IDENTIFICACIÓN DE CODE SMELLS

Clases demasiado grandes o con demasiadas responsabilidades.

La clase Hotel es demasiado grande y tiene demasiadas responsabilidades. Se encarga de la gestión de reservas, clientes y habitaciones.

Métodos demasiado largos o complejos.

El método main() de la clase Main es demasiado largo y tiene demasiadas responsabilidades. Se encarga de registrar habitaciones, clientes y de actuar como menú.

El método reservarHabitacion() de la clase Hotel() es demasiado largo y tiene demasiadas responsabilidades.

if anidados o complejos. Posiblidad de aplicar Early Return.

El método reservarHabitacion() de la clase Hotel tiene demasiados if anidados. Se pueden aplicar Early return para simplificar el código.

Nombres de variables y métodos poco descriptivos.

Variable “tipo” en Main poco descriptiva.

Código duplicado.

Él método registrarHabitacion() de la clase Hotel se repite más adelante (registrarHabitaciones())

Código inalcanzable.

El método reservarHabitación() no puede alcanzar return 0.

Comentarios poco adecuados o redundantes.

Hay partes del código muy comentadas y otras que carecen de comentarios.

Uso de *Magic Numbers*.

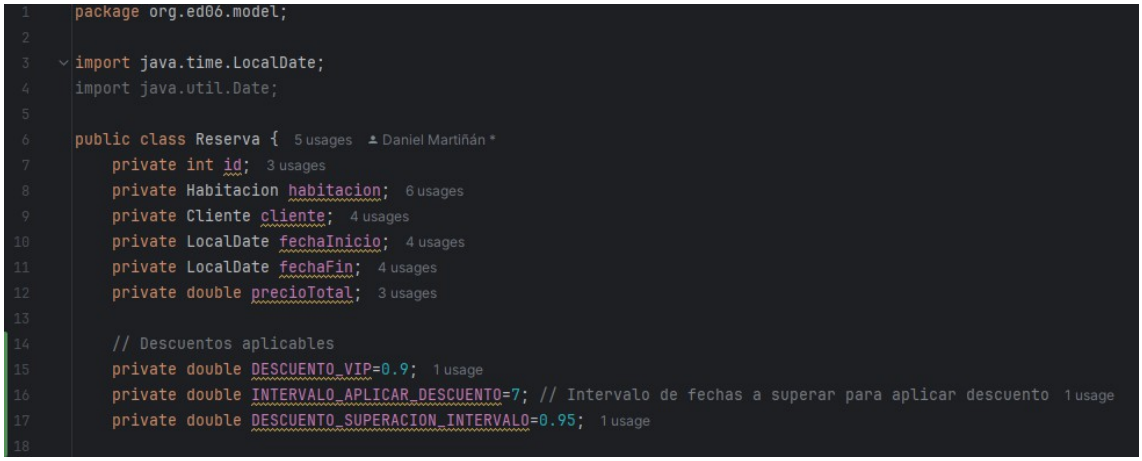
La clase Reserva tiene números mágicos para aplicar los descuentos.

Variables redundantes.

La variables pb en calcularPrecioFinal() es redundante.

REFACTORIZACION

Eliminación de Magic Numbers en Reserva.

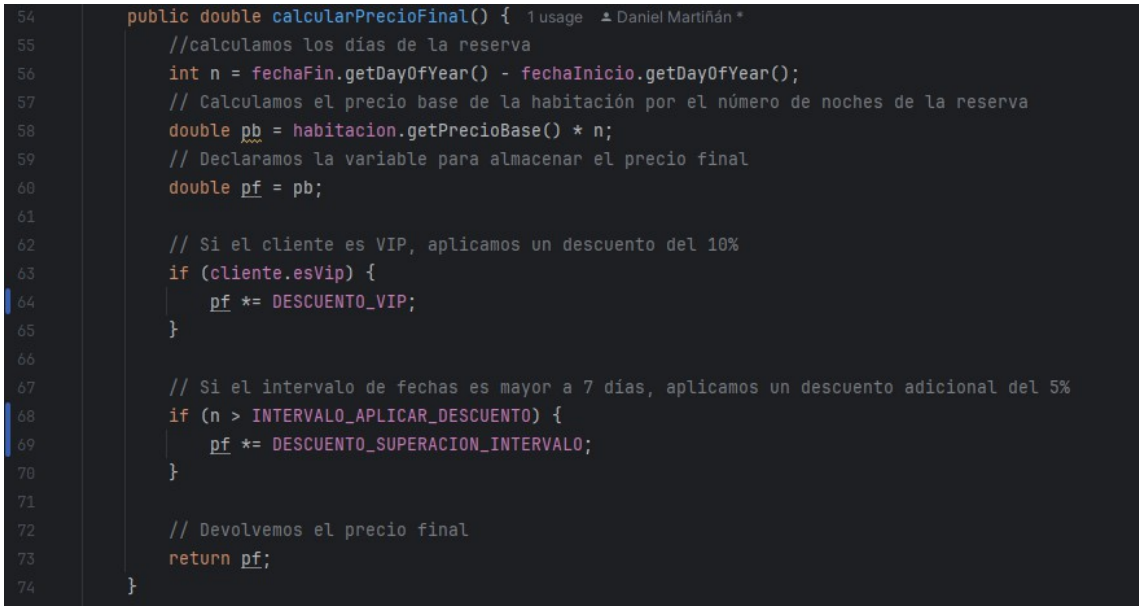


```java
package org.ed06.model;

import java.time.LocalDate;
import java.util.Date;

public class Reserva {  5 usages  Daniel Martiñán *
    private int id;  3 usages
    private Habitacion habitacion;  6 usages
    private Cliente cliente;  4 usages
    private LocalDate fechaInicio;  4 usages
    private LocalDate fechaFin;  4 usages
    private double precioTotal;  3 usages

    // Descuentos aplicables
    private double DESCUENTO_VIP=0.9;  1 usage
    private double INTERVALO_APLICAR_DESCUENTO=7; // Intervalo de fechas a superar para aplicar descuento  1 usage
    private double DESCUENTO_SUPERACION_INTERVALO=0.95;  1 usage
```



```java
    public double calcularPrecioFinal() {  1 usage  Daniel Martiñán *
        //calculamos los días de la reserva
        int n = fechaFin.getDayOfYear() - fechaInicio.getDayOfYear();
        // Calculamos el precio base de la habitación por el número de noches de la reserva
        double pb = habitacion.getPrecioBase() * n;
        // Declaramos la variable para almacenar el precio final
        double pf = pb;

        // Si el cliente es VIP, aplicamos un descuento del 10%
        if (cliente.esVip) {
            pf *= DESCUENTO_VIP;
        }

        // Si el intervalo de fechas es mayor a 7 días, aplicamos un descuento adicional del 5%
        if (n > INTERVALO_APLICAR_DESCUENTO) {
            pf *= DESCUENTO_SUPERACION_INTERVALO;
        }

        // Devolvemos el precio final
        return pf;
    }
```

Eliminación de variables redundantes

```java
    // Calcula el precio total de la reserva. Para calcular el precio total, se debe calcular el precio ba
    // Devuelve precio total de la reserva
    public double calcularPrecioFinal() {  1 usage   Daniel Martiñán *
        //calculamos los días de la reserva
        int n = fechaFin.getDayOfYear() - fechaInicio.getDayOfYear();

        // Calculamos el precio base de la habitación por el número de noches de la reserva
        double pf = habitacion.getPrecioBase() * n;

        // Si el cliente es VIP, aplicamos un descuento del 10%
        if (cliente.esVip) {
            pf *= DESCUENTO_VIP;
        }

        // Si el intervalo de fechas es mayor a 7 días, aplicamos un descuento adicional del 5%
        if (n > INTERVALO_APLICAR_DESCUENTO) {
            pf *= DESCUENTO_SUPERACION_INTERVALO;
        }

        // Devolvemos el precio final
        return pf;
    }
```

Eliminación del método registrarHabitacion() y unificar en registrarHabitaciones.

```java
    // Método para agregar una nueva habitación al hotel
    public void registrarHabitacion(String tipo, double precioBase) {  9 usages   Daniel Martiñán +1
        Habitacion habitacion = new Habitacion( numero: habitaciones.size() + 1, tipo, precioBase);
        habitaciones.add(habitacion);
        reservasPorHabitacion.put(habitacion.getNumero(), new ArrayList<>());
    }

    // Eliminado código duplicado


    public void listarHabitacionesDisponibles() {  1 usage   Daniel Martiñán
        for(Habitacion habitacion : habitaciones) {
            if(habitacion.isDisponible()) {
                System.out.println("Habitación #" + habitacion.getNumero() + " - Tipo: " + habitacion.getTipo() + " - Precio base: " + habitacion.getPrecioBase());
            }
        }
    }
```

Extraemos el código para rellenar el hotel y lo metemos en un método:

```java


    private static void rellenarDummy(){  no usages   new *
        // Registramos algunas habitaciones
        List<String> tiposHabitaciones = Arrays.asList("SIMPLE", "DOBLE", "SUITE", "LITERAS");
        List<Double> preciosHabitaciones = Arrays.asList(50.0, 80.0, 120.0, 200.0, 65.0, 100.0, 150.0, 250.0);
        hotel.registrarHabitaciones(tiposHabitaciones, preciosHabitaciones);

        // Registramos algunos clientes
        hotel.registrarCliente( nombre: "Daniel",  email: "daniel@daniel.com",  dni: "12345678A",  esVip: true);
        hotel.registrarCliente( nombre: "Adrián",  email: "adrian@adrian.es",  dni: "87654321B",  esVip: false);
    }

}
```

División de responsabilidades del Hotel en Gestores:

```java
package org.ed06.model;
import java.time.LocalDate;
import java.util.List;

public class Hotel {  3 usages  cielo +2
    private final String nombre;  1 usage
    private final String direccion;  1 usage
    private final String telefono;  1 usage

    public final GestorClientes gestorClientes = new GestorClientes();  5 usages
    public final GestorHabitaciones gestorHabitaciones = new GestorHabitaciones();  6 usages
    public final GestorReservas gestorReservas = new GestorReservas();  3 usages

    public Hotel(String nombre, String direccion, String telefono) {  1 usage  Daniel Martiñán
        this.nombre = nombre;
        this.direccion = direccion;
        this.telefono = telefono;
    }

    public void inicializar() {  no usages  cielo
        gestorReservas.inicializarReservas(gestorHabitaciones.getTodasHabitaciones());
    }

    public int reservarHabitacion(int clienteId, String tipo, LocalDate entrada, LocalDate salida) {  1 usage  cielo +1
        Cliente cliente = gestorClientes.obtenerCliente(clienteId);
        if (cliente == null) {
            System.out.println("No existe el cliente con id " + clienteId);
            return -3;
        }

        Habitacion habitacion = gestorHabitaciones.buscarDisponiblePorTipo(tipo);
        if (habitacion == null) {
            System.out.println("No hay habitaciones disponibles del tipo " + tipo);
            return -1;
        }
```

```java
package org.ed06.model;

import java.time.LocalDate;
import java.util.*;

public class GestorReservas {  2 usages  cielo
    private final Map<Integer, List<Reserva>> reservasPorHabitacion = new HashMap<>();  5 usages

    public void inicializarReservas(List<Habitacion> habitaciones) {  1 usage  cielo
        for (Habitacion habitacion : habitaciones) {
            reservasPorHabitacion.put(habitacion.getNumero(), new ArrayList<>());
        }
    }

    public int reservarHabitacion(Habitacion habitacion, Cliente cliente, LocalDate entrada, LocalDate salida) {  1 usage  cielo
        if (entrada.isAfter(salida)) {
            System.out.println("La fecha de entrada es posterior a la fecha de salida");
            return -2;
        }

        if (!habitacion.isDisponible()) {
            System.out.println("La habitación no está disponible");
            return -1;
        }

        // Verificar si el cliente debe pasar a VIP
        int numReservasUltimoAnio = contarReservasEnUltimoAnio(cliente);
        if (numReservasUltimoAnio > 3 && !cliente.isVip()) {
            cliente.setVip(true);
            System.out.println("El cliente " + cliente.getNombre() + " ha pasado a ser VIP");
        }

        Reserva reserva = new Reserva(
                generarIdReserva(), habitacion, cliente, entrada, salida
        );
        reservasPorHabitacion.get(habitacion.getNumero()).add(reserva);
        habitacion.reservar();
```

```java
package org.ed06.model;

import java.util.*;

public class GestorHabitaciones {  2 usages  cielo
    private final List<Habitacion> habitaciones = new ArrayList<>();  6 usages

    public void registrarHabitaciones(List<String> tipos, List<Double> preciosBase) {  2 usages  cielo
        for (int i = 0; i < tipos.size(); i++) {
            Habitacion habitacion = new Habitacion( numero: habitaciones.size() + 1, tipos.get(i), preciosBase.get(i));
            habitaciones.add(habitacion);
        }
    }

    public void listarHabitacionesDisponibles() {  1 usage  cielo
        for (Habitacion habitacion : habitaciones) {
            if (habitacion.isDisponible()) {
                System.out.println("Habitación #" + habitacion.getNumero() + " - Tipo: " + habitacion.getTipo() +
                        " - Precio base: " + habitacion.getPrecioBase());
            }
        }
    }

    public Habitacion buscarDisponiblePorTipo(String tipo) {  1 usage  cielo
        for (Habitacion h : habitaciones) {
            if (h.getTipo().equalsIgnoreCase(tipo) && h.isDisponible()) {
                return h;
            }
        }
        return null;
    }

    public Habitacion getHabitacion(int numero) {  1 usage  cielo
        return habitaciones.stream().filter( Habitacion h -> h.getNumero() == numero).findFirst().orElse( other: null);
    }
```

```java
package org.ed06.model;

import java.util.*;

public class GestorClientes {  2 usages  cielo
    private final Map<Integer, Cliente> clientes = new HashMap<>();  5 usages

    public void registrarCliente(String nombre, String email, String dni, boolean esVip) {  3 usages  cielo
        Cliente cliente = new Cliente( id: clientes.size() + 1, nombre, dni, email, esVip);
        clientes.put(cliente.id, cliente);
    }

    public Cliente obtenerCliente(int id) {  1 usage  cielo
        return clientes.get(id);
    }

    public void listarClientes() {  1 usage  cielo
        for (Cliente cliente : clientes.values()) {
            System.out.println("Cliente #" + cliente.id + " - Nombre: " + cliente.nombre +
                    " - DNI: " + cliente.dni + " - VIP: " + cliente.esVip);
        }
    }

    public Collection<Cliente> getTodos() {  no usages  cielo
        return clientes.values();
    }
}
```

Extracción de validación de cliente

```java
import static org.ed06.model.Validacion.*;

public class Cliente {  16 usages  Daniel Martiñán *  3 related problems
    public int id;  5 usages
    public String nombre;  5 usages
    public String dni;  4 usages
    public String email;  3 usages
    public boolean esVip;  5 usages

    public Cliente(int id, String nombre, String dni, String email, boolean esVip) {  1 usage  Daniel Martiñán *
        this.id = id;
        this.esVip = esVip;

        // Validación de nombre, DNI y email
        if(validarNombre(nombre)) {this.nombre = nombre;}
        if(validarDni(dni)) {this.dni = dni;}
        if(validarEmail(email)) {this.email = email;}
    }

    // Métodos getter y setter
    public int getId() {return id;}  no usages  new *
    public void setId(int id) {this.id = id;}  no usages  new *

    public String getNombre() {return nombre;}  2 usages  new *
    public void setNombre(String nombre) {this.nombre = nombre;}  no usages  new *
```

Reducción de complejidad en sentencia if:

```java
    public int reservarHabitacion(Habitacion habitacion, Cliente cliente, LocalDate entrada, LocalDate salida) {  1 usage  cielo
        if (entrada.isAfter(salida)) {
            System.out.println("La fecha de entrada es posterior a la fecha de salida");
            return -2;
        }

        if (!habitacion.isDisponible()) {
            System.out.println("La habitación no está disponible");
            return -1;
        }

        // Verificar si el cliente debe pasar a VIP
        int numReservasUltimoAnio = contarReservasEnUltimoAnio(cliente);
        if (numReservasUltimoAnio > 3 && !cliente.isVip()) {
            cliente.setVip(true);
            System.out.println("El cliente " + cliente.getNombre() + " ha pasado a ser VIP");
        }

        Reserva reserva = new Reserva(
                generarIdReserva(), habitacion, cliente, entrada, salida
        );
        reservasPorHabitacion.get(habitacion.getNumero()).add(reserva);
        habitacion.reservar();
        System.out.println("Reserva realizada con éxito");

        return habitacion.getNumero();
    }
```