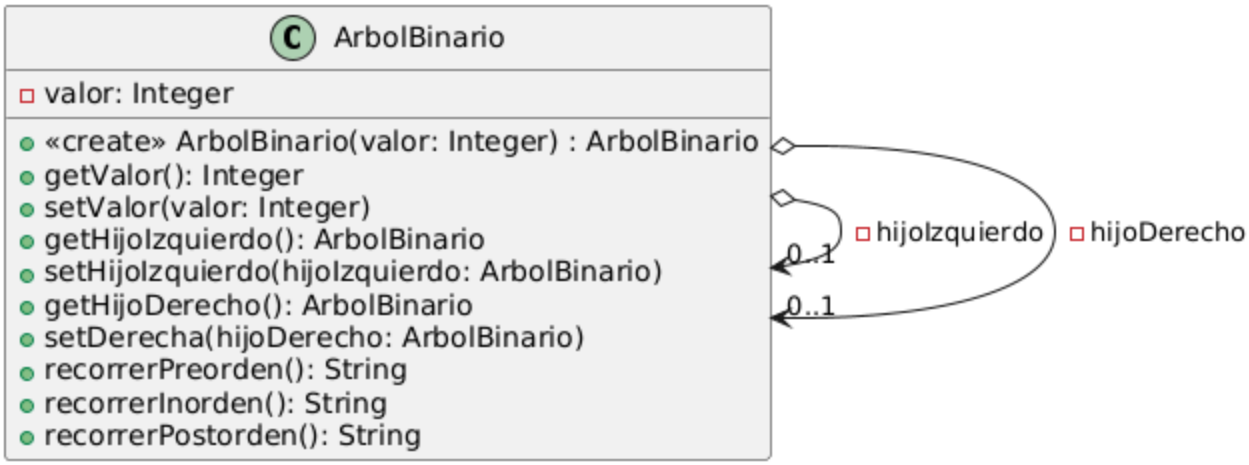
Ejercicio 6

UML Inicial



Codigo que nos dan:

```
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;
}



}
```

Code smell: Null Check
En varios metodos, se esta preguntando por posibles valores nulos

Refactoring: Introduce Null Object
Empezamos a aplicar el patron null object

Paso 1: Creamos la interfaz IArbolBinariom hacemos un pull up method para que las clases que implementen la interfaz compartan la firma de metodos.

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

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


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

// queda todo igual despues....
```

```
......
```

Paso 2: Creamos el null object aplicando Extract subclass sobre la clase que se quiere evitar el chequeo de null, y hacemos que la clase nullObject implemente la interfaz.

```
public class NullArbolBinario implements IArbolBinario{

    @Override
    public int getValor() {
    return 0;
    }

    @Override
    public void setValor(int valor) {
    }

    @Override
    public void setHijoIzquierdo(ArbolBinario hijoIzquierdo) {
    }


    @Override
    public void setDerecha(ArbolBinario hijoDerecho) {
    }

    @Override
    public IArbolBinario getHijoIzquierdo() {
        return this;
    }

    @Override
    public IArbolBinario getHijoDerecho() {
        return this;
    }

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

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

```
    }

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

}
```

Paso 3: En las clases concretas y la interfaz, cambiamos las referencias, ya no deberian referenciar mas a ArbolBinario, sino a IArbolBinario.

Antes:

```
// Interfaz:
public interface IArbolBinario {
    public int getValor();
    public void setValor(int valor);
    public void setHijoIzquierdo(ArbolBinario hijoIzquierdo);
    public void setDerecha(ArbolBinario hijoDerecho);
    public ArbolBinario getHijoIzquierdo(); // cambiar a IArbolBinario
    public ArbolBinario getHijoDerecho();   // cambiar a IArbolBinario
    public String recorrerPreorden();
    public String recorrerInorden();
    public String recorrerPostorden();
}

// Clase concreta:
public class ArbolBinario implements IArbolBinario {
private int valor;
private ArbolBinario hijoIzquierdo; // cambiar a IArbolBinario
private ArbolBinario hijoDerecho;  // cambiar a IArbolBinario

.....

    // cambiar retorno a IArbolBinario
    public ArbolBinario getHijoIzquierdo() {
        return hijoIzquierdo;
    }

    //cambiar retorno a IArbolBinario
    public ArbolBinario getHijoDerecho() {
        return hijoDerecho;
    }
```

```
// Clase null object
public class NullArbolBinario implements IArbolBinario{
.....
    // cambiar retorno a IArbolBinario
    @Override
    public ArbolBinario getHijoIzquierdo() {
        return this;
    }

    // cambiar retorno a IArbolBinario
    @Override
    public ArbolBinario getHijoDerecho() {
        return this;
    }
```

Despues:

```
// Interfaz:
public interface IArbolBinario {
    public int getValor();
    public void setValor(int valor);
    public void setHijoIzquierdo(ArbolBinario hijoIzquierdo);
    public void setDerecha(ArbolBinario hijoDerecho);
    public IArbolBinario getHijoIzquierdo(); // cambiamos a IArbolBinario
    public IArbolBinario getHijoDerecho();   // cambiamos a IArbolBinario
    public String recorrerPreorden();
    public String recorrerInorden();
    public String recorrerPostorden();
}

// Clase concreta:
public class ArbolBinario implements IArbolBinario {
private int valor;
private IArbolBinario hijoIzquierdo; // cambiamos a IArbolBinario
private IArbolBinario hijoDerecho;   // cambiamos a IArbolBinario

.....

// cambiamos retorno a IArbolBinario
public IArbolBinario getHijoIzquierdo() {
    return hijoIzquierdo;
```

```
}
// cambiamos retorno a IArbolBinario
public IArbolBinario getHijoDerecho() {
    return hijoDerecho;
}

// Clase null object
public class NullArbolBinario implements IArbolBinario{
.....
    // cambiamos retorno a IArbolBinario
    @Override
    public IArbolBinario getHijoIzquierdo() {
        return this;
    }

    // cambiamos retorno a IArbolBinario
    @Override
    public IArbolBinario getHijoDerecho() {
        return this;
    }
```

Paso 4: En la clase ArbolBinario ,reemplazamos los valores nulos del constructor por NullArbolBinario

Antes:

```
public class ArbolBinario implements IArbolBinario {

....

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

     ...........
```

Despues:

```
public class ArbolBinario implements IArbolBinario {
public ArbolBinario(int valor) {
    this.valor = valor;
    this.hijoIzquierdo = new NullArbolBinario();
```

```
    this.hijoDerecho = new NullArbolBinario();
}
    ...........
```

Paso 5: Encontrar un chequeo por null y eliminarlo.

Antes:

```
public class ArbolBinario implements IArbolBinario {

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

 .....
```

Despues:

```
public class ArbolBinario implements IArbolBinario {
   . . . . . .
public String recorrerPreorden() {
String resultado = valor + " - ";
    resultado += this.getHijoIzquierdo().recorrerPreorden();
    resultado += this.getHijoDerecho().recorrerPreorden();
return resultado;
}

 .....
```

Paso 6: Repetir paso 5 hasta eliminar todos los chequeos por null

Luego de eliminar todos los null check

```
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;
}
```

Solucion final:

Clase ArbolBinario

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

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

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


public int getValor() {
return valor;
}
```

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

public IArbolBinario getHijoIzquierdo() {
return hijoIzquierdo;
}

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

public IArbolBinario getHijoDerecho() {
return hijoDerecho;
}

public void setDerecha(ArbolBinario 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;
}
}
```

Interfaz IArbolBinario

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

Clase NullArbolBinario

```
public class NullArbolBinario implements IArbolBinario{

@Override
public int getValor() {
return 0;
}

@Override
public void setValor(int valor) {
}

@Override
public void setHijoIzquierdo(ArbolBinario hijoIzquierdo) {
}

@Override
public void setDerecha(ArbolBinario hijoDerecho) {
}

@Override
public IArbolBinario getHijoIzquierdo() {
return this;
}

@Override
public IArbolBinario getHijoDerecho() {
return this;
}
```

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

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

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

}
```