Práctico 1 – Tipo Abstracto de datos Listas

1. Defina el tipo abstracto de datos Lista Encadenada (LinkedList) de enteros, con las operaciones básicas para su manipulación (deberá tener por lo menos operaciones de agregar, eliminar y obtener). Para la definición realice un diagrama UML del mismo.

2. Realice una implementación TAD LinkedList de enteros, colocándolo en el paquete uy.edu.um.prog2.tad.linkedlist

3. Agregue al TAD LinkedList una operación que indique si un elemento se encuentra en la lista.

4. Agregue al TAD LinkedList las siguientes operaciones si ya no las tenía como parte de su definición:

```
public void addFirst(int value) //** Agrega al principio de la
lista **//
public void addLast(int value) //** Agrega al final de la lista
**//
```

5. Define e implemente el TAD Lista doblemente encadenada de enteros, donde cada nodo además de tener una referencia al siguiente elemento tiene una referencia al elemento anterior. Se deberá tener una operación agregar que ingrese los elementos de forma ascendente.

6. Dado el TAD lista doblemente encadenada del ejercicio anterior, defina e implemente el TAD Lista encadenada circular.

7. Con la implementación de los TAD, implementar un juego que solucione el "problema de Josefo": existen N personas numeradas de 1 a N, las mismas se encuentra dispuestas en un circulo. Empezando con la persona 1 se comienza a saltar de a M lugares, siendo la M-esima persona la invitada a abandonar el circulo. Luego que dicha persona abandona su lugar, el circulo se cierra y el juego continua con la persona que estaba dispuesta después del que abandonó. La ultima persona es la ganadora de este juego.

   Además de saber quien es la persona que no abandonará el circulo se desea saber el orden en el que el resto de las personas fueron abandonando el circulo.

   /**
   *
   *@param m indica la cantidad de lugares antes de sacar una persona, este valor puede ser negativo y indica la dirección con la cual se recorre la ronda.
   *@param colIntegrantes los integrantes participantes del juego, es una

lista de enteros.
*@throws IllegalArgumentsException cuando la cantidad m es mayor que el total de integrantes
* @return la lista de jugadores que fueron eliminados en el orden de eliminación
*/
public List juego(int m, List collIntegrantes) { ... };

8. Modifica la TAD LinkedList de entero para ahora disponer de un LinkedList de Objetos. Intente realizar la definición de su interfaz usando Generics.

9. Agregue a TAD LinkedList de objetos una operación agregar que permita ingresar los elementos en orden. Investigue cual sería la mejor estrategia para saber si un elemento es mayor que otro.

10. Supongamos que se tiene una lista l de Objetos y una lista p de enteros que está ordenada de forma ascendente.
El método visualizar(LinkedList P) de la clase LinkedList imprime los elementos de la lista L que estén en posiciones dadas por P. Por ejemplo: LinkedList l = new LinkedList(); // se carga l con objetos ..... LinkedList p = new LinkedList(); Integer i = new Integer(1); Integer j = new Integer(3); p.agregarPrimero(j); p.agregarPrimero(i); l.visualizar(P);

    Imprime el primer y el tercer elemento de l. Escriba el método visualizar usando las operaciones básicas de las listas.

11. Implemente el siguiente método de una lista: intercambiar (Object, dirección). El método debe ubicar el Elemento que contiene el objeto pasado como parámetro e intercambiar el Elemento encontrado (no solo el objeto del Elemento) con el Elemento anterior si el parámetro dirección vale –1 y con el Elemento siguiente si dirección vale 1.
Implemente el método en el caso de una LinkedList y de una lista doblemente enlazada (Elemento contiene un campo next y otro prev).

12. Implemente un método en una lista que reciba como parámetro otra lista y devuelva:

    a. Una nueva lista conteniendo los Objetos que están en ambas listas (append).
    b. Una nueva lista conteniendo los Objetos que estén en una u otra lista (pero no en ambas).

13. Se considera la clase List con la siguiente estructura:
public class ListNode { public int item;
public ListNode next; //.....
}

```
public class List{ ListNode head;
//..... }
```

a. Escribir un método NO recursivo de la clase List que recorra la lista y genere una nueva lista con todos los nodos cuyos datos (atributo ítem) sean mayores o iguales a 0.
   public List crearNuevaLista();
b. Escribir un método NO recursivo de la clase List que recorra la lista y elimine de la lista todos los nodos cuyos datos (atributo ítem) sean mayores o iguales a 0.
   public void eliminarNodosPositivos();