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

Recuperatorio del Primer Parcial de Programación Orientada a Objetos (72.33)

22/11/2018

Ejercicio 1	*Ejercicio 2*	*Ejercicio 3*	*Nota*	*Firma Docente*
/3	/2	/5		

- **<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.**

Ejercicio 1

Un **iterador cíclico de ventana deslizante** es un iterador que, a partir de una colección, permite recorrerla infinitamente obteniendo subconjuntos de un tamaño fijo (número entero que se recibe como parámetro) de forma que la ventana se desplaza recorriendo todos los elementos de la colección.

La clase **`CyclicSlidingWindowIterator`** recibe en su constructor la colección a iterar y el tamaño de la ventana, y cuenta con un método `each` para poder acceder al iterador cíclico de ventana deslizante.

Implementar todo lo necesario para que el siguiente programa de prueba imprima la salida indicada

```ruby
elements = ['Instituto', 'Tecnológico', 'de',
'Buenos', 'Aires']

my_iterator =
CyclicSlidingWindowIterator.new(elements, 3)
p my_iterator.each.next
p my_iterator.each.next
p my_iterator.each.next
p my_iterator.each.next
p my_iterator.each.next
p my_iterator.each.next
p my_iterator.each.next

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

p my_iterator.each.take(4)

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

begin
 CyclicSlidingWindowIterator.new(nil, 1)
rescue ArgumentError => e
 puts e.message
end

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

begin
 CyclicSlidingWindowIterator.new(elements, 0)
rescue ArgumentError => e
 puts e.message
end
```

```
["Instituto", "Tecnológico", "de"]
["Tecnológico", "de", "Buenos"]
["de", "Buenos", "Aires"]
["Buenos", "Aires", "Instituto"]
["Aires", "Instituto", "Tecnológico"]
["Instituto", "Tecnológico", "de"]
["Tecnológico", "de", "Buenos"]
##########
[["Instituto", "Tecnológico", "de"],
["Tecnológico", "de", "Buenos"], ["de",
"Buenos", "Aires"], ["Buenos", "Aires",
"Instituto"]]
##########
Collection missing
##########
Invalid argument
```

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			

```ruby
class A
 def m_1
   self.m_3
 end

 def m_2
   0
 end

 def m_3
   m_2
 end
end
```

```ruby
class B < A
 def m_1
   super
 end

 def m_2
   m_2
 end

 def m_3
   3
 end
end
```

```ruby
class C < B
 def m_2
   2
 end

 def m_3
   super.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 desea modelar un sistema de compra de pasajes aéreos. Dicho sistema debe consultar la información de los vuelos utilizando las clases **`FlightCatalog`** y **`Flight`** que se muestran a continuación:

```ruby
class FlightCatalog

 def initialize
   @flights = Hash.new
 end

 def add_flight(flight)
   @flights[flight.code] = flight
 end

 def get_flight(flight_code)
   @flights[flight_code]
 end

end
```

```ruby
class Flight

 attr_reader :code, :airline, :price, :miles

 def initialize(code, airline, price, miles)
   @code = code
   @airline = airline
   @price = price
   @miles = miles
 end

end
```

Se desea contar con la clase **`FlightOperator`** que modela a un operador de pasajes aéreos. Dicha clase permite realizar la compra de un pasaje acumulando las millas que este otorga.

Existen tres categorías de **miembros** del programa de millas:

- **Standard**: Acumula las millas del vuelo y permite acumular hasta 1000 millas.
- **Gold**: Acumula un **10%** más de millas que la categoría Standard (multiplicador 1.10) y permite acumular hasta 2000 millas.
- **Platinum**: Acumula un **25%** más de millas que la categoría Standard (multiplicador 1.25) y no tiene límite de acumulación de millas.

Así cada miembro pertenece a una de estas tres categorías y la misma sirve para todas las aerolíneas.

El método `buy_flight` retorna el **precio del vuelo**, el cual puede ser:

- **El precio original** si no cuenta con la totalidad de millas necesarias para canjearlo. Se deberán además sumarle las millas correspondientes al miembro para dicha aerolínea, con los criterios de su categoría.
- **Cero**, si cuenta con la totalidad de millas necesarias para canjearlo. Se deberán además restarle las millas al miembro para dicha aerolínea.

La clase `FlightOperator` ofrece además el método `miles_status` que dado un miembro y una aerolínea, lista el millaje acumulado en dicha aerolínea.

Implementar todo lo necesario y completar los ….. para que con el siguiente programa de prueba

```
flight_catalog = FlightCatalog.new

flight_catalog.add_flight(Flight.new('LA1','LATAM',200,600))
flight_catalog.add_flight(Flight.new('LA2','LATAM',400,3000))
flight_catalog.add_flight(Flight.new('LA3','LATAM',40,200))
flight_catalog.add_flight(Flight.new('UA1','United',3500,5000))

flight_operator = FlightOperator.new(flight_catalog)

juan = Member.new('Juan', .....................) # Juan tiene Categoría Gold

puts flight_operator.miles_status(juan, 'United') # 0
puts flight_operator.miles_status(juan, 'LATAM') # 0

puts flight_operator.buy_flight('UA1', juan) # 3500

puts flight_operator.miles_status(juan, 'United') # 2000 por el tope de categoría
puts flight_operator.miles_status(juan, 'LATAM') # 0

malena = Member.new('Malena', .....................) # Malena tiene Categoría Platinum

puts flight_operator.miles_status(malena, 'United') # No registra millas
puts flight_operator.miles_status(malena, 'LATAM') # No registra millas

puts flight_operator.buy_flight('UA1', malena) # 3500
puts flight_operator.buy_flight('LA1', malena) # 200

puts flight_operator.miles_status(malena, 'United') # 6250.0 = 5000 * 1.25
puts flight_operator.miles_status(malena, 'LATAM') # 750.0 = 600 * 1.25

puts flight_operator.buy_flight('LA2', malena) # 400

puts flight_operator.miles_status(malena, 'United') # 6250.0
puts flight_operator.miles_status(malena, 'LATAM') # 4500.0 = 750.0 + 3000 * 1.25

puts flight_operator.buy_flight('LA3', malena) # 0 porque tenía 200 millas

puts flight_operator.miles_status(malena, 'United')  # 6250.0
puts flight_operator.miles_status(malena, 'LATAM') # 4300.0 = 4500.0 - 200.0
```

se obtenga la siguiente salida:

```
Juan no registra millas acumuladas en United
Juan no registra millas acumuladas en LATAM
3500
Millas de Juan en United: 2000
Millas de Juan en LATAM: 0
Malena no registra millas acumuladas en United
Malena no registra millas acumuladas en LATAM
3500
200
Millas de Malena en United: 6250.0
Millas de Malena en LATAM: 750.0
400
Millas de Malena en United: 6250.0
Millas de Malena en LATAM: 4500.0
0
Millas de Malena en United: 6250.0
Millas de Malena en LATAM: 4300.0
```