**UNIDAD TEMÁTICA 3: Listas, Pilas y Colas**

**PRACTICOS DOMICILIARIOS INDIVIDUALES - FORMULACION DE SEUDOCÓDIGO**

**Incorporar un nuevo producto**

**clase TLista**

**Lenguaje Natural :**

En equipo:

El método debe ver si la lista está vacía. Si es el caso, el primer nodo de la lista es el nodo ha insertar. Si la lista no está vacía, se debe recorrer toda la lista hasta que el puntero de siguiente sea null, indicando que estamos parados en el último elemento. Entonces, asignamos al siguiente del último nodo, el nodo a insertar.

Corrección Individual:

Aplicando TDA LISTA, este algoritmo inserta un nuevo elemento al comienzo de la lista siendo la referencia al siguiente el que con anterioridad fuese el primero. En el caso de ser una lista vacía se insertará como primer elemento de esta.

**Precondiciones :**

En equipo :

Debe existir la lista a la cual quiero insertar el elemento

Todos los nodos insertados son del tipo Producto

El objeto ha insertar no debe ser null

No debe existir otro objeto igual en la lista

Corrección Individual:

El método de TLista es mejor que reciba para este problema un TNodo como parámetro, por lo cual se debe tener en cuenta:

-> El nodo ingresado por parámetro no debe ser nulo.

-> El nodo ingresado por parámetro no debe estar en la lista.

**Postcondiciones :**

En equipo:

El objeto será agregado a al final de la lista

El largo de la lista aumentará en 1

Corrección Individual:

-> El nodo queda insertado al comienzo de la lista.

-> La lista aumenta la cantidad de nodos en 1.

**Ordenes de ejecución :**

O(N)

AHORA O(1)

**Seudocódigo :**

TLista.insertarAlFinal (dato : Objeto, etiqueta : Comparable)

COM

TNodo <- new nuevoNodo(etiqueta, obj)

Si primero de la lista esVacia():

primero <- nuevoNodo

SiNo:

actual <- primero

Mientras actual.siguiente <> null: O(N)

actual <- actual.siguiente

FinMientras

actual.siguiente <- nuevoNodo

FinSi

FIN

TLista.insertarDelante (TNodo : unNodo) : Void

COM

SI this.esVacia()

this.primero <- unNodo

SI NO

unNodo.setSiguiente(primero)

this.primero <- unNodo

FIN SI

FIN

**Agregar stock**

**Clase Almacén y Clase TLista**

\*Almacen debe recibir el TNodo referente a la etiqueta buscada, TLista implementa buscar por la etiqueta.

**Lenguaje Natural :**

En equipo:

El método debe ver si la lista está vacía. Si es el caso, debe devolver null dado que no existen nodos. Si la lista no está vacía, se debe recorrer la lista hasta que la etiqueta del nodo actual sea igual a la etiqueta pasada por parámetro, devolviendo dicho nodo.

Corrección Individual:

\*El método que encontrara el nodo del cual queremos modificar su dato, recibe una etiqueta y recorre la lista buscando el nodo que tenga esa etiqueta, retornando el nodo en caso de encontrarlo o nulo en caso de no hacerlo.

**PRECONDICIONES:**

En equipo:

La lista de donde buscaremos debe existir

La lista debe contener nodos identificados con su etiqueta

Corrección individual:

En este caso el parámetro que recibe es un dato Comparable, no debe tener precondiciones el método Buscar.

**POSTCONDICIONES:**

En equipo:

El método debe retornar el nodo en caso de haber encontrado el correcto

El método debe retornar null en caso de no haber encontrado el elemento

La lista no debe quedar alterada de ninguna forma

Corrección individual:

Las postcondiciones del trabajo en equipo son las correctas

**Orden de ejecución :**

En el peor de los casos O(N) Correcto

**PSEUDOCÓDIGO:**

**TLista.buscar** (unaEtiqueta :  Comparable): TNodo      // **devuelve un TNodo**

**COM**

Si esVacía() entonces

Devolver nulo                             // **salir**

SiNo

Actual **<-** primero

Mientras Actual **<>** nulo  O(N)

Si Actual.getEtiqueta() **=** unaEtiqueta

Devolver Actual                            **// salir**

FinSi

Actual **<-** Actual.Siguiente

FinMientras

Devolver nulo                            **// salir**

FinSiNO

**FIN** CORRECTO

**Listar Ordenado Por Nombre**

**Clases TListaProducto**

\*SE PIDE LISTAR LOS PRODUCTOS ORDENADOS POR NOMBRE, la clase TLista NO tiene no debe acceder a los datos del nodo. Se crea una nueva clase que hereda de TLista e implementa listas ordenadas por nombre de productos.

**No sirve para esta tarea**

**TLista.insertarOrdenado** (dato : Objeto, etiqueta : Comparable)

**COM**

nuevoNodo **<-** new nuevoNodo(etiqueta, obj)

**Si** lista esVacia()

primero **<-** nuevoNodo

**SiNo**

**Si** nuevoNodo.getEtiqueta() **menor** primero.getEtiqueta() // **comparable**

primero **<-** nuevoNodo   // **Si con el primero ya es menor, finaliza**

**SiNo**

Actual **<-** primero.siguiente

Anterior **<-** primero

**Mientras** Actual **<>** nulo:

**Si** nuevoNodo.getEtiqueta()  **menor** Actual.getEtiqueta():

nuevoNodo.siguiente **<-** Actual

Actual.siguiente **<-** nuevoNodo

**FinSi**

Actual **<-** Actual.siguiente

**FinMientras**

**FINSI**

**FinSi**

**FIN**

**Lenguaje Natural :**

El método recorrerá una lista armando otra con un criterio de ordenado, esta vez por uno de los sub-datos de los datos de los TNodos, este será el “Nombre del Producto”.

**Precondiciones :**

No recibe ningún parametro. La TLista que se recorre, puede ser null, entonces no hay precondiciones.

**Poscondiciones :**

Resulta en una lista con los mismos elementos de TLista ordenados por nombre.

**Seudocódigo :**

productosPorNombre() : TListaProducto

COM

TListaProducto listaProducto <- new TListaProducto();

Si lista esVacia()

DEVOLVER listaProducto

FIN SI

TNodo nodoProd = new TNodo(primero.getEtiq(),primero.getDato())

TNodo actual <- this.primero

TNodo anterior <- actual

listaProducto.insertar(nodoProd) //primero de la TListaProducto

Mientras actual <> null

TNodo actualP <- listaProducto.getPrimero();

Mientras actualP <> null

Si actual.getDato().getNombre.compareTo

actualP) < 0 //actual < actualP

TNodo nodoProducto <- new TNodo (actual.getEtiqueta(),actual.getDato())//nodoProd

nodoProducto.setSiguiente(actual)//Orden Crecient

anterior.setSiguiente(nodoProducto)

//queda insertado entre anterior y actual

anterior <- actual

actual <- actual.getSiguiente()

actualP <- NULL //para salir del loop

SI NO

actualP <- actualP.getSiguiente() //sigo avanzando

FIN MIENTRAS

FIN MIENTRAS