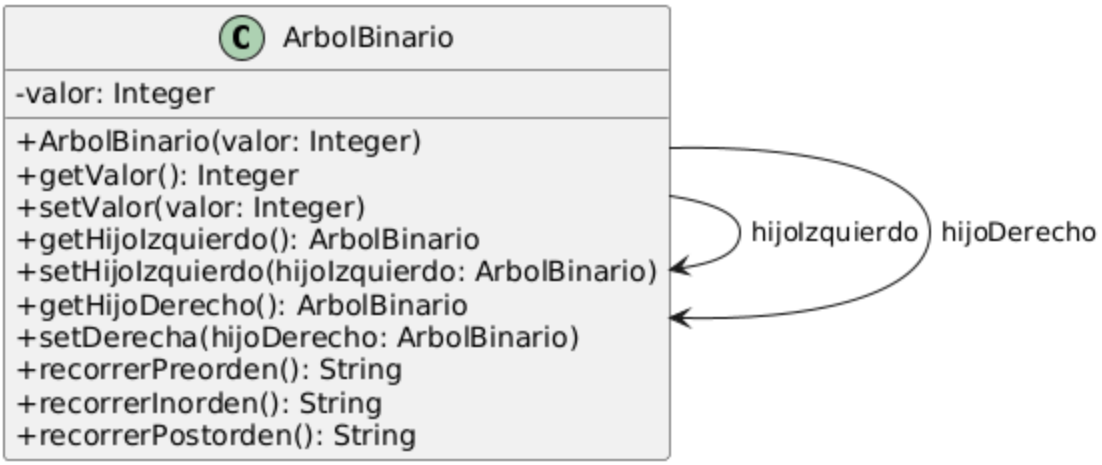
Ejercicio 6




```
package ar.info.unlp.arboles;

public class ArbolBinario {
    private int valor;
    private ArbolBinario hijoIzquierdo;
    private ArbolBinario hijoDerecho;

    public ArbolBinario(int valor) {
        this.valor = valor;
        this.hijoIzquierdo = null;
        this.hijoDerecho = null;
    }

    public int getValor() {
        return valor;
    }

    public void setValor(int valor) {
        this.valor = valor;
```

```
    }

    public ArbolBinario getHijoIzquierdo() {
        return hijoIzquierdo;
    }

    public void setHijoIzquierdo(ArbolBinario hijoIzquierdo) {
        this.hijoIzquierdo = hijoIzquierdo;
    }

    public ArbolBinario getHijoDerecho() {
        return hijoDerecho;
    }

    public void setDerecha(ArbolBinario hijoDerecho) {
        this.hijoDerecho = hijoDerecho;
    }

    public String recorrerPreorden() {
        String resultado = valor + " - ";
        if (this.getHijoIzquierdo() != null) {
            resultado += this.getHijoIzquierdo().recorrerPreorden();
    }
        if (this.getHijoDerecho() != null) {
            resultado += this.getHijoDerecho().recorrerPreorden();
        }
        return resultado;
    }

    public String recorrerInorden() {
        String resultado = "";
        if (this.getHijoIzquierdo() != null) {
            resultado += this.getHijoIzquierdo().recorrerInorden();
    }
        resultado += valor + " - ";
        if (this.getHijoDerecho() != null) {
```

```
            resultado += this.getHijoDerecho().recorrerInorden();
        }
        return resultado;
    }

    public String recorrerPostorden() {
        String resultado = "";
        if (this.getHijoIzquierdo() != null) {
            resultado += this.getHijoIzquierdo().recorrerPostorden();
    }
        if (this.getHijoDerecho() != null) {
            resultado += this.getHijoDerecho().recorrerPostorden();
        }
        resultado += valor + " - ";
        return resultado;
    }

}
```

Paso uno: Análisis

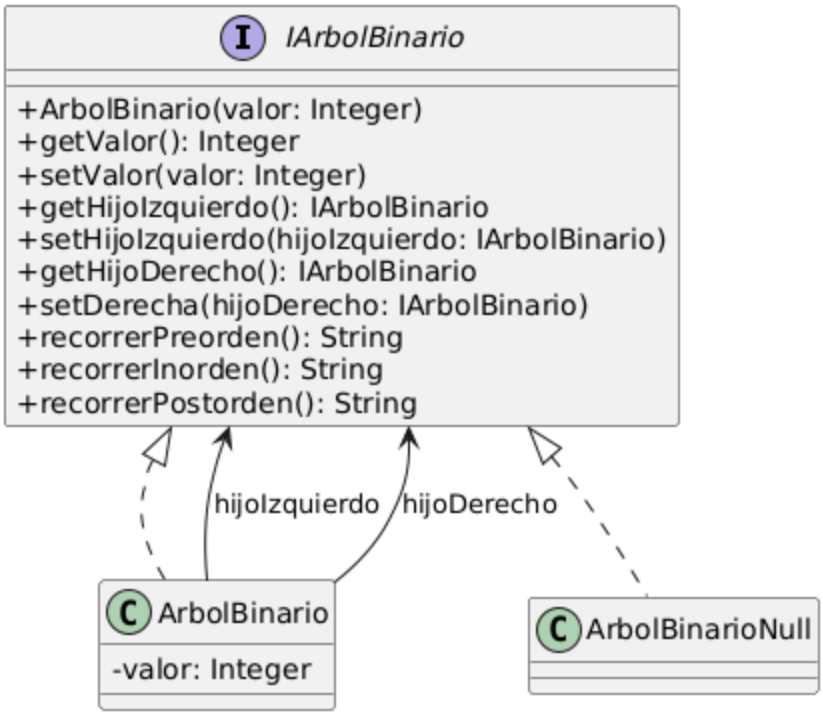
- En los métodos de los recorridos se pregunta primero si los hijos son nulos antes de proceder, ya que las variables **hijoIzquierdo** e **hijoDerecho** se inicializan en null cuando se crea el objeto por si el arbol no tiene hijos. Esto huele mal porque implica comprobar, siempre que se quiera hacer algo con los hijos, que estos existan.

Paso dos: Refactoring

- **Code Smell:** Conditional Complexity
- **Refactoring pattern a aplicar**: Introduce Null Object
    - Crear una interfaz para **ArbolBinario** que tenga las mismas declaraciones de métodos

- Cambiar los métodos que piden y que devuelven un objeto tipo **ArbolBinario** por *IArbolBinario*
- Crear una clase *ArbolBinarioNull* que implemente la interfaz *IArbolBinario*, y que en su implementación de los métodos no haga nada o devuelva un valor basura

Resultado final




```
ARBOLBINARIO.JAVA ///
package unlp.oo2.patrones.ej22;

public class ArbolBinario implements IArbolBinario {
    private int valor;
    private IArbolBinario hijoIzquierdo;
    private IArbolBinario hijoDerecho;

    public ArbolBinario(int valor) {
        this.valor = valor;
```

```
        this.hijoIzquierdo = new ArbolBinarioNull();
        this.hijoDerecho = new ArbolBinarioNull();
    }

    public int getValor() {
        return valor;
    }

    public void setValor(int valor) {
        this.valor = valor;
    }

    public IArbolBinario getHijoIzquierdo() {
        return hijoIzquierdo;
    }

    public void setHijoIzquierdo(IArbolBinario hijoIzquierdo) {
        this.hijoIzquierdo = hijoIzquierdo;
    }

    public IArbolBinario getHijoDerecho() {
        return hijoDerecho;
    }

    public void setDerecha(IArbolBinario hijoDerecho) {
        this.hijoDerecho = hijoDerecho;
    }

    public String recorrerPreorden() {
        String resultado = valor + " - ";
        resultado += this.getHijoIzquierdo().recorrerPreorden();
        resultado += this.getHijoDerecho().recorrerPreorden();
        return resultado;
    }

    public String recorrerInorden() {
```

```
        String resultado = "";
        resultado += this.getHijoIzquierdo().recorrerInorden();
        resultado += valor + " - ";
        resultado += this.getHijoDerecho().recorrerInorden();
        return resultado;
    }

   public String recorrerPostorden() {
        String resultado = "";
        resultado += this.getHijoIzquierdo().recorrerPostorden();
        resultado += this.getHijoDerecho().recorrerPostorden();
        resultado += valor + " - ";
        return resultado;
    }

}



ARBOLBINARIONULL.JAVA ///
package unlp.oo2.patrones.ej22;

public class ArbolBinarioNull implements IArbolBinario {
    //
    @Override
    public int getValor() {
        return 0;
    }

    @Override
    public void setValor(int valor) {
    }

    @Override
    public IArbolBinario getHijoIzquierdo() {
        return new ArbolBinario(0);
```

```
    }

    @Override
    public void setHijoIzquierdo(IArbolBinario hijoIzquierdo) {
    }

    @Override
    public IArbolBinario getHijoDerecho() {
        return new ArbolBinario(0);
    }

    @Override
    public void setDerecha(IArbolBinario hijoDerecho) {
    }

    @Override
    public String recorrerPreorden() {
        return "";
    }

    @Override
    public String recorrerInorden() {
        return "";
    }

    @Override
    public String recorrerPostorden() {
        return "";
    }
}

IARBOLBINARIO.JAVA ///
package unlp.oo2.patrones.ej22;

public interface IArbolBinario {
    public int getValor();
```

```
    public void setValor(int valor);
    public IArbolBinario getHijoIzquierdo();
    public void setHijoIzquierdo(IArbolBinario hijoIzquierdo);
    public IArbolBinario getHijoDerecho();
    public void setDerecha(IArbolBinario hijoDerecho);
    public String recorrerPreorden();
    public String recorrerInorden();
    public String recorrerPostorden();
}
```