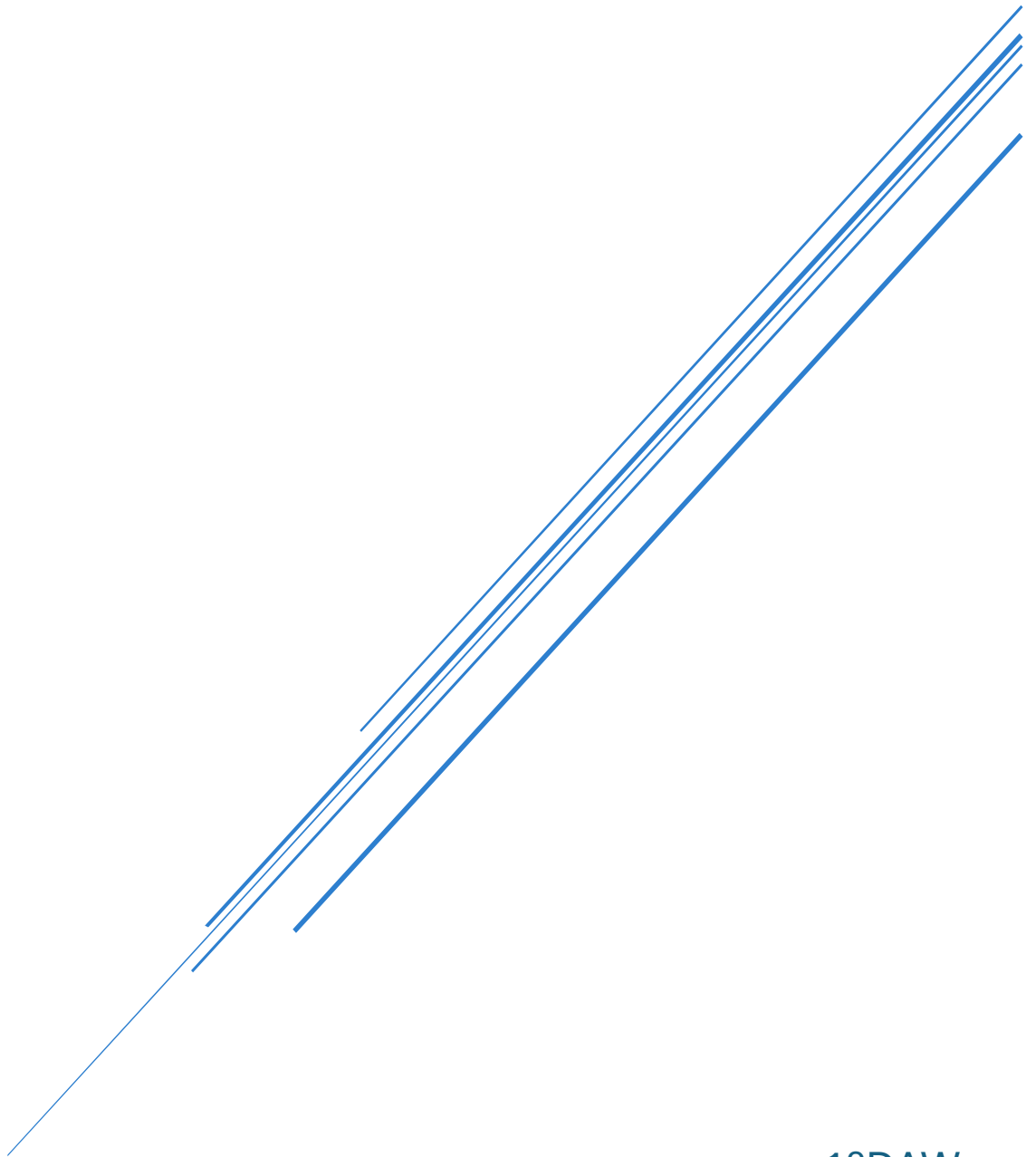


ACTIVIDAD PATRON BUILDER

Patrones de diseño Entornos de desarrollo



1ºDAW

Andreu Orenga Ramon

CONTENIDO

Ejemplo de implementación del Patron Builder	2
Paso 1: Definir la interfaz AutomovilBuilder	2
Paso 2: Crear un ConcreteBuilder	2
Paso 3: Definir el Director	3
Paso 4: Definir el Producto (Automovil)	3
Paso 5: Usar el patrón Builder para construir el automóvil	4
Diagrama del patron builder de Automovil	4
Expansión del Ejemplo del Automóvil	5
Paso 6: Añadir más características al AutomovilBuilder	5
Paso 7: Implementar las nuevas características en SedanBuilder	5
Paso 8: Modificar Concesionario para usar las nuevas opciones	6
Paso 9: Actualizar la clase Automovil	6
Paso 10: Demostración en Main	7
Diagrama UML con las modificaciones	7

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
1  interface AutomovilBuilder {
2      void buildMotor();
3      void buildAsientos(int numeroAsientos);
4      void buildColor(String color);
5      Automovil obtenerAutomovil();
6  }
7
```

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 > ...
1  class SedanBuilder implements AutomovilBuilder {
2      private Automovil automovil;
3
4      public SedanBuilder() {
5          this.automovil = new Automovil();
6      }
7
8      @Override
9      public void buildMotor() {
10         automovil.setMotor("Motor de 4 cilindros");
11     }
12
13     @Override
14     public void buildAsientos(int numeroAsientos) {
15         automovil.setAsientos(numeroAsientos);
16     }
17
18     @Override
19     public void buildColor(String color) {
20         automovil.setColor(color);
21     }
22
23     @Override
24     public Automovil obtenerAutomovil() {
25         return automovil;
26     }
27 }
28
```

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.

```
Actividades_PD > Actividad_PatronBuilder > src > Concesionario.java > ...
1  class Concesionario {
2      public Automovil construirAutomovil(AutomovilBuilder builder) {
3          builder.buildMotor();
4          builder.buildAsientos(5); // Número estándar de asientos para un Sedán
5          builder.buildColor(color:"Rojo");
6          return builder.obtenerAutomovil();
7      }
8  }
9  |
```

PASO 4: DEFINIR EL PRODUCTO (AUTOMOVIL)

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

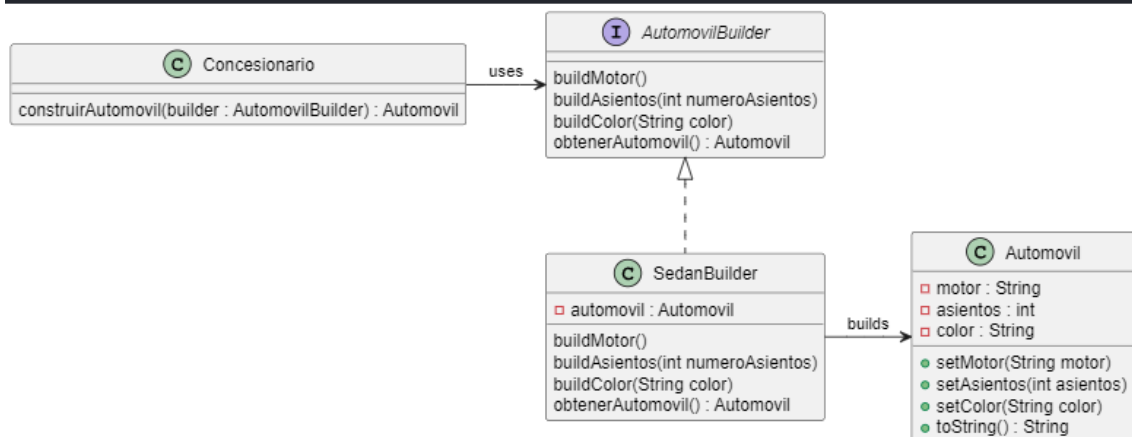
```
Actividades_PD > Actividad_PatronBuilder > src > Automovil.java > ...
1  class Automovil {
2      private String motor;
3      private int asientos;
4      private String color;
5
6      public void setMotor(String motor) {
7          this.motor = motor;
8      }
9
10     public void setAsientos(int asientos) {
11         this.asientos = asientos;
12     }
13
14     public void setColor(String color) {
15         this.color = color;
16     }
17
18     @Override
19     public String toString() {
20         return "Automovil{" +
21             "motor='" + motor + '\'' +
22             ", asientos=" + asientos +
23             ", color='" + color + '\'' +
24             '}';
25     }
26 }
27 |
```

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

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

```
dades_PD > Actividad_PatronBuilder > src > AutomovilBuilder.java > ...  
interface AutomovilBuilder {  
    void buildMotor(String tipoMotor);  
    void buildAsientos(int numeroAsientos);  
    void buildColor(String color);  
    void buildSistemaEntretenimiento(boolean tiene);  
    void buildPaqueteSeguridad(String paquete);  
    Automovil obtenerAutomovil();  
}
```

PASO 7: IMPLEMENTAR LAS NUEVAS CARACTERÍSTICAS EN SEDANBUILDER

Actualizamos nuestra clase concreta SedanBuilder para manejar estas nuevas opciones:

```
dades_PD > Actividad_PatronBuilder > src > SedanBuilder.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 8: MODIFICAR CONCESIONARIO PARA USAR LAS NUEVAS OPCIONES

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

```
class Concesionario {
    public void construirAutomovil(AutomovilBuilder builder) {
        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 > Actividad_PatronBuilder > src > Main.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