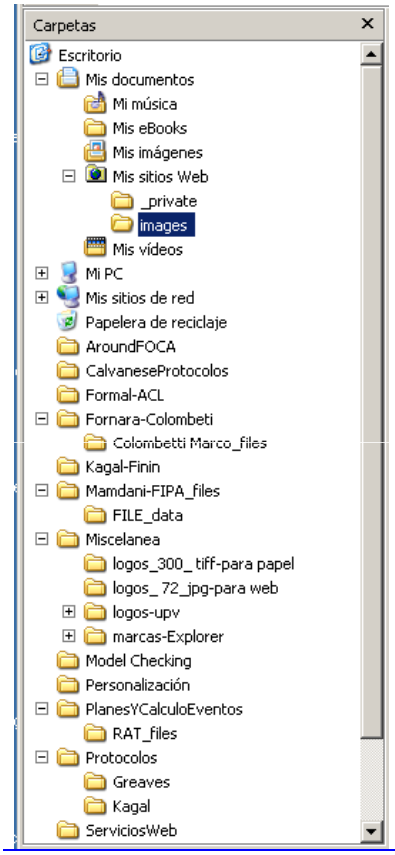
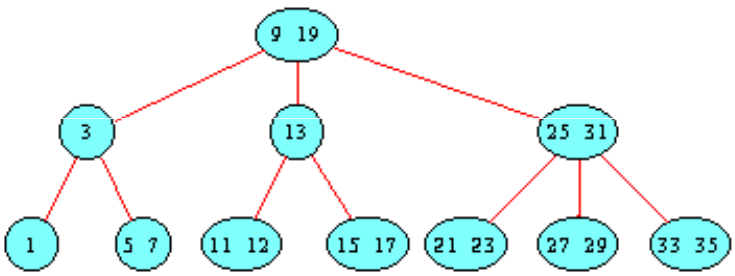
Zuhaitzak



Zer da zuhaitz bat?

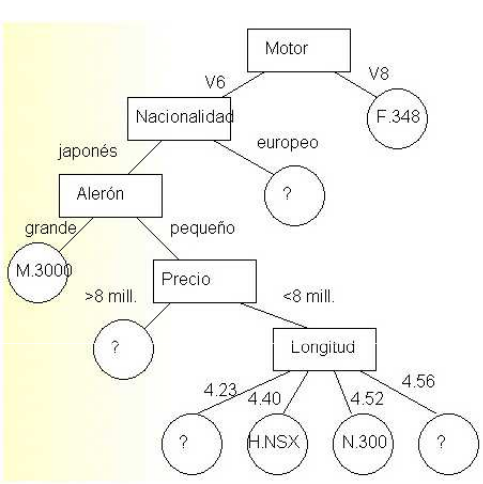
Egitura hierarkikoa,
ez lineala

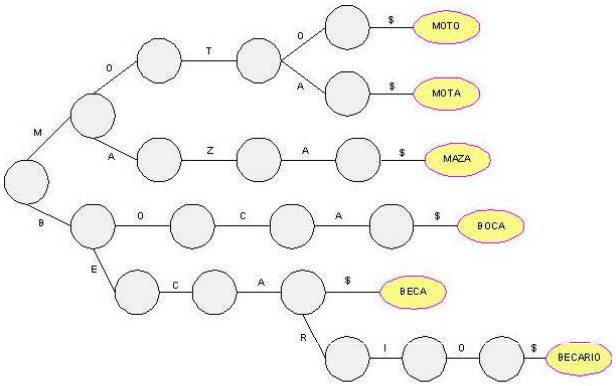


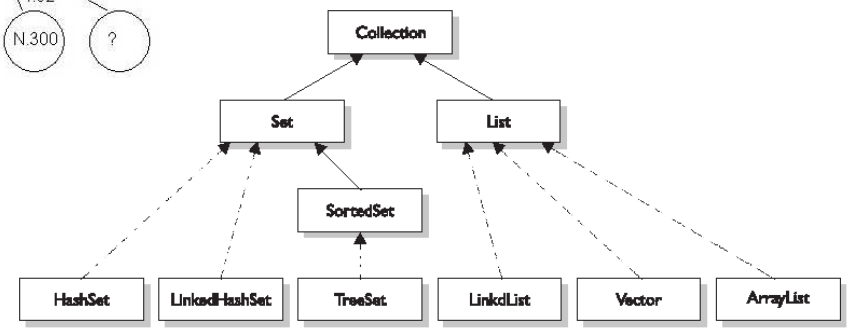
Oinarrizko terminologia

- Zuhaitz bat adabegiez eta arkuez osatuta dago. Arkuek adabegiak konektatzen dituzte
- Adabegiek entitateak adierazten dituzte, eta arkuek entitateen arteko erlazioak

Adibideak







Zuhaitzaren kontzeptua

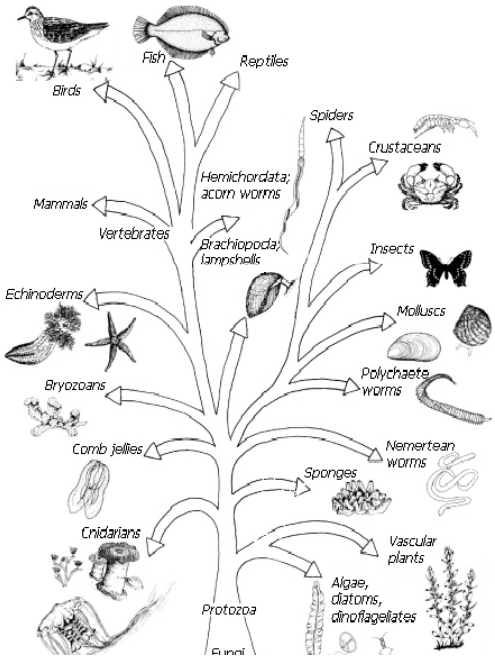
- Zuhaitz bat arkuek konektatutako adabegiez osatuta dago:
  - Erro-adabegi bakarra dago
  - Adabegi bakoitzak, erroak ezezik, guraso bakarra du
- Ondorioa: zuhaitz batean bide bakarra dago errotik edozein adabegiraino

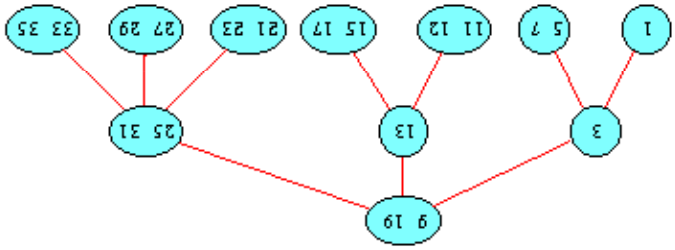
Zuhaitzaren definizio errekurtsiboa

- T motako zuhaitzen definizioa:
  - <> $\in$ Zuhaitz<T> [zuhaitz hutsa, edo bestela]
  - $A_1, A_2, \ldots, A_n \in$ Zuhaitz<T> y t $\in$ T $\rightarrow$
    t ($A_1, A_2, \ldots, A_n$) $\in$ Zuhaitz<T>

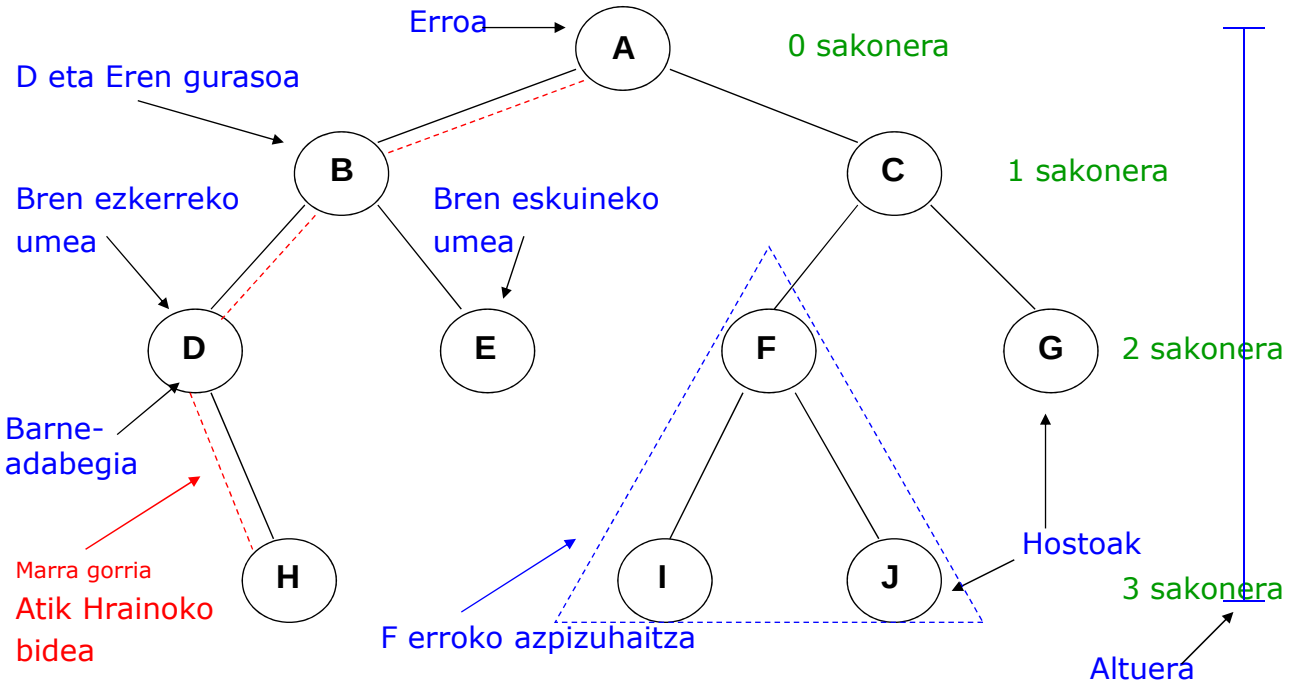
    [T motako objektu bat (erroa) eta T motako zuhaitzen kopuru finitua]

Zuhaitzen diagramak





Oinarrizko terminologia



Oinarrizko terminologia

- Adabegi baten altuera adabegi horretatik hosto batera dagoen bide luzeenaren tamaina da
- Hosto-adabegi baten altuera zero da
- Zuhaitz baten altuera erroaren altuera da
- Adabegi baten sakonera errotik adabegi horretara doan bidearen luzera da
- Adabegi baten aurrekoak, adabegia bera eta bere gurasoaren aurreko guztiak dira
- Adabegi baten ondorengoak, adabegia bera eta bere umeen ondorengoak dira

Oinarrizko terminologia

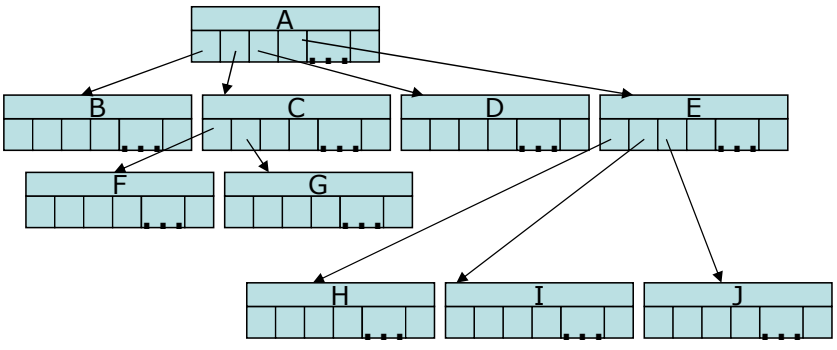
- Adabegi baten gradua adabegi horren ume-kopurua da
- Zuhaitz baten gradua bere adabegien gradu handiena da
  - Zuhaitz n-tarra: n graduko zuhaitza
  - Zuhaitz bitarra: 2 graduko zuhaitza

Zuhaitzen adierazpena eta inplementazioa

- Zuhaitzen adierazpena eta inplementatutako metodoak zuhaitz horien erabileraren araberakoak izango dira

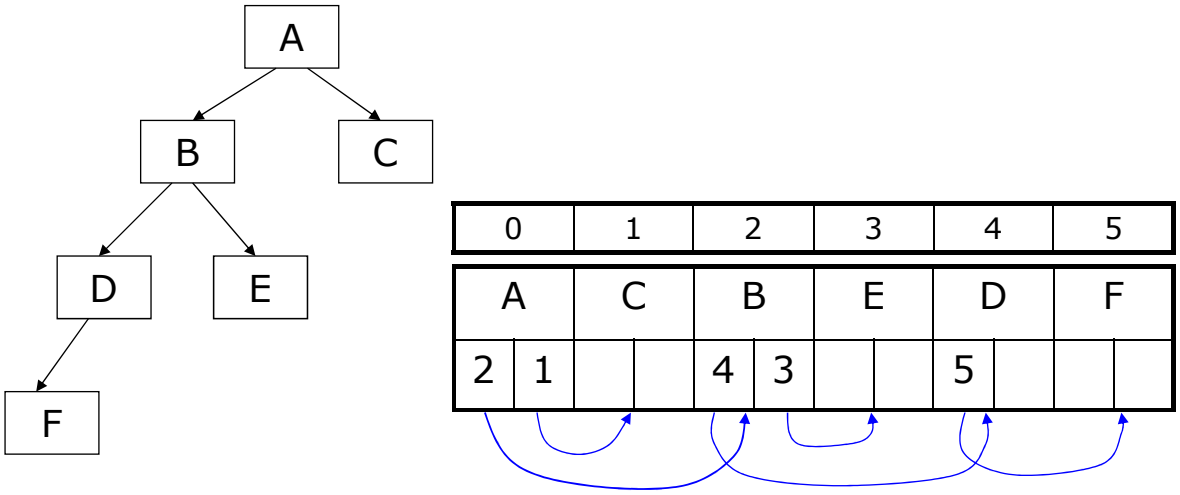
Zuhaitz orokorrak inplementatzeko estrategiak (1)

- Adabegiak erreferentzien arrayak
- Memoria alferrik erabiliko da ume-kopurua oso aldakorra bada
- Arazoak arrayaren luzerarekin




```
class Adabegi<T>{
T info;
Adabegi<T>[]  umeak;
.........
}
class ZuhaitzOrokorra<T> {
Adabegi<T> erroa;
......
}
```

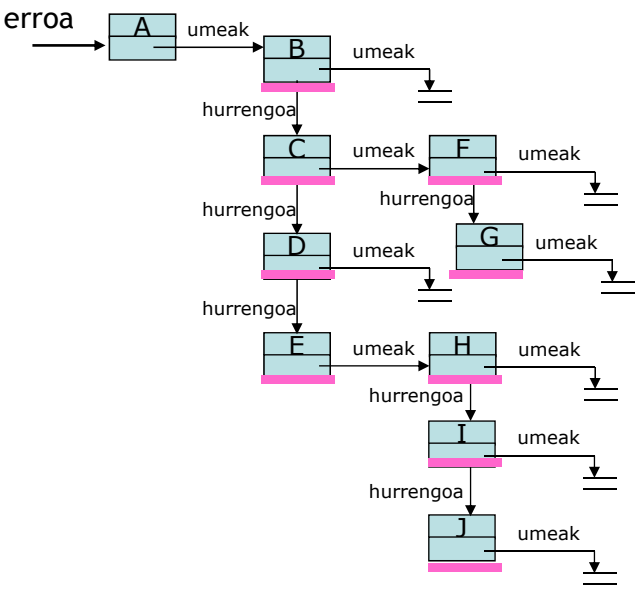
Zuhaitz orokorrak inplementatzeko estrategiak (2)



Arrayaren bidezko adierazpena, estekak simulatuz

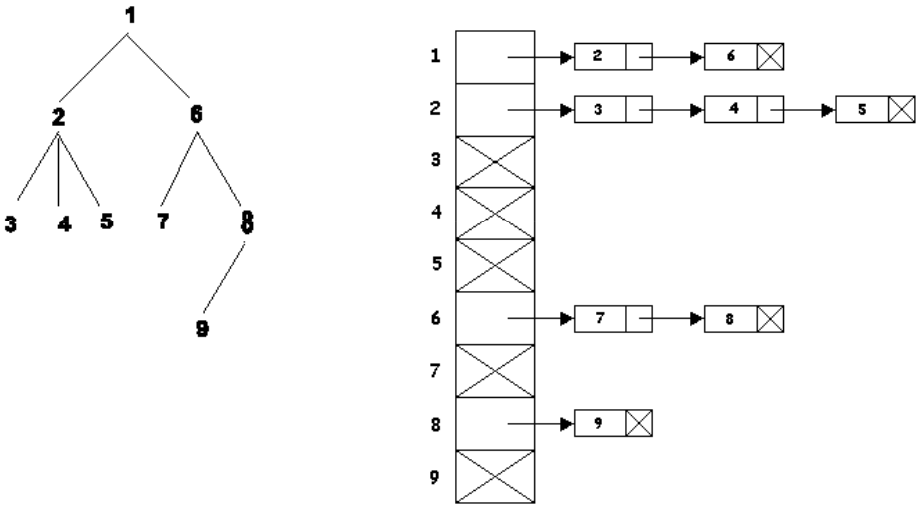
Zuhaitz orokorrak inplementatzeko estrategiak (3)

- Umeak zerrenda estekatu batean



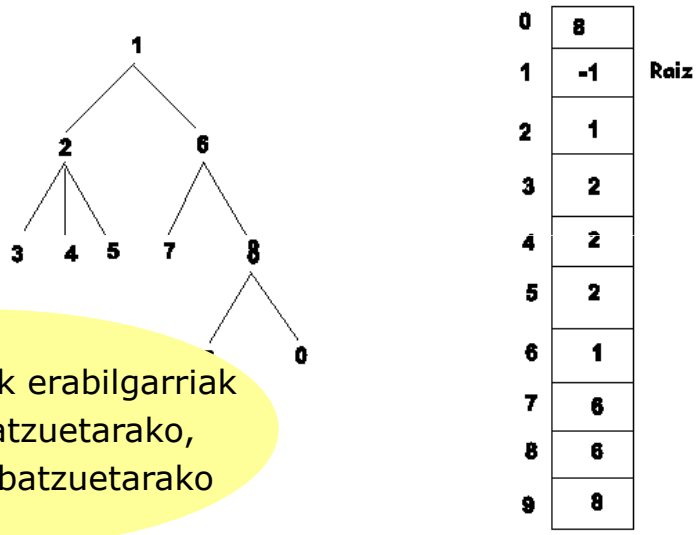

```
class ZuhaitzOrokorra<T>{
   AdabegiZO<T> erroa;
   …
}
class AdabegiZO<T>{
T info;
 LinkedList<T> umeak;
…
}
class LinkedList<T>{
Adabegi<T> hasiera;
}
class Adabegi<T>{
AdabegiZO<T> data;
Adabegi<T> hurrengoa;
}
```

Zuhaitz orokorrak inplementatzeko estrategiak (4)



Umeen zerrenden bidezko adierazpena

Zuhaitz orokorrak inplementatzeko estrategiak (5)

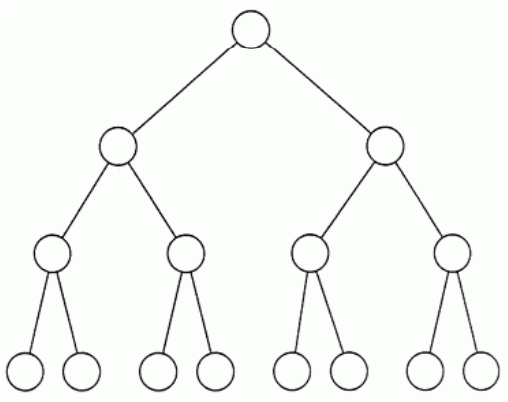


Adierazpen batzuk erabilgarriak dira prozesu batzuetarako, baina ez beste batzuetarako

Adabegi bakoitzaren gurasoa adieraziz

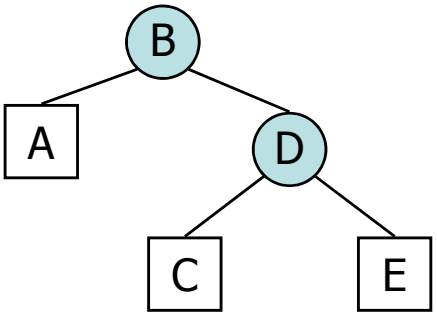
Zuhaitz bitarra

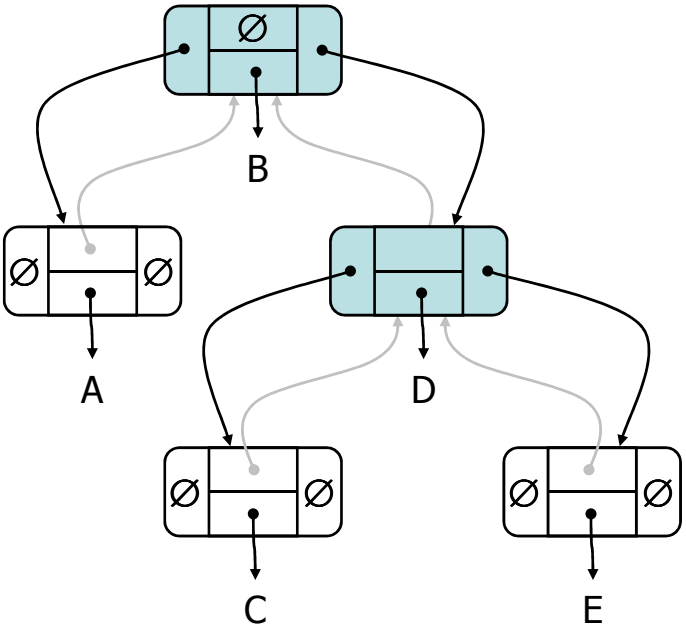
- Adabegi bakoitzak, gehienez 2 ume ditu (eskerreko eta eskuineko umea)



Zuhaitz bitarrak inplementatzeko estrategiak (1)

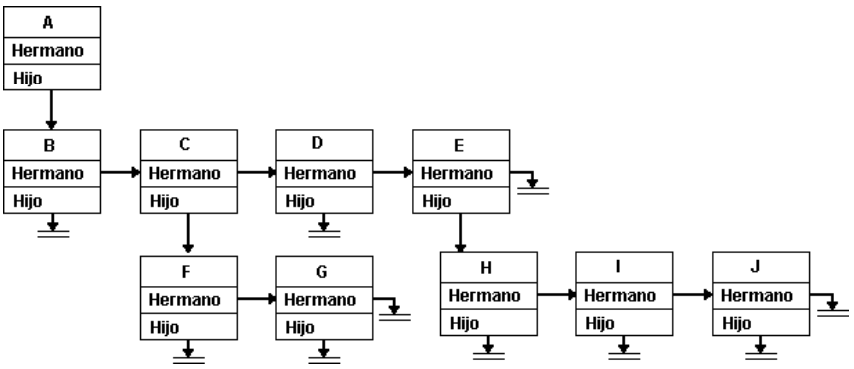
- Adabegi bakoitzak:
  - Elementua
  - Ezkerreko umearen adabegia
  - Eskuineko umearen adabegia
  - *[Adabegi gurasoa]*
- Zuhaitza: erro-adabegiaren erreferentzia





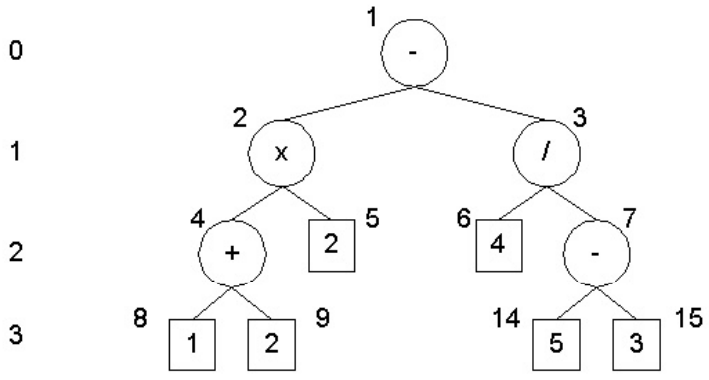
Edozein zuhaitz inplementa daiteke zuhaitz bitar baten bidez

- Erreferentzia bat erlazio bakoitzeko (umea/anaia)




```
class Adabegi<T> {
T info;
 Adabegi<T> umea;
 Adabegi<T> anaia;
…
}
class
ZuhaitzOrokorra<T> {
 Adabegi<T> erroa;
…
}
```


Zuhaitz bitarrak inplementatzeko estrategiak (2)



	-	x	/	+	2	4	-	1	2					5	3
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Zuhaitz bitarrak posizio kalkulatuen bidezko adierazpena

Zuhaitz bitarrak inplementatzeko estrategiak (3)

- Inplementazio errekurtsiboa, definizio honetan oinarrituta:
  - <> $\in$ ZuhaitzBit<T> (zuhaitz hutsa), edo
  - A_1, $A_2 \in$ ZuhaitzBit<T> y t $\in$ T $\rightarrow$
    t (A_1, A_2) $\in$ ZuhaitzBit<T>

Gure zuhaitz bitarrentzako interfazea

```
public interface TADBinaryTree<T> {

// ADABEGIEN ATZIPENERAKO
public boolean isEmpty();

/*.equals(elem) duen balioa topatzen du
(Hau da, T motan equals metodoaren
      inplementazioaren arabera funtzionatuko du),
eta objektuaren erreferentzia bueltatuko du,
      topatzen badu
null bueltatuko du ez baldin badago
*/
public T find (T elem);


// ALDAKETARAKO
// ezkerreko azpizuhaitza kentzen du
public void removeLeftSubTree();
// eskuineko azpizuhaitza kentzen du
public void removeRightSubTree();
// adabegi guztiak kentzen ditu
public void removeAll();

// KONTSULTARAKO
// Zuhaitzaren adabegi-kopurua bueltatuko du
public int size();

// true bueltatuko du zuhaitzak balio hori badu,
// eta false bestela
public boolean contains(T elem);

public String toString();

// ITERADOREAK
public Iterator<T> iteradoreAurreOrdena();
public Iterator<T> iteradoreInOrdena();
public Iterator<T> iteradorePostOrdena();
public Iterator<T> iteradoreMailaka();

}
```

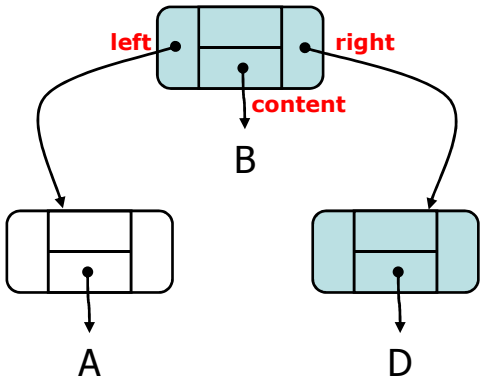
Adabegien klasea zuhaitz bitarretan

```
public class BinaryTreeNode<T> {

    protected T content;
    protected BinaryTreeNode<T> left;
    protected BinaryTreeNode<T> right;

    public BinaryTreeNode(T elem){
          content = elem;
          left = null;
          right = null;
    }

}
```

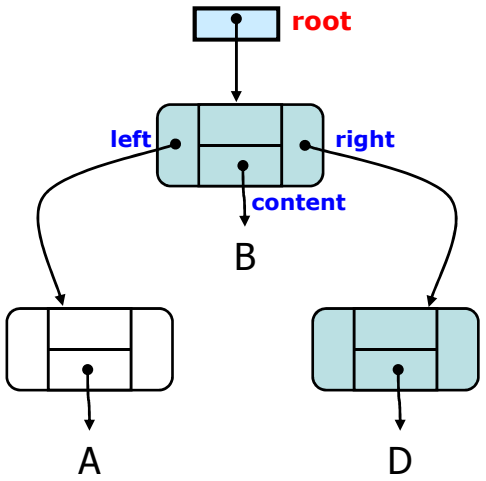


Adabegien klasea zuhaitz bitarretan

```
public class BinTree<T> implements TADBinaryTree<T> {

    protected BinaryTreeNode<T> root;
    protected int count;

    public BinTree(){
          root = null;
          count = 0;
    }
```



Zuhaitzak eta errekurtsibitatea

```
// ezkerreko azpizuhaitza kentzen du
public void removeLeftSubTree(){
    count = count - numDescendants(root.left);
    root.left = null;
}

// "adabegia"ren ondorengoak kalkulatzeko
private int numDescendants(BinaryTreeNode<T> adabegia){
    if (adabegia == null)
        return 0;
    else
        return 1 + numDescendants(adabegia.left) +
                   numDescendants(adabegia.right);
}
```


Errekurtsibitatea eta murgilketa

```
public boolean contains(T elem){
    return contains(elem, root);
}

private boolean contains(T elem, BinaryTreeNode<T> temp){
    if (temp==null)  // elem ez dago
        return false;
    else if (temp.content.equals(elem)) // elem temp-en dago
        return true;
    else
        return (contains(elem, temp.left) || contains(elem, temp.right));
}
```

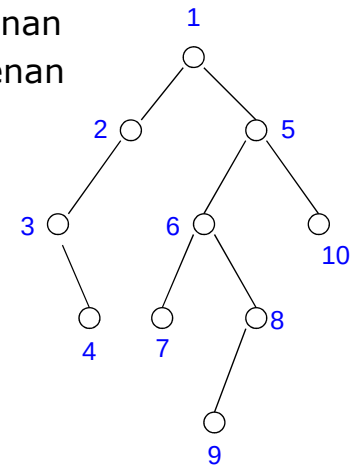
contains(elem) murgilduta dago hemen: contains(elem, root)

Zuhaitz bitarretako elementu guztien atzipena

Eragiketa	Deskribapena
Iterator<T> iteradoreInOrdena()	In-Ordena azterketarako iteradorea bueltatzen du
Iterator<T> iteradoreAurreOrdena()	Aurre-Ordena azterketarako iteradorea bueltatzen du
Iterator<T> iteradorePostOrdena()	Post-Ordena azterketarako iteradorea bueltatzen du
Iterator<T> iteradoreMailaka()	Mailakako azterketarako iteradorea bueltatzen du

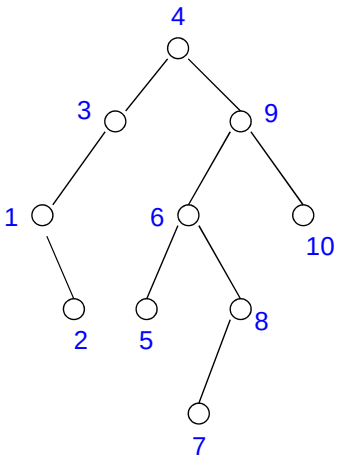
Aurre-ordena

1. Erroa *bisitatu*
2. Ezkerreko azpizuhaitza bisitatu aurre-ordenan
3. Eskuineko azpizuhaitza bisitatu aurre-ordenan



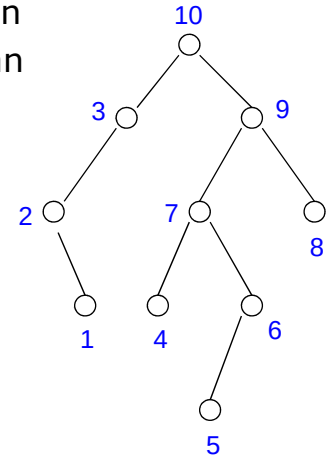
In-Ordena

1. Ezkerreko azpizuhaitza bisitatu in-ordenan
2. Erroa *bisitatu*
3. Eskuineko azpizuhaitza bisitatu in-ordenan



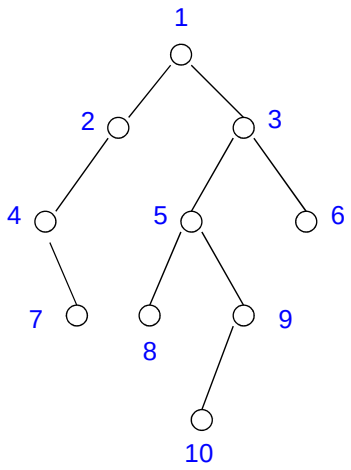
Post-Ordena

1. Ezkerreko azpizuhaitza bisitatu post-ordenan
2. Eskuineko azpizuhaitza bisitatu post-ordenan
3. Erroa *bisitatu*



Mailakako korritzea

- Sakonera bakoitzeko (k=0; k++):
  ezkerretik eskuinera korritu k sakonerako adabegiak



Post-Ordenaren bidezko iteradorea

```
public Iterator<T> iteradorPostOrdena(){
    LinkedList<T> tempList = new LinkedList<T>();
    postOrdena(root, tempList);
    return tempList.iterator();
}


private void postOrdena(BinaryTreeNode<T> erroa, LinkedList<T> lista){
    if (erroa != null ){
            postOrdena(erroa.left, lista);
            postOrdena(erroa.right, lista);
            lista.insertLast(erroa.content); // Adabegiaren prozesaketa
    }
}
```

Mailakako iteradorea

```
public Iterator<T> iteradoreMailaka(){
    LinkedList<T> tempList = new LinkedList<T>();
    mailaka(root, tempList);
    return tempList.iterator();
}


private void mailaka(BinaryTreeNode<T> erroa, LinkedList<T> lista){
    if (erroa != null ){
        LinkedQueue<BinaryTreeNode<T>> ilara = new LinkedQueue<BinaryTreeNode<T>>();
        ilara.insert(erroa);
        while ( !ilara.isEmpty() ){
            BinaryTreeNode<T> temp = ilara.remove();
            lista.insertLast(temp.content);  // Adabegiaren prozesaketa
            if ( temp.left != null )
                ilara.insert(temp.left);
            if ( temp.right != null )
                ilara.insert(temp.right);

        }
    }
}
```


Irakurketa

[Lewis, Chase 2010]

–9. kapitulua

Zuhaitza(informatika):
http://es.wikipedia.org/wiki/Árbol_(informática)

Zuhaitz bitarra:
http://es.wikipedia.org/wiki/Árbol_binario

Bilaketa-zuhaitz bitarra:
http://es.wikipedia.org/wiki/Árbol_binario_de_búsqueda