TDA Diccionario: lista ordenada de palabras con significados.

Operación	Significado
Object **buscar** (Comparable x)	devolver significado de palabra x (null si no existe)
boolean **agregar** (Comparable x,Object y) throws DiccLleno	agregar palabra x con significado y (false si ya existe)
boolean **borrar** (Comparable x)	borrar palabra x (devolver false si no existe)
boolean **cambiar** (Comparable x,Object y)	cambiar significado de palabra x por y (false si no existe)

```
interface Comparable{
  int compareTo(Comparable x);
}
```

Problema. Escribir un programa para el siguiente diálogo:

Button	*Button*	*Button*	*Button*	*Button*
buscar	agregar	borrar	cambiar	quit
palabra:		significado:		mensaje
Label	*TextField*	*Label*	*TextField*	*Label*

- mensaje: "Ok" o "explicación" en caso que la operación fracase
- programa se invoca: Java Dic X
- cada línea del archivo X contiene palabra:significado.

```
static protected Diccionario D=new Diccionario();
static public void main(String[]args)throws Exception{
BR a=new BR(new FR(args[0])); String linea;
while((linea=a.readLine())!=null){
  int i=linea.indexOf(":");
  D.agregar(linea.substring(0,i),linea.substring(i+1));
}
new Dic().show();
}
```

Propuesto: escribir método actionPerformed

TDA Diccionario: Implementación con arreglo ordenado

índice	palabra	significado
0		
...		
n-1		
...		
N-1		

```
class Registro
{
public Comparable palabra;
public Object significado;
public Registro(Comparable x,Object y){
   palabra=x; significado=y;
}
}
```

```
class Diccionario{
protected static final int N=100;
protected int n;
protected Registro[] reg;
public Diccionario(){
   reg=new Registro[N]; n=0;
}
public Object buscar(Comparable x){
   int i=indice(x);
   return i<0 ? null : reg[i].significado;
}
public boolean cambiar(Comparable x,Object y){
   int i=indice(x);
   if(i<0) return false;
   reg[i].significado = y;
   return true;
}
```

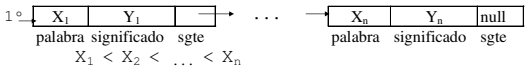
```
//búsqueda binaria: O(log2n)
protected int indice(Comparable x){
 //repetir mientras queden elementos
 int ip=0,iu=n-1;
 while(ip<=iu)
 {
   //comprobar si está justo en la mitad
   int im=(ip+iu)/2;
   int c=reg[im].palabra.compareTo(x);
   if(c==0) return im;
   //descartar la mitad del arreglo
   if(c<0) ip=im+1; else iu=im-1;
 }
 //entregar -1 si no está
 return -1;
}
```

```
//borrar palabra x: O(n)
public boolean borrar(Comparable x)
{
  //buscar palabra: O(log2n)
   int i=indice(x);
   if(i<0) return false;

  //"subir" elementos siguientes: O(n)
  for(int j=i;j<n-1;++j)
     reg[j]=reg[j+1];

  //decrementar nº de elementos
   --n;
  return true;
}
}
```

```
//agregar palabra x con significado y: O(n)
public boolean agregar(
Comparable x,Object y)
throws DiccLleno{
  //buscar palabra x: O(log2n)
   if(indice(x)>=0) return false; //O(log2n)
   if(n>=N) throw new DiccLleno();
  //"bajar" elementos: O(n)
   int i;
   for(i=n-1; i>=0 ;--i){
       if(reg[i].palabra.compareTo(x)<0) break;
       reg[i+1]=reg[i];
   }
   //incrementar elementos
  reg[i+1]=new Registro(x,y); ++n;
   return true;
}
```

Propuesto. Reescribir realizando sólo una pasada

Implementación con lista enlazada ordenada por palabra

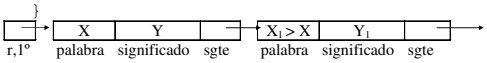



```
class Nodo{
public Comparable palabra;
public Object significado;
public Nodo sgte;
public Nodo(Comparable x,Object y,Nodo z){
   palabra=x;
   significado=y;
   sgte=z;
}
}
```

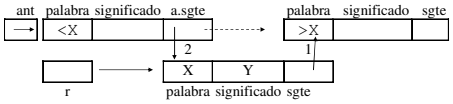
```
class Diccionario
{
protected Nodo primero;

public Diccionario(){
  primero=null;
}
public Object buscar(Comparable x){
   Nodo r=referencia(x,primero);
   return r==null ? null : r.significado;
}
public boolean cambiar(Comparable x,Object y){
   Nodo r=referencia(x,primero);
   if(r==null) return false;
   r.significado=y;
  return true;
}
```

```
//buscar nodo con palabra x (a partir de ref y)
protected Nodo referencia(Comparable x,Nodo y){
  for(Nodo r=y; r!=null; r=r.sgte){
    int c=x.compareTo(r.palabra);
    if(c==0) return r;
    if(c<0) break;
  }
  return null;//no está
}
//recursivo
protected Nodo referencia(Comparable x,Nodo y){
  if(y==null) return null;
  int c=x.compareTo(y.palabra);
  if(c==0) return y;
  if(c<0) return null;
  return referencia(x,y.sgte);
}
```

```
//agregar palabra x con significado y
public boolean agregar(Comparable x,Object y)
throws DiccLleno{
  //si palabra ya estaba, devolver false
  if(referencia(x,primero)!=null)return false;
  //crear nuevo nodo
  Nodo r=new Nodo(x,y,null);
  //si lista vacía o x<1ª, agregar al comienzo
  if(primero==null
  || x.compareTo(primero.palabra)<0){
    r.sgte=primero;
    primero=r;
    return true;
  }
```



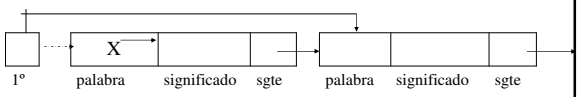

```
//agregar entre dos nodos
```



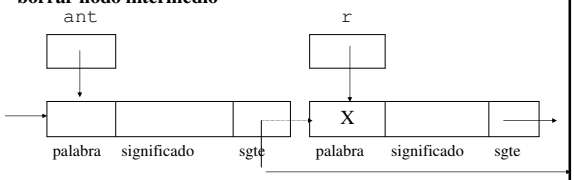

```
//determinar referencia a nodo anterior
Nodo ant;
for(ant=primero;ant.sgte!=null;ant=ant.sgte){
  if(ant.sgte.palabra.compareTo(x)>0) break;

//insertar
r.sgte=ant.sgte; //1
ant.sgte=r;      //2
return true;
}
```

Propuesto. Implementar en un sólo recorrido de la lista enlazada

Ejercicio: operación borrar de TDA diccionario
borrar primer nodo



borrar nodo intermedio




```
public boolean borrar(Comparable x){
```

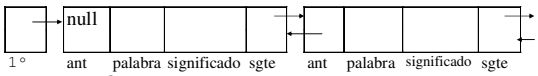
Solución 1.

```
public boolean borrar(Comparable x){
   //buscar ref de nodo con palabra x
   Nodo r=referencia(x,primero);
   if(r==null) return false;
   //si es el 1º, eliminar nodo
   if(r==primero){//1°→x y1→x2y2→…
      primero=primero.sgte; //1°→x2y2→…
      return true;
   }
   //buscar anterior: →xi-1yi-1→xyi→xi+1yi+1→…
  Nodo anterior=primero;
  while(anterior.sgte!=r)
    anterior=anterior.sgte;
  //eliminar nodo: →xi-1yi-1→xi+1yi+1→…
   anterior.sgte=r.sgte;
   return true;
}
```

Solución 2: con un sólo recorrido de lista

```
public boolean borrar(Comparable x){
   if(primero==null)return false;
   //si es el 1º, eliminar nodo
   if(primero.palabra.equals(x)){
      primero=primero.sgte; return true;
   }
   //determinar ref de nodo y anterior
   Nodo ant=primero, r=primero.sgte;
   while(r!=null && !r.palabra.equals(x)){
      ant=r; r=r.sgte;
   }
   if(r==null) return false;
   //eliminar nodo
   ant.sgte = r.sgte;
   return true;
}
```

Solución 3: con lista de doble enlace



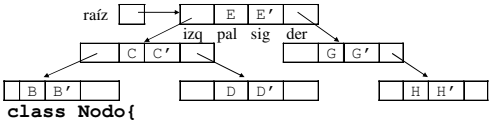

```
class Nodo{
public Comparable palabra;
public Object significado;
public Nodo sgte, ant;
public Nodo(
Comparable x,Object y,Nodo z,Nodo w){
   palabra=x; significado=y; sgte=z; ant=w;
}
}
class Diccionario{
protected Nodo primero;
public Diccionario(){primero=null;}
```

```
public boolean borrar(Comparable x){
   //buscar ref de nodo con palabra x
   Nodo r=referencia(x,primero);
   if(r==null) return false;
   //si es el 1º, eliminar nodo
   if(r==primero){
      primero=primero.sgte;
      if(primero!=null) primero.ant=null;
      return true;
   }
   //eliminar nodo
   r.ant.sgte=r.sgte;
   if(r.sgte!=null) r.sgte.ant=r.ant;
   return true;
}
```

Propuesto. Escribir otro métodos

Diccionario: implementación con Arbol Binario (de búsqueda)

- a la izquierda los menores, a la derecha los mayores
- tiempo de todas las operaciones proporcional a altura del árbol
- si está balanceado, altura=$\log_2 n$




```
class Nodo{
public Comparable palabra;
public Object significado;
public Nodo izq, der;
public Nodo(
Comparable x,Object y,Nodo z,Nodo w){
   palabra=x; significado=y; izq=z; der=w;
}
}
```