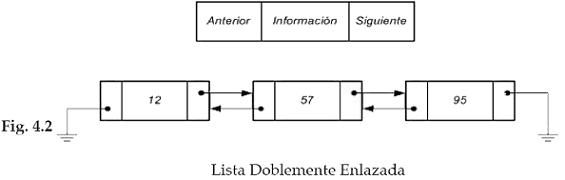
**Java - estructura de datos - Listas doblemente enlazadas**

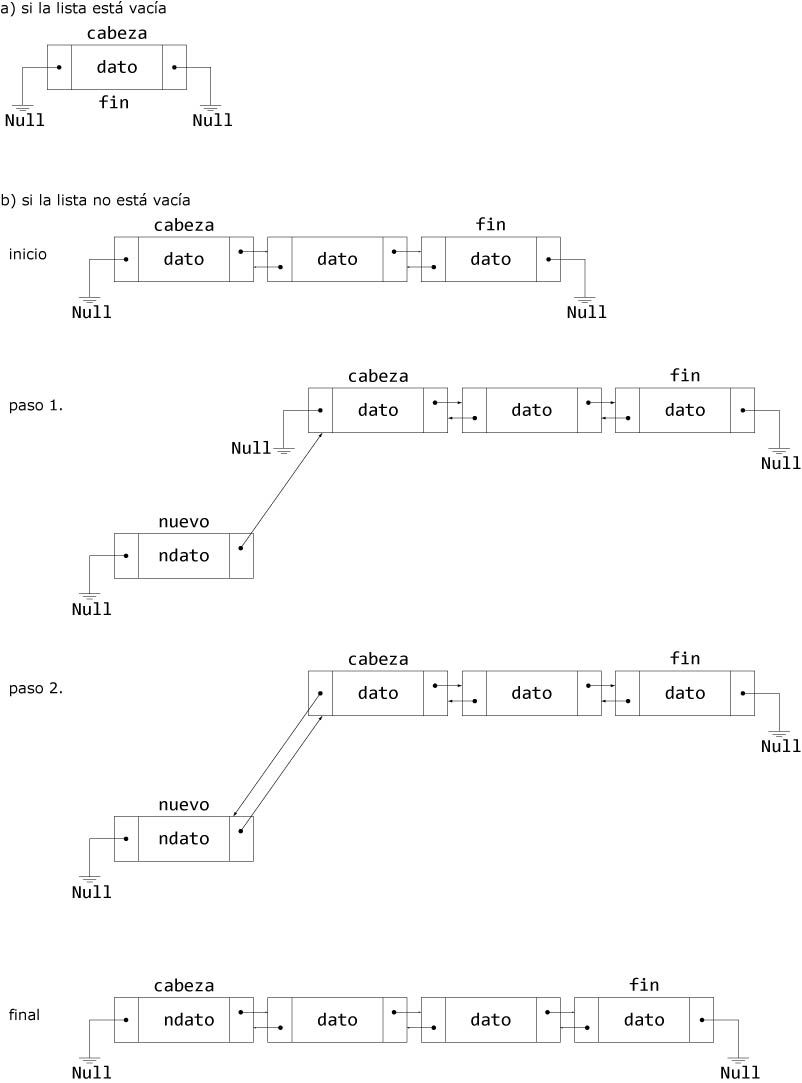
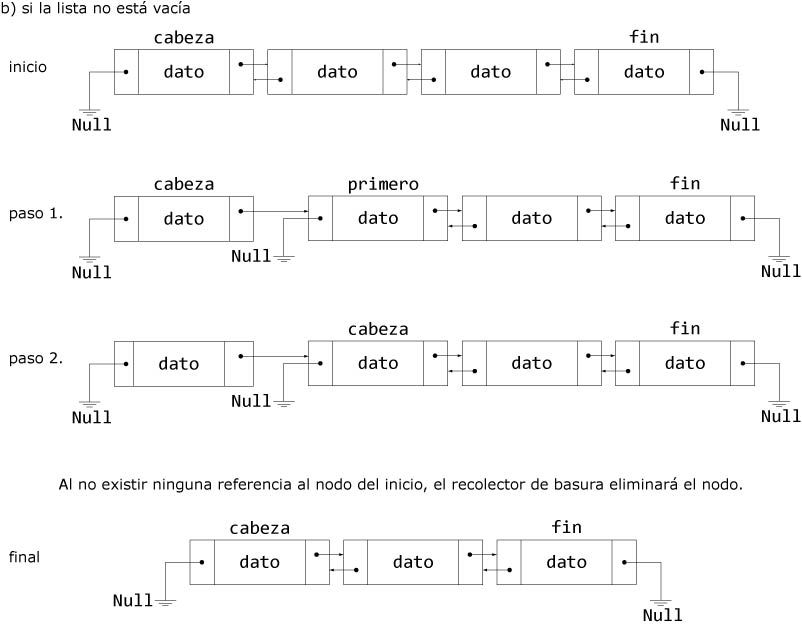
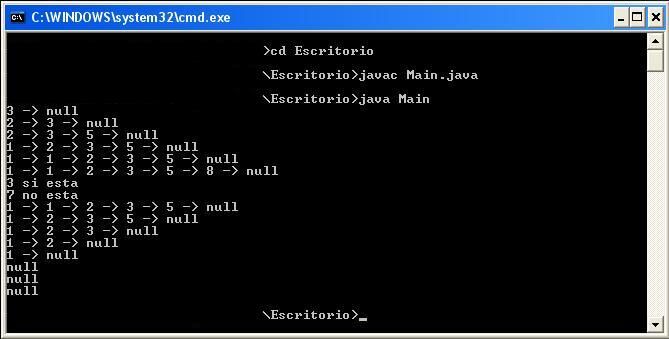
**Listas doblemente enlazadas o doblemente ligadas**

Hola, en este post les voy a presentar una estructura de datos llamada **listas doblemente enlazadas** y vamos a construir el código pasito a pasito (a ver quien resiste).  
  
Una lista doblemente enlazada es simplemente un conjunto de elementos o datos que aparecen unodetrás de otro (al igual que en una lista simplemente ligada o enlazada). En la siguiente imagen se observa una lista doblemente enlazada (LDE a partir de ahora) con tres datos numéricos simples (números enteros): 12, 57 y 95.



Como se puede ver de la imagen anterior una lista está compuesto de **nodos**. Cada nodo tendrá tres campos:  
  
- información o dato  
- anterior  
- siguiente  
  
Ahora sí, podemos representar un nodo mediante una clase en Java. A esta clase la llamaremos **Nodo**. A continuación el código de la clase Nodo.  
  
**public** **class** Nodo {  
    *//Campos del nodo*  
    **int** informacion;  
    Nodo anterior;  
    Nodo siguiente;  
  
    *//constructor que inicializa un Nodo con cierta informacion o dato*  
    **public** Nodo(**int** dato) {  
        informacion = dato;  
        anterior = null;  
        siguiente = null;  
    }  
}  
  
Ahora que tenemos la implementación del nodo podemos pasar a implementar la LDE. Para implementar una LDE vamos a nombrar al primer nodo **cabeza** haciendolo un nodo con nombre propio (esta es sólo una opción pues otras personas prefieren usar una referencia al primer nodo en lugar de hacer especial el primer nodo). Además, nombraremos al último nodo **fin**. Por lo tanto tenemos los siguientes casos:  
  
- Si la lista está vacía (no hay ningún elemento) no existirán los nodos cabeza ni fin y por lo tanto seránnull.  
- Si solamente hay un nodo en la lista esta será cabeza y fin a la misma vez. Por ejemplo, la siguiente imagen es una lista en donde hay un solo dato (15), el nodo cabeza tiene como dato 15 y comosiguiente y anterior igual a null y el nodo fin también.



A continuación se muestra el código que esquematizará a una LDE  
  
**public** **class** ListaDoblementeEnlazada {  
    Nodo cabeza;  
    Nodo fin;  
  
    *//constructor que crea una LDE vacia.*  
    **public** ListaDoblementeEnlazada() {  
        cabeza = null;  
        fin = null;  
    }  
}  
  
Ahora debemos implementar las operaciones que se han de realizar en la LDE (esto va a ocupar muuuchas líneas). Algunas de estas operaciones son  
  
- **Insertar al frente:** Inserta un nodo delante del actual nodo cabeza (en este caso, 'cabeza' se actualiza con el nuevo nodo).  
  
- **Insertar al final:** Inserta un nodo al final de la lista, es decir, insertar detrás del nodo 'fin' actualizándolo con el nuevo nodo.  
  
- **Eliminar del frente:** Elimina el nodo del frente ( 'cabeza' ) y actualiza 'cabeza' con el nodo que le sigue en la lista.  
  
- **Eliminar del final:** Elimina el nodo final ( 'fin' ) y lo actualiza con el nodo que lo antecedía.  
  
- **Buscar:** Busca un dato en la lista y si lo encuentra devuelve una referencia al nodo buscado, si no lo encuentra devuelve null.  
  
Existen otras operaciones dependiendo de la necesidad que se tenga. Por ejemplo, se puede eliminar un elemento dentro de la lista como el quinto elemento.   
  
Antes de implementar estas operaciones vamos a crear un método de la lista que nos indique si la lista está vacía o no.  
  
Para saber si una lista está vacía o no es averiguando si el nodo cabeza es null o no. Si el nodo cabeza es null la lista estará vacía. A continuación el código de este método dentro de la claseListaDoblementeEnlazada (en color rojo).  
  
**public** **class** ListaDoblementeEnlazada {  
    Nodo cabeza;  
    Nodo fin;  
  
    *//constructor que crea una LDE vacia.*  
    **public** ListaDoblementeEnlazada() {  
        cabeza = null;  
        fin = null;  
    }  
  
    *//indica si la lista está vacia*  
    **private** **boolean** estaVacia() {  
        **boolean** vacia = false;  
        if ( cabeza == null ) {  
            vacia = true;  
        }  
        **return** vacia;  
    }  
}  
  
Ahora sí, vamos a las operaciones mencionadas.  
  
**Insertar al frente**  
  
Tenemos dos casos: La lista está o no vacía. Esto se puede saber mediante el método estaVacia()implementado arriba.  
  
**a)** Si la lista está vacía simplemente nombramos el nuevo nodo como 'cabeza' y 'fin'. Esto se puede hacer con:  
cabeza = nuevo;  
fin = nuevo;  
  
**b)** Si la lista no está vacía entonces seguimos los siguientes pasos:  
  
**paso 1.** Hacemos que el nuevo nodo apunte a 'cabeza' como su siguiente nodo. Esto se puede hacer con:  
nuevo.siguiente = cabeza;  
  
**paso 2.** Hacemos que el nodo 'cabeza' apunte al nuevo nodo como su anterior nodo. Esto se puede hacer con:  
cabeza.anterior = nuevo;  
  
Al **final** nombramos al nuevo nodo como 'cabeza'. Esto se puede hacer con:  
cabeza = nuevo;  
  
La siguiente imagen muestra los pasos descritos  
  
  
  
A los pasos 1 y 2 le vamos a llamar **enlazar**. Así creamos un método enlazar "utilitario" al igual que el método buscar (en color rojo).  
  
**public** **class** ListaDoblementeEnlazada {  
    Nodo cabeza;  
    Nodo fin;  
  
    *//constructor que crea una LDE vacia.*  
    **public** ListaDoblementeEnlazada() {  
        cabeza = null;  
        fin = null;  
    }  
  
    *//indica si la lista está vacia*  
    **private** **boolean** estaVacia() {  
        **boolean** vacia = false;  
        if ( cabeza == null ) {  
            vacia = true;  
        }  
        **return** vacia;  
    }  
  
    *//enlaza dos nodos mediante enlace doble*  
    **private** **void** enlazar(Nodo nodoA, Nodo nodoB) {  
        nodoA.siguiente = nodoB;  
        nodoB.anterior = nodoA;  
    }  
}  
  
A continuación se muestra el código del método insertarInicio(int ndato) que realiza la operación insertar al inicio.  
  
**public** **class** ListaDoblementeEnlazada {  
    Nodo cabeza;  
    Nodo fin;  
  
    *//constructor que crea una LDE vacia.*  
    **public** ListaDoblementeEnlazada() {  
        cabeza = null;  
        fin = null;  
    }  
  
    *//indica si la lista está vacia*  
    **private** **boolean** estaVacia() {  
        **boolean** vacia = false;  
        if ( cabeza == null ) {  
            vacia = true;  
        }  
        **return** vacia;  
    }  
  
    *//enlaza dos nodos mediante enlace doble*  
    **private** **void** enlazar(Nodo nodoA, Nodo nodoB) {  
        nodoA.siguiente = nodoB;  
        nodoB.anterior = nodoA;  
    }  
  
    *//inserta un nuevo nodo al inicio de la lista*  
    **public** **void** insertarInicio(**int** ndato) {  
        Nodo nuevo = **new** Nodo(ndato);  
        **if** ( estaVacia() ) {  
            cabeza = nuevo;  
            fin = nuevo;  
        } **else** {  
            enlazar(nuevo, cabeza);  
            cabeza = nuevo;  
        }  
    }  
}  
  
**Insertar al final**  
  
Los pasos son muy similares a la operación insertar al inicio. El código se muestra a continuación en color rojo  
  
**public** **class** ListaDoblementeEnlazada {  
    Nodo cabeza;  
    Nodo fin;  
  
    *//constructor que crea una LDE vacia.*  
    **public** ListaDoblementeEnlazada() {  
        cabeza = null;  
        fin = null;  
    }  
  
    *//indica si la lista está vacia*  
    **private** **boolean** estaVacia() {  
        **boolean** vacia = false;  
        if ( cabeza == null ) {  
            vacia = true;  
        }  
        **return** vacia;  
    }  
  
    *//enlaza dos nodos mediante enlace doble*  
    **private** **void** enlazar(Nodo nodoA, Nodo nodoB) {  
        nodoA.siguiente = nodoB;  
        nodoB.anterior = nodoA;  
    }  
  
    *//inserta un nuevo nodo al inicio de la lista*  
    **public** **void** insertarInicio(**int** ndato) {  
        Nodo nuevo = **new** Nodo(ndato);  
        **if** ( estaVacia() ) {  
            cabeza = nuevo;  
            fin = nuevo;  
        } **else** {  
            enlazar(nuevo, cabeza);  
            cabeza = nuevo;  
        }  
    }  
  
    *//inserta un nuevo nodo al final de la lista*  
    **public** **void** insertarFinal(**int** ndato) {  
        Nodo nuevo = **new** Nodo(ndato);  
        **if** ( estaVacia() ) {  
            cabeza = nuevo;  
            fin = nuevo;  
        } **else** {  
            enlazar(fin, nuevo);  
            fin = nuevo;  
        }  
    }  
}  
  
**Eliminar del frente**  
  
Tenemos dos casos:  
  
**a)** Si la lista está vacía entonces no se hace nada.  
  
**b)** Si la lista no está vacía seguimos los siguientes pasos  
  
**paso1.** Referenciamos al nodo después de 'cabeza' como 'primero' y hacemos que 'primero' apunte apunte a 'null' como anterior. Esto se puede hacer con:  
primero = cabeza.siguiente;  
primero.anterior = null;  
  
**Peligro:** Es posible que la lista contenga un solo elemento, en ese caso 'primero' toma el valor de 'null' y la eliminación deja la lista vacía por lo que haciendo una simple verificación de si 'primero' es igual a 'null' podemos hacer la eliminación como sigue  
if (primero == null) {  
    cabeza = null;  
    fin = null;  
}  
  
**paso2.** Cambiamos de nombre a 'primero' para que sea 'cabeza'. Esto se hace con:  
cabeza = primero;  
  
Puesto que ya no existen referencias apuntando al nodo que antes era 'cabeza' el recolector de basura de Java lo eliminará, es decir, ya no debemos preocuparnos por ese nodo.  
  
La siguiente imagen muestra los pasos descritos.  
  
  
  
El código del método eliminarInicio se muestra a continuación en color rojo  
  
**public** **class** ListaDoblementeEnlazada {  
    Nodo cabeza;  
    Nodo fin;  
  
    *//constructor que crea una LDE vacia.*  
    **public** ListaDoblementeEnlazada() {  
        cabeza = null;  
        fin = null;  
    }  
  
    *//indica si la lista está vacia*  
    **private** **boolean** estaVacia() {  
        **boolean** vacia = false;  
        if ( cabeza == null ) {  
            vacia = true;  
        }  
        **return** vacia;  
    }  
  
    *//enlaza dos nodos mediante enlace doble*  
    **private** **void** enlazar(Nodo nodoA, Nodo nodoB) {  
        nodoA.siguiente = nodoB;  
        nodoB.anterior = nodoA;  
    }  
  
    *//inserta un nuevo nodo al inicio de la lista*  
    **public** **void** insertarInicio(**int** ndato) {  
        Nodo nuevo = **new** Nodo(ndato);  
        **if** ( estaVacia() ) {  
            cabeza = nuevo;  
            fin = nuevo;  
        } **else** {  
            enlazar(nuevo, cabeza);  
            cabeza = nuevo;  
        }  
    }  
  
    *//inserta un nuevo nodo al final de la lista*  
    **public** **void** insertarFinal(**int** ndato) {  
        Nodo nuevo = **new** Nodo(ndato);  
        **if** ( estaVacia() ) {  
            cabeza = nuevo;  
            fin = nuevo;  
        } **else** {  
            enlazar(fin, nuevo);  
            fin = nuevo;  
        }  
    }  
  
    *//elimina el nodo del frente de la lista*  
    **public** **void** eliminarInicio() {  
        **if** ( !estaVacia() ) {  
            Nodo primero = cabeza.siguiente;  
            **if** ( primero == null ) {  
                cabeza = null;  
                fin = null;  
            } **else** {  
                primero.anterior = null;  
                cabeza = primero;  
            }  
        }  
    }  
}  
  
**Eliminar del final**  
  
Esta operación es similar al caso anterior. A continuación el código en color rojo:  
  
**public** **class** ListaDoblementeEnlazada {  
    Nodo cabeza;  
    Nodo fin;  
  
    *//constructor que crea una LDE vacia.*  
    **public** ListaDoblementeEnlazada() {  
        cabeza = null;  
        fin = null;  
    }  
  
    *//indica si la lista está vacia*  
    **private** **boolean** estaVacia() {  
        **boolean** vacia = false;  
        if ( cabeza == null ) {  
            vacia = true;  
        }  
        **return** vacia;  
    }  
  
    *//enlaza dos nodos mediante enlace doble*  
    **private** **void** enlazar(Nodo nodoA, Nodo nodoB) {  
        nodoA.siguiente = nodoB;  
        nodoB.anterior = nodoA;  
    }  
  
    *//inserta un nuevo nodo al inicio de la lista*  
    **public** **void** insertarInicio(**int** ndato) {  
        Nodo nuevo = **new** Nodo(ndato);  
        **if** ( estaVacia() ) {  
            cabeza = nuevo;  
            fin = nuevo;  
        } **else** {  
            enlazar(nuevo, cabeza);  
            cabeza = nuevo;  
        }  
    }  
  
    *//inserta un nuevo nodo al final de la lista*  
    **public** **void** insertarFinal(**int** ndato) {  
        Nodo nuevo = **new** Nodo(ndato);  
        **if** ( estaVacia() ) {  
            cabeza = nuevo;  
            fin = nuevo;  
        } **else** {  
            enlazar(fin, nuevo);  
            fin = nuevo;  
        }  
    }  
  
    *//elimina el nodo del frente de la lista*  
    **public** **void** eliminarInicio() {  
        **if** ( !estaVacia() ) {  
            Nodo primero = cabeza.siguiente;  
            **if** ( primero == null ) {  
                cabeza = null;  
                fin = null;  
            } **else** {  
                primero.anterior = null;  
                cabeza = primero;  
            }  
        }  
    }  
  
    *//elimina el nodo del final de la lista*  
    **public** **void** eliminarFinal() {  
        **if** ( !estaVacia() ) {  
            Nodo ultimo = fin.anterior;  
            **if** ( ultimo == null ) {  
                cabeza = null;  
                fin = null;  
            } **else** {  
                ultimo.siguiente = null;  
                fin = ultimo;  
            }  
        }  
    }  
}  
  
**Buscar**  
  
Esta es simple. Recorremos la lista nodo por nodo hasta encontrar el buscado. Declaramos dos referencias 'Nodo' 'buscado' e 'iterador'. 'buscado' será el nodo que será devuelto si se encuentra lo que se busca. Con 'iterador' vamos a recorrer todos los nodos de la lista uno por uno empezando por 'cabeza'. Este efecto se consigue con la sentencia:  
  
Nodo iterador = cabeza;  
while ( iterador != null ) {  
    //... codigo ...  
    iterador = iterador.siguiente;  
}  
  
En cada iteración debemos comparar el valor actual de 'informacion' de 'iterador' con el valor buscado ( 'dato' ). Si se ha encontrado entonces se actualiza 'buscado' con el valor actual de iterador y se debe romper el 'while'. Esto se puede hacer con un break, sin embargo, se hará insertando una condición más al 'while'. Cuando 'buscado' es diferente de null entonces debe parar el while. El siguiente código muestra el método buscar en color rojo:  
  
**public** **class** ListaDoblementeEnlazada {  
    Nodo cabeza;  
    Nodo fin;  
  
    *//constructor que crea una LDE vacia.*  
    **public** ListaDoblementeEnlazada() {  
        cabeza = null;  
        fin = null;  
    }  
  
    *//indica si la lista está vacia*  
    **private** **boolean** estaVacia() {  
        **boolean** vacia = false;  
        if ( cabeza == null ) {  
            vacia = true;  
        }  
        **return** vacia;  
    }  
  
    *//enlaza dos nodos mediante enlace doble*  
    **private** **void** enlazar(Nodo nodoA, Nodo nodoB) {  
        nodoA.siguiente = nodoB;  
        nodoB.anterior = nodoA;  
    }  
  
    *//inserta un nuevo nodo al inicio de la lista*  
    **public** **void** insertarInicio(**int** ndato) {  
        Nodo nuevo = **new** Nodo(ndato);  
        **if** ( estaVacia() ) {  
            cabeza = nuevo;  
            fin = nuevo;  
        } **else** {  
            enlazar(nuevo, cabeza);  
            cabeza = nuevo;  
        }  
    }  
  
    *//inserta un nuevo nodo al final de la lista*  
    **public** **void** insertarFinal(**int** ndato) {  
        Nodo nuevo = **new** Nodo(ndato);  
        **if** ( estaVacia() ) {  
            cabeza = nuevo;  
            fin = nuevo;  
        } **else** {  
            enlazar(fin, nuevo);  
            fin = nuevo;  
        }  
    }  
  
    *//elimina el nodo del frente de la lista*  
    **public** **void** eliminarInicio() {  
        **if** ( !estaVacia() ) {  
            Nodo primero = cabeza.siguiente;  
            **if** ( primero == null ) {  
                cabeza = null;  
                fin = null;  
            } **else** {  
                primero.anterior = null;  
                cabeza = primero;  
            }  
        }  
    }  
  
    *//elimina el nodo del final de la lista*  
    **public** **void** eliminarFinal() {  
        **if** ( !estaVacia() ) {  
            Nodo ultimo = fin.anterior;  
            **if** ( ultimo == null ) {  
                cabeza = null;  
                fin = null;  
            } **else** {  
                ultimo.siguiente = null;  
                fin = ultimo;  
            }  
        }  
    }  
  
    *//devuelve una referencia al nodo buscado, si no se encuentra devuelve null*  
    **public** Nodo buscar(**int** dato) {  
        Nodo buscado = null;  
        Nodo iterador = cabeza;  
        **while** ( buscado == null && iterador != null ) {  
            **if** ( iterador.informacion == dato ) {  
                buscado = iterador;  
            }  
            iterador = iterador.siguiente;  
        }  
        **return** buscado;  
    }  
}  
  
**Por último** vamos a crear un método para mostrar la lista. para esto vamos a recorrer la lista nodo por nodo como se hizo en el método buscar sólo que en lugar de hacer comparaciones vamos a mostrar cada elemento. A continuación se muestra el código del método mostrar en color rojo. Este código es el final de nuestra clase ListaDoblementeEnlazada:  
  
**public** **class** ListaDoblementeEnlazada {  
    Nodo cabeza;  
    Nodo fin;  
  
    *//constructor que crea una LDE vacia.*  
    **public** ListaDoblementeEnlazada() {  
        cabeza = null;  
        fin = null;  
    }  
  
    *//indica si la lista está vacia*  
    **private** **boolean** estaVacia() {  
        **boolean** vacia = false;  
        if ( cabeza == null ) {  
            vacia = true;  
        }  
        **return** vacia;  
    }  
  
    *//enlaza dos nodos mediante enlace doble*  
    **private** **void** enlazar(Nodo nodoA, Nodo nodoB) {  
        nodoA.siguiente = nodoB;  
        nodoB.anterior = nodoA;  
    }  
  
    *//inserta un nuevo nodo al inicio de la lista*  
    **public** **void** insertarInicio(**int** ndato) {  
        Nodo nuevo = **new** Nodo(ndato);  
        **if** ( estaVacia() ) {  
            cabeza = nuevo;  
            fin = nuevo;  
        } **else** {  
            enlazar(nuevo, cabeza);  
            cabeza = nuevo;  
        }  
    }  
  
    *//inserta un nuevo nodo al final de la lista*  
    **public** **void** insertarFinal(**int** ndato) {  
        Nodo nuevo = **new** Nodo(ndato);  
        **if** ( estaVacia() ) {  
            cabeza = nuevo;  
            fin = nuevo;  
        } **else** {  
            enlazar(fin, nuevo);  
            fin = nuevo;  
        }  
    }  
  
    *//elimina el nodo del frente de la lista*  
    **public** **void** eliminarInicio() {  
        **if** ( !estaVacia() ) {  
            Nodo primero = cabeza.siguiente;  
            **if** ( primero == null ) {  
                cabeza = null;  
                fin = null;  
            } **else** {  
                primero.anterior = null;  
                cabeza = primero;  
            }  
        }  
    }  
  
    *//elimina el nodo del final de la lista*  
    **public** **void** eliminarFinal() {  
        **if** ( !estaVacia() ) {  
            Nodo ultimo = fin.anterior;  
            **if** ( ultimo == null ) {  
                cabeza = null;  
                fin = null;  
            } **else** {  
                ultimo.siguiente = null;  
                fin = ultimo;  
            }  
        }  
    }  
  
    *//devuelve una referencia al nodo buscado, si no se encuentra devuelve null*  
    **public** Nodo buscar(**int** dato) {  
        Nodo buscado = null;  
        Nodo iterador = cabeza;  
        **while** ( buscado == null && iterador != null ) {  
            **if** ( iterador.informacion == dato ) {  
                buscado = iterador;  
            }  
            iterador = iterador.siguiente;  
        }  
        **return** buscado;  
    }  
  
    *//muestra los valores en la lista*  
    **public** **void** mostrar() {  
        Nodo iterador = cabeza;  
        **while** ( iterador != null ) {  
            System.out.print( iterador.informacion + " -> " );  
            iterador = iterador.siguiente;  
        }  
        System.out.println( "null" );  
    }  
}  
  
**A probar nuestro código!**  
  
Nuestra clase principal será:  
  
**public** **class** Main {  
    **public** **static** **void** main(String[] args) {  
        ListaDoblementeEnlazada lista = **new** ListaDoblementeEnlazada();  
        lista.insertarInicio(3);  
        lista.mostrar();  
        lista.insertarInicio(2);  
        lista.mostrar();  
        lista.insertarFinal(5);  
        lista.mostrar();  
        lista.insertarInicio(1);  
        lista.mostrar();  
        lista.insertarInicio(1);  
        lista.mostrar();  
        lista.insertarFinal(8);  
        lista.mostrar();  
        Nodo a = lista.buscar(3);  
        System.out.println((a != null)? "3 si esta" : "3 no esta" );  
        Nodo b = lista.buscar(7);  
        System.out.println((b != null)? "7 si esta" : "7 no esta" );  
        lista.eliminarFinal();  
        lista.mostrar();  
        lista.eliminarInicio();  
        lista.mostrar();  
        lista.eliminarFinal();  
        lista.mostrar();  
        lista.eliminarFinal();  
        lista.mostrar();  
        lista.eliminarFinal();  
        lista.mostrar();  
        lista.eliminarFinal();  
        lista.mostrar();  
        lista.eliminarFinal();  
        lista.mostrar();  
        lista.eliminarFinal();  
        lista.mostrar();  
    }  
}  
  
Para compilarlo debemos tener las 3 clases juntas:  
  
- Nodo (Nodo.java)  
- ListaDoblementeEnlazada (ListaDoblementeEnlazada.java)  
- Main (Main.java)  
  
  
  
La salida del programa será:  
  
  
  
Bueno. Eso fue todo, espero que esta guía le ayude a alguien.