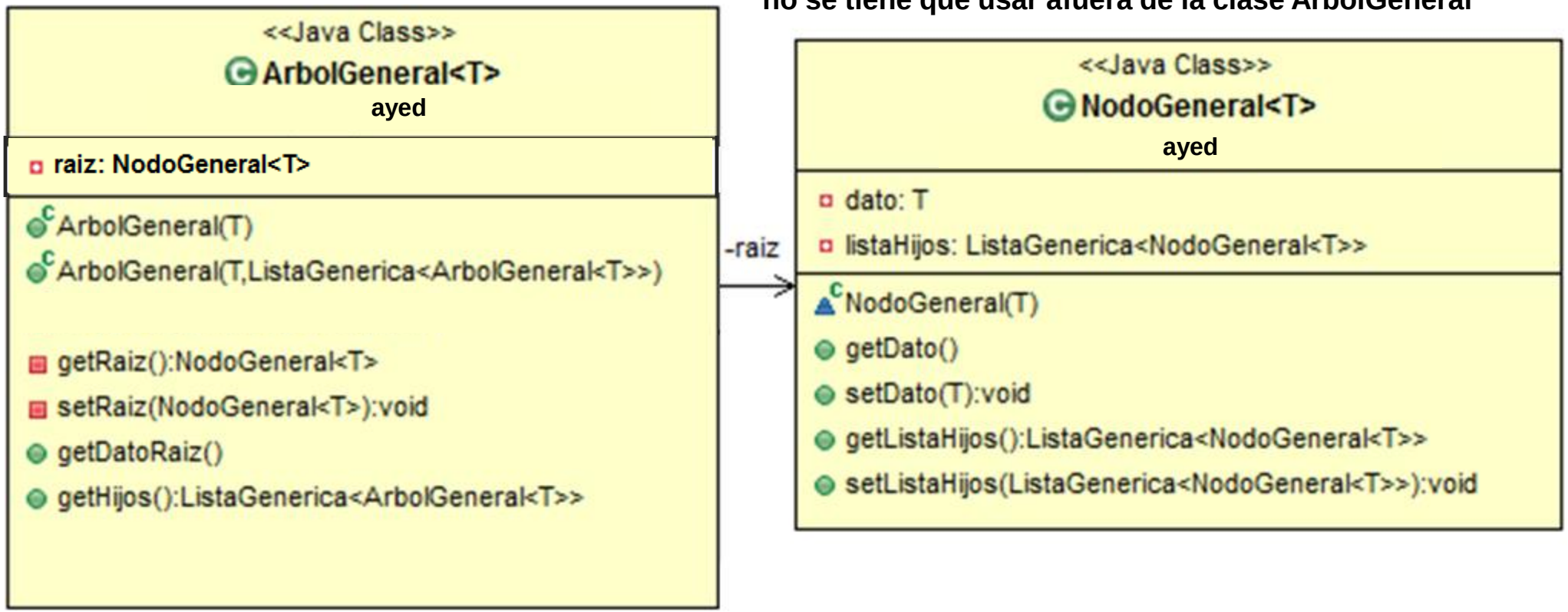
Agenda

Árboles Generales

- -Implementación en JAVA
- -Ejemplos
  - -Recorrido Pre-Orden
  - -Contar la Cantidad de Nodos de un Árbol: 2 soluciones
  - -Búsqueda de un camino “accesible” a una princesa

Arboles Generales
Diseño de clases

La clase NodoGeneral es utilitaria de la clase ArbolGeneral, no se tiene que usar afuera de la clase ArbolGeneral



altura(), ancho, esVacio(), preOrden(), etc se definen en la clase ArbolGeneral<T>

Si necesitamos métodos que usan la clase NodoGeneral, dichos métodos los declaramos **privados**

Arboles Generales

```
package ayed;
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>> newLista =
            new ListaEnlazadaGenerica<NodoGeneral<T>>();
   hijos.comenzar();
   while (!hijos.fin()) {
     ArbolGeneral<T> arbolTemp = hijos.proximo();
     newLista.agregar(arbolTemp.getRaiz());
   }
   raiz.setListaHijos(newLista);
 }

 private NodoGeneral<T> getRaiz() {
   return raiz;
 }

// AQUÍ se declaran todos los métodos que manipulan AG
//altura(), ancho, esVacio(), preOrden()

}
```

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

NodoGeneral(T dato) {
 this.dato = dato;
 listaHijos=new ListaEnlazadaGenerica<NodoGeneral<T>>();
}

public T getDato() {
  return dato;
}

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

public ListaGenerica<NodoGeneral<T>> getListaHijos()
 {
  return listaHijos;
 }

public void setListaHijos(ListaGenerica<NodoGeneral<T>>
                                              listaHijos)
{
    this.listaHijos = listaHijos;
}
}
```

Arboles Generales
Recorrido 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;
  }
}
```

El método **preorden()** visita los nodos del árbol en PREORDEN y devuelve una lista que contiene estos datos (en ese orden)

Visita la Raíz
Visita el sub-árbol izquierdo
Visita el sub-árbol derecho

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

Aquí se usan objetos como parámetro

Ejemplo

```
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);
System.out.println("Datos del Arbol: "+a.preOrden());
```

Arboles Generales

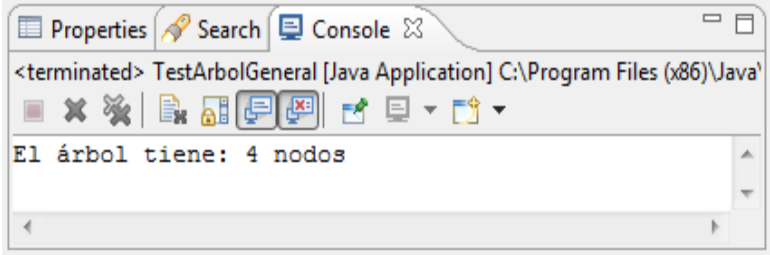
Contar cantidad de nodos

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

public int contarEnPreOrden() {
 return this.getRaiz().contarEnPreOrden();
}

}
```

```
package ayed;

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

 int contarEnPreOrden(){
  int res=1;
  ListaGenerica<NodoGeneral<T>> lHijos =
                                    this.getListaHijos();
  lHijos.comenzar();
  while(!lHijos.fin()){
     res = res + lHijos.proximo().contarEnPreOrden();
  }
  return res;
 }
}
```

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.agregarFinal(a1); hijos.agregarFinal(a2); hijos.agregarFinal(a3);
ArbolGeneral<String>  a = new ArbolGeneral<String>("0", hijos);
System.out.println("El árbol tiene:"+a.contarEnPreOrden()+" nodos");
```



Arboles Generales

Contar cantidad de nodos – otra versión

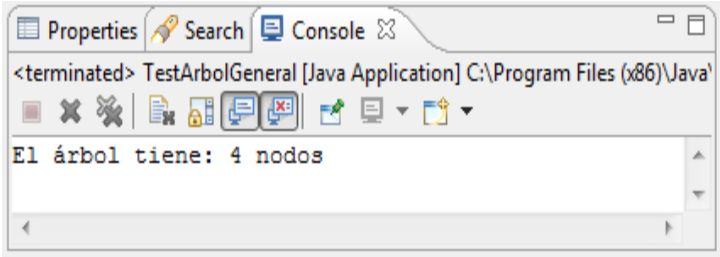
```
package ayed;
public class ArbolGeneral {
private NodoGeneral<T> raiz;

public int contarEnPreOrden(){
 int[] a = new int[1];
 this.getRaiz().contarEnPreOrden(a);
 return a[0];
}
}
```

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

 void contarEnPreOrden(int[] x){
  x[0]++;
  ListaGenerica<NodoGeneral<T>> lHijos =
                                        this.getListaHijos();
  lHijos.comenzar();
  while(!lHijos.fin()){
     lHijos.proximo().contarEnPreOrden(x);
  }
 }
}
```

Aquí se usan objetos como parámetro (arreglo)

Caso de uso

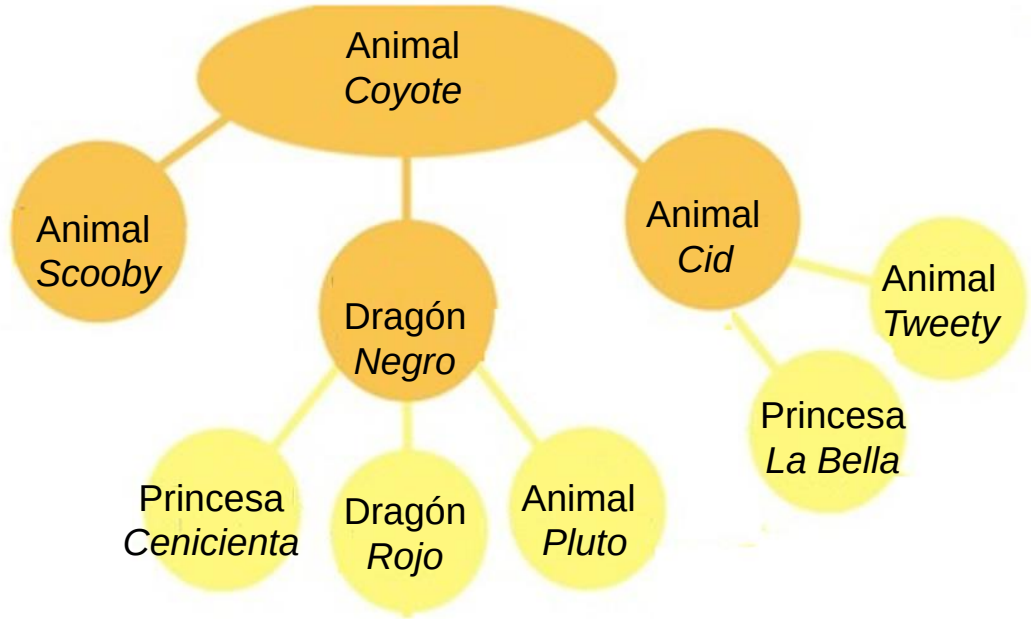
```
ArbolGeneral<String> a1 = new ArbolGeneral<String>("1");
ArbolGeneral<String> a2 = new ArbolGeneral<String>("2");
ArbolGeneral<String> a3 = new ArbolGeneral<String>("3");
ListaGenerica<ArbolGeneral<String>> hijos = new
ListaEnlazadaGenerica<ArbolGeneral<String>>();
hijos.agregarFinal(a1); hijos.agregarFinal(a2); hijos.agregarFinal(a3);
ArbolGeneral<String>  a = new ArbolGeneral<String>("0", hijos);
System.out.println("El árbol tiene: "+arbol.contarEnPreOrden() +" nodos");
```



Arboles Generales

Camino accesible a una princesa

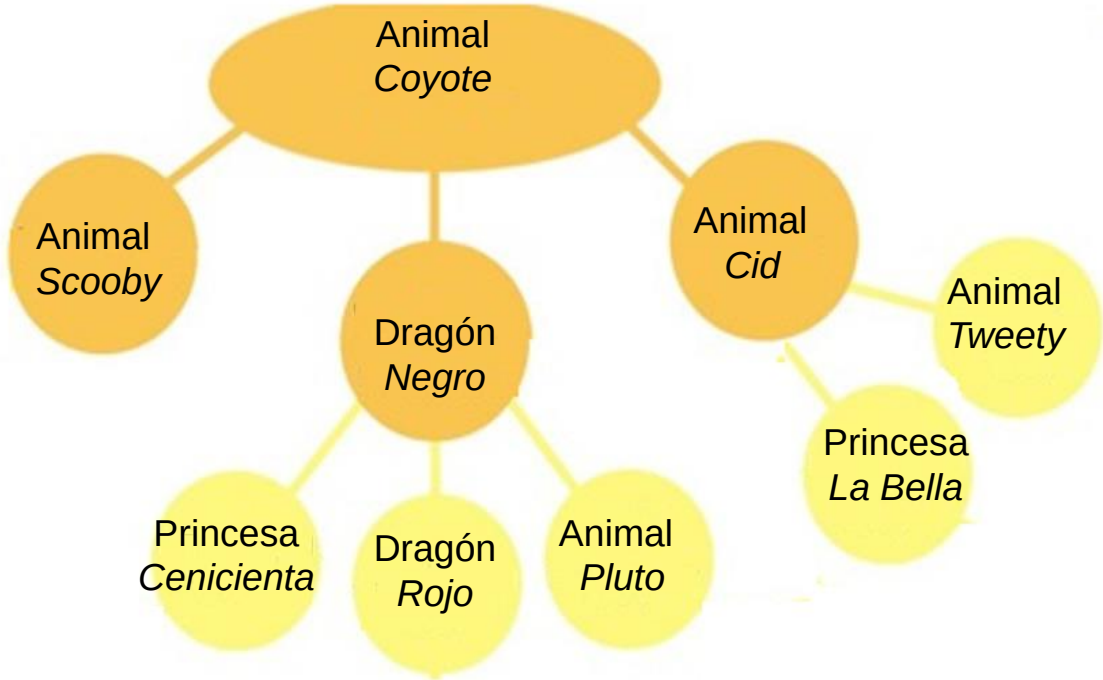
Dado un árbol general compuesto por personajes, que pueden ser **dragones**, **princesas** y **otros**, se denominan **nodos accesibles** a aquellos nodos que a lo largo del camino desde el nodo raíz hasta un nodo (ambos inclusive) no se encuentra ningún **Dragón**.
Implementar un método que retorne el camino a una “princesa accesible”.



Encontrar un camino accesible a una 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 String getTipo(){
  return return tipo;
 }

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

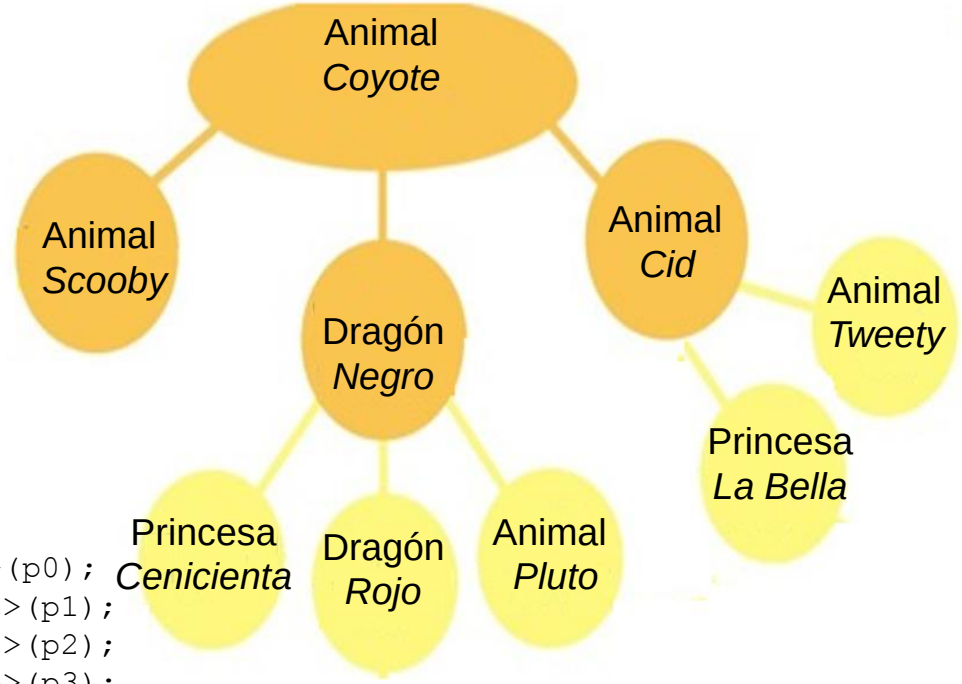
Encontrar un camino a una Princesa sin pasar por un Dragón



Encontrar un camino accesible a una princesa

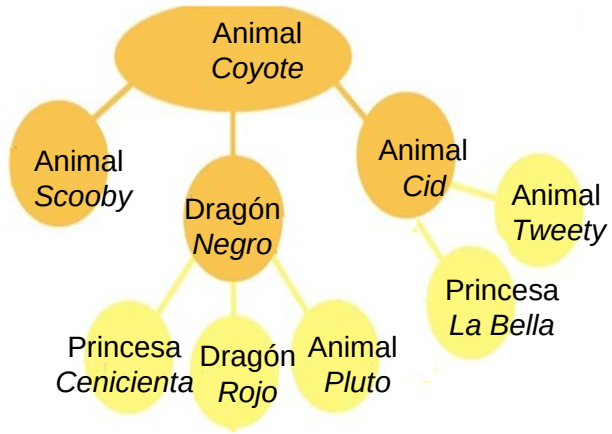
Armado del árbol

```
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.tamanio());
 hijosa2.agregarFinal(a22, hijosa2.tamanio());
 hijosa2.agregarFinal(a23, hijosa2.tamanio());
 ArbolGeneral<Personaje> a2 = new ArbolGeneral<Personaje>(p4, hijosa2);
 //se constuyen primero los subárboles y finalmente el árbol
 ArbolGeneral<Personaje>  a = new ArbolGeneral<Personaje>(p8, hijos);
 Juego juego = new Juego();  //implementado en la próxima página
 juego.encontrarPrincesa(a);
  }
}
```



Encontrar un camino accesible a una princesa

```
package parcial.juego;
public class Juego {
 private ArbolGeneral<Personaje> personajes = null;
 boolean encontre = false;
 ListaGenerica<Personaje> camino =  new ListaEnlazadaGenerica<Personaje>();

 public void encontrarPrincesa(ArbolGeneral<Personaje> arbol) {
    ListaGenerica<Personaje> lista = new ListaEnlazadaGenerica<Personaje>();
    lista.agregarFinal(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()) {
      encontre = 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.tamanio());
              }
       }
  }
 }}
```



Aquí se usa una variable de instancia para devolver un resultado