Las listas son estructuras de datos que permiten tener cierta flexibilidad en su manejo, pueden crecer o acortarse según se lo requiera, existen varias formas de implementar una lista en Java en este caso se presenta un ejemplo en código utilizando punteros mediante la referencia a objetos.

Las listas básicamente se forman del enlace de nodos los cuales funcionan como contenedores para almacenar el valor y enlace al siguiente nodo.  
  
**Estructura de un nodo:**  
  
Valor: valor del dato a guardar.  
Enlace: apunta a la dirección en memoria del siguiente nodo.  
  
**Lista simplemente enlazada:**



**Características:**

* El último nodo de la lista no apunta a ninguno (null).
* Se accede a la lista mediante el primer nodo o también llamado inicio de la lista.
* No tiene acceso aleatorio es decir para acceder a un valor se debe recorrer toda la lista.

**Operaciones básicas:**

* agregar (valor): agrega el valor al final de la lista.
* insertar (referencia, valor): inserta el valor después del valor de referencia en la lista.
* remover (referencia): elimina el nodo con el valor que coincida con la referencia.
* editar (referencia): actualiza el valor de nodo con el valor que coincida con la referencia.
* esVacia (): retorna true si la lista está vacía, false en caso contrario.
* buscar (valor): retorna la true si el elemento existe en la lista, false caso contrario.
* eliminar(): elimina la lista
* listar (): imprime en pantalla los elementos de la lista.

**Programa Java:**  
  
  
**Clase Nodo.java**

/\*\*

**\*** Clase que define los elementos que debe tener un Nodo de la lista.

**\*** **@author** **xavier**

\*/

**public** **class** Nodo {

// Variable en la cual se va a guardar el valor.

**private** int valor;

// Variable para enlazar los nodos.

**private** Nodo siguiente;

/\*\*

**\*** Constructor que inicializamos el valor de las variables.

\*/

**public** void Nodo(){

**this**.valor = 0;

**this**.siguiente = **null**;

}

// Métodos get y set para los atributos.

**public** int getValor() {

**return** valor;

}

**public** void setValor(int valor) {

**this**.valor = valor;

}

**public** Nodo getSiguiente() {

**return** siguiente;

}

**public** void setSiguiente(Nodo siguiente) {

**this**.siguiente = siguiente;

}

}

**Clase Lista.java**

/\*\*

**\*** Clase que define las operaciones básicas que debe tener una lista simple.

**\*** **@author** **xavier**

\*/

**public** **class** Lista {

// Puntero que indica el inicio de la lista o conocida tambien

// como cabeza de la lista.

**private** Nodo inicio;

// Variable para registrar el tamaño de la lista.

**private** int tamanio;

/\*\*

**\*** Constructor por defecto.

\*/

**public** void Lista(){

inicio = **null**;

tamanio = 0;

}

/\*\*

**\*** Consulta si la lista esta vacia.

**\*** **@return** true si el primer nodo (inicio)**,** no apunta a otro nodo.

\*/

**public** boolean esVacia(){

**return** inicio == **null**;

}

/\*\*

**\*** Consulta cuantos elementos (nodos) tiene la lista.

**\*** **@return** numero entero entre [0**,**n] donde n es el numero de elementos

**\*** que contenga la lista.

\*/

**public** int getTamanio(){

**return** tamanio;

}

/\*\*

**\*** Agrega un nuevo nodo al final de la lista.

**\*** **@param** valor a agregar.

\*/

**public** void agregarAlFinal(int valor){

// Define un nuevo nodo.

Nodo nuevo = **new** Nodo();

// Agrega al valor al nodo.

nuevo.setValor(valor);

// Consulta si la lista esta vacia.

**if** (esVacia()) {

// Inicializa la lista agregando como inicio al nuevo nodo.

inicio = nuevo;

// Caso contrario recorre la lista hasta llegar al ultimo nodo

//y agrega el nuevo.

} **else**{

// Crea ua copia de la lista.

Nodo aux = inicio;

// Recorre la lista hasta llegar al ultimo nodo.

**while**(aux.getSiguiente() != **null**){

aux = aux.getSiguiente();

}

// Agrega el nuevo nodo al final de la lista.

aux.setSiguiente(nuevo);

}

// Incrementa el contador de tamaño de la lista

tamanio++;

}

/\*\*

**\*** Agrega un nuevo nodo al inicio de la lista.

**\*** **@param** valor a agregar.

\*/

**public** void agregarAlInicio(int valor){

// Define un nuevo nodo.

Nodo nuevo = **new** Nodo();

// Agrega al valor al nodo.

nuevo.setValor(valor);

// Consulta si la lista esta vacia.

**if** (esVacia()) {

// Inicializa la lista agregando como inicio al nuevo nodo.

inicio = nuevo;

// Caso contrario va agregando los nodos al inicio de la lista.

} **else**{

// Une el nuevo nodo con la lista existente.

nuevo.setSiguiente(inicio);

// Renombra al nuevo nodo como el inicio de la lista.

inicio = nuevo;

}

// Incrementa el contador de tamaño de la lista.

tamanio++;

}

/\*\*

**\*** Inserta un nuevo nodo despues de otro**,** ubicado por el valor que contiene.

**\*** **@param** referencia valor del nodo anterios al nuevo nodo a insertar.

**\*** **@param** valor del nodo a insertar.

\*/

**public** void insertarPorReferencia(int referencia, int valor){

// Define un nuevo nodo.

Nodo nuevo = **new** Nodo();

// Agrega al valor al nodo.

nuevo.setValor(valor);

// Verifica si la lista contiene elementos

**if** (!esVacia()) {

// Consulta si el valor existe en la lista.

**if** (buscar(referencia)) {

// Crea ua copia de la lista.

Nodo aux = inicio;

// Recorre la lista hasta llegar al nodo de referencia.

**while** (aux.getValor() != referencia) {

aux = aux.getSiguiente();

}

// Crea un respaldo de la continuación de la lista.

Nodo siguiente = aux.getSiguiente();

// Enlaza el nuevo nodo despues del nodo de referencia.

aux.setSiguiente(nuevo);

// Une la continuacion de la lista al nuevo nodo.

nuevo.setSiguiente(siguiente);

// Incrementa el contador de tamaño de la lista.

tamanio++;

}

}

}

/\*\*

**\*** Inserta un nuevo nodo despues en una posición determinada.

**\*** **@param** posicion en la cual se va a insertar el nuevo nodo.

**\*** **@param** valor valor del nuevo nodo de la lista.

\*/

**public** void insrtarPorPosicion(int posicion, int valor){

// Verifica si la posición ingresada se encuentre en el rango

// >= 0 y <= que el numero de elementos del la lista.

**if**(posicion>=0 && posicion<=tamanio){

Nodo nuevo = **new** Nodo();

nuevo.setValor(valor);

// Consulta si el nuevo nodo a ingresar va al inicio de la lista.

**if**(posicion == 0){

// Inserta el nuevo nodo al inicio de la lista.

nuevo.setSiguiente(inicio);

inicio = nuevo;

}

**else**{

// Si el nodo a inserta va al final de la lista.

**if**(posicion == tamanio){

Nodo aux = inicio;

// Recorre la lista hasta llegar al ultimo nodo.

**while**(aux.getSiguiente() != **null**){

aux = aux.getSiguiente();

}

// Inserta el nuevo nodo despues de del ultimo.

aux.setSiguiente(nuevo);

}

**else**{

// Si el nodo a insertar va en el medio de la lista.

Nodo aux = inicio;

// Recorre la lista hasta llegar al nodo anterior

// a la posicion en la cual se insertara el nuevo nodo.

**for** (int i = 0; i < (posicion-1); i++) {

aux = aux.getSiguiente();

}

// Guarda el nodo siguiente al nodo en la posición

// en la cual va a insertar el nevo nodo.

Nodo siguiente = aux.getSiguiente();

// Inserta el nuevo nodo en la posición indicada.

aux.setSiguiente(nuevo);

// Une el nuevo nodo con el resto de la lista.

nuevo.setSiguiente(siguiente);

}

}

// Incrementa el contador de tamaño de la lista.

tamanio++;

}

}

/\*\*

**\*** Obtiene el valor de un nodo en una determinada posición.

**\*** **@param** posicion del nodo que se desea obtener su valor.

**\*** **@return** un numero entero entre [0**,**n**-**1] n = numero de nodos de la lista.

**\*** **@throws** Exception

\*/

**public** int getValor(int posicion) **throws** **Exception**{

// Verifica si la posición ingresada se encuentre en el rango

// >= 0 y < que el numero de elementos del la lista.

**if**(posicion>=0 && posicion<tamanio){

// Consulta si la posicion es el inicio de la lista.

**if** (posicion == 0) {

// Retorna el valor del inicio de la lista.

**return** inicio.getValor();

}**else**{

// Crea una copia de la lista.

Nodo aux = inicio;

// Recorre la lista hasta la posición ingresada.

**for** (int i = 0; i < posicion; i++) {

aux = aux.getSiguiente();

}

// Retorna el valor del nodo.

**return** aux.getValor();

}

// Crea una excepción de Posicion inexistente en la lista.

} **else** {

**throw** **new** **Exception**("Posicion inexistente en la lista.");

}

}

/\*\*

**\*** Busca si existe un valor en la lista.

**\*** **@param** referencia valor a buscar.

**\*** **@return** true si existe el valor en la lista.

\*/

**public** boolean buscar(int referencia){

// Crea una copia de la lista.

Nodo aux = inicio;

// Bandera para indicar si el valor existe.

boolean encontrado = **false**;

// Recorre la lista hasta encontrar el elemento o hasta

// llegar al final de la lista.

**while**(aux != **null** && encontrado != **true**){

// Consulta si el valor del nodo es igual al de referencia.

**if** (referencia == aux.getValor()){

// Canbia el valor de la bandera.

encontrado = **true**;

}

**else**{

// Avansa al siguiente. nodo.

aux = aux.getSiguiente();

}

}

// Retorna el resultado de la bandera.

**return** encontrado;

}

/\*\*

**\*** Consulta la posición de un elemento en la lista

**\*** **@param** referencia valor del nodo el cual se desea saber la posición.

**\*** **@return** un valor entero entre [0**,**n] que indica la posición del nodo.

**\*** **@throws** Exception

\*/

**public** int getPosicion(int referencia) **throws** **Exception**{

// Consulta si el valor existe en la lista.

**if** (buscar(referencia)) {

// Crea una copia de la lista.

Nodo aux = inicio;

// COntado para almacenar la posición del nodo.

int cont = 0;

// Recoore la lista hasta llegar al nodo de referencia.

**while**(referencia != aux.getValor()){

// Incrementa el contador.

cont ++;

// Avansa al siguiente. nodo.

aux = aux.getSiguiente();

}

// Retorna el valor del contador.

**return** cont;

// Crea una excepción de Valor inexistente en la lista.

} **else** {

**throw** **new** **Exception**("Valor inexistente en la lista.");

}

}

/\*\*

**\*** Actualiza el valor de un nodo que se encuentre en la lista ubicado

**\*** por un valor de referencia.

**\*** **@param** referencia valor del nodo el cual se desea actualizar.

**\*** **@param** valor nuevo valor para el nodo.

\*/

**public** void editarPorReferencia(int referencia, int valor){

// Consulta si el valor existe en la lista.

**if** (buscar(referencia)) {

// Crea ua copia de la lista.

Nodo aux = inicio;

// Recorre la lista hasta llegar al nodo de referencia.

**while**(aux.getValor() != referencia){

aux = aux.getSiguiente();

}

// Actualizamos el valor del nodo

aux.setValor(valor);

}

}

/\*\*

**\*** Actualiza el valor de un nodo que se encuentre en la lista ubicado

**\*** por su posición.

**\*** **@param** posicion en la cual se encuentra el nodo a actualizar.

**\*** **@param** valor nuevo valor para el nodo.

\*/

**public** void editarPorPosicion(int posicion , int valor){

// Verifica si la posición ingresada se encuentre en el rango

// >= 0 y < que el numero de elementos del la lista.

**if**(posicion>=0 && posicion<tamanio){

// Consulta si el nodo a eliminar es el primero.

**if**(posicion == 0){

// Alctualiza el valor delprimer nodo.

inicio.setValor(valor);

}

**else**{

// En caso que el nodo a eliminar este por el medio

// o sea el ultimo

Nodo aux = inicio;

// Recorre la lista hasta lleger al nodo anterior al eliminar.

**for** (int i = 0; i < posicion; i++) {

aux = aux.getSiguiente();

}

// Alctualiza el valor del nodo.

aux.setValor(valor);

}

}

}

/\*\*

**\*** Elimina un nodo que se encuentre en la lista ubicado

**\*** por un valor de referencia.

**\*** **@param** referencia valor del nodo que se desea eliminar.

\*/

**public** void removerPorReferencia(int referencia){

// Consulta si el valor de referencia existe en la lista.

**if** (buscar(referencia)) {

// Consulta si el nodo a eliminar es el pirmero

**if** (inicio.getValor() == referencia) {

// El primer nodo apunta al siguiente.

inicio = inicio.getSiguiente();

} **else**{

// Crea ua copia de la lista.

Nodo aux = inicio;

// Recorre la lista hasta llegar al nodo anterior

// al de referencia.

**while**(aux.getSiguiente().getValor() != referencia){

aux = aux.getSiguiente();

}

// Guarda el nodo siguiente del nodo a eliminar.

Nodo siguiente = aux.getSiguiente().getSiguiente();

// Enlaza el nodo anterior al de eliminar con el

// sguiente despues de el.

aux.setSiguiente(siguiente);

}

// Disminuye el contador de tamaño de la lista.

tamanio--;

}

}

/\*\*

**\*** Elimina un nodo que se encuentre en la lista ubicado

**\*** por su posición.

**\*** **@param** posicion en la cual se encuentra el nodo a eliminar.

\*/

**public** void removerPorPosicion(int posicion){

// Verifica si la posición ingresada se encuentre en el rango

// >= 0 y < que el numero de elementos del la lista.

**if**(posicion>=0 && posicion<tamanio){

// Consulta si el nodo a eliminar es el primero

**if**(posicion == 0){

// Elimina el primer nodo apuntando al siguinte.

inicio = inicio.getSiguiente();

}

// En caso que el nodo a eliminar este por el medio

// o sea el ultimo

**else**{

// Crea una copia de la lista.

Nodo aux = inicio;

// Recorre la lista hasta lleger al nodo anterior al eliminar.

**for** (int i = 0; i < posicion-1; i++) {

aux = aux.getSiguiente();

}

// Guarda el nodo siguiente al nodo a eliminar.

Nodo siguiente = aux.getSiguiente();

// Elimina la referencia del nodo apuntando al nodo siguiente.

aux.setSiguiente(siguiente.getSiguiente());

}

// Disminuye el contador de tamaño de la lista.

tamanio--;

}

}

/\*\*

**\*** Elimina la lista

\*/

**public** void eliminar(){

// Elimina el valor y la referencia a los demas nodos.

inicio = **null**;

// Reinicia el contador de tamaño de la lista a 0.

tamanio = 0;

}

/\*\*

**\*** Mustra en pantalla los elementos de la lista.

\*/

**public** void listar(){

// Verifica si la lista contiene elementoa.

**if** (!esVacia()) {

// Crea una copia de la lista.

Nodo aux = inicio;

// Posicion de los elementos de la lista.

int i = 0;

// Recorre la lista hasta el final.

**while**(aux != **null**){

// Imprime en pantalla el valor del nodo.

**System**.out.print(i + ".[ " + aux.getValor() + " ]" + " -> ");

// Avanza al siguiente nodo.

aux = aux.getSiguiente();

// Incrementa el contador de la posión.

i++;

}

}

}

}

**Clase Main.java**

**public** **class** Main {

/\*\*

**\*** **@param** args

**\*** **@throws** Exception

\*/

**public** **static** void main(**String**[] args) **throws** **Exception** {

Lista lista = **new** Lista();

**System**.out.println("<<-- Ejemplo de lista simple -->>\n");

// Agregar al final de la lista

lista.agregarAlFinal(12);

lista.agregarAlFinal(15);

lista.agregarAlFinal(9);

// Agregar in inicio de la lista

lista.agregarAlInicio(41);

lista.agregarAlInicio(6);

**System**.out.println("<<-- Lista -->>");

lista.listar();

**System**.out.println("\n\n<<-- Tamaño -->");

**System**.out.println(lista.getTamanio());

**System**.out.println("\n<<-- Obtener el valor del nodo en la posicion 3 -->>");

**System**.out.println(lista.getValor(3));

**System**.out.println("\nInsrta un nodo con valor 16 despues del 15");

lista.insertarPorReferencia(15, 16);

lista.listar();

**System**.out.print(" | Tamaño: ");

**System**.out.println(lista.getTamanio());

**System**.out.println("\n\nInsrta un nodo con valor 44 en la posición 3");

lista.insrtarPorPosicion(3, 44);

lista.listar();

**System**.out.print(" | Tamaño: ");

**System**.out.println(lista.getTamanio());

**System**.out.println("\nActualiza el valor 12 del tercer nodo por 13");

lista.editarPorReferencia(12, 13);

lista.listar();

**System**.out.print(" | Tamaño: ");

**System**.out.println(lista.getTamanio());

**System**.out.println("\nActualiza el valor nodo en la posición 0 por 17");

lista.editarPorPosicion(0, 17);

lista.listar();

**System**.out.print(" | Tamaño: ");

**System**.out.println(lista.getTamanio());

**System**.out.println("\nElimina el nodo con el valor 41");

lista.removerPorReferencia(41);

lista.listar();

**System**.out.print(" | Tamaño: ");

**System**.out.println(lista.getTamanio());

**System**.out.println("\nElimina el nodo en la posición 4");

lista.removerPorPosicion(4);

lista.listar();

**System**.out.print(" | Tamaño: ");

**System**.out.println(lista.getTamanio());

**System**.out.println("\nConsulta si existe el valor 30");

**System**.out.println(lista.buscar(30));

**System**.out.println("\nConsulta la posicion del valor 9");

**System**.out.println(lista.getPosicion(9));

**System**.out.println("\nElimina la lista");

lista.eliminar();

**System**.out.println("\nConsulta si la lista está vacia");

**System**.out.println(lista.esVacia());

**System**.out.println("\n\n<<-- Fin de ejemplo lista simple -->>");

}

}