Esquema Mental del Patrón Builder

1. **Builder (Constructor):** Define los métodos para construir las diferentes partes del producto.

2. **ConcreteBuilder (Constructor Concreto):** Implementa los pasos definidos por Builder para crear un producto específico.

3. **Director (Director):** Se encarga de llamar a los métodos de Builder en un orden específico para construir el producto.

4. **Product (Producto):** El objeto final construido, compuesto por varias partes.

Ejemplo Completo: Construcción de una Residencia Lego

```pseudo

// Paso 1: Definir la interfaz Builder

interface ResidenciaBuilder {

    construirCocina()

    construirDormitorio()

    construirComedor()

    construirBaño()

    construirJardín()

    construirPiscina()

    construirGaraje()

    // Otros métodos para construir partes adicionales...

    obtenerResidencia(): Residencia

}

// Paso 2: Crear un ConcreteBuilder

class ResidenciaLegoBuilder implementa ResidenciaBuilder {

    residencia = nueva Residencia()

    construirCocina() {

        residencia.añadir('Bloque Cocina')

    }

    // Implementaciones similares para los otros métodos...

    obtenerResidencia(): Residencia {

        return residencia

    }

}

// Paso 3: Definir el Director

class ConstructorResidencia {

    construirResidencia(builder: ResidenciaBuilder): Residencia {

        builder.construirCocina()

        builder.construirDormitorio()

        builder.construirComedor()

        builder.construirBaño()

        builder.construirJardín()

        builder.construirPiscina()

        builder.construirGaraje()

        // Llamadas a otros métodos...

        return builder.obtenerResidencia()

    }

}

// Paso 4: Definir el Producto

class Residencia {

    bloques = []

    añadir(bloque) {

        bloques.añadir(bloque)

    }

}

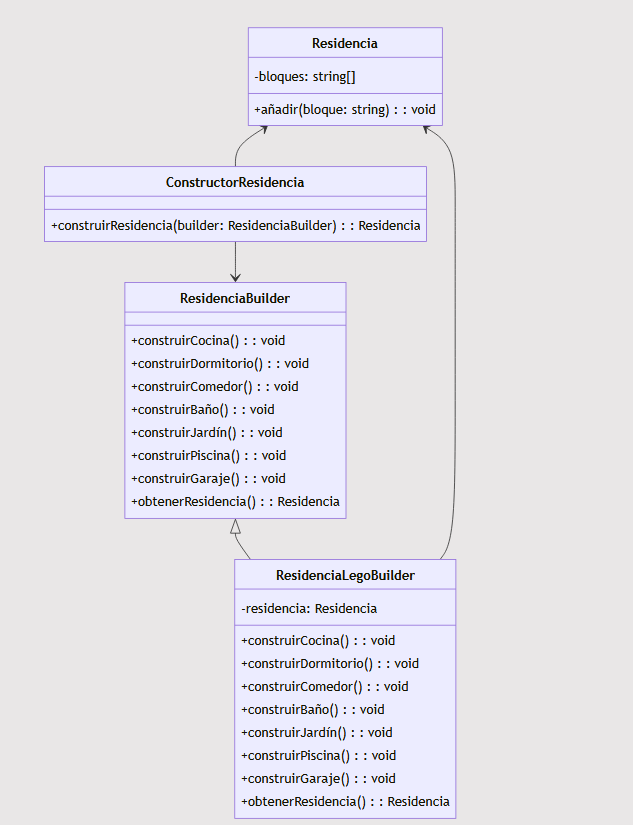
// Paso 5: Utilizar el patrón Builder

constructor = nuevo ConstructorResidencia()

builder = nuevo ResidenciaLegoBuilder()

miResidencia = constructor.construirResidencia(builder)

```



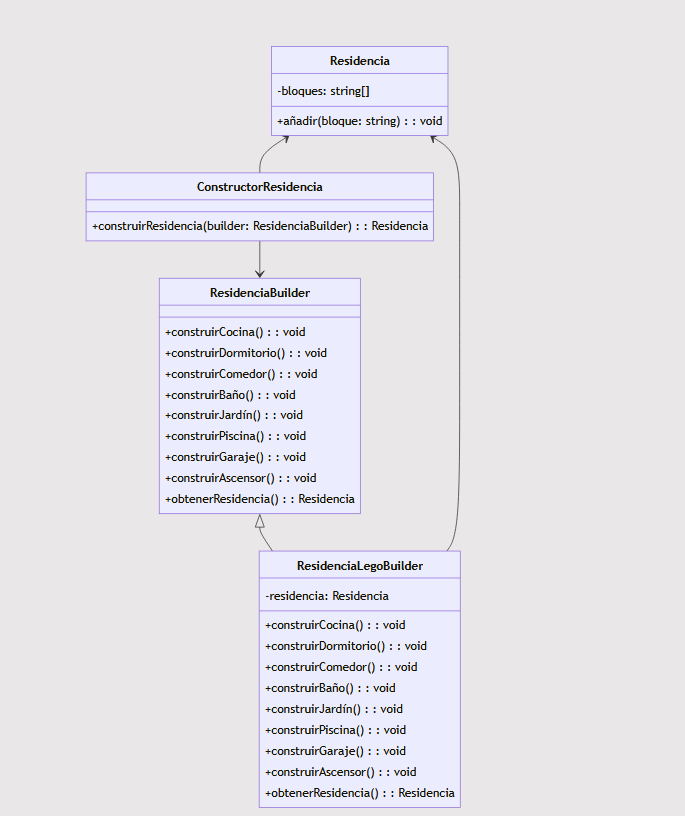
### Actividad: Añadir un Bloque Ascensor

\*\*Instrucciones:\*\*

1. Añade la funcionalidad para construir un 'Bloque Ascensor' en la interfaz `ResidenciaBuilder`.

2. Implementa el método correspondiente en `ResidenciaLegoBuilder`.

3. Modifica el `ConstructorResidencia` para incluir la construcción del ascensor en el proceso.



\*\*Solución:\*\*

```pseudo

// Paso 1: Modificar la interfaz Builder

interface ResidenciaBuilder {

    // ...otros métodos...

    construirAscensor()

    // ...otros métodos...

}

// Paso 2: Modificar el ConcreteBuilder

class ResidenciaLegoBuilder implementa ResidenciaBuilder {

    // ...otros métodos...

    construirAscensor() {

        residencia.añadir('Bloque Ascensor')

    }

    // ...otros métodos...

}

// Paso 3: Modificar el Director

class ConstructorResidencia {

    construirResidencia(builder: ResidenciaBuilder): Residencia {

        // ...llamadas a otros métodos...

        builder.construirAscensor()

        // ...llamadas a otros métodos...

        return builder.obtenerResidencia()

    }

}

```

Con esta práctica, debería ser capaces de entender cómo el patrón Builder les permite añadir nuevas partes a un producto complejo de manera estructurada y mantenible.

classDiagram

class Algo{

-esto: String

-otro: String

-aquello: int

}

class Algo.Builder{

-esto: String

-otro: String

-aquello: int

+setEsto(esto: String): Builder

+setOtro(otro: String): Builder

+setAquello(aquello: int): Builder

+build(): Algo

}

class Main{

+main(args: String[]): void

}

Algo \*-- Algo.Builder: has

Main -- Algo: creates

En este diagrama, la clase Algo tiene tres atributos: esto, otro y aquello. También tiene una clase interna Builder que tiene los mismos atributos y métodos para establecer estos atributos. El método build() en la clase Builder crea una instancia de Algo.

Además, hay una clase Main que tiene un método main(). Este método crea una instancia de Algo usando el patrón de diseño Builder.