Some Design Patterns In Ruby

Just because you have duck-typing doesn't mean you can ignore common OO idioms!

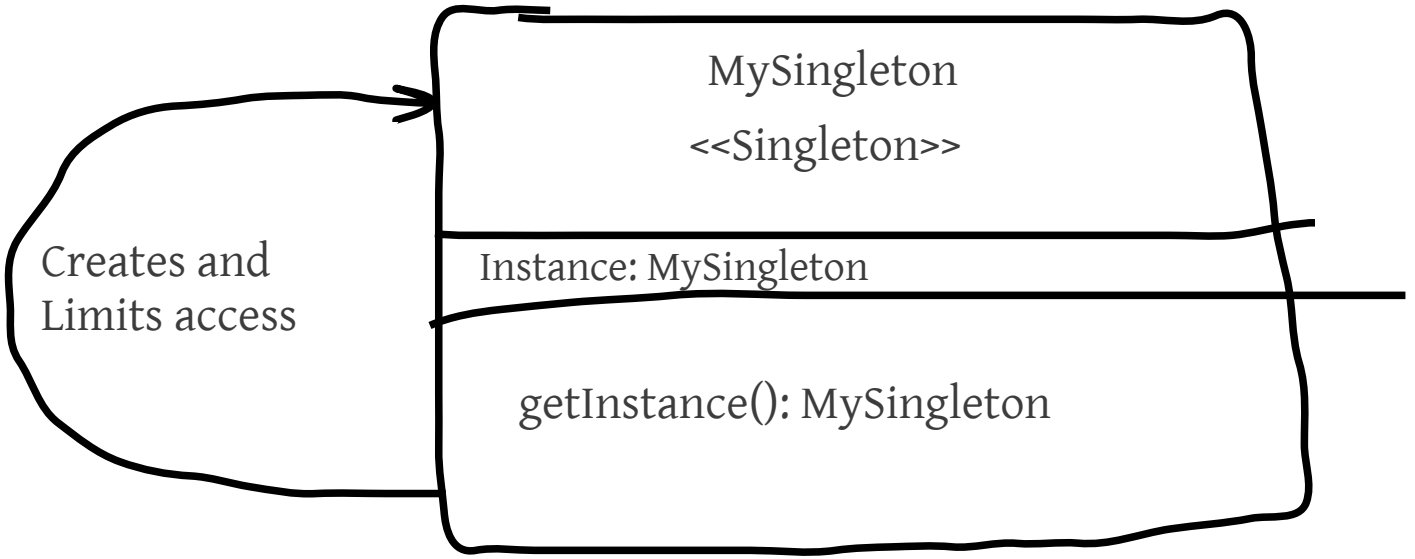
https://github.com/abramhindle/ruby-design-patterns
http://softwareprocess.es
abram.hindle@softwareprocess.es

Singleton Pattern

- Restrict instatiation of a class to 1 object.
- Allow only 1 instance of a class.
- Like a global, but hopefully less mutable.
- Can be lazily instantiated
- Much maligned.

Singleton Pattern

- Necessary UML



Square boxes are for suckers

Singleton Pattern

- In Ruby you generally limit access via a gentleman's agreement we won't peak the guts.
  - I think we should avoid overcomplicating things and stick to this agreement.
-
```
# http://apidock.com/ruby/Module/private_class_method
# http://dalibornasevic.com/posts/9-ruby-singleton-pattern-again
class MySingleton
   # attempt to limit access to the constructor
   private_class_method :new
   # class variable instance
   @@instance = nil
   def self.instance()
      if (@@instance.nil?())
          # self. and MySingleton tend not to work
          @@instance = self.new()
      end
      return @@instance
   end
end
```

Singleton

- There are many more ways to make a singleton
- The singleton module is pretty useful

```
require 'singleton'
# http://dalibornasevic.com/posts/9-ruby-singleton-pattern-again

class IncSingleton
   include Singleton
end
```

Singleton Resources

- Ruby Singleton Pattern Again
  http://dalibornasevic.com/posts/9-ruby-singleton-pattern-again
  Dalibor Nasevic
- Singleton Pattern http://c2.com/cgi/wiki?SingletonPattern
- Singleton Pattern
  http://en.wikipedia.org/wiki/Singleton_pattern
- Method private_class_method
  http://apidock.com/ruby/Module/private_class_method

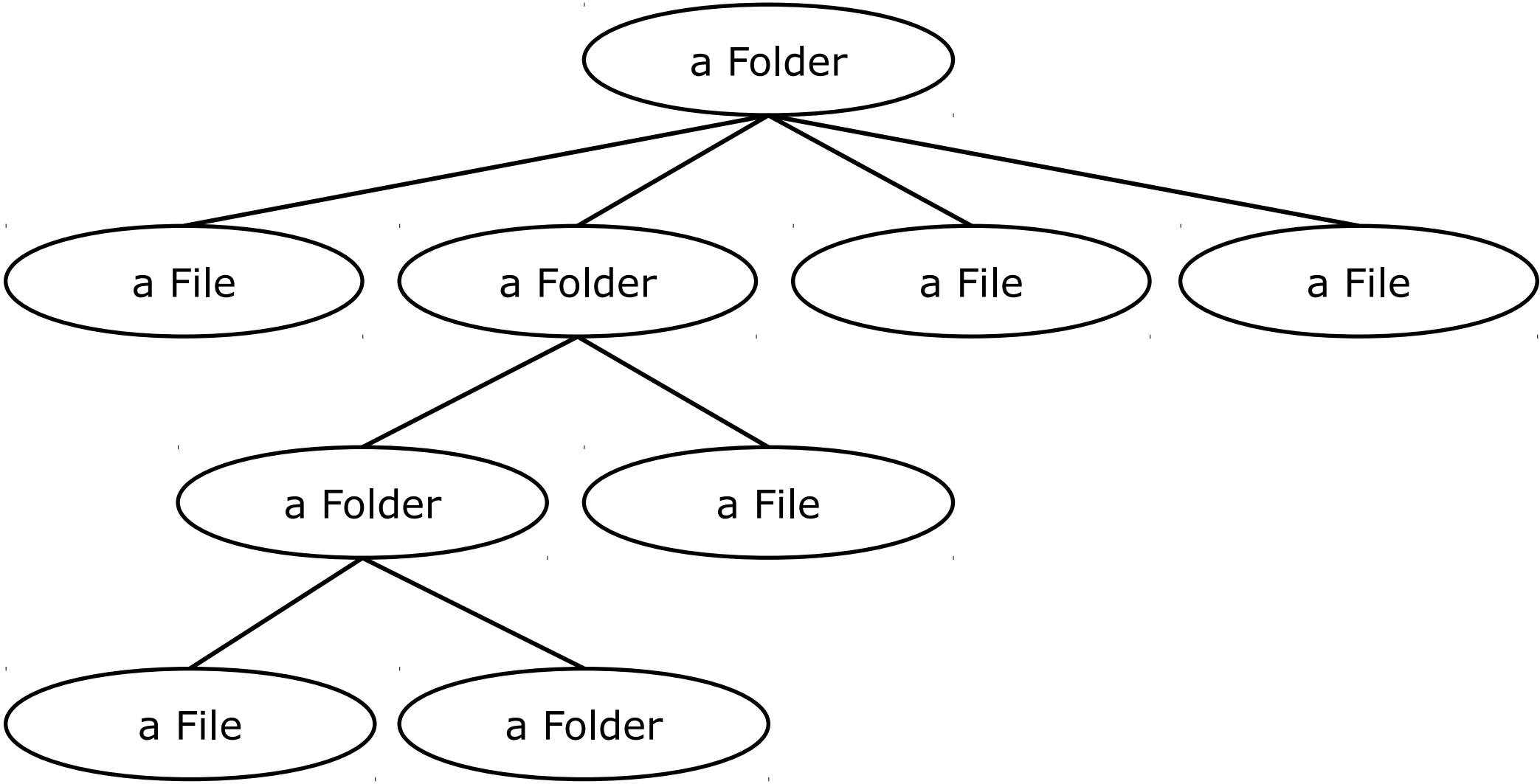
Composite Pattern

- Design intent:
  - to compose individual objects to build up a tree structure
    - e.g., a folder can contain files and other folders

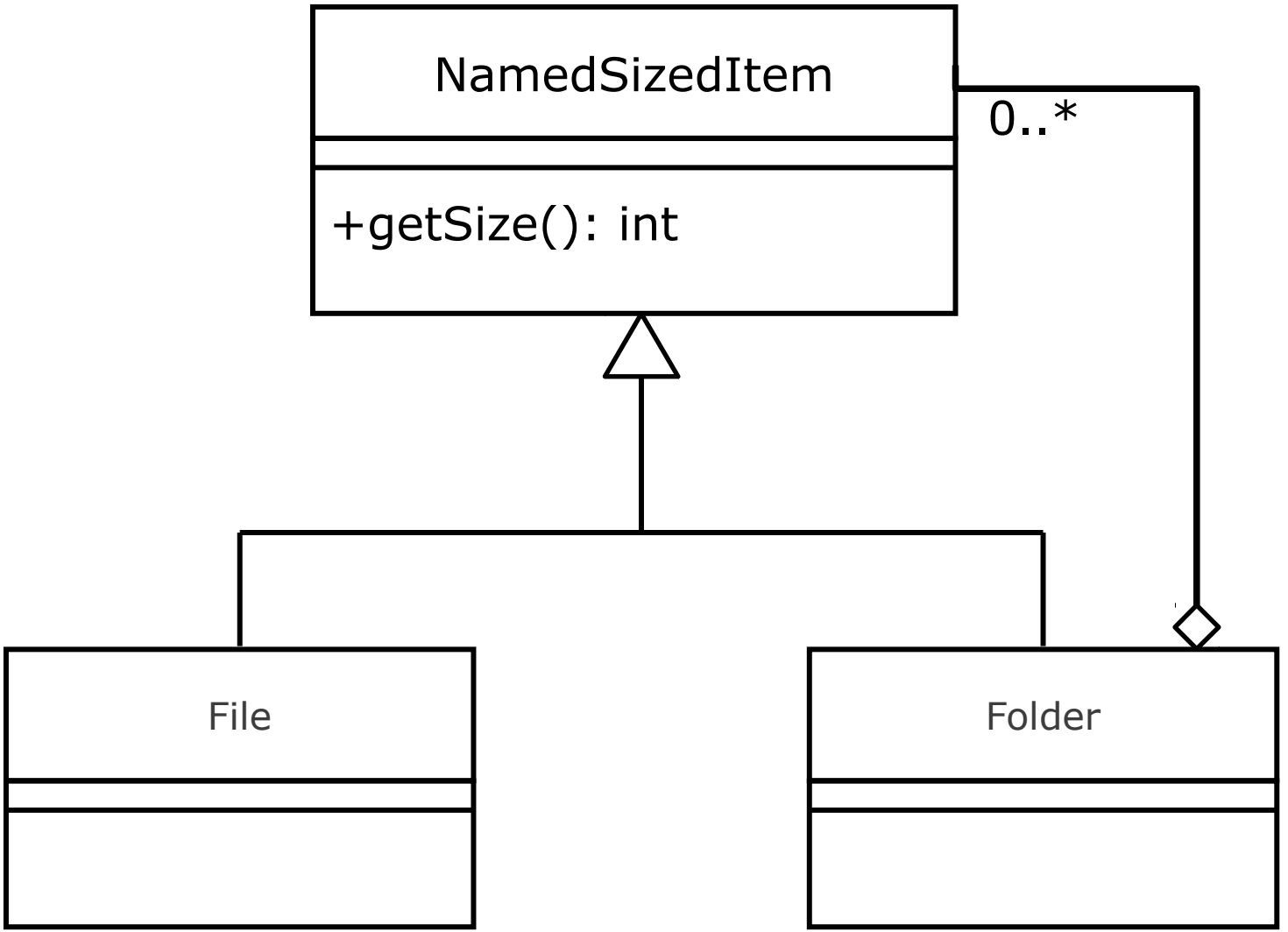
- The individual objects and the composed objects are treated uniformly
  - e.g., files and folders both have a name or a size

- Each object is responsible for answering its own query.

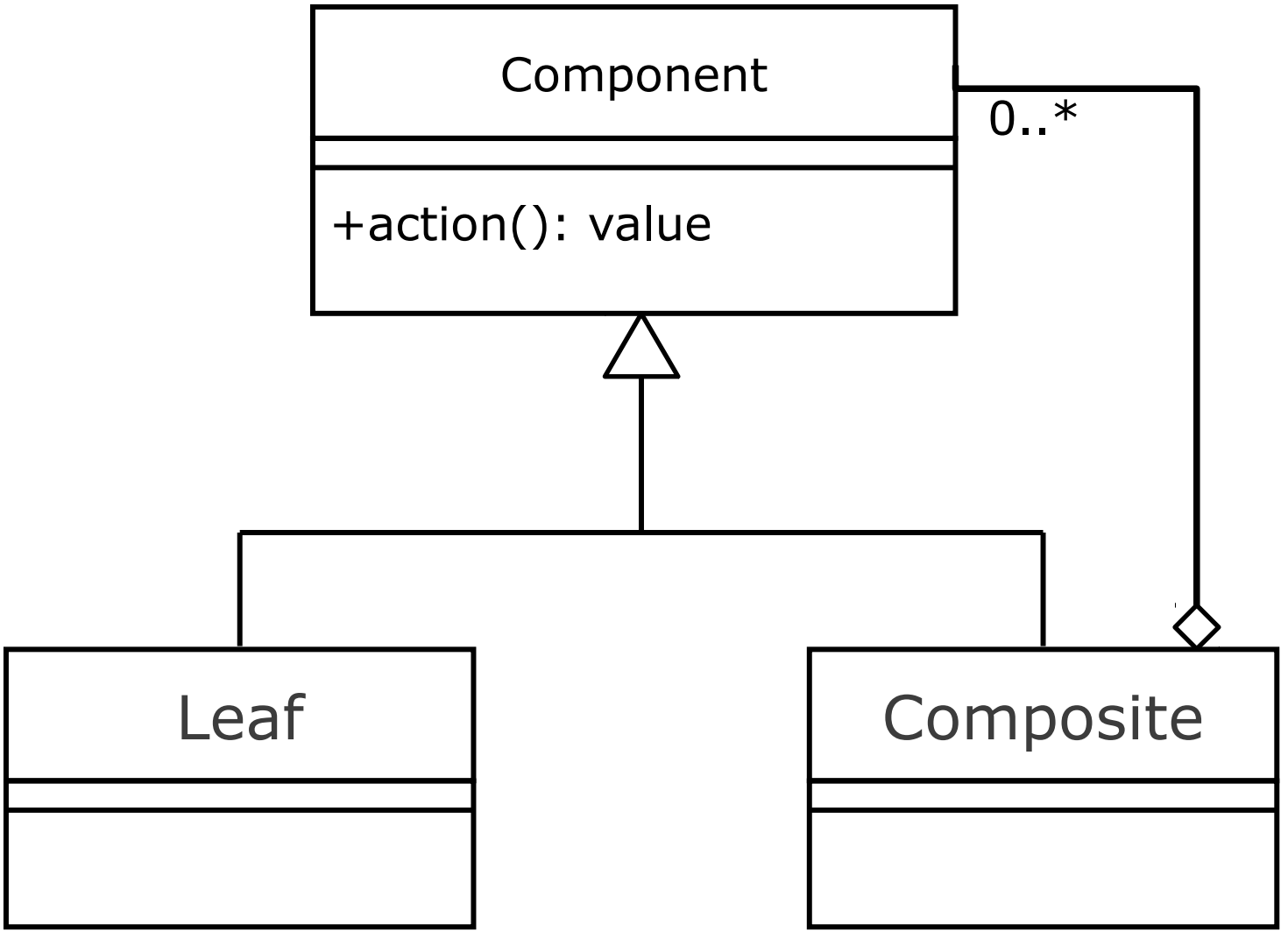
“Recursive (De)composition”



Composite Example Structure



Composite Example Structure



Composite Code

```
class NamedSizedItem
    @name = nil
    @size = 0
    def initialize(name)
        @name = name
    end
    def name()
        return @name
    end
    def size()
        return @size
    end
end

class NamedFile < NamedSizedItem
    def initialize(name)
        @name = name
        @size = 1
    end
end
```

```
class NamedFolder < NamedSizedItem
    @files = []
    def initialize(name)
        @name = name
        @files = []
    end
    def addFile(file)
        @files << file
    end
    def size()
        return @files.inject(0)
            { |res,elm| res + elm.size() }
    end
end

file1 = NamedFile.new("readme")
file2 = NamedFile.new("license")
file3 = NamedFile.new("a.out")
subFolder1 = NamedFolder.new("Sub1");
subFolder2 = NamedFolder.new("Sub2");
subFolder3 = NamedFolder.new("Sub3");
subFolder1.addFile(file1)
subFolder2.addFile(file2)
subFolder3.addFile(file3)
subFolder2.addFile(subFolder1)
subFolder3.addFile(subFolder2)
puts(subFolder3.size())
```

Composite Pattern Resources

- Ruby Best Practices, Issue 1.26: Structural Design Patterns http://ur1.ca/b5zwf by Gregory Brown
- Design Patterns in Ruby [Composite, Iterator, Command] http://ur1.ca/b5zrc by Ashwin Raghav
- Composite Pattern http://en.wikipedia.org/wiki/Composite_pattern
- Gamma, Erich; Richard Helm, Ralph Johnson, John M. Vlissides (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. pp. 395.
- Composite Pattern http://c2.com/cgi/wiki?CompositePattern

Command Pattern

- In OO we often have to do something. Sometimes Verbs turn into nouns.
- Intent: Encapsulate a request as an object, thereby letting you parametrize clients with different requests, queue or log requests, and support undo-able operations. -- http://c2.com/cgi/wiki?CommandPattern
- Separate Execution from invocation

Command Pattern Motivation

- Idea:
  - a class may want to issue a request without knowing anything about the operation being requested or the receiver object for the request
  - make request itself as a *command* object, so we can store it, run it, and pass it around

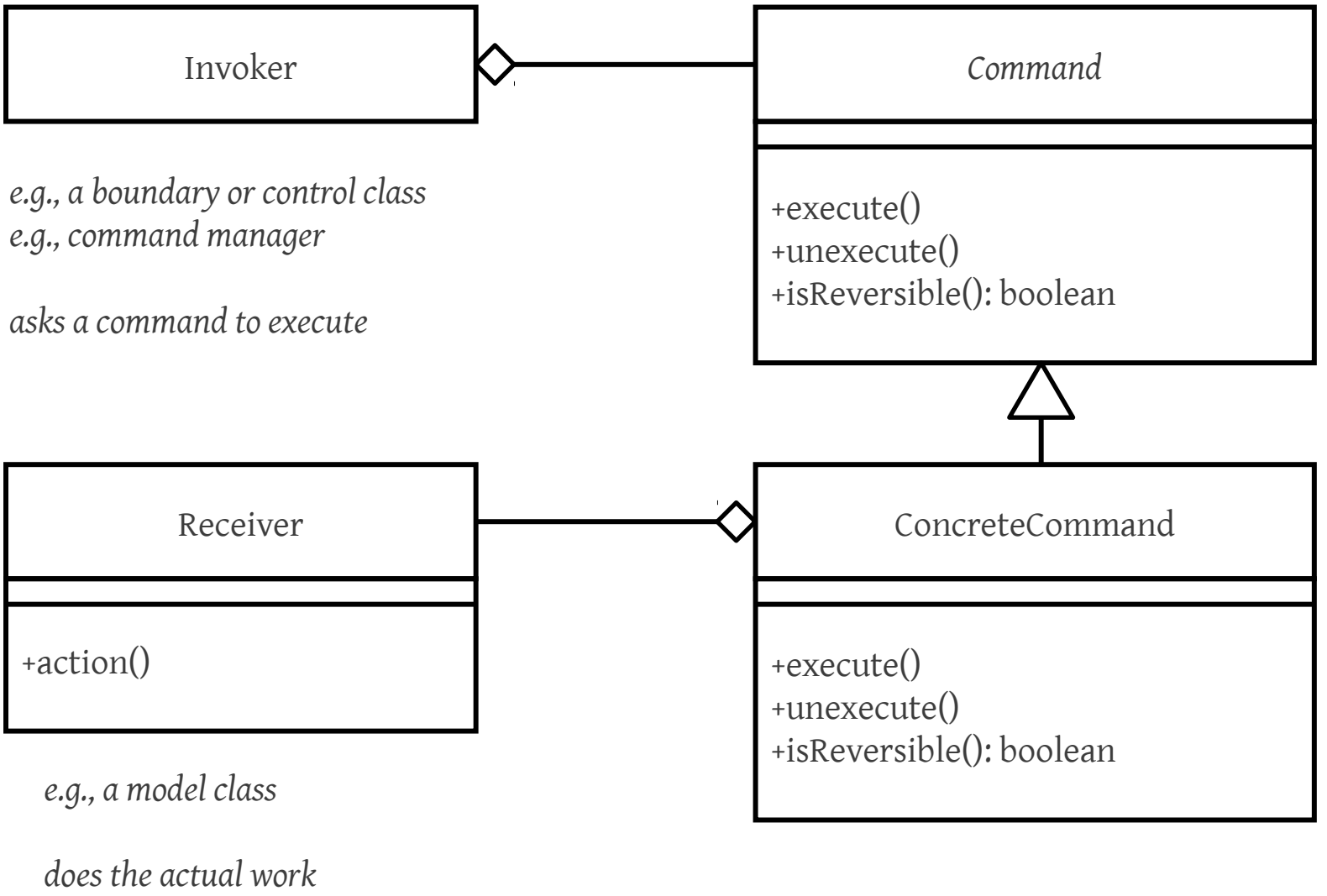
Command Pattern Uses

- Remove logic from menu bars or other UI components
  - Menu bar send a Command to the invoker instead of implementing the logic themselves.
- Allows definition of primitive operations that can be composed
- Enabling Macros

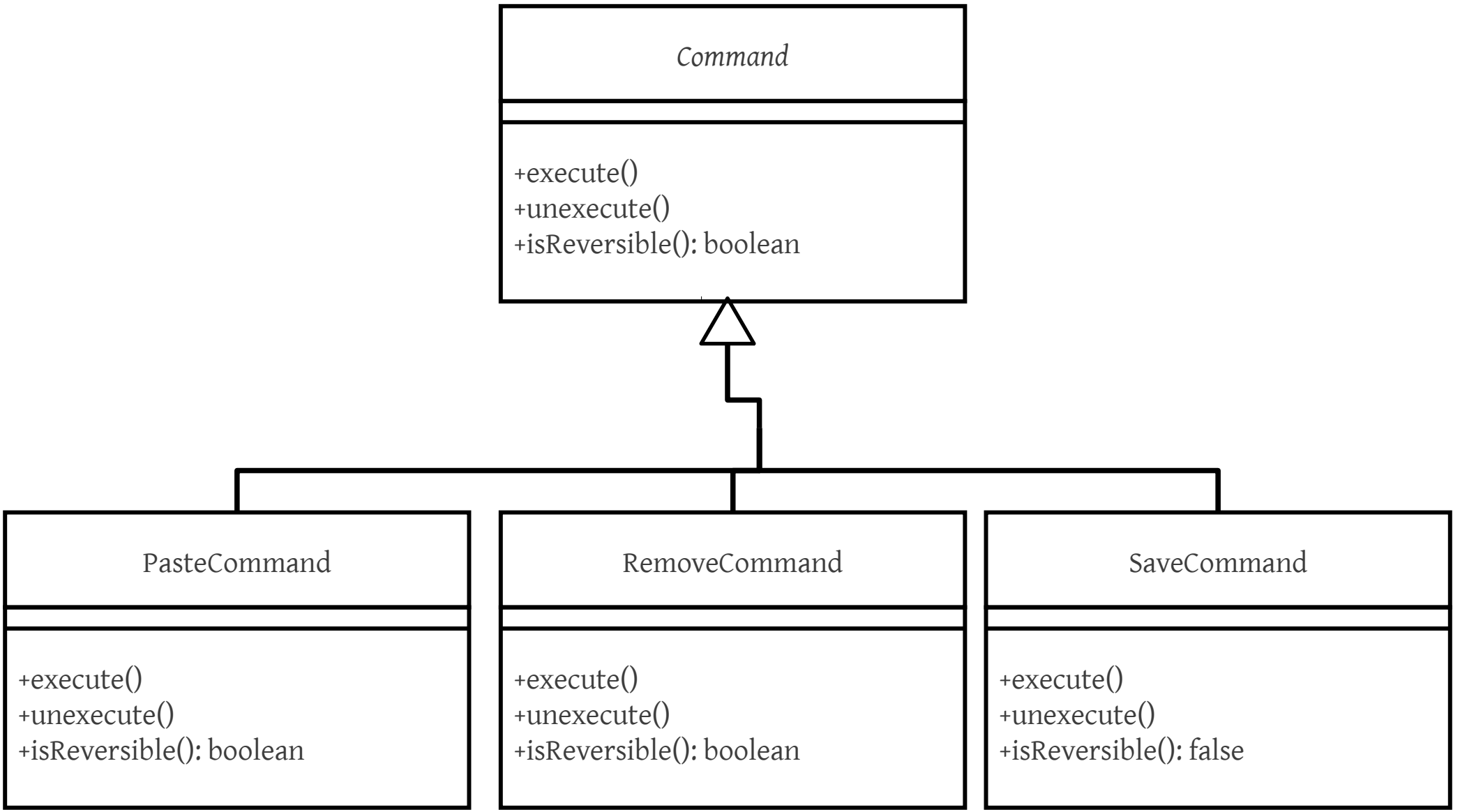
When to use Command?

- When you want to control the order of operations or when operations are executed.
  - Queuing, Parallelizing, Lazy Evaluation, Etc.
- When you want to support Undo/Redo or history of operations
- When you want a clear callback interface
  - Like function pointers
- Log operations
- When your system design has primitive operations

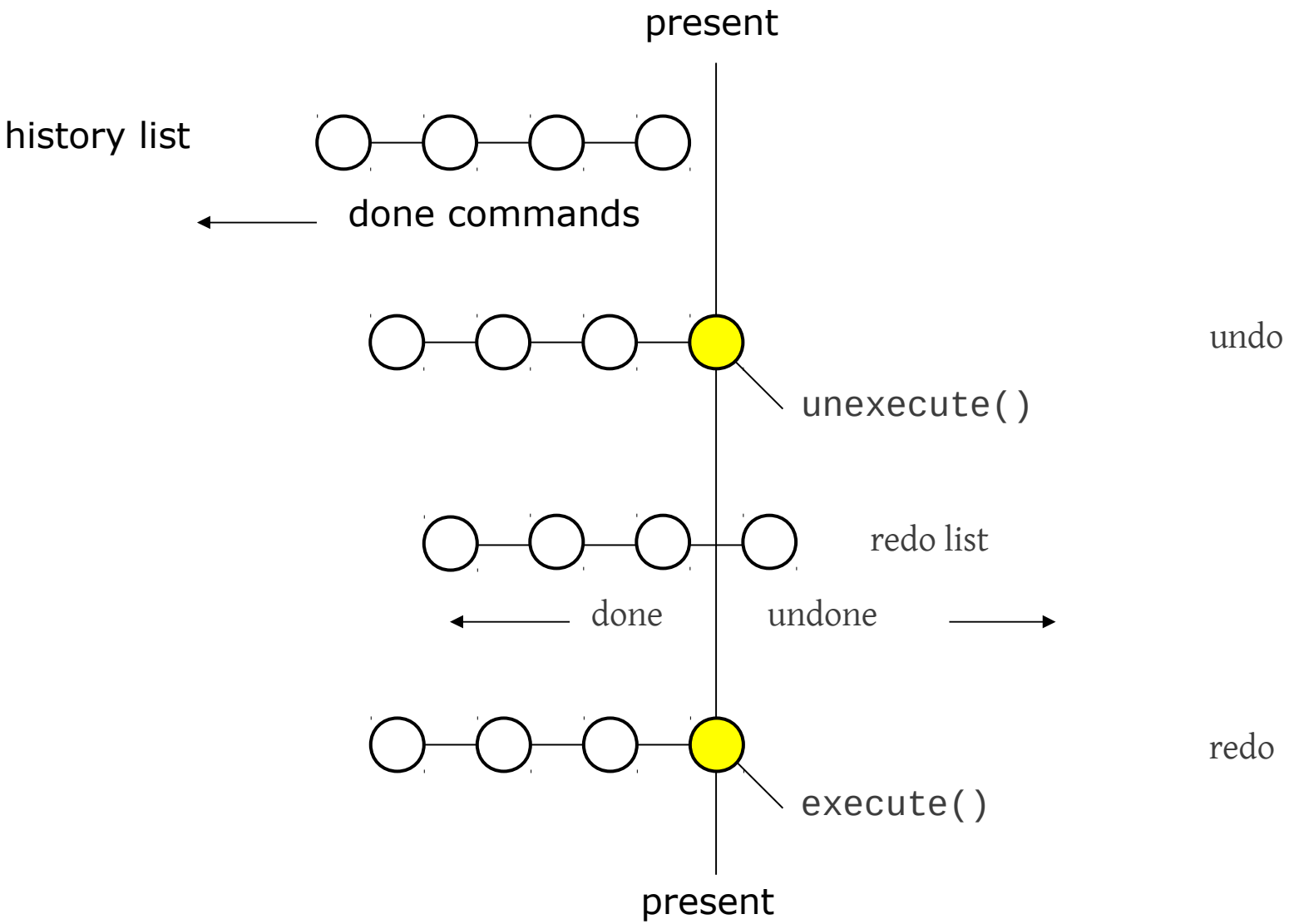
Command Pattern UML



Command Example



Command Pattern: Undo/Redo



Command Pattern Code Example 1/5

```
class Command
    def execute()
    end
    def unexecute()
    end
end
```

```
require 'singleton'

class Buffer
    include Singleton
    @buffer = []
    def initialize()
        @buffer = []
    end
    def insert(n, token)
        @buffer.insert( n, token )
    end
    def string()
        return @buffer.join( " " )
    end
    def remove(n)
     val = @buffer[n]
        @buffer.delete_at(n)
     return val
    end
end
```

Command Pattern Code Example 2/5

```
class PasteCommand < Command
    def initialize(n, token)
        @n = n
        @token = token
    end
    def execute()
       Buffer.instance.insert(@n, @token)
    end
    def unexecute()
        Buffer.instance.remove(@n)
    end
end
```

```
class RemoveCommand < Command
    @token = nil
    def initialize(n)
        @n = n
    end
    def execute()
        @token = Buffer.instance.remove(@n)
    end
    def unexecute()
        Buffer.instance.insert(@n, @token)
    end
end
```

Command Pattern Code Example 3/5

```
def example_driver()
    puts(Buffer.instance.string())
    actions = [
        PasteCommand.new(0,"Hello"),
        PasteCommand.new(1,"World"),
        PasteCommand.new(1,"Beautiful"),
        RemoveCommand.new(2),
        RemoveCommand.new(0),
    ]
    for action in actions
        action.execute()
        puts(Buffer.instance.string())
    end
    revactions = actions.reverse
    for action in revactions
        action.unexecute()
        puts(Buffer.instance.string())
    end
end
```

Command Pattern Code Example 4/5

```
class Invoker
    def initialize()
        @undoqueue = []
    end
    def do(x)
     x.execute()
        @undoqueue << x
    end
    def undo()
        x = @undoqueue.pop()
     x.unexecute() if x
    end
end
class BufferInvoker < Invoker
    def do(x)
        super(x)
        puts(Buffer.instance.string())
    end
    def undo()
        super()
        puts(Buffer.instance.string())
    end
end
```

```
def example_invoker()
    invoker = BufferInvoker.new()
    [
        PasteCommand.new(0,"Snakes"),
        PasteCommand.new(1,"Hiss"),
        PasteCommand.new(1,"Go"),
        RemoveCommand.new(2),
        RemoveCommand.new(0),
    ].each { |x| invoker.do( x ) }
    for i in (1..5)
        invoker.undo()
    end
end
```

Command Pattern Code Example 5/5

```
hindle1@eraser:~/projects/ruby-design-patterns$ ruby CommandExample.rb

Hello
Hello World
Hello Beautiful World
Hello Beautiful
Beautiful
Hello Beautiful
Hello Beautiful World
Hello World
Hello

Snakes
Snakes Hiss
Snakes Go Hiss
Snakes Go
Go
Snakes Go
Snakes Go Hiss
Snakes Hiss
Snakes
```

Command Pattern Consequences

- Results:
  - decouples the object that invokes the operation from the one that knows how to perform it
- Easy to add new commands or manipulate them because they are first-class objects

Command Pattern Resources

- Imperator By Wynn Netherland http://thechangelog.com/post/22650376319/imperator-command-patter
- Design Patterns in Ruby [Composite, Iterator, Command] http://ur1.ca/b5zrc by Ashwin Raghav
- Command Pattern http://en.wikipedia.org/wiki/Command_pattern
- Gamma, Erich; Richard Helm, Ralph Johnson, John M. Vlissides (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
- Command Pattern http://c2.com/cgi/wiki?CommandPattern