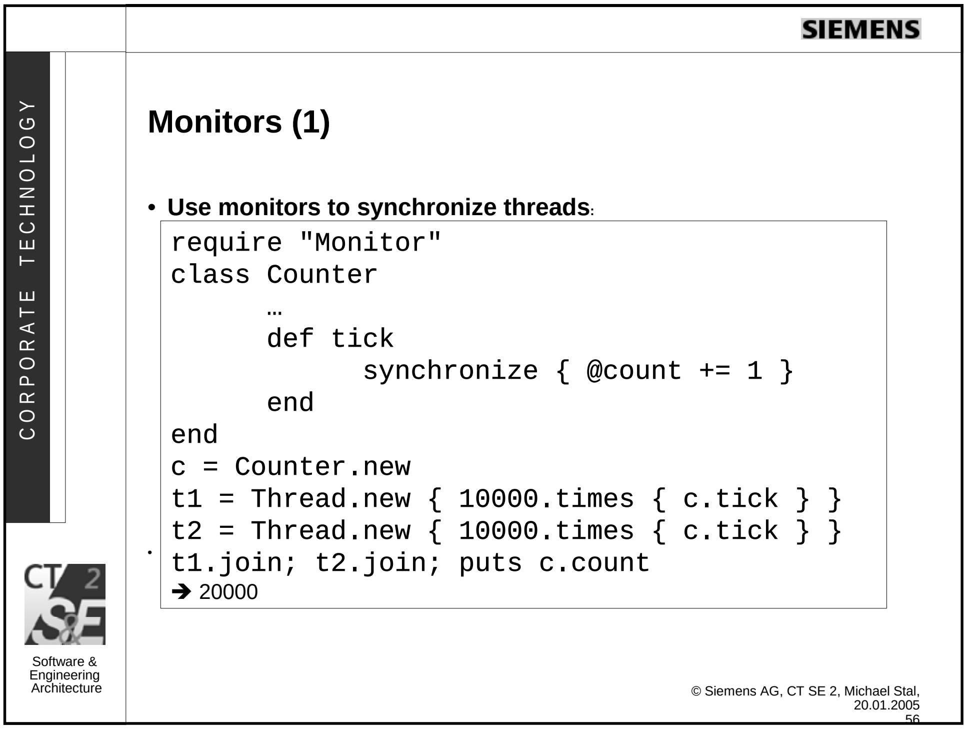
- Using threads naively leads to problems such as deadlock, starvation, race conditions. For example,

```
class NaiveCounter
      attr_reader :count
      def initialize
            @count = 0; super
      end
      def tick
            @count += 1
      end
end
nc = NaiveCounter.new
t1 = Thread.new { 10000.times { nc.tick } }
t2 = Thread.new { 10000.times { nc.tick } }
t1.join; t2.join; puts nc.count
```

➔ 14148

Software & Engineering Architecture

l Stal,
.2005
55



SIEMENS

CORPORATE TECHNOLOGY

Monitors (1)

- Use monitors to synchronize threads:

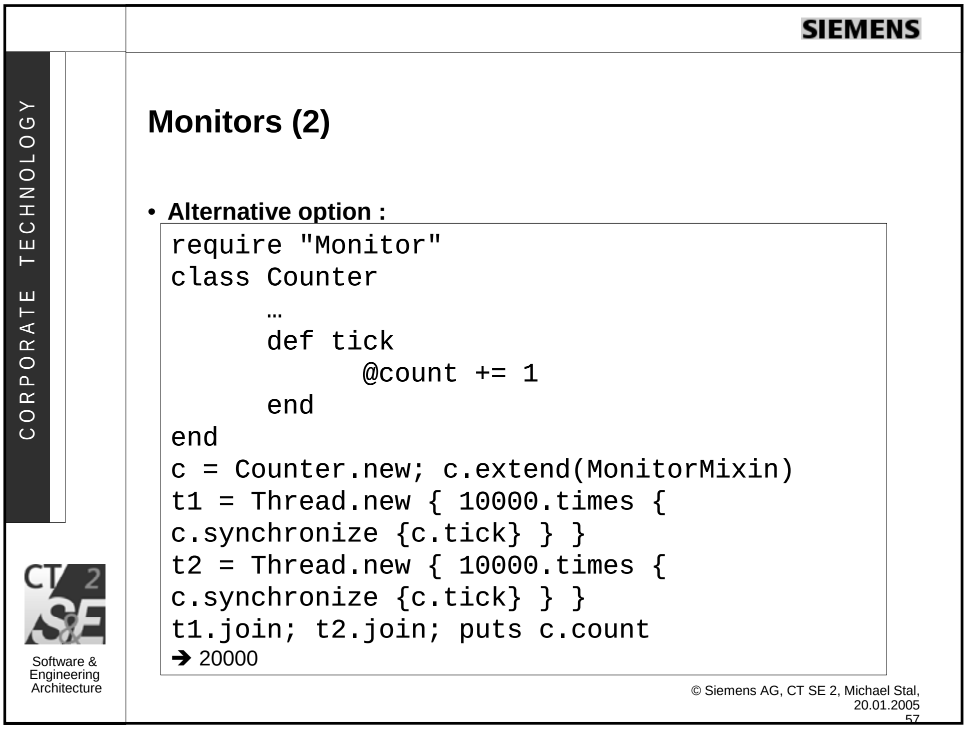
```
require "Monitor"
class Counter
      …
      def tick
            synchronize { @count += 1 }
      end
end
c = Counter.new
t1 = Thread.new { 10000.times { c.tick } }
t2 = Thread.new { 10000.times { c.tick } }
t1.join; t2.join; puts c.count
```

➔ 20000

•

Software & Engineering Architecture

© Siemens AG, CT SE 2, Michael Stal,
20.01.2005
56



SIEMENS

CORPORATE TECHNOLOGY

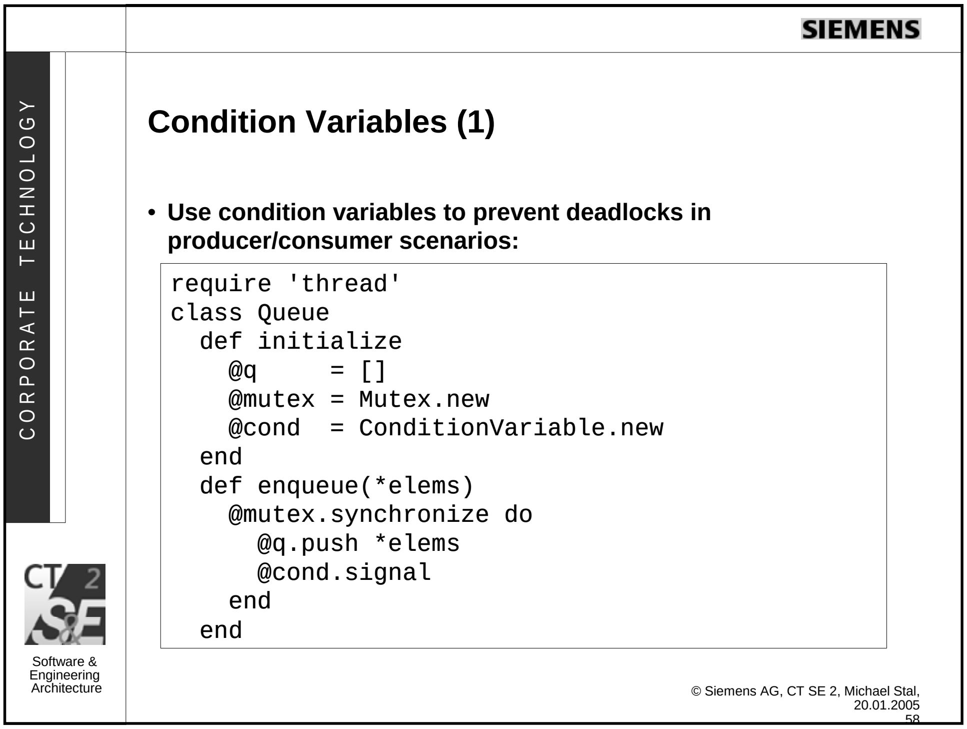
Monitors (2)

- **Alternative option :**

```
require "Monitor"
class Counter
      …
      def tick
            @count += 1
      end
end
c = Counter.new; c.extend(MonitorMixin)
t1 = Thread.new { 10000.times {
c.synchronize {c.tick} } }
t2 = Thread.new { 10000.times {
c.synchronize {c.tick} } }
t1.join; t2.join; puts c.count
```

➔ 20000

Software & Engineering Architecture



SIEMENS

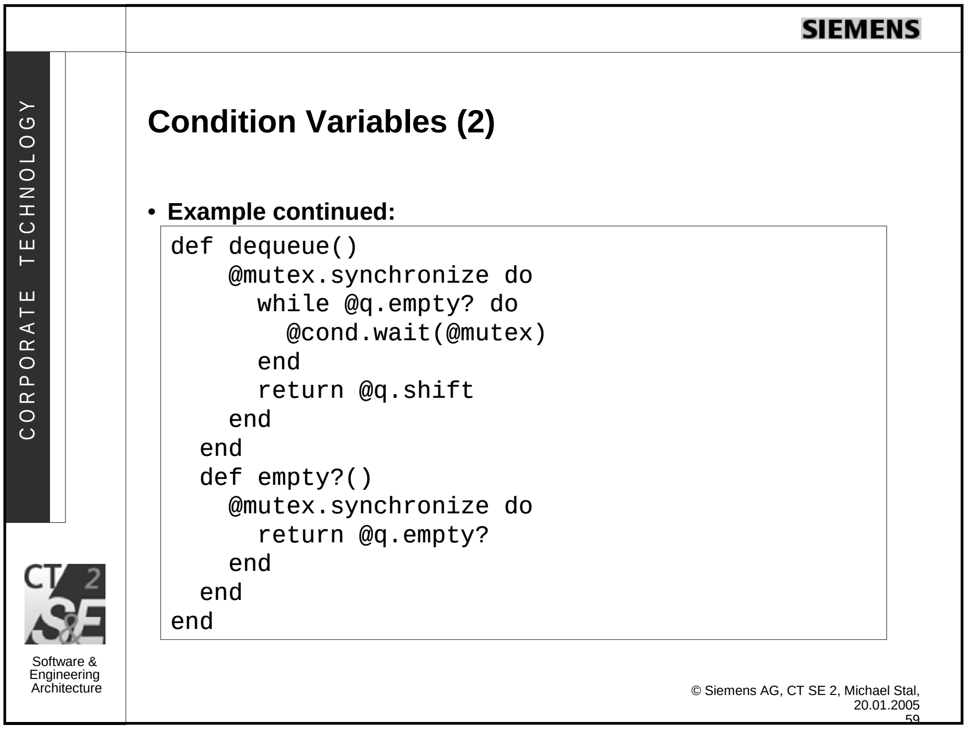
CORPORATE TECHNOLOGY

Condition Variables (1)

- **Use condition variables to prevent deadlocks in producer/consumer scenarios:**

```
require 'thread'
class Queue
  def initialize
    @q     = []
    @mutex = Mutex.new
    @cond  = ConditionVariable.new
  end
  def enqueue(*elems)
    @mutex.synchronize do
      @q.push *elems
      @cond.signal
    end
  end
```

Software & Engineering Architecture



CORPORATE TECHNOLOGY

Condition Variables (2)

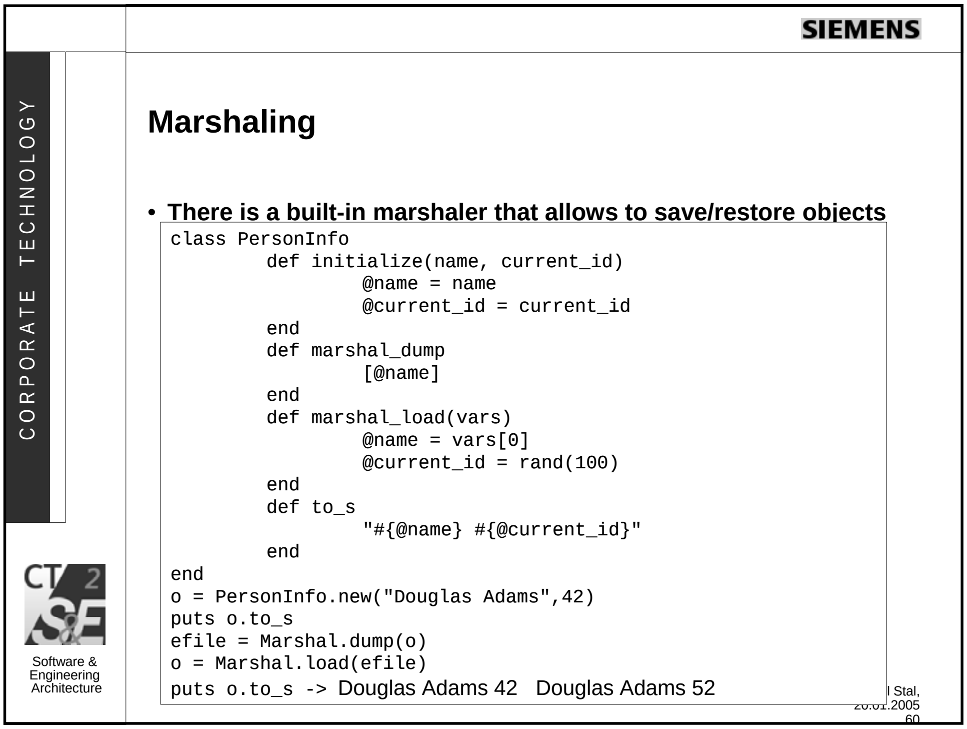
- **Example continued:**

```
def dequeue()
    @mutex.synchronize do
      while @q.empty? do
        @cond.wait(@mutex)
      end
      return @q.shift
    end
  end
  def empty?()
    @mutex.synchronize do
      return @q.empty?
    end
  end
end
```

Software & Engineering Architecture

© Siemens AG, CT SE 2, Michael Stal, 20.01.2005

59



CORPORATE TECHNOLOGY

Marshaling

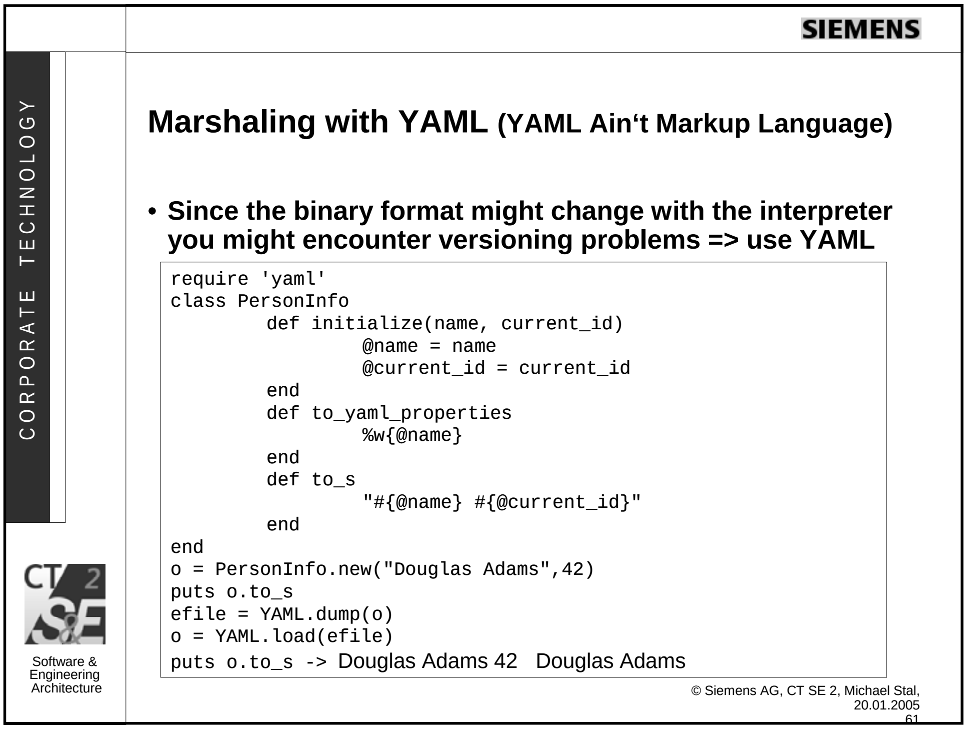
- **There is a built-in marshaler that allows to save/restore objects**

```
class PersonInfo
        def initialize(name, current_id)
                @name = name
                @current_id = current_id
        end
        def marshal_dump
                [@name]
        end
        def marshal_load(vars)
                @name = vars[0]
                @current_id = rand(100)
        end
        def to_s
                "#{@name} #{@current_id}"
        end
end
o = PersonInfo.new("Douglas Adams",42)
puts o.to_s
efile = Marshal.dump(o)
o = Marshal.load(efile)
puts o.to_s -> Douglas Adams 42   Douglas Adams 52
```

Software & Engineering Architecture

l Stal, 20.01.2005

60



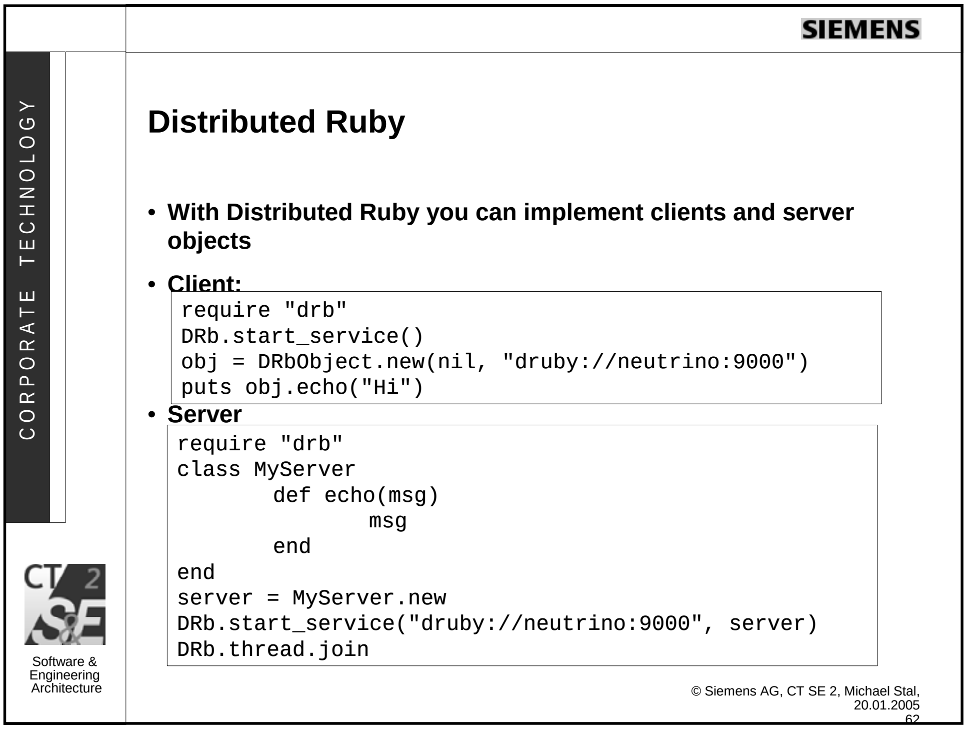
CORPORATE TECHNOLOGY

Marshaling with YAML (YAML Ain‘t Markup Language)

- Since the binary format might change with the interpreter you might encounter versioning problems => use YAML

```
require 'yaml'
class PersonInfo
        def initialize(name, current_id)
                @name = name
                @current_id = current_id
        end
        def to_yaml_properties
                %w{@name}
        end
        def to_s
                "#{@name} #{@current_id}"
        end
end
o = PersonInfo.new("Douglas Adams",42)
puts o.to_s
efile = YAML.dump(o)
o = YAML.load(efile)
puts o.to_s -> Douglas Adams 42   Douglas Adams
```

Software & Engineering Architecture



CORPORATE TECHNOLOGY

Distributed Ruby

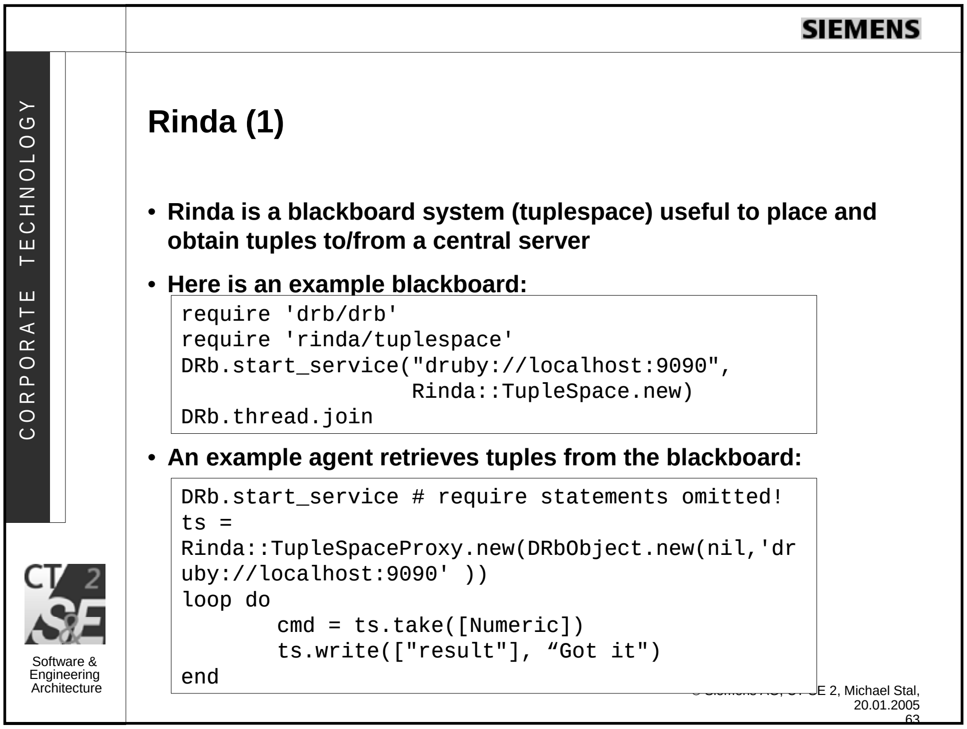
- With Distributed Ruby you can implement clients and server objects
- Client:

```
require "drb"
DRb.start_service()
obj = DRbObject.new(nil, "druby://neutrino:9000")
puts obj.echo("Hi")
```

- Server

```
require "drb"
class MyServer
        def echo(msg)
                msg
        end
end
server = MyServer.new
DRb.start_service("druby://neutrino:9000", server)
DRb.thread.join
```

Software & Engineering Architecture



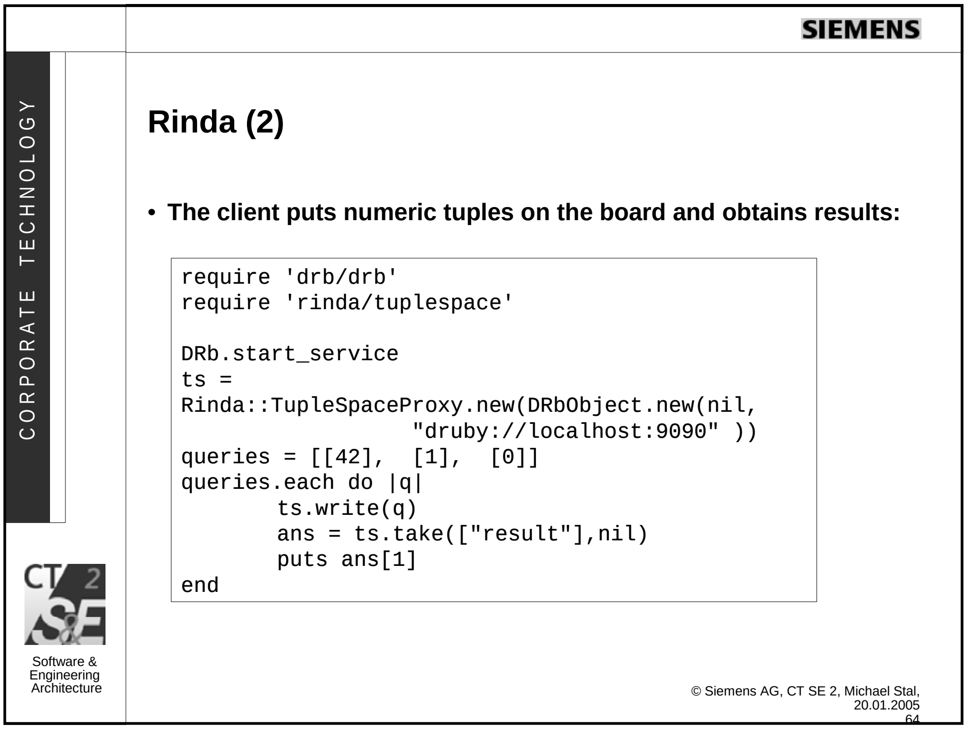
Rinda (1)

- Rinda is a blackboard system (tuplespace) useful to place and obtain tuples to/from a central server
- Here is an example blackboard:

```
require 'drb/drb'
require 'rinda/tuplespace'
DRb.start_service("druby://localhost:9090",
                  Rinda::TupleSpace.new)
DRb.thread.join
```

- An example agent retrieves tuples from the blackboard:

```
DRb.start_service # require statements omitted!
ts =
Rinda::TupleSpaceProxy.new(DRbObject.new(nil,'dr
uby://localhost:9090' ))
loop do
       cmd = ts.take([Numeric])
       ts.write(["result"], “Got it")
end
```



Rinda (2)

- The client puts numeric tuples on the board and obtains results:

```
require 'drb/drb'
require 'rinda/tuplespace'

DRb.start_service
ts =
Rinda::TupleSpaceProxy.new(DRbObject.new(nil,
                  "druby://localhost:9090" ))
queries = [[42],  [1],  [0]]
queries.each do |q|
       ts.write(q)
       ans = ts.take(["result"],nil)
       puts ans[1]
end
```

SOAP

- **Web services using Ruby [example taken from Ruby book]:**

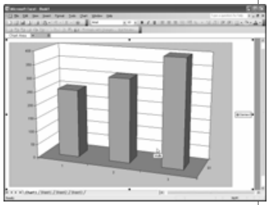
```
require 'soap/wsdlDriver'
require 'cgi'
WSDL_URL = "http://api.google.com/GoogleSearch.wsdl"
soap = SOAP::WSDLDriverFactory.new(WSDL_URL).createDriver
query = ‘ruby'
key = # get your Google key from file or elsewehere
result = soap.doGoogleSearch(key, query, 0, 1, false,
                             nil, false, nil, nil, nil)
printf "Estimated number of results is %d.\n",
       result.estimatedTotalResultsCount
printf "Your query took %6f seconds.\n", result.searchTime
first = result.resultElements[0]
puts first.title
puts first.URL
puts CGI.unescapeHTML(first.snippet)
->
```

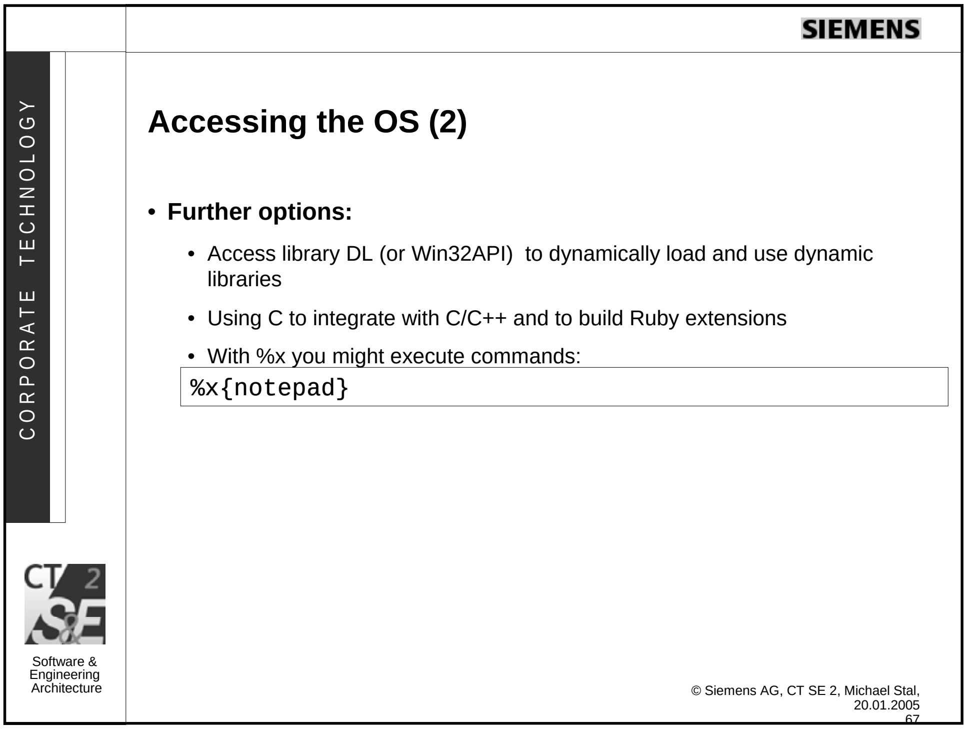
Estimated number of results is 4930000.
Your query took 0.229882 seconds.
<b>Ruby</b> Home Page
http://www.ruby-lang.org/ …

Accessing the OS (1) (Example: Win32)

- **WIN32OLE helps directly accessing OLE Automation:**

```
require 'win32ole'
excel = WIN32OLE.new("excel.application")
excel.Workbooks.Add()
excel['Visible'] = true
excel.Range("a1")['Value'] = 254
excel.Range("a2")['Value'] = 312
excel.Range("a3")['Value'] = 400
excel.Range("a1:a3").Select()
exchart = excel.charts.Add()
exchart['Type'] = -4100
30.step(180,5) do |rot|
       xchart.rotation = rot
       sleep(0.1)
end
excel.ActiveWorkbook.Close(0)
excel.Quit()
```



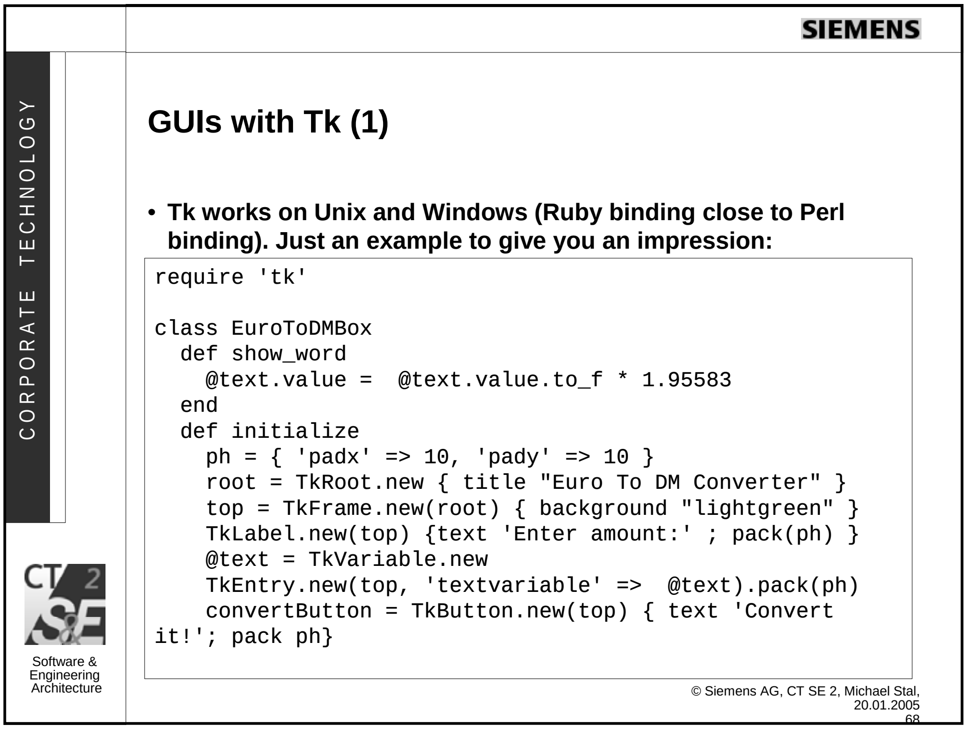


Accessing the OS (2)

- **Further options:**
  - Access library DL (or Win32API) to dynamically load and use dynamic libraries
  - Using C to integrate with C/C++ and to build Ruby extensions
  - With %x you might execute commands:

```
%x{notepad}
```

Software & Engineering Architecture



GUIs with Tk (1)

- **Tk works on Unix and Windows (Ruby binding close to Perl binding). Just an example to give you an impression:**

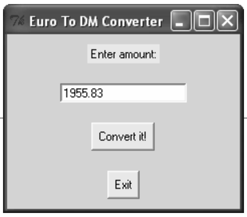
```
require 'tk'

class EuroToDMBox
  def show_word
    @text.value =  @text.value.to_f * 1.95583
  end
  def initialize
    ph = { 'padx' => 10, 'pady' => 10 }
    root = TkRoot.new { title "Euro To DM Converter" }
    top = TkFrame.new(root) { background "lightgreen" }
    TkLabel.new(top) {text 'Enter amount:' ; pack(ph) }
    @text = TkVariable.new
    TkEntry.new(top, 'textvariable' =>  @text).pack(ph)
    convertButton = TkButton.new(top) { text 'Convert
it!'; pack ph}
```

Software & Engineering Architecture

GUIs with Tk (2)

- **Example continued**

```
    . . .
    convertButton.command { show_word }
    exitButton = TkButton.new(top) {text 'Exit'; pack ph}
    exitButton.command { exit }
    top.pack('fill'=>'both', 'side' =>'top')
  end
end

EuroToDMBox.new
Tk.mainloop
```

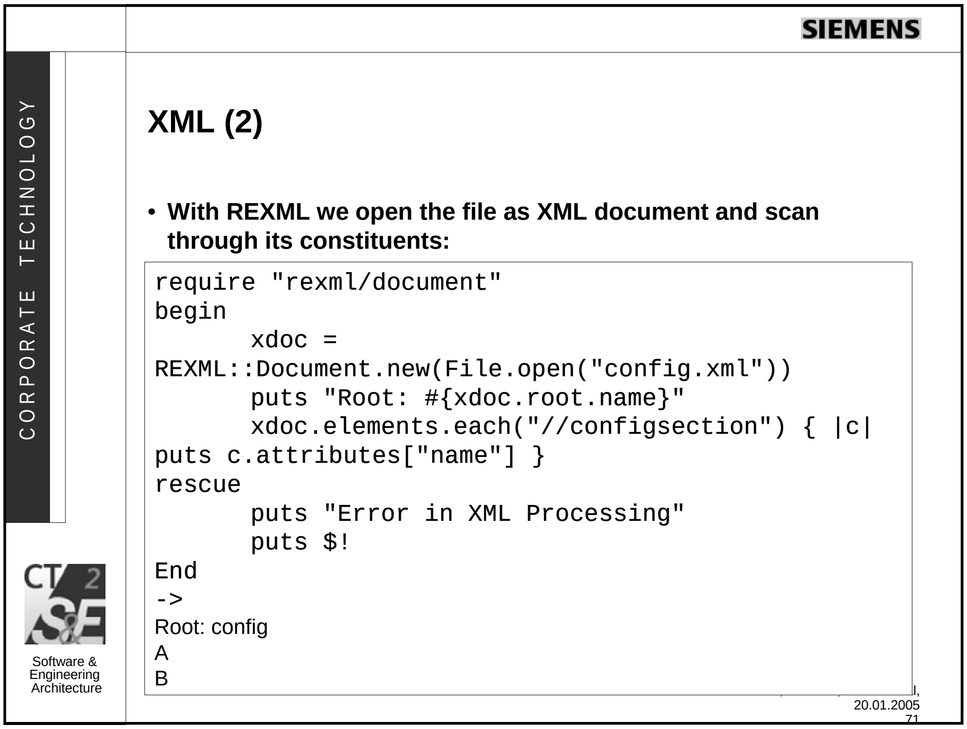


Enter amount: 1955.83

XML (1)

- **With the library REXML you are able to read and write XML documents**
- **In this talk we will focus on reading XML**
- **Assume, we have the following XML file „config.xml“:**

```
<config>
       <configsection name = "A">
              <iterations>
                     100
              </iterations>
       </configsection>
       <configsection name = "B">
              <repeatonerror>
                     true
              </repeatonerror>
       </configsection>
</config>
```



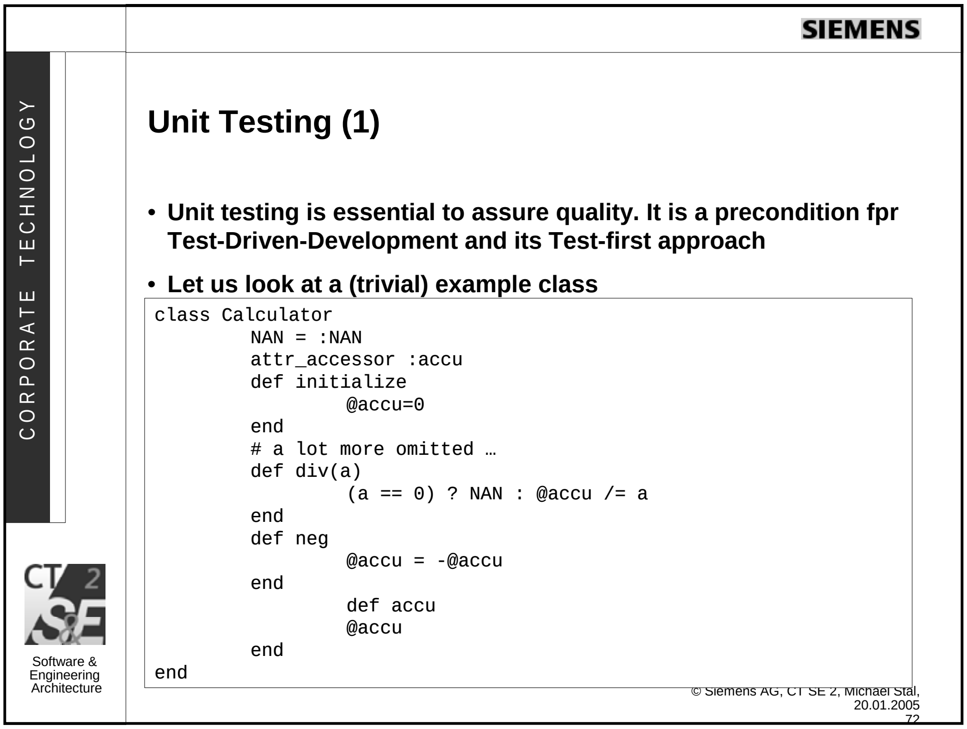
CORPORATE TECHNOLOGY

XML (2)

- With REXML we open the file as XML document and scan through its constituents:

```
require "rexml/document"
begin
	xdoc =
REXML::Document.new(File.open("config.xml"))
	puts "Root: #{xdoc.root.name}"
	xdoc.elements.each("//configsection") { |c|
puts c.attributes["name"] }
rescue
	puts "Error in XML Processing"
	puts $!
End
->
Root: config
A
B
```

Software & Engineering Architecture



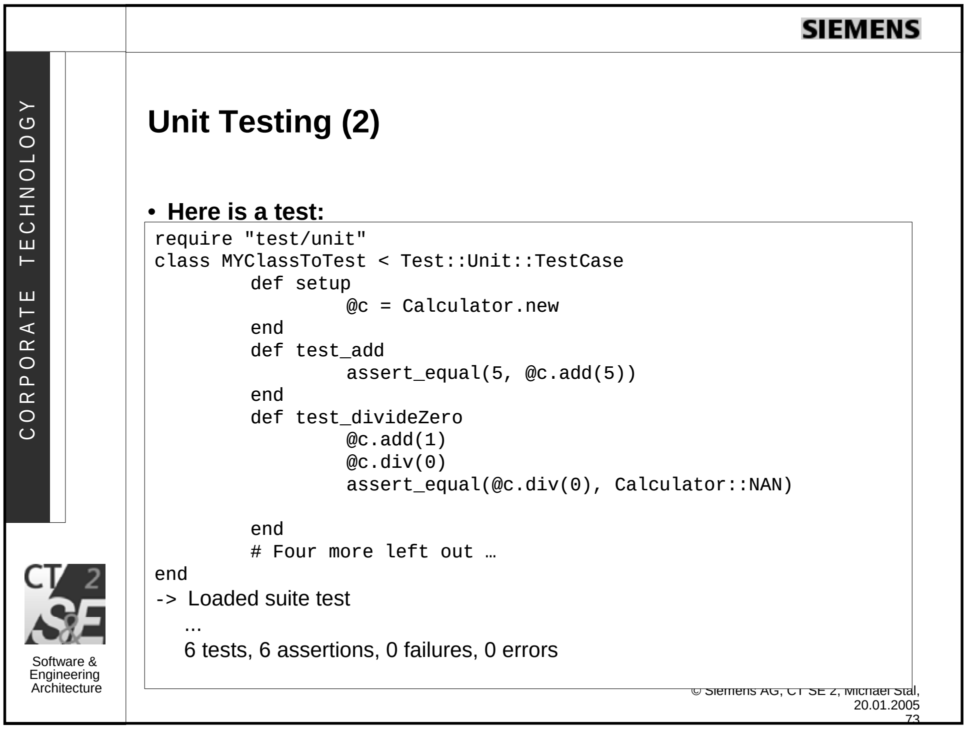
CORPORATE TECHNOLOGY

Unit Testing (1)

- Unit testing is essential to assure quality. It is a precondition fpr Test-Driven-Development and its Test-first approach
- Let us look at a (trivial) example class

```
class Calculator
	NAN = :NAN
	attr_accessor :accu
	def initialize
		@accu=0
	end
	# a lot more omitted …
	def div(a)
		(a == 0) ? NAN : @accu /= a
	end
	def neg
		@accu = -@accu
	end
		def accu
		@accu
	end
end
```

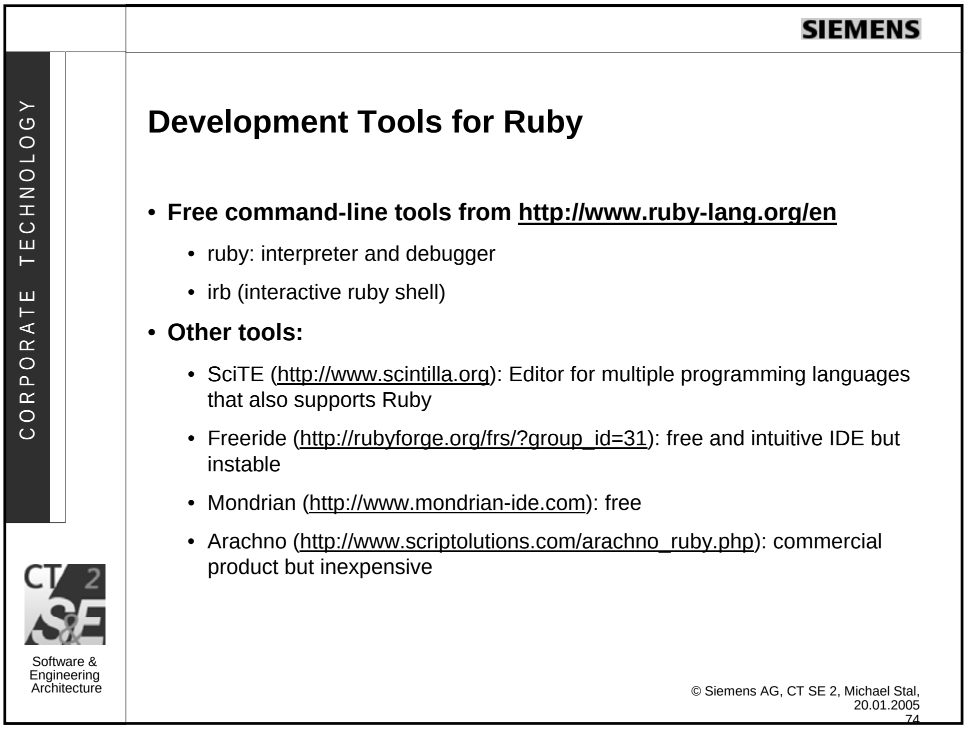
Software & Engineering Architecture



Unit Testing (2)

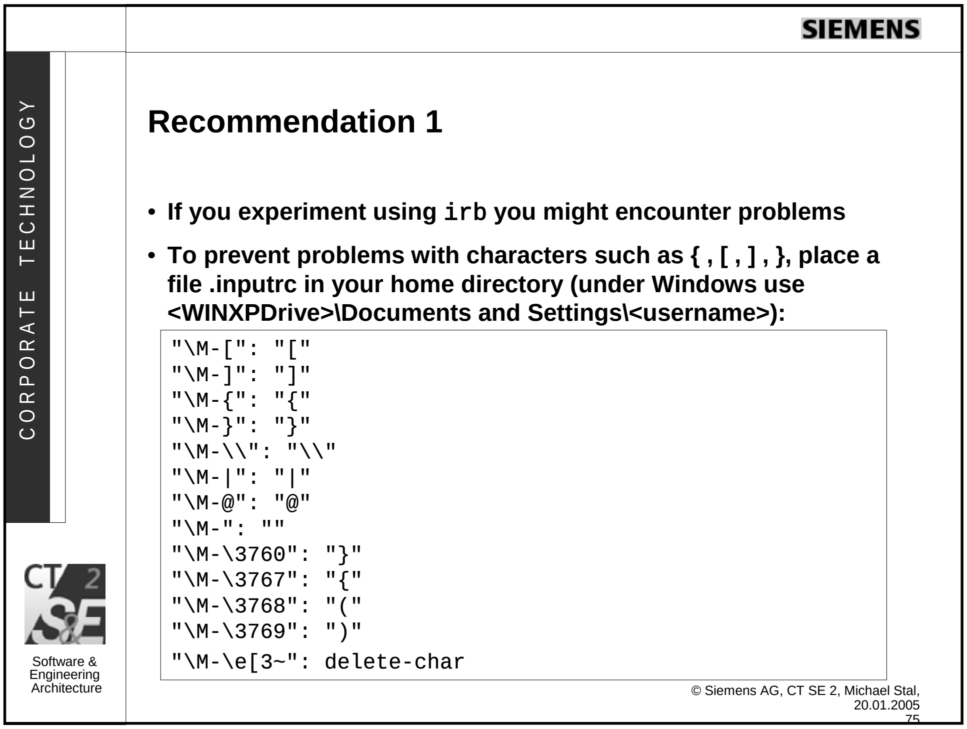
- **Here is a test:**

```
require "test/unit"
class MYClassToTest < Test::Unit::TestCase
        def setup
                @c = Calculator.new
        end
        def test_add
                assert_equal(5, @c.add(5))
        end
        def test_divideZero
                @c.add(1)
                @c.div(0)
                assert_equal(@c.div(0), Calculator::NAN)

        end
        # Four more left out …
end
-> Loaded suite test
   ...
   6 tests, 6 assertions, 0 failures, 0 errors
```



Development Tools for Ruby

- **Free command-line tools from http://www.ruby-lang.org/en**
  - ruby: interpreter and debugger
  - irb (interactive ruby shell)
- **Other tools:**
  - SciTE (http://www.scintilla.org): Editor for multiple programming languages that also supports Ruby
  - Freeride (http://rubyforge.org/frs/?group_id=31): free and intuitive IDE but instable
  - Mondrian (http://www.mondrian-ide.com): free
  - Arachno (http://www.scriptolutions.com/arachno_ruby.php): commercial product but inexpensive



SIEMENS

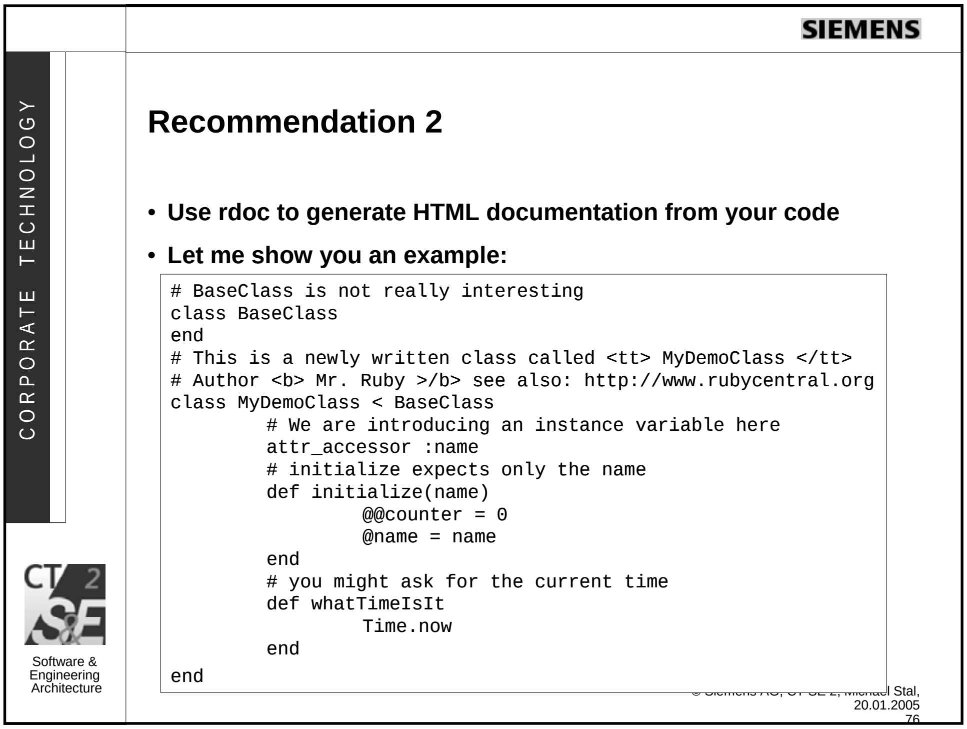
CORPORATE TECHNOLOGY

Recommendation 1

- If you experiment using `irb` you might encounter problems
- To prevent problems with characters such as { , [,] , }, place a file .inputrc in your home directory (under Windows use <WINXPDrive>\Documents and Settings\<username>):

```
"\M-[": "["
"\M-]": "]"
"\M-{": "{"
"\M-}": "}"
"\M-\\": "\\"
"\M-|": "|"
"\M-@": "@"
"\M-": ""
"\M-\3760": "}"
"\M-\3767": "{"
"\M-\3768": "("
"\M-\3769": ")"
"\M-\e[3~": delete-char
```

Software & Engineering Architecture



SIEMENS

CORPORATE TECHNOLOGY

Recommendation 2

- Use rdoc to generate HTML documentation from your code
- Let me show you an example:

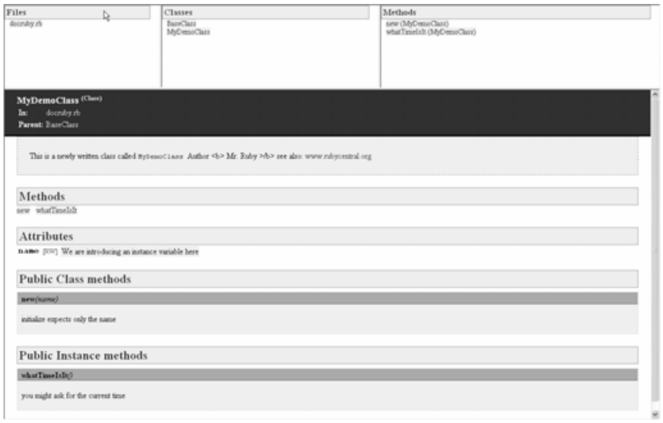
```
# BaseClass is not really interesting
class BaseClass
end
# This is a newly written class called <tt> MyDemoClass </tt>
# Author <b> Mr. Ruby >/b> see also: http://www.rubycentral.org
class MyDemoClass < BaseClass
        # We are introducing an instance variable here
        attr_accessor :name
        # initialize expects only the name
        def initialize(name)
                @@counter = 0
                @name = name
        end
        # you might ask for the current time
        def whatTimeIsIt
                Time.now
        end
end
```

Software & Engineering Architecture

Recommendation 2 (2)

- Now generate html files with

```
rdoc file_name.rb –o output_dir
```



docruby.rb

BaseClass
MyDemoClass

new (MyDemoClass)
whatTimeIsIt (MyDemoClass)

MyDemoClass (Class)
In: docruby.rb
Parent: BaseClass

This is a newly written class called `MyDemoClass` Author <b> Mr. Ruby >/b> see also: www.rubycentral.org

Methods

new whatTimeIsIt

Attributes

name [RW] We are introducing an instance variable here

Public Class methods

new(name)

initialize expects only the name

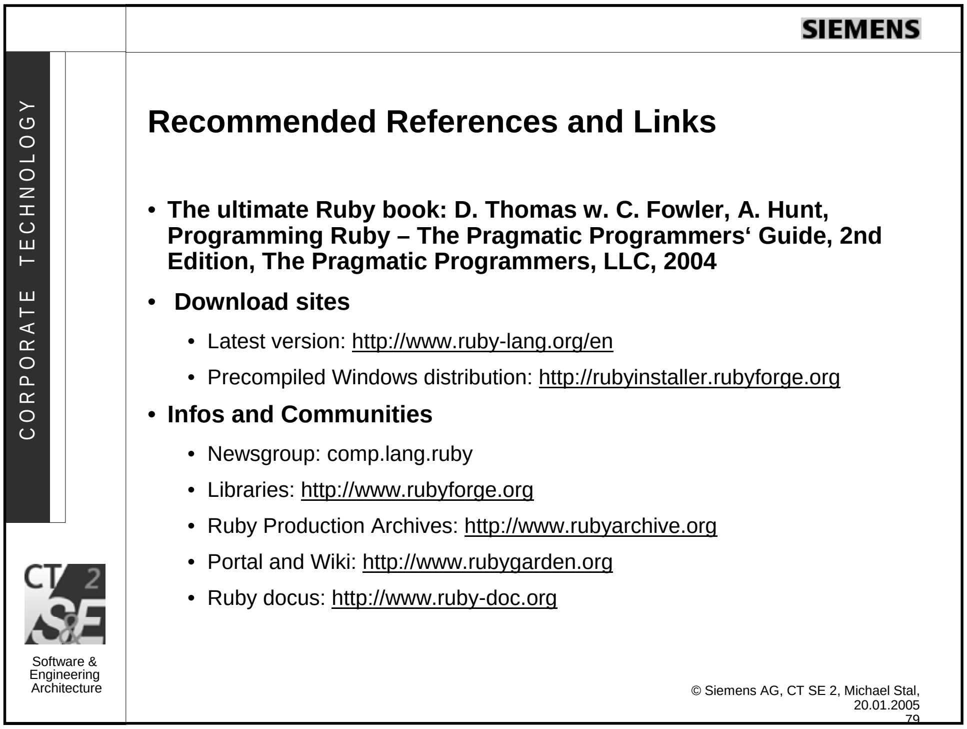
Public Instance methods

whatTimeIsIt()

you might ask for the current time

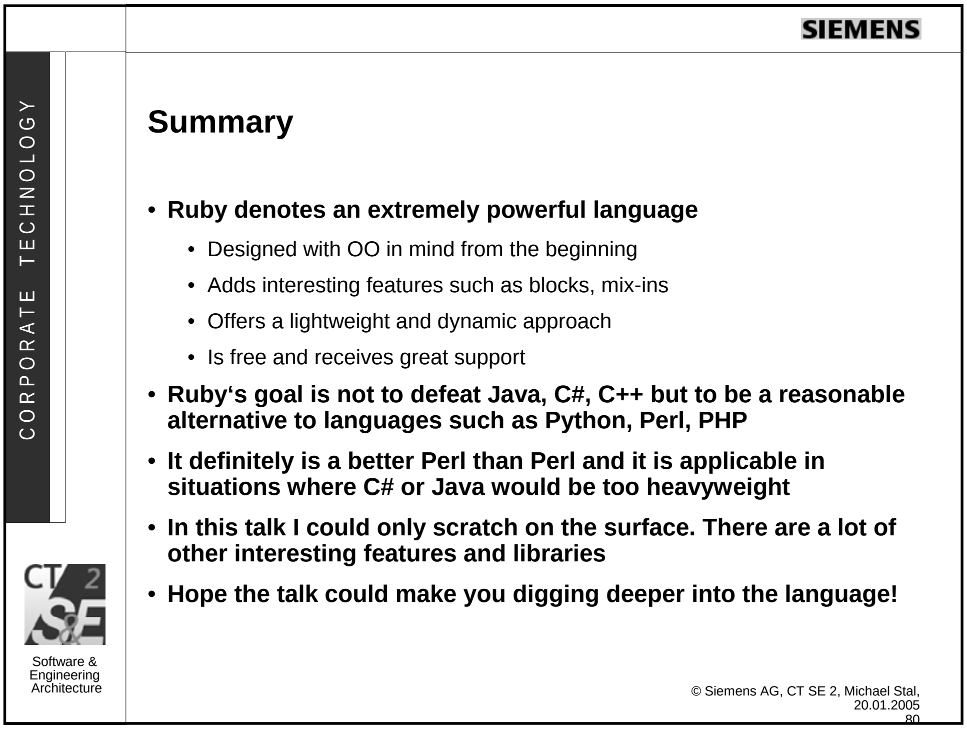
Recommendation 3

- Ruby finds all installed packages per default in the libraries denoted by `puts $:`
- Set `RUBYLIB` to add your own libraries or use `–i` as command line option
- Or better: consider RubyGems as an standard packaging and installation framework for libraries
  - Standardized package format
  - Central repository
  - Installation/management of different versions
  - End-user tools – query, modify, ...



Recommended References and Links

- **The ultimate Ruby book: D. Thomas w. C. Fowler, A. Hunt, Programming Ruby – The Pragmatic Programmers‘ Guide, 2nd Edition, The Pragmatic Programmers, LLC, 2004**
- **Download sites**
  - Latest version: http://www.ruby-lang.org/en
  - Precompiled Windows distribution: http://rubyinstaller.rubyforge.org
- **Infos and Communities**
  - Newsgroup: comp.lang.ruby
  - Libraries: http://www.rubyforge.org
  - Ruby Production Archives: http://www.rubyarchive.org
  - Portal and Wiki: http://www.rubygarden.org
  - Ruby docus: http://www.ruby-doc.org



Summary

- **Ruby denotes an extremely powerful language**
  - Designed with OO in mind from the beginning
  - Adds interesting features such as blocks, mix-ins
  - Offers a lightweight and dynamic approach
  - Is free and receives great support
- **Ruby‘s goal is not to defeat Java, C#, C++ but to be a reasonable alternative to languages such as Python, Perl, PHP**
- **It definitely is a better Perl than Perl and it is applicable in situations where C# or Java would be too heavyweight**
- **In this talk I could only scratch on the surface. There are a lot of other interesting features and libraries**
- **Hope the talk could make you digging deeper into the language!**