

Recuperatorio Primer Parcial de Programación Orientada a Objetos (72.33)

28/11/2019

Ejercicio 1	Ejercicio 2	Ejercicio 3	Nota	Firma Docente
/3	/2.5	/4.5		

- ❖ **Condición mínima de aprobación: SUMAR 5 PUNTOS.**
- ❖ Las soluciones que no se ajusten al paradigma OO, no serán aceptadas.
- ❖ Las soluciones que no se ajusten estrictamente al enunciado, no serán aceptadas.
- ❖ Puede entregarse en lápiz.
- ❖ No es necesario escribir las sentencias require.
- ❖ Además de las clases solicitadas se pueden agregar las que consideren necesarias.
- ❖ Escribir en cada hoja Nombre, Apellido, Legajo, Número de Hoja y Total Hojas entregadas.
- ❖ Resolver los ejercicios en hojas separadas.

Ejercicio 1

Se desea **modelar una lista simplemente encadenada**. Para ello deberá modelar las clases **Node** (que cuenta con un elemento y una referencia al nodo siguiente) y **LinkedList** (que al instanciarla recibe el nodo cabeza de la lista y **ofrece un método iterator para poder recorrerla de principio a fin**).

Al obtener el iterador **se indica si el mismo debe ser o no cíclico**. Un iterador cíclico debe permitir **recorrer infinitamente la lista**.

**Implementar las clases Node, LinkedList y todo lo necesario para que con el siguiente programa de prueba se obtenga la salida indicada.**

<pre>third_node = Node.new('Mundo', nil) second_node = Node.new('Hola', third_node) first_node = Node.new('ITBA', second_node)  list = LinkedList.new(first_node)  cyclic_list_iterator = list.iterator(true) p cyclic_list_iterator.next p cyclic_list_iterator.next p cyclic_list_iterator.next p cyclic_list_iterator.next puts '#####' p cyclic_list_iterator.take(5) puts '#####' non_cyclic_list_iterator = list.iterator p non_cyclic_list_iterator.next p non_cyclic_list_iterator.next p non_cyclic_list_iterator.next begin   non_cyclic_list_iterator.next rescue StopIteration =&gt; e   p e.message end puts '#####' p non_cyclic_list_iterator.take(5) puts '#####' empty_list = LinkedList.new(nil) begin   empty_list.iterator rescue ArgumentError =&gt; e   p e.message end puts '#####' p cyclic_list_iterator.next.is_a?(Node)</pre>	<pre>"ITBA" "Hola" "Mundo" "ITBA" ##### ["ITBA", "Hola", "Mundo", "ITBA", "Hola"] ##### "ITBA" "Hola" "Mundo" "iteration reached an end" ##### ["ITBA", "Hola", "Mundo"] ##### "Empty List" ##### false</pre>
--	---

Ejercicio 2

Dada la siguiente jerarquía de clases, con los métodos de instancia indicados para cada una, se cuenta con tres instancias homónimas a la clase a la cual pertenecen.

	m_1	m_2	m_3
A			
B			
C			

class A

def m\_1

7

end

def m\_2

self.m\_3

end

def m\_3

m\_1

end

end

class B < A

def m\_1

9

end

def m\_3

super

end

end

class C < B

def m\_1

super.m\_1

end

def m\_2

self.m\_2

end

def m\_3

m\_3

end

end

Completar el cuadro de doble entrada (clase y mensaje) indicando qué salida se obtiene al enviar cada uno de los mensajes a las instancias de cada una de las clases. En caso de obtenerse un error, indicar qué tipo de error se obtiene.

Ejercicio 3

Se cuenta con la clase **CellPhone** que modela un teléfono celular.

Se quiere implementar una **promoción** en la cual existen **grupos de celulares que comparten el crédito** (se considera que se consume una unidad de crédito por llamada). **Las llamadas entre celulares de un mismo grupo son gratuitas.**

Para esto se deben implementar las clases **FamilyCellPhone** y **FamilyCellPhoneGroup**. **FamilyCellPhone** modela al celular que puede pertenecer a un grupo y **FamilyCellPhoneGroup** modela al grupo de celulares que comparten el crédito.

**Implementar todo lo necesario para que con el siguiente programa de prueba se obtenga la salida indicada. La clase CellPhone no debe ser modificada.**

```
class CellPhone

  attr_reader :number

  def initialize(number)
    @number = number
  end

  def make_call(to_number)
    puts "#{@number} making call to #{to_number}"
  end

  def to_s
    "CellPhone #{@number}"
  end

end
```

<pre>group1 = FamilyCellPhoneGroup.new('Group 1', 2) # Dos créditos phone1 = FamilyCellPhone.new(group1, '111-111') phone3 = FamilyCellPhone.new(group1, '333-333') phone2 = FamilyCellPhone.new(group1, '222-222') phone1.make_call('222-222') # No se cobra porque es del grupo phone3.make_call('555-555') phone2.make_call('111-111') # No se cobra porque es del grupo group2 = FamilyCellPhoneGroup.new('Group 2', 2) # Dos créditos puts group1 # El grupo 1 tiene crédito para una llamada puts group2 # El grupo 2 tiene crédito para dos llamadas group1.load_credit(2) # Agrega crédito para dos llamadas más phone1.make_call('444-444') phone3.make_call('888-888') phone2.make_call('777-777') puts 'Group 1 members:' group1.members.sort.each {  cell_phone  puts "- #{cell_phone}\n" } begin   phone1.make_call('555-555') rescue NoCreditError =&gt; e   puts e.message end</pre>	<pre>111-111 making call to 222-222 333-333 making call to 555-555 222-222 making call to 111-111 Group 1 has 1 credit/s left Group 2 has 2 credit/s left 111-111 making call to 444-444 333-333 making call to 888-888 222-222 making call to 777-777 Group 1 members: - CellPhone 111-111 - CellPhone 222-222 - CellPhone 333-333 No credit available</pre>
---	---

2

