








Arboles Generales

Estructura

<<Java Class>>

 **ArbolGeneral<T>**

tp04.ejercicio1

-  **ArbolGeneral(T)**
-  **ArbolGeneral(T, ListaGenerica<ArbolGeneral<T>>)**
-  **ArbolGeneral(NodoGeneral<T>)**
-  **getRaiz():NodoGeneral<T>**
-  **setRaiz(NodoGeneral<T>):void**
-  **getDatoRaiz()**
-  **getHijos():ListaGenerica<ArbolGeneral<T>>**








-raiz



<<Java Class>>

 **NodoGeneral<T>**

tp04.ejercicio1

-  **dato: T**
-  **listaHijos: ListaGenerica<NodoGeneral<T>>**
-  **NodoGeneral(T)**
-  **getDato()**
-  **setDato(T):void**
-  **getListaHijos():ListaGenerica<NodoGeneral<T>>**
-  **setListaHijos(ListaGenerica<NodoGeneral<T>>):void**

Arboles Generales

Código Fuente – Constructores, this()

```
package tp04;
public class ArbolGeneral<T> {
    private NodoGeneral<T> raiz;

    public ArbolGeneral(T dato) {
        raiz = new NodoGeneral<T>(dato);
    }
    public ArbolGeneral(T dato, ListaGenerica<ArbolGeneral<T>> hijos) {
        this(dato);
        ListaGenerica<NodoGeneral<T>> newList =
            new ListaEnlazadaGenerica<NodoGeneral<T>>();
        hijos.comenzar();
        while (!hijos.fin()) {
            ArbolGeneral<T> arbolTemp = hijos.proximo();
            newList.agregar(arbolTemp.getRaiz());
        }
        raiz.setListaHijos(newList);
    }
    private ArbolGeneral(NodoGeneral<T> nodo){
        raiz = nodo;
    }
    private NodoGeneral<T> getRaiz() {
        return raiz;
    }
    . . .
}
```

```
package tp04;
public class NodoGeneral<T> {
    private T dato;
    private ListaGenerica<NodoGeneral<T>> listaHijos;

    NodoGeneral(T dato) {
        this.dato=dato;
        listaHijos=new ListaEnlazadaGenerica<NodoGeneral<T>>();
    }
    // getters y setters
}
```

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 NodoGeneral<T> raiz;
    . . .
    public ListaEnlazadaGenerica<T> preOrden() {
        ListaEnlazadaGenerica<T> lis = new ListaEnlazadaGenerica<T>();
        this.getRaiz().preOrden(lis);
        return lis;
    }
}
```

```
package ayed;
public class NodoGeneral<T> {
    private T dato;
    private ListaGenerica<NodoGeneral<T>> listaHijos;
    . . .
    void preOrden(ListaGenerica<T> l) {
        l.agregar(this.getDatos());
        ListaGenerica<NodoGeneral<T>> lHijos = this.getListaHijos();
        lHijos.comenzar();
        while(!lHijos.fin()) {
            (lHijos.proximo()).preOrden(l);
        }
    }
}
```

Caso de uso

```
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.agregar(a1); hijos.agregar(a2); hijos.agregar(a3);
ArbolGeneral<String> a = new ArbolGeneral<String>("0", hijos);
System.out.println("Datos del Arbol: "+a.preOrden());
```

Arboles Generales

Ejercicio de parcial - Mínimo Caudal

Sea una red de agua potable, la cual comienza en un caño maestro y el mismo se va dividiendo sucesivamente hasta llegar a cada una de las casas.

Por el caño maestro ingresan x cantidad de litros y en la medida que el caño se divide, el caudal se divide en partes iguales en cada una de las divisiones. Es decir, si un caño maestro recibe 1000 litros y se divide en 4 partes, cada división tiene un caudal de 250 litros. Luego, si una de esas divisiones se vuelve a dividir en 5 partes, cada una tendrá un caudal de 50 litros y así sucesivamente.

Usted debe implementar un método en la clase `ArbolGeneral`, que calcule el caudal de cada nodo y determine cuál es el mínimo caudal que recibe una casa.

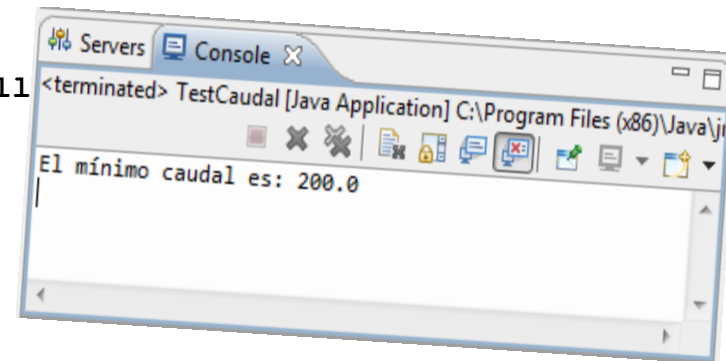
Arboles Generales

Ejercicio de parcial - Mínimo Caudal

```
double minimoCaudal(double caudalEntrante) {
    double minCaudal = caudalEntrante;
    ListaGenerica<ArbolGeneral<T>> hijos = this.getHijos();
    if (!(hijos.esVacia())) {
        hijos.comenzar();
        while (!hijos.fin()) {
            minCaudal = Math.min(minCaudal, hijos.proximo().minimoCaudal(caudalEntrante/hijos.tamano()));
        }
    }
    return minCaudal;
}
```

Caso de uso

```
ArbolGeneral<String> raiz = new ArbolGeneral<String>("central ABSA");
ArbolGeneral<String> h1 = new ArbolGeneral<String>("Tolosa");
ArbolGeneral<String> h2 = new ArbolGeneral<String>("Gonnet");
ArbolGeneral<String> h3 = new ArbolGeneral<String>("Villa Castell");
ArbolGeneral<String> h31 = new ArbolGeneral<String>("casa 31");
ArbolGeneral<String> h32 = new ArbolGeneral<String>("casa 32");
ArbolGeneral<String> h33 = new ArbolGeneral<String>("casa 33");
ArbolGeneral<String> h34 = new ArbolGeneral<String>("casa 34");
h3.agregarHijo(h31);
h3.agregarHijo(h32);
h3.agregarHijo(h33);
h3.agregarHijo(h34);
raiz.agregarHijo(h1); raiz.agregarHijo(h2); raiz.agregarHijo(h3);
System.out.println("El mínimo caudal es: "+raiz.minimoCaudal(2400));
```

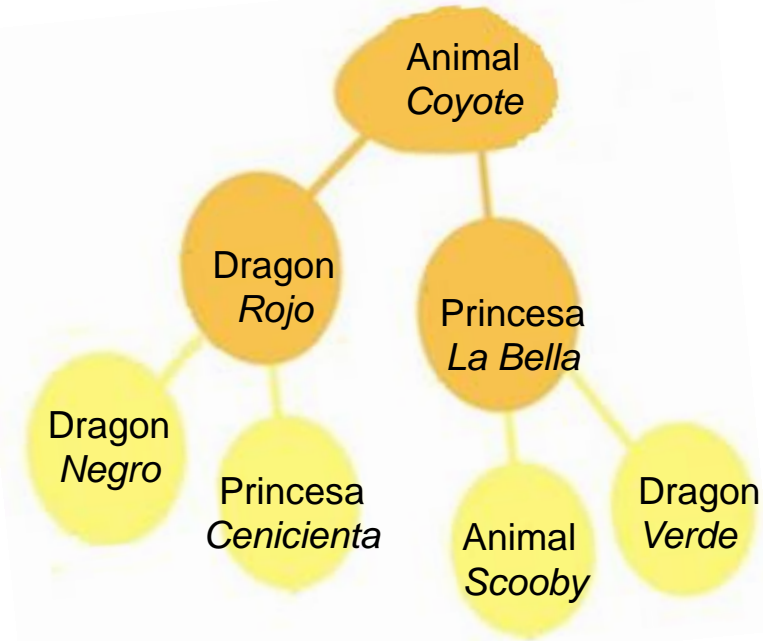


Arboles Generales

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

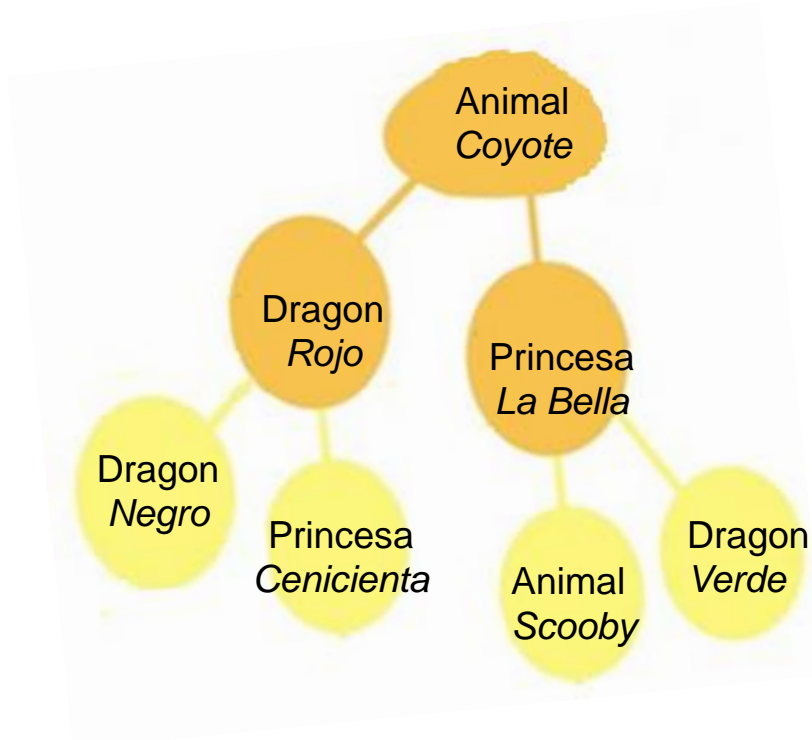
Ejercicio de parcial – Encontrar a la Princesa

```
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

```
package parcial.juego;
```

```
...
```

```
public class Juego {
```

```
    private ArbolGeneral<Personaje> personajes = null;
```

```
    boolean encuentre = false;
```

```
    ListaGenerica<Personaje> camino = new ListaEnlazadaGenerica<Personaje>();
```

```
    public void encontrarPrincesa(ArbolGeneral<Personaje> arbol) {
```

```
        ListaGenerica<Personaje> lista = new ListaEnlazadaGenerica<Personaje>();
```

```
        lista.agregar(arbol.getDatoRaiz());
```

```
        encontrarPrincesa(arbol, lista);
```

```
        System.out.print("Se encontró a la Princesa en el camino: " + camino);
```

```
    }
```

```
    public void encontrarPrincesa(ArbolGeneral<Personaje> arbol, ListaGenerica<Personaje> lista) {
```

```
        Personaje p = arbol.getDatoRaiz();
```

```
        if (p.esPrincesa()) {
```

```
            encuentre = true;
```

```
            camino = this.duplicar(lista);
```

```
        }
```

```
        if (!encontre) {
```

```
            ListaGenerica<ArbolGeneral<Personaje>> lHijos = arbol.getHijos();
```

```
            lHijos.comenzar();
```

```
            while (!lHijos.fin()) {
```

```
                aux = lHijos.proximo();
```

```
                if (!aux.getDatoRaiz().esDragon()) {
```

```
                    lista.agregarFinal(aux.getDatoRaiz());
```

```
                    encontrarPrincesa(aux, lista);
```

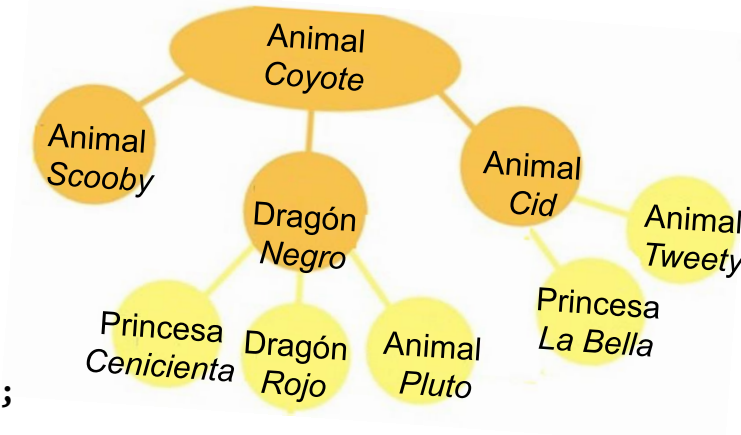
```
                    lista.eliminarEn(lista.tamano());
```

```
                }
```

```
            }
```

```
        }
```

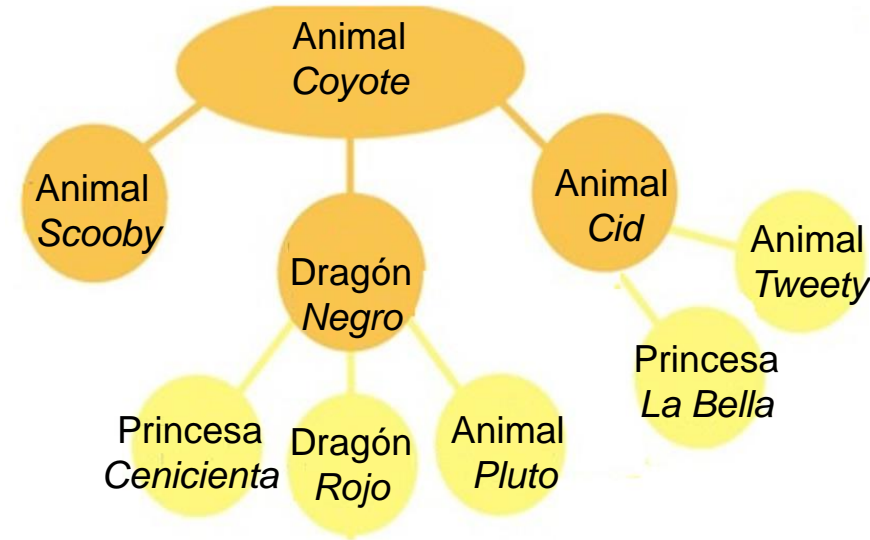
```
    }}
```



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.agregar(a21, hijosa2.tamano());
    hijosa2.agregar(a22, hijosa2.tamano());
    hijosa2.agregar(a23, hijosa2.tamano());
    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

Antiguamente el pueblo judío usaba un sistema de numeración llamado Gematria 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).

Usted cuenta con una estructura como la que aparece en el gráfico, donde **cada camino en este árbol representa 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.”**

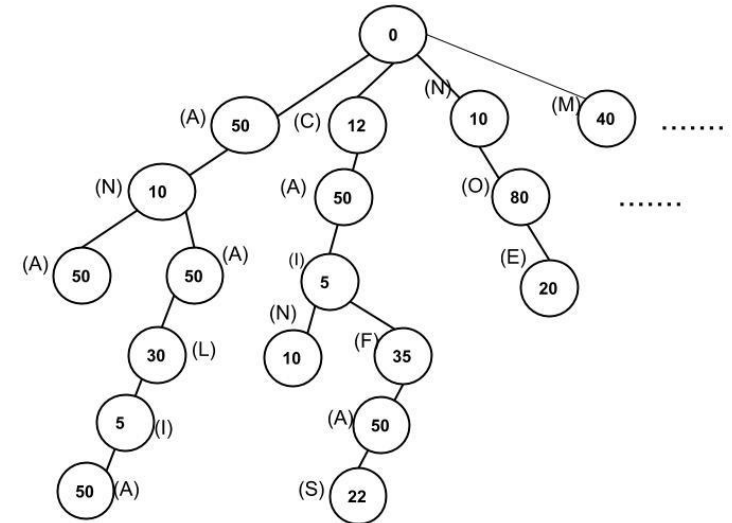
Su tarea será: **dado un valor numérico, contar cuantos nombres** completos suman exactamente ese **valor**. Usted recibe el árbol con las letras ya sustituidas por sus valores; las letras ya no importan.

Para esto, escriba una clase llamada **ProcesadorGematría** (que NO contenga variables de instancia), con sólo un método público con la siguiente firma:

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

estructura que contiene
los números

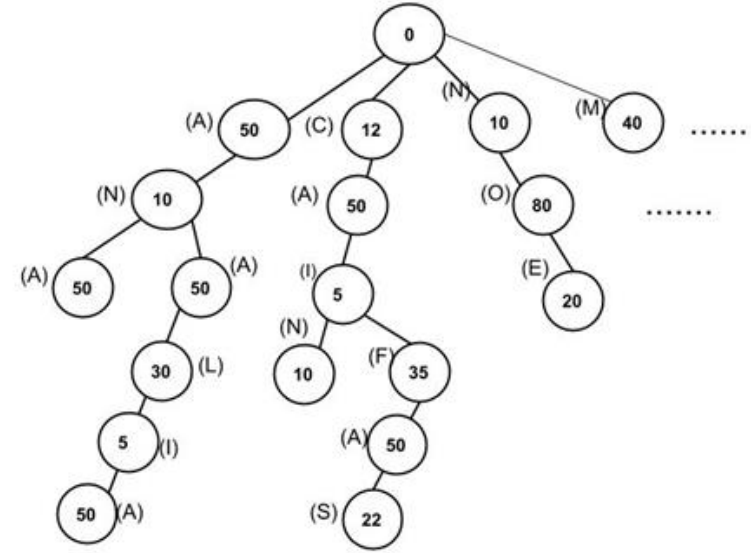
valor es el valor que se debería
obtener al sumar el valor de las
letras de un nombre



Estructura de números que representa nombres. Dado el valor 110 el método debe devolver 2 (porque ANA y NOE suman 110), dado el valor 77 el método debe devolver 1 (porque sólo CAIN suma 77).

Arboles Generales

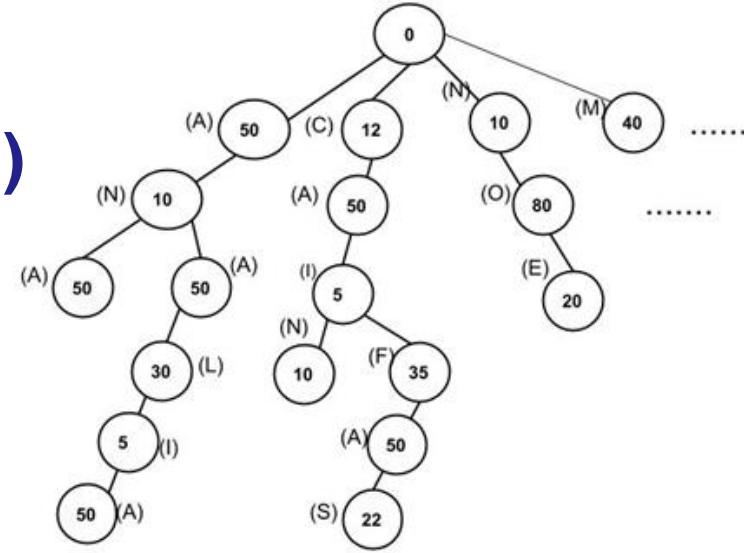
Ejercicio de Parcial (versión 1)



```
public static int contadorGematria(ArbolGeneral<Integer> ag, int valor) {
    int resta = valor - ag.getDatoRaiz();
    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;
    }
}
```

Arboles Generales

Ejercicio de Parcial (versión 2)



```

public static int contadorGematriaBis(ArbolGeneral<Integer> ag, int valor) {
    int[] contador = { 0 };
    contadorGematriaBis(ag, valor, contador, 0);
    return contador[0];
}

private static void contadorGematriaBis(ArbolGeneral<Integer> ag, int valor, int[] contador, int suma) {
    if ((suma == valor) && (ag.esHoja())) {
        contador[0]++;
    } else {
        ListaGenerica<ArbolGeneral<Integer>> lista = ag.getHijos();
        lista.comenzar();
        while (!lista.fin()) {
            ArbolGeneral<Integer> arbol = lista.proximo();
            if (suma + arbol.getDatoRaiz() <= valor)
                contadorGematriaBis(arbol, valor, contador, suma + arbol.getDatoRaiz());
        }
    }
}

```