UNIVERSIDAD EUROPEA DE MADRID

Escuela de arquitectura, ingeniería y diseño Doble Grado en ingeniería informática e ingeniería matemática aplicada al análisis de datos

Proyecto Final

JHOSUA CALLEJAS RAMOS

CURSO 2022-2023

TÍTULO: Jhosua Resort

AUTOR: Jhosua Callejas Ramos

ASIGNATURA: Programación orientada a objetos

DIRECTOR DEL PROYECTO: Antonio Barba Salvador

FECHA: mayo de 2023

Índice

1. DESARROLLO

La gestión de un hotel puede ser una tarea desafiante, especialmente cuando se trata de manejar reservas, clientes y habitaciones. Con el objetivo de facilitar este proceso y hacerlo más eficiente, se ha desarrollado un programa en Java para gestionar un hotel.

Este programa ofrece varias funcionalidades, desde la gestión de reservas y habitaciones hasta la gestión de clientes y consultas sobre habitaciones disponibles en un período de tiempo determinado. Al automatizar estos procesos, se espera que el programa ayude a los gerentes y empleados del hotel a ahorrar tiempo y esfuerzo en la gestión diaria del hotel.

Los requisitos son usar los conceptos de encapsulación, polimorfismo y herencia vistos en clase he creado una clase abstracta que la he llamada habitaciones donde sus atributos como numeroHabitacion, metrosCuadrados que definirán el tamaño de la habitación y precio son los atributos que heredaran las subclases como habitacionIndividual, habitacionDoble y habitacionSuite, tambien he creado las funciones get por que como son atributos privados poder importarlas a las otras clases aquí muestro el código de la clase abstracta.

1.1 Clase Habitaciones

Este código define la clase abstracta **Habitaciones**, que es la clase base para todas las clases de habitaciones del hotel. La clase tiene propiedades comunes a todas las habitaciones, como el número de habitación, los metros cuadrados y el precio.

También hay métodos para obtener y establecer estas propiedades, así como métodos estáticos para comprobar si una habitación existe en un hotel y borrar una habitación existente del hotel.

La clase es abstracta, lo que significa que no se puede crear una instancia directa de ella. En cambio, se utiliza como una clase base para definir clases de habitaciones concretas, como **HabitacionDoble**, **HabitacionSuite** y **HabitacionIndividual**.

```
public abstract class Habitaciones {
    private int numeroHabitacion;
    private double metrosCuadrados;
    private double precio;
    public Habitaciones(int numeroHabitacion, double precio, double
metrosCuadrados) {
        this.numeroHabitacion = numeroHabitacion;
        this.precio = precio;
        this.metrosCuadrados = metrosCuadrados;
    }
    public int getNumeroHabitacion() {

        return numeroHabitacion;
    }
    public double getMetrosCuadrados() {
        return metrosCuadrados;
    }
```

```
    public double getPrecio() {
        return precio;
    }

    public void setNumeroHabitacion(int numeroHabitacion) {

        this. numeroHabitacion = numeroHabitacion;
    }

    public void setMetrosCuadrados(double metrosCuadrados) {
        this.metrosCuadrados = metrosCuadrados;
    }

    public void setPrecio(double precio) {
        this.precio = precio;
    }
```

1.1.1 Habitacionindividual

El código define una clase llamada HabitacionIndividual, que es una subclase de la clase "Habitaciones". La clase tiene un constructor que toma tres argumentos (el número de la habitación, los metros cuadrados y el precio) y llama al constructor de la superclase con estos argumentos.

```
public class HabitacionIndividual extends Habitaciones{
    public HabitacionIndividual(int numeroHabitacion, double
metrosCuadrados, double precio) {
        super(numeroHabitacion, metrosCuadrados, precio);

    }
    @Override
    public String toString() {
        return "Informacion sobre la habitacion individual: numero de
habitacion " + getNumeroHabitacion() + ", el tamaño es " +
getMetrosCuadrados() +
                " metros cuadrados, precio: " + getPrecio() + " euros
";
    }
}
```

1.1.2 HabitacionDoble

Este código define una subclase de **Habitaciones** llamada **HabitacionDoble**. La clase **HabitacionDoble** tiene un atributo adicional **estiloCama**, que especifica el estilo de la cama en la habitación doble. El constructor de la clase inicializa tanto los atributos heredados de la clase padre **Habitaciones** como el atributo **estiloCama**. Los métodos **getEstiloCama()** y **setEstiloCama()** se utilizan para acceder y modificar el atributo **estiloCama**.

```
public class HabitacionDoble extends Habitaciones{
    private String estiloCama;

    public HabitacionDoble(int numeroHabitacion, double
metrosCuadrados, double precio, String estiloCama) {
```

```
        super(numeroHabitacion, metrosCuadrados, precio);
        this.estiloCama = estiloCama;
    }

    public String getEstiloCama() {
        return estiloCama;
    }
    public void setEstiloCama(String estiloCama) {
        this.estiloCama = estiloCama;
    }

    @Override
    public String toString() {
        return "Indormacion sobre la habitacion doble: numero de
habitacion " + getNumeroHabitacion() + ", el tamaño es " +
getMetrosCuadrados() +
                " metros cuadrados, la cama es " + this.estiloCama +
", y el precio de la habitacion es " + getPrecio() + " euros ";
    }
}
```

1.1.3 HabitacionSuite

La clase **HabitacionSuite** es una subclase de la clase **Habitaciones**, lo que significa que hereda sus atributos y métodos. Además, la clase **HabitacionSuite** tiene un atributo adicional llamado **metrosCuadradosDormitorio**, que representa los metros cuadrados del dormitorio dentro de la suite.

El constructor de **HabitacionSuite** toma cuatro parámetros: **numeroHabitacion**, **precio**, **metrosCuadradosDormitorio** y **metrosCuadrados**. El constructor llama al constructor de la superclase (**Habitaciones**) pasando **numeroHabitacion**, **precio** y **metrosCuadrados**. A continuación, inicializa el atributo **metrosCuadradosDormitorio** con el valor de **metrosCuadradosDormitorio**.

La clase **HabitacionSuite** también proporciona un método **getMetrosCuadradosDormitorio()** y un método **setMetrosCuadradosDormitorio()**, que permiten obtener y establecer el valor del atributo **metrosCuadradosDormitorio**, respectivamente.

```
public class HabitacionSuite extends Habitaciones {
    private double metrosCuadradosDormitorio;

    public HabitacionSuite(int numeroHabitacion, double precio, double
metrosCuadradosDormitorio, double metrosCuadrados) {
        super(numeroHabitacion, metrosCuadrados, precio);
        this.metrosCuadradosDormitorio = metrosCuadradosDormitorio;
    }

    public double getMetrosCuadradosDormitorio() {
        return metrosCuadradosDormitorio;
    }

    public void setMetrosCuadradosDormitorio(double
metrosCuadradosDormitorio) {
        this.metrosCuadradosDormitorio = metrosCuadradosDormitorio;
    }
```

```
    @Override
    public String toString() {
        return "Informacion de la suite: numero de habitacion " +
getNumeroHabitacion() + ", metros cuadrados del dormitorio: " +
                this.metrosCuadradosDormitorio + ", metros cuadrados
de la sala: " + getMetrosCuadrados() +
                ", el precio es " + getPrecio() + " euros ";
    }
}
```

1.2 Clase Clientes

Defino la clase **Clientes** que representa a un cliente en un hotel. Los atributos de la clase son **nombre**, **apellidos** e **ID**, que se inicializan mediante un constructor en el momento de la creación del objeto.

La clase también define los métodos **get** y **set** para cada uno de los atributos, lo que permite acceder y modificar estos valores desde otras partes del programa.

Además, la clase incluye dos métodos estáticos **clientesExiste** y **borrarCliente**. El método **clientesExiste** recibe como parámetros una instancia de la clase **Hotel** y un ID de cliente, y devuelve **true** si el cliente existe en la lista de clientes del hotel, y **false** en caso contrario.

Por otro lado, el método **borrarCliente** recibe como parámetros una instancia de la clase **Hotel** y un ID de cliente, y elimina al primer cliente que encuentre en la lista de clientes del hotel con el ID especificado.

```
public class Clientes {
    private String nombre;
    private String apellidos;
    private String ID;
    public Clientes(String nombre, String apellidos, String ID) {
        this.nombre=nombre;
        this.apellidos=apellidos;
        this.ID=ID;
    }
    public String getNombre() {
        return nombre;
    }
    public String getApellidos() {
        return apellidos;
    }
    public String getID() {
        return ID;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
    }
    public void setID(String ID) {
        this.ID = ID;
    }
```

```
    //Compruebo si el cliente ya existe
    static boolean clientesExiste (Hotel hotel, String ID) {
        boolean existe= false;
        for (Clientes clientes : hotel.getClientes()) {
            if(clientes.getID().equals(ID)) {
                existe=true;
                break;
            }
        }
        return existe;
    }
    //Borrar cliente
    static void borrarCliente (Hotel hotel, String ID) {
        for(Clientes clientes : hotel.getClientes()) {
            hotel.borrarCliente(clientes);
            break;
        }
    }
    public String toStrign() {
        return "Detalles de clientes: " + this.apellidos + "," +
this.nombre + ", ID:" + this.ID;
    }
}
```

1.3 Clase Reservas

Esta clase tiene varios atributos como el número de reserva, la fecha de entrada, la fecha de salida, una lista de clientes y una lista de habitaciones. También tiene varios métodos que realizan diferentes funciones como agregar un cliente, agregar una habitación, verificar la disponibilidad de habitaciones, verificar la disponibilidad de fechas y buscar habitaciones disponibles en un período de tiempo determinado.

Algunos de los métodos incluidos son:

"ultimaReserva": Este método toma un objeto "Hotel" como parámetro y devuelve el número de la última reserva realizada en ese hotel.

"analizarFecha": Este método toma una cadena de fecha en formato "dd-mm-YYYY" como parámetro y devuelve un objeto "LocalDate" que representa esa fecha.

"habitacionDisponible": Este método toma dos objetos "LocalDate" que representan la fecha de entrada y salida de una nueva reserva y devuelve "true" si la habitación está disponible en ese período de tiempo.

"fechaDisponible": Este método toma dos objetos "LocalDate" que representan un período de tiempo y dos objetos "LocalDate" que representan la fecha de entrada y salida de una reserva existente y devuelve "true" si la fecha de la reserva existe dentro del período de tiempo.

"periodoHabitacion": Este método toma un objeto "Hotel" y dos objetos "LocalDate" que representan un período de tiempo como parámetros. Este método busca en todas las habitaciones del hotel y muestra las habitaciones disponibles en ese período de tiempo. También filtra por tipo de habitación, ya sea habitación individual, habitación doble o cualquier otro tipo de habitación.

"borrarReserva": Este método toma un objeto "Hotel" y un número de reserva como parámetros y elimina la reserva correspondiente de la lista de reservas del hotel.

```
2. import java.time.LocalDate;
   import java.time.format.DateTimeParseException;
   import java.util.ArrayList;
   public class Reservas {
       private int numeroReserva;
       private LocalDate entrada;
       private LocalDate salida;
       private ArrayList<Clientes>clientes;
       private ArrayList<Habitaciones> habitaciones;

       public Reservas(int numeroReserva, LocalDate entrada,
   LocalDate salida) {
           this.numeroReserva=numeroReserva;
           this.entrada=entrada;
           this.salida=salida;
           this.clientes= new ArrayList<>();
           this.habitaciones= new ArrayList<>();
       }
       public int getNumeroReserva() {
           return numeroReserva;
       }
       public LocalDate getEntrada() {
           return entrada;
       }
       public LocalDate getSalida() {
           return salida;
       }
       public ArrayList<Clientes> getClientes() {
           return clientes;
       }

       public ArrayList<Habitaciones> getHabitaciones() {
           return habitaciones;
       }
       public void setNumeroReserva(int numeroReserva) {
           this.numeroReserva = numeroReserva;
       }

       public void setEntrada(LocalDate entrada) {
           this.entrada = entrada;
       }

       public void setSalida(LocalDate salida) {
           this.salida = salida;
       }

       public void setClientes(ArrayList<Clientes> clientes) {
           this.clientes = clientes;
       }

       public void setHabitaciones(ArrayList<Habitaciones>
   habitaciones) {
           this.habitaciones = habitaciones;
       }
       public void añadirCliente(Clientes c) {
           clientes.add(c);
       }
```

```
    public void añadirHabitacion(Habitaciones h) {
        habitaciones.add(h);
    }
    //Obtener la ultima reserva
    public static int ultimaReserva(Hotel hotel) {
        int numeroUltimaReserva = 0;
        ArrayList<Reservas>reservas = hotel.getReservas();
        numeroUltimaReserva = (reservas.get(reservas.size() -
1).getNumeroReserva()) + 1;
        return numeroUltimaReserva;
    }
    //Analizar LocalDate
    public static LocalDate analizarFecha(String string) {
            LocalDate analizarDate = null;
            do try {
                analizarDate = LocalDate.parse(string);
            } catch (DateTimeParseException e) {
                System.out.println("Introducir un valor de fecha
valida (dd-mm-YYY)" + e);
            }   while (analizarDate == null);

            return analizarDate;
        }
    //Comprobar si la habitacion esta disponible para la reserva
    public static boolean habitacionDisponible (LocalDate
entradaReservaNueva, LocalDate salidaNuevaReserva) {
        return entradaReservaNueva.isAfter(salidaNuevaReserva)
&&
                entradaReservaNueva.isEqual(salidaNuevaReserva);
    }
    // Comprobar si la fecha esta disponible en un periodo de
tiempo
    public static boolean fechaDisponible(LocalDate fechaInicio,
LocalDate fechaFin, LocalDate entrada,
                                          LocalDate salida) {
        return salida.isAfter(fechaInicio) &&
entrada.isBefore(fechaFin);
    }

    // Consultar habitaciones disponibles en un período de
tiempo, filtradas por tipo de habitación
    public static void peridoHabitacion(Hotel hotel, LocalDate
fechaInicio, LocalDate fechaFin) {
        for (Habitaciones habitaciones :
hotel.getHabitaciones()) {
            if (habitaciones instanceof HabitacionIndividual) {
                comprobarPeriodoHabitacion(hotel, fechaInicio,
fechaFin, habitaciones);
            }
        }

        for (Habitaciones habitaciones :
hotel.getHabitaciones()) {
            if (habitaciones instanceof HabitacionDoble) {
                comprobarPeriodoHabitacion(hotel, fechaInicio,
fechaFin, habitaciones);
            }
        }

        for (Habitaciones habitaciones :
hotel.getHabitaciones()) {
```

```
            comprobarPeriodoHabitacion(hotel, fechaInicio,
fechaFin, habitaciones);
        }
    }

    public static void comprobarPeriodoHabitacion(Hotel hotel,
LocalDate fechaInicio, LocalDate fechaFin, Habitaciones
habitaciones) {
        boolean ocupado = false;
        for (Reservas reservas : hotel.getReservas()) {
            LocalDate entrada = reservas.getEntrada();

            LocalDate salida = reservas.getSalida();

            // Si la reserva es en el mismo periodo de tiempo y
es la misma habitación
            if ((fechaDisponible(fechaInicio, fechaFin, entrada,
salida)) && (reservas.getHabitaciones().contains(habitaciones)))
{
                ocupado = true;
                break;
            }
        }
        if (!ocupado) {
            System.out.println(habitaciones);
        }
    }

    public static void borrarReserva(Hotel hotel, int
numeroReserva) {
        for (Reservas reservas : hotel.getReservas()) {
            if (reservas.getNumeroReserva() == numeroReserva) {
                hotel.borrarReserva(reservas);
                break;
            }
        }
    }

    @Override
    public String toString() {
        return "Numero de la reserva: " + this.numeroReserva +
", el cliente es " + this.clientes +
                " en la habitacion " + this.habitaciones + ",
fecha de entrada: " + this.entrada + ", fecha de salida: " +
this.salida;
    }
}
```

1.4 Clase Hotel

Este es un programa en Java que define una clase Hotel con variables de instancia y métodos para administrar los clientes, reservas y habitaciones del hotel. El programa incluye constructores, métodos getter y setter, métodos para agregar y eliminar clientes, reservas y habitaciones, y métodos para cargar datos en la aplicación.

La clase Hotel tiene las siguientes variables de instancia:

- nombreHotel: un String que representa el nombre del hotel
- direccion: un String que representa la dirección del hotel
- ciudad: un String que representa la ciudad donde se encuentra el hotel
- cliente: un ArrayList de objetos Clientes que almacena los clientes del hotel
- reserva: un ArrayList de objetos Reservas que almacena las reservas del hotel
- habitacion: un ArrayList de objetos Habitaciones que almacena las habitaciones del hotel

La clase Hotel tiene los siguientes métodos:

- Un constructor que inicializa las variables de instancia.
- Métodos getter y setter para las variables de instancia.
- getClientes(String ID): un método que toma un ID de cliente como parámetro y devuelve el objeto Clientes con ese ID del ArrayList cliente.
- getHabitaciones(int numero): un método que toma un número de habitación como parámetro y devuelve el objeto Habitaciones con ese número del ArrayList habitacion.
- añadirReservas(Reservas reservas): un método que toma un objeto Reservas como parámetro y lo agrega al ArrayList reserva.
- añadirClientes(Clientes clientes): un método que toma un objeto Clientes como parámetro y lo agrega al ArrayList cliente.
- añadirHabitacion(Habitaciones habitaciones): un método que toma un objeto Habitaciones como parámetro y lo agrega al ArrayList habitacion.
- borrarReserva(Reservas reservas): un método que toma un objeto Reservas como parámetro y lo elimina del ArrayList reserva.
- borrarCliente(Clientes clientes): un método que toma un objeto Clientes como parámetro y lo elimina del ArrayList cliente.
- borrarHabitacion(Habitaciones habitaciones): un método que toma un objeto Habitaciones como parámetro y lo elimina del ArrayList habitacion.
- borrarDatos(): un método que borra todos los datos de los ArrayList cliente, reserva y habitacion.
- cargarDatos(Hotel hotel): un método estático que toma un objeto Hotel como parámetro y carga datos en los ArrayList cliente, reserva y habitacion. El método crea objetos HabitacionIndividual, HabitacionDoble, HabitacionSuite, Clientes y Reservas, los agrega a los ArrayList y borra los datos antes de cargarlos nuevamente.

```java
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.Iterator;

public class Hotel {
    private String nombreHotel;
    private String direccion;
    private String ciudad;
    private ArrayList<Clientes> cliente;
    private ArrayList<Reservas> reserva;
    private ArrayList<Habitaciones> habitacion;

    public Hotel(String nombreHotel, String direccion, String
ciudad) {
        this.nombreHotel = nombreHotel;
        this.direccion = direccion;
        this.ciudad = ciudad;
        this.cliente = new ArrayList<>();
        this.reserva = new ArrayList<>();
        this.habitacion = new ArrayList<>();
    }

    public String getNombreHotel() {
        return nombreHotel;
    }

    public String getDireccion() {
        return direccion;
    }

    public String getCiudad() {
        return ciudad;
    }

    public ArrayList<Clientes> getClientes() {
        return cliente;
    }

    public ArrayList<Reservas> getReservas() {
        return reserva;
    }

    public ArrayList<Habitaciones> getHabitaciones() {
        return habitacion;
    }

    public void setNombreHotel(String nombreHotel) {
        this.nombreHotel = nombreHotel;
    }

    public void setDireccion(String direccion) {
        this.direccion = direccion;
    }

    public void setCiudad(String ciudad) {
        this.ciudad = ciudad;
    }

    public void setClientes(ArrayList<Clientes> clientes) {
        this.cliente = clientes;
    }
```

```
    public void setReservas(ArrayList<Reservas> reservas) {
        this.reserva = reserva;
    }

    public void setHabitaciones(ArrayList<Habitaciones>
habitaciones) {
        this.habitacion = habitacion;
    }

    // Metodos de clases
    // Obtener el clinete de la arrayList con el ID
    public Clientes getClientes(String ID) {
        Clientes clientes = null;
        boolean encontrado = false;
        Iterator<Clientes> it = cliente.iterator();
        while (!encontrado && it.hasNext()){
            Clientes c = it.next();
            if (c.getID().equals(ID)) {
                clientes = c;
                encontrado = true;
            }
        }
        return clientes;
    }

    public Habitaciones getHabitaciones(int numero) {
        Habitaciones habitaciones = null;
        boolean encontrado = false;
        Iterator<Habitaciones> it = habitacion.iterator();
        while (!encontrado && it.hasNext()) {
            Habitaciones r = it.next();
            if (r.getNumeroHabitacion() == numero) {
                habitaciones = r;
                encontrado = true;
            }
        }
        return habitaciones;
    }

    // Metodos para añadir al arrayList
    public void añadirReservas(Reservas reservas)
{reserva.add(reservas);}

    public void añadirClientes(Clientes clientes)
{cliente.add(clientes);}

    public void añadirHabitacion(Habitaciones habitaciones) {
habitacion.add(habitaciones);}

    // Metodos para borrar del arrayList
    public void borrarReserva(Reservas reservas)
{reserva.remove(reservas);}

    public void borrarCliente(Clientes clientes)
{cliente.remove(clientes);}

    public void borrarHabitacion(Habitaciones habitaciones)
{habitacion.remove(habitaciones);}

    // Limpiar informacion en el arrayList
```

```
    public void borrarDatos() {
        reserva.clear();
        habitacion.clear();
        cliente.clear();
    }

    // Precarga de informacion en la aplicacion
    public static void cargarDatos(Hotel hotel) {
        HabitacionIndividual Individual;
        HabitacionDoble Doble;
        HabitacionSuite Suite;
        Habitaciones habitaciones;
        Clientes clientes;
        Reservas reservas;

        hotel.borrarDatos();

        // Habitaciones
        Individual = new HabitacionIndividual(1, 24.5, 80.4);
        hotel.añadirHabitacion(Individual);
        Individual = new HabitacionIndividual(4, 12.5, 78.3);
        hotel.añadirHabitacion(Individual);
        Individual = new HabitacionIndividual(5, 20.4, 79.4);
        hotel.añadirHabitacion(Individual);
        Doble = new  HabitacionDoble(2, 30.1, 100.76, "cama
mediana");
        hotel.añadirHabitacion(Doble);
        Doble = new  HabitacionDoble(6, 30.1, 100.76, "cama
mediana");
        hotel.añadirHabitacion(Doble);
        Doble = new  HabitacionDoble(7, 30.1, 100.76, "cama
mediana");
        hotel.añadirHabitacion(Doble);
        Suite = new HabitacionSuite(3, 30.8, 24.5, 145.89);
        hotel.añadirHabitacion(Suite);
        Suite = new HabitacionSuite(8, 30.8, 24.5, 145.89);
        hotel.añadirHabitacion(Suite);
        Suite = new HabitacionSuite(9, 30.8, 24.5, 145.89);
        hotel.añadirHabitacion(Suite);

        // Clientes
        clientes = new Clientes("Diego", "Garcia Lopez",
"2345673A");
        hotel.añadirClientes(clientes);
        clientes = new Clientes("Nicolas", "Prado Sanchez",
"33529651R");
        hotel.añadirClientes(clientes);
        clientes = new Clientes("Paula", "Garcia Perez",
"1234567Q");
        hotel.añadirClientes(clientes);
        //Reservas
        clientes = hotel.getClientes("2345673A");
        LocalDate entrada = LocalDate.of(2021, 12, 22);
        LocalDate salida = LocalDate.of(2021, 12, 30);
        reservas = new Reservas(1, entrada, salida);
        habitaciones = hotel.getHabitaciones(1);
        reservas.añadirHabitacion(habitaciones);
        reservas.añadirCliente(clientes);
        hotel.añadirReservas(reservas);

        clientes = hotel.getClientes("2345673A");
```

```
            entrada = LocalDate.of(2021, 12, 02);
            salida = LocalDate.of(2021, 12, 14);
            reservas = new Reservas(5,  entrada, salida);
            habitaciones = hotel.getHabitaciones(8);
            reservas.añadirHabitacion(habitaciones);
            reservas.añadirCliente(clientes);
            hotel.añadirReservas(reservas);

            clientes = hotel.getClientes("33529651R");
            entrada = LocalDate.of(2021, 12, 24);
            salida = LocalDate.of(2021, 12, 28);
            reservas = new Reservas(2,  entrada, salida);
            habitaciones = hotel.getHabitaciones(4);
            reservas.añadirHabitacion(habitaciones);
            reservas.añadirCliente(clientes);
            hotel.añadirReservas(reservas);

            clientes = hotel.getClientes("1234567Q");
            entrada = LocalDate.of(2021, 12, 17);
            salida= LocalDate.of(2021, 12, 22);
            reservas = new Reservas(3,  entrada, salida);
            habitaciones = hotel.getHabitaciones(9);
            reservas.añadirHabitacion(habitaciones);
            reservas.añadirCliente(clientes);
            hotel.añadirReservas(reservas);

            clientes = hotel.getClientes("2345673A");
            entrada = LocalDate.of(2021, 12, 16);
            salida = LocalDate.of(2021, 12, 20);
            reservas = new Reservas(4,  entrada, salida);
            habitaciones = hotel.getHabitaciones(5);
            reservas.añadirHabitacion(habitaciones);
            reservas.añadirCliente(clientes);
            hotel.añadirReservas(reservas);


            System.out.println("Datos cargados correctamente.");
        }

        @Override
        public String toString() {
            return "Informacion sobre el hotel: Nombre " +
    this.nombreHotel + " la calle es " + this.direccion + " en la
    ciudad " + this.ciudad;
        }
    }
```

El programa está bien estructurado y organizado, con una clase principal que define un objeto Hotel y sus atributos, así como una serie de métodos para gestionar los clientes, reservas y habitaciones del hotel.

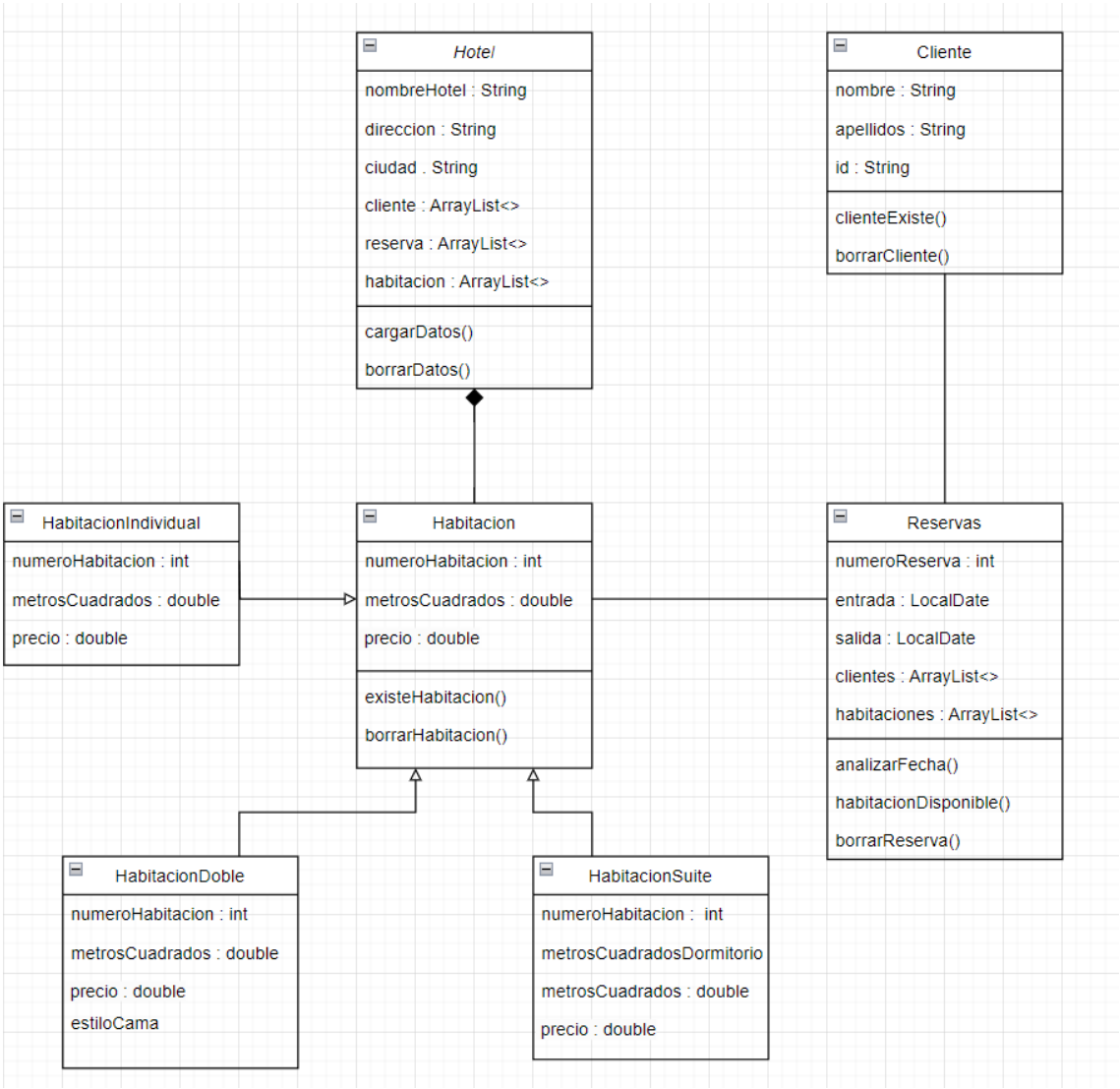
El uso de ArrayLists es una buena elección para almacenar los objetos de Cliente, Reserva y Habitación, ya que permite la adición, eliminación y búsqueda de elementos de forma eficiente.

Además, el uso de métodos getter y setter es una buena práctica para asegurar la encapsulación de los datos de la clase y facilitar el acceso a ellos de forma controlada.

También es destacable la inclusión de métodos para pre-cargar datos en la aplicación, lo que puede ser útil para pruebas y desarrollo.

En general, el programa parece estar bien desarrollado y estructurado, y parece cumplir con los requisitos de gestión de clientes, reservas y habitaciones de un hotel.

3. DIAGRAMA DE CLASES



4. REFERENCIAS BIBLIOGRAFICAS

Jiménez Marín, A. & Pérez Montes, F. M. (2016). Aprende a Programar con Java (INFORMÁTICA). España: Ediciones ENI.