Especificación lógica TAD: LDECircularG

1. Elementos:

La estructura almacenará todo tipo de elementos, por lo que declararemos el tipo E en Java, usando la genericidad.

Será una estructura heterogénea, podemos almacenar todo tipo de datos primitivos.

1. Tipo de organización:

La lista enlazada tendrá una organización lineal, puesto que los nodos solo se conectan con el siguiente y con el anterior.

1. Dominio:

El dominio del TAD al ser heterogéneo, podemos almacenar cualquier tipo de dato.

1. Operaciones:
   1. Void insertar(Elemento):

* Nombre: insertar(E x)
* Descripción: inserta un elemento al principio de la lista.
* Datos de entrada: objeto.
* Salida: ninguno.
* Precondiciones: tener el objeto a agregar ya creado.
* Postcondiciones: tener el objeto agregado en la lista, como primer elemento.

* 1. Void insertar(elemento, indice):
* Nombre: insertar(E x, int e)
* Descripción: insertar un objeto en una posición concreta.
* Datos de entrada: un objeto o y un índice entero.
* Salida: ninguna.
* Precondiciones: ninguna.
* Postcondiciones: tener el objeto insertado en una posición concreta.
  1. Void insertarEnFin(elemento):
* Nombre: insertarEnFin(E x)
* Descripción: insertar un elemento al final de la lista.
* Datos de entrada: un objeto o.
* Salida: ninguna.
* Precondiciones: ninguna
* Postcondiciones: tener el objeto insertado al final de la lista.
  1. Int índiceDe(elemento):
* Nombre: indiceDe(E e)
* Descripción: pasamos una referencia, y si existe en la lista nos devuelve la posición, y si no existe, nos devuelve -1.
* Datos de entrada: Referencia de un objeto.
* Salida: Entero posición del objeto, o -1 si no existe.
* Precondiciones: ninguna.
* Postcondiciones: conocer la posición del objeto dentro de la lista.
  1. E recuperar(posición):
* Nombre: recuperar(int i).
* Descripción: devuelve el elemento que esté en la posición dada.
* Datos de entrada: Entero.
* Salida: objeto.
* Precondiciones: ninguna.
* Postcondiciones: tener el elemento de cierta posición.
  1. Boolean vaciarLista():
* Nombre: vaciarLista()
* Descripción: vacía la lista.
* Datos de entrada: ninguno.
* Salida: booleano.
* Precondiciones: ninguna.
* Postcondiciones: lista vacía.
  1. Boolean eliminar(elemento):
* Nombre: eliminar(E x)
* Descripción: elimina el nodo de un objeto que pasamos por referencia.
* Datos de entrada: objeto.
* Salida: booleano.
* Precondiciones: ninguna.
* Postcondiciones: elemento eliminado de la lista.
  1. E eliminar(posicion):
* Nombre: eliminar(int i)
* Descripción: eliminar el nodo que esté en la posición dada.
* Datos de entrada: Entero.
* Salida: objeto.
* Precondiciones: ninguna.
* Postcondiciones: eliminado el nodo de la posición dada.
  1. Boolean esVacia():
* Nombre: esVacia()
* Descripción: devuelve true si la lista está vacía, y false si la lista no está vacía.
* Datos de entrada: ninguno.
* Salida: booleano.
* Precondiciones: ninguno.
* Postcondiciones: saber si la lista está vacía.
  1. Boolean contiene(elemento):
* Nombre: contiene(object o)
* Descripción: devuelve true si la lista contiene el elemento, y false si no.
* Datos de entrada: objeto.
* Salida: booleano.
* Precondiciones: ninguna.
* Postcondiciones: saber si la lista contiene el elemento.
  1. Int talla():
* Nombre: talla()
* Descripción: devuelve el número de elementos de la lista.
* Datos de entrada: ninguno.
* Salida: entero.
* Precondiciones: ninguna.
* Postcondiciones: saber el tamaño de la lista.

* 1. E[] toArray(array):
* Nombre: toArray(E[] a)
* Descripción: transforma la lista en un array del tipo indicado.
* Datos de entrada: ninguno.
* Salida: Array que contiene en cada posición un nodo de la lista.
* Precondiciones: ninguna.
* Postcondiciones: tener un array que contenga la lista.
  1. String toString(): (lanzadera)
* Nombre: toString()
* Descripción: transformar el contenido de la lista en un String.
* Datos de entrada: ninguno.
* Salida: String.
* Precondiciones: que los datos sean convertibles a string.
* Postcondiciones: tener la lista en forma de string.
  1. String toString(nodo, String): (recursivo)
* Nombre: toString(NodoLDEG<E>, String s)
* Descripción: transformar el contenido de la lista en un String de forma recursiva.
* Datos de entrada: nodo ultimo, y string vacio.
* Salida: String.
* Precondiciones: que los datos sean convertibles a string.
* Postcondiciones: tener la lista en forma de string.
  1. String toStringRev():
* Nombre: toStringRev()
* Descripción: transformal la lista invertida en un String.
* Datos de entrada: ninguno.
* Salida: String.
* Precondiciones: que los datos sean convertibles en string.
* Postcondiciones: tener la lista invertida en forma de string.
  1. String toStringRev(nodo, String): (recursivo)
* Nombre: toString(NodoLDEG<E>, String s)
* Descripción: transformar el contenido de la lista en un String de forma recursiva.
* Datos de entrada: nodo ultimo, y string vacio.
* Salida: String.
* Precondiciones: que los datos sean convertibles a string.
* Postcondiciones: tener la lista en forma de string.
  1. contarNodos(): (lanzadera)
* Nombre: contarNodos()
* Descripción: contar de forma recursiva el numero de nodos que hay en la lista.
* Datos de entrada: ninguno
* Salida: entero.
* Precondiciones: ninguna.
* Postcondiciones: conocer el numero de nodos de la lista.
  1. contarNodos(): (recursivo)
* Nombre: contarNodos(NodoLDE<E>, int i)
* Descripción: contar de forma recursiva el numero de nodos que hay en la lista.
* Datos de entrada: nodo ultimo, y 0.
* Salida: entero.
* Precondiciones: ninguna.
* Postcondiciones: conocer el numero de nodos de la lista.
  1. invertirLista: (lanzadera)
* Nombre: invertirLista()
* Descripción: invertir el orden de los nodos de la lista.
* Datos de entrada: ninguno
* Salida: ninguno
* Precondiciones: ninguna.
* Postcondiciones: tener la lista invertida.
  1. invertirLista(nodo, nodo): (lrecursivo)
* Nombre: invertirLista()
* Descripción: invertir el orden de los nodos de la lista.
* Datos de entrada: nodo ultimo, nodo ultimo.siguiente.
* Salida: ninguno
* Precondiciones: ninguna.
* Postcondiciones: tener la lista invertida.
  1. duplicarNodosPares(): (lanzadera)
* Nombre: duplicarNodosPares()
* Descripción: duplicar los nodos cuyo valor sea par.
* Datos de entrada: ninguno
* Salida: ninguno
* Precondiciones: ninguna.
* Postcondiciones: tener la lista con el doble de nodos pares.
  1. duplicarNodosPares(nodo): (recursivo)
* Nombre: duplicarNodosPares(NodoLDEG<E> n)
* Descripción: duplicar los nodos cuyo valor sea par de forma recursiva.
* Datos de entrada: nodo ultimo.
* Salida: ninguno
* Precondiciones: ninguna.
* Postcondiciones: tener duplicados los nodos cuyo valor sea par.
  1. premiarClaves(): (lanzadera)
* Nombre: premiarClaves()
* Descripción: multiplicar por 2 el valor de los nodos cuyo valor sea igual a la talla de la lista.
* Datos de entrada: ninguno
* Salida: ninguno
* Precondiciones: ninguna.
* Postcondiciones: el valor de los nodos cuyo valor sea igual a la talla ahora valen el doble.
  1. premiarClaves(nodo, entero): (recursivo)
* Nombre: premiarClaves(NodoLDE<E> n, int c)
* Descripción: multiplicar por 2 el valor de los nodos cuyo valor sea igual a la talla de la lista.
* Datos de entrada: nodo ultimo, 0.
* Salida: entero.
* Precondiciones: ninguna.
* Postcondiciones: el valor de los nodos cuyo valor sea igual a la talla ahora valen el doble.