Nombre y Apellido: ..... N° Legajo: .....

Primer Parcial de Programación Orientada a Objetos (72.33)

18/09/2019

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

- **<u>Condición mínima de aprobación: SUMAR 5 PUNTOS.</u>**
- **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**).

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

```ruby
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)

list_iterator = list.iterator

p list_iterator.next
p list_iterator.next
p list_iterator.next
begin
 list_iterator.next
rescue StopIteration => e
 p e.message
end

puts '##########'

p list_iterator.take(2)

puts '##########'

other_iterator = list.iterator

p other_iterator.next

puts '##########'

p list_iterator.take(5)

puts '##########'

empty_list = LinkedList.new(nil)
begin
 empty_list.iterator
rescue ArgumentError => e
 p e.message
end
```

```
"ITBA"
"Hola"
"Mundo"
"iteration reached an end"
##########
["ITBA", "Hola"]
##########
"ITBA"
##########
["ITBA", "Hola", "Mundo"]
##########
"Empty List"
```

Ejercicio 2

Se desea modelar un conjunto de clases para un **sistema de reservación de cocheras de un estacionamiento**. El **estacionamiento** (***parking lot***) cuenta con **pisos**. Cada **piso** es identificado por una letra. No pueden haber dos pisos con la misma identificación. Cada **cochera** (*parking space*) pertenece a un piso y es identificada por un número. No pueden haber dos cocheras con la misma identificación en el mismo piso.

Al crear un estacionamiento éste se crea sin pisos (y por ende sin cocheras). Es responsabilidad del usuario dar de alta cada una de las cocheras de cada uno de los pisos con el método `add_parking_space`. Dando de alta una cochera se da de alta también al piso correspondiente si es que éste no existía.

Cada cochera puede estar o no reservada. El estacionamiento debe ofrecer métodos para reservar o cancelar una reserva de una cochera (métodos `park` y `unpark` respectivamente).

Ya cuenta con la implementación del método `information` de la clase `ParkingLot` que le indica la colección que deberá utilizar para almacenar la información del estacionamiento. Este método muestra el estado actual de todas las cocheras del estacionamiento.

```
class ParkingLot

...

 def information
   s = "Parking Lot #{@name}\n"
   @parking_spaces_by_level.keys.sort.each do |level|
     s += "Level #{level}\n"
     @parking_spaces_by_level[level].values.sort.each do |parking_space|
       s += "#{parking_space}\n"
     end
   end
   s
 end

end
```

Implementar todo lo necesario, agregando métodos a `ParkingLot` y/o creando clases nuevas para que, con el siguiente programa, se obtenga la salida indicada.

```
parking_lot = ParkingLot.new('EstacionARTE')
parking_lot.add_parking_space('A', 1030)
parking_lot.add_parking_space('A', 1000)
parking_lot.add_parking_space('A', 1001)
parking_lot.add_parking_space('B', 1001)
puts parking_lot.information
puts '##########'
begin
 parking_lot.park('Z',1001)
rescue ArgumentError => e
 puts e.message
end
puts '##########'
begin
 parking_lot.unpark('A',9999)
rescue ArgumentError => e
 puts e.message
end
puts '##########'
parking_lot.park('A',1001)
parking_lot.unpark('A',1001)
parking_lot.park('A',1001)
begin
 parking_lot.park('A',1001)
rescue ParkReservedSpaceError => e
 puts e.message
end
puts '##########'
begin
 parking_lot.unpark('B',1001)
rescue UnparkAvailableSpaceError => e
 puts e.message
end
puts '##########'
parking_lot.park('B',1001)
puts parking_lot.information
```

```
Parking Lot EstacionARTE
Level A
#1000: Available
#1001: Available
#1030: Available
Level B
#1001: Available
##########
Invalid Level
##########
Invalid Parking Space
##########
Cannot Park Reserved Parking Space
##########
Cannot Unpark Available Parking Space
##########
Parking Lot EstacionARTE
Level A
#1000: Available
#1001: Reserved
#1030: Available
Level B
#1001: Reserved
```

Ejercicio 3

Se cuenta con la clase **`Exam`** que modela la **hoja de inscripción de un examen de una materia**. Esta clase permite **inscribir y desinscribir alumnos**, y se define de la siguiente manera:

```ruby
class Exam
 def initialize(name)
   @name, @enrolled = name, []
 end

 def enroll(student)
   @enrolled << student
   puts "Enrolled #{student}"
 end

 def unenroll(student)
   @enrolled.delete(student)
   puts "Unenrolled #{student}"
 end

 def enrolled?(student)
   @enrolled.include?(student)
 end

 def enrolled_count
   @enrolled.length
 end

 def enrolled_students
   @enrolled
 end
end
```

Ahora se desean modelar **dos hojas de inscripción más**:

- **<u>Hoja de inscripción única (`UniqueExam`):</u>** Al momento de inscribir un alumno, se debe verificar que el mismo no se encuentre ya inscripto en el mismo.

- **<u>Hoja de inscripción única con cupo máximo (`LimitedExam`)</u>**. Al momento de inscribir un alumno, además de verificar que el mismo no se encuentre ya inscripto en el mismo, se debe controlar que aún quede cupo disponible.
  En caso de que no hubiera cupo, se debe registrar que el alumno intentó inscribirse para otorgarle la próxima vacante disponible.
  Cuando se desinscribe un alumno, en caso de que hubieran alumnos esperando que se libere alguna vacante, se debe inscribir automáticamente al primero que lo haya intentado.

Implementar todo lo necesario <u>(sin modificar la clase Exam)</u> para que, con el siguiente programa, se obtenga la salida indicada.

```ruby
exam = Exam.new('Primer Parcial PI')
exam.enroll('Matias')
exam.enroll('Matias')
exam.enroll('Natalia')
puts "Enrolled Students: #{exam.enrolled_students}"
exam.unenroll('Matias')
puts "Enrolled Students: #{exam.enrolled_students}"
puts '##########'
unique_exam = UniqueExam.new('Primer Parcial POO')
unique_exam.enroll('Matias')
unique_exam.enroll('Matias')
unique_exam.enroll('Natalia')
puts "Enrolled Students: #{unique_exam.enrolled_students}"
unique_exam.unenroll('Matias')
puts "Enrolled Students: #{unique_exam.enrolled_students}"
puts '##########'
limited_exam = LimitedExam.new('TPE POD', 2)
limited_exam.enroll('Matias')
limited_exam.enroll('Matias')
limited_exam.enroll('Natalia')
limited_exam.enroll('Solange')
limited_exam.enroll('Jose')
limited_exam.enroll('Micaela')
puts "Enrolled Students:
#{limited_exam.enrolled_students}"
puts "Pending Students: #{limited_exam.pending_students}"
limited_exam.unenroll('Matias')
puts "Enrolled Students:
#{limited_exam.enrolled_students}"
puts "Pending Students: #{limited_exam.pending_students}"
limited_exam.unenroll('Jose')
limited_exam.unenroll('Natalia')
puts "Enrolled Students:
#{limited_exam.enrolled_students}"
puts "Pending Students: #{limited_exam.pending_students}"
```

```
Enrolled Matias
Enrolled Matias
Enrolled Natalia
Enrolled Students: ["Matias",
"Matias", "Natalia"]
Unenrolled Matias
Enrolled Students: ["Natalia"]
##########
Enrolled Matias
Enrolled Natalia
Enrolled Students: ["Matias",
"Natalia"]
Unenrolled Matias
Enrolled Students: ["Natalia"]
##########
Enrolled Matias
Enrolled Natalia
Enrolled Students: ["Matias",
"Natalia"]
Pending Students: ["Solange",
"Jose", "Micaela"]
Unenrolled Matias
Enrolled Solange
Enrolled Students: ["Natalia",
"Solange"]
Pending Students: ["Jose",
"Micaela"]
Unenrolled Natalia
Enrolled Micaela
Enrolled Students: ["Solange",
"Micaela"]
Pending Students: []
```