Resolución Primer Parcial de Programación Orientada a Objetos (72.33)

20/09/2018

Ejercicio 1

```ruby
class SlidingWindowIterator

 def initialize(elements, window_dim)
   raise ArgumentError, 'Collection missing' if elements.nil?
   @each = Enumerator.new do |aux|
     start_index = 0
     loop do
       raise StopIteration if start_index + window_dim > elements.length
       aux << elements[start_index, window_dim]
       start_index += 1
     end
   end
 end

 def each
   @each
 end

end
```

Otras alternativas a **aux << elements[start_index, window_dim]** propuestas por los alumnos son:

```ruby
aux << elements.first(start_index + window_dim).last(window_dim)

aux << elements.first(start_index + window_dim).drop(start_index)

aux << elements[start_index...start_index+window_dim]

aux << (start_index..start_index+window_dim-1).map { |i| elements[i] }
```

Ejercicio 2

```ruby
class PriorityQueue
 def initialize
   @queue = SortedSet.new
 end

 def enqueue(element, priority)
   @queue.add(PriorityQueueElement.new(element, priority))
 end

 def dequeue
   raise EmptyPriorityQueueError if empty?
   pq_elem = @queue.max
   @queue.delete(pq_elem)
   pq_elem.element
 end
```

```
 def empty?
   @queue.empty?
 end

 def size
   @queue.size
 end
end
```

```
class PriorityQueueElement
 def initialize(element, priority)
   @element = element
   @priority = priority
 end

 def element
   @element
 end

 def priority
   @priority
 end

 def ==(other)
   return false unless other.is_a?(PriorityQueueElement)
   @element == other.element && @priority == other.priority
 end

 def eql?(other)
   self.==(other)
 end

 def hash
   [@element, @priority].hash
 end

 def <=>(other)
   raise 'Error' unless other.is_a?(PriorityQueueElement)
   other.priority <=> @priority # Descendente
 end

 def inspect
   "{#{@element}, #{@priority}}"
 end

end
```

```
class EmptyPriorityQueueError < StandardError
 def message
   'La cola de prioridades está vacía'
 end
end
```

Ejercicio 3

```ruby
class SystemAccess
 def initialize
   raise 'Cannot instantiate abstract class'
 end

 protected def init(name, parent_folder)
   @name = name
   @authorized_members = []
   @parent_folder = parent_folder
 end

 def grant_access(user)
   @authorized_members.push(user)  # podria validar para no insertar repetidos
 end

 def access?(user)
   access = @authorized_members.include?(user)
   access ||= @parent_folder.access?(user) unless @parent_folder.nil?
   access
 end
end
```

```ruby
class SystemFolder < SystemAccess
 def initialize(name, parent_folder = nil)
   init(name, parent_folder)
 end

 def add(system_file_name)
   SystemFile.new(system_file_name, self)
 end
end
```

```ruby
class SystemFile < SystemAccess
 def initialize(name, parent_folder)
   init(name, parent_folder)
 end
end
```

```ruby
class User
 def initialize(name)
   @name = name
 end

 def ==(other)
   return false unless other.is_a?(User)
   @name == other.name
 end

 def name
   @name
 end
end
```