Arboles Generales

Estructura

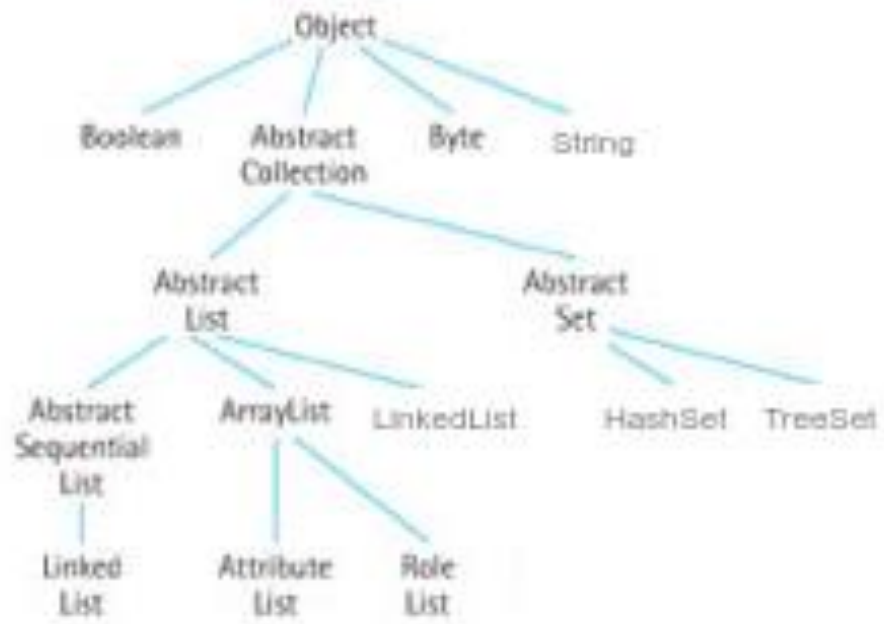
<<Java Class>>

ArbolGeneral<T>

tp05.ejercicio1

- dato: T
- hijos: ListaGenerica<ArbolGeneral<T>>

- ArbolGeneral(T)
- ArbolGeneral(T,ListaGenerica<ArbolGeneral<T>>)
- getDato()
- setDato(T):void
- setHijos(ListaGenerica<ArbolGeneral<T>>):void
- getHijos():ListaGenerica<ArbolGeneral<T>>
- agregarHijo(ArbolGeneral<T>):void
- esHoja():boolean
- tieneHijos():boolean
- esVacio():boolean
- eliminarHijo(ArbolGeneral<T>):void
- altura():Integer
- nivel(T):Integer
- ancho():Integer



Arboles Generales

Código Fuente – Constructores, this()

```
package tp04.ejercicio1;

public class ArbolGeneral<T> {
 private T dato;
 private ListaGenerica<ArbolGeneral<T>> hijos =
                                    new ListaEnlazadaGenerica<T>();

  public ArbolGeneral(T dato) {
            this.dato = dato;
  }
  public ArbolGeneral(T dato, ListaGenerica<ArbolGeneral<T>> hijos){
            this(dato);
            this.hijos = hijos;
  }

  public T getDato() {
            return dato;
  }

  public void setDato(T dato) {
      this.dato = dato;
  }

  public void setHijos(ListaGenerica<ArbolGeneral<T>> hijos) {
            if (hijos==null)
                        hijos=new ListaEnlazadaGenerica<T>();
            this.hijos = hijos;
    }

  public ListaGenerica<ArbolGeneral<T>> getHijos() {
      return this.hijos;
  }
```

Constructores

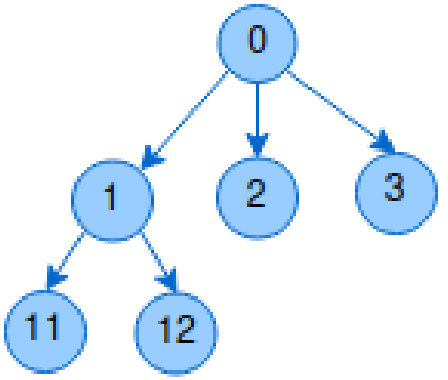
```
public void agregarHijo(ArbolGeneral<T> unHijo) {
            this.getHijos().agregarFinal(unHijo);
}

public void eliminarHijo(ArbolGeneral<T> hijo) {
   if (this.tieneHijos()) {
      ListaGenerica<ArbolGeneral<T>> hijos=this.getHijos();
   if (hijos.incluye(hijo))
      hijos.eliminar(hijo);
 }

 public boolean esHoja() {
    return !this.tieneHijos();
 }

 public boolean tieneHijos() {
  return this.hijos != null && !this.hijos.esVacia();
 }

 public boolean esVacio() {
   return this.dato == null && !this.tieneHijos();
 }
}
```

Arboles Generales

Recorrido PreOrden

Implementar un método en **ArbolGeneral** que imprima los valores en preorden.

```
package ayed;
public class ArbolGeneral<T> {
  private T dato;
  private ListaGenerica<ArbolGeneral<T>> hijos;
  . . .
  public void pre_Orden() {
            System.out.println(this.getDato());
            ListaGenerica<ArbolGeneral<T>> lHijos = this.getHijos();
            lHijos.comenzar();
            while (!lHijos.fin()) {
                      (lHijos.proximo()).pre_Orden();
            }
  }
}
```



Fragmento de código que crea un árbol general e invoca el recorrido pre_Orden() (puede ser una main en una clase de test)

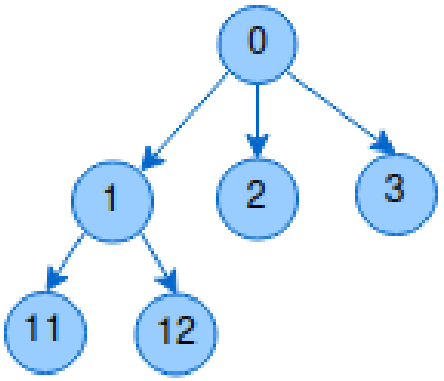
```
ArbolGeneral<String> a1 = new ArbolGeneral<String>("1");
ArbolGeneral<String> a2 = new ArbolGeneral<String>("2");
ArbolGeneral<String> a3 = new ArbolGeneral<String>("3");
ListaGenerica<ArbolGeneral<String>> hijos = new ListaEnlazadaGenerica<ArbolGeneral<String>>();
hijos.agregarFinal(a1); hijos.agregarFinal(a2); hijos.agregarFinal(a3);
ArbolGeneral<String> a = new ArbolGeneral<String>("0", hijos);
ListaGenerica<ArbolGeneral<String>> hijos_a1 = new ListaEnlazadaGenerica<ArbolGeneral<String>>();
ArbolGeneral<String> a11 = new ArbolGeneral<String>("11");
ArbolGeneral<String> a12 = new ArbolGeneral<String>("12");
hijos_a1.agregarFinal(a11); hijos_a1.agregarFinal(a12);
a1.setHijos(hijos_a1);
System.out.println("Datos en preorden: "+a.pre_Orden());
```

Arboles Generales

Recorrido PreOrden

Implementar un método en **ArbolGeneral** que retorne una lista con los datos del árbol recorrido en preorden

```
package ayed;
public class ArbolGeneral<T> {
  private T dato;
  private ListaGenerica<ArbolGeneral<T>> hijos;
  . . .
  public ListaEnlazadaGenerica<T> preOrden() {
       ListaEnlazadaGenerica<T> lis = new ListaEnlazadaGenerica<T>();
            this.preOrden(lis);
            return lis;
  }
  private void preOrden(ListaGenerica<T> l) {
            l.agregarFinal(this.getDato());
            ListaGenerica<ArbolGeneral<T>> lHijos = this.getHijos();
            lHijos.comenzar();
            while (!lHijos.fin()) {
                      (lHijos.proximo()).preOrden(l);
            }
  }
}
```

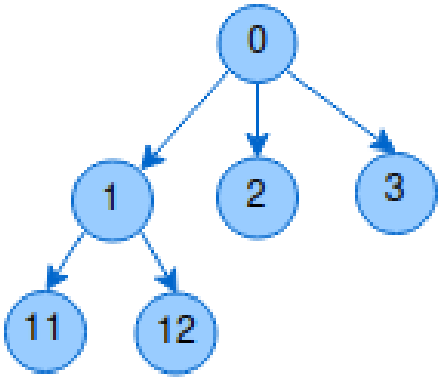


Fragmento de código que crea un árbol general e invoca el recorrido preOrden() (puede ser una main en una clase de test)

Arboles Generales

Contar sus hojas

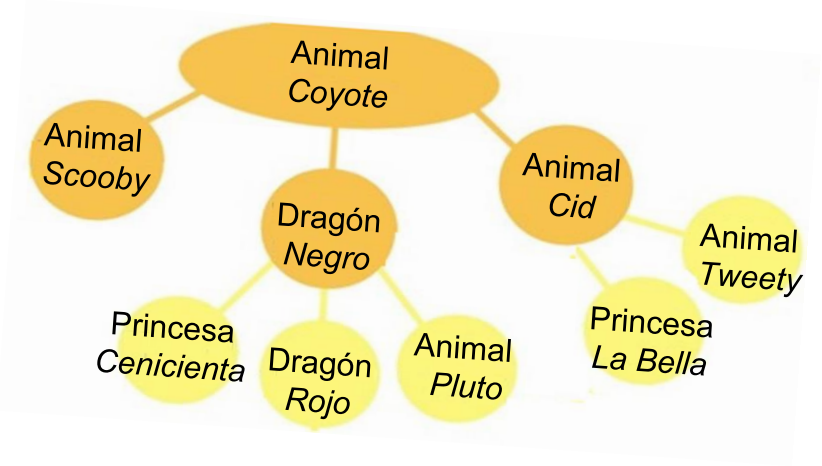
```
package tp03.ejercicio10;

public class ArbolGeneral<T> {
  private T dato;
  private ListaGenerica<ArbolGeneral<T>> hijos;
   . . .

   /**
    * Contar las hojas del arbol general
    */
   public int contarHojas() {
           if (this.esHoja())
                   return 1;
           else {
                   int cont = 0;
                   ListaGenerica<ArbolGeneral2<T>> lista = this.getHijos();
                   lista.comenzar();
                   while (!lista.fin()) {
                           ArbolGeneral2<T> arbol = lista.proximo();
                           cont = cont + arbol.contarHojas();
                   }
                   return cont;
           }
   }

}
```



Arboles Generales

Ejercicio de parcial – Encontrar a la Princesa

Dado un árbol general compuesto por personajes, donde puede haber dragones, princesas y otros, se denominan nodos accesibles a aquellos nodos tales que a lo largo del camino del nodo raíz del árbol hasta el nodo (ambos inclusive) no se encuentra ningún dragón.

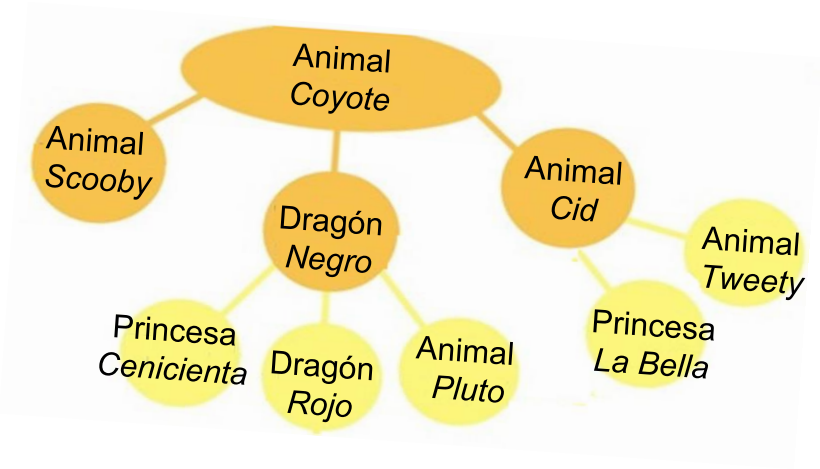


Implementar un método que devuelva una lista con un camino desde la raíz a una Princesa sin pasar por un Dragón –sin necesidad de ser el más cercano a la raíz-. Asuma que existe al menos un camino accesible.

Arboles Generales

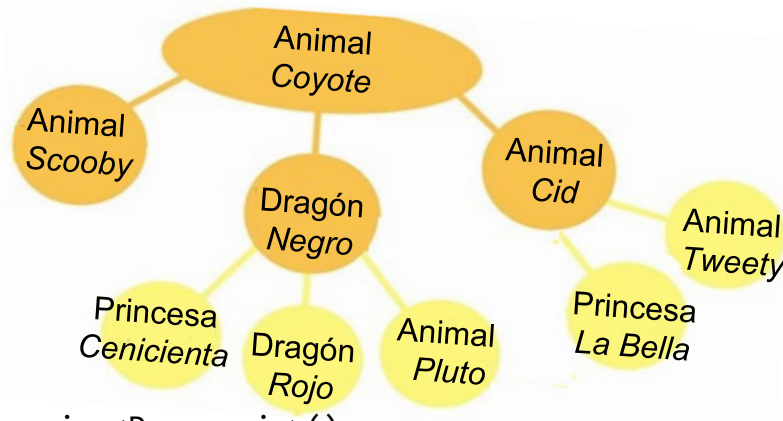
Ejercicio de parcial – Encontrar a la Princesa

```
package parcial.juego;

public class Personaje {
 private String nombre;
 private String tipo;    //Dragon, Princesa, Animal, etc.

 public Personaje(String nombre, String tipo) {
  this.nombre = nombre;
  this.tipo = tipo;
 }
 public String getNombre() {
  return nombre;
 }
 public void setNombre(String nombre) {
  this.nombre = nombre;
 }
  . . .

 public boolean esDragon(){
  return this.getTipo().equals("Dragon");
 }
 public boolean esPrincesa(){
  return this.getTipo().equals("Princesa");
 }
}
```



Arboles Generales

Ejercicio de parcial – Encontrar a la Princesa
Versión I



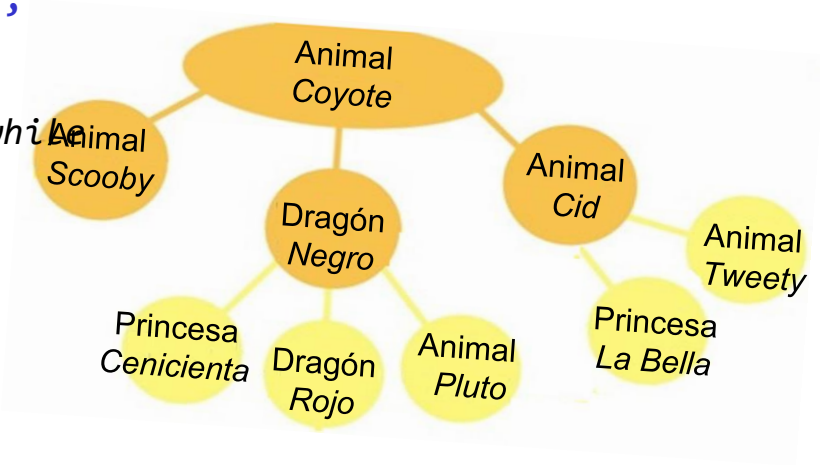

```
public class Juego {

	public void encontrarPrincesa(ArbolGeneral<Personaje> arbol) {
			ListaGenerica<Personaje> lista = new ListaEnlazadaGenerica<Personaje>();
			lista.agregarInicio(arbol.getDato());
			ListaGenerica<Personaje> camino = new ListaEnlazadaGenerica<Personaje>();
			encontrarPrincesa(arbol, lista, camino);
			System.out.print("Se encontró a la Princesa en el camino: " + camino);
	}
	private void encontrarPrincesa(ArbolGeneral<Personaje> arbol, ListaGenerica<Personaje> lista,
									ListaGenerica<Personaje> camino) {
			Personaje p = arbol.getDato();
			if (p.esPrincesa()) clonar(lista, camino);

			if (camino.esVacia()) {
				ListaGenerica<ArbolGeneral<Personaje>> lHijos = arbol.getHijos();
				lHijos.comenzar();
				while (!lHijos.fin() && camino.esVacia()) {
					ArbolGeneral<Personaje> aux = lHijos.proximo();
					if (!aux.getDato().esDragon()) {
						lista.agregarFinal(aux.getDato());
						encontrarPrincesa(aux, lista, camino);
						lista.eliminarEn(lista.tamanio());
					}
				}
			}
	}
}
```

```
public void clonar(ListaGenerica<Personaje> origen,
		ListaGenerica<Personaje> destino) {
	origen.comenzar();
	while (!origen.fin())
	destino.agregarFinal(origen.proximo());
}
```

Arboles Generales

Ejercicio de parcial – Encontrar a la Princesa
Versión II

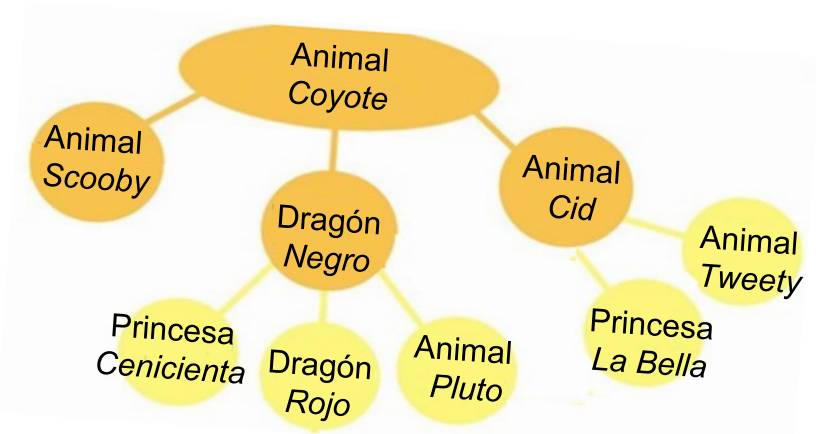
```
public class Juego {

  public ListaEnlazadaGenerica<Personaje> encontrarPrincesa(ArbolGeneral<Personaje> arbol){
    ListaEnlazadaGenerica<Personaje> lista = new ListaEnlazadaGenerica<Personaje>();
    if (arbol.getDato().esPrincesa() || arbol.getDato().esDragon() || arbol.esHoja()){
      if (arbol.getDato().esPrincesa()){
         Personaje p = arbol.getDato();
         lista.agregarInicio(p);
      }
      return lista;
    }
         ListaGenerica<ArbolGeneral<Personaje>> lHijos = arbol.getHijos();
         lHijos.comenzar();
         while(!lHijos.fin() && lista.esVacia()){
             lista = encontrarPrincesa(lHijos.proximo());
             if(!lista.esVacia()){
               lista.agregarInicio(arbol.getDato());
              //break; en vez de lista.esVacia() en el while
             }
          }
         return lista;

  }
}
```



Arboles Generales

Ejercicio de parcial – Encontrar a la Princesa

```
package parcial.juego;
…
public class JuegoTest {
public static void main(String[] args) {
 Personaje p0 = new Personaje("Scooby", "Animal");
 Personaje p1 = new Personaje("Cenicienta", "Princesa");
 Personaje p2 = new Personaje("Rojo", "Dragon");
 Personaje p3 = new Personaje("Pluto", "Animal");
 Personaje p4 = new Personaje("Negro", "Dragon");
 Personaje p5 = new Personaje("La Bella", "Princesa");
 Personaje p6 = new Personaje("Tweety", "Animal");
 Personaje p7 = new Personaje("Cid", "Animal");
 Personaje p8 = new Personaje("Coyote", "Animal");
 ArbolGeneral<Personaje> a1 = new ArbolGeneral<Personaje>(p0);
 ArbolGeneral<Personaje> a21 = new ArbolGeneral<Personaje>(p1);
 ArbolGeneral<Personaje> a22 = new ArbolGeneral<Personaje>(p2);
 ArbolGeneral<Personaje> a23 = new ArbolGeneral<Personaje>(p3);
 ListaGenerica<ArbolGeneral<Personaje>> hijosa2 = new ListaEnlazadaGenerica<ArbolGeneral<Personaje>>();
 hijosa2.agregarFinal(a21);
 hijosa2.agregarFinal(a22);
 hijosa2.agregarFinal(a23);
 ArbolGeneral<Personaje> a2 = new ArbolGeneral<Personaje>(p4, hijosa2);
 . . .
 ArbolGeneral<Personaje>  a = new ArbolGeneral<Personaje>(p8, hijos);
 Juego juego = new Juego();
 juego.encontrarPrincesa(a);
 }
}
```

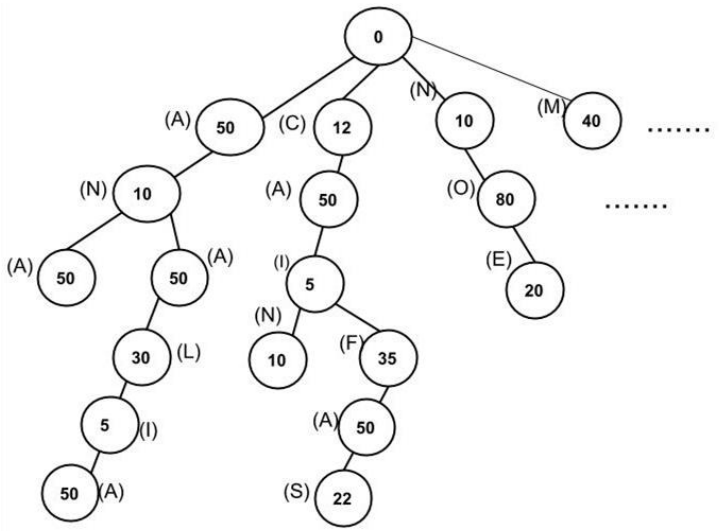


Arboles Generales

Ejercicio de Parcial - Gematría

Antiguamente el pueblo judío usaba un sistema de numeración llamado **Gematría** para asignar valores a las letras y así “ocultar” nombres, de aquí que se asocia el nombre de Nerón César al valor 666 (la suma de los valores de sus letras).
Contamos con una estructura de datos como la que aparece en el gráfico, donde **cada camino en el árbol codifica un nombre**. Cada nodo **contiene un valor** asociado a una letra, excepto el nodo raíz que contiene el valor 0 y no es parte de ningún nombre, y simplemente significa “comienzo”. “**Un nombre completo SIEMPRE es un camino que comienza en la raíz y termina en una hoja.”**

La tarea a llevar adelante consiste en escribir un método que **dado un valor numérico**, **cuenta los nombres completos** que suman exactamente dicho **valor**. Se recibe el árbol con las letras ya sustituidas por sus valores; las letras ya no importan.
Escriba una clase llamada **`Gematria`**, que NO contenga variables de instancia, con sólo un método público con la siguiente firma:



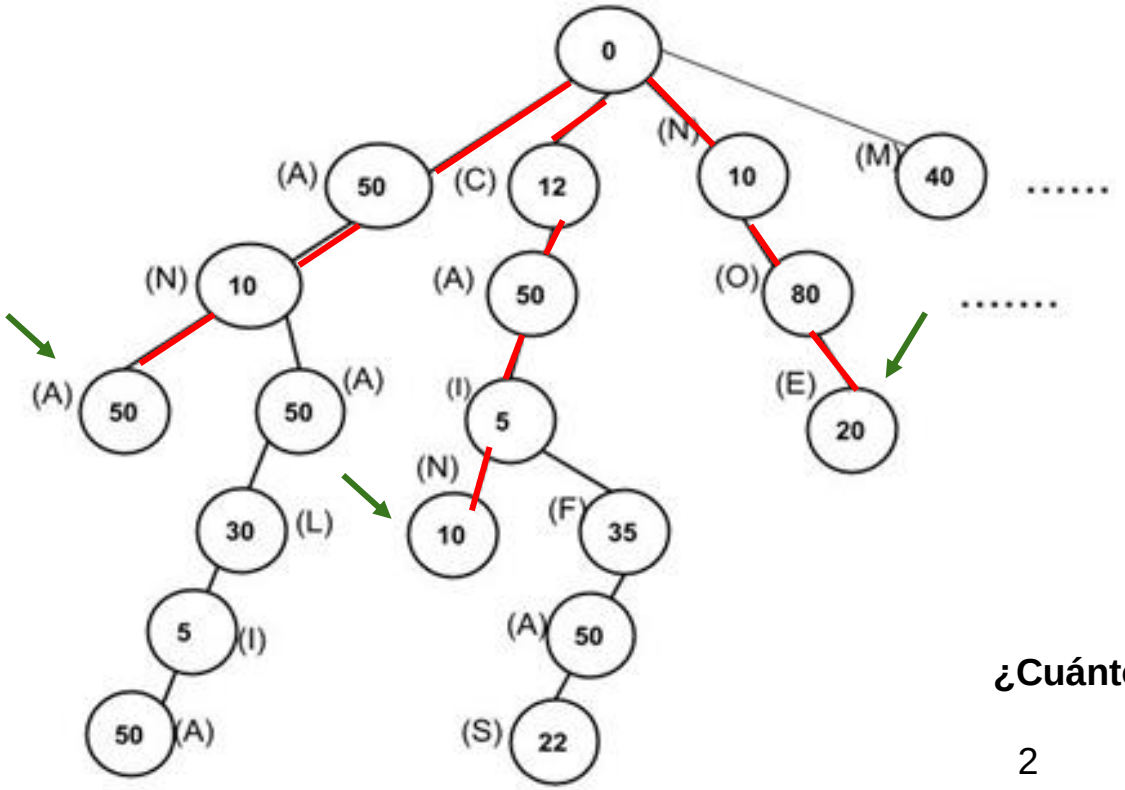

```
public int contadorGematria(xxx, int valor)
```

árbol general que contiene los números

valor es el valor que se debería obtener al sumar el valor de las letras de un nombre

Arboles Generales

Ejercicio de Parcial - Gematría



Dado el valor 110: ¿Qué nombres se pueden codificar con este árbol de Gematría?:

ANA

NOE

Dado el valor 77: ¿Qué nombres se pueden codificar con este árbol de Gematría?:

CAIN

¿Cuántos nombres se pueden codificar con el valor 110?

2

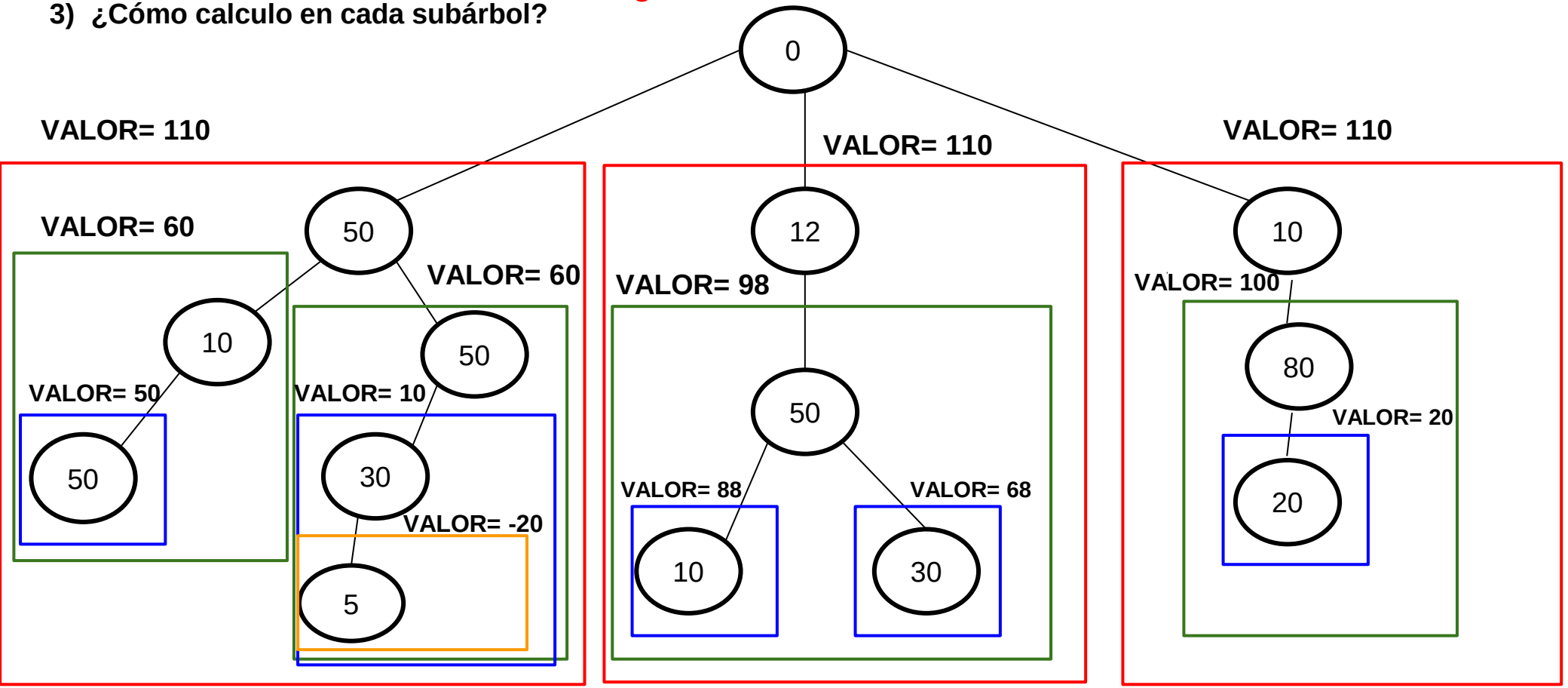
¿Cuántos nombres se pueden codificar con el valor 77?

1

¿Cómo calculo la cantidad de palabras que se pueden codificar con un valor dado?

1) Necesito **acumular**, es decir **un contador** en el que ir contando la cantidad de palabras codificadas con el valor dado.
2) Podría **calcular cuántas palabras** se pueden codificar con dicho valor **en cada subárbol** e ir **sumando 1 e**n el contador.
3) **¿Cómo calculo en cada subárbol?**

¿Cuál sería el valor de codificación de cada subárbol?



¿Cómo calculo la cantidad de palabras que se pueden codificar con un valor dado?

Sintetizando

1) Es un algoritmo recursivo, se aplica el mismo algoritmo en cada subárbol para resolver el mismo problema pero más chico.
2) Se achica el problema porque el árbol cada vez es más reducido.
3) **¿Cuáles son los datos de cada invocación recursiva?**
    a) El subárbol.
    b) El valor de codificación actual menos el valor de la raíz del árbol.
4) **¿Cuál es el caso base de la recursión?**
    a) Cuando llego a **una hoja y el valor coincide con el de la hoja**, ahí **encontré una codificación**.
    b) Cuando **llegué a una hoja pero el valor no coincide con el de la hoja**. Ese camino no me llevó a una codificación para ese valor.

Arboles Generales

Ejercicio de parcial

```
public class Gematria {
   public static int contadorGematria(ArbolGeneral<Integer> ag, int valor) {
          int resta = valor - ag.getDato();
          if (ag.esHoja() && resta == 0)
                return 1;
          else {
                int cont = 0;
                ListaGenerica<ArbolGeneral<Integer>> lista = ag.getHijos();
                lista.comenzar();
                while (!lista.fin()) {
                      ArbolGeneral<Integer> arbol = lista.proximo();
                      if (resta > 0)
                        cont = cont + contadorGematria(arbol, resta);
                }
                return cont;
          }
   }
}
```