



Taller 7

Fecha: Junio de 2021

Profesor: Fray León Osorio Rivera

Competencia a evaluar: Aplicar los conceptos básicos de la orientación a objetos y las estructuras de datos en el desarrollo de una aplicación con interfaz gráfica de usuario.

NOTAS:

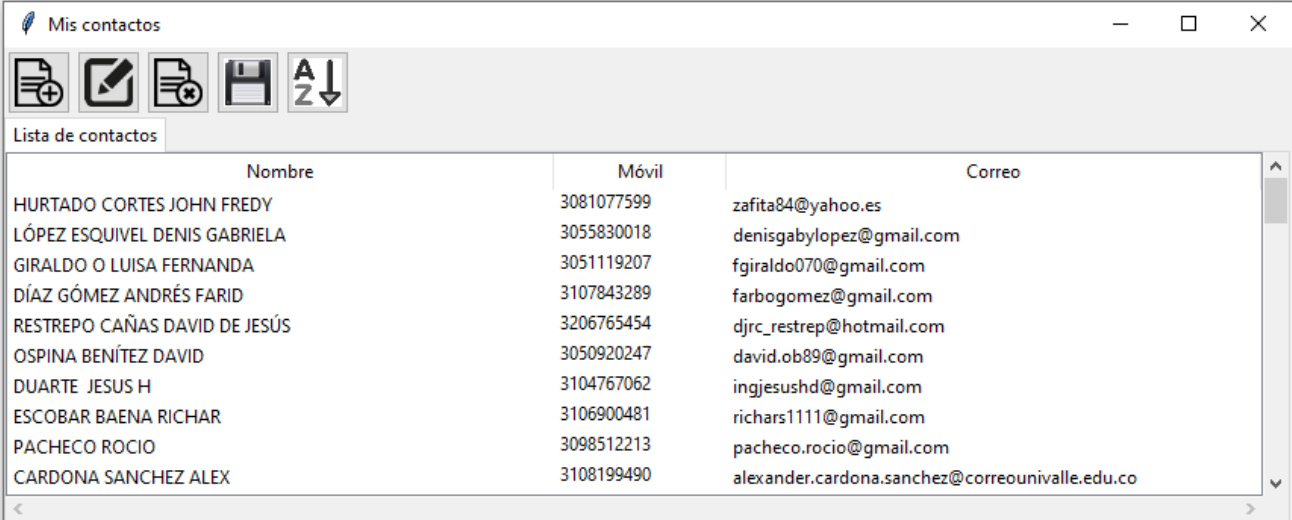
- Este taller se debe hacer como preparación para evaluaciones. En ningún caso representará una calificación.
- Los primeros ejercicios se entregan resueltos como ejemplo para el desarrollo de los demás

Elaborar el diagrama de clases básico (sin las clases correspondientes a la interface de usuario según el lenguaje de implementación) y la respectiva aplicación en un lenguaje orientado a objetos para los siguientes enunciados:

1. Implementar un directorio de contactos personal que permita realizar las operaciones básicas de actualización y consulta:
 - Agregar
 - Modificar
 - Eliminar
 - Listar
 - Ordenar

La información deberá almacenarse en un archivo plano y cargarse en memoria en una lista ligada.

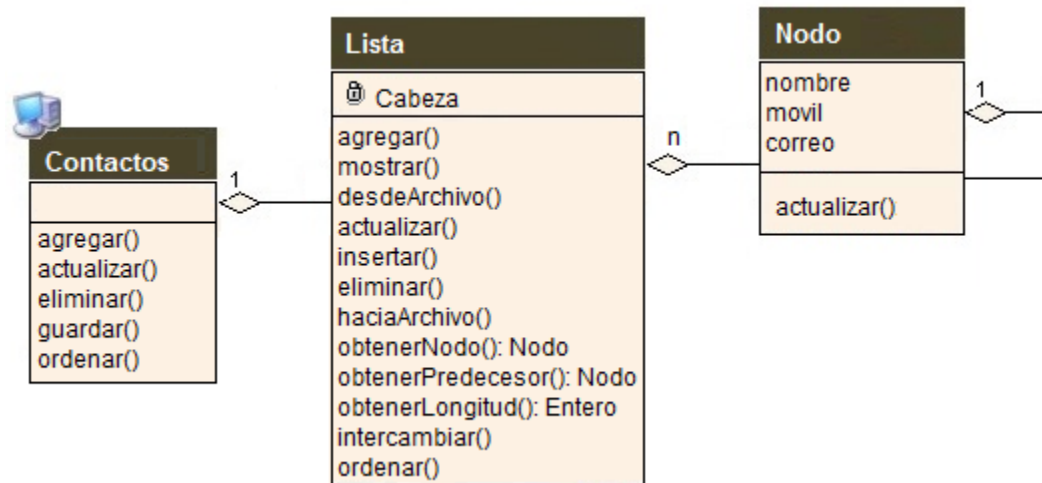
La ejecución podría lucir así:



| Nombre | Móvil | Correo |
|-------------------------------|------------|---|
| HURTADO CORTES JOHN FREDY | 3081077599 | zafita84@yahoo.es |
| LÓPEZ ESQUIVEL DENIS GABRIELA | 3055830018 | denisgabylopez@gmail.com |
| GIRALDO O LUISA FERNANDA | 3051119207 | fgiraldo070@gmail.com |
| DÍAZ GÓMEZ ANDRÉS FARID | 3107843289 | farbogomez@gmail.com |
| RESTREPO CAÑAS DAVID DE JESÚS | 3206765454 | djrc_restrep@hotmail.com |
| OSPINA BENÍTEZ DAVID | 3050920247 | david.ob89@gmail.com |
| DUARTE JESUS H | 3104767062 | ingjesushd@gmail.com |
| ESCOBAR BAENA RICAR | 3106900481 | richars1111@gmail.com |
| PACHECO ROCIO | 3098512213 | pacheco.rocio@gmail.com |
| CARDONA SANCHEZ ALEX | 3108199490 | alexander.cardona.sanchez@correounivalle.edu.co |

R/

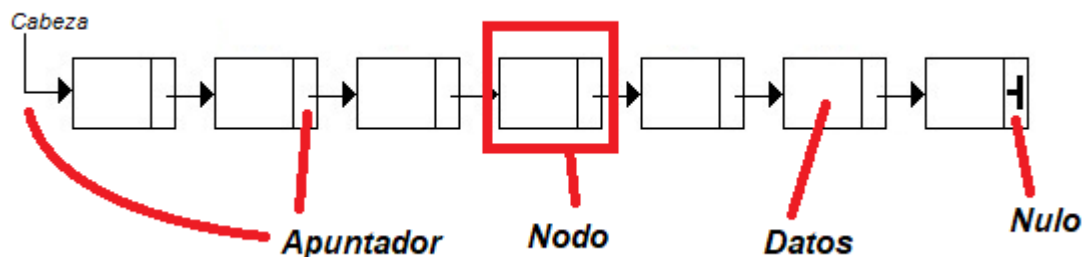
- El modelado bajo el paradigma Orientado a Objetos se ilustra en el siguiente diagrama de clases:



Para comprender este diagrama y el ejercicio, es importante tener en cuenta los siguientes fundamentos:

Listas Ligadas

Una **Lista ligada** es un grupo de datos organizados secuencialmente, pero a diferencia de los vectores, la organización no está dada implícitamente por su posición en el arreglo. En una lista ligada cada elemento es un nodo que contiene los datos y además un apuntador (o liga) al siguiente dato



Para poder manejar una lista es necesario contar con un apuntador al primer elemento de la lista llamado **nodo Cabeza**

Las ventajas de las listas ligadas son que:

- Permiten que sus tamaños cambien durante la ejecución del programa
- Proveen una mayor flexibilidad en el manejo de los datos.

La definición del nodo se haría mediante la siguiente clase:

```
class Nodo():

    def __init__(varClase, nombre, movil, correo):
        varClase.nombre = nombre
```



```
varClase.movil = movil  
varClase.correo = correo  
varClase.siguiente = None
```

La definición inicial de la lista sería mediante el siguiente código:

```
from Nodo import Nodo  
  
class Lista():  
  
    def __init__(varClase):  
        varClase.cabeza = None
```

Agregar de Nodo

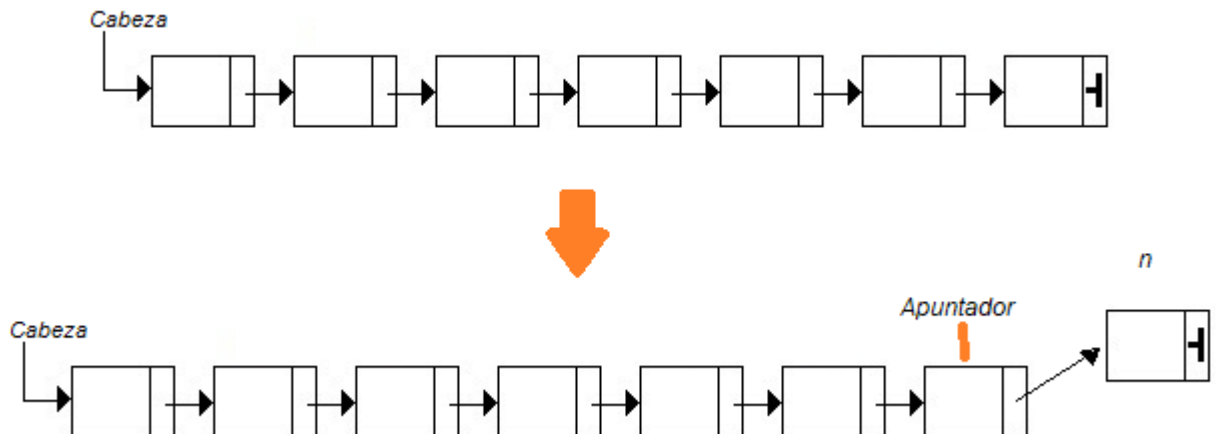
La operación inicial que se puede realizar con una lista es agregar un nodo. Para ello se tienen 2 situaciones

- La lista está vacía, es decir, el nodo *Cabeza* es nulo



En este caso el nuevo nodo (*n*) pasa a ser el nodo *Cabeza*

- La lista tiene nodos, así que hay que recorrerla hasta llegar al último nodo



En este caso, el apuntador *siguiente* del ultimo nodo (referenciado por la variable *apuntador*) apuntará al nuevo nodo (*n*)

Las siguientes serían las instrucciones del método para este propósito:

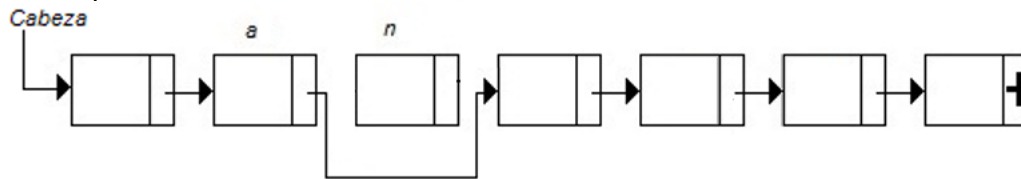
```
#metodo para agregar un nodo a la lista  
def agregar(varClase, n):  
    #Verificar que haya un nodo para agregar  
    if n != None:
```



```
if varClase.cabeza == None:
    #La lista esta vacia
    #El nodo agregado es la cabeza
    varClase.cabeza = n
else:
    #recorrer la lista hasta el ultimo nodo
    apuntador = varClase.cabeza
    while apuntador.siguiente != None:
        apuntador = apuntador.siguiente
    apuntador.siguiente = n
    n.siguiente = None
```

Eliminar Nodos

El retiro de un nodo implica básicamente poner a apuntar el nodo antecesor al nodo siguiente del nodo que se está retirando:



Se debe tener en cuenta si el nodo a retirar es el nodo cabeza:

Se debe definir en primer lugar el método *obtenerPredecesor()* que halla el apuntador al nodo antecesor de cualquier nodo de la lista, muy útil para diversas operaciones como el eliminado o el intercambio de nodos:

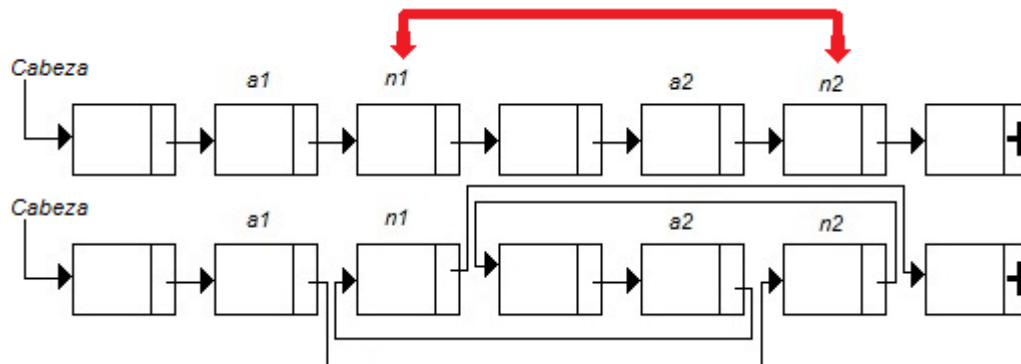
```
def obtenerPredecesor(varClase, n):
    predecesor = None
    if n != None and varClase.cabeza != None and varClase.cabeza!=n:
        predecesor = varClase.cabeza
        while predecesor != None and predecesor.siguiente!=n:
            predecesor = predecesor.siguiente
    return predecesor
```

El Código del método para eliminación sería el siguiente:

```
def eliminar(varClase, n):
    if n!= None:
        if n == varClase.cabeza:
            varClase.cabeza.siguiente = n.siguiente
        else:
            predecesor = varClase.obtenerPredecesor(n)
            predecesor.siguiente = n.siguiente
```

Intercambio de Nodos

Para el ordenamiento de los datos en la lista ligada se requiere una operación de **Intercambio** de nodos que se puede representar físicamente así:



Donde:

- a1** Nodo antecesor al primer nodo a intercambiar
- n1** Primer nodo a intercambiar
- a2** Nodo antecesor al segundo nodo a intercambiar
- n2** Segundo nodo a intercambiar

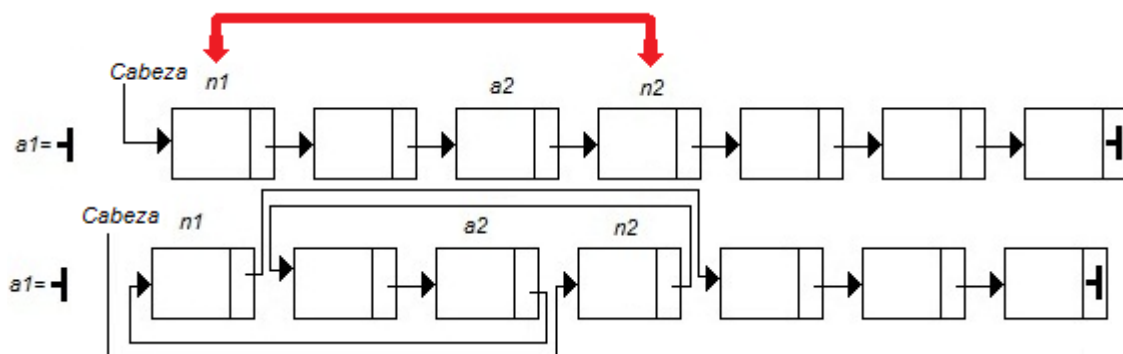
Lo cual equivale a las siguientes instrucciones:

```

a1.siguiente = n2
#Se guarda temporalmente el apuntador siguiente del segundo nodo
t = n2.siguiente
n2.siguiente = n1.siguiente
a2.siguiente = n1
n1.siguiente = t
    
```

El asunto es que se pueden presentar situaciones donde no se puedan ejecutar todas las anteriores instrucciones o se deba prever algunas situaciones:

a. El nodo **n1** es el nodo **Cabeza**



Se debe condicionar la asignación del nodo siguiente al antecesor del nodo **n1** (que en este caso no existe), así como cambiar el nodo **cabeza**.

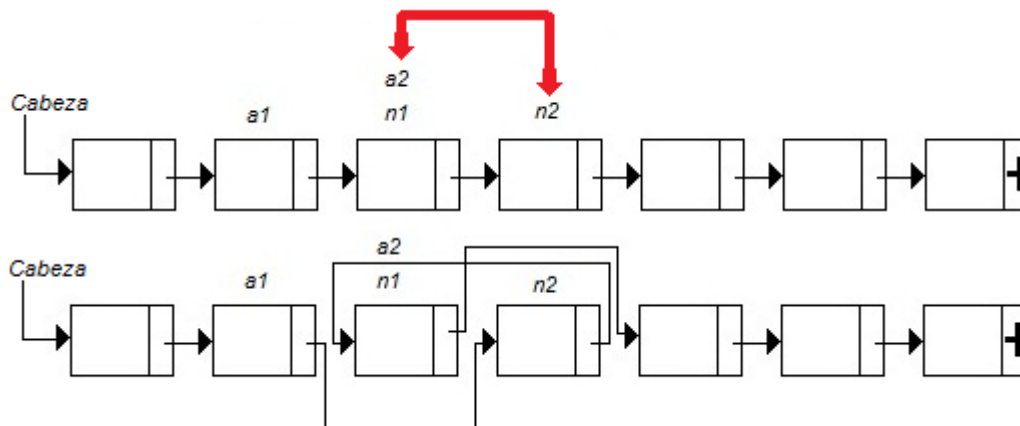
```

if a1 != None:
    a1.siguiente = n2
else:
    cabeza = n2
#Se guarda temporalmente el apuntador siguiente del segundo nodo
t = n2.siguiente
n2.siguiente = n1.siguiente
a2.siguiente = n1
    
```



```
n1.siguiente = t
```

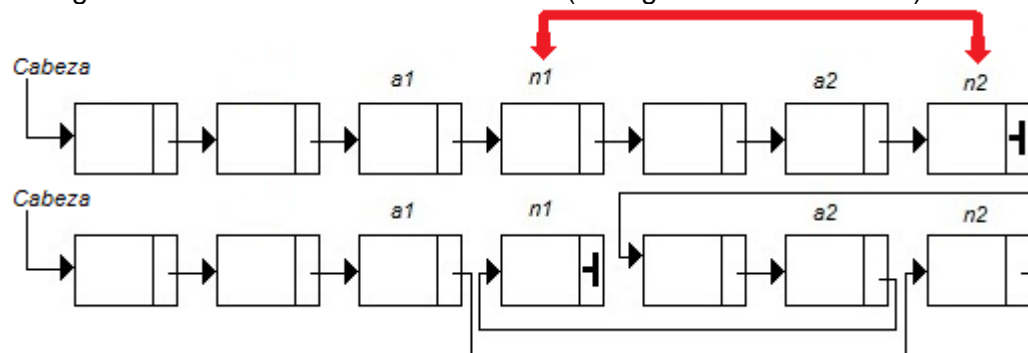
- b. El antecesor del nodo **n2** es el nodo **n1**



Se condiciona adicionalmente la asignación de los siguientes nodos del nodo **n2** y de su antecesor **a2** para evitar redundancia de asignaciones:

```
if a1 != None:
    a1.siguiente = n2
else:
    cabeza = n2
#Se guarda temporalmente el apuntador siguiente del segundo nodo
t = n2.siguiente
if n1 != a2:
    n2.siguiente = n1.siguiente
    a2.siguiente = n1
else:
    n2.siguiente = n1
n1.siguiente = t
```

- c. El segundo nodo **n2** es el último de la lista (Su siguiente nodo es nulo):



El código no requiere ningún cambio

Finalmente, el siguiente sería el código completo del método para realizar el intercambio de nodos:



```
def intercambiar(varClase, n1, n2) :
    if n1 != None and n2 != None and n1 != n2:
        cambiarSiguientes = False
        predecesor1 = varClase.obtenerPredecesor(n1)
        predecesor2 = varClase.obtenerPredecesor(n2)
        if predecesor1 != None:
            if predecesor1 != n2:
                predecesor1.siguiente = n2
                cambiarSiguientes = True
            else:
                n2.siguiente = n1.siguiente
                n1.siguiente = n2
                if predecesor2 != None:
                    predecesor2.siguiente = n1
        else:
            #El Cabeza va a ser el segundo nodo
            varClase.cabeza = n2
            cambiarSiguientes = True

        if predecesor2 != None:
            if predecesor2 != n1:
                predecesor2.siguiente = n1
                cambiarSiguientes = True
            else:
                cambiarSiguientes = False
                n1.siguiente = n2.siguiente
                n2.siguiente = n1
                if predecesor1 != None:
                    predecesor1.siguiente = n2
        else:
            #El Cabeza va a ser el primer nodo
            varClase.cabeza = n1
            cambiarSiguientes = True

    if cambiarSiguientes:
        siguiente1 = n1.siguiente
        n1.siguiente = n2.siguiente
        n2.siguiente = siguiente1
```

Se puede observar en este código que toda la operación de intercambio se condiciona a que haya nodos en la lista, que los nodos a intercambiar no sean los mismos, y que los nodos no sean nulos.

Ordenamiento de los Nodos de la Lista

Tomando como base el método de ordenamiento de la burbuja, se comparan los valores almacenados en los nodos de la siguiente manera:

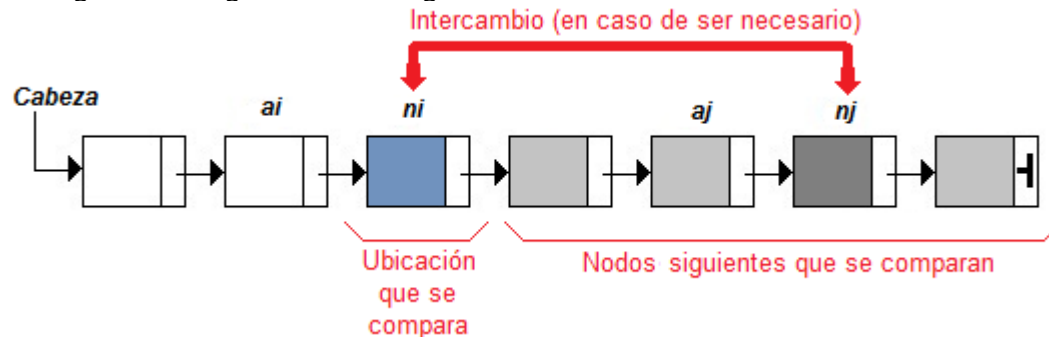
Se inicia con el nodo **cabeza**, el cual se compara contra los nodos restantes (Lo que realmente se compara es alguno de los valores del nodo). Para que la lista quede ordenada ascendentemente, se verifica que el valor del primer nodo sea mayor que el del segundo, y en



caso afirmativo, se hace el intercambio de nodos. Al final de estas comparaciones se garantiza que en la primera posición quede el nodo con el menor valor.

El anterior ciclo de comparaciones se repite para los demás nodos de la lista, hasta llegar al penúltimo nodo que será el último que tenga nodos restantes con quién compararse (el último nodo de la lista).

La siguiente imagen ilustra el algoritmo en un momento dado



Donde:

- ai** Nodo antecesor al nodo en la ubicación que se está comparando
- ni** Nodo en la ubicación que se está comparando
- aj** Nodo antecesor de alguno de los restantes nodos que se compara
- nj** Alguno de los restantes nodos que se compara

El siguiente código resuelve el ordenamiento de los nodos:

```
#Ordenar ascendentemente la lista por el nombre
def ordenar(varClase):
    if varClase.cabeza != None:
        apuntador1 = varClase.cabeza
        while apuntador1.siguiente != None:
            apuntador2 = apuntador1.siguiente
            while apuntador2 != None:
                #Intercambiar siempre y cuando el nombre siguiente sea menor
                if apuntador1.nombre > apuntador2.nombre:
                    n1 = apuntador1
                    n2 = apuntador2
                    varClase.intercambiar(n1, n2)
                    apuntador1 = n2
                    apuntador2 = n1
                apuntador2 = apuntador2.siguiente
            apuntador1 = apuntador1.siguiente
```

Se puede observar lo siguiente en este código:

- La condición para el primer ciclo busca evitar que se llegue al último nodo. Esto permite que siempre se pueda iniciar el primer nodo restante (**nj**) en el siguiente
- En caso de intercambio de los nodos, también se debe hacer intercambio de las variables apuntadoras (**ni** y **nj**)
- El ordenamiento se hará por el valor almacenado en el campo **nombre** de los nodos.