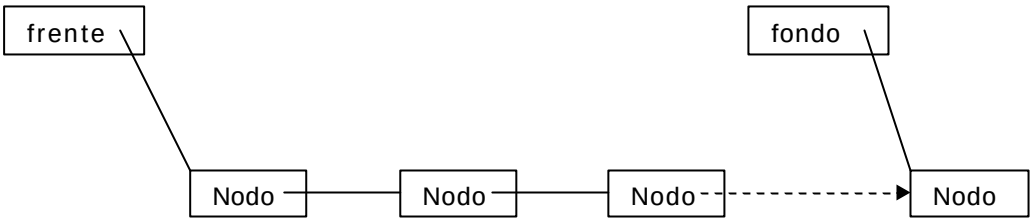
Implementando pilas, colas y listas

Usaremos, para todo, una única estructura de nodos enlazados.



Nuestra clase Nodo

```
import Item;
class Nodo{    // Nodo de una estructura enlazada simple
        protected Item item; //  Nodo tiene un Item y
        protected Nodo prox;  //  una referencia al próximo Nodo...

        public Nodo(){              // Constructor default
                this(0, 0, null);
        }

        public Nodo(int cod, float val, Nodo sig) {// Constructor parametrico
                item = new Item();
                item.setCodigo(cod);
                item.setValor(val);
                prox = sig;
        }

        public Item getItem(){return item;}     // Retornamos el objeto item
                                                // almacenado en Nodo

        public void setItem(Item it) {item = it;};      // Inicializamos idem

        public Nodo getProx(){return prox;} // Retornamos puntero al proximo nodo

        public void setProx(Nodo ptr){prox = ptr;}// Inicilizamos puntero prox

        public String toString(){ //       Exhibimos el contenido de Nodo
                String aux = ""+item.toString()+"\n";   // Usamos toString() de Item
                return aux;
        }
}
```

La clase NodosEnl

```
import Nodo;
class NodosEnl{    //      Nodos enlazados mediante referencias
        protected Nodo frente, fondo;  //  Referencia de entrada
        protected boolean trace; // Trazamos ?
        public NodosEnl(){
                frente = null; trace = false;}   //   Un constructor sin argumentos;
        public boolean vacia(){return frente == null;} // No hay nodos
        public String toString(){
                String aux = "NodosEnl contiene \n";
                for(Nodo ptr = frente; ptr!=null; ptr=ptr.getProx())
                        aux+= ptr.toString();
                return aux;
        }
        public void setTrace(boolean trazar){trace = trazar;}
        public void inclFirst(int cod, float val){
                frente = new Nodo(cod,val,frente);
                if(frente.getProx() == null) // Primer nodo
                        fondo = frente;
                if(trace)
                        System.out.print("incluido "+frente);
        }
        public Nodo exclFirst(){
                if(vacia()) return null; // Retornamos referencia nula
                Nodo ptr = frente;   // Guardamos en aux la referencia del primer nodo
                frente = frente.getProx(); // Redirecionamos frente, ➔ siguiente nodo
                if(frente == null)     //      No hay ya nodos ...
                        fondo = null;
                if (trace)
                        System.out.print("excluido "+ptr);
                return ptr;                     //      Retornamos referencia nodo liberado
        }
        public void demo(){
                System.out.println("Demo class NodosEnl");
                setTrace(true);
                inclFirst(10,20);
                inclFirst(15,15);
                inclFirst(20,15);
                exclFirst();
                System.out.print(this);
                System.out.println("Demo NodosEnl finished !!!");
        }
        public static void main(String args[]){
                NodosEnl obj = new NodosEnl();
                obj.demo();
        }
}
```

```
Demo class NodosEnl

incluido 10 - 20.0
incluido 15 - 15.0
incluido 20 - 15.0
excluido 20 - 15.0
NodosEnl contiene
15 - 15.0
10 - 20.0
Demo NodosEnl finished !!!

Process Exit...
```

La clase Pila

```
import NodosEnl;
class Pila extends NodosEnl{ //  Implementando Pila a partir de NodosEnl

        public Pila(){
                super();
        }

        public void demo(){
                System.out.println("Demo class Pila");
                setTrace(true);
                for(int i= 1;i<=5;i++)
                        inclFirst(i,(float)i);
                for(int i = 1;i<=3;i++)
                        exclFirst();
                System.out.print(this);
                System.out.println("Demo Pila finished !!!");
        }

        public static void main(String args[]){
                Pila pila = new Pila();
                pila.demo();
        }
}
```

```
Demo class Pila

incluido 1 - 1.0
incluido 2 - 2.0
incluido 3 - 3.0
incluido 4 - 4.0
incluido 5 - 5.0
excluido 5 - 5.0
excluido 4 - 4.0
excluido 3 - 3.0
NodosEnl contiene
2 - 2.0
1 - 1.0
Demo Pila finished !!!

Process Exit...
```

La clase Cola

```
import Pila;
class Cola extends Pila {  //  Implementando Cola a partir de Pila
        public Cola(){              //      Un constructor sin argumentos
                super();
        }
        public void inclLast(int cod,float val){
                Nodo aux;
                if (frente == null){ //       no teníamos nodos, este es el primero
                        frente = aux = new Nodo(cod,val,null);
                        fondo = frente;
                }else{          //  Ya teniamos, este es uno mas ...
                        aux = new Nodo(cod,val,null);
                        fondo.setProx(aux);
                        fondo = aux;
                }
                if(trace)
                        System.out.print("incluido "+aux);
        }
        public void demo(){
                System.out.println("Demo class Cola");
                setTrace(true);
                for(int i= 1;i<=5;i++)                  // Incluyo
                        inclLast(i,(float)i);           // 5 nodos
                for(int i = 1;i<=3;i++)                 // Excluyo
                        exclFirst();                    // solo 3 nodos
                for(int i= 1;i<=2;i++)                  // Incluyo
                        inclLast(i+10,(float)i+10);     // mas 2 nodos
                System.out.print(this);
                for(int i = 1;fondo!=null;i++)          // excluyo
                        exclFirst();                    // Toda la cola
                System.out.println("Demo Cola finished !!!");
        }
        public static void main(String args[]){
                Cola cola = new Cola();
                cola.demo();
        }
}
```

```
Demo class Cola

incluido 1 - 1.0
incluido 2 - 2.0
incluido 3 - 3.0
incluido 4 - 4.0
incluido 5 - 5.0
excluido 1 - 1.0
excluido 2 - 2.0
excluido 3 - 3.0
incluido 11 - 11.0
incluido 12 - 12.0
NodosEnl contiene
4 - 4.0
5 - 5.0
11 - 11.0
12 - 12.0
excluido 4 - 4.0
excluido 5 - 5.0
excluido 11 - 11.0
excluido 12 - 12.0
Demo Cola finished !!!
Process Exit...
```

La clase Lista

```
import NodosEnl;
class Lista extends Pila { //  Implementando Lista a partir de Pila
        private Nodo shadow;

        public Lista(){
                super();
                shadow = null;
        }

        protected boolean existe(int codPar){
                boolean exis = false;                   // A priori, suponemos no existe
                Nodo ptr = frente; // Puntero para recorrer la lista.
                Nodo sombra = frente; // Puntero sombra, irá siempre atras de ptr
                int codNod;
                if (vacia())        //  La lista no tiene nodos
                        shadow = null;
                else{               //  Tenemos una lista con nodos
                        for(;ptr!=null;sombra = ptr, ptr = ptr.getProx()){
                                codNod = ptr.getItem().getCodigo();
                                if(codNod == codPar){ // hemos encontrado igual, basta
                                                      // de buscar ...
                                        exis = true; // existencia verdadera ...
                                        if(ptr == frente)       // ptr apunta al primer nodo,
                                                shadow = null; // código parámetro es ==
                                                               // que el del 1er nodo
                                        else    // ptr apunta a un nodo intermedio o final,
                                                shadow = sombra;// Referenciamos la
                                                                //  posicion del nodo anterior
                                        break;
                                }     //  if(codNod == codPar)
                                if(codNod > codPar){ // ya no vamos a encontrar igual,
                                                     // basta de nuevo
                                        if(ptr == frente)       // ptr apunta al primer nodo,
                                                shadow = null; // código parámetro es < que
                                                               // el del 1er nodo
                                        else    // ptr apunta a un nodo intermedio o final,
                                                shadow = sombra;// Referenciamos la
                                                                // posicion del nodo anterior
                                        break;
                                }     // if(codNod > codPar
                        } // for
                        if(ptr == null)  // Recorrimos lista completa, codPar es mayor al
                                         // del último nodo
                                shadow = sombra; //  posicion del último nodo
                }       // else if (vacia())
                return exis;
        }
```

```
/* el método existe() tiene una doble funcionalidad:
        1) retorna
                true si el  código del item parámetro existe
                false en caso contrario

        2) inicializa la referencia shadow

                        shadow = null cuando:
                                la lista está vacia
                                el código del item parámetro es menor al del primer nodo
                                el codigo parametro = codigo nodo en el primer nodo

                        shadow = sombra (puntero que sique "pegado" a ptr)
                                el código nodo  = código parámetro, no en el primer nodo
                                el código nodo  > código parámetro
                                el código parámetro > al del último nodo        */

private void inclOrd(int codPar, float vlr){ // Inserta nodos. Lista ordenada
        Nodo ptr;
        boolean exist = existe(codPar);
        if(exist){      // ya tenemos ese código, no podemos re incluirlo
                if(trace)System.out.println("inclOrd(), codigo " + codPar + " ya existe");
        }else{          // No existe el código del item parámetro, podemos incluir
                        if(shadow != null){   // es una inserción intermedia o al fin
                                ptr = shadow.getProx(); // puntero al próximo
                                shadow.setProx(new Nodo(codPar,vlr,ptr));
                                if(trace) System.out.println("inclOrd() " + codPar + "
                                                                        incluido");
                        }else  // Es una inserción al inicio de la lista o la lista está vacía
                                inclFirst(codPar, vlr);// usamos el método heredado
        }   // else del if(exist)
}

protected Nodo exclOrd(int codigo) {  //Excluye nodos.Lista ordenada por código
        Nodo ptr;               // para puntero al nodo
        boolean exist = existe(codigo);  // Verificamos su existencia
        Nodo nodo = null;               // Lo necesitamos para almacenar el nodo excluido
        if(exist)       // Tenemos igual codigo en lista, entonces podemos excluir
                if(shadow != null){     // Debemos excluir un nodo intermedio o final
                        nodo = shadow.getProx(); // puntero al nodo a excluir
                        ptr  = nodo.getProx();          // puntero al siguiente al excluido
                        shadow.setProx(ptr);// Vinculamos anterior al siguiente al excl.
                        if(trace) System.out.print("exclOrd(), codigo " + codigo + "
                                                                        excluido");
                }else                           // El nodo a excluir es el primero
                        nodo = exclFirst();   // usamos el método adecuado, el heredado
                if(trace) System.out.print("exclFirst(), codigo " + codigo + " excluido");
        else                            // No existe el item buscado
                if(trace) System.out.print("exclOrd(), codigo " + codigo + " no existe");
        return nodo;                    // retornamos referencia al nodo excluido (o no)
}
```

```
    protected void actValor(int codigo, float val){ // Actualiza valor del item del nodo.
        Nodo ptr;                           // para puntero al nodo
        boolean exist = existe(codigo);      // Verificamos su existencia
        Nodo ptrAct;        // Puntero al nodo cuyo valor de item actualizaremos
        if(exist){      // Tenemos código en lista, podemos actualizar valor asociado
            if(shadow != null)      // Actualizar en un nodo intermedio o final
                ptrAct = shadow.getProx(); // ptrAct con la dirección correcta
            else            // El valor a actualizar está en el item del primer nodo
                ptrAct = frente;
            //  ptrAct apunta al nodo cuyo valor queremos actualizar
            Item it = new Item(codigo,val); // Generamos objeto item
            ptrAct.setItem(it);           // reemplazar el del nodo
            if(trace) System.out.print("actValor(), código " + codigo + " a " + val);
        }else                               // No existe el item buscado
            if(trace)System.out.print("actValor(), código " + codigo + " no existe");
    }

    protected void implDemo1(){
        System.out.println("implDemo1() class Lista");
        System.out.println("Incluimos 4 nodos - inclFirst()");
        for(int i=4;i>0;--i)inclFirst(i*2,(float)(i*4));
        System.out.println(this);
        System.out.println("Borramos el primero");
        exclFirst();
        System.out.println(this);
        System.out.println("Demo implDemo1() terminado !!!");
    }

    public static void main(String args[]){
        Lista list = new Lista();
        list.implDemo1();
    }
} //  Class Lista
```

```
implDemo1() class Lista
Incluimos 4 nodos - inclFirst()

NodosEnl contiene
2 - 4.0
4 - 8.0
6 - 12.0
8 - 16.

Borramos el primero

NodosEnl contiene
4 - 8.0
6 - 12.0
8 - 16.0

Demo implDemo1() terminado !!!
Process Exit...
```

```
protected void implDemo2(){
        System.out.println("implDemo2() class Lista");
        System.out.println("Incluimos 4 nodos - inclOrd()");
        for(int i=1;i<5;++i)inclOrd(i*2+1,(float)(i*4.2));
        System.out.println(this);
        System.out.println("Borramos el primero");
        exclFirst();
        System.out.println("Borramos el código 7");
        exclOrd(7);
        System.out.println(this);
        System.out.println("Demo implDemo2() terminado !!!");
}

public static void main(String args[]){
        Lista list = new Lista();
        list.implDemo 2();
}
```

```
implDemo2() class Lista
Incluimos 4 nodos - inclOrd()
NodosEnl contiene
3 - 4.2
5 - 8.4
7 - 12.6
9 - 16.8

Borramos el primero
Borramos el código 7

NodosEnl contiene
5 - 8.4
9 - 16.8
Demo implDemo2() terminado !!!
Process Exit...
```

```
protected void implDemo3(){
        System.out.println("implDemo3() class Lista");
        System.out.println("Incluimos 4 nodos - inclFirst()");
        for(int i=4;i>1;--i)inclFirst(i*2,(float)(i*4));
        System.out.println(this);
        System.out.println("Incluimos 3 nodos - inclOrd()");
        for(int i=1;i<5;++i)inclOrd(i*2+1,(float)(i*4.2));
        System.out.println(this);
        System.out.println("Demo implDem
}
public static void main(String args[]){
        Lista list = new Lista();
        list.implDemo 3();
}
```

```
implDemo3() class Lista
Incluimos 3 nodos - inclFirst()
NodosEnl contiene
4 - 8.0
6 - 12.0
8 - 16.0
Incluimos 4 nodos - inclOrd()
NodosEnl contiene
3 - 4.2
4 - 8.0
5 - 8.4
6 - 12.0
7 - 12.6
8 - 16.0
9 - 16.8
Demo implDemo3() terminado !!!
Process Exit...
```

```
protected void implDemo4(){
        System.out.println("implDemo4() class Lista");
        System.out.println("Incluimos 5 nodos - inclOrd()");
        for(int i=1;i<6;++i)inclOrd(i*2+1,(float)(i*4.2));
        System.out.println(this);
        System.out.println("Borraremos el codigo 5");
        exclOrd(5);
        System.out.println("Borraremos el código 9");
        exclOrd(9);
        System.out.println("Borraremos el código 8 ");
        exclOrd(8);
        System.out.println(toString());
        System.out.println("Demo implDemo4() terminado !!!");
}

public static void main(String args[]){
        Lista list = new Lista();
        list.implDemo 4();
}
```

```
implDemo4() class Lista
Incluimos 5 nodos - inclOrd()
NodosEnl contiene
3 - 4.2
5 - 8.4
7 - 12.6
9 - 16.8
11 - 21.0
Borraremos el codigo 5
Borraremos el código 9
Borraremos el código 8
NodosEnl contiene
3 - 4.2
7 - 12.6
11 - 21.0
Demo implDemo4() terminado !!!
Process Exit...
```

```
protected void implDemo5(){
        System.out.println("implDemo5() class Lista");
        System.out.println("Incluiremos 3 nodos - inclOrd()");
        for(int i=1;i<4;++i)inclOrd(i*2+1,(float)(i*4.2));
        System.out.println(this);
        System.out.println("Actualizaremos el código 5");
        actValor(5,10);
        System.out.println("Actualizamos el código 3");
        actValor(3,20);
        System.out.println("Actualizamos el código 8");
        actValor(8,20);
        System.out.println(this);
        System.out.println("Demo implDemo5
}

public static void main(String args[]){
        Lista list = new Lista();
        list.implDemo 4();
}
```

```
implDemo5() class Lista
Incluiremos 3 nodos - inclOrd()
NodosEnl contiene
3 - 4.2
5 - 8.4
7 - 12.6
Actualizaremos el código 5
Actualizamos el código 3
Actualizaremos el código 8
NodosEnl contiene
3 - 20.0
5 - 10.0
7 - 12.6
Demo implDemo5()terminado !!!
Process Exit...
```