Escuela Superior Politécnica del Litoral

Diseño de Software

Taller:

Taller 11-Smells

Integrantes:

Norman Pereira

Jared Martinez

Isaac Bolaños

Fecha

11/06/2025

Tabla de Contenido

1. Sección A: Patrones y Diagrama de Clases

- **Justificación de patrones de diseño**

1) **Creacional:** Factory Method

Permite crear objetos de diferentes tipos de alojamientos (habitaciones, departamentos, casas) de forma flexible y extensible, sin acoplar el código cliente a clases concretas.

2) **Estructural:** Decorator

Permite añadir dinámicamente funcionalidades adicionales, como múltiples canales de notificación o cargos extras, sin modificar las clases base.

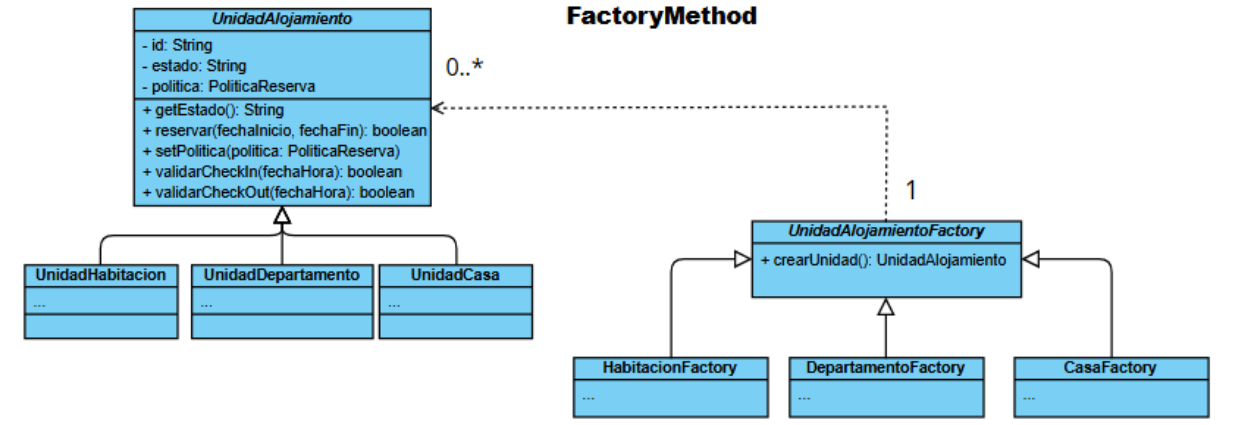
3) **Comportamental:** Strategy

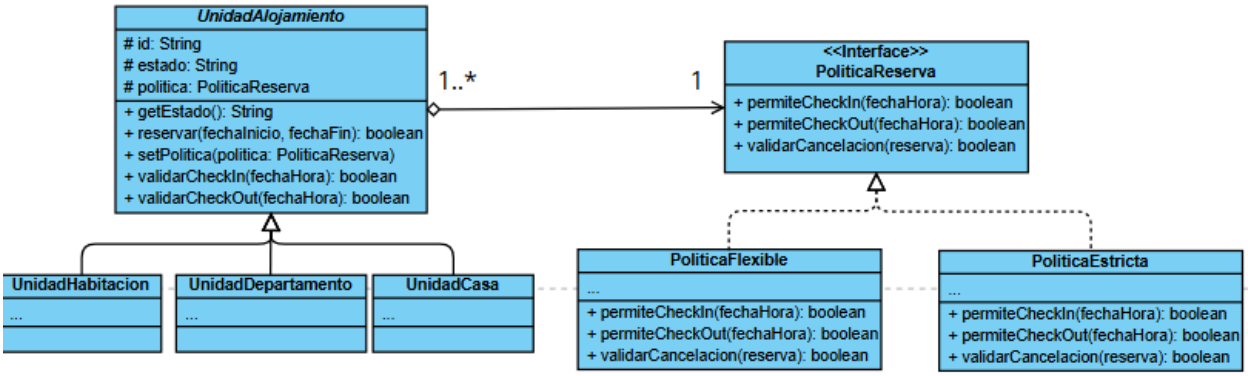
Encapsula las diferentes políticas y reglas específicas de cada anfitrión (check-in, cancelación), facilitando su intercambio y extensión sin afectar la lógica principal.

4) **Comportamental:** Chain of Responsibility

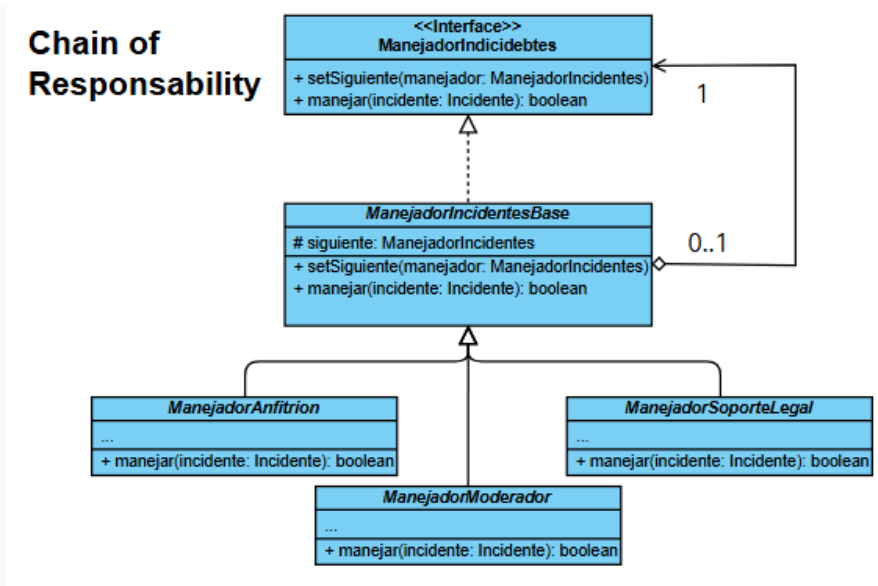
Gestiona el flujo escalonado de atención a incidentes, permitiendo que cada nivel decida si lo resuelve o pasa al siguiente, reduciendo condicionales y acoplamientos.

Diagrama de clases de los patrones aplicados

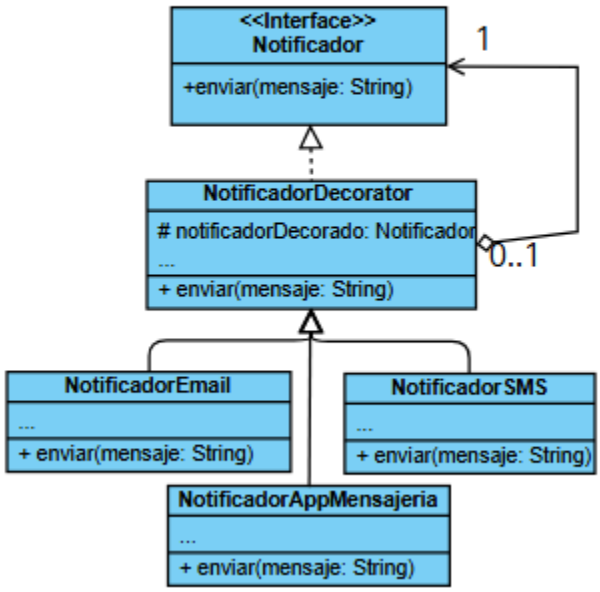




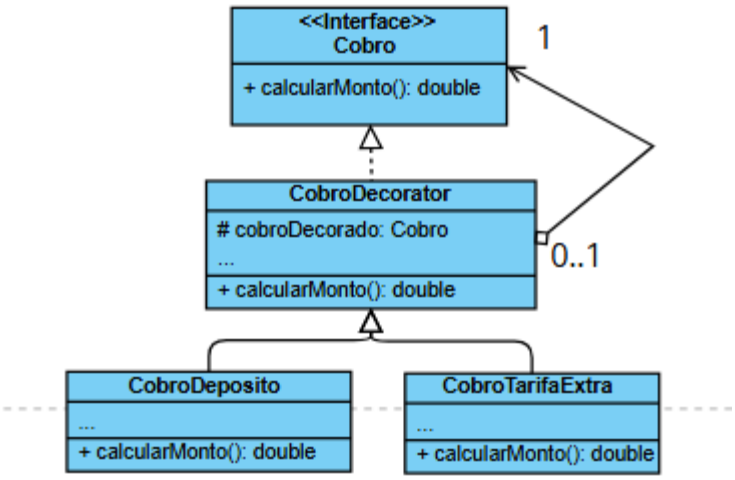
Chain of Responsability



Decorator



Decorator



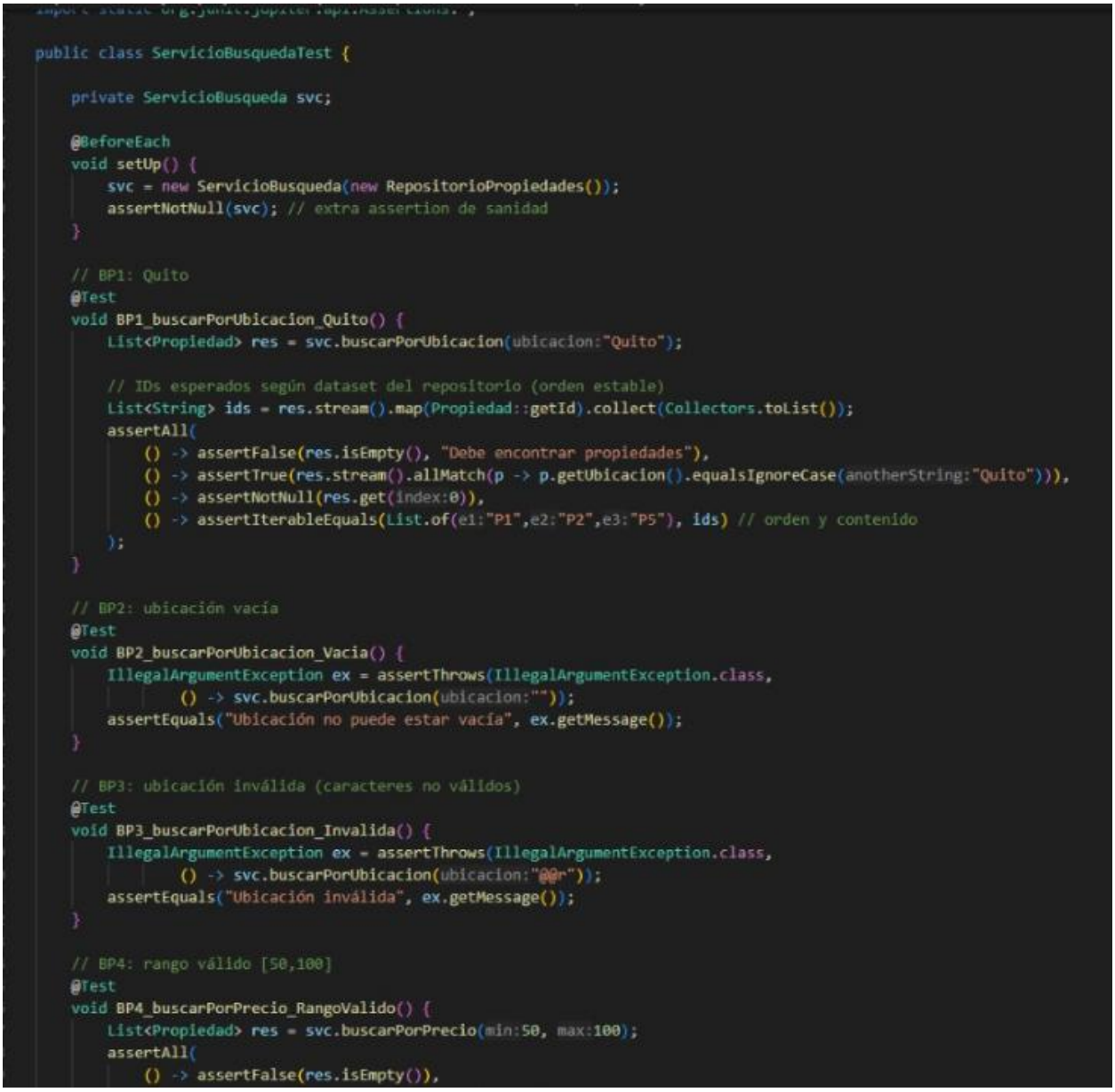
PD: Los puntos suspensivos (...) son usados para denotar que podria ir varios atributos a interes de lo requerido a presente o a futuro para agregar u obtener nuevos comportamientos, por ejemplo, en la clase CobroDecortor, para cobrar por precioPorDia y el atributo DiasDeHospedaje

2. Sección B: Plan de Pruebas

ID	Metodo	Datos de Entrada	Salida Esperada	Proposito
BP1	buscarPorUbicacion	"Quito"	Lista de Propiedades en Quito	Validar la búsqueda por ubicación
BP2	buscarPorUbicacion	""	Error: Ubicación no puede estar vacía	Probar ubicación vacía
BP3	buscarPorUbicacion	"@@r"	Error: Ubicación invalida	Probar caracteres no validos
BP4	buscarPorPrecio	(50 , 100)	Lista de Propiedades con aquel rango	Validar la busqueda por precio
BP5	buscarPorPrecio	(-50,200)	Error: Precio no puede ser negativo	Validar reestricción de precio
BP6	buscarPorPrecio	(600, 200)	Error: Rango invertido	Probar rango invertido
BP7	buscarPorTipo	"Casas"	Lista de Propiedades del tipo ingresado	Validar la búsqueda por tipo
BP8	buscarPorTipo	"Castillo"	Error: Tipo inválido	Probar tipo inexistente
BP9	buscarPorTipo	" "	Error: Tipo no puede estar vacío	Probar tipo vacío

PD: El resto de las pruebas unitarias se encuentra dentro en el archivo de Excel en GitHub

3. Sección C: Implementación y Pruebas Unitarias



```
public class ServicioBusquedaTest {

    private ServicioBusqueda svc;

    @BeforeEach
    void setUp() {
        svc = new ServicioBusqueda(new RepositorioPropiedades());
        assertNotNull(svc); // extra assertion de sanidad
    }

    // BP1: Quito
    @Test
    void BP1_buscarPorUbicacion_Quito() {
        List<Propiedad> res = svc.buscarPorUbicacion(ubicacion:"Quito");

        // IDs esperados según dataset del repositorio (orden estable)
        List<String> ids = res.stream().map(Propiedad::getId).collect(Collectors.toList());
        assertAll(
            () -> assertFalse(res.isEmpty(), "Debe encontrar propiedades"),
            () -> assertTrue(res.stream().allMatch(p -> p.getUbicacion().equalsIgnoreCase(anotherString:"Quito"))),
            () -> assertNotNull(res.get(index:0)),
            () -> assertIterableEquals(List.of(e1:"P1",e2:"P2",e3:"P5"), ids) // orden y contenido
        );
    }

    // BP2: ubicación vacía
    @Test
    void BP2_buscarPorUbicacion_Vacia() {
        IllegalArgumentException ex = assertThrows(IllegalArgumentException.class,
                () -> svc.buscarPorUbicacion(ubicacion:""));
        assertEquals("Ubicación no puede estar vacía", ex.getMessage());
    }

    // BP3: ubicación inválida (caracteres no válidos)
    @Test
    void BP3_buscarPorUbicacion_Invalida() {
        IllegalArgumentException ex = assertThrows(IllegalArgumentException.class,
                () -> svc.buscarPorUbicacion(ubicacion:"@@r"));
        assertEquals("Ubicación inválida", ex.getMessage());
    }

    // BP4: rango válido [50,100]
    @Test
    void BP4_buscarPorPrecio_RangoValido() {
        List<Propiedad> res = svc.buscarPorPrecio(min:50, max:100);
        assertAll(
            () -> assertFalse(res.isEmpty()),
```

PD: El resto del código se encuentra dentro en la carpeta que contiene el proyecto de Java en GitHub

Repositorio de GitHub:

https://github.com/JaredB-Dev/Tarea03---Patrones-y-Pruebas