# UW Ruby Programming 110

# Winter 2015

# Michael Cohen

# Lecture 7

# Feb 19, 2015

# Lecture 7

1. Exceptions
2. Enumerable
3. Observable
4. Caching
5. Assignments

# Section 1

# Exceptions

## Section 1: Exceptions

# Exception class hierarchy

```
Exception
- NoMemoryError
- ScriptEror
  - SyntaxError
- SecurityError
- StandardError
  - ArgumentError
  - IndexError
  - IOError
  - NameError
    - NoMethodError
  - TypeError
```

# Defining your own subclass

```
class MyError < StandardError
end
```

# Raising an exception

```
raise 'An error occured'

raise MyError.new('A message about the error')
```

# Rescuing an exception

```
begin
  # ... do work
rescue
  # ... handle exception
ensure
  # ... cleanup
end
```

# Specifying exception class

```
begin
  # ... do work
rescue IOError
  # ... handle IOError exception
end
```

# Specifying exception class

```
begin
  # ... do work
rescue SecurityError
  # ... handle security error exception
rescue IOError
  # ... handle IOError exception
rescue
  # ... handle any other exception
end
```

# Grabbing the exception

```
begin
  # ... do work
rescue Exception => e
  puts e.message
  puts e.backtrace.inspect
end
```

# Section 2

# Enumerable

## Section 2: Enumerable

# Intro

**The `Enumerable` module provides a set of methods to traverse, search, sort and manipulate collections.**

## Section 2: Enumerable

# Methods

```
:all?, :any?, :chunk, :collect, :collect_concat, :count, :cycle,
:detect, :drop, :drop_while, :each_cons, :each_entry, :each_slice,
:each_with_index, :each_with_object, :entries, :find, :find_all,
:find_index, :first, :flat_map, :grep, :group_by, :include?, :inject,
:map, :max, :max_by, :member?, :min, :min_by, :minmax, :minmax_by,
:none?, :one?, :partition, :reduce, :reject, :reverse_each, :select,
:slice_before, :sort, :sort_by, :take, :take_while, :to_a, :zip
```

## Section 2: Enumerable

# Requirement

**To use `Enumerable`**
**a class must support an `each` method**

```
class MyEnumerableClass
  include Enumerable
  def each
    ...
  end
end
```

## Section 2: Enumerable

# Silly Example

```
class Roygbiv
  include Enumerable
  def each
    yield "red"
    yield "orange"
    yield "yellow"
    yield "green"
    yield "blue"
    yield "indigo"
    yield "violet"
  end
end
```

## Section 2: Enumerable

# Silly Example

```
roygbiv = Roygbiv.new
roygbiv.sort
roygbiv.each_with_index {|c,i| puts "#{i} => #{c}"}
roygbiv.max
```

## Section 2: Enumerable

# Real Example - Set

```
class Set
  def initialize
    @items = []
  end
  def length
    @items.length
  end
  def <<(item)
    @items << item
    @items.uniq!
    self
  end
  def empty?
    @items.empty?
  end
  def include?(item)
    @items.include? item
  end
  def each(&block)
    @items.each(&block)
  end
end
```

## Section 2: Enumerable

# Real Example - Set

```
class Set
  include Enumerable
  def initialize
    @items = []
  end
  def length
    @items.length
  end
  def <<(item)
    @items << item
    @items.uniq!
    self
  end
  def empty?
    @items.empty?
  end
  def each(&block)
    @items.each(&block)
  end
end
```

# Section 3

# Observable

# Purpose

**Observable provides a simple mechanism for one object to inform a set of interested third-party objects when its state changes.**

## Section 3: Observable

# API

`add_observer(obj)`

**Add obj as an observer on this object. obj will now receive notifications.**

## Section 3: Observable

# API

`delete_observer(obj)`

**Delete obj as an observer on this object. It will no longer receive notifications.**

`delete_observers`

**Delete all observers associated with this object.**

## Section 3: Observable

# API

`count_observers`

**Return the count of observers associated with this object.**

## Section 3: Observable

# API

`changed(newState=true)`

**Set the changed state of this object. Notifications will be sent only if the changed state is true.**

`changed?`

**Query the changed state of this object.**

## Section 3: Observable

# API

`notify_observers(*args)`

**If this object's changed state is true, invoke the update method in each currently associated observer in turn, passing it the given arguments. The changed state is then set to false.**

## Section 3: Observable

# Example: Thermometer

```
class Thermometer
  include Observable

  def initialize(temp)
    @temp = temp
  end

  def up
    @temp += 1
    notify_observers @temp
  end

  def down
    @temp -= 1
    notify_observers @temp
  end
end
```

## Section 3: Observable

# Example: Thermometer

```
class ThermometerControl
  def initialize(thermometer)
    @thermometer = thermometer
  end

  def click_up
    @thermometer.up
  end

  def click_down
    @thermometer.down
  end
end
```

## Section 3: Observable

# Example: Thermometer

```
class ThermometerDisplay
  def initialize(thermometer)
    thermometer.add_observer self
  end

  def render(temp)
    puts temp
  end

  def update(temp)
    render temp
  end
end
```

## Section 3: Observable

# Example: Thermometer

```
class HeatDisplay
  def initialize(thermometer)
    thermometer.add_observer self
  end

  @@colors = %w(white blue green yellow orange red)
  def render(temp)
    bucket = [0, [temp, 100].min].max / 20
    puts @@colors[bucket]
  end

  def update(temp)
    render temp
  end
end
```

# Example

```
t = Thermometer.new 68
tc = ThermometerControl.new t
td = ThermometerDisplay.new t
hd = HeatDisplay.new t

tc.up
tc.down
```

# Section 4

# Caching

# Situation

```
def some_value
  # ... expensive calculation
end
```

# Add caching

```
def some_value
  # use instance var as cache:
  @some_value || @some_value = begin
    # ... expensive calculation
  end
end
```

# Rewrite

```
def some_value
  @some_value ||= begin
    # ... expensive calculation
  end
end
```

## Section 4: Caching

# Add timeout

```
def some_value
  # pseudo-code:
  # if cached and fresh
  #   @some_value
  # else
  #   @some_value = begin
  #     expensive calculation
  #   end
  # end
end
```

## Section 4: Caching

# Add timeout

```
def some_value
  # pseudo-code:
  # if @some_value and fresh
  #   @some_value
  # else
  #   @some_value = begin
  #     expensive calculation
  #   end
  # end
end
```

## Section 4: Caching

# Add timeout

```
def some_value
  # pseudo-code:
  # if @some_value && ((Time.now - @some_value_ts) < threshold)
  #   @some_value
  # else
  #   @some_value_ts = Time.now
  #   @some_value = begin
  #     expensive calculation
  #   end
  # end
end
```

## Section 4: Caching

# Add timeout

```ruby
def some_value_fresh?
  (Time.now - @some_value_ts) < threshold
end

def some_value
  if @some_value && some_value_fresh?
    @some_value
  else
    @some_value_ts = Time.now
    @some_value = begin
      # expensive calculation
    end
  end
end
```

## Section 4: Caching

# Metaprogramming

```
def some_value
   # expensive calculation
end
cache :some_value, 60
```

## Section 4: Caching

# Metaprogramming

```
def cache(value, threshold)
   # metaprogramming magic
end
```

## Section 4: Caching

# Cacheable

```
module Cacheable
  def cache(value, threshold)
    alias_method :"calc_#{value}", value
    define_method "#{value}_fresh?" do
      (Time.now - instance_variable_get("@#{value}_ts")) < threshold
    end
    define_method value do
      if instance_variable_get("@#{value}") && send("#{value}_fresh?")
        instance_variable_get("@#{value}")
      else
        instance_variable_set "@#{value}_ts", Time.now
        instance_variable_set "@#{value}", send("calc_#{value}")
      end
    end
  end
end
```

## Section 4: Caching

# Using Cacheable

```
class SomeClass
  extend Cacheable

  def some_value
    # expensive calculation
  end
  cache :some_value, 30
end
```

## Section 5

# Assignment #7

## Section 5: Assignment #7

# Problem 1: Observable

```
# implement Observable
# validate using MiniTest unit tests

module Assignment07
  module Observable
    def add_observer(obj)
      # your implementation here
    end
    def delete_observer(obj)
      # your implementation here
    end
    def delete_observers
      # your implementation here
    end
    def changed(new_state=true)
      # your implementation here
    end
    def changed?
      # your implementation here
    end
    def notify_observers(*args)
      # your implementation here
    end
  end
end
```