UNLP. Facultad de Informática.

Algoritmos y Estructuras de Datos 2007. **Trabajo Práctico 3**

Encapsulamiento y Abstracción

Objetivos:

- Acostumbrarse a utilizar las operaciones para acceder a las estructuras de datos y no violar el encapsulamiento.
- Acostumbrarse a reutilizar el código y no volver codificar desde cero.
- Implementar y usar de las estructuras de datos: lista, pila y cola

Ejercicio 1

Considere la siguiente especificación de la lista posicional:

begin() // Se prepara para iterar los elementos de la lista.
next() // Avanza al próximo elemento de la lista.
end() // Determina si llegó o no al final de la lista.
get() // Retorna el elemento actual.
get(int pos) // Retorna el elemento de la posición indicada
add(String elem, int pos) // Agrega el elemento en la posición indicada y retorna true:si pudo agregar y false; si no pudo agregar.
remove() // Elimina el elemento actual.
remove(int pos) // Elimina el elemento de la posición indicada
isEmpty() // Retorna true si la lista está vacía, false en caso contrario
includes (String elem) // Retorna true si elem está contenido en la lista, false en caso contrario.
size() // Retorna la longitud de la lista

Implemente en JAVA (pase por máquina) la clase **ListaPosicionalDeStrings** de acuerdo a la especificación dada.
a) Usando arreglos de longitud fija.
b) Usando una estructura recursiva. En este caso la clase Lista tendrá una referencia al elemento inicial y actual. Luego, se definirá otra clase recursiva que contenga el elemento y que referencie a una instancia similar a sí misma.

Ejercicio 2

Sea la siguiente especificación de la Pila:

push (String elem) // Agrega elem a la pila
pop // Elimina y devuelve el elemento en el tope de la pila.

a) Completar la especificación de la Pila con las operaciones que considere necesarias.
b) Implemente en JAVA (pase por máquina) la clase **PilaDeStrings** (utilizando la clase ListaPosicionalDeStrings) de acuerdo a la especificación dada.
c) Escriba una clase llamada TestPila para ejecutar el siguiente código:

```
PilaDeStrings p1, p2;
String valor2;
p1=new PilaDeStrings();
p1.push("1");
p1.push("2");
p2=p1;
valor2 = p2.pop();
System.out.println("El valor del tope de la pila p1 es:  " + p1.pop());
```

c) ¿Qué valor imprime? ¿Qué conclusión saca?

UNLP. Facultad de Informática.

Algoritmos y Estructuras de Datos 2007. **Trabajo Práctico 3**

Ejercicio 3

Sea la siguiente especificación de Cola

push (String elem) // Agrega elem a la cola.
pop // Elimina y devuelve el primer elemento de la cola.

a) Completar la especificación de la Cola con las operaciones que considere necesarias.
b) Implemente en JAVA (pase por máquina) la clase **ColaDeStrings** (utilizando la clase ListaPosicionalDeStrings) de acuerdo a la especificación dada.

Ejercicio 4

Considere un string de caracteres S, el cual comprende únicamente los caracteres: (,),[,],{,}. Decimos que S está balanceado si tiene alguna de las siguientes formas:
S = '' S es el string de longitud cero.
S = '(T)'
S = '[T]'
S = '{T}'
S = 'TU'
Donde ambos T y U son strings balanceados. Por ejemplo, '{( ) [( )] }' está balanceado, pero ' ([ )]' no lo está.

Implemente una clase llamada **TestBalanceo** (pase por máquina), cuyo objetivo es determinar si un string dado está balanceado. El string a verificar es un parámetro de entrada (no es un dato predefinido).

Ejercicio 5

Dada la siguiente especificación de la clase lista:

begin() // Se prepara para iterar los elementos de la lista.
next() // Avanza al próximo elemento de la lista.
end() // Determina si llegó o no al final de la lista.
get() // Retorna el elemento actual.
add(String elem) // Agrega el elemento en la posición actual, si no hay actual se agrega al principio.
remove() // Elimina el elemento actual.
remove(String elem) // Elimina la primer ocurrencia de elem
isEmpty() // Retorna true si la lista está vacía, false en caso contrario
includes (String elem) // Retorna true si elem está contenido en la lista, false en cc.
size() // Retorna la longitud de la lista

Note que las operaciones son un subconjunto del ejercicio 1 y se agregaron mensajes add y remove con distintos parámetros.

a) Implemente en JAVA (pase por máquina) la clase **ListaDeStrings** de acuerdo a la especificación dada.
b) Cuelgue la clase **ListaPosicionalDeStrings** del ejercicio 1 de la clase **ListaDeStrings**. Elimine de ListaPosicionalDeStrings los mensajes que no necesita puesto que los hereda. Verifique que ListaPosicionalDeStrings sigue funcionando correctamente.
c) Escribir una clase llamada **ListaOrdenadaDeStrings como subclase de ListaDeStrings**. Un objeto ListaOrdenadaDeStrings mantiene todos los elementos de la lista ordenados de acuerdo al orden alfabético.

Sobrescribir el método **add(String elem)** para que al agregar elem a la lista conserve el orden.

d) Escribir una clase llamada **TestListaOrdenada** que cree un objeto **ListaOrdenadaDeStrings**, le agregue 10 nombres de materias de la Facultad usando el método add(elem) y luego la recorra e imprima en la consola los nombres de las materias.
e) Comente el método **add(String elem)** de la clase **ListaOrdenadaDeStrings.** Vuelva a ejecutar la clase **TestListaOrdenada.** ¿Los nombres de las materias se imprimen ordenados? ¿qué método add() se ejecutó?
f) Sobrescriba los métodos **public boolean equals(Object)** y **public String toString()** de la clase Object en la clase **ListaOrdenadaDeStrings**. El método equals() devuelve true si las dos listas que se comparan contienen los mismos valores y en el mismo orden y false en otro caso; el método toString() convierte la lista a String, generando una secuencia de Strings separados por comas. ¿Podrían estos métodos estar en la superclase?
g) Modifique **TestListaOrdenada** y cree otro objeto **ListaOrdenadaDeStrings**, agréguele los mismos 10 elementos que a la lista creada en d). Luego, compárelas usando el método equals(). Ejecute **TestListaOrdenada.** ¿Qué resultado obtuvo? Luego, comente el método equals() de **ListaOrdenadaDeStrings**. Vuelva a ejecutar **TestListaOrdenada.** ¿Qué resultado obtuvo? ¿Cuál es la conclusión?.
h) Modifique **TestListaOrdenada** para que imprima las dos listas invocando a System.out.println() una vez por cada lista. ¿qué resultado obtuvo? Luego comente el método toString() de **ListaOrdenadaDeStrings**. Vuelva a ejecutar **TestListaOrdenada.** ¿Qué resultado obtuvo? ¿Cuál es la conclusión?

Ejecicio 6

a) Defina una clase EstructuraDeDato con las siguientes operaciones:

add (Object elem)// agrega elem en la posición actual, si no hay actual se agrega al principio.
remove (Object elem) // Elimina la primer ocurrencia de elem.
isEmpty() // Retorna true si la lista está vacía, false en caso contrario
includes (Object elem) // Retorna true si elem está contenido en la lista, false en caso contrario.
size() // Retorna la cantidad de elementos de la estructura

Note que la estructura trabaja con Object y no Strings.

b) Implementar la clase Lista y sus subclases en el paquete **estructurasdedatos.colecciones** como subclases de la clase definida en a)

Ejercicio 7

Escriba la clase **PilaMin** como subclase de la clase Pila. Las operaciones deben tener complejidad **O(1)**

Especificación de los métodos de PilaMin
min // Retorna, sin eliminar, el elemento menor de la pila.
push: elem // Agrega elem a la pila (sobreescribe el de Pila)
pop // Elimina y devuelve el elemento en el tope de la pila (sobreescribe el de Pila)