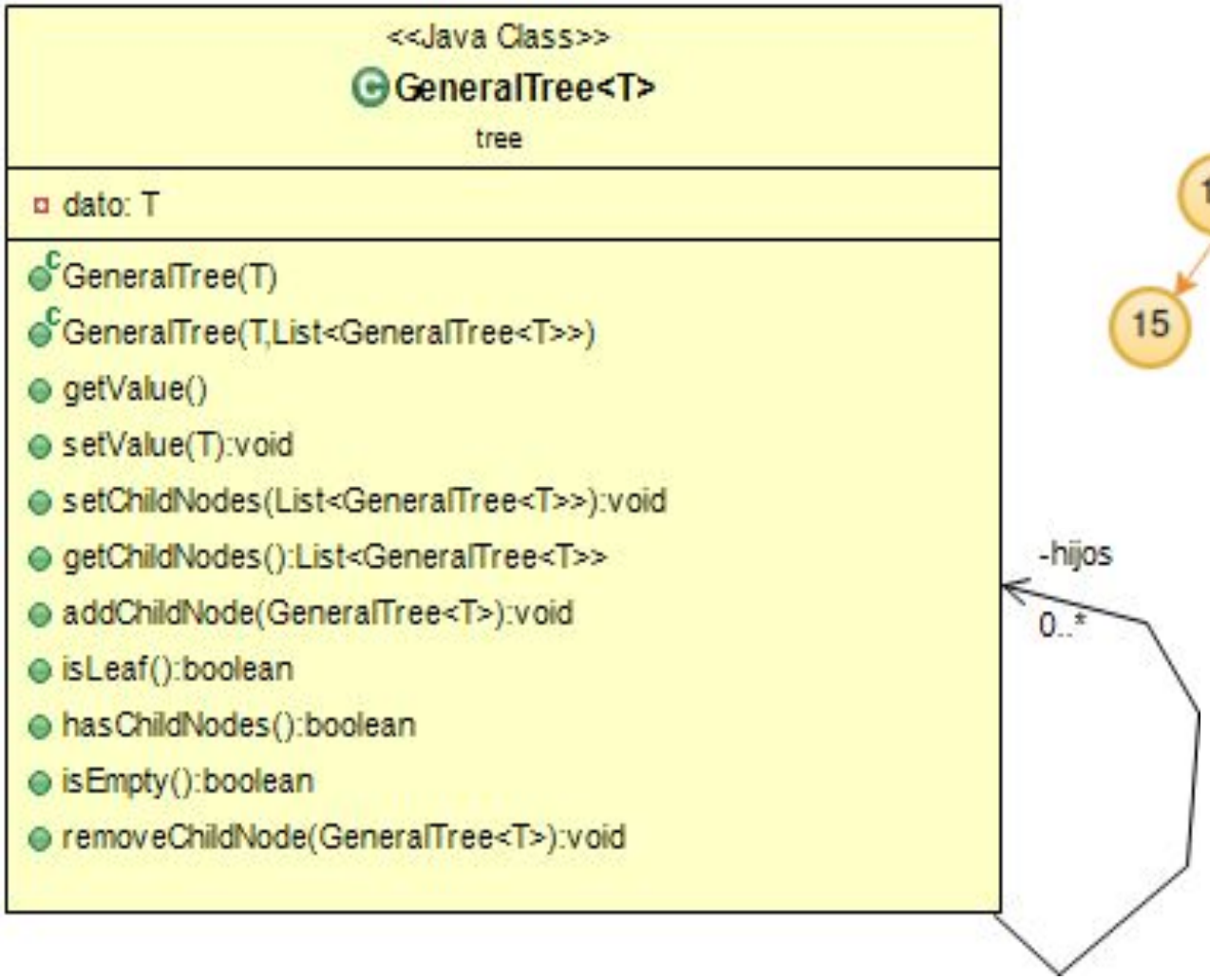
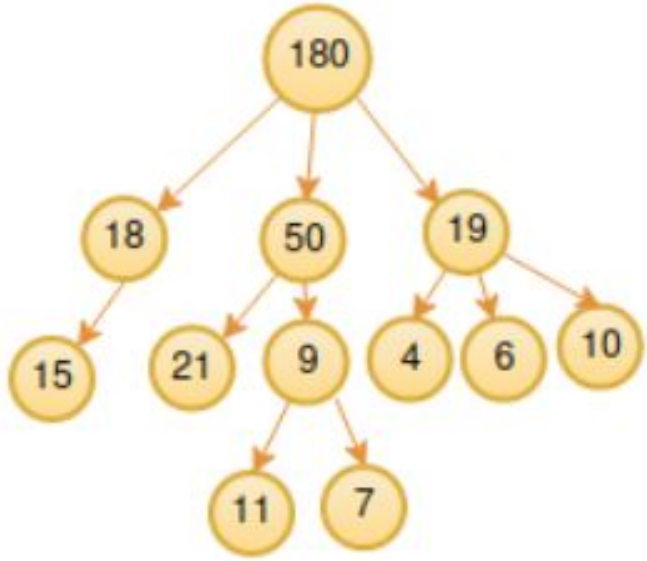
Arboles Generales

Implementación en JAVA
Ejemplos de parciales

Arboles Generales

Estructura





Arboles Generales

Código Fuente – Constructores, this()

```
public class GeneralTree<T> {

   private T dato;
   private List<GeneralTree<T>> hijos =
                        new LinkedList<GeneralTree<T>>();

   public GeneralTree(T dato) {
       this.dato = dato;
   }

   public GeneralTree(T dato, List<GeneralTree<T>> hijos){
      this(dato);
      if (hijos==null)
           this.hijos = new LinkedList<GeneralTree<T>>();
       else
       this.hijos = hijos;
   }

   public T getValue() {
       return dato;
   }

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

   public void setChildNodes(List<GeneralTree<T>> hijos) {
     if (hijos==null)
        this.hijos = new LinkedList<GeneralTree<T>>();
       else
        this.hijos = hijos;
  }
```

```
  public List<GeneralTree<T>> getChildNodes() {
        return this.hijos;
  }


  public void addChildNode(GeneralTree<T> unHijo) {
        this.getChildNodes().add(unHijo);
  }

  public boolean isLeaf() {
        return !this.hasChildNodes();
  }

  public boolean hasChildNodes() {
        return this.hijos!=null && !this.hijos.isEmpty();
  }

  public boolean isEmpty() {
        return this.dato == null && !this.hasChildNodes();
  }

  public void removeChildNode(GeneralTree<T> hijo) {
     if (this.hasChildNodes()) {
         List<GeneralTree<T>> hijos = this.getChildNodes();
         if (hijos.contains(hijo))
             hijos.remove(hijo);
         }
     }
}
...

}
```

Arboles Generales
Recorrido PreOrden

Implementar un método en `GeneralTree` que retorne una lista con los datos del árbol recorrido en preorden

```
package ayed;
public class GeneralTree<T> {

     public List<T> preOrden() {
      List<T> lis = new LinkedList<T>();
      this.preOrden(lis);
      return lis;
    }
    private void preOrden(List<T> l) {
      l.add(this.getValue());
      List<GeneralTree<T>> childs = this.getChildNodes();
      for(GeneralTree<T> child : childs ) {
           child.preOrden(l);
      }
    }
}
```

Ejemplo de uso:

```
GeneralTree<String> a1 = new GeneralTree<String>("1");
GeneralTree<String> a2 = new GeneralTree<String>("2");
GeneralTree<String> a3 = new GeneralTree<String>("3");
List<GeneralTree<String>> childs= new LinkedList<GeneralTree<String>>();
childs.add(a1);
childs.add(a2);
childs.add(a3);
GeneralTree<String>  a = new GeneralTree<String>("0", childs);
System.out.println("Datos del Arbol: "+a.preOrden());
```

Arboles Generales
Recorrido por niveles

```
public void porNiveles() {
   encolar(raíz);
   mientras cola no se vacíe {
         v ← desencolar();
         imprimir (dato de v);
         para cada hijo de v
               encolar(hijo);
   }
}
```

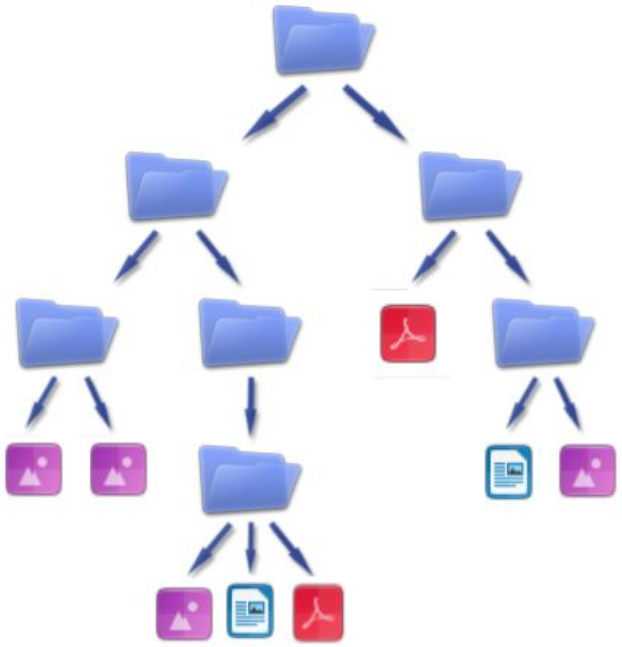
Recorrido por niveles en una clase externa a `GeneralTree`.

```
public List<T> getPorNiveles(GeneralTree <T> arbol) {

 List<T> result = new LinkedList<T>();
 Queue<GeneralTree<T>> queue = new Queue<GeneralTree<T>>();
 GeneralTree<T> tree_aux;
 queue.enqueue(arbol);
 while (!queue.isEmpty()) {
      tree_aux = queue.dequeue();
      result.add(tree_aux.getValue());
      List<GeneralTree<T>> childs = tree_aux.getChildNodes();
      for(GeneralTree<T> child : childs ) {
           queue.enqueue(child);
      }
 }
 return result;
}
```

Arboles Generales

Ejercicio: Devolver las imágenes de un nivel

Dado un árbol que representa una estructura de directorios como la que muestra la imagen, implementar un método que reciba un nivel y retorne una lista con las imágenes encontradas en ese nivel.
Modelar el recurso para representar las carpetas y los archivos.

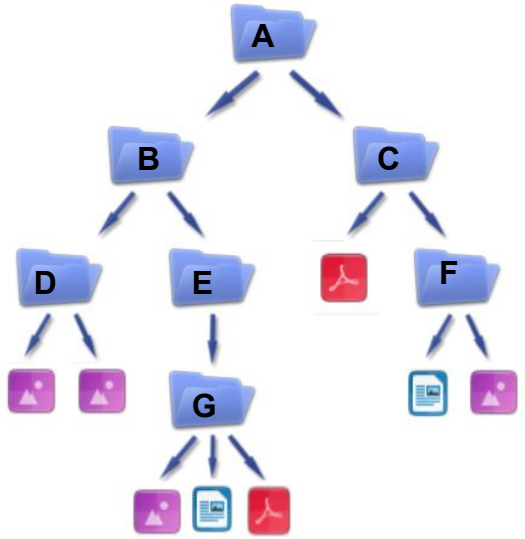


```
public class Recurso {
  String nombre;
  String tipo; // archivo o carpeta


  public Recurso(String nombre, String tipo) {
       super();
       this.nombre = nombre;
       this.tipo = tipo;
  }

  public boolean esImagen() {
  if (tipo.equals("archivo")) {
       String ext = nombre.substring(nombre.indexOf('.') + 1);
       if (ext.equals("jpg") || ext.equals("png") || ext.equals("jpeg"))
                  return true;
       }
       return false;
    }
  . . .
}
```

Arboles Generales

Ejercicio de parcial – Devolver imágenes

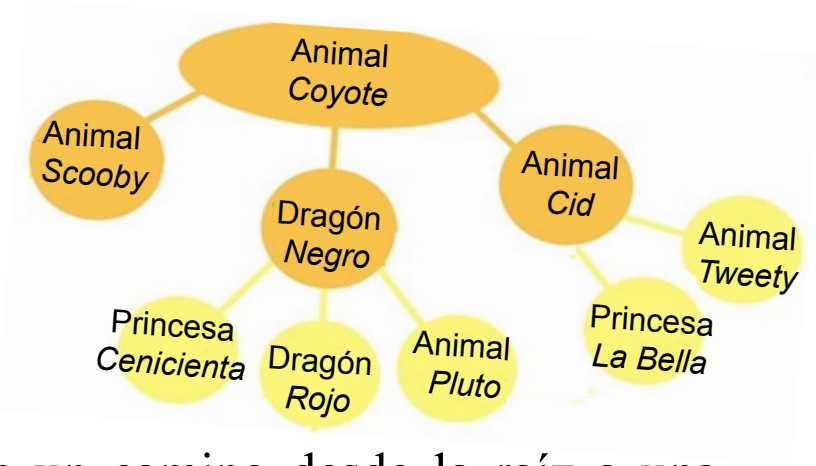
```
public List<Recurso> getImagenes(GeneralTree<Recurso> gt, int nivel_pedido) {
    List<Recurso> result = new LinkedList<Recurso>();
    Queue<GeneralTree<Recurso>> queue = new Queue<GeneralTree<Recurso>>();
    GeneralTree<Recurso> tree_aux;
    queue.enqueue(gt);
    queue.enqueue(null);
    int nivel = 0;
    while (!queue.isEmpty() && nivel<=nivel_pedido) {
      tree_aux= queue.dequeue();
      if (tree_aux != null) {
           if (nivel==nivel_pedido && tree_aux.getValue().esImagen() )
                result.add(tree_aux.getValue());
           if (tree_aux.hasChildNodes() && nivel<nivel_pedido) {
                List<GeneralTree<Recurso>> childs = tree_aux.getChildNodes();
                for(GeneralTree<Recurso> child : childs ) {
                     queue.enqueue(child);
                }
           }
      } else {
           if (!queue.isEmpty()) {
                nivel++;
                queue.enqueue(null);
           }
      }
   }
   return result;
 }
```



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.

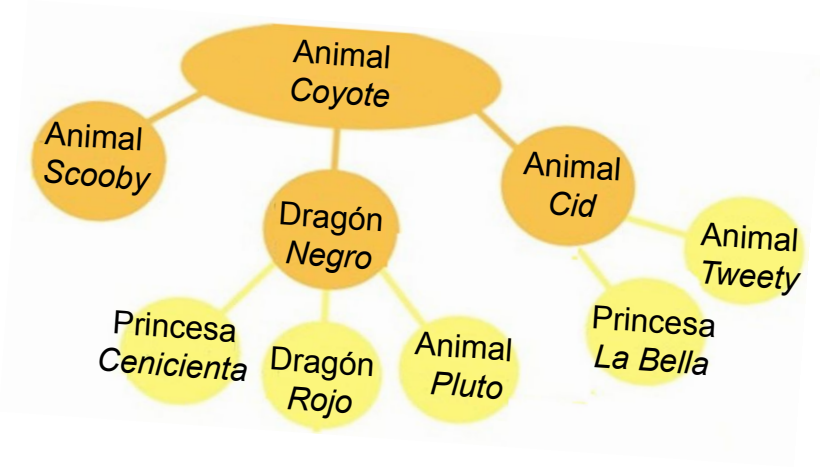


Implementar un método que devuelva una lista con un camino desde la raíz a una Princesa sin pasar por un Dragón –sin necesidad de ser el más cercano a la raíz-. Asuma que existe al menos un camino accesible.

Arboles Generales

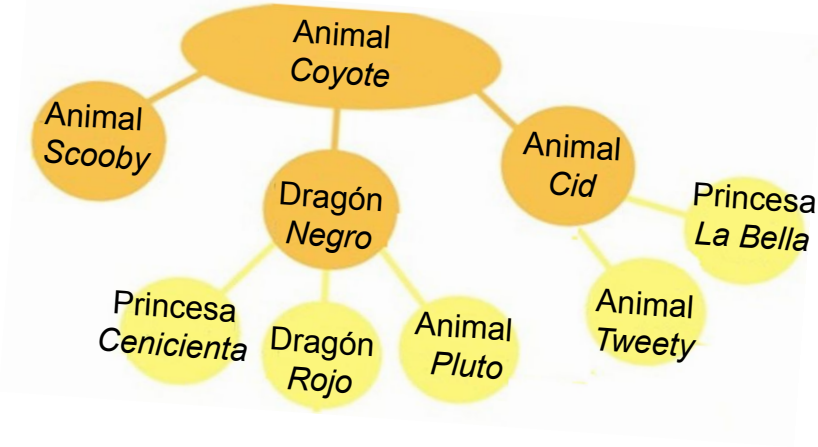
Ejercicio de parcial – Encontrar a la Princesa

```
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

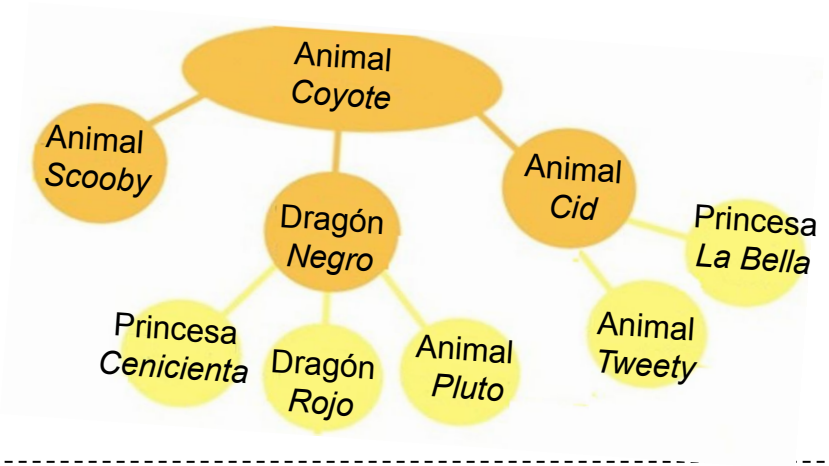



```
public class Juego {
    public void encontrarPrincesa(GeneralTree<Personaje> arbol) {
          List<Personaje> lista = new LinkedList<Personaje>();
          lista.add(0, arbol.getValue());
          List<Personaje> camino = new LinkedList<Personaje>();
          camino = encontrarPrincesa(arbol, lista);
          System.out.print("Se encontró a la Princesa en el camino: " + camino);
    }
    private List<Personaje>encontrarPrincesa(GeneralTree<Personaje> arbol, List<Personaje> lista) {
          List<Personaje> camino
          Personaje p = arbol.getValue();
          if (p.esPrincesa()) {
             camino = new LinkedList<Personaje>(lista);
          }
          if (camino.isEmpty()) {
                List<GeneralTree<Personaje>> childs = arbol.getChildNodes();
                for(GeneralTree<Personaje> child : childs ) {
                     if (!child.getValue().esDragon()) {
                          lista.add(child.getValue());
                          camino = encontrarPrincesa(child, lista);
                          if (!camino.isEmpty()) {
                               return camino;
                          }
                          lista.remove(lista.size()-1);
                     }

                }
          }
          return camino;
```

Arboles Generales

Ejercicio de parcial – Encontrar a la Princesa
Versión II

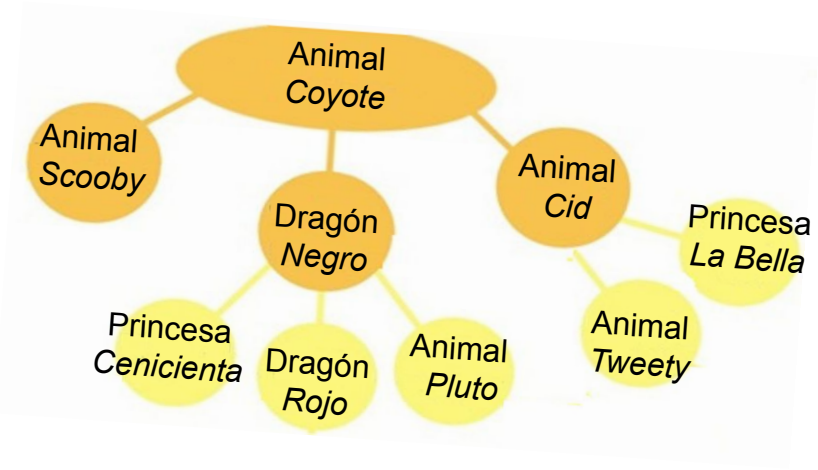
```
public class Juego {

  public List<Personaje> encontrarPrincesa(GeneralTree<Personaje> arbol){
    LinkedList<Personaje> lista = null;
    if (arbol.getValue().esPrincesa() || arbol.getValue().esDragon() || arbol.isLeaf()){
      if (arbol.getValue().esPrincesa()){
        lista=new LinkedList<Personaje>();
        Personaje p = arbol.getValue();
        lista.add(0, p);
      }
      return lista;
    }
    List<GeneralTree<Personaje>> childs = arbol.getChildNodes();
    for(GeneralTree<Personaje> child : childs ) {
        lista = encontrarPrincesa(child);
        if(lista!=null){
              lista.add(0, arbol.getValue());
              return lista;
        }

    }
    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");
              GeneralTree<Personaje> a1 = new GeneralTree<Personaje>(p0);
              GeneralTree<Personaje> a21 = new GeneralTree<Personaje>(p1);
              GeneralTree<Personaje> a22 = new GeneralTree<Personaje>(p2);
              GeneralTree<Personaje> a23 = new GeneralTree<Personaje>(p3);
              GeneralTree<Personaje> a31 = new GeneralTree<Personaje>(p6);
              GeneralTree<Personaje> a32 = new GeneralTree<Personaje>(p5);
              List<GeneralTree<Personaje>> hijosa2 = new LinkedList<GeneralTree<Personaje>>();
              hijosa2.add(a21);
              hijosa2.add(a22);
              hijosa2.add(a23);
              List<GeneralTree<Personaje>> hijosa3 = new LinkedList<GeneralTree<Personaje>>();
              hijosa3.add(a31);
              hijosa3.add(a32);
              GeneralTree<Personaje> a2 = new GeneralTree<Personaje>(p4, hijosa2);
              GeneralTree<Personaje> a3 = new GeneralTree<Personaje>(p7, hijosa3);
              List<GeneralTree<Personaje>> hijos = new LinkedList<GeneralTree<Personaje>>();
              hijos.add(a1);
              hijos.add(a2);
              hijos.add(a3);
              GeneralTree<Personaje> a = new GeneralTree<Personaje>(p8, hijos);
              Juego juego = new Juego();
              juego.encontrarPrincesa(a);
 }
}
```

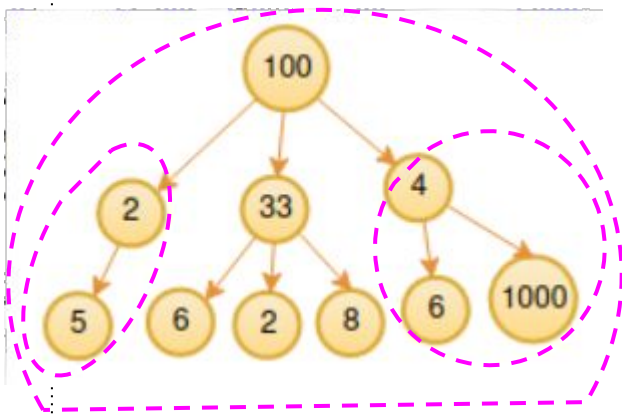


Arboles Generales

Ejercicio de parcial - Encontrar árboles de padres más pequeño

Implementar una clase con un método que reciba un árbol general de enteros y retorne todos los árboles cuya raíz tenga un valor más pequeño que la suma de los valores de sus descendientes.

```
public class BuscarPadreMenor {

     public static int buscar(GeneralTree<Integer> ag) {
          if (ag.isLeaf()) {
               return ag.getValue();
          }
          int cont = 0;
          List<GeneralTree<Integer>> childs = ag.getChildNodes();
          for(GeneralTree<Integer> child : childs ) {
               cont = cont + buscar(child);
          }
          if (ag.getValue() < cont) {
               //guardo en otra clase o en una lista
               Repositorio.add(ag);
          }
          return cont+ag.getValue();
     }
}
```



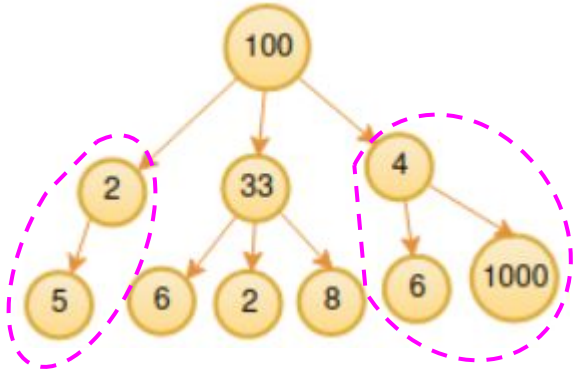

```
public class Repositorio {

private static List<GeneralTree<Integer>> lista =
    new LinkedList<GeneralTree<Integer>>();

    public static void add(GeneralTree<Integer> a) {
       lista.add(a);
    }
     . . .
}
```

Arboles Generales

Ejercicio de parcial - Encontrar árboles de padres más pequeño

¿Qué cambios se deberían hacer para devolver todos los árboles cuya raíz tenga un valor más pequeño que la suma de los valores de sus hijos? y si no se puede usar una clase auxiliar?

```
public class BuscarPadreMenor {
 public static List<GeneralTree<Integer>> buscarDatoHijos(GeneralTree<Integer> ag) {
   List<GeneralTree<Integer>> listaArboles = new LinkedList<GeneralTree<Integer>>();
   buscarDatoHijos(ag, listaArboles);
   return listaArboles;
 }

 private static int buscarDatoHijos(GeneralTree<Integer> ag,List<GeneralTree<Integer>> listaArboles){
   if (ag.isLeaf()) {
      return ag.getValue();
   }
   int cont = 0;
   List<GeneralTree<Integer>> childs = ag.getChildNodes();
   for(GeneralTree<Integer> child : childs ) {
      cont = cont + buscarDatoHijos(child, listaArboles);
   }
   if (ag.getValue() < cont) {
      listaArboles.add(ag);
   }
   return cont+ag.getValue();
 }
}
```



Arboles Generales

Ejercicio de Parcial - Gematría

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.**

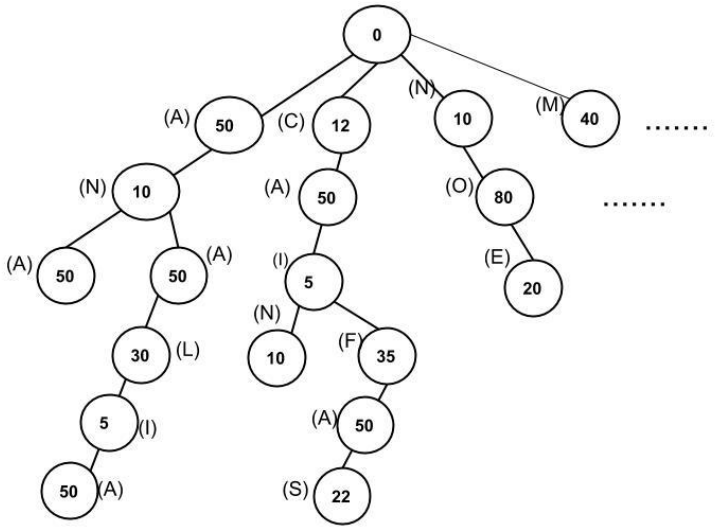
Su tarea será: **dado un valor numérico**, **contar cuántos nombres** completos suman exactamente ese **valor**. Usted recibe el árbol con las letras ya sustituidas por sus valores.

Para esto, escriba una clase llamada **`ProcesadorGematria`** (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. Se supone >0



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 - Gematría

```
public class Gematria {
    public static int contadorGematria(GeneralTree<Integer> ag, int valor) {
      int resta = valor - ag.getValue();
      if (ag.isLeaf() && resta == 0)
           return 1;
      else {
           int cont = 0;
           if (resta > 0) {
                List<GeneralTree<Integer>> childs = ag.getChildNodes();
                for(GeneralTree<Integer> child : childs ) {
                     cont = cont + contadorGematria(child, resta);
                }
            }
           return cont;
      }
    }
}
```

supongamos que buscamos palabras que sumen 110