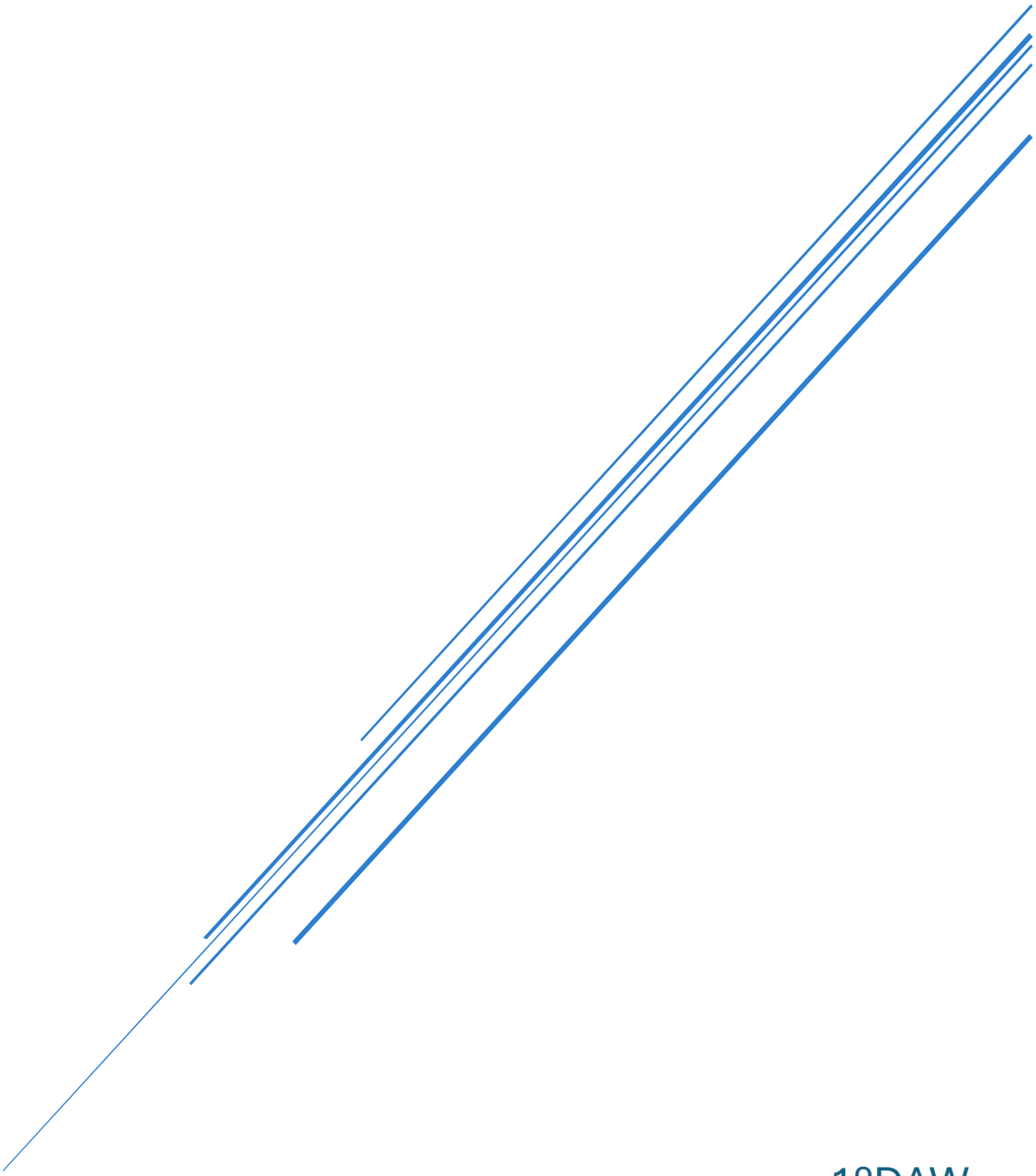
ACTIVIDAD PATRON BUILDER

Patrones de diseño Entornos de desarrollo



1ºDAW

Andreu Orenga Ramon

CONTENIDO

PATRON BUILDER

EJEMPLO DE IMPLEMENTACIÓN DEL PATRON BUILDER

PASO 1: DEFINIR LA INTERFAZ AUTOMOVILBUILDER

Primero, definiremos una interfaz AutomovilBuilder que especifica los métodos para construir las diferentes partes del automóvil.

Actividades_PD > Actividad_PatronBuilder > src > AutomovilBuilder.java

```java
interface AutomovilBuilder {
    void buildMotor();
    void buildAsientos(int numeroAsientos);
    void buildColor(String color);
    Automovil obtenerAutomovil();
}
```

PASO 2: CREAR UN CONCRETEBUILDER

Ahora, crearemos una clase concreta que implementa AutomovilBuilder. Esta clase será responsable de construir un tipo específico de automóvil, por ejemplo, un Sedán.

Actividades_PD > Actividad_PatronBuilder > src > SedanBuilder.java > ...

```java
class SedanBuilder implements AutomovilBuilder {
    private Automovil automovil;

    public SedanBuilder() {
        this.automovil = new Automovil();
    }

    @Override
    public void buildMotor() {
        automovil.setMotor(motor:"Motor de 4 cilindros");
    }

    @Override
    public void buildAsientos(int numeroAsientos) {
        automovil.setAsientos(numeroAsientos);
    }

    @Override
    public void buildColor(String color) {
        automovil.setColor(color);
    }

    @Override
    public Automovil obtenerAutomovil() {
        return automovil;
    }
}
```

PASO 3: DEFINIR EL DIRECTOR

El Director se encarga de construir un automóvil usando el builder proporcionado, asegurándose de que los pasos se ejecuten en orden.

PASO 4: DEFINIR EL PRODUCTO (AUTOMOVIL)

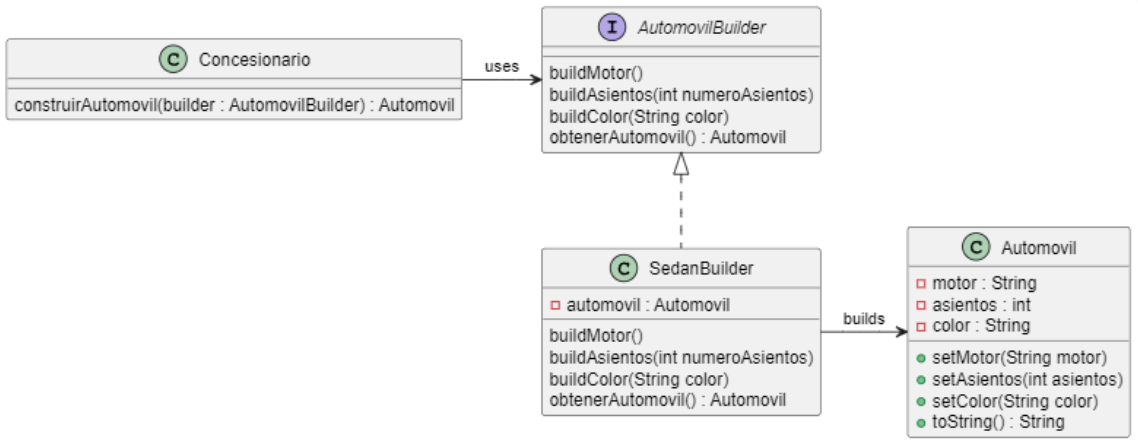
La clase Automovil representa el producto final que está siendo construido.

PASO 5: USAR EL PATRÓN BUILDER PARA CONSTRUIR EL AUTOMÓVIL

Finalmente, utilizamos nuestro patrón Builder en la aplicación principal para crear un automóvil.

```
Actividades_PD > Actividad_PatronBuilder > src > Main.java > ...
1   public class Main {
2       public static void main(String[] args) {
3           Concesionario concesionario = new Concesionario();
4           AutomovilBuilder builder = new SedanBuilder();
5           Automovil miAutomovil = concesionario.construirAutomovil(builder);
6           System.out.println(miAutomovil);
7       }
8   }
9
10
11
```

DIAGRAMA DEL PATRON BUILDER DE AUTOMOVIL

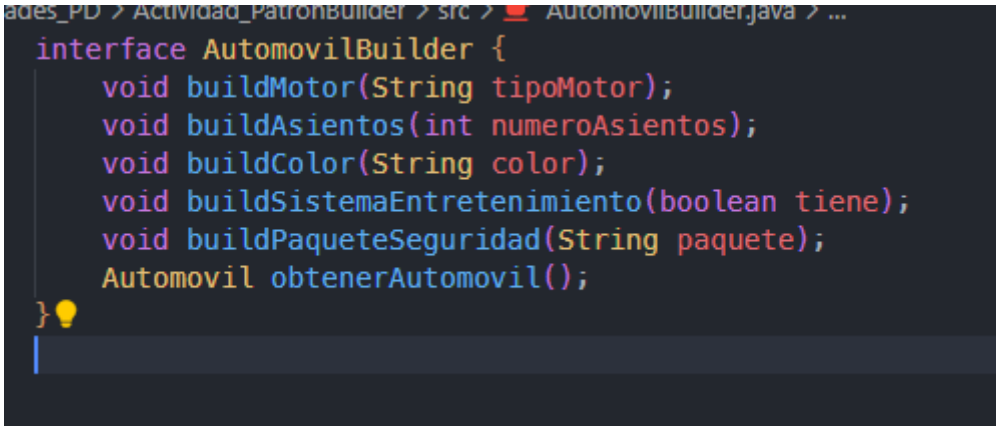


EXPANSIÓN DEL EJEMPLO DEL AUTOMÓVIL

Podemos añadir más funcionalidades al patrón Builder para hacerlo más flexible y poderoso. Por ejemplo, podríamos querer permitir más personalización en los tipos de motor o incluso añadir nuevas características como un sistema de entretenimiento o características de seguridad. Vamos a añadir estas opciones a nuestro patrón Builder.

PASO 6: AÑADIR MÁS CARACTERÍSTICAS AL AUTOMOVILBUILDER

Ampliemos la interfaz AutomovilBuilder para incluir métodos adicionales que permitan más personalización:



PASO 7: IMPLEMENTAR LAS NUEVAS CARACTERÍSTICAS EN SEDANBUILDER

Actualizamos nuestra clase concreta SedanBuilder para manejar estas nuevas opciones:

PASO 8: MODIFICAR CONCESIONARIO PARA USAR LAS NUEVAS OPCIONES

Ajustamos el Director para que pueda construir automóviles utilizando las nuevas características disponibles:

```
class Co  AutomovilBuilder builder =
    publ  Concesionario.construirAutomovil(AutomovilBuilder)
        builder.buildMotor(tipoMotor:"Motor V6");
        builder.buildAsientos(numeroAsientos:5);
        builder.buildColor(color:"Azul Metálico");
        builder.buildSistemaEntretenimiento(tiene:true);
        builder.buildPaqueteSeguridad(paquete:"Avanzado");
        return builder.obtenerAutomovil();
    }
}💡
```

PASO 9: ACTUALIZAR LA CLASE AUTOMOVIL

Asegurémonos de que la clase Automovil puede manejar estas nuevas características:

```
class Automovil {
    private String motor;
    private int asientos;
    private String color;
    private boolean sistemaEntretenimiento;
    private String paqueteSeguridad;

    // Setters
    public void setMotor(String motor) {
        this.motor = motor;
    }

    public void setAsientos(int asientos) {
        this.asientos = asientos;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public void setSistemaEntretenimiento(boolean sistemaEntretenimiento) {
        this.sistemaEntretenimiento = sistemaEntretenimiento;
    }

    public void setPaqueteSeguridad(String paqueteSeguridad) {
        this.paqueteSeguridad = paqueteSeguridad;
    }

    // Getters
    public String getMotor() {
        return motor;
    }

    public int getAsientos() {
        return asientos;
    }

    public String getColor() {
        return color;
    }

    public boolean isSistemaEntretenimiento() {
        return sistemaEntretenimiento;
    }

    public String getPaqueteSeguridad() {
        return paqueteSeguridad;
    }
```

PASO 10: DEMOSTRACIÓN EN MAIN

Finalmente, actualizamos el método main para demostrar las capacidades expandidas:

des_PD > ActividadPatronBuilder > src > Main.java

```java
public class Main {
    public static void main(String[] args) {
        Concesionario concesionario = new Concesionario();
        AutomovilBuilder builder = new SedanBuilder();
        Automovil miAutomovil = concesionario.construirAutomovil(builder);
        System.out.println(miAutomovil);
    }
}
```

DIAGRAMA UML CON LAS MODIFICACIONES

TODO EL CÓDIGO ADJUNTO EN LA CARPETA SRC DEL PROYECTO ACTIVIDAD_PATRONBUILDER