Implementación de Patrones de Diseño Creacionales, Estructurales y de Comportamiento

Integrantes
Edwin Romeo Rivas Díaz
Luis Ernesto Figueroa Vásquez
Kevin Alexander Aquino Vásquez
Luis Alexis Velázquez Godoy
Guillermo Alberto Asensio Jiménez
Moisés Roberto Hernández Hernández
Jonathan Alexander Ramírez Vázquez

Tabla de contenido

Solucion tecnica ***3***

Notification **4**

Observer interface 4

Notifier class 4

Customer **5**

Customer class 5

Reservation **6**

Reservation 6

ServiceDecorator 7

BaseReservation 7

BreakFastService 8

SpaService 8

TransportService 9

Room **9**

Factories 10

RoomType 11

Room 11

RoomFactory 12

StandardRoom 12

DoubleRoomFactory 13

PresidentialRoom 13

System **14**

ReservationSystem 14

Clase Main **15**

Ejecucion de la aplicación **18**

Creacion de un usuario 18

Agregar Habitacion 18

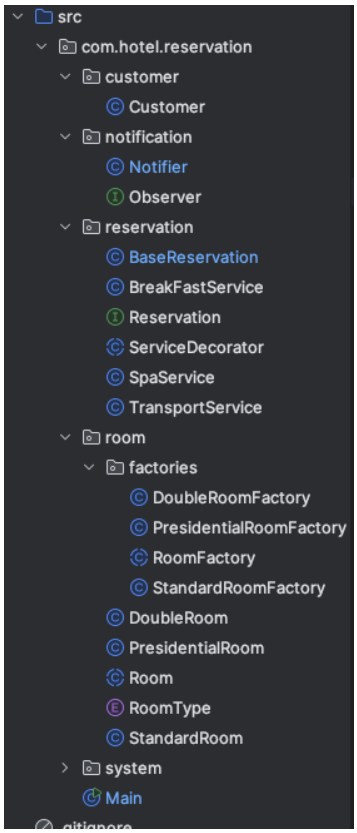
Crear reserva 19

Ver reservas 19

Solucion tecnica

Como solucion tenica a nuestro proyecto llamado '**Hotel Reservation**' se ha implementado la siguiente arquitectura, la cual consta de:

- customer
  - Customer (Clase)
- notication
  - Notifier (Clase)
  - Observer (interface)
- reservation
  - BaseReservation
  - BreakFastService
  - Reservation
  - ServiceDecorator
  - SpaService
  - TransportService
- room
  - factories
    - DoubleRoomFactory
    - PresidentialRoomFactory
    - RoomFactory
    - StandardRoomFactory
  - DoubleRoom
  - PresidentialRoom
  - Room
  - RoomType
  - StandardRoom
- system
  - ReservationSystem
- Main



Notification

Paquete creado para implementar el patron observador (observer), el cual se crearan los siguientes archivos:

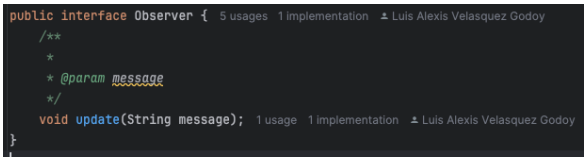
- Notifier (clase)
- Observer (interface)

Observer interface

Interfaz que contiene el método que se utilizará para notificar nuevas reservas.

Esta interface cuenta con las siguientes propiedades:

- Metodo update

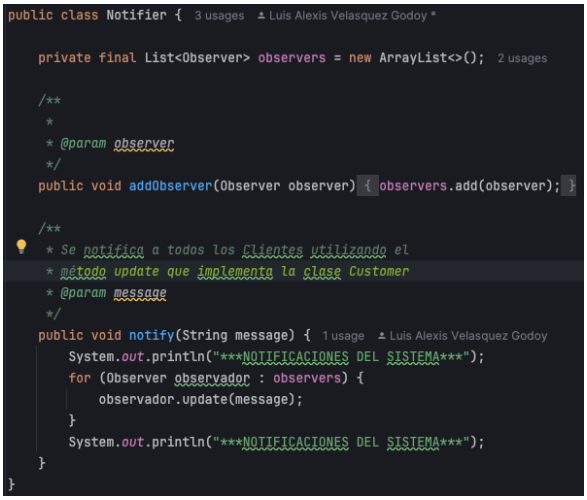


Notifier class

Clase que almacena todos los observadores (Clientes) que serán notificados al momento de crearse una reserva.

Esta clase cuenta con las siguientes propiedades:

- observers : propiedad que contiene un listado de observadores
- addObserver: metodo que recive por parametro un observador y lo agrega a la propiedad **observers**
- notify: metodo que notificara a todos los Clientes utilizando el método update que implementa la clase Customer



Customer

Paquete creado para implementar la clase customer y su funcionalidad.

Customer class

Clase que define el comportamiento y atributos de los clientes. Se implementa Observer para que los clientes puedan ser notificados siempre que hagan una reserva

Esta clase cuenta con las siguientes propiedades:

- id: propiedad de tipo int con sus metodos de acceso
- name: prpiedad de tipo String con sus metodos de acceso
- update: metodo que recive por parametro la propiedad message y retorna el valor del mensaje. Método que se utiliza Observer para enviar mensajes.

Ademas, en dicha clase se sobre escribe el metodo toString, esto según a la necesidad del desarrollo.

```java
public class Customer implements Observer {  14 usages  Luis Alexis Velasquez Godoy

    private int id;  4 usages
    private String name;  5 usages

    public Customer(int id, String name) {  1 usage  Luis Alexis Velasquez Godoy
        this.id = id;
        this.name = name;
    }

    public int getId() { return id; }

    public void setId(int id) { this.id = id; }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    /**
     * Método que se utiliza Observer para enviar mensajes
     * @param message
     */
    @Override  1 usage  Luis Alexis Velasquez Godoy
    public void update(String message) {
        System.out.println("Usuario " + this.name + ", tienes el siguiente mensaje: " + message);
    }

    @Override  Luis Alexis Velasquez Godoy
    public String toString() {
        return "Customer{" +
                "id=" + id +
                ", name='" + name + '\'' +
                '}';
    }
}
```

Reservation

Paquete creado para implementar la logicade las reservas, el cual se crearan los siguientes archivos:

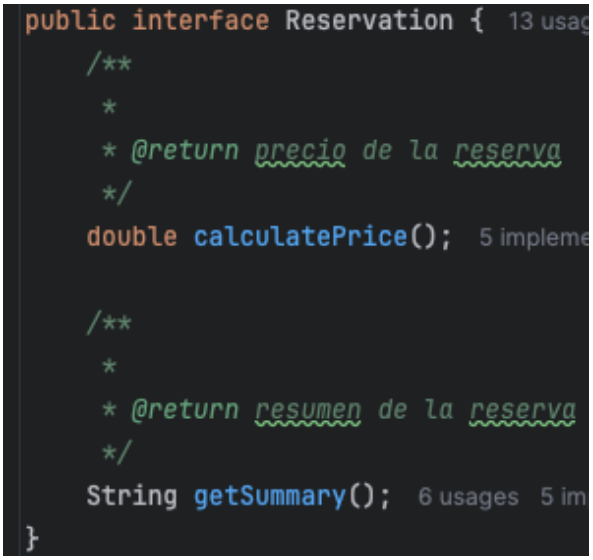
- Reservation (interface)
- ServiceDecorator (class)
- BaseReservation (class)
- BreakFastService (class)
- SpaService (class)
- TransportService (class)

Reservation

Interface que define el comportamiento básico de una reserva

Esta interface cuenta con las siguientes propiedades:

- calculatePrice: metodo que retorna un valor de tipo double
- getSummary: propiedad de tipo String

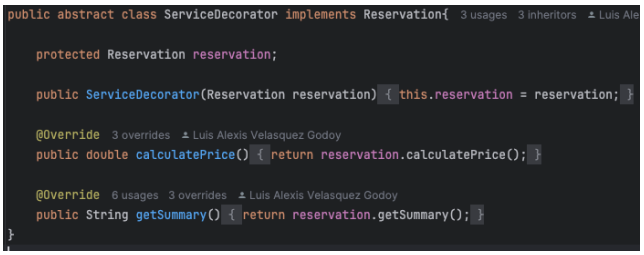


ServiceDecorator

Clase decoradora que permite agregar un servicio extra a la reserva.

Esta clase cuenta con las siguientes propiedades:

- reservation: propiedad de tipo Reservation
- calculatePrice: propiedad de la implementacion de la interface Reservation
- getSummary : propiedad de la implementacion de la interface Reservation

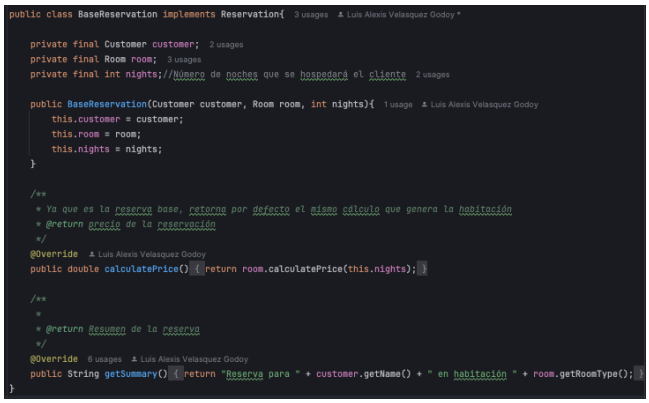


BaseReservation

Clase base de una reserva. Implementa la interfaz Reservation

Esta clase cuenta con las siguientes propiedades:

- customer: propiedad de tipo Customer
- room: propiedad de tipo Room
- nights: propiedad de tipo int
- calculatePrice: propiedad de la implementacion de la interface Reservation. Ya que es la reserva base, retorna por defecto el mismo cálculo que genera la habitación.
- getSummary : propiedad de la implementacion de la interface Reservation. Resumen de la reserva



BreakFastService

Decorador que agregar el servicio de desayuno a la reserva.

Esta clase cuenta con las siguientes propiedades:

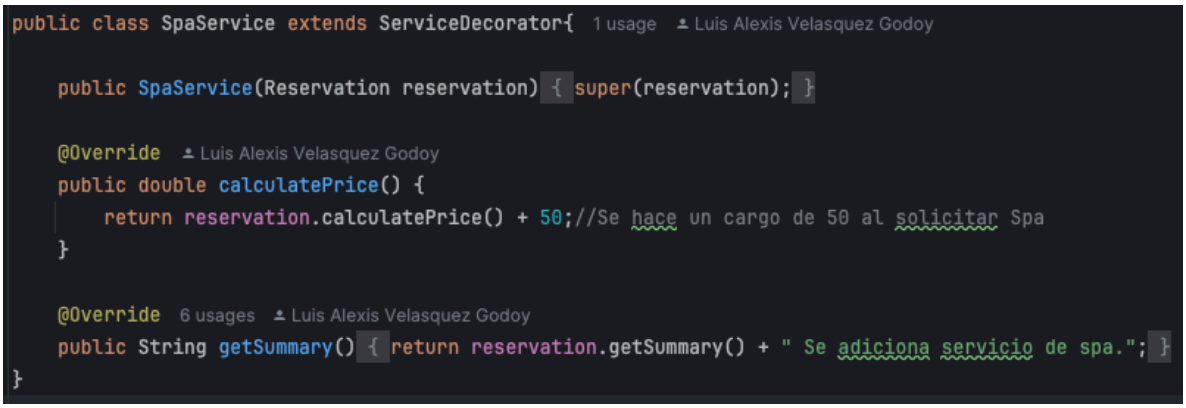
- calculatePrice: propiedad de la implementacion de la interface Reservation. Se hace un cargo de 20 al solicitar desayuno.
- getSummary : propiedad de la implementacion de la interface Reservation.

SpaService

Decorador que agregar el servicio de SPA a la reserva.

Esta clase cuenta con las siguientes propiedades:

- calculatePrice: propiedad de la implementacion de la interface Reservation. Se hace un cargo de 50 al solicitar SPA.
- getSummary : propiedad de la implementacion de la interface Reservation.



TransportService

Decorador que agregar el servicio de transporte a la reserva.

Esta clase cuenta con las siguientes propiedades:

- calculatePrice: propiedad de la implementacion de la interface Reservation. Se hace un cargo de 100 al solicitar transporte.
- getSummary : propiedad de la implementacion de la interface Reservation.

Room

Paquete creado para implementar la logicade los cuartos (room), el cual se crearan los siguientes archivos:

- factories
  - RoomFactory (class)
  - DoubleRoomFactory (class)
  - PresidentialRoomFactory (class)
  - StandardRoomFactory (class)
- DoubleRoom (class)
- PresidentialRoom (class)
- Room (class)
- RoomType (enum)
- StandardRoom (class)

Factories

RoomFactory

Clase abstracta que define el comportamiento de las factories de habitaciones.

Esta clase abstracta cuenta con las siguientes propiedades:

- Create: Retorna un objeto de tipo Room o que extienda de la clase Room (StandardRoom, DoubleRoom, PresidentialRoom)

```
public abstract class RoomFactory {  3 usages  3 inheritors  Luis Alexis Ve
    /**
     *
     * @return un objeto de tipo Room o que extienda de
     * la clase Room (StandardRoom, DoubleRoom, PresidentialRoom)
     */
    public abstract Room create();  3 usages  3 implementations  Luis Ale
}
```

DoubleRoomFactory

Factoría que retorna objetos de tipo DoubleRoom

```
public class DoubleRoomFactory extends RoomFactory{  2 usages  Luis Alex
    @Override  3 usages  Luis Alexis Velasquez Godoy
    public Room create() { return new DoubleRoom(RoomType.DOUBLE); }
}
```

PresidentialRoomFactory

Factoría que retorna objetos de tipo PresidentialRoom

```
public class PresidentialRoomFactory extends RoomFactory{  2 usages  Luis Alexis Velasc
    @Override  3 usages  Luis Alexis Velasquez Godoy
    public Room create() { return new PresidentialRoom(RoomType.PRESIDENTIAL); }
}
```

StandardRoomFactory

Factoría que retorna objetos de tipo StandardRoom

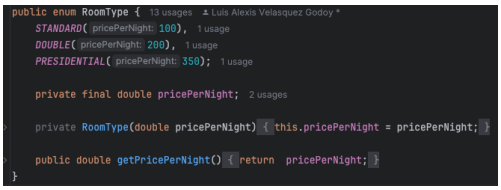
```
public class StandardRoomFactory extends RoomFactory{  2 usages  Luis Alexis
    @Override  3 usages  Luis Alexis Velasquez Godoy
    public Room create() { return new StandardRoom(RoomType.STANDARD); }
}
```

RoomType

Enum que define el tipo de habitación que ofrece el hotel y costo por noche.

Este enum cuenta con las siguientes propiedades:

- pricePerNight: propiedad de tipo double con sus metodo de acceso (get)



```
public enum RoomType {  13 usages  Luis Alexis Velasquez Godoy *
    STANDARD( pricePerNight: 100),  1 usage
    DOUBLE( pricePerNight: 200),  1 usage
    PRESIDENTIAL( pricePerNight: 350);  1 usage

    private final double pricePerNight;  2 usages

    private RoomType(double pricePerNight) { this.pricePerNight = pricePerNight; }

    public double getPricePerNight() { return  pricePerNight; }
}
```

Room

Clase abstracta que define el comportamiento y atributos de una habitación.

Esta clase cuenta con las siguientes propiedades:

- roomType: propiedad de tipo RoomType con sus metodo de acceso (get)
- isAvailable: propiedad de tipo boolean con sus metodo de acceso (get)
- calculatePrice: propiedad que recive por parametro la propiedad nights de tipo int, y retorna un double.
- toString: Sobre escritura del metodo troString

```
public abstract class Room {  3 inheritors  Luis Alexis Velasquez Godoy *
    protected RoomType roomType;//Tipo de habitación definida por enum RunTime  6 usages
    protected boolean isAvailable;//Determina si la habitación está ocupada o libre  4

    public Room(RoomType type) {  Luis Alexis Velasquez Godoy
        this.roomType = type;
        this.isAvailable = true;
    }

    /**
     * Método que calcula el precio de la habitación en base a las noches
     * que se hospedará el cliente
     * @param nights
     * @return
     */
    public abstract double calculatePrice(int nights);  3 implementations  Luis Alexis Velasque

    /**
     *
     * @return si está libre(true) o ocupada(false
     */
    public boolean isAvailable() { return isAvailable; }

    public RoomType getRoomType() { return roomType; }

    /**
     * Setea el estado de la habitación Libre(true)/Ocupada(false)
     * @param isAvailable
     */
    public void setAvailable(boolean isAvailable) { this.isAvailable = isAvailable; }

    @Override  Luis Alexis Velasquez Godoy
    public String toString() {
        return "Room{" +
                "roomType=" + roomType +
                ", isAvailable=" + isAvailable +
                '}';
    }
}
```

RoomFactory

Clase abstracta que define el comportamiento de las factories de habitaciones.

Esta clase cuenta con las siguientes propiedades:

- create: metodo que retorna una propiedad de tipo Room. Retorna un objeto de tipo Room o que extienda de la clase Room (StandardRoom, DoubleRoom, PresidentialRoom)

```
public abstract class RoomFactory {  3 usages  3 inheritors  Luis Alexis Vel
    /**
     *
     * @return un objeto de tipo Room o que extienda de
     * la clase Room (StandardRoom, DoubleRoom, PresidentialRoom)
     */
    public abstract Room create();  3 usages  3 implementations  Luis Alex
}
```

StandardRoom

Clase que determina el comportamiento de una habitación tipo Standard. Se extiende de la clase Room

Esta clase cuenta con las siguientes propiedades:

- calculatePrice: metodo que recive el parametro nights de tipo int y retorna el precio de total de las noches de reserva por el precio de la habitacion.

```
public class StandardRoom extends Room{  2 usages  Luis Alexis Velasquez Godoy

    public StandardRoom(RoomType roomType) { super(roomType); }

    /**
     *
     * @param nights
     * @return
     */
    @Override  Luis Alexis Velasquez Godoy
    public double calculatePrice(int nights) { return roomType.getPricePerNight() * nights; }
}
```

DoubleRoomFactory

Clase que determina el comportamiento de una habitación tipo doble. Se extiende de la clase Room

Esta clase cuenta con las siguientes propiedades:

- calculatePrice: metodo que recive el parametro nights de tipo int y retorna el precio de total de las noches de reserva por el precio de la habitacion. Si se quede mas de 3 dias se le dara un descuento del 10%

```
public class DoubleRoom extends Room{  2 usages  Luis Alexis Velasquez Godoy *

    public DoubleRoom(RoomType roomType) { super(roomType); }

    @Override  Luis Alexis Velasquez Godoy *
    public double calculatePrice(int nights) {
        //Descuento del 10% si se queda más de 3 días
        return roomType.getPricePerNight() * nights * (nights > 3 ? 0.9: 1);
    }
}
```

PresidentialRoom

Clase que determina el comportamiento de una habitación tipo presidencial. Se extiende de la clase Room

Esta clase cuenta con las siguientes propiedades:

- calculatePrice: metodo que recive el parametro nights de tipo int y retorna el precio de total de las noches de reserva por el precio de la habitacion. Si se quede mas de 5 dias se le dara un descuento del 25%

```
public class PresidentialRoom extends Room{  2 usages  Luis Alexis Velasquez Godoy *

    public PresidentialRoom(RoomType roomType) { super(roomType); }

    @Override  Luis Alexis Velasquez Godoy *
    public double calculatePrice(int nights) {
        //Descuento del 25% si se queda más de 5 días
        return roomType.getPricePerNight() * nights * (nights > 5 ? 0.75:1);
    }
}
```

System

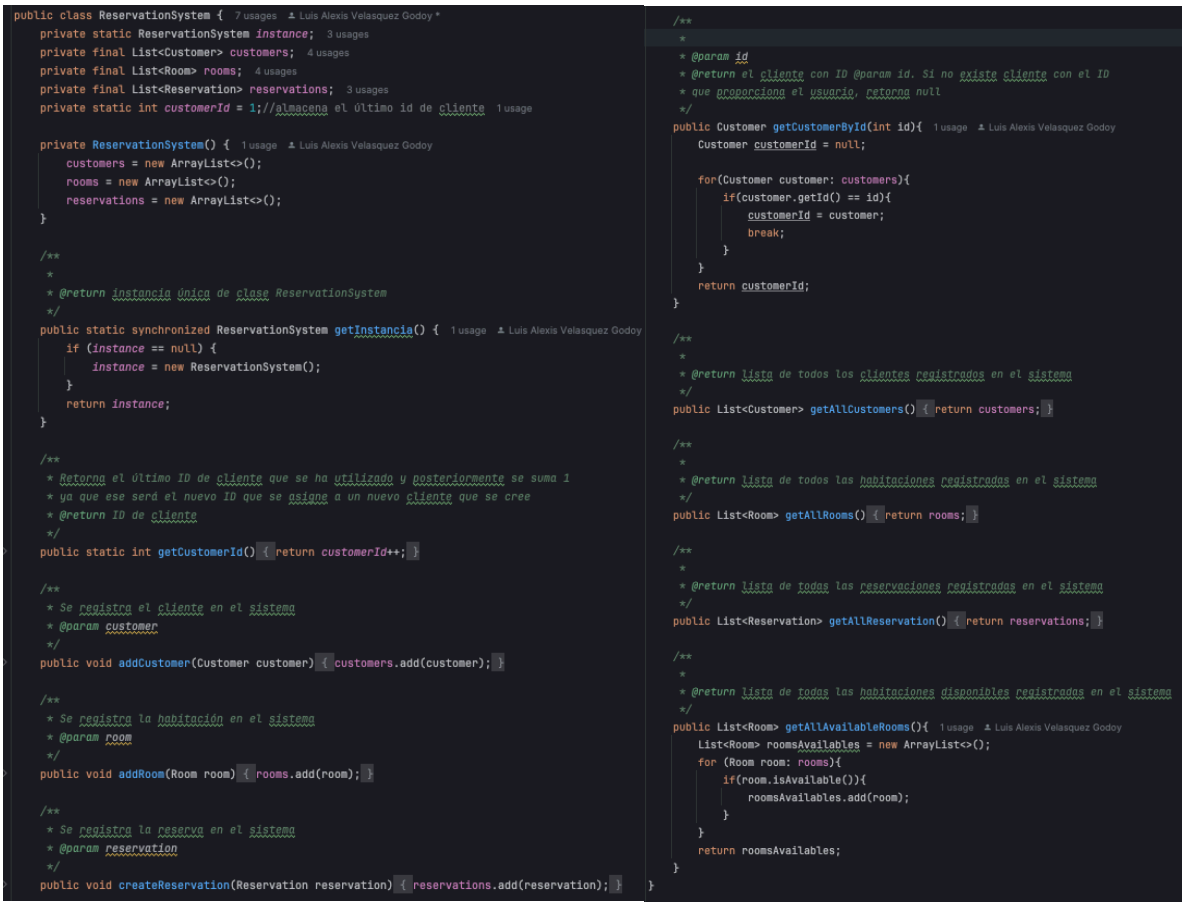
Paquete creado para implementar la logica del almacenamiento de las reservas. En dicho paquete se implementa el patron singleton. Ademas, todo esta implementado en la clase ReservationSystem

ReservationSystem

Clase general que determina el comportameinto del sistema. Almacena todos los clientes, Cuartos y Reservas. Se utiliza el patrón de diseño Singleton para tener una única instancia de esta clase general.

Esta clase cuenta con las siguientes propiedades:

- instance: propiedad que almacena la instancia de la clase
- customers: propiedad que almacena el listado clientes
- rooms: propiedad que almacena el listado de cuartos
- reservations: propiedad que almacena el listado de reservas
- customerId: lmacena el último id de cliente
- getInstancia: retorna una instancia única de clase ReservationSystem.
- getCustomerId: Retorna el último ID de cliente que se ha utilizado y posteriormente se suma 1 ya que ese será el nuevo ID que se asigne a un nuevo cliente que se cree.
- addCustomer: Se registra el cliente en el sistema.
- addRoom: Se registra la habitación en el sistema
- createReservation: Se registra la reserva en el sistema
- getCustomerById: retorna el cliente con ID @param id. Si no existe cliente con el ID que proporciona el usuario, retorna null.
- getAllCustomers: retorna una lista de todos los clientes registrados en el sistema.
- getAllRooms: retorna una lista de todos las habitaciones registradas en el sistema.
- getAllReservation: retorna una lista de todas las reservaciones registradas en el sistema.
- getAllAvailableRooms: retorna una lista de todas las habitaciones disponibles registradas en el sistema.



```java
public class ReservationSystem {  7 usages  Luis Alexis Velasquez Godoy *
    private static ReservationSystem instance;  3 usages
    private final List<Customer> customers;  4 usages
    private final List<Room> rooms;  4 usages
    private final List<Reservation> reservations;  3 usages
    private static int customerId = 1;//almacena el último id de cliente  1 usage

    private ReservationSystem() {  1 usage  Luis Alexis Velasquez Godoy
        customers = new ArrayList<>();
        rooms = new ArrayList<>();
        reservations = new ArrayList<>();
    }

    /**
     *
     * @return instancia única de clase ReservationSystem
     */
    public static synchronized ReservationSystem getInstancia() {  1 usage  Luis Alexis Velasquez Godoy
        if (instance == null) {
            instance = new ReservationSystem();
        }
        return instance;
    }

    /**
     * Retorna el último ID de cliente que se ha utilizado y posteriormente se suma 1
     * ya que ese será el nuevo ID que se asigne a un nuevo cliente que se cree
     * @return ID de cliente
     */
    public static int getCustomerId() { return customerId++; }

    /**
     * Se registra el cliente en el sistema
     * @param customer
     */
    public void addCustomer(Customer customer) { customers.add(customer); }

    /**
     * Se registra la habitación en el sistema
     * @param room
     */
    public void addRoom(Room room) { rooms.add(room); }

    /**
     * Se registra la reserva en el sistema
     * @param reservation
     */
    public void createReservation(Reservation reservation) { reservations.add(reservation); }
```

```java
    /**
     *
     * @param id
     * @return el cliente con ID @param id. Si no existe cliente con el ID
     * que proporciona el usuario, retorna null
     */
    public Customer getCustomerById(int id){  1 usage  Luis Alexis Velasquez Godoy
        Customer customerId = null;

        for(Customer customer: customers){
            if(customer.getId() == id){
                customerId = customer;
                break;
            }
        }
        return customerId;
    }

    /**
     *
     * @return lista de todos los clientes registrados en el sistema
     */
    public List<Customer> getAllCustomers() { return customers; }

    /**
     *
     * @return lista de todos las habitaciones registradas en el sistema
     */
    public List<Room> getAllRooms() { return rooms; }

    /**
     *
     * @return lista de todas las reservaciones registradas en el sistema
     */
    public List<Reservation> getAllReservation() { return reservations; }

    /**
     *
     * @return lista de todas las habitaciones disponibles registradas en el sistema
     */
    public List<Room> getAllAvailableRooms(){  1 usage  Luis Alexis Velasquez Godoy
        List<Room> roomsAvailables = new ArrayList<>();
        for (Room room: rooms){
            if(room.isAvailable()){
                roomsAvailables.add(room);
            }
        }
        return roomsAvailables;
    }
}
```

Clase Main

La clase main contiene la funcionalidad de recibir los valores al usuario que esta corriendo el aplicativo.

Como punto de partida, se obtiene la instancia de la clase Notifier, luego se muestra el menu con las siguientes opciones:

1. Registrar Cliente
    a. Se ingresaran los datos del cliente
2. Agregar Habitación
    a. Se solicitara que ingrese el tipo de habitacion
3. Crear Reserva
    a. Se solicitara el id del cliente para anexar la reserva.
    b. Se verifica las habitaciones disponibles
    c. Se muestra el listado de habitaciones disponibles
    d. Se selecciona una habitacion
    e. Se solicita el numero de noches que se requieren para la reserva

    f. Se le muestra el menu de opciones adicionales a la habitacion
    g. Se ingresa la reserva
    h. Se notifica la creacion de la reserva
4. Ver Reservas
    a. Se muestra el listado de las reservas creadas
5. Salir
    a. Finaliza la ejecucion de la aplicacion

```java
package com.hotel.reservation;

// Importación de clases necesarias del proyecto
import com.hotel.reservation.customer.Customer;
import com.hotel.reservation.notification.Notifier;
import com.hotel.reservation.reservation.*;
import com.hotel.reservation.room.Room;
import com.hotel.reservation.room.factories.DoubleRoomFactory;
import com.hotel.reservation.room.factories.PresidentialRoomFactory;
import com.hotel.reservation.room.factories.StandardRoomFactory;
import com.hotel.reservation.system.ReservationSystem;

import java.util.Scanner;

public class Main {  Luis Alexis Velasquez Godoy *
    public static void main(String[] args) {  Luis Alexis Velasquez Godoy *
        // Obtiene la instancia única del sistema de reservas (Singleton)
        ReservationSystem system = ReservationSystem.getInstancia();

        // Instancia de la clase Notificable para manejar observadores
        Notifier notificable = new Notifier();

        // Scanner para capturar la entrada del usuario
        Scanner scanner = new Scanner(System.in);

        // Variable de control para salir del bucle principal
        boolean exit = false;

        // Bucle principal del menú
        while (!exit) {
            // Despliega las opciones del sistema
            System.out.println("\n--- Sistema de Reservas ---");
            System.out.println("1. Registrar Cliente");
            System.out.println("2. Agregar Habitación");
            System.out.println("3. Crear Reserva");
            System.out.println("4. Ver Reservas");
            System.out.println("5. Salir");
            System.out.print("Seleccione una opción: ");

            // Captura la opción ingresada por el usuario
            int opcion = scanner.nextInt();
            scanner.nextLine(); // Limpia el buffer del Scanner
```

```java
try {
    outerLoop: // Etiqueta para manejar la salida del flujo
    switch (opcion) {
        case 1 -> { // Opción para registrar un cliente
            System.out.print("Ingrese el nombre del cliente: ");
            String name = scanner.nextLine();
            // Crea un nuevo cliente y lo agrega al sistema
            Customer customer = new Customer(ReservationSystem.getCustomerId(), name);
            system.addCustomer(customer);
            notificable.addObserver(customer); // Registra al cliente como observador
            System.out.println("Cliente registrado exitosamente.");
            System.out.println("Detalle del cliente: " + customer.toString());
        }
        case 2 -> { // Opción para agregar una habitación
            System.out.print("Ingrese el tipo de Habitación (Standard/Double/Presidential): ");
            String roomType = scanner.nextLine().toUpperCase();

            // Valida que el tipo de habitación sea válido
            if (
                    !roomType.equals("STANDARD") &&
                            !roomType.equals("DOUBLE") &&
                            !roomType.equals("PRESIDENTIAL")
            ) {
                System.out.println("El tipo de habitación ingresado no existe");
            } else {
                // Crea la habitación según el tipo especificado
                Room room;
                if (roomType.equals("STANDARD")) {
                    room = new StandardRoomFactory().create();
                } else if (roomType.equals("DOUBLE")) {
                    room = new DoubleRoomFactory().create();
                } else {
                    room = new PresidentialRoomFactory().create();
                }

                // Agrega la habitación al sistema
                system.addRoom(room);
                System.out.println("Habitación agregada exitosamente.");
                System.out.println("Detalle de la habitación: " + room.toString());
            }

        }
```

```java
case 3 -> { // Opción para crear una reserva
    // Verifica si hay clientes registrados
    int customerCounts = system.getAllCustomers().size();

    if (customerCounts == 0) {
        System.out.println("No hay clientes registrados en el sistema.");
        break;
    }

    System.out.print("Ingrese el ID del cliente: ");
    int customerId = scanner.nextInt();
    scanner.nextLine();
    // Busca al cliente por su ID
    Customer customerById = system.getCustomerById(customerId);

    if (customerById == null) {
        System.out.println("Id del usuario no existe.");
    } else {
        // Verifica si hay habitaciones disponibles
        int availablesRoomsCount = system.getAllAvailableRooms().size();

        if (availablesRoomsCount == 0) {
            System.out.println("No hay habitaciones disponibles. Inténtalo más tarde");
            break;
        }

        // Lista las habitaciones disponibles
        System.out.println("Habitaciones disponibles:");
        for (int i = 0; i < system.getAllRooms().size(); i++) {
            Room room = system.getAllRooms().get(i);

            System.out.println(i + 1 + ". " + room.getClass().getSimpleName() + " - " + (room.isAvailable()

        }

        // Selecciona una habitación disponible
        System.out.print("Selecciona el número de la habitación: ");
        int roomNumber = scanner.nextInt() - 1;
        scanner.nextLine();
        Room room = system.getAllRooms().get(roomNumber);
```

```java
    // Valida que la habitación está disponible
    while (!room.isAvailable()) {
        System.out.println("La habitación está ocupada. Elije otra habitación: ");
        roomNumber = scanner.nextInt() - 1;
        scanner.nextLine();
        room = system.getAllRooms().get(roomNumber);
    }

    System.out.print("Ingrese el número de noches: ");
    int nights = scanner.nextInt();

    // Crea la reserva base
    BaseReservation reservaBase = new BaseReservation(customerById, room, nights);

    // Menú para servicios adicionales
    System.out.println("----------------------------------");
    System.out.println("¿Desea agregar alguno de estos servicios?");
    System.out.println("1. Desayuno");
    System.out.println("2. Transporte");
    System.out.println("3. Spa");
    System.out.println("4. No desea agregar servicios extras");

    int respuesta = scanner.nextInt();
    scanner.nextLine();

    // Valida la opción ingresada
    while (respuesta < 1 || respuesta > 4) {
        System.out.println("Opción no válida. Ingresa otra opción");
        respuesta = scanner.nextInt();
        scanner.nextLine();
    }

    // Agrega servicios adicionales según la opción seleccionada
    Reservation reservation;

    if (respuesta == 1) {
        reservation = new BreakFastService(reservaBase);
    } else if (respuesta == 2) {
        reservation = new TransportService(reservaBase);
    } else if (respuesta == 3) {
        reservation = new SpaService(reservaBase);
    } else {
        reservation = reservaBase;
    }
```

```
                        // Crea la reserva en el sistema
                        system.createReservation(reservation);
                        room.setAvailable(false); // Marca la habitación como ocupada


                        // Notifica la creación de la reserva
                        notificable.notify("Se crea una reservación a nombre de " + customerById.getName() + " en un
                        System.out.println("Reserva creada exitosamente: " + reservation.getSummary() + " | Total: $

                    }

                }
                case 4 -> { // Opción para ver todas las reservas
                    System.out.println("Reservas realizadas:");
                    for (Reservation reservation : system.getAllReservation()) {
                        System.out.println(reservation.getSummary() + " | Total: $" + reservation.calculatePrice());
                    }
                }
                case 5 -> { // Opción para salir del sistema
                    exit = true;
                    System.out.println("Saliendo del sistema...");
                }
                default -> System.out.println("Opción no válida. Intente de nuevo.");
            }
        } catch (Exception e) {
            // Captura y muestra cualquier error que ocurra
            System.err.println("Error: " + e.getMessage());
        }
    }

    // Cierra el scanner al finalizar
    scanner.close();
  }
}
```

Ejecucion de la aplicación

Creacion de un usuario

Al usuario dentro del aplicativo, se le solicitara registrar a un nuevo cliente, dicha opcion es la numero uno, se le solicitara el nombre, y si todo esta correctamente, se ingresara el registro, y se podra observar el identificador del cliente, que en este caso es 1.

```
--- Sistema de Reservas ---
1. Registrar Cliente
2. Agregar Habitación
3. Crear Reserva
4. Ver Reservas
5. Salir
Seleccione una opción: 1
Ingrese el nombre del cliente: kevin aquino
Cliente registrado exitosamente.
Detalle del cliente: Customer{id=1, name='kevin aquino'}
```

Agregar Habitacion

Al usuario dentro del aplicativo, se le solicitara registrar a una nueva habitacion, dicha opcion es la numero dos, se le solicitara el nombre, dicho nombre debe de ser una de las opciones que aparece en el enunciado (Standard/Double/Presidential), y si todo esta correctamente, se ingresara el registro.

```
--- Sistema de Reservas ---
1. Registrar Cliente
2. Agregar Habitación
3. Crear Reserva
4. Ver Reservas
5. Salir
Seleccione una opción: 2
Ingrese el tipo de Habitación (Standard/Double/Presidential): Presidential
Habitación agregada exitosamente.
Detalle de la habitación: Room{roomType=PRESIDENTIAL, isAvailable=true}
```

Crear reserva

Al usuario dentro del aplicativo, se le solicitara registrar a una nueva reseva, dicha opcion es la numero tres, se le solicitara el id del cliente, en este caso es 1, luego se le solicitara la habitacion, el numero de noches de la reserva, y por ultimo si tiene algun extra para agregar a la reserva, y si todo esta correctamente, se ingresara el registro, y se podra observar el resumen de la reserva.

```
--- Sistema de Reservas ---
1. Registrar Cliente
2. Agregar Habitación
3. Crear Reserva
4. Ver Reservas
5. Salir
Seleccione una opción: 3
Ingrese el ID del cliente: 1
Habitaciones disponibles:
1. PresidentialRoom - Disponible
Seleccione el número de la habitación: 1
Ingrese el número de noches: 10
---------------------------------
¿Desea agregar alguno de estos servicios?
1. Desayuno
2. Transporte
3. Spa
4. No deseo agregar servicios extras
1
***NOTIFICACIONES DEL SISTEMA***
Usuario kevin aquino, tienes el siguiente mensaje: Se crea una reservación a nombre de kevin aquino en una habitación tipo PRESIDENTIAL
***NOTIFICACIONES DEL SISTEMA***
Reserva creada exitosamente: Reserva para kevin aquino en habitación PRESIDENTIAL Se adiciona servicio de desayuno. | Total: $2645.0
```

Ver reservas

Al usuario dentro del aplicativo, podra observar el listado de reservas, el cual podra ver el detalle de cada reserva que hay.

```
--- Sistema de Reservas ---
1. Registrar Cliente
2. Agregar Habitación
3. Crear Reserva
4. Ver Reservas
5. Salir
Seleccione una opción:
4
Reservas realizadas:
Reserva para kevin aquino en habitación PRESIDENTIAL Se adiciona servicio de desayuno. | Total: $2645.0
```