

**ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL**

**FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y  
COMPUTACIÓN**

**DISEÑO DE SOFTWARE**

**TAREA DE PATRONES DE DISEÑO**

**INTEGRANTES:**

ICAZA BORBOR ARIHUZ ZAMIR

PACHECO CANTOS JOSUE JORDAN

VILLON PALMA MARIA GABRIELA

ROMAN TORRES AXEL SAMUEL

**II PAO 2025**

<b>Sección A: Plan de Pruebas</b>	<b>3</b>
<b>1.1 Clase Hotel.java</b>	<b>3</b>
1.3 Clase ConstructorMensaje.java	5
1.4 Clase Usuario.java	8
1.5 Clase Email.java	9
1.6 Clase PaseoTuristico.java	10
1.7 Clase Hospedaje.java	11
<b>Sección B: Implementación</b>	<b>13</b>
<b>Sección C: Detección de Code Smells</b>	<b>13</b>
1) Duplicate Code	13
2) Temporary Variable	14
3) Feature envy	15
4) Shotgun Surgery	15
5) Primitive Obsession	16
6) Data Class	17
7) Refused Bequest	18
8) Speculative Generality	19
9) Comments	20
10) Dead Code	20
<b>Sección D: Refactorización del Código</b>	<b>21</b>
Extract Method	21
Form Template Method	23
Move Method	25
Replace Data Value with Object	27
Encapsulate Field	29
Speculative Generality	31
Implementación de Método	33
Rename Method	35
Inline Temp	37
Remove Dead Code	38
<b>Link del repositorio</b>	<b>40</b>

## Sección A: Plan de Pruebas

### 1. Sección A: Plan de Pruebas

#### 1.1 Clase Hotel.java

Clase	ID	Método a probar	Precondiciones	Datos de entrada	Salida esperada	Propósito de la prueba
Hotel	HT01	agregar Habitación() ( )	Hotel creado correctamente con nombre y dirección válida.	habitación = null	Se lanza IllegalArgumentException con el mensaje "La habitación no puede ser null"	Comprobar que no se puede agregar null como habitación y que se lanza la excepción correcta
Hotel	HT02	Constructor	Ninguna	nombre= null, dirección= "Calle x"	Se espera que se lance IllegalArgumentException con el mensaje "Los parámetros no pueden ser nulos"	Verificar que no se pueden crear hoteles con nombres nulos.
Hotel	HT03	Constructor	Ninguna	nombre= "Hotel x", direccion= null	Se espera que se lance IllegalArgumentException con el mensaje "Los parámetros	Verificar que no se pueden crear direcciones nulas.

					no pueden ser nulos"	
--	--	--	--	--	----------------------	--

Al aplicar Extract Method, se crearon las clases Validador y ConstructorMensaje.

## 1.2 Clase Validador.java

Clase	ID	Método a probar	Precondiciones	Datos de entrada	Salida esperada	Propósito de la prueba
Validador	VR01	validar Reserva()	Ninguna	usuario=null, mensaje="mensaje ejemplo"	Lanza IllegalArgumentException con mensaje "El usuario y su nombre no pueden ser nulos".	Verificar que se valida usuario nulo

Validador	VO02	validar Observer()	Ninguna	observer!= null	No lanza excepción	Confirmar que observer válido no genera error
-----------	------	--------------------	---------	-----------------	--------------------	---

Validador	VR03	validar Reserva()	usuario != null, usuario.nombre != null	mensaje= null	Lanza IllegalArgumentException con mensaje "El mensaje no puede ser nulo".	Verificar que se valida mensaje nulo
-----------	------	-------------------	---	---------------	--	--------------------------------------

Validador	VR04	validar Reserva ()	usuario != null, usuario.nombre != null	mensaje = "mensaje valido"	No lanza excepción	Confirmar que datos válidos no generan error
Validador	VO01	validar Observer ()	ninguna	observer = null	Lanza IllegalArgumentException con mensaje "Observer no puede ser null"	Verificar que se detecta observer nulo

Validador	VO02	validar Observer ()	ninguna	observer != null	No lanza excepción	Confirmar que observer válido no genera error
-----------	------	---------------------	---------	------------------	--------------------	---

### 1.3 Clase ConstructorMensaje.java

Clase	ID	Método a probar	Precondiciones	Datos de entrada	Salida esperada	Propósito de la prueba
-------	----	-----------------	----------------	------------------	-----------------	------------------------

Constru- ctor Mensaje	CM01	construir Reserva Creada	usuario con nombre "Miriam "	usuario = (id = 1, nombre = "Miriam ", email = "miriam@ example. com") mensaje = "Reserva confirmada para el 05/10"	"Hola Miriam,\n\nReserva confirmada para el 05/10\n\nGracias por elegir TravelStay."	Verificar construcción normal del mensaje de creación de reserva.
Constru ctor Mensaje	CM02	construir Reserva Creada	usuario valido	usuario = (id=2, nombre = "Miriam ", email = "miriam@ example. com") mensaje = ""	"Hola Miriam,\n\n\n\nGracias por elegir TravelStay."	Verificar que funciona con mensaje vacío.
Constru ctor Mensaje	CM03	construir Reserva Creada	usuario con nombre vacío	usuario = (id = 3, nombre = "", email = "usuario3 @example. com") mensaje = "Reserva para el 05/10"	"Hola,\n\nReserva para el 05/10\n\nGracias por elegir TravelStay."	Verificar que el formato se mantiene aunque el nombre esté vacío.

Constru ctor torMensaje	CM04	construir Reserva Cancelada	usuario con nombre "Samuel"	usuario = (id=4, nombre = "Samuel ", email = "samuel@ example. com") mensaje = "Reserva cancelada para el 10/05"	"Hola Samuel,\n\nReserva cancelada para el 10/05\n\nLamentamos que hayas cancelado. Esperamos verte pronto."	Verificar construcci ón normal del mensaje de cancela- ción.
Constru ctor torMensaje	CM05	construir Reserva Cancelad a	usuario válido	usuario = (id=5, nombre = "Samuel ", email = "samuel@ example.com") mensaje = ""	"Hola Samuel,\n\n\n\n\nLamentamos que hayas cancelado. Esperamos verte pronto."	Verificar que funciona con mensaje vacío.

ConstrutorMensaje	CM06	construir Reserva Cancelada	usuario con nombre vacío	usuario = (id=6, nombre= "", email=" usuario6@example.com") mensaje= "Reserva cancelada para el 05/10"	"Hola, \n\nReserva cancelada para el 05/10\nm Lamentamos que hayas cancelado. Esperamos verte pronto."	Verificar que el formato se mantiene, aunque el nombre esté vacío.
-------------------	------	-----------------------------	--------------------------	--	--	--

#### 1.4 Clase Usuario.java

Al cambiar el atributo correo de tipo String a una clase Email, se hicieron los siguientes test:

Clase	ID	Método a probar	Precondiciones	Datos de entrada	Salida esperada	Propósito de la prueba
Usuario	UT-USUARIO-01	getId()	Usuario válido creado	id=1, nombre= "Maria", email="maria@mail.com"	1	Verificar que el id se almacena y retorna correctamente



Usuario	UT-USUARIO-02	getNombre()	Usuario válido creado	id=2, nombre="Juan", email="juan@mail.com"	"Juan"	Verificar que el nombre se almacena y retorna correcto
Usuario	UT-USUARIO-03	getCorreo()	Usuario válido creado	id=3, nombre="Ana", email="ana@mail.com"	Email("ana@mail.com")	Verificar que el correo se almacena y retorna correcto
Usuario	UT-USUARIO-04	Constructor	Ninguna	id=4, nombre="Bad", email="bademail "	IllegalArgumentException	Validar que no se permite crear usuario con email inválido
Usuario	UT-USUARIO-05	Constructor	Ninguna	id=5, nombre="Null", email=null	NullPointerException	Validar que no se permite crear usuario con email nulo

### 1.5 Clase Email.java

Para la refactorización de la clase Usuario, se creó la clase Email.

Clase	ID	Método a probar	Precondiciones	Datos de entrada	Salida esperada	Propósito de la prueba
-------	----	-----------------	----------------	------------------	-----------------	------------------------

Email	UT-EM AIL-01	Constructor/getValor	Ninguna	"test@mail.com"	"test@mail.com"	Verificar creación y obtención de email válido
Email	UT-EM AIL-02	Constructor	Ninguna	"noemail "	IllegalArgumentException	Validar que no se permite email inválido
Email	UT-EM AIL-03	Constructor	Ninguna	null	IllegalArgumentException	Validar que no se permite email nulo
Email	UT-EM AIL-04	equals/hashCode	Ninguna	"a@mail.com", <a href="#">"a@mail.com"</a> , "b@mail.com"	equals true/false, hashCode igual/distinto	Verificar igualdad y hashCode

Email	UT-EM AIL-05	toString	Ninguna	"string@mail.com"	"string@mail.com"	Verificar representación en texto

## 1.6 Clase PaseoTuristico.java

Clase	ID	Método a probar	Precondiciones	Datos de entrada	Salida esperada	Propósito de la prueba
PaseoTuristico	pt1	verificar Disponibilidad	Estado de la instancia : CANCELADO	Ninguna (solo configuración del estado)	false	Verificar que un paseo cancelado no se considere disponible
PaseoTuristico	pt2	calcular Precio	Precio de la instancia : 0	Ninguna	0.0	Comprobar que calcular Precio() retorna correctamente 0
PaseoTuristico	pt3	calcularPrecio	Precio de la instancia: -50.0	Ninguna	-50.0	Verificar manejo de precios negativos

PaseoTuristico	pt4	mostrar Detalles	Instancia con nombre, precio y estado válidos	Ninguna	Imprime nombre, precio y estado sin excepciones	Comprobar que mostrar Detalles() imprime información básica correctamente

### 1.7 Clase Hospedaje.java

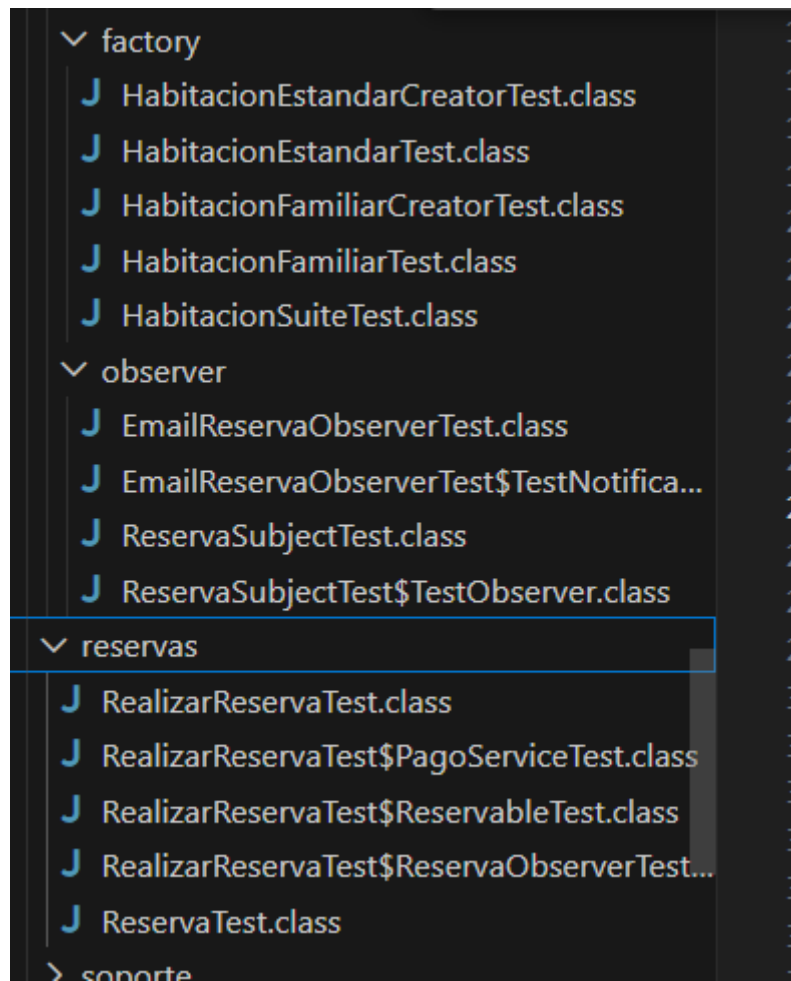
Clase	ID	Método a probar	Precondiciones	Datos de entrada	Salida esperada	Propósito de la prueba
-------	----	-----------------	----------------	------------------	-----------------	------------------------

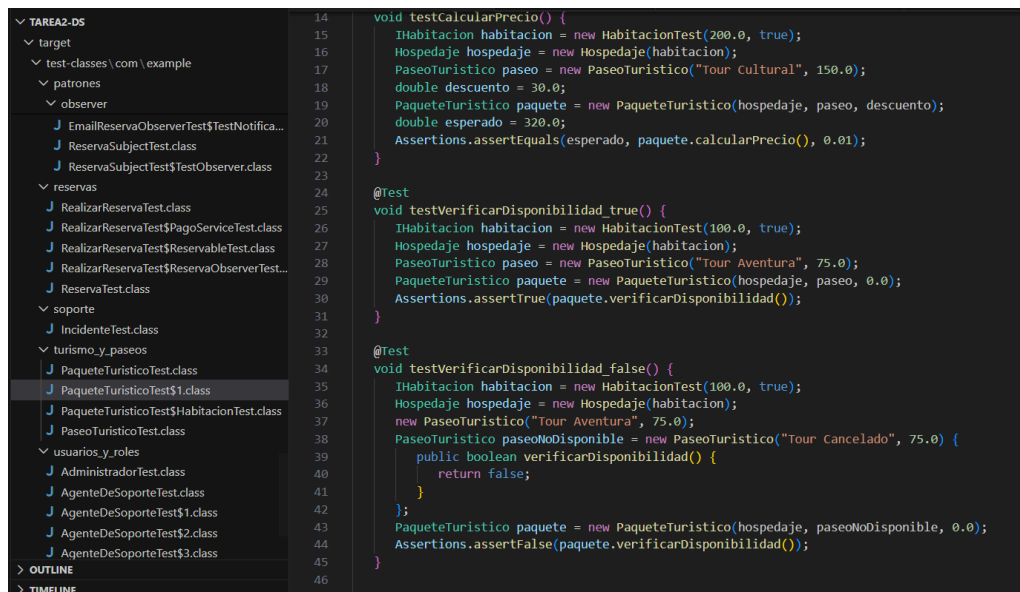
Hospedaje	Hp1	calcular Precio	Instancia con precio=-50.0	Ninguno	Retorna -50.0	Verificar robustez del método ante valores negativos
Hospedaje	hp2	verificar Disponibilidad	Instancia con habitación=null	Ninguno	Lanza NullPointerException o manejo controlado	Comprobar que la clase maneja correctamente la ausencia de habitación
Hospedaje	hp3	calcular Precio	Instancia con habitación=null	Ninguno	Lanza NullPointerException o manejo controlado	Verificar robustez al calcular precio sin habitación

Hospedaje	hp4	mostrar Detalles	Instancia con habitación válida	Ninguno	Imprime detalles de la habitación sin excepción	Confirmar que mostrar Detalles() funciona correctamente y no lanza errores

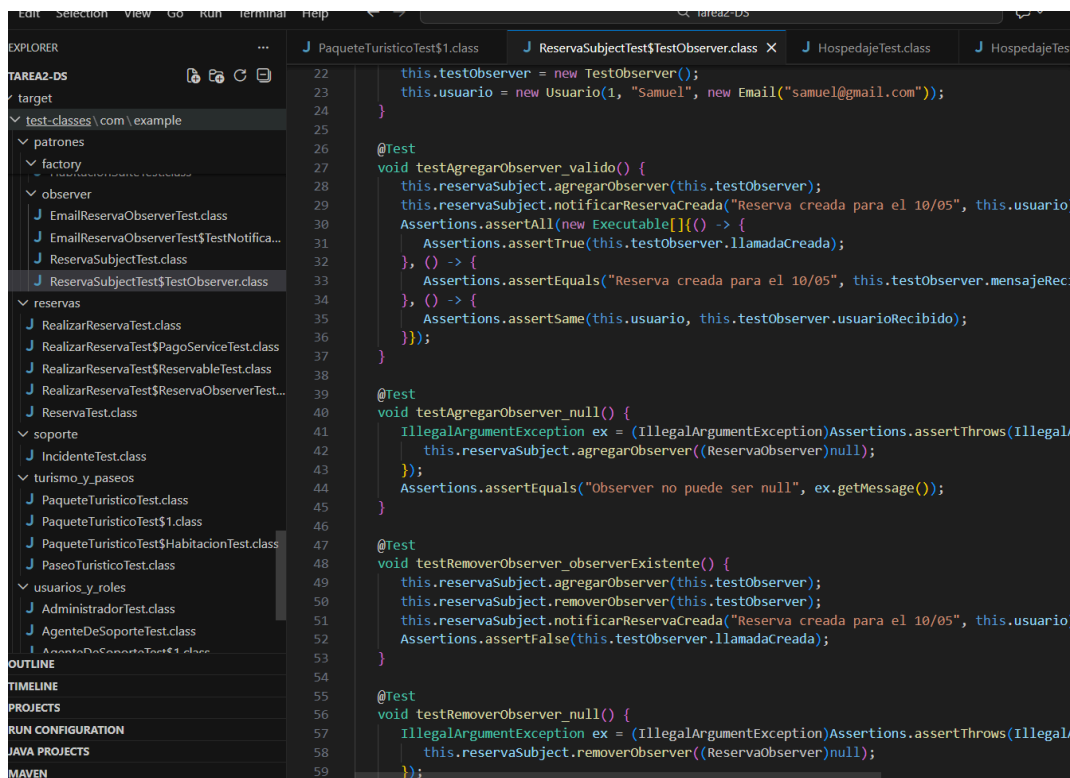
## Sección B: Implementación

Dentro del código se encuentra toda la implementación de las pruebas





```
14 void testCalcularPrecio() {
15     IHabitacion habitacion = new HabitacionTest(200.0, true);
16     Hospedaje hospedaje = new Hospedaje(habitacion);
17     PaseoTuristico paseo = new PaseoTuristico("Tour Cultural", 150.0);
18     double descuento = 30.0;
19     PaqueteTuristico paquete = new PaqueteTuristico(hospedaje, paseo, descuento);
20     double esperado = 320.0;
21     Assertions.assertEquals(esperado, paquete.calcularPrecio(), 0.01);
22 }
23
24 @Test
25 void testVerificarDisponibilidad_true() {
26     IHabitacion habitacion = new HabitacionTest(100.0, true);
27     Hospedaje hospedaje = new Hospedaje(habitacion);
28     PaseoTuristico paseo = new PaseoTuristico("Tour Aventura", 75.0);
29     PaqueteTuristico paquete = new PaqueteTuristico(hospedaje, paseo, 0.0);
30     Assertions.assertTrue(paquete.verificarDisponibilidad());
31 }
32
33 @Test
34 void testVerificarDisponibilidad_false() {
35     IHabitacion habitacion = new HabitacionTest(100.0, true);
36     Hospedaje hospedaje = new Hospedaje(habitacion);
37     new PaseoTuristico("Tour Aventura", 75.0);
38     PaseoTuristico paseoNoDisponible = new PaseoTuristico("Tour Cancelado", 75.0) {
39         public boolean verificarDisponibilidad() {
40             return false;
41         }
42     };
43     PaqueteTuristico paquete = new PaqueteTuristico(hospedaje, paseoNoDisponible, 0.0);
44     Assertions.assertFalse(paquete.verificarDisponibilidad());
45 }
46
```



```
22 this.testObserver = new TestObserver();
23 this.usuario = new Usuario(1, "Samuel", new Email("samuel@gmail.com"));
24 }
25
26 @Test
27 void testAgregarObserver_valido() {
28     this.reservaSubject.agregarObserver(this.testObserver);
29     this.reservaSubject.notificarReservaCreada("Reserva creada para el 10/05", this.usuario);
30     Assertions.assertThat(new Executable[]() -> {
31         Assertions.assertTrue(this.testObserver.llamadaCreada);
32     }, () -> {
33         Assertions.assertEquals("Reserva creada para el 10/05", this.testObserver.mensajeRecibido);
34     }, () -> {
35         Assertions.assertSame(this.usuario, this.testObserver.usuarioRecibido);
36     });
37 }
38
39 @Test
40 void testAgregarObserver_null() {
41     IllegalArgumentException ex = (IllegalArgumentException) Assertions.assertThrows(IllegalArgumentException.class, () -> {
42         this.reservaSubject.agregarObserver((ReservaObserver)null);
43     });
44     Assertions.assertEquals("Observer no puede ser null", ex.getMessage());
45 }
46
47 @Test
48 void testRemoverObserver_observerExistente() {
49     this.reservaSubject.agregarObserver(this.testObserver);
50     this.reservaSubject.removerObserver(this.testObserver);
51     this.reservaSubject.notificarReservaCreada("Reserva creada para el 10/05", this.usuario);
52     Assertions.assertFalse(this.testObserver.llamadaCreada);
53 }
54
55 @Test
56 void testRemoverObserver_null() {
57     IllegalArgumentException ex = (IllegalArgumentException) Assertions.assertThrows(IllegalArgumentException.class, () -> {
58         this.reservaSubject.removerObserver((ReservaObserver)null);
59     });
60 }
```

## Sección C: Detección de Code Smells

### 1) Duplicate Code

**Descripción:** Contienen lógica duplicada: Validaciones de parámetros y construcción del asunto y cuerpo del mensaje. En la clase Reserva Subject los métodos de `notificarReservaCreada` y `notificarReservaCancelada` repiten la validaciones de parámetros usuario y mensaje y la validación del observer.

**Impacto:** La duplicación afecta en la mantenibilidad del código, cada vez que se deba modificar la lógica habrá que actualizar múltiples métodos

**Ubicación exacta:**

- Archivo: src\main\java\com\example\patrones\observer\EmailReservaObserver.java
- Clase: EmailReservaObserver
- Método: onReservaCreada y onReservaCancelada

**Evidencia:**

```
@Override
public void onReservaCreada(String mensaje, Usuario usuario) {
    if (usuario == null || usuario.getNombre() == null) {
        throw new IllegalArgumentException(s: "El usuario y su nombre no pueden ser nulos.");
    }
    if (mensaje == null) {
        throw new IllegalArgumentException(s: "El mensaje no puede ser nulo.");
    }

    String asunto = "Confirmación de reserva TravelStay";
    String cuerpo = String.format(format: "Hola %,n\n%s\n\nGracias por elegir TravelStay",
        usuario.getNombre(),
        mensaje);
    notificador.enviar(asunto + "\n" + cuerpo, usuario);
}

@Override
public void onReservaCancelada(String mensaje, Usuario usuario) {
    if (usuario == null || usuario.getNombre() == null) {
        throw new IllegalArgumentException(s: "El usuario y su nombre no pueden ser nulos.");
    }
    if (mensaje == null) {
        throw new IllegalArgumentException(s: "El mensaje no puede ser nulo.");
    }

    String asunto = "Cancelación de reserva TravelStay";
    String cuerpo = String.format(
        format: "Hola %,n\n%s\n\nLamentamos que hayas cancelado. Esperamos verte pronto",
        usuario.getNombre(),
        mensaje);
    notificador.enviar(asunto + "\n" + cuerpo, usuario);
}
```

**Refactor recomendado:**

Extract Method: mover la lógica repetida a métodos reutilizables.

## 2) Temporary Variable

**Descripción:** Declara una variable temporal precioHabitacion que solo sirve para almacenar un valor intermedio y luego se usa una vez en la expresión de retorno.

**Impacto:** El código es más largo y difícil de seguir. Al existir variables innecesarias aumenta la probabilidad de errores.

**Ubicación exacta :**

- Archivo: src\main\java\com\example\turismo\_y\_paseos\PaqueteTuristico.java
- Clase: PaqueteTuristico
- Método: calcularPrecio (...)

**Evidencia:**

```

@Override
public double calcularPrecio() {
    // Usar el precio decorado de la habitación
    double precioHabitacion = 0.0;
    if (hospedaje.getHabitacion() != null) {
        precioHabitacion = hospedaje.getHabitacion().calcularPrecio();
    }
    return precioHabitacion + paseo.calcularPrecio() - descuento;
}

```

Refactor recomendado: Se recomienda aplicar Inline Temp y devolver el valor de la variable directamente en el return.

### 3) Feature envy

**Descripción:** Se accede directamente a los datos y métodos de otros objetos (hospedaje y paseo) en lugar de delegar la responsabilidad de mostrar sus detalles a las propias clases correspondientes.

**Impacto:** Disminuye la mantenibilidad, ya que los cambios en la forma de mostrar detalles de hospedaje o paseo requieren modificar PaqueteTuristico.

**Ubicación exacta:**

- Archivo: src\main\java\com\example\turismo\_y\_paseos\PaqueteTuristico.java
- Clase: PaqueteTuristico
- Método: mostrarDetalles(...)

**Evidencia:**

```

@Override
public void mostrarDetalles() {
    System.out.println(x: "Paquete turístico:");
    if (hospedaje.getHabitacion() != null) {
        hospedaje.getHabitacion().mostrarDetalles();
    }
    System.out.println("Incluye paseo: " + paseo);
    System.out.println("Descuento aplicado: $" + descuento);
}

```

Refactor recomendado: Usar Move Method para trasladar la lógica de mostrar detalles a las clases Hospedaje y PaseoTuristico.

### 4) Shotgun Surgery

**Descripción:** Si mañana se quisiera cambiar la forma de mostrar detalles se tendría que modificar todas las subclases



**Impacto:** Afecta a la mantenibilidad del código. Cada vez que se cambia el formato, hay que recordar actualizar todas las subclases. Así como la legibilidad, tener código duplicado en varias clases hace que sea más difícil de entender la lógica.

**Ubicación exacta:**

- Archivo: src\main\java\com\example\patrones\factory\Habitacion.java
- Clase: HabitacionSuite, HabitacionFamiliar, HabitacionEstandar (subclases de Habitacion)
- Método: mostrarDetaller()

**Evidencia:**

```
public class HabitacionSuite extends Habitacion {
    public HabitacionSuite(int idHotel, int numero) {
        super(idHotel, numero);
    }

    @Override
    public void mostrarDetalles() {
        System.out.println("Habitación SUITE Nº " + numero + " - Hotel ID: " + idHotel + " - ");
    }
}
```

```
public class HabitacionFamiliar extends Habitacion {
    public HabitacionFamiliar(int idHotel, int numero) {
        super(idHotel, numero);
    }

    @Override
    public void mostrarDetalles() {
        System.out.println("Habitación FAMILIAR Nº " + numero + " - Hotel ID: " + idHotel + " - ");
    }
}
```

```
public class HabitacionEstandar extends Habitacion {
    public HabitacionEstandar(int idHotel, int numero) {
        super(idHotel, numero);
    }

    @Override
    public void mostrarDetalles() {
        System.out.println("Habitación ESTÁNDAR Nº " + numero + " - Hotel ID: " + idHotel + " - ");
    }
}
```

**Refactor recomendado:** Form Template Method para definir la estructura del método mostrarDetalles() en la superclase.

## 5) Primitive Obsession

**Descripción:** El atributo correo en la clase Usuario es de tipo String, lo que permite asignar cualquier valor sin validación ni restricciones. Esto puede llevar a errores, datos inconsistentes y lógica de validación dispersa en el código.

**Impacto:** Mayor probabilidad de errores por valores inválidos y dificultad para centralizar y reutilizar la lógica de validación de correos.

**Ubicación exacta:**

- Archivo: src\main\java\com\example\usuarios\_y\_rols\Usuario.java
- Clase: Usuario
- Atributo: correo

**Evidencia:**

```
package com.example.usuarios_y_rols;

public class Usuario {
    private int id;
    private String nombre;
    private String correo;
}
```

**Refactor recomendado:** Usar Replace Data Value with Object (Reemplazar valor de dato con objeto). Esto implica crear una clase específica Email para encapsular el campo correo, junto con su lógica de validación y comportamiento asociado, en lugar de usar un String simple.

## 6) Data Class

**Descripción:** La principal función de esta clase es almacenar datos (nombre, estado, precio). Sus métodos (calcularPrecio, verificarDisponibilidad) solo retornan valores de sus campos, sin lógica compleja ni comportamiento propio relevante.

**Impacto:** Dificulta la evolución del sistema, ya que la lógica relacionada con el paseo turístico puede dispersarse en otras clases. Aumenta el acoplamiento y reduce la cohesión, pues otras clases deben manipular directamente los datos.

**Ubicación exacta:**

- Archivo: src\main\java\com\example\turismo\_y\_paseos\PaseoTuristico.java
- Clase: PaseoTuristico
- Método:

**Evidencia:**

```

import com.example.interfaces.Reservable;

public class PaseoTuristico implements Reservable {
    private String nombre;
    private EstadoPaseo estado; // disponible, agotado, cancelado
    private double precio;

    public PaseoTuristico(String nombre, double precio) {
        this.nombre = nombre;
        this.precio = precio;
        this.estado = EstadoPaseo.DISPONIBLE;
    }

    public double calcularPrecio() {
        return precio;
    }

    public boolean verificarDisponibilidad() {
        if(estado.equals(EstadoPaseo.DISPONIBLE) ) return true;
        else{return false;}
    }
}

```

**Refactor recomendado:** Simplify Conditional Expression Se reemplaza el if redundante en verificarDisponibilidad() por una sola expresión booleana, también se aplica Rename field en precio se renombra a preciobase para aclarar que es el valor inicial antes de aplicar cualquier regla de negocio. Además, es importante el uso de Encapsulate Field puesto que se agregan getters para acceder a los atributos de forma segura, evitando exposición directa.

## 7) Refused Bequest

**Descripción:** Las clases HabitaciónFamiliar y HabitaciónSuite heredan de la clase Habitación, pero sobrescriben el método calcularPrecio() únicamente para lanzar una excepción (UnsupportedOperationException). Esto indica que estas subclasses no pueden implementar correctamente toda la funcionalidad heredada.

**Impacto:**

- Rompe el principio de sustitución de Liskov: las subclasses no pueden ser usadas en lugar de la superclase sin errores.
- Puede causar errores en el tiempo de ejecución si se invoca el método no implementado.
- Indica una jerarquía de herencia inapropiada, lo que dificulta el mantenimiento y la extensión del sistema.

**Ubicación exacta:**

- Archivo:  
src\main\java\com\example\patrones\factory\HabitacionFamiliar.java  
src\main\java\com\example\patrones\factory\HabitacionSuite.java
- Clase: HabitaciónFamiliar y HabitaciónSuite
- Método: la herencia

**Evidencia:**

```

public class HabitacionFamiliar extends Habitacion {
    public HabitacionFamiliar(int idHotel, int numero) {
        super(idHotel, numero);
    }

    @Override
    public void mostrarDetalles() {
        System.out.println("Habitación FAMILIAR Nº " + numero + " - Hotel ID: " + idHotel + " - ");
    }

    @Override
    public double calcularPrecio() {
        throw new UnsupportedOperationException(message: "Unimplemented method 'calcularPrecio()'");
    }
}

```

**java.lang.UnsupportedOperationException**

public UnsupportedOperationException(String message)

Constructs an UnsupportedOperationException with the specified detail message.

**Parameters:**

message - the detail message

```

public class HabitacionSuite extends Habitacion {
    public HabitacionSuite(int idHotel, int numero) {
        super(idHotel, numero);
    }

    @Override
    public void mostrarDetalles() {
        System.out.println("Habitación SUITE Nº " + numero + " - Hotel ID: " + idHotel + " - ");
    }

    @Override
    public double calcularPrecio() {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException(message: "Unimplemented method 'calcularPrecio()'");
    }
}

```

**Refactor recomendado:** Extract Interface de modo que sólo las clases que necesitan tal método la implementen, o por el contrario, implementar el método basado en la lógica de las clases de ser necesario.

## 8) Speculative Generality

**Descripción:** Se declara un atributo que no se. En Hospedaje existe hotel pero en el código actual no participa.

**Impacto:** Añade complejidad innecesaria al código y puede confundir a otros desarrolladores sobre su propósito o uso.

**Ubicación exacta:**

- Archivo: src/main/java/com/example/hospedaje/Hospedaje.java
- Clase: Hospedaje
- Campo: Hotel

**Evidencia:**

```
public class Hospedaje implements Reservable {
    private Hotel hotel;
```

**Refactor recomendado:** Utilizará Remove Dead Field para eliminar el campo hotel de la clase para simplificar y Simplify Class si el campo no se usa.

## 9) Comments

**Descripción:** En la clase AgenteDeSoporte se han agregado varios comentarios para explicar líneas obvias de código las cuales fácilmente se puede reconocer su funcionalidad.

**Impacto:** Añade líneas innecesarias de código y aumenta el ruido visual, dificultando leer el código relevante.

**Ubicación exacta:**

- Archivo: src/main/java/com/example/usuarios\_y\_roles/AgenteDeSoporte.java
- Clase: AgenteDeSoporte
- Métodos:

**Evidencia:**

```
public void gestionarIncidente(Incidente incidente, GestorDeAccion gestorDeAcci
    if (incidente == null) {
        System.out.println(x: "Error: incidente es null. No se puede gestionar."
        return; // o lanzar una excepción controlada si quieres
    }

    if (gestorDeAccion == null) {
        System.out.println(x: "Error: gestorDeAccion es null. No se puede obtener"
```

**Refactor recomendado:** Podemos eliminar las líneas de comentarios y además usar la técnica de refactorización Rename Method de modo que sea mucho más entendible a simple vista

## 10) Dead Code

**Descripción:** El método void reservar() en la clase no está implementado (lanza una excepción UnsupportedOperationException) y no es llamado ni utilizado en ningún otro método de la clase ni desde fuera de ella.

**Impacto:** Añade ruido y confusión al código, dificultando su comprensión y mantenimiento.

**Ubicación exacta:**

- Archivo: src/main/java/com/example/hospedaje/Hospedaje.java
- Clase: Hospedaje
- Método: reservar()

**Evidencia:**

```
void reservar() {
    throw new UnsupportedOperationException(message: "Not supported yet.");
}
```

**Refactor recomendado:** Eliminar el código no utilizado y los archivos innecesarios.

## Sección D: Refactorización del Código

Refactoring:

Extract Method

Code Smell Corregido: Duplicate Code

Antes/Después:

```
@Override
public void onReservaCreada(String mensaje, Usuario usuario) {
    if (usuario == null || usuario.getNombre() == null) {
        throw new IllegalArgumentException(s: "El usuario y su nombre no pueden ser nulos.");
    }
    if (mensaje == null) {
        throw new IllegalArgumentException(s: "El mensaje no puede ser nulo.");
    }

    String asunto = "Confirmación de reserva TravelStay";
    String cuerpo = String.format(format: "Hola %,n\n%,n\nGracias por elegir TravelStay",
        usuario.getNombre(),
        mensaje);
    notificador.enviar(asunto + "\n" + cuerpo, usuario);
}

@Override
public void onReservaCancelada(String mensaje, Usuario usuario) {

    if (usuario == null || usuario.getNombre() == null) {
        throw new IllegalArgumentException(s: "El usuario y su nombre no pueden ser nulos.");
    }
    if (mensaje == null) {
        throw new IllegalArgumentException(s: "El mensaje no puede ser nulo.");
    }

    String asunto = "Cancelación de reserva TravelStay";
    String cuerpo = String.format(
        format: "Hola %,n\n%,n\nLamentamos que hayas cancelado. Esperamos verte pronto",
        usuario.getNombre(),
        mensaje);
    notificador.enviar(asunto + "\n" + cuerpo, usuario);
}
```

```

package com.example.patrones.observer;

import com.example.interfaces.Notificador;
import com.example.mensajes.ConstructorMensaje;
import com.example.usuarios_y_rols.Usuario;
import com.example.validaciones.Validador;

public class EmailReservaObserver implements ReservaObserver {

    private Notificador notificador;

    public EmailReservaObserver(Notificador notificador) {
        if (notificador == null) {
            throw new IllegalArgumentException(s: "El notificador no puede ser nulo.");
        }
        this.notificador = notificador;
    }

    @Override
    public void onReservaCreada(String mensaje, Usuario usuario) {

        Validador.validarReserva(usuario, mensaje);
        String asunto = "Confirmación de reserva TravelStay";
        String cuerpo = ConstructorMensaje.construirReservaCreada(usuario, mensaje);
        notificador.enviar(asunto + "\n" + cuerpo, usuario);
    }

    @Override
    public void onReservaCancelada(String mensaje, Usuario usuario) {

        Validador.validarReserva(usuario, mensaje);

        String asunto = "Cancelación de reserva TravelStay";
        String cuerpo = ConstructorMensaje.construirReservaCancelada(usuario, mensaje);
        notificador.enviar(asunto + "\n" + cuerpo, usuario);
    }
}

```

Justificación:

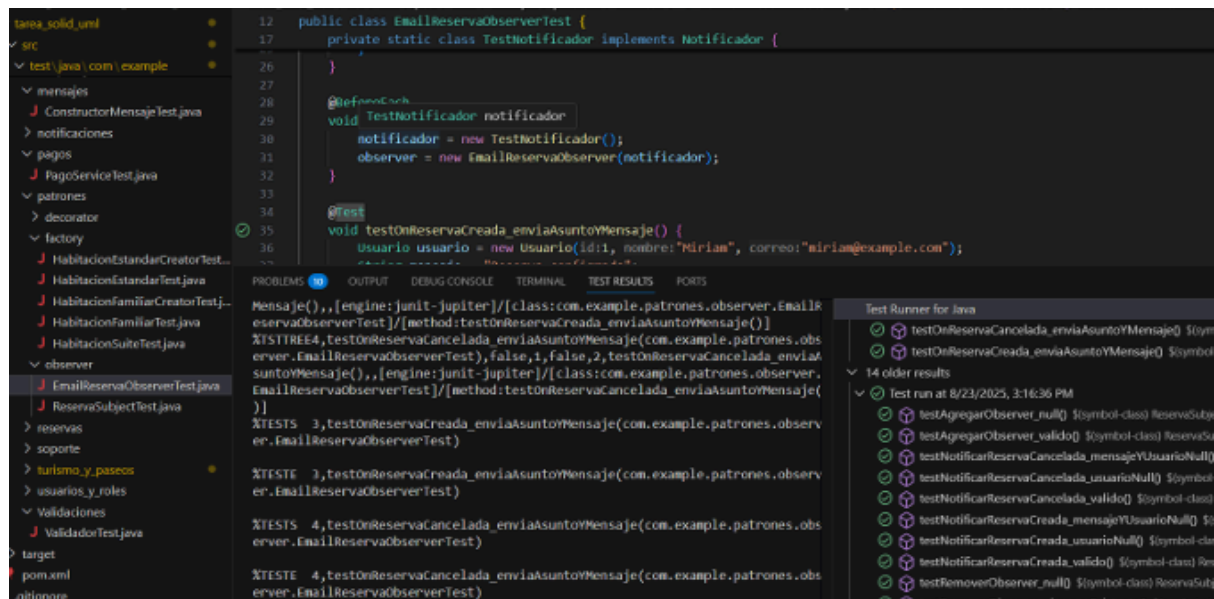
En la clase EmailReservaObserver, los métodos onReservaCreada y onReservaCancelada contenían código duplicado. Al aplicar Extract Method: las validaciones se movieron a validarReserva(), ahora la validación se hace en un solo lugar, la construcción del cuerpo del mensaje se movió a construirCuerpoReservaCreada() y construirCuerpoReservaCancelada() cada método tiene una responsabilidad

Pruebas unitarias:

The screenshot shows the IDE interface with the following components:

- Left Panel (Project Explorer):** Shows the project structure with folders like 'pagos', 'patrones', 'decorator', 'factory', and 'observer'. The file 'EmailReservaObserverTest.java' is selected under the 'observer' folder.
- Code Editor:** Displays the unit test code for 'EmailReservaObserverTest'. It includes methods like 'testOnReservaCancelada\_mensajeVacio()', 'testOnReservaCancelada\_usuarioNulo()', and 'testOnReservaCreada\_nombreVacio()'. The code uses 'assertEquals' and 'assertThrows' for assertions.
- Bottom Panel (Test Results):** Shows the results of the unit tests. It lists 14 tests, all of which passed (indicated by green checkmarks). The tests are grouped by method: 'testOnReservaCancelada' (11 tests) and 'testOnReservaCreada' (13 tests).





Refactoring:

## Form Template Method

Code Smell Corregido: Shotgun Surgery

Antes:

```

public class ReservaSubject {

    private List<ReservaObserver> observers = new ArrayList<>();

    public void agregarObserver(ReservaObserver observer) {

        if (observer == null) {
            throw new IllegalArgumentException(s: "observer no puede ser null");
        }

        observers.add(observer);
    }

    public void removerObserver(ReservaObserver observer) {

        if (observer == null) {
            throw new IllegalArgumentException(s: "observer no puede ser null");
        }

        observers.remove(observer);
    }

    public void notificarReservaCreada(String mensaje, Usuario usuario) {

        if (mensaje == null || usuario == null) {
            throw new IllegalArgumentException(s: "Mensaje y Usuario no pueden ser null");
        }

        for (ReservaObserver observer : observers) {
            observer.onReservaCreada(mensaje, usuario);
        }
    }

    public void notificarReservaCancelada(String mensaje, Usuario usuario) {

```

Despues



```

1 package com.example.patrones.observer;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import com.example.usuarios_y_roles.Usuario;
7 import com.example.validaciones.Validator;
8
9 public class ReservaSubject {
10
11     private List<ReservaObserver> observers = new ArrayList<>();
12
13     public void agregarObserver(ReservaObserver observer) {
14
15         Validator.validarObserver(observer);
16         observers.add(observer);
17     }
18
19     public void removerObserver(ReservaObserver observer) {}
20
21     public void notificarReservaCreada(String mensaje, Usuario usuario) {
22
23         Validator.validarReserva(usuario, mensaje);
24
25         for (ReservaObserver observer : observers) {
26             observer.onReservaCreada(mensaje, usuario);
27         }
28     }
29
30     public void notificarReservaCancelada(String mensaje, Usuario usuario) {
31
32         Validator.validarReserva(usuario, mensaje);
33     }
34 }

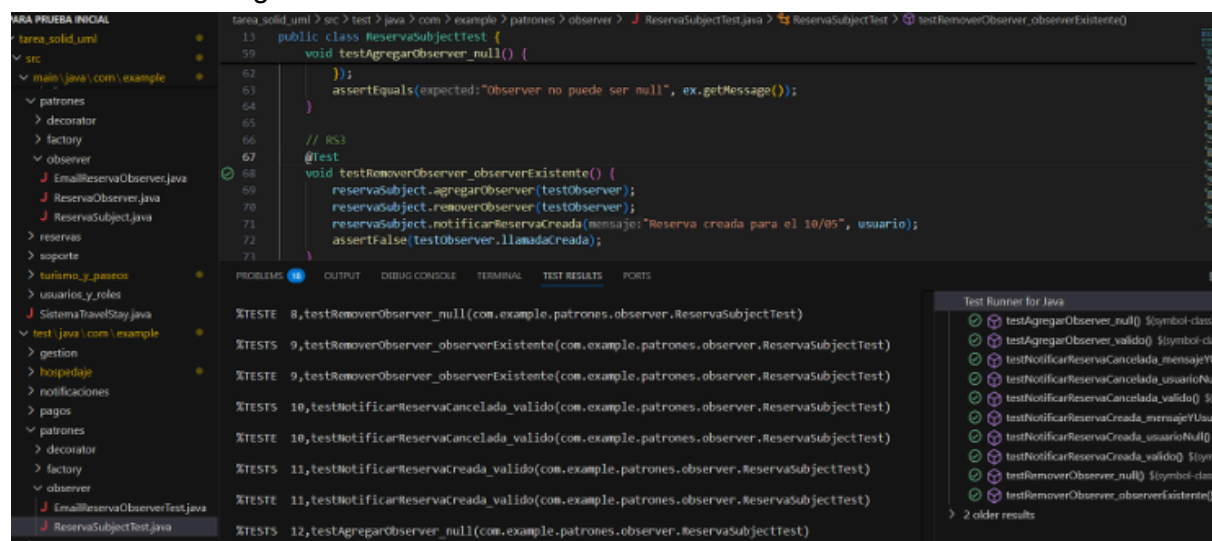
```

Justificación:

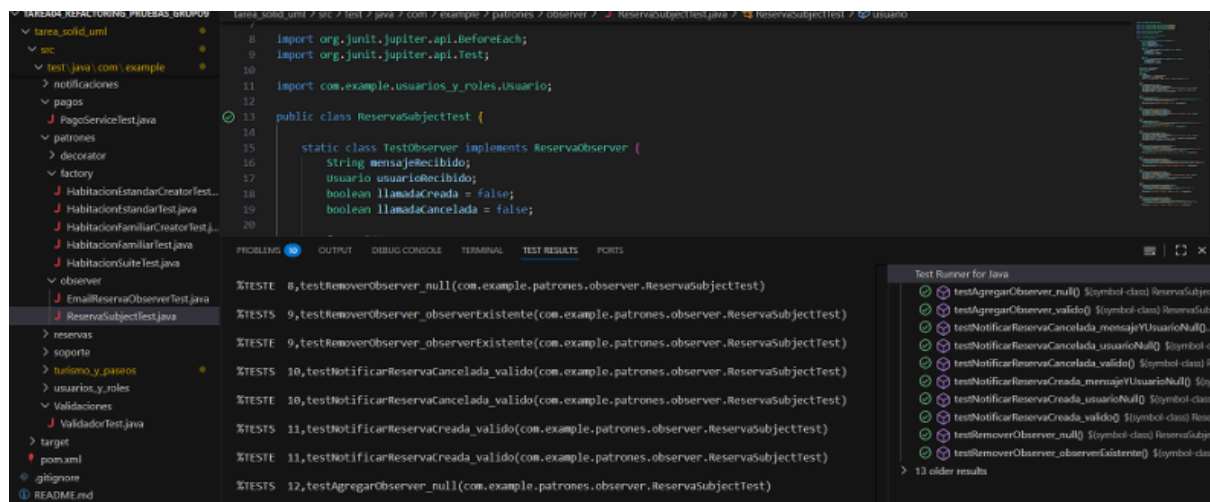
El problema era que cada subclase implementa su propia versión de mostrarDetalles(). La solución consiste en aplicar Form Template Method y definir en mostrarDetalles toda la lógica común.

Pruebas unitarias:

Antes de refactoring



Despues de refactoring



Refactoring:

## Move Method

Code Smell Corregido: Feature envy

Antes:

```
public PaqueteTuristico(Hospedaje hospedaje, PaseoTuristico paseo, double descuento) {
    this.hospedaje = hospedaje;
    this.paseo = paseo;
    this.descuento = descuento;
}

@Override
public double calcularPrecio() {
    // Usar el precio decorado de la habitación
    double precioHabitacion = 0.0;
    if (hospedaje.getHabitacion() != null) {
        precioHabitacion = hospedaje.getHabitacion().calcularPrecio();
    }
    return precioHabitacion + paseo.calcularPrecio() - descuento;
}

@Override
public boolean verificarDisponibilidad() {
    return hospedaje.verificarDisponibilidad() && paseo.verificarDisponibilidad();
}

@Override
public void mostrarDetalles() {
    System.out.println(x: "Paquete turistico:");
    if (hospedaje.getHabitacion() != null) {
        hospedaje.getHabitacion().mostrarDetalles();
    }
    System.out.println("Incluye paseo: " + paseo);
    System.out.println("Descuento aplicado: $" + descuento);
}
```

Después:

```

public class PaqueteTuristico implements IPaqueteTuristico {
    private Hospedaje hospedaje;
    private PaseoTuristico paseo;
    private double descuento;

    public PaqueteTuristico(Hospedaje hospedaje, PaseoTuristico paseo, double descuento) {
        this.hospedaje = hospedaje;
        this.paseo = paseo;
        this.descuento = descuento;
    }

    @Override
    public double calcularPrecio() {
        if (hospedaje.getHabitacion() != null) {
            return hospedaje.getHabitacion().calcularPrecio() + paseo.calcularPrecio() - descuento;
        } else { return paseo.calcularPrecio() - descuento; }
    }

    @Override
    public boolean verificarDisponibilidad() {
        return hospedaje.verificarDisponibilidad() && paseo.verificarDisponibilidad();
    }

    @Override
    public void mostrarDetalles() {
        logger.info(msg: "Paquete turistico:");
        if (hospedaje != null) {
            hospedaje.mostrarDetalles();
        }
        if (paseo != null) {
            paseo.mostrarDetalles();
        }
        logger.info("Descuento aplicado: $" + descuento);
    }
}

```

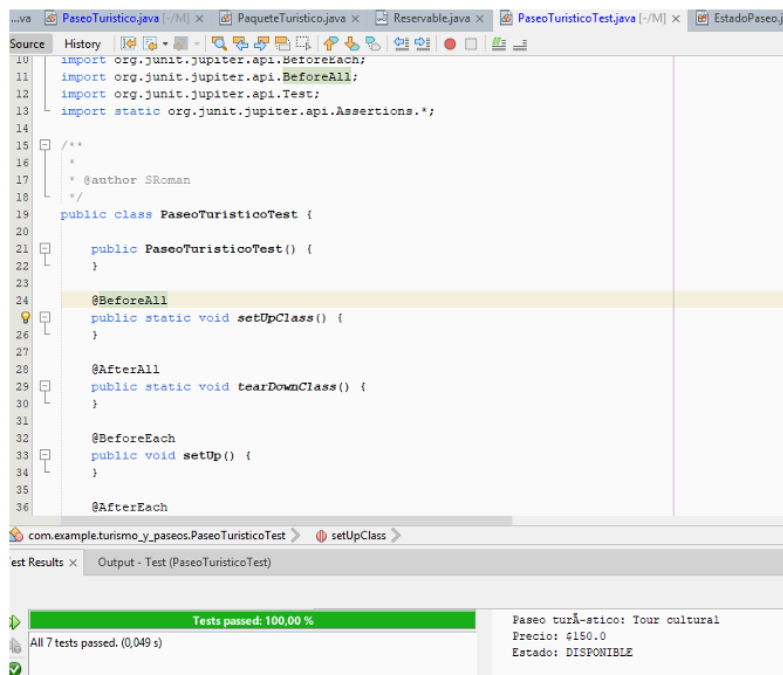
Justificación: Se aplicó la técnica Move Method trasladando la lógica de presentación de detalles desde la clase PaqueteTuristico hacia las clases responsables de los datos: Hospedaje y PaseoTuristico. Ahora, cada clase muestra sus propios detalles, y PaqueteTuristico solo coordina la llamada a estos métodos.

Pruebas unitarias:

Antes de refactoring

```

Source
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
26
```



Refactoring:

## Replace Data Value with Object

Code Smell Corregido: Primitive Obsession

Antes:

```

package com.example.usuarios_y_roles;
public class Usuario {
    private int id;
    private String nombre;
    private String correo;

    public Usuario(int id, String nombre, String correo) {
        this.id = id;
        this.nombre = nombre;
        this.correo = correo;
    }

    public String getCorreo() {
        return correo;
    }

    public String getNombre() {
        return nombre;
    }

    public int getId(){
        return id;
    }
}

```

Después:

```

package com.example.usuarios_y_rols;

public class Usuario {
    private int id;
    private String nombre;
    private Email correo;

    public Usuario(int id, String nombre, Email correo) {
        if (correo == null) {
            throw new NullPointerException(s: "El correo no puede ser null");
        }
        this.id = id;
        this.nombre = nombre;
        this.correo = correo;
    }

    public Email getCorreo() {
        return correo;
    }

    public String getNombre() {
        return nombre;
    }

    public int getId() {
        return id;
    }
}

```

Justificación:

Se reemplazó el campo correo de tipo String en la clase Usuario por un objeto de valor Email, el cual encapsula la validación y manejo de direcciones de correo electrónico. Esto centraliza la lógica y evita el uso de valores primitivos sin control.

Pruebas unitarias:

Antes de refactoring:

```

src > test > java > com > example > usuarios_y_rols > J UsuarioTest.java
14  /**
15
16  public class UsuarioTest {
17
18      public UsuarioTest() {
19      }
20
21
22
23      @BeforeAll
24      public static void setUpClass() {
25      }
26
27      @AfterAll
28      public static void tearDownClass() {
29      }
30
31  }
32
33  PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL TEST RESULTS PORTS
34
35  [/class:com.example.usuarios_y_rols.UsuarioTest]
36  %TSTREE3,testgetId(com.example.usuarios_y_rols.UsuarioTest),false,1,false,2,testgetId(),,[engine:j
37  unit-jupiter]/[class:com.example.usuarios_y_rols.UsuarioTest]/[method:testgetId()]
38  %TSTTREE4,testGetCorreo(com.example.usuarios_y_rols.UsuarioTest),false,1,false,2,testGetCorreo(),,[
39  engine:junit-jupiter]/[class:com.example.usuarios_y_rols.UsuarioTest]/[method:testGetCorreo()]
40  %TSTTREE5,testGetNombre(com.example.usuarios_y_rols.UsuarioTest),false,1,false,2,testGetNombre(),,[
41  engine:junit-jupiter]/[class:com.example.usuarios_y_rols.UsuarioTest]/[method:testGetNombre()]
42  %TSTESTS 3,testgetId(com.example.usuarios_y_rols.UsuarioTest)
43
44  %TSTESTE 3,testgetId(com.example.usuarios_y_rols.UsuarioTest)
45
46  %TSTESTS 4,testGetCorreo(com.example.usuarios_y_rols.UsuarioTest)
47
48  %TSTESTE 4,testGetCorreo(com.example.usuarios_y_rols.UsuarioTest)
49
50  %TSTESTS 5,testGetNombre(com.example.usuarios_y_rols.UsuarioTest)
51
52  %TSTESTE 5,testGetNombre(com.example.usuarios_y_rols.UsuarioTest)
53
54  %TSTESTS 5,1 old
55
56  %TSTESTE 5,1 old
57
58  %TSTESTS 5,1 old
59
60  %TSTESTE 5,1 old
61
62  %TSTESTS 5,1 old
63
64  %TSTESTE 5,1 old
65
66  %TSTESTS 5,1 old
67
68  %TSTESTE 5,1 old
69
70  %TSTESTS 5,1 old
71
72  %TSTESTE 5,1 old
73
74  %TSTESTS 5,1 old
75
76  %TSTESTE 5,1 old
77
78  %TSTESTS 5,1 old
79
80  %TSTESTE 5,1 old
81
82  %TSTESTS 5,1 old
83
84  %TSTESTE 5,1 old
85
86  %TSTESTS 5,1 old
87
88  %TSTESTE 5,1 old
89
90  %TSTESTS 5,1 old
91
92  %TSTESTE 5,1 old
93
94  %TSTESTS 5,1 old
95
96  %TSTESTE 5,1 old
97
98  %TSTESTS 5,1 old
99
100 %TSTESTE 5,1 old

```

Despues de refactoring:

```

11 public class UsuarioTest {
12
13     @Test
14     public void testGetId() {
15         Email email = new Email(valor:"maria@mail.com");
16         Usuario usuario = new Usuario(id:1, nombre:"Maria", email);
17         assertEquals(expected:1, usuario.getId());
18     }
19
20     @Test
21     public void testGetNombre() {
22         Email email = new Email(valor:"juan@mail.com");
23         Usuario usuario = new Usuario(id:2, nombre:"Juan", email);
24         assertEquals(expected:"Juan", usuario.getNombre());
25     }
26 }

```

TEST RESULTS

```

XTESTE 3, testGetId(com.example.usuarios_y_roles.UsuarioTest)
XTESTE 4, testUsuarioConEmailNulo(com.example.usuarios_y_roles.UsuarioTest)
XTESTE 4, testUsuarioConEmailNulo(com.example.usuarios_y_roles.UsuarioTest)
XTESTE 5, testUsuarioConEmailInvalido(com.example.usuarios_y_roles.UsuarioTest)
XTESTE 5, testUsuarioConEmailInvalido(com.example.usuarios_y_roles.UsuarioTest)
XTESTE 6, testGetCorreo(com.example.usuarios_y_roles.UsuarioTest)
XTESTE 6, testGetCorreo(com.example.usuarios_y_roles.UsuarioTest)
XTESTE 7, testGetNombre(com.example.usuarios_y_roles.UsuarioTest)
XTESTE 7, testGetNombre(com.example.usuarios_y_roles.UsuarioTest)
XRUNTIME 174

```

Test Runner for Java

- testGetCorreo()
- testGetId() Succeeded
- testGetNombre()
- testUsuarioConEmailNulo()
- testUsuarioConEmailInvalido()

> 5 older results

Refactoring:

## Encapsulate Field

Code Smell Corregido: Data Class

Antes:

```

import com.example.interfaces.Reservable;

public class PaseoTuristico implements Reservable {
    private String nombre;
    private EstadoPaseo estado; // disponible, agotado, cancelado
    private double precio;

    public PaseoTuristico(String nombre, double precio) {
        this.nombre = nombre;
        this.precio = precio;
        this.estado = EstadoPaseo.DISPONIBLE;
    }

    public double calcularPrecio() {
        return precio;
    }

    public boolean verificarDisponibilidad() {
        if (estado.equals(EstadoPaseo.DISPONIBLE)) return true;
        else {return false;}
    }
}

```

Después:

```

private Hospedaje hospedaje;
private PaseoTuristico paseo;
private double descuento;

public PaqueteTuristico(Hospedaje hospedaje, PaseoTuristico paseo, double descuento) {
    this.hospedaje = hospedaje;
    this.paseo = paseo;
    this.descuento = descuento;
}

@Override
public double calcularPrecio() {
    if (hospedaje.getHabitacion() != null) {
        return hospedaje.getHabitacion().calcularPrecio() + paseo.calcularPrecio() - descu
    }
    else{ return paseo.calcularPrecio() - descuento; }
}

@Override
public boolean verificarDisponibilidad() {
    return hospedaje.verificarDisponibilidad() && paseo.verificarDisponibilidad();
}

@Override
public void mostrarDetalles() {
    logger.info(msg: "Paquete turistico:");
    if (hospedaje != null) {
        hospedaje.mostrarDetalles();
    }
    if (paseo != null) {
        paseo.mostrarDetalles();
    }
    logger.info("Descuento aplicado: $" + descuento);
}

```

Justificación: Se encapsularon los campos nombre, precio y estado dentro de la clase PaseoTuristico, eliminando la lógica trivial que solo devolvía los valores.

```

private Hospedaje hospedaje;
private PaseoTuristico paseo;
private double descuento;

public PaqueteTuristico(Hospedaje hospedaje, PaseoTuristico paseo, double descuento) {
    this.hospedaje = hospedaje;
    this.paseo = paseo;
    this.descuento = descuento;
}

@Override
public double calcularPrecio() {
    if (hospedaje.getHabitacion() != null) {
        return hospedaje.getHabitacion().calcularPrecio() + paseo.calcularPrecio() - descu
    }
    else{ return paseo.calcularPrecio() - descuento; }
}

@Override
public boolean verificarDisponibilidad() {
    return hospedaje.verificarDisponibilidad() && paseo.verificarDisponibilidad();
}

@Override
public void mostrarDetalles() {
    logger.info(msg: "Paquete turistico:");
    if (hospedaje != null) {
        hospedaje.mostrarDetalles();
    }
    if (paseo != null) {
        paseo.mostrarDetalles();
    }
    logger.info("Descuento aplicado: $" + descuento);
}

```

Pruebas unitarias:  
Antes de refactoring

```

79 PaseoTuristico paseo = new PaseoTuristico("Tour cultural", 150.0);
80
81 java.lang.reflect.Field estadoField = paseo.getClass().getDeclaredField("estado");
82 estadoField.setAccessible(true);
83 estadoField.set(paseo, EstadoPaseo.CANCELADO);
84
85 assertFalse(paseo.verificarDisponibilidad());
86 }
87
88 @Test
89 public void testCalcularPrecioCero() {
90     PaseoTuristico paseo = new PaseoTuristico("Tour gratuito", 0.0);
91
92     assertEquals(0.0, paseo.calcularPrecio(), 0.001);
93 }
94
95 @Test
96 public void testCalcularPrecioNegativo() {
97     PaseoTuristico paseo = new PaseoTuristico("Tour especial", -50.0);
98
99     assertEquals(-50.0, paseo.calcularPrecio(), 0.001);
100 }
101
102 @Test
103 public void testMostrarDetalles() {
104     PaseoTuristico paseo = new PaseoTuristico("Tour cultural", 150.0);
105 }

```

com.example.turismo\_y\_paseos.PaseoTuristicoTest

Test Results x Output - Test (PaseoTuristicoTest)

Tests passed: 100,00 %

All 7 tests passed. (0,064 s)

Paseo turístico: Tour cultural  
Precio: \$150.0  
Estado: DISPONIBLE

Despues de refactoring:

```

10 import org.junit.jupiter.api.BeforeEach;
11 import org.junit.jupiter.api.BeforeAll;
12 import org.junit.jupiter.api.Test;
13 import static org.junit.jupiter.api.Assertions.*;
14
15 /**
16  *
17  * @author S.Roman
18  */
19 public class PaseoTuristicoTest {
20
21     public PaseoTuristicoTest() {
22     }
23
24     @BeforeAll
25     public static void setUpClass() {
26     }
27
28     @AfterAll
29     public static void tearDownClass() {
30     }
31
32     @BeforeEach
33     public void setUp() {
34     }
35
36     @AfterEach

```

com.example.turismo\_y\_paseos.PaseoTuristicoTest

Test Results x Output - Test (PaseoTuristicoTest)

Tests passed: 100,00 %

All 7 tests passed. (0,049 s)

Paseo turístico: Tour cultural  
Precio: \$150.0  
Estado: DISPONIBLE

Refactoring:

## Speculative Generality

Code Smell Corregido: Remove Dead Field

Antes:



```
public class Hospedaje implements Reservable {
    private Hotel hotel;
```

Después:

```
public class Hospedaje implements Reservable {
    private IHabitacion habitacion;

    public Hospedaje(IHabitacion habitacion) {
```

Justificación: Se eliminó el campo hotel de la clase Hospedaje, ya que no se inicializaba ni se utilizaba en ningún método. La clase ahora solo mantiene la referencia a IHabitacion, simplificando su estructura y evitando complejidad innecesaria.

Pruebas unitarias:

Antes de refactoring:

```
hospedaje.bloquearTemporalmente();
assertFalse(h.isReservadoLlamado(), "No debe llamar a reservar() c

}

@Test
void testGetHabitacion() {
    HabitacionStub h = new HabitacionStub(40.0, true);
    Hospedaje hospedaje = new Hospedaje(h);
    assertEquals(h, hospedaje.getHabitacion());
}

//Test complementarios
@Test
public void testCalcularPrecioCero() {
    HabitacionStub h = new HabitacionStub(0.0, true);
    Hospedaje hospedaje = new Hospedaje(h);
    assertEquals(0.0, hospedaje.calcularPrecio(), 0.0001);
}

@Test
public void testCalcularPrecioNegativo() {
    HabitacionStub h = new HabitacionStub(-50.0, true);
    Hospedaje hospedaje = new Hospedaje(h);
    assertEquals(-50.0, hospedaje.calcularPrecio(), 0.0001);
}

@Test
```

om.example.hospedaje.HospedajeTest

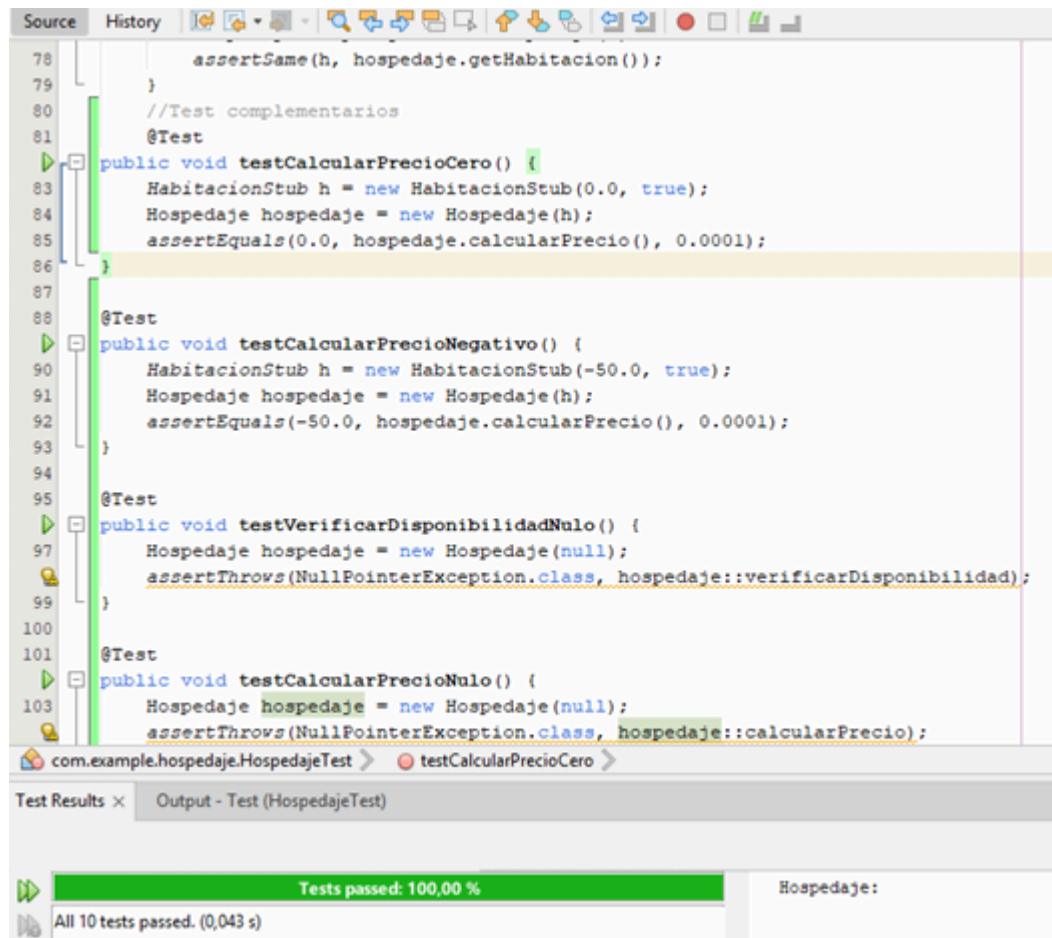
results × Output - Test (HospedajeTest)

Tests passed: 100.00 %

Hospedaje:

All 10 tests passed. (0.044 s)

Despues de refactoring:



Refactoring:

## Implementación de Método

Code Smell Corregido: Refused Bequest

Antes:

```

public class HabitacionFamiliar extends Habitacion {
    public HabitacionFamiliar(int idHotel, int numero) {
        super(idHotel, numero);
    }

    @Override
    public void calcularPrecio() {
        System.out.println("Calculando precio para HabitacionFamiliar");
        throw new UnsupportedOperationException("Método no implementado");
    }
}

```

**java.lang.UnsupportedOperationException**

public UnsupportedOperationException(String message)

Constructs an UnsupportedOperationException with the specified detail message.

**Parameters:**

message - the detail message

```

public class HabitacionSuite extends Habitacion {
    public HabitacionSuite(int idHotel, int numero) {
        super(idHotel, numero);
    }

    @Override
    public void mostrarDetalles() {
        System.out.println("Habitación SUITE Nº " + numero + " - Hotel ID: " + idHotel + " - ");
    }

    @Override
    public double calcularPrecio() {
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException(message: "Unimplemented method 'calcularPrecio()'");
    }
}

```

Después:

```

package com.example.patrones.factory;

import com.example.enums.TipoHabitacion;

public class HabitacionFamiliar extends Habitacion {
    public HabitacionFamiliar(int idHotel, int numero) {
        super(idHotel, numero, TipoHabitacion.FAMILIAR);
    }

    @Override
    public double calcularPrecio() {
        return 120.00;
    }
}

package com.example.patrones.factory;

import com.example.enums.TipoHabitacion;

public class HabitacionSuite extends Habitacion {
    public HabitacionSuite(int idHotel, int numero) {
        super(idHotel, numero, TipoHabitacion.SUITE);
    }

    @Override
    public double calcularPrecio() {
        return 180.00;
    }
}

```

Justificación: Se implementó el método `calcularPrecio` de las clases `HabitaciónFamiliar` y `HabitaciónSuite` según la lógica que manejan estas clases. El método ahora retorna un valor específico para cada clase, de modo que no es innecesaria su sobrescritura.

Pruebas unitarias:

```

    }

    @Test
    void reservarTestCambiaReservado() {
        HabitacionSuite suite = new HabitacionSuite(2, 201);
        suite.estado = EstadoHabitacion.OCUPADA;
        suite.reservar();
        Assertions.assertEquals(EstadoHabitacion.RESERVADA, suite.estado);
    }

    @Test
    void mostrarDetallesTestSinErrores() {
        HabitacionSuite suite = new HabitacionSuite(3, 303);
        suite.reservar();
        Assertions.assertDoesNotThrow(suite::mostrarDetalles);
        Assertions.assertEquals(EstadoHabitacion.RESERVADA, suite.estado);
    }

    @Test
    void estaDisponibleRetornaTrueSiDisponible() {
        HabitacionSuite suite = new HabitacionSuite(5, 501);
        Assertions.assertTrue(suite.estaDisponible());
    }

    @Test
    void calcularPrecioLanzaUnsupportedOperation() {
        HabitacionSuite suite = new HabitacionSuite(5, 501);
        Assertions.assertEquals(100.0, suite.calcularPrecio(), 0.01, "calcularPrecio debe retornar 100.0");
    }

```

Refactoring:

## Rename Method

Code Smell Corregido: Comments

Antes:

```

public void gestionarIncidente(Incidente incidente, GestorDeAccion gestorDeAcci
    if (incidente == null) {
        System.out.println(x: "Error: incidente es null. No se puede gestionar."
        return; // o lanzar una excepción controlada si quieres
    }

    if (gestorDeAccion == null) {
        System.out.println(x: "Error: gestorDeAccion es null. No se puede obten

```

Después:

```

public AgenteDeSoporte(int id, String nombre, Email correo) {
    super(id, nombre, correo);
}

public void gestionarIncidente(Incidente incidente, GestorDeAccion gestorDeAccion) {
    if (incidente == null) {
        logger.warning(msg: "Error: incidente es null. No se puede gestionar.");
        return;
    }

    if (gestorDeAccion == null) {
        logger.warning(msg: "Error: gestorDeAccion es null. No se puede obtener acción.");
        return;
    }
}

```

Justificación: Se renombró el método `escalar()` de la clase `Incidente` de modo que la interpretación de su funcionalidad sea mucho más sencilla. Además se eliminaron comentarios innecesarios de modo que el código es mucho más comprensible y sin tanto ruido visual.

Pruebas unitarias:

```

@Test
void testGestionarIncidente_resolver_sinMockito() {
    Email correo = new Email("agente@mail.com");
    Usuario usuario = new Usuario(1, "Juan", correo);
    Incidente incidente = new Incidente(1, usuario, "Problema X");
    AgenteDeSoporte agente = new AgenteDeSoporte(2, "Agente1", correo);
    GestorDeAccion gestor = new GestorDeAccion() {
        public String obtenerAccion(Incidente inc) {
            return "resolver";
        }
    };
    agente.gestionarIncidente(incidente, gestor);
    Assertions.assertEquals(EstadoIncidente.RESUELTO, incidente.getEstado());
}

@Test
void testGestionarIncidente_escalar() {
    Email correo = new Email("agente2@mail.com");
    Usuario usuario = new Usuario(1, "Juan", correo);
    Incidente incidente = new Incidente(2, usuario, "Problema Y");
    AgenteDeSoporte agente = new AgenteDeSoporte(2, "Agente2", correo);
    GestorDeAccion gestor = new GestorDeAccion() {
        public String obtenerAccion(Incidente inc) {
            return "escalar";
        }
    };
    agente.gestionarIncidente(incidente, gestor);
}

```

Refactoring:

## Inline Temp

Code Smell Corregido: Temporary Variable

Antes:

```
@Override
public double calcularPrecio() {
    // Usar el precio decorado de la habitación
    double precioHabitacion = 0.0;
    if (hospedaje.getHabitacion() != null) {
        precioHabitacion = hospedaje.getHabitacion().calcularPrecio();
    }
    return precioHabitacion + paseo.calcularPrecio() - descuento;
}
```

Después:

```
@Override
public double calcularPrecio() {
    if (hospedaje.getHabitacion() != null) {
        return hospedaje.getHabitacion().calcularPrecio() + paseo.calcularPrecio() - descu
    }
    else{ return  paseo.calcularPrecio() - descuento; }
}

@Override
```

Justificación: Se refactoriza el código eliminando la variable temporal precioHabitacion que sólo era utilizada para un retorno posterior sin alguna utilidad importante.

Pruebas unitarias:

```

public class PaqueteTuristicoTest {
    public PaqueteTuristicoTest() {
    }

    @Test
    void testCalcularPrecio() {
        IHabitacion habitacion = new HabitacionTest(200.0, true);
        Hospedaje hospedaje = new Hospedaje(habitacion);
        PaseoTuristico paseo = new PaseoTuristico("Tour Cultural", 150.0);
        double descuento = 30.0;
        PaqueteTuristico paquete = new PaqueteTuristico(hospedaje, paseo, descuento);
        double esperado = 320.0;
        Assertions.assertEquals(esperado, paquete.calcularPrecio(), 0.01);
    }

    @Test
    void testVerificarDisponibilidad_true() {
        IHabitacion habitacion = new HabitacionTest(100.0, true);
        Hospedaje hospedaje = new Hospedaje(habitacion);
        PaseoTuristico paseo = new PaseoTuristico("Tour Aventura", 75.0);
        PaqueteTuristico paquete = new PaqueteTuristico(hospedaje, paseo, 0.0);
        Assertions.assertTrue(paquete.verificarDisponibilidad());
    }

    @Test
    void testVerificarDisponibilidad_false() {
        IHabitacion habitacion = new HabitacionTest(100.0, true);
        Hospedaje hospedaje = new Hospedaje(habitacion);
        new PaseoTuristico("Tour Aventura", 75.0);
        PaseoTuristico paseoNoDisponible = new PaseoTuristico("Tour Cancelado", 75.0) {
            public boolean verificarDisponibilidad() {
                return false;
            }
        };
    }
}

```

Refactoring:

## Remove Dead Code

Code Smell Corregido: Dead code

Antes:

```

void reservar() {
    throw new UnsupportedOperationException(message: "Not supported yet.");
}

```

Después:

```

    public void bloquearTemporalmente() {
        if (habitacion.estaDisponible()) {
            habitacion.reservar();
        }
    }

    public IHabitacion getHabitacion() {
        return habitacion;
    }
}

```

Justificación: Se eliminó el método reservar de la clase Hospedaje puesto que no tenía objetivo definido ni funcionalidad alguna dentro del código formando parte del code smell Dead Code. Al no tener lógica implementada, su eliminación elimina líneas innecesarias de código que no permitan una correcta interpretación del código.

Pruebas unitarias:

```

import org.junit.jupiter.api.Test;

public class HospedajeTest {
    public HospedajeTest() {
    }

    @Test
    void testCalcularPrecio() {
        HabitacionStub h = new HabitacionStub(100.0, true);
        Hospedaje hospedaje = new Hospedaje(h);
        Assertions.assertEquals(100.0, hospedaje.calcularPrecio(), 1.0E-4);
    }

    @Test
    void testVerificarDisponibilidad() {
        HabitacionStub h = new HabitacionStub(50.0, true);
        Hospedaje hospedaje = new Hospedaje(h);
        Assertions.assertTrue(hospedaje.verificarDisponibilidad());
    }

    @Test
    void testBloquearTemporalmente_ReservarSiDisponible() {
        HabitacionStub h = new HabitacionStub(80.0, true);
        Hospedaje hospedaje = new Hospedaje(h);
        hospedaje.bloquearTemporalmente();
        Assertions.assertTrue(h.isReservadoLlamado(), "Debe llamar a reservar() cuando está disp");
    }

    @Test
    void testBloquearTemporalmente_NoReservarSiNoDisponible() {
        HabitacionStub h = new HabitacionStub(80.0, false);
        Hospedaje hospedaje = new Hospedaje(h);
        hospedaje.bloquearTemporalmente();
        Assertions.assertFalse(h.isReservadoLlamado(), "No debe llamar a reservar() cuando no es");
    }
}


















```



# Link del repositorio

URL: <https://github.com/JosuePach3co/Tarea2-DS.git>

Colaboracion:

<b>Corrigiendo detalles menores en las cla...</b>  JosuePach3co • 1 hour ago	
<b>correcciones sugeridas</b>  AxelRoman1 • 1 hour ago	
<b>correcciones sugeridas</b>  AxelRoman1 • 1 hour ago	
<b>correcciones sugeridas</b>  AxelRoman1 • 2 hours ago	
<b>correcciones sugeridas</b>  AxelRoman1 • 2 hours ago	
<b>correcciones sugeridad</b>  AxelRoman1 • 2 hours ago	<b>correcciones sugeridas</b>  AxelRoman1 • 2 hours ago
<b>correcciones sugeridas</b>  AxelRoman1 • 2 hours ago	<b>correcciones sugeridad</b>  AxelRoman1 • 2 hours ago
<b>Merge branch 'main' of https://github....</b>  JosuePach3co • 2 hours ago	<b>Se agrega pruebas unitarias en Junit</b>  JosuePach3co • 2 hours ago
<b>Conflicto resuelto</b>  JosuePach3co • 2 hours ago	<b>Implement null checks in Hotel class m...</b>  AxelRoman1 • 2 hours ago
	<b>replace, speculative, implemetation, ra...</b>  Arihuz124 • 3 hours ago
	<b>Replace Data Value with Object</b>  Arihuz124 • 5 hours ago
	<b>ferm template method y move mothod</b>  Arihuz124 • 5 hours ago
	<b>Extract Method</b>  Arihuz124 • 5 hours ago
	<b>reviso</b>