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

Primer Parcial de Programación Orientada a Objetos (72.33)

20/09/2018

Ejercicio 1	*Ejercicio 2*	*Ejercicio 3*	*Nota*	*Firma Docente*
/3	**/3**	**/4**		

- **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.**

Ejercicio 1

Un **iterador de ventana deslizante** es un iterador que, a partir de una colección, permite recorrerla obteniendo subconjuntos de un tamaño fijo (número entero que se recibe como parámetro) de forma que la ventana se desplaza hasta que alcance el final de la colección.

La clase **`SlidingWindowIterator`** 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 de ventana deslizante.

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

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

my_iterator = SlidingWindowIterator.new(elements, 3)
p my_iterator.each.next
p my_iterator.each.next
p my_iterator.each.next
begin
p my_iterator.each.next
rescue StopIteration => e
 puts e.message
end

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

p my_iterator.each.take(4)

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

begin
 SlidingWindowIterator.new(elements, 10).each.next
rescue StopIteration => e
 puts e.message
end

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

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

```
["Instituto", "Tecnológico", "de"]
["Tecnológico", "de", "Buenos"]
["de", "Buenos", "Aires"]
iteration reached an end
##########
[["Instituto", "Tecnológico", "de"],
["Tecnológico", "de", "Buenos"], ["de",
"Buenos", "Aires"]]
##########
iteration reached an end
##########
Collection missing
```

Ejercicio 2

Una **cola de prioridad** es una colección ordenada. El orden de los elementos depende del orden de inserción, pero además de un número que establece la prioridad de ese elemento sobre el resto. Dados dos elementos de la colección, si tienen la misma prioridad, el que se insertó primero está antes que el segundo. Si la prioridad de ambos difiere, quien tenga menor prioridad está primero. Al sacar un elemento siempre se saca el primero.

Se cuenta con la clase `PriorityQueue`:

```
require 'set'

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
end
```

Implementar todo lo necesario (creando nuevas clases y/o agregando métodos a `PriorityQueue`) para que el siguiente programa de prueba imprima la salida indicada

```
pq = PriorityQueue.new
pq.enqueue('Hola', 2)
pq.enqueue('Chau', 5)
pq.enqueue('Mundo', 2)
pq.enqueue('ZZZZ', 1)
pq.enqueue('Java', 1)
pq.enqueue('Java', 5)
pq.enqueue('Algo', 1)
puts pq.dequeue
pq.enqueue('Otro', 1)
puts pq.size
p pq
puts pq.dequeue until pq.empty?
p pq
begin
 puts pq.dequeue
rescue EmptyPriorityQueueError => e
 puts e.message
end
```

```
ZZZZ
7
#<PriorityQueue:0x00007ff946086d70
@queue=#<SortedSet: {{Chau, 5}, {Java, 5},
{Hola, 2}, {Mundo, 2}, {Java, 1}, {Algo, 1},
{Otro, 1}}>>
Java
Algo
Otro
Hola
Mundo
Chau
Java
#<PriorityQueue:0x00007ff946086d70
@queue=#<SortedSet: {}>>
La cola de prioridades está vacía
```

Ejercicio 3

Un sistema de archivos está compuesto por archivos y carpetas. La clase `SystemFile` representa un archivo del sistema (por ejemplo "main.c") que pertenece a una carpeta de tipo `SystemFolder` (por ejemplo "pi"). El constructor de `SystemFolder` puede aceptar como segundo parámetro otro `SystemFolder`, indicando que es una subcarpeta de éste.

La clase `SystemFolder` ofrece un método `add` que permite crear un archivo de cierto nombre de forma que este nuevo archivo pertenezca a la carpeta en cuestión, retornando la instancia recién creada.

Por defecto, el acceso a los archivos y carpetas es restringido. Para poder acceder a uno de ellos se requiere un permiso. Se desea entonces contar con una forma de otorgar permisos para que los usuarios del sistema puedan acceder a los archivos y carpetas para los cuales fueron autorizados.

Para simplificar el uso del sistema, se le puede otorgar a un usuario el permiso sobre una carpeta y, en ese caso, podrá acceder además a todos los archivos y carpetas que pertenezcan a esa carpeta.

Implementar todo lo necesario para que, con el siguiente programa de prueba, se imprima la salida indicada en los comentarios.

```
user1 = User.new('User 1')
user2 = User.new('User 2')

pi_folder = SystemFolder.new('pi') # se crea la carpeta pi en la raíz
main_file = pi_folder.add('main.c')

pi_folder.grant_access(user1)
main_file.grant_access(user2)

puts pi_folder.access?(user1) # true
puts main_file.access?(user1) # true
puts pi_folder.access?(user2) # false
puts main_file.access?(user2) # true

getnumc_file = pi_folder.add('getnum.c')
getnumh_file = pi_folder.add('getnum.h')

puts pi_folder.access?(user1) # true
puts main_file.access?(user1) # true
puts getnumc_file.access?(user1) # true
puts getnumh_file.access?(user1) # true
puts pi_folder.access?(user2) # false
puts main_file.access?(user2) # true
puts getnumc_file.access?(user2) # false
puts getnumh_file.access?(user2) # false

pi_docs_folder = SystemFolder.new('docs', pi_folder) # docs es subcarpeta de pi
doc_file = pi_docs_folder.add('README')

puts pi_docs_folder.access?(user2) # false
puts pi_docs_folder.access?(user1) # true
puts doc_file.access?(user2) # false
puts doc_file.access?(user1) # true

pi_docs_folder.grant_access(user2)

puts doc_file.access?(user2) # true
```