

Alumne: Carlos Martínez

Professor: Albert Solé

**ESTRUCTURES DE DADES**

**-LLISTA DOBLEMENT ENCADENADA-**

Contenido

[Classes utilitzades 3](#_Toc106388637)

[Aspectes a destacar 3](#_Toc106388638)

[Problemes trobats durant el desenvolupament 3](#_Toc106388639)

[Creació de les excepcions pròpies 3](#_Toc106388640)

[Mètodes 4](#_Toc106388641)

[Creació de la llista 4](#_Toc106388642)

[Inserció d’un element al final 4](#_Toc106388643)

[Inserció d’un element donat un índex 4](#_Toc106388644)

[Obtenció d’un element donat un índex 5](#_Toc106388645)

[Esborrat d’un element donada una posición 6](#_Toc106388646)

[Obtenció de la longitud de la llista 6](#_Toc106388647)

[Joc de proves 7](#_Toc106388648)

[Crear llista 7](#_Toc106388649)

[Imprimir llista 7](#_Toc106388650)

[Inserir al final 8](#_Toc106388651)

[Inserir en una posició 8](#_Toc106388652)

[Buscar en una posición 9](#_Toc106388653)

[Buscar per DNI 9](#_Toc106388654)

[Codi Font 10](#_Toc106388655)

# Classes utilitzades

* Ciutadà: conté els atributs i mètodes pròpis que necessitarem per a poder tractar amb un ciutadà
* CiutadaIterator: classe necessaria per a poder utilitzar l’iterator
* LlistaDoble: llista doblement encadenada amb tots els seus atributs i mètodes propis
* Nodo: clase que conté l’enllaç al següent node, a l’anterior, i la dada que guardem al mateix node
* TADCiutada: conté la definición dels mètodes que tindrà la nostra llista doblement encadenada

# Aspectes a destacar

## Problemes trobats durant el desenvolupament

-Casting entre classes, ja que a vegades operem amb dades genèriques, a vegades tipus primitius o de tipus ciutadà.

-Mal enllaçament entre nodes: a vegades enllaçava l’element actual amb el següent però no el següent amb l’actual i aixó causava problemes.

-Error al treballar amb mètodes genèrics ja que a vegades alguns eren de tipus <T> i no <T extends Comparable<T>> .

## Creació de les excepcions pròpies

-ElementoNoEncontrado:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

-NoSePuede:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

# Mètodes

## Creació de la llista

Texto

Descripción generada automáticamente

## Inserció d’un element al final

Texto

Descripción generada automáticamente

## Inserció d’un element donat un índex

Texto

Descripción generada automáticamente

## Obtenció d’un element donat un índex

Texto

Descripción generada automáticamente

## Esborrat d’un element donada una posición

Texto

Descripción generada automáticamente

## Obtenció de la longitud de la llista

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

# Joc de proves

Primer de tot he creat una llista amb 4 elements per no h’haver d’introduir-los manualment i li passo al mètode on es criden els mètodes.

Texto

Descripción generada automáticamente

## Crear llista

Tabla

Descripción generada automáticamente con confianza media

## Imprimir llista

Texto, Carta

Descripción generada automáticamente

## Inserir al final

Texto

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

## Inserir en una posició

Texto

Descripción generada automáticamenteTexto, Carta

Descripción generada automáticamente

## Buscar en una posición

Texto, Carta

Descripción generada automáticamente

Texto

Descripción generada automáticamente

## Buscar per DNI

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente con confianza media

# Codi Font

*package* Data;  
*import* java.util.\*;  
  
*import* Exceptions.\*;  
*public class* ListaDoble<T *extends* Comparable<T>> *implements* TADCiutada <T>,Iterable<Ciutada>{  
 *protected* Nodo<T> inicio,fin;  
 *private int* nElems;  
 *private int* posicioIterator;  
  
 *public* ListaDoble() {  
 inicio=fin=*null*;  
 }  
  
   
 */\*\*  
 \* Inicialització del primer element de la llista encadenada  
 \*/  
 public void* crear() {  
 inicio=*new* Nodo(*null*,*null*,*null*);  
 }  
 */\*\*  
 \* Mètode que comprova si la llista està plena o no  
 \* @return true if empty | false if not  
 \*/  
 public boolean* empty() {  
  
 *return* inicio==*null*;  
 }  
 */\*\*  
 \* Mètode per a inserir un element al final de la llista  
 \* @param data: dada a afegir  
 \*/  
 public void* Inserir(T data) {  
 *if*(!empty()) {  
 fin=*new* Nodo<>(*null*,fin,data);  
 fin.anterior.siguiente=fin;  
  
 }  
 *else* {  
 inicio=fin=*new* Nodo<T>(data);  
 }  
 nElems++;  
 }  
 */\*\*  
 \* Mètode per a inserir un element donada una posició  
 \* @param posi: index on afegir  
 \* @param data: dada a afegir  
 \*/  
 public void* Inserir(*int* posi,T data) *throws* NoSePuede {  
 *if*(posi<0||posi>nElems)   
 *throw new* NoSePuede(posi);  
 Nodo<T>aux=*new* Nodo<T>(data);  
 *if*(inicio==*null*) {  
 inicio=aux;  
 fin=aux;  
 }  
 *else if*(posi==0) {  
 aux.siguiente=inicio;  
 inicio.anterior=aux;  
 inicio=aux;  
 }  
 *else if*(posi==nElems) {  
 aux.anterior=fin;  
 fin.siguiente=aux;  
 fin=aux;  
 }  
 *else* {  
 Nodo<T>nodoAux=inicio;  
 *for*(*int* i=1;i<posi;i++) {  
 nodoAux=nodoAux.siguiente;  
 }  
 aux.siguiente=nodoAux.siguiente;  
 nodoAux.siguiente=aux;  
 aux.anterior=nodoAux;  
 aux.siguiente.anterior=aux;  
 }  
 nElems++;  
 }  
 */\*\*  
 \* Mètode per a retornar l'element d'una posició donada en cas de ser possible  
 \* @param posi: index de la posició  
 \*/  
 public* T Obtenir(*int* posi)*throws* NoSePuede {  
 *if*(posi<nElems&&posi>=0) {  
 *int* i=0;  
 Nodo<T> aux=inicio;  
 *while*(i<posi) {  
 aux=aux.siguiente;  
 i++;  
 }  
 *return* aux.data;  
 }  
 *else* {  
 *throw new* NoSePuede(posi);  
 }  
  
 }  
   
 */\*\*  
 \* Mètode per a esborrar un node, utilitzat al mètode 'Esborrar(int posi)'  
 \* @param elem: node a eliminar de la llista  
 \*/  
 public void* eliminarNodo(Nodo<T> elem) {  
 *if*(inicio==*null*||elem==*null*)  
 *return*;  
 *if*(inicio==elem)  
 inicio=elem.siguiente;  
  
 *//Ant<->Actual<->Siguiente  
 //Ant<->Siguiente  
 if*(elem.siguiente!=*null*)  
 elem.siguiente.anterior=elem.anterior;  
  
 *if*(elem.anterior!=*null*)  
 elem.anterior.siguiente=elem.siguiente;  
 }  
 */\*\*  
 \* Mètode per a esborrar un element donada una posició  
 \* @param posi: índex de l'element que es vol esborrar  
 \*/  
 public void* Esborrar(*int* posi) {  
 *if*(posi>=0 && posi<nElems) {  
 Nodo<T> aux=inicio;  
 *int* i;  
 *for*(i=1;aux!=*null*&&i<posi;i++) {  
 aux=aux.siguiente;  
 }  
 *if*(aux==*null*)  
 *return*;  
  
 eliminarNodo(aux);  
 nElems--;  
  
 }  
  
 }  
 */\*\*  
 \* Mètode per a comprovar si existeix a la llista un element passat per paràmetre  
 \* @param dato: element que volem buscar a la llista  
 \*/  
 public int* Buscar(T dato)*throws* ElementoNoEncontrado {  
 *int* n=1,i=0;  
 Nodo<T> aux=inicio;  
 String temp;  
  
 *while*(i<nElems) {  
 *if*(aux.data.compareTo(dato)==0)  
 *return* n;  
  
 aux=aux.siguiente;  
 n++;i++;  
 }  
 *throw new* ElementoNoEncontrado(i);  
  
 }  
   
 */\*\*  
 \* Mètode de l'iterator  
 \*/  
 public* Iterator<Ciutada>iterator(){  
 *return new* CiutadaIterator<T>(*this*);  
 }  
 */\*\*  
 \* Mètode per a duplicar llista actual  
 \* @return copia  
 \*/  
 public* ListaDoble<T> copia() {  
 *return this*;  
 }  
 */\*\*  
 \* Mètode per a imprimir la llista, utilitzem el mètode iterator()  
 \*/  
 public void* recorrer() {  
 *for* (Ciutada ciutada : *this*) {  
 System.out.println(ciutada);  
 }  
 }  
 */\*\*  
 \* Mètode que retorna la longitud de la llista actual  
 \*/  
 public int* Longitud() {  
 *return* nElems;  
 }  
  
  
}