20-1-2021

# Práctica Final Gestión de un Balneario

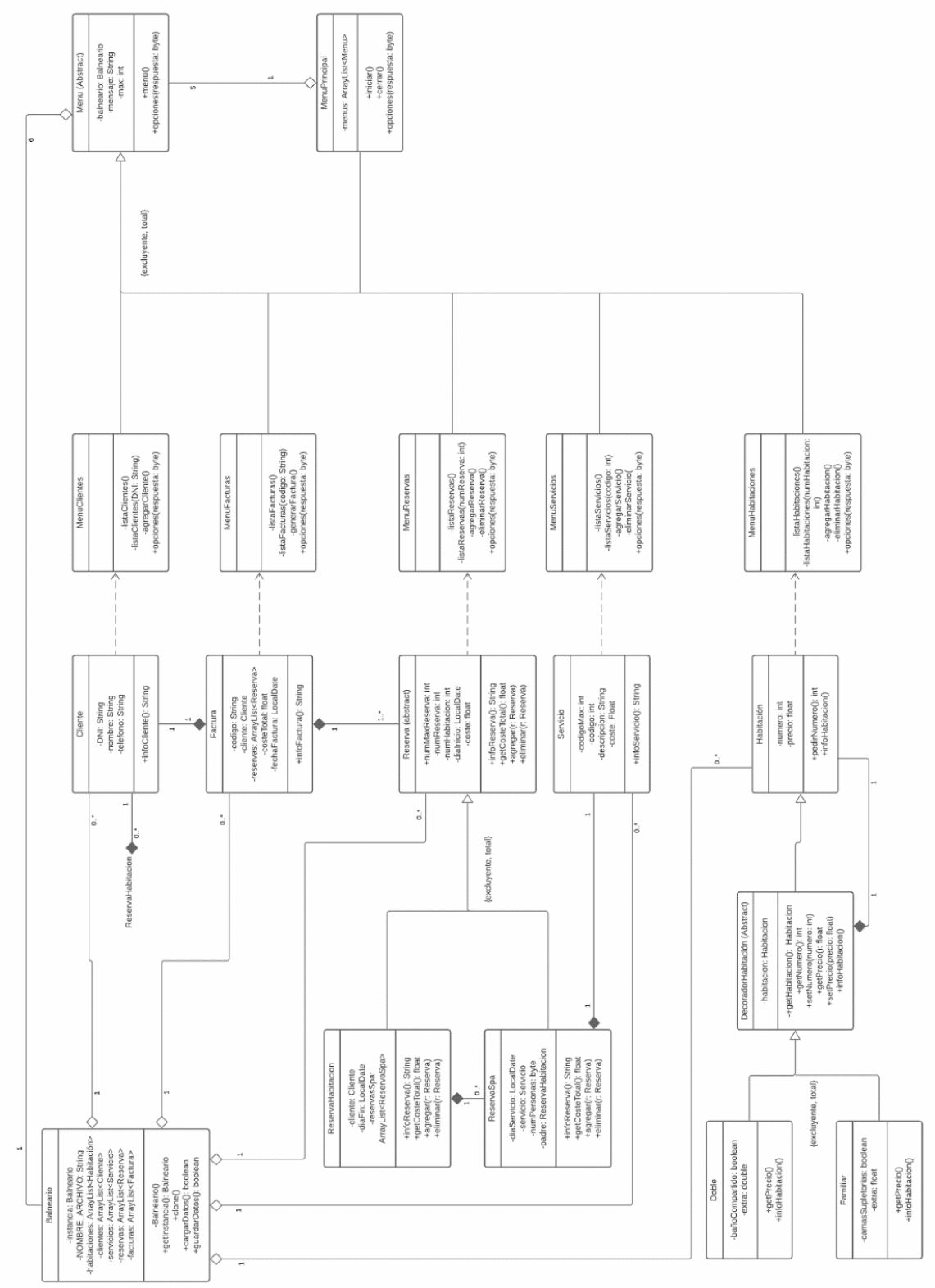
Curso 2020 – 2021

Gustavo Cortés Jiménez
Rodrigo Lázaro Escudero

# Contenido

# Diseño del sistema

## Diagrama de clases



## Especificaciones

Para favorecer la comprensión del esquema anterior, no se han incluido las clases factory, ya que su única función es crear objetos.

Además, también se han excluido las funciones getter y setter, así como diversas funciones de apoyo: buscarHabitacion(), buscarCliente(), etc, en Balneario, pedirId() y las validaciones.

## Patrones empleados

Para la planificación del proyecto hemos empleado diversos patrones.

La clase Balneario emplea el patrón Singleton.

Para el funcionamientos de los menús, hemos implementado los patrones Strategy y Template Method.

Se ha seguido el patrón Decorator para definir los diferentes tipos de habitaciones.

Para definir la estructura de Reservas, se ha empleado el patrón Composite.

La interfaz Factory junto a sus subclases siguen, efectivamente, el patrón Factory.

# Implementación

A continuación se incluye el código de las clases principales para la lógica de negocio de la aplicación de gestión de balnearios. Se presentará en primer lugar una breve explicación de su funcionalidad, estructuras de datos u observaciones que se estimen importantes para su mejor comprensión; y se incluirá el código de la clase a continuación. Se omitirán las partes menos importantes del código (como por ejemplo los paquetes o imports realizados) a fin de hacer la exposición más legible. Se han conservado las cabeceras de javadoc por su valor informativo.

Finalmente, se incluye una explicación de cómo se han implementado los patrones utilizados en la aplicación.

## Clase Balneario

La clase Balneario implementa el patrón Singleton, garantizando su unicidad. En esta clase se almacenan como ArrayLists todos los datos necesarios para la gestión del Balneario (habitaciones, clientes, servicios, reservas y facturas). La clase contiene además los métodos necesarios para almacenar y recuperar posteriormente dichos datos. El procedimiento empleado a tal fin es la serialización, convirtiendo los objetos en streams de datos que se escriben/leen secuencialmente en/desde un archivo en almacenamiento secundario. Asimismo, Balneario ofrece métodos auxiliares (llamados BuscaX, donde X indica el recurso a buscar en cada caso) para facilitar la búsqueda de objetos específicos.

```
/**
 * Contiene la información principal de la aplicación.
 * @author Gustavo Cortés Jiménez.
 * @author Rodrigo Lázaro Escudero.
 */
public class Balneario implements Serializable{
    private static Balneario instancia;
    private final String NOMBRE_ARCHIVO = "Configuracion.dat";
    private ArrayList<Habitacion> habitaciones= new ArrayList();
    private ArrayList<Servicio> servicios = new ArrayList();
    private ArrayList<Reserva> reservas = new ArrayList();
    private ArrayList<Cliente> clientes = new ArrayList();
    private ArrayList<Factura> facturas = new ArrayList();

    private Balneario(){}
```

```
@Override
public Object clone() throws CloneNotSupportedException{
   throw new CloneNotSupportedException();
}

/**
 * Carga los datos almacenados en el archivo de configuración en la aplicación.
 * @return false si han ocurrido fallos; si todo ha ido bien, true.
 */
public boolean cargarDatos(){
   try{
      FileInputStream in = new FileInputStream(NOMBRE_ARCHIVO);
      ObjectInputStream ois = new ObjectInputStream(in);
      habitaciones = (ArrayList<Habitacion>)ois.readObject();
      servicios = (ArrayList<Servicio>)ois.readObject();
      reservas = (ArrayList<Reserva>)ois.readObject();
      clientes = (ArrayList<Cliente>)ois.readObject();
      facturas = (ArrayList<Factura>)ois.readObject();
      Servicio.codigoMax = ois.readInt();
      Reserva.numMaxReserva = ois.readInt();
      ois.close();
      ois.close();
   }catch(FileNotFoundException e){
      System.out.println("No se ha encontrado el archivo de configuración.");
      return false;
   }catch(Exception e){
      System.out.println("Se ha producido un error en la lectura del archivo.");
      return false;
   }
   System.out.println("Datos cargados");
   return true;
```

```
}

/**
 * Guarda los datos de la aplicación en el archivo de configuración.
 * @return false si han ocurrido fallos; si todo ha ido bien, true.
 */
public boolean guardarDatos(){
   try{
      FileOutputStream out = new FileOutputStream(NOMBRE_ARCHIVO);
      ObjectOutputStream oos = new ObjectOutputStream(out);
      oos.writeObject(habitaciones);
      oos.writeObject(servicios);
      oos.writeObject(reservas);
      oos.writeObject(clientes);
      oos.writeObject(facturas);
      oos.writeInt(Servicio.codigoMax);
      oos.writeInt(Reserva.numMaxReserva);
      oos.flush();
      oos.close();
      out.close();
   }catch(FileNotFoundException e){
      System.out.println("No se ha podido crear el archivo de configuración.");
      return false;
   }catch(IOException e){
      System.out.println("Se ha producido un error en la escritura.");
      return false;
   }
   System.out.println("\nDatos guardados\n");
   return true;
}
```

```
/**
 * Devuelve la habitación cuyo número se corresponda con el recibido.
 * @param numero el número de la habitación.
 * @return la habitación correspondiente; si no, null
 */
public Habitacion buscarHabitacion(int numero){
   for (Habitacion h: habitaciones){
      if (h.getNumero() == numero){
         return h;
      }
   }
   return null;
}

/**
 * Devuelve el servicio cuyo id se corresponda con el recibido.
 * @param idServicio el id del servicio.
 * @return el servicio correspondiente; si no, null
 */
public Servicio buscarServicio(int idServicio){
   for (Servicio s: servicios){
      if (s.getCodigo() == idServicio){
         return s;
      }
   }
   return null;
}

/**
 * Devuelve la reserva cuyo id se corresponda con el recibido.
 * @param idReserva el id de la reserva.
```

```
 * @return la reserva correspondiente; si no, null
 */
public Reserva buscarReserva(int idReserva){
   for (Reserva r: reservas){
      if (r.getNumReserva() == idReserva){
         return r;
      }
   }
   return null;
}

/**
 * Devuelve el cliente cuyo id se corresponda con el recibido.
 * @param idCliente el id del cliente.
 * @return el cliente correspondiente; si no, null.
 */
public Cliente buscarCliente(String idCliente){
   for (Cliente c: clientes){
      if (c.getDni().compareToIgnoreCase(idCliente) == 0){
         return c;
      }
   }
   return null;
}

/**
 * Devuelve la factura cuyo número de corresponda con el recibido.
 * @param numero el número de la factura.
 * @return la factura correspondiente; si no, null.
 */
public Factura buscarFacura(String numero){
```

```
    for (Factura f: facturas){
        if (f.getCodigo().equals(numero)){
            return f;
        }
    }
    return null;
}

//Getters y Setters
public static Balneario getInstancia(){
    if(instancia == null){
        instancia = new Balneario();
    }

    return instancia;
}

public ArrayList<Habitacion> getHabitaciones() {
    return habitaciones;
}

public ArrayList<Servicio> getServicios() {
    return servicios;
}

public ArrayList<Reserva> getReservas() {
    return reservas;
}

public ArrayList<Cliente> getClientes() {
    return clientes;
```

```
    }

    public ArrayList<Factura> getFacturas() {
        return facturas;
    }
}
```

## Clase Cliente

El propósito de esta clase es almacenar los datos de los clientes del balneario (DNI, Nombre y Apellidos, Teléfono Móvil). Ofrece además ciertos métodos auxiliares para validación, petición de identificadores, etc.

```
/**
 * Contiene la información y las funciones de un cliente.
 * @author Gustavo Cortés Jiménez.
 * @author Rodrigo Lázaro Escudero.
 */
public class Cliente implements Serializable{
    //Atributos
    private String dni;
    private String nombreApellidos;
    private String telefonoMovil;

    //Constructores
    public Cliente(String dni, String nombreApellidos, String telefonoMovil) {
        this.dni = dni;
        this.nombreApellidos = nombreApellidos;
        this.telefonoMovil = telefonoMovil;
    }

    //Métodos
    /**
```

```
 * Muestra la información del cliente.
 * @return los datos del cliente.
 */
public String infoCliente(){
   return "DNI: " + dni
         + "\nNombre y Apellidos: " + nombreApellidos
         + "\nTeléfono Móvil: " + telefonoMovil + "\n";
}

/**
 * Pide un DNI.
 * @return un DNI.
 * @post el DNI debe tener el formato correspondiente.
 */
public static String pedirId(){
   boolean flag = true;
   BufferedReader br;
   String dni = " ";
   do {
      System.out.println("Introduce el DNI del cliente: ");
      try {
         br = new BufferedReader(new InputStreamReader(System.in),1);
         dni = br.readLine();
         //Validar
         if (!validaDni(dni)) {
            throw new Exception();
         }

         flag = false;
      }
      catch (Exception e) {
```

```
            System.out.println("Error de formato, vuelve a intentarlo (un DNI consiste de 8 dígitos y una letra).");
        }
    } while (flag);
    return dni;
  }

  /**
   * Comprueba si el DNI recibido tiene la estructura correcta.
   * @param dni el DNI a validar
   * @return true si es correcto; si no, false.
   */
  public static boolean validaDni(String dni) {
    if (dni.length() != 9) {
      return false;
    }
    //Va a ser más rápido comprobar con regex que ir caso a caso
    if (dni.matches("\\d{8}[A-Za-z]")) {
      return true;
    }
    return false;
  }

  /**
   * Comprueba si el teléfono recibido tiene la estructura correcta.
   * @param telefonoMovil un teléfono a validar.
   * @return true si es correcto; si no, false.
   */
  public static boolean validaMovil(String telefonoMovil) {
    int telMov;
```

```
        //No vamos a imponer más restricciones que que sea un número de 9 cifras
        if (telefonoMovil.length() != 9 || telefonoMovil.charAt(0) == '+') {
            return false;
        }
        try {
            telMov = Integer.parseUnsignedInt(telefonoMovil);
        } catch (NumberFormatException e) {
                return false;
        }
        return true;
    }

    //Getters y Setters
    public String getDni() {
        return dni;
    }
}
```

## Clase Servicio

El propósito de esta clase es almacenar los datos de los servicios ofrecidos por el balneario (código de servicio, descripción de servicio y coste). Ofrece además ciertos métodos auxiliares para validación, mostrar información por pantalla con formato, etc.

```
/**
 * Contiene la información y las funciones de un servicio.
 * @author Gustavo Cortés Jiménez.
 * @author Rodrigo Lázaro Escudero.
 */
public class Servicio implements Serializable{
    //Atributos
    public static int codigoMax = 0;
    private int codigo;
```

```
private String descripcion;
private float coste;

//Constructores
public Servicio(int codigo, String descripcion, float coste) {
   this.codigo = codigo;
   this.descripcion = descripcion;
   this.coste = coste;
}

//Métodos
/**
 * Muestra la información de la habitación.
 * @return los datos de la factura.
 */
public String infoServicio(){
   return "Código de servicio: " + codigo
         + "\nDescripción: " + descripcion
         + "\nCoste: " + coste + " euros.\n";
}

/**
 * Pide un código de un servicio.
 * @return un código de servicio.
 */
public static int pedirId(){
   int id = 0;
   boolean flag = true;
   Scanner sc;
   do{
      System.out.println("Introduce el código del servicio: ");
```

```
            try{
                sc = new Scanner(System.in);
                id = sc.nextInt();
                flag = false;
            }catch(Exception e){
                System.out.println("\nEl valor introducido debe ser un número.\n");
            }
        }while(flag);
        return id;
    }

    //Getters y Setters
    public int getCodigo() {
        return codigo;
    }
    public String getDescripcion() {
        return descripcion;
    }
    public float getCoste() {
        return coste;
    }
}
```

## Clase Factura

El propósito de esta clase es almacenar los datos de las facturas del balneario (código de factura, cliente y reserva a los que está asociada, coste y fecha de facturación). Ofrece además ciertos métodos auxiliares para validación, mostrar información por pantalla con formato, etc.

```
/**
 * Contiene la información y las funciones sobre una factura.
 * @author Gustavo Cortés Jiménez.
 * @author Rodrigo Lázaro Escudero.
```

```
 */
public class Factura implements Serializable{
   //Atributos
   private String codigo;
   private Cliente cliente;
   private Reserva reserva;
   private float costeTotal;
   private LocalDate fechaFactura;

   //Constructores
     public Factura(String codigo, Cliente cliente, Reserva reserva, float costeTotal, LocalDate fechaFactura) {
       this.codigo = codigo;
       this.cliente = cliente;
       this.reserva = reserva;
       this.costeTotal = costeTotal;
       this.fechaFactura = fechaFactura;
   }

   //Métodos
   /**
    * Muestra la información de la factura.
    * @return los datos de la factura.
    */
   public String infoFactura(){
      return "Código de factura: " + codigo
            + "\nDNI de cliente: " + cliente.getDni()
            + "\nFecha de facturación: " + fechaFactura + "\n";
   }

   /**
```

```
    * Pide un número de fectura.
    * @return un número de factura.
    */
   public static String pedirId(){
      boolean flag = true;
      BufferedReader br;
      String codigo = " ";
      do {
         System.out.println("Introduce el ID de la factura: ");
         try {
            br = new BufferedReader(new InputStreamReader(System.in),1);
            codigo = br.readLine();
            //Validar
            if (!validaCodigo(codigo)) {
               throw new Exception();
            }
            flag = false;
         }
         catch (Exception e) {
             System.out.println("Error de formato, vuelve a intentarlo (un código de factura tiene
8 dígitos).");
         }
      } while (flag);
      return codigo;
   }

   /**
    * Comprueba si el código recibido tiene la estructura correcta.
    * @param codigo el código de factura a comprobar.
    * @return true si es correcto; si no, false.
    */
```

```
    public static boolean validaCodigo(String codigo) {
        if (codigo.length() != 8) {
            return false;
        }
        //Comprobamos formato con regex
        if (codigo.matches("\\d{8}")) {
            return true;
        }
        return false;
    }

    //Getters y Setters
    public String getCodigo() {
        return codigo;
    }
    public Cliente getCliente() {
        return cliente;
    }
    public Reserva getReserva() {
        return reserva;
    }
    public float getCosteTotal() {
        return costeTotal;
    }
    public LocalDate getFechaFactura() {
        return fechaFactura;
    }
}
```

## Clase abstracta Reserva

El propósito de esta clase es almacenar los datos de las reservas del balneario (número de reserva, número de habitación, día de inicio y coste). Incluye varios métodos abstractos que

serán luego implementados por las subclases ReservaHabitacion y ReservaSpa. Ofrece además ciertos métodos auxiliares. Junto con sus subclases ejemplifica el uso del patrón Composite.

```
/**
 * Contiene la información y las funciones de una reserva.
 * @author Gustavo Cortés Jiménez.
 * @author Rodrigo Lázaro Escudero.
 */
public abstract class Reserva implements Serializable{
   //Atributos
   public static int numMaxReserva = 0;
   private int numReserva;
   private int numHabitacion;
   private LocalDate diaInicio;
   private float coste;

   //Constructores
   public Reserva(int numReserva, int numHabitacion, LocalDate diaInicio, float coste) {
      this.numReserva = numReserva;
      this.numHabitacion = numHabitacion;
      this.diaInicio = diaInicio;
      this.coste = coste;
   }

   //Métodos
   /**
    * Muestra la información de la reserva.
    * @return los datos de la reserva.
    */
   public abstract String infoReserva();
```

```
/**
 * Calcula el coste del servicio.
 * @return el coste total del servicio.
 */
public abstract float getCosteTotal();

/**
 * Añade una nueva reserva al ArrayList de reservas.
 * @param r una reserva que añadir.
 */
public abstract void agregar(Reserva r);

/**
 * Elimina una reserva del ArrayList de reservas.
 * @param r una reserva que eliminar.
 */
public abstract void eliminar(Reserva r);

/**
 * Pide un número de reserva.
 * @return un número de reserva.
 */
public static int pedirId(){
   int id = 0;
   boolean flag = true;
   Scanner sc;
   do{
      System.out.println("Introduce el número de la reserva: ");
      try{
         sc = new Scanner(System.in);
         id = sc.nextInt();
```

```
            flag = false;
        }catch(Exception e){
            System.out.println("\nEl valor introducido debe ser un número.\n");
        }
    }while(flag);
    return id;
  }

  //Getters y Setters
  public int getNumReserva() {
    return numReserva;
  }
  public int getNumHabitacion() {
    return numHabitacion;
  }
  public LocalDate getDiaInicio() {
    return diaInicio;
  }
  public float getCoste() {
    return coste;
  }
}
```

## Clase ReservaHabitacion

El propósito de esta clase es almacenar los datos de las reservas de habitaciones del balneario (además de los campos que hereda de Reserva, almacena información del cliente, el día en que termina la reserva y una lista con las reservas de spa asociadas a la habitación). Implementa los métodos abstractos de su superclase Reservas.

```
/**
 * Concreta la reserva de una habitación.
```

```
 * @author Gustavo Cortés Jiménez.
 * @author Rodrigo Lázaro Escudero.
 */
public class ReservaHabitacion extends Reserva{
   //Atributos
   private Cliente cliente;
   private LocalDate diaFin;
    private ArrayList<Reserva> reservasSpa = new ArrayList<Reserva>(); //Tipo Reserva en vez de ReservaSpa porque usamos patrón Composite

   //Constructores
      public ReservaHabitacion(int numReserva, int numHabitacion, LocalDate diaInicio, float coste, Cliente cliente, LocalDate diaFin){
      super(numReserva, numHabitacion, diaInicio, coste);
      this.cliente = cliente;
      this.diaFin = diaFin;
   }

   //Métodos
   @Override
   public String infoReserva(){
      String concatReservasSpa = "\n";
      for(Reserva rs: reservasSpa){
         concatReservasSpa = concatReservasSpa + rs.infoReserva();
      }

      return "Identificador de reserva: " + getNumReserva()
            + "\nNúmero de habitación: " + getNumHabitacion()
            + "\nDía de inicio: " + getDiaInicio()
            + "\nCoste de la habitación: " + getCoste()
            + "\nCoste total de habitación + servicios: " + getCosteTotal()
            + "\nDía de fin de reserva: " + diaFin
```

```
        + "\nServicios de Spa asociados: " + concatReservasSpa;
}

@Override
public float getCosteTotal() {
   float total = getCoste();
   for(Reserva rs: reservasSpa){
      total = total + rs.getCoste();
   }
   return total;
}

@Override
public void agregar(Reserva r){
   reservasSpa.add(r);
   System.out.println("Se ha agregado Reserva de Spa: " + r.getNumReserva()
        + " a la Reserva de Habitación: " + getNumReserva()
        + ", con número de habitación: " + getNumHabitacion() + "\n");
}

@Override
public void eliminar(Reserva r){
   reservasSpa.remove(r);
   System.out.println("Se ha eliminado la Reserva de Spa: " + r.getNumReserva()
        + " de Reserva de Habitación: " + getNumReserva()
        + ", con número de habitación: " + getNumHabitacion() + "\n");
}

//Getters y Setters
public ArrayList<Reserva> getReservasSpa() {
   return reservasSpa;
```

```
    }
    public Cliente getCliente() {
        return cliente;
    }
    public LocalDate getDiaFin() {
        return diaFin;
    }
}
```

## Clase ReservaSpa

El propósito de esta clase es almacenar los datos de las reservas de servicios de spa del balneario (además de los campos que hereda de Reserva, almacena el día del servicio, el servicio, el número de personas y una referencia a la habitación a la que se vincula). Implementa los métodos abstractos de su superclase Reservas.

```
/**
 * Concreta una reserva del Spa.
 * @author Gustavo Cortés Jiménez.
 * @author Rodrigo Lázaro Escudero.
 */
public class ReservaSpa extends Reserva{
    //Atributos
    private LocalDate diaServicio;
    private Servicio servicio;
    private byte numPersonas;
    private ReservaHabitacion padre;

    //Constructores
    public ReservaSpa(int numReserva, int numHabitacion, LocalDate diaInicio, float coste, LocalDate diaServicio, Servicio servicio, byte numPersonas, ReservaHabitacion padre){
        super(numReserva, numHabitacion, diaInicio, coste);
        this.diaServicio = diaServicio;
        this.servicio = servicio;
```

```
    this.numPersonas = numPersonas;
    this.padre = padre;
}

//Métodos
@Override
public String infoReserva(){
    return "Identificador de reserva: " + getNumReserva()
            + "\nCódigo del servicio: " + servicio.getCodigo()
            + "\nDescripción del servicio: " + servicio.getDescripcion()
            + "\nCoste base del servicio: " + getCoste()
            + "\nCoste total del servicio (coste base * número de personas): " + getCosteTotal()
            + "\nDía servicio: " + diaServicio
            + "\nNúmero de personas: " + numPersonas + "\n";
}

@Override
public float getCosteTotal() {
    return getCoste() * numPersonas;
}

@Override
/**
 * Muestra un mensaje de error.
 */
public void agregar(Reserva r){
    System.out.println("Error. Las Reservas de Spa no tienen sub-reservas.\n");
}

/**
 * Muestra un mensaje de error.
```

```
     */
    @Override
    public void eliminar(Reserva r){
        System.out.println("Error. Las Reservas de Spa no tienen sub-reservas.\n");
    }

    //Getters y Setters
    public LocalDate getDiaServicio() {
        return diaServicio;
    }
    public Servicio getServicio() {
        return servicio;
    }
    public ReservaHabitacion getPadre() {
        return padre;
    }
}
```

## Clase Habitacion

El propósito de esta clase es almacenar los datos de las habitaciones del balneario (número y precio). Implementa métodos auxiliares para validación y dar información con formato por pantalla. Junto con Familiar, Doble y DecoradorHabitación, implementa el patrón Decorator.

```
/**
 * Contiene la información y las funciones básicas de una habitación.
 * @author Gustavo Cortés Jiménez
 * @author Rodrigo Lázaro Escudero
 */
public class Habitacion implements Serializable{
    private int numero;
    private float precio;
```

```
public Habitacion(int numero, float precio){
   this.numero = numero;
   this.precio = precio;
}

//setter - getter
public int getNumero() {
   return numero;
}
public void setNumero(int numero) {
   this.numero = numero;
}
public float getPrecio() {
   return precio;
}
public void setPrecio(float precio) {
   this.precio = precio;
}

/**
 * Pide un número de habitación
 * @return un número de habitación
 * @post el número debe tener el formato adecuado
 */
public static int pedirNumero(){
   int numero = 0;
   boolean flag = true;
   do{
      System.out.println("Introduce un número de habitación: ");
      try{
         Scanner sc = new Scanner(System.in);
```

```
            numero = sc.nextInt();
            if((numero < 100) || (numero > 999)){
               System.out.println("\nEl numero debe ser de tres cifras\n");
            }else{
               flag = false;
            }
         }catch(Exception e){
            System.out.println("\nEl valor introducido no es un numero\n");
         }
      }while(flag);
      return numero;
   }

   /**
    * Muestra la información de la habitación
    */
   public void infoHabitacion(){
      System.out.println("Habitacion individual");
      System.out.println("Numero de habitación: " + getNumero());
      System.out.println("Precio de la habitación: " + getPrecio());
   }
}
```

## Clase DecoradorHabitacion

Clase adicional necesaria para implementar sobre Habitacion el patrón Decorator.

```
/**
 * Complementa a una Habitación, añadiéndola nueva información.
 * @author Gustavo Cortés Jiménez
 * @author Rodrigo Lázaro Escudero
 */
```

```
public class Familiar extends DecoradorHabitacion{
    boolean camasSupletorias;
    private static float extra = 20;

    public Familiar(Habitacion habitacion, boolean camasSupletorias){
        super(habitacion);
        this.camasSupletorias = camasSupletorias;
    }

    public static float getExtra() {
        return extra;
    }

    public static void setExtra(float extra) {
        Familiar.extra = extra;
    }

    @Override
    public float getPrecio() {
        if (camasSupletorias){
            return (float)(super.getPrecio() + getExtra());
        }
        return super.getPrecio();
    }

    @Override
    public void infoHabitacion(){
        System.out.println("Habitacion familiar");
        super.infoHabitacion();
        if (camasSupletorias){
            System.out.println("Tiene camas supletorias");
```

```
        }else{
            System.out.println("No tiene camas supletorias");
        }
    }
}
```

## Clase Doble

Clase adicional necesaria para implementar sobre Habitacion el patrón Decorator.

```
/**
 * Complementa a una Habitación, añadiéndola nueva información.
 * @author Gustavo Cortés Jiménez
 * @author Rodrigo Lázaro Escudero
 */
public class Doble extends DecoradorHabitacion{
    private final boolean banoCompartido;
    private static double extra = 1.15;

    public Doble(Habitacion habitacion, boolean banoCompartido){
        super(habitacion);
        this.banoCompartido = banoCompartido;
    }

    public static double getExtra() {
        return extra;
    }

    public static void setExtra(double extra) {
        Doble.extra = extra;
    }
```

```
    @Override
    public float getPrecio() {
        if (banoCompartido){
            return super.getPrecio();
        }
        return (float)(super.getPrecio() * getExtra());
    }

    @Override
    public void infoHabitacion(){
        System.out.println("Habitacion doble");
        super.infoHabitacion();

        if (banoCompartido){
            System.out.println("El baño está compartido");
        }else{
            System.out.println("El baño es privado");
        }
    }
}
```

## Clase Familiar

Clase adicional necesaria para implementar sobre Habitacion el patrón Decorator.

```
/**
 * Complementa a una Habitación, añadiéndola nueva información.
 * @author Gustavo Cortés Jiménez
 * @author Rodrigo Lázaro Escudero
 */
public class Familiar extends DecoradorHabitacion{
    boolean camasSupletorias;
```

```
private static float extra = 20;
public Familiar(Habitacion habitacion, boolean camasSupletorias){
   super(habitacion);
   this.camasSupletorias = camasSupletorias;
}

public static float getExtra() {
   return extra;
}

public static void setExtra(float extra) {
   Familiar.extra = extra;
}

@Override
public float getPrecio() {
   if (camasSupletorias){
      return (float)(super.getPrecio() + getExtra());
   }
   return super.getPrecio();
}

@Override
public void infoHabitacion(){
   System.out.println("Habitacion familiar");
   super.infoHabitacion();
   if (camasSupletorias){
      System.out.println("Tiene camas supletorias");
   }else{
      System.out.println("No tiene camas supletorias");
   }
```

```
    }
}
```

## Interfaz Factoria

Interfaz simple que permite homogeneizar las factorías que lo implementan; en conjunción con las cuales implementa el patrón FactoryMethod.

```
/**
 * Permite definir clases destinadas a la creación de objetos.
 * @author Gustavo Cortés Jiménez.
 * @author Rodrigo Lázaro Escudero.
 */
public interface Factoria <T>{

    /**
     * Crea una nuevo objeto
     * @param id identificador del objeto a crear
     * @return el objeto creado
     * @pre el identificador debe tener la estructura correcta
     */
    public T getInstancia(String id);
}
```

## Clase FactoryCliente

Factoría encargada de la creación de objetos de clase Cliente. Encapsula todas las peticiones de datos por teclado al usuario. Debido a la independencia de Cliente respecto a otros datos del balneario, sólo se comprueba que los datos de entrada tengan un formato válido. Si el usuario los introduce mal, se le vuelven a pedir inmediatamente sin necesidad de regresar al menú.

```
/**
 * Se encarga del proceso de creación de Clientes
 * @author Gustavo Cortés Jiménez.
```

```
 * @author Rodrigo Lázaro Escudero.
 */
public class FactoryCliente implements Factoria<Cliente>{

   @Override
   public Cliente getInstancia(String idCliente){
      String nombreApellidos = " ";
      String telefonoMovil = " ";
      boolean flag = true;
      BufferedReader br;

      System.out.println("Introduce el nombre y apellidos del cliente:\n");
      do {
         try {
            br = new BufferedReader(new InputStreamReader(System.in),1);
            nombreApellidos = br.readLine();
            flag = false;
         } catch (Exception e) {
            System.out.println("Ha ocurrido un error de lectura. Vuelve a intentarlo.\n");
         }
      } while (flag);
      System.out.println("Introduce el número de teléfono móvil del cliente:\n");
      do {
         try {
            br = new BufferedReader(new InputStreamReader(System.in),1);
            telefonoMovil = br.readLine();
            flag = false;
         } catch (Exception e) {
            System.out.println("Ha ocurrido un error de lectura. Vuelve a intentarlo.\n");
            flag = true;
         }
```

```
            //Validación teléfono móvil
            flag = !Cliente.validaMovil(telefonoMovil);
            if (flag) {
               System.out.println("Formato incorrecto. El número de móvil debe tener 9 cifras.\n");
            }
        } while (flag);
        Cliente cliente = new Cliente(idCliente, nombreApellidos, telefonoMovil);
        return cliente;
    }
}
```

## Clase FactoryServicio

Factoría encargada de la creación de objetos de clase Servicio. Encapsula todas las peticiones de datos por teclado al usuario. Debido a la independencia de Servicio respecto a otros datos del balneario, sólo se comprueba que los datos de entrada tengan un formato válido. Si el usuario los introduce mal, se le vuelven a pedir inmediatamente sin necesidad de regresar al menú.

```
/**
 * Se encarga del proceso de creación de Servicios
 * @author Gustavo Cortés Jiménez.
 * @author Rodrigo Lázaro Escudero.
 */
public class FactoryServicio implements Factoria<Servicio>{

    @Override
    public Servicio getInstancia(String idServicio){
        String descripcion = " ";
        float coste = 0f;
        boolean flag = true;
        BufferedReader br;
        Scanner sc;
```

```
        System.out.println("Introduce una descripción del servicio:\n");
        do {
            try {
                br = new BufferedReader(new InputStreamReader(System.in),1);
                descripcion = br.readLine();
                flag = false;
            } catch (Exception e) {
                System.out.println("Ha ocurrido un error de lectura. Vuelve a intentarlo.");
            }
        } while (flag);
        System.out.println("Introduce el coste del servicio (usar coma): \n");
        flag = true;
        do{
            try {
                sc = new Scanner(System.in);
                coste = sc.nextFloat();
                if (coste < 0){
                    System.out.println("\nEl precio debe ser superior a 0.");
                }else{
                    flag = false;
                }
            } catch(Exception e) {
                    System.out.println("\nEl coste del servicio debe ser un dato de tipo float. Inténtalo
otra vez:\n");
            }
        }while(flag);
        Servicio servicio = new Servicio(Integer.parseInt(idServicio), descripcion, coste);
        return servicio;
    }
}
```

## Clase FactoryFactura

Factoría encargada de la creación de objetos de clase Factura. Encapsula todas las peticiones de datos por teclado al usuario. Comprueba que los datos de entrada tengan un formato válido, y si el usuario los introduce mal se le vuelven a pedir inmediatamente sin necesidad de regresar al menú.

Sin embargo, cuando el usuario introduce determinados datos con formato correcto pero erróneos (ej: intentar hacer una factura sobre una habitación que no existe), se devuelve al usuario al menú. Se trata de una elección de diseño deliberada, ya que podría suceder que el usuario no conozca qué habitaciones existen, y devolviéndolo al menú podrá investigar cuáles son los datos que le faltan. Se provee de un mensaje de error explicativo para que el usuario sepa en todo momento cuál ha sido el problema.

```
/**
 * Se encarga del proceso de creación de Facturas
 * @author Gustavo Cortés Jiménez.
 * @author Rodrigo Lázaro Escudero.
 */
public class FactoryFactura implements Factoria<Factura>{
   @Override
   public Factura getInstancia(String idFactura) {

      Cliente cliente = null;
      ReservaHabitacion reserva = null;
      float costeTotal = 0;
      LocalDate fechaFactura = LocalDate.now();

      int numHabitacion;
      LocalDate fechaInicio = LocalDate.now();
      boolean flag = true;
      BufferedReader br;

      numHabitacion = Habitacion.pedirNumero();
      //Comprobamos que la habitación especificada esté registrada en el sistema
      if (Balneario.getInstancia().buscarHabitacion(numHabitacion) == null){
```

```
            System.out.println("La habitación indicada no está registrada en el sistema. Imposible generar factura.\n");
            return null;
        }

        /** Pedimos al usuario que introduzca fecha de inicio de la reserva para poder identificar unívocamente la
         * habitación y el periodo. Si no lo hiciésemos así, habría problemas en caso de estar reservada la habitación más
         * de una vez.
        */
        System.out.println("Introduce la fecha de inicio de la reserva cuya factura se desea: \n");
        flag = true;
        do{
            try {
                br = new BufferedReader(new InputStreamReader(System.in),1);
                fechaInicio = LocalDate.parse(br.readLine());

                flag = false;
            } catch(Exception e) {
                System.out.println("\nError, fecha inválida. El formato de la fecha debe ser aaaa-mm-dd (Ejemplo: 2000-10-25).\n");
            }
        }while(flag);

        //Comprobamos que la habitación esté reservada y que la fecha de Inicio usada para identificar unívocamente el periodo coincide
        flag = false;
        for (Reserva r: Balneario.getInstancia().getReservas()){
            //r instanceof ReservaHabitacion
            //r.getNumHabitacion() == numHabitacion
            //r.getDiaInicio().isEqual(fechaInicio)
```

```
            if ((r instanceof ReservaHabitacion) && (r.getNumHabitacion() == numHabitacion) &&
(r.getDiaInicio().isEqual(fechaInicio))){
                reserva = (ReservaHabitacion)r;
                flag = true;
            }
        }
        if (!flag) {
            System.out.println("La habitación indicada no tiene una reserva en vigencia o la fecha
de"
                    +"\ninicio de la reserva no coincide. Imposible generar factura.\n");

            return null;
        }
        //Si la habitación está reservada, tendrá un cliente válido porque se comprueba al hacer la
reserva
        cliente = reserva.getCliente();
        //Calculamos coste total (antes de posibles descuentos)
        costeTotal = reserva.getCosteTotal();
        //Pedir fecha del servicio
         System.out.println("Introduce la fecha de facturación (formato aaaa-mm-dd). La fecha de
facturación debe ser posterior a la fecha de salida de la habitación: \n");
        flag = true;
        do{
            try {
                br = new BufferedReader(new InputStreamReader(System.in),1);
                fechaFactura = LocalDate.parse(br.readLine());
                flag = false;
            } catch(Exception e) {
                System.out.println("\nError, fecha inválida. El formato de la fecha debe ser aaaa-mm-
dd (Ejemplo: 2000-10-25).\n");
            }
        }while(flag);
        if (fechaFactura.isBefore(reserva.getDiaFin())){
```

```
            System.out.println("\nError, fecha inválida. La fecha de la factura debe ser posterior al momento en que concluye la reserva.\n");
            return null;
        }

        //Aplicar descuento (5% a clientes que tuvieron reservas el mismo año natural)
        flag = false;
        for (Factura factura : Balneario.getInstancia().getFacturas()) {
            if ((factura.getCliente() == cliente) && (factura.getReserva().getDiaInicio().getYear() == fechaFactura.getYear())){
                flag = true;
            }
        }
        if (flag){
            costeTotal = costeTotal * 0.95f;
        }
        Factura factura = new Factura(idFactura, cliente, reserva, costeTotal, fechaFactura);
        return factura;
    }
}
```

## Clase FactoryReserva

Factoría encargada de la creación de objetos de las subclases de Reserva, decidiendo en virtud de los datos que solicita al usuario si crear objetos de tipo ReservaHabitación o ReservaSpa. Encapsula todas las peticiones de datos por teclado al usuario. Comprueba que los datos de entrada tengan un formato válido, y si el usuario los introduce mal se le vuelven a pedir inmediatamente sin necesidad de regresar al menú.

Sin embargo, cuando el usuario introduce determinados datos con formato correcto pero erróneos (ej: intentar hacer una reserva sobre una habitación que ya está reservada durante una determinada fecha), se devuelve al usuario al menú. Se trata de una elección de diseño deliberada, ya que podría suceder que el usuario no conozca qué fechas tienen reservas, y devolviéndolo al menú podrá investigar cuáles son los datos que le faltan. Se provee de un mensaje de error explicativo para que el usuario sepa en todo momento cuál ha sido el problema.

```
/**
```

```
 * Se encarga del proceso de creación de Reservas
 * @author Gustavo Cortés Jiménez.
 * @author Rodrigo Lázaro Escudero.
 */
public class FactoryReserva implements Factoria<Reserva>{

   @Override
   public Reserva getInstancia(String idReserva){
      final int MAX_PERSONAS = 10;
      int numReserva = Integer.parseInt(idReserva);
      int numHabitacion;
      LocalDate diaInicio = LocalDate.now();
      float coste = 0f;

      boolean flag = true;
      BufferedReader br;
      Scanner sc;
      byte opcion = 0;

      numHabitacion = Habitacion.pedirNumero();
      //Comprobamos que la habitación especificada esté registrada en el sistema
      Habitacion h = Balneario.getInstancia().buscarHabitacion(numHabitacion);
      if (h == null){
          System.out.println("La habitación indicada no está registrada en el sistema. Regístrela
primero antes de usarla.\n");
         return null;
      }
      //Haremos las comprobaciones de solapamiento de fechas en las reservas más adelante,
      //al recibir las fechas de fin de reserva y de servicio, respectivamente
      System.out.println("Introduce la fecha de inicio de la reserva (formato aaaa-mm-dd): \n");
      flag = true;
```

```
        do{
            try {
                br = new BufferedReader(new InputStreamReader(System.in),1);
                diaInicio = LocalDate.parse(br.readLine());
                flag = false;
            } catch(Exception e) {
                System.out.println("\nError, fecha inválida. El formato de la fecha debe ser aaaa-mm-
dd (Ejemplo: 2000-10-25).\n");
            }
        }while(flag);
        System.out.println("Introduce el tipo de reserva:\n"
                + "1. Reserva de habitación.\n"
                + "2. Reserva de servicio de Spa.\n");
        flag = true;
        do{
            try{
                sc = new Scanner(System.in);
                opcion = sc.nextByte();
                if ( (opcion < 1) || (opcion > 2)){
                    throw new Exception();
                }else{
                    flag = false;
                }
            }catch(Exception e){
                System.out.println("\nPor favor, introduce una opción válida.\n");
            }
        }while(flag);
        //Generamos el tipo de Reserva solicitado
        if (opcion == 2) {
            Servicio servicio = null;
            ReservaHabitacion reservaPadre = null;
```

```
int codigoServicio;
LocalDate diaServicio = LocalDate.now();
byte numPersonas = 0;


   //Comprobar lo primero de todo que la habitación indicada está reservada, y que la fecha de inicio de la reserva de
//spa coincide con la fecha de inicio de la reserva de la habitación
flag = false;
for (Reserva r : Balneario.getInstancia().getReservas()) {
   //r instanceof ReservaHabitacion
   //r.getNumHabitacion() == numHabitacion
   //diaInicio.isEqual(r.getDiaInicio())
    if ((r instanceof ReservaHabitacion) && (r.getNumHabitacion() == numHabitacion) && (diaInicio.isEqual(r.getDiaInicio()))){
       reservaPadre = (ReservaHabitacion)r;
       flag = true;
   }
}
if (!flag) {
     System.out.println("No se puede reservar servicio de spa para esa habitación en ese periodo. (Debes asegurarte de que la habitación está reservada y que"
             + "\nla fecha de inicio indicada para el servicio de spa coincide con la fecha de inicio de la habitación).\n");
   return null;
}
//Pedir código servicio y comprobar si existe
codigoServicio = Servicio.pedirId();
servicio = Balneario.getInstancia().buscarServicio(codigoServicio);
if (servicio == null){
     System.out.println("El servicio indicado no está registrado en el sistema. Regístrelo primero antes de usarlo.\n");
   return null;
}
```

```
        //Pedir fecha del servicio

         System.out.println("Introduce la fecha para reservar servicio de spa (formato aaaa-mm-dd): \n");

        flag = true;

        do{

           try {

              br = new BufferedReader(new InputStreamReader(System.in),1);

              diaServicio = LocalDate.parse(br.readLine());

              flag = false;

           } catch(Exception e) {

                 System.out.println("\nError, fecha inválida. El formato de la fecha debe ser aaaa-mm-dd (Ejemplo: 2000-10-25).\n");

           }

        }while(flag);

        //Comprobamos si el servicio ya está pedido ese día, y pedimos fecha distinta de ser así

        for (Reserva r : Balneario.getInstancia().getReservas()) {

           //r instanceof ReservaSpa

           //((ReservaSpa)r).getServicio().getCodigo() == codigoServicio

           //((ReservaSpa)r).getDiaServicio() == diaServicio

                 if ((r instanceof ReservaSpa) && (((ReservaSpa)r).getServicio().getCodigo() == codigoServicio) && (((ReservaSpa)r).getDiaServicio().isEqual(diaServicio))){

                System.out.println("\nEl servicio no puede ser reservado ese día porque ya existe una reserva previa. Elije una fecha distinta.\n");

              return null;

           }

        }

         //Comprobamos que la fecha del servicio esté dentro del intervalo en que la habitación ha sido reservada

        for (Reserva r: Balneario.getInstancia().getReservas()) {

            //Comprobamos sólo sobre reservas de habitaciones con el mismo número y con la fecha inicial adecuada

           if ((r instanceof ReservaHabitacion) && (r.getNumHabitacion() == numHabitacion) && (diaInicio.isEqual(r.getDiaInicio()))){
```

```
                //Comprobamos que la fecha del servicio esté comprendida en el periodo de reserva de la habitación
                if (!fechaEntre(diaServicio, (ReservaHabitacion)r)){
                    System.out.println("Error. La reserva de spa debe estar comprendida entre el día de inicio y el día final de la reserva de habitación asociada.\n");
                    return null;
                }
            }
        }
        //Pedir número de personas
        System.out.println("Introduce el número de personas que atenderán al servicio: \n");
        do{
            try{
                sc = new Scanner(System.in);
                numPersonas = sc.nextByte();
                if ( (numPersonas < 1) || (numPersonas > MAX_PERSONAS)){
                    throw new Exception();
                }else{
                    flag = false;
                }
            }catch(Exception e){
                    System.out.println("\nPor favor, introduce un número entre 1 y " + MAX_PERSONAS + " (inclusive).\n");
            }
        }while(flag);
                ReservaSpa reserva = new ReservaSpa(numReserva, numHabitacion, diaInicio, servicio.getCoste(), diaServicio, servicio, numPersonas, reservaPadre);
        //Referenciamos la ReservaSpa creada en su ReservaHabitacion padre
        reservaPadre.agregar(reserva);
        return (Reserva)reserva;
    } else { //opcion == 1
        System.out.println("¿A nombre de quién se hace la reserva?\n");
```

```
String idCliente = Cliente.pedirId();
LocalDate diaFin = LocalDate.now();

//Comprobamos que el DNI esté registrado en el sistema
Cliente c = Balneario.getInstancia().buscarCliente(idCliente);
if (c == null){
  //Si el cliente no existe, indicamos al usuario que lo cree primero
      System.out.println("El DNI del cliente indicado no está registrado en el sistema. Regístrelo antes de hacer la reserva a su nombre.\n");
  return null;
}
 System.out.println("Introduce la fecha de finalización de la reserva (formato aaaa-mm-dd): \n");
flag = true;
do{
  try {
    br = new BufferedReader(new InputStreamReader(System.in),1);
    diaFin = LocalDate.parse(br.readLine());
    if (!(diaInicio.isBefore(diaFin))) {
      throw new Exception();
    }
    flag = false;
  } catch(Exception e) {
       System.out.println("\nLa fecha de fin de reserva debe ser superior a la fecha de inicio. "
         + "El formato de la fecha debe ser aaaa-mm-dd (Ejemplo: 2000-10-25).\n");
  }
}while(flag);

//Comprobaciones adicionales de solapamientos de fechas
for (Reserva r: Balneario.getInstancia().getReservas()) {
```

```
                //Comprobamos sólo sobre reservas de habitaciones con el mismo número que pudiesen generar conflictos
            if ((r instanceof ReservaHabitacion) && (r.getNumHabitacion() == numHabitacion)){
              //Comprobamos que ni la fecha de inicio ni la fecha de final estén comprendidas en periodos ocupados por otras reservas
                                if (fechaEntre(diaInicio, (ReservaHabitacion)r) || fechaEntre(diaFin, (ReservaHabitacion)r)){
                    System.out.println("Error. La habitación ya estaba reservada para ese periodo. Elija fechas no ocupadas.\n");
                  return null;
                }
              }
          }
          //Buscamos el coste de la habitación
          Habitacion hab = Balneario.getInstancia().buscarHabitacion(numHabitacion);
          coste = hab.getPrecio();
              ReservaHabitacion reserva = new ReservaHabitacion(numReserva, numHabitacion, diaInicio, coste, c, diaFin);
          return (Reserva)reserva;
      }
    }

    /**
     * Comprueba si una fecha está comprendida entre el día de inicio y el día de finalización de la ReservaHabitacion.
     * @param fecha a evaluar, ReservaHabitación rh proporciona el intervalo de fechas
     * @return true si la fecha está entre inicio y el final de la reserva rh indicada; si no, false.
     */
    public static boolean fechaEntre(LocalDate fecha, ReservaHabitacion rh) {
            //Devolvemos negación de OR en vez de comprobación sobre AND para evitar solapamientos en los días de inicio y final
        return !(fecha.isBefore(rh.getDiaInicio()) || fecha.isAfter(rh.getDiaFin()));
    }
}
```

## Clase FactoryHabitacion

Factoría encargada de la creación de objetos de clase Habitacion. Encapsula todas las peticiones de datos por teclado al usuario. Debido a la independencia de Habitacion respecto a otros datos del balneario, sólo se comprueba que los datos de entrada tengan un formato válido. Si el usuario los introduce mal, se le vuelven a pedir inmediatamente sin necesidad de regresar al menú.

```
/**
 * Se encarga del proceso de creación de Habitaciones
 * @author Gustavo Cortés Jiménez.
 * @author Rodrigo Lázaro Escudero.
 */
public class FactoryHabitacion implements Factoria<Habitacion>{
   @Override
   public Habitacion getInstancia(String numero){
      float precio = 0;
      boolean flag = true;
      byte respuesta = 0;

      System.out.println("Introduce el precio de la habitación (usar coma)");
      do{
         flag = true;
         try{
            Scanner sc = new Scanner(System.in);
            precio = sc.nextFloat();
            if (precio < 0){
               System.out.println("El precio tiene que ser superior a 0");
            }else{
               flag = false;
            }
         }catch(Exception e){
            System.out.println("El valor introducido no es un numero");
```

```
    }
}while(flag);
Habitacion habit = new Habitacion(Integer.valueOf(numero), precio);
System.out.println("Introduce el tipo de habitación:\n"
        + "1. Habitación individual\n"
        + "2. Habitación doble\n"
        + "3. Habitación familiar\n");
flag = true;
do{
    try{
        Scanner sc = new Scanner(System.in);
        respuesta = sc.nextByte();
        if ( (respuesta < 1) || (respuesta > 3)){
            throw new Exception();
        }else{
            flag = false;
        }
    }catch(Exception e){
        System.out.println("\nPor favor, introduce una opción correcta\n");
    }
}while(flag);

switch (respuesta){
    case 2:
        String compartido = "";
        flag = true;
        System.out.println("\n¿El baño está compartido? (s/n): \n");
        do{
            try{
                Scanner sc = new Scanner(System.in);
                compartido = sc.next("[ns]");
```

```
                flag = false;
            }catch(Exception e){
                System.out.println("\nPor favor, introduce \"s\" o \"n\"\n");
            }
        }while(flag);
        habit = new Doble(habit, (compartido.equals("s")));
        break;
      case 3:
        String camas = "";
        flag = true;
        System.out.println("\n¿Tiene camas supletorias? (s/n): \n");
        do{
            try{
                Scanner sc = new Scanner(System.in);
                camas = sc.next("[ns]");
                flag = false;
            }catch(Exception e){
                System.out.println("\nPor favor, introduce \"s\" o \"n\"\n");
            }
        }while(flag);
        habit = new Familiar(habit, (camas.equals("s")));
        break;
    }
    return habit;
  }
}
```

## Clase abstracta Menu

Se trata de la superclase encargada de la gestión de los distintos menús. Las subclases implementan su método abstracto "opciones". Junto con sus subclases, implementa los patrones Strtegy y Template Method.

```
/**
 * Muestra y gestiona las funciones con las que puede interactuar el usuario.
 * @author Gustavo Cortés Jiménez
 * @author Rodrigo Lázaro Escudero
 */
public abstract class Menu {
    private final Balneario balneario;
    private final String mensaje;
    private final int max;

    /**
     *
     * @param mensaje Opciones que muestra el menú
     * @param max número máximo de opción
     */
    public Menu(String mensaje, int max){
        balneario = Balneario.getInstancia();
        this.mensaje = mensaje;
        this.max = max;
    }

    //getters
    public Balneario getBalneario() {
        return balneario;
    }
    public String getMensaje() {
        return mensaje;
    }
    public int getMax() {
        return max;
    }
```

```
/**
 * Muestra las opciones disponibles, y ejecuta una de ellas según la respuesta recibida
 */
public void menu(){
   byte respuesta = 0;
   boolean flag = true;
   do{
      System.out.print(getMensaje());
      do{
         try{
            flag = true;
            Scanner sc = new Scanner(System.in);
            respuesta = sc.nextByte();
            if( (respuesta > getMax()) || (respuesta < 0)){
               throw new Exception();
            }else{
               flag = false;
            }
         }catch(Exception e){
            System.out.println("\nPor favor, introduce una opción válida.\n");
         }
      }while(flag);
      opciones(respuesta);
   }while(respuesta != 0);
}

/**
 * Llama a la función correspondiente a la respuesta recibida
 * @param respuesta la opción elegida por el usuario
 * @pre respuesta debe estar entre 0 y max
```

```
    */
    public abstract void opciones(byte respuesta);
}
```

## Clase MenuClientes

Se trata de la clase encargada de la gestión del menú de Clientes. Ofrece las opciones de lectura, creación y listado completo sobre el ArrayList de Clientes almacenado en el Balneario. No proporciona funcionalidad de eliminación de Clientes porque el guion de la práctica no lo indica. Junto con las demás clases del paquete menus, implementa los patrones Strategy y Template Method.

```
/**
 * Permite al usuario acceder a las funciones que actúan sobre los clientes.
 * @author Gustavo Cortés Jiménez
 * @author Rodrigo Lázaro Escudero
 */
public class MenuClientes extends Menu{
    public MenuClientes(){
        super("1. Listado de todos los clientes\n"
                + "2. Información de un cliente\n"
                + "3. Añadir un cliente\n"
                + "4. Guardar cambios\n"
                + "0. Volver\n"
                + "\nElige una opción: ", 4);
    }

    /**
     * Muestra los datos de todos los clientes guardados
     */
    private void listaClientes(){
        if(getBalneario().getClientes().isEmpty()){
            System.out.println("\nNo hay clientes registrados.\n");
        }else{
```

```
        for(Cliente c: getBalneario().getClientes()){
            System.out.println(c.infoCliente());
        }
    }
}

/**
 * Muestra los datos de un cliente
 * @param dni El DNI del usuario que mostrar
 */
private void listaClientes(String dni){
    Cliente c = getBalneario().buscarCliente(dni);
    if (c != null){
        System.out.println(c.infoCliente());
        return;
    }
    System.out.println("No existe ningún cliente con el DNI indicado.\n");
}

/**
 * Crea y añade un nuevo cliente al sistema.
 */
private void agregarCliente(){
    String idCliente;
    FactoryCliente fc;
    //System.out.println("Introduce el DNI del cliente:\n"); //Ya lo pide Cliente.pedirId
    idCliente = Cliente.pedirId();
    //Comprobamos que el DNI no esté ya en el sistema
    Cliente c = getBalneario().buscarCliente(idCliente);
    if (c != null){
        System.out.println("El DNI indicado ya está registrado en el sistema.\n");
```

```
            return;
        }
        fc = new FactoryCliente();
        getBalneario().getClientes().add(fc.getInstancia(idCliente));
        System.out.println("Cliente añadido.\n");
    }

    @Override
    public void opciones(byte respuesta){
        switch (respuesta){
            case 1:
                listaClientes();
                break;
            case 2:
                listaClientes(Cliente.pedirId());
                break;
            case 3:
                agregarCliente();
                break;
            case 4:
                getBalneario().guardarDatos();
                break;
        }
    }
}
```

## Clase MenuServicios

Se trata de la clase encargada de la gestión del menú de Servicios. Ofrece las opciones de lectura, creación, eliminación y listado completo sobre el ArrayList de Clientes almacenado en el Balneario.

La política de diseño seguida a la hora de eliminar un Servicio para el cuál existe una Reserva en vigencia es eliminar el servicio, pero no la Reserva asociada. La lógica de esta decisión obedece a las necesidades de facturación del Balneario. Aunque se elimine un servicio y éste

ya no esté disponible para futuras reservas, los datos del mismo deben seguir disponibles para poder expedir una factura. El usuario podrá eliminar manualmente las reservas necesarias en los casos en que no proceda tal consideración.

Junto con las demás clases del paquete menus, implementa los patrones Strategy y Template Method.

```
/**
 * Permite al usuario acceder a las funciones que actúan sobre los servicios.
 * @author Gustavo Cortés Jiménez
 * @author Rodrigo Lázaro Escudero
 */
public class MenuServicios extends Menu{

   public MenuServicios(){
      super("1. Listado de todos los servicios\n"
            + "2. Información de un servicio\n"
            + "3. Añadir un servicio\n"
            + "4. Eliminar un servicio\n"
            + "5. Guardar cambios\n"
            + "0. Volver\n"
            + "\nElige una opción: ", 5);

   }

   /**
    * Muestra los datos de todos los servicios guardados
    */
   private void listaServicios() {
      if(getBalneario().getServicios().isEmpty()){
         System.out.println("\nNo hay servicios registrados.\n");
      }else{
         for(Servicio s: getBalneario().getServicios()){
```

```
            System.out.println(s.infoServicio());
        }
    }
}

/**
 * Muestra los datos de un servicio
 * @param codigo El codigo del servicio que mostrar
 */
private void listaServicios(int codigo) {
    Servicio s = getBalneario().buscarServicio(codigo);
    if (s != null){
        System.out.println(s.infoServicio());
        return;
    }
    System.out.println("No existe ningún servicio con el código indicado.\n");
}

/**
 * Crea y añade un nuevo servicio al sistema.
 */
private void agregarServicio(){
    int idServicio = Servicio.codigoMax++;
    FactoryServicio fs = new FactoryServicio();
    getBalneario().getServicios().add(fs.getInstancia(Integer.toString(idServicio)));
    System.out.println("Servicio añadido.\n");
}

/**
 * Elimina un servicio almacenado en el sistema
 * @param id el código del servicio que eliminar
```

```
    */
    private void eliminarServicio(int id){
        //Se pueden borrar servicios y habitaciones sin comprometer la integridad estructural, ya
que
        //sólo se eliminan las referencias, no los objetos. Éstos quedan vinculados a las reservas y/
o
        //a las facturas, manteniendo así el registro histórico y evitando null point exceptions.
        Servicio s = getBalneario().buscarServicio(id);
        if (s != null){
            getBalneario().getServicios().remove(s);
            return;
        }
        System.out.println("No existe ningún servicio con el código indicado.\n");
    }

    @Override
    public void opciones(byte respuesta){
        switch (respuesta){
            case 1:
                listaServicios();
                break;
            case 2:
                listaServicios(Servicio.pedirId());
                break;
            case 3:
                agregarServicio();
                break;
            case 4:
                eliminarServicio(Servicio.pedirId());
                break;
            case 5:
                getBalneario().guardarDatos();
```

```
            break;
        }
    }
}
```

## Clase MenuFactura

Se trata de la clase encargada de la gestión del menú de Facturas. Ofrece las opciones de lectura, creación y listado completo sobre el ArrayList de Facturas almacenado en el Balneario. No proporciona funcionalidad de eliminación de Facturas porque el guion de la práctica no lo indica. Junto con las demás clases del paquete menus, implementa los patrones Strategy y Template Method.

```
/**
 * Permite al usuario acceder a las funciones que actúan sobre las facturas.
 * @author Gustavo Cortés Jiménez
 * @author Rodrigo Lázaro Escudero
 */
public class MenuFacturas extends Menu{
    public MenuFacturas(){
        super("1. Listado de todas las facturas\n"
                + "2. Información de una factura\n"
                + "3. Generar una factura\n"
                + "4. Guardar cambios\n"
                + "0. Volver\n"
                + "\nElige una opción: ", 4);
    }

    /**
     * Muestra los datos de todas las facturas guardadas
     */
    private void listaFacturas(){
        if(getBalneario().getFacturas().isEmpty()){
            System.out.println("\nNo hay facturas registradas.\n");
```

```
    }else{
        for(Factura f: getBalneario().getFacturas()){
            System.out.println(f.infoFactura());
        }
    }
}

/**
 * Muestra los datos de una factura
 * @param codigo El código de la factura que mostrar
 */
private void listaFacturas(String codigo){
    Factura f = getBalneario().buscarFacura(codigo);
    if (f != null){
        System.out.println("Código de factura: " + codigo
            + "\nInformación del cliente: " + f.getCliente().infoCliente()
            + "\nListado de reservas: " + f.getReserva().infoReserva()
            + "\nCoste total: " + f.getReserva().getCosteTotal()
            + "\nFecha de facturación: " + f.getFechaFactura() + "\n");
        return;
    }
    System.out.println("No existe ninguna factura con el código indicado.\n");
}

/**
 * Crea y añade una nueva factura al sistema
 */
private void generarFactura(){
    int idFactura = 0;
    FactoryFactura ff;
```

```
    //Generamos automáticamente un Id nuevo y no repetido
    boolean duplicado = false;
    do {
      for(Factura f: getBalneario().getFacturas()){
        if(idFactura == Integer.parseInt(f.getCodigo())) {
          duplicado = true;
          idFactura++;
        } else {
          duplicado = false;
        }
      }
    } while(duplicado);
    ff = new FactoryFactura();
    Factura factura = ff.getInstancia(String.format("%08d", idFactura));
    if (factura != null) {
      getBalneario().getFacturas().add(factura);
      System.out.println("Factura generada con éxito.\n");
    } else {
            System.out.println("La factura NO se ha creado. Por favor verifique que las
habitaciones, clientes y reservas indicadas "
          + "estén registradas previamente en el sistema.\n");
    }
  }

  @Override
  public void opciones(byte respuesta){
    switch (respuesta){
      case 1:
        listaFacturas();
        break;
      case 2:
```

```
            listaFacturas(Factura.pedirId());
            break;
        case 3:
            generarFactura();
            break;
        case 4:
            getBalneario().guardarDatos();
            break;
    }
  }
}
```

## Clase MenuReservas

Se trata de la clase encargada de la gestión del menú de Reservas. Ofrece las opciones de lectura, creación, eliminación y listado completo sobre el ArrayList de Reservas almacenado en el Balneario.

La política de diseño seguida a la hora de eliminar una Reserva en vigencia es simplemente eliminar dicha reserva en caso de que sea de tipo ReservaSpa; o si es una reserva de habitación, su eliminación conllevará la eliminación de las reservas de spa asociadas a la misma.

Junto con las demás clases del paquete menus, implementa los patrones Strategy y Template Method.

```
/**
 * Permite al usuario acceder a las funciones que actúan sobre las reservas.
 * @author Gustavo Cortés Jiménez
 * @author Rodrigo Lázaro Escudero
 */
public class MenuReservas extends Menu{
  public MenuReservas(){
    super("1. Listado de todas las reservas\n"
          + "2. Información de una reserva\n"
          + "3. Añadir una reserva\n"
```

```
            + "4. Eliminar una reserva\n"
            + "5. Guardar cambios\n"
            + "0. Volver\n"
            + "\nElige una opción: ", 5);

}

/**
 * Muestra los datos de todas las reservas guardadas
 */
private void listaReservas(){
   if(getBalneario().getReservas().isEmpty()){
      System.out.println("\nNo hay reservas registradas.\n");
   }else{
      for(Reserva r: getBalneario().getReservas()){
         System.out.println(r.infoReserva());
      }
   }
}

/**
 * Muestra los datos de una reserva
 * @param numReserva El número de la reserva que mostrar
 */
private void listaReservas(int numReserva){
   Reserva r = getBalneario().buscarReserva(numReserva);
   if (r != null){
      System.out.println(r.infoReserva());
      return;
   }
   System.out.println("No existe ninguna reserva con el número indicado.\n");
```

```
    }

    /**
     * Crea y añade una nueva reserva al sistema.
     */
    private void agregarReserva(){
        int idReserva = Reserva.numMaxReserva++;
        FactoryReserva fr = new FactoryReserva();

        Reserva reserva = fr.getInstancia(Integer.toString(idReserva));
        if (reserva != null) {
            getBalneario().getReservas().add(reserva);
            System.out.println("Reserva añadida.\n");
        } else {
                    System.out.println("La reserva NO se ha creado. Por favor verifique que las habitaciones, clientes y servicios necesarios "
                + "estén registrados previamente en el sistema.\n");
            Reserva.numMaxReserva--;
        }
    }

    /**
     * Elimina una reserva almacenada en el sistema
     */
    private void eliminarReserva(int numReserva){
        Reserva r = getBalneario().buscarReserva(numReserva);
        if (r != null){
            //Si es una ReservaSpa, eliminar de la lista del padre
            if (r instanceof ReservaSpa) {
                ReservaHabitacion padre = ((ReservaSpa) r).getPadre();
                padre.eliminar(r);
```

```
        } else { //Si es una ReservaHabitacion, eliminar todas las reservasSpa que contiene
            for (Reserva rs: ((ReservaHabitacion)r).getReservasSpa()) {
                getBalneario().getReservas().remove(rs);
            }
        }
        //Finalmente, eliminar el propio nodo
        getBalneario().getReservas().remove(r);
        return;
    }
    System.out.println("No existe ninguna reserva con el código indicado.\n");
  }

  @Override
  public void opciones(byte respuesta){
    switch (respuesta){
        case 1:
            listaReservas();
            break;
        case 2:
            listaReservas(Reserva.pedirId());
            break;
        case 3:
            agregarReserva();
            break;
        case 4:
              System.out.println("ATENCIÓN: eliminar una reserva de habitación eliminará todos
los servicios de Spa asociados a esa reserva.");
            eliminarReserva(Reserva.pedirId());
            break;
        case 5:
            getBalneario().guardarDatos();
```

```
            break;
        }
    }
}
```

## Clase MenuHabitaciones

Se trata de la clase encargada de la gestión del menú de Habitaciones. Ofrece las opciones de lectura, creación, eliminación y listado completo sobre el ArrayList de Habitaciones almacenado en el Balneario.

La política de diseño seguida a la hora de eliminar una Habitacion para el cuál existe una Reserva en vigencia es eliminar la habitación, pero no la Reserva asociada. La lógica de esta decisión obedece a las necesidades de facturación del Balneario. Aunque se elimine una habitación y ésta ya no esté disponible para futuras reservas, los datos de la misma deben seguir disponibles para poder expedir una factura. El usuario podrá eliminar manualmente las reservas necesarias en los casos en que no proceda tal consideración.

Junto con las demás clases del paquete menus, implementa los patrones Strategy y Template Method.

```
/**
 * Permite al usuario acceder a las funciones que actúan sobre las habitaciones.
 * @author Gustavo Cortés Jiménez
 * @author Rodrigo Lázaro Escudero
 */
public class MenuHabitaciones extends Menu{
    public MenuHabitaciones(){
        super("1. Listado de todas las habitaciones\n"
                + "2. Información de una habitación\n"
                + "3. Añadir una habitación\n"
                + "4. Eliminar una habitación\n"
                + "5. Guardar cambios\n"
                + "0. Volver\n"
                + "\nElige una opción: ", 5);
    }
```

```
/**
 * Muestra los datos de todas las habitaciones guardadas
 */
private void listaHabitaciones(){
   if(getBalneario().getHabitaciones().isEmpty()){
      System.out.println("\nNo hay habitaciones registradas\n");
   }else{
      for (Habitacion h: getBalneario().getHabitaciones()){
         h.infoHabitacion();
         System.out.println();
      }
   }
}

/**
 * Muestra los datos de una habitación
 * @param codigo El código de la factura que mostrar
 */
private void listaHabitaciones(int numHabitacion){
   Habitacion habit = getBalneario().buscarHabitacion(numHabitacion);
   if (habit != null){
      habit.infoHabitacion();
         System.out.println("");
         return;
   }
   System.out.println("\nLa habitación no está registrada\n");
}

/**
 * Crea y añade una nueva habitación al sistema.
 */
```

```
private void agregarHabitacion(){
   int numero = 0;
   boolean flag = true;
   do{
      numero = Habitacion.pedirNumero();
      if (getBalneario().buscarHabitacion(numero) != null){
         System.out.println("Ya hay registrada una habitación con ese número");
      }else{
         flag = false;
      }
   }while(flag);
   Habitacion fh = new FactoryHabitacion().getInstancia(String.valueOf(numero));
   getBalneario().getHabitaciones().add(fh);
   System.out.println("Habitación añadida.");
}

/**
 * Elimina una habitación almacenada en el sistema
 */
private void eliminarHabitacion(){
   boolean flag = true;
   Habitacion habit = null;
   do{
      habit = getBalneario().buscarHabitacion(Habitacion.pedirNumero());
      if (habit == null){
         System.out.println("No hay registrada una habitación con ese número");
      }else{
         flag = false;
      }
   }while(flag);
   habit.infoHabitacion();
```

```
        System.out.println("\n¿Seguro que quieres borrar esta habitación? (s/n)");
        Scanner sc = new Scanner(System.in);
        if (sc.next().equals("s")){
            getBalneario().getHabitaciones().remove(habit);
            System.out.println("Se ha eliminado la habitación");
        }else{
            System.out.println("Operación cancelada");
        }
    }

    @Override
    public void opciones(byte respuesta){
        switch (respuesta){
            case 1:
                listaHabitaciones();
                break;
            case 2:
                listaHabitaciones(Habitacion.pedirNumero());
                break;
            case 3:
                agregarHabitacion();
                break;
            case 4:
                eliminarHabitacion();
                break;
            case 5:
                getBalneario().guardarDatos();
                break;
        }
    }
}
```

## Clase MenuPrincipal

Contiene el menú inicial que se mostrará al usuario al arrancar la aplicación, así como el método main.

Junto con las demás clases del paquete menus, implementa los patrones Strategy y Template Method.

```
/**
 * Permite al usuario acceder al resto de menús.
 * @author Gustavo Cortés Jiménez
 * @author Rodrigo Lázaro Escudero
 */
public class MenuPrincipal extends Menu{
   private final ArrayList<Menu> menus = new ArrayList();

   public MenuPrincipal(){
      super("1. Habitaciones\n"
            + "2. Clientes\n"
            + "3. Servicios\n"
            + "4. Reservas\n"
            + "5. Facturas\n"
            + "0. Salir\n"
            + "\nElige una opción: ", 5);
      menus.add(new MenuHabitaciones());
      menus.add(new MenuClientes());
      menus.add(new MenuServicios());
      menus.add(new MenuReservas());
      menus.add(new MenuFacturas());
   }

   /**
    * Carga los datos del archivo de configuración, y llama al menú principal
```

```
     */
    public void iniciar(){
        if (!getBalneario().cargarDatos()){
            Servicio.codigoMax = 0;
            Reserva.numMaxReserva = 0;
        }
        menu();
    }

    /**
     * Guarda los datos del programa en el archivo de configuración
     */
    public void cerrar(){
        getBalneario().guardarDatos();
        System.out.println("Gracias por usar esta aplicación.");
    }

    @Override
    public void opciones(byte respuesta){
        if(respuesta != 0){
            menus.get(respuesta - 1).menu();
        }else{
            cerrar();
        }
    }
    public static void main(String[] args) {
        MenuPrincipal menu = new MenuPrincipal();
        menu.iniciar();
    }
}
```

## Patrones

Para la implementación del patrón Singleton, usado para la clase Balneario, el procedimiento que hemos seguido es asegurarnos de su constructor sea privado, evitando así su invocación por otras clases. Luego hemos añadido un método estático getInstancia() que comprueba si ya existe una instancia de Balneario, creándola si no existe o recuperando la existente si ya la hay. Por último, sobrecargamos el método clone() para evitar que otras clases puedan duplicar el Balneario.

Para el funcionamiento de los menús, hemos implementado los patrones Template Method y Strategy.

Para el primero, hemos definido la clase abstracta Menu, con la estructura que deben tener todos los menús, siendo la función opciones() es un placeholder que debe ser modificado por sus subclases.

Para la implementación del patrón Strategy, hemos definido en la clase MenuPrincipal un ArrayList con el resto de los menús. Esto nos permite elegir dinámicamente qué menú mostrar, ya que todos contienen las funciones menú() y opciones(), con similar comportamiento.

Se ha seguido el patrón Decorator para definir los diferentes tipos de habitaciones. Para ello, hemos definido una clase Habitación, representando una habitación individual; y una clase abstracta DecoradorHabitacion, subclase de Habitación, cuyo objetivo es agregar funcionalidad a esta. Para ello, guarda la Habitación correspondiente como atributo, modifica las funciones de acceso a sus atributos, y agrega nuevas funciones como atributos, mediante las clases Doble y Familiar.

Hemos empleado el patrón Composite para estructurar las clases Reserva, ReservaHabitacion (Compuesto) y ReservaSpa (Hoja) porque presentan entre sí una estructura jerárquica y recursiva (ReservaHabitacion contiene varias ReservaSpa). El procedimiento empleado es el siguiente: en primer lugar, la clase Reserva incluye varios métodos abstractos que las clases hijas implementan de modo distinto, pero que presentan características recursivas. Después, la clase de tipo Compuesto (ReservaHabitación) se puebla con componentes de tipo Hoja (ReservaSpa).

Para implementar el patrón FactoryMethod, hemos empleado una interfaz Factory en la que definimos el método getinstancia(String s), que las factorías concretas individuales implementan en cada caso. Con ello logramos hacer uniformes la creación de objetos (todos ellos reciben un parámetro de tipo string), encapsulamosy delegamos en las distintas factorías los detalles y requisitos específicos de cada tipo de objeto. El código interno de las factorías es complejo, pero su invocación por otras clases se simplifica mucho y resulta más modular.

## Casos de prueba

Por cuestiones de espacio y legibilidad, se agrupan varias pruebas en una sola imagen, situada a la izquierda de la página. Se ofrecerá una breve explicación de la prueba realizada y a continuación se incluirán las capturas que lo acreditan. Las imágenes son capturas de pantalla de la salida por pantalla que ofrece la aplicación. Para poder adaptar el formato de las capturas a la página, en ciertos casos se cortarán las líneas que sean demasiado largas y se pondrán debajo. Se incluirá con la entrega del código el archivo Configuracion.dat poblado con los datos resultantes tras realizar estas pruebas.

Probamos que las funcionalidades del menú de Habitaciones funcionan correctamente.

```
Datos cargados
1. Habitaciones
2. Clientes
3. Servicios
4. Reservas
5. Facturas
0. Salir

Elige una opción: 1
1. Listado de todas las habitaciones
2. Información de una habitación
3. Añadir una habitación
4. Eliminar una habitación
5. Guardar cambios
0. Volver

Elige una opción: 1
Habitacion doble
Numero de habitación: 111
Precio de la habitación: 115.0
El baño es privado

Habitacion individual
Numero de habitación: 222
Precio de la habitación: 50.0

Habitacion familiar
Numero de habitación: 333
Precio de la habitación: 130.5
Tiene camas supletorias
```

```
1. Listado de todas las habitaciones
2. Información de una habitación
3. Añadir una habitación
4. Eliminar una habitación
5. Guardar cambios
0. Volver

Elige una opción: 2
Introduce un número de habitación:
333
Habitacion familiar
Numero de habitación: 333
Precio de la habitación: 130.5
Tiene camas supletorias

1. Listado de todas las habitaciones
2. Información de una habitación
3. Añadir una habitación
4. Eliminar una habitación
5. Guardar cambios
0. Volver

Elige una opción: 3
Introduce un número de habitación:
444
Introduce el precio de la habitación (usar coma)
91,2
Introduce el tipo de habitación:
1. Habitación individual
2. Habitación doble
3. Habitación familiar

1
Habitación añadida.
```

```
1. Listado de todas las habitaciones
2. Información de una habitación
3. Añadir una habitación
4. Eliminar una habitación
5. Guardar cambios
0. Volver

Elige una opción: 2
Introduce un número de habitación:
444
Habitacion individual
Numero de habitación: 444
Precio de la habitación: 91.2

1. Listado de todas las habitaciones
2. Información de una habitación
3. Añadir una habitación
4. Eliminar una habitación
5. Guardar cambios
0. Volver

Elige una opción: 4
Introduce un número de habitación:
444
Habitacion individual
Numero de habitación: 444
Precio de la habitación: 91.2

¿Seguro que quieres borrar esta habitación? (s/n)
s
Se ha eliminado la habitación
1. Listado de todas las habitaciones
2. Información de una habitación
3. Añadir una habitación
4. Eliminar una habitación
5. Guardar cambios
0. Volver

Elige una opción: 2
Introduce un número de habitación:
444

La habitación no está registrada
```

Probamos que las funcionalidades del menú de Clientes funcionan correctamente.

```
Datos cargados
1. Habitaciones
2. Clientes
3. Servicios
4. Reservas
5. Facturas
0. Salir

Elige una opción: 2
1. Listado de todos los clientes
2. Información de un cliente
3. Añadir un cliente
4. Guardar cambios
0. Volver

Elige una opción: 1
DNI: 12345678a
Nombre y Apellidos: Pedro Palanca
Teléfono Móvil: 123456789

DNI: 87654321z
Nombre y Apellidos: Marco Machado
Teléfono Móvil: 987654321

DNI: 11223344b
Nombre y Apellidos: Hugo Hierro
Teléfono Móvil: 999999999

DNI: 44332211y
Nombre y Apellidos: Antonio Andrada
Teléfono Móvil: 666666666

1. Listado de todos los clientes
2. Información de un cliente
3. Añadir un cliente
4. Guardar cambios
0. Volver
```

```
Elige una opción: 2
Introduce el DNI del cliente:
12345678a
DNI: 12345678a
Nombre y Apellidos: Pedro Palanca
Teléfono Móvil: 123456789

1. Listado de todos los clientes
2. Información de un cliente
3. Añadir un cliente
4. Guardar cambios
0. Volver

Elige una opción: 3
Introduce el DNI del cliente:
88776655x
Introduce el nombre y apellidos del cliente:

Renato Revuelta
Introduce el número de teléfono móvil del cliente:

678678678
Cliente añadido.

1. Listado de todos los clientes
2. Información de un cliente
3. Añadir un cliente
4. Guardar cambios
0. Volver

Elige una opción: 2
Introduce el DNI del cliente:
88776655x
DNI: 88776655x
Nombre y Apellidos: Renato Revuelta
Teléfono Móvil: 678678678
```

Probamos que las funcionalidades del menú de Servicios funcionan correctamente.

```
Datos cargados
1. Habitaciones
2. Clientes
3. Servicios
4. Reservas
5. Facturas
0. Salir

Elige una opción: 3
1. Listado de todos los servicios
2. Información de un servicio
3. Añadir un servicio
4. Eliminar un servicio
5. Guardar cambios
0. Volver

Elige una opción: 1
Código de servicio: 0
Descripción: Masaje relajante
Coste: 50.5 euros.

Código de servicio: 1
Descripción: Cura de aguas
Coste: 65.0 euros.

Código de servicio: 2
Descripción: Tratamiento facial con pepino
Coste: 75.0 euros.

1. Listado de todos los servicios
2. Información de un servicio
3. Añadir un servicio
4. Eliminar un servicio
5. Guardar cambios
0. Volver

Elige una opción: 2
Introduce el código del servicio:
1
Código de servicio: 1
Descripción: Cura de aguas
Coste: 65.0 euros.

1. Listado de todos los servicios
2. Información de un servicio
3. Añadir un servicio
4. Eliminar un servicio
5. Guardar cambios
0. Volver

Elige una opción: 3
Introduce una descripción del servicio:

Masaje con aceite de pomelo
Introduce el coste del servicio (usar coma):

80
Servicio añadido.

1. Listado de todos los servicios
2. Información de un servicio
3. Añadir un servicio
4. Eliminar un servicio
5. Guardar cambios
0. Volver

Elige una opción: 1
Código de servicio: 0
Descripción: Masaje relajante
Coste: 50.5 euros.

Código de servicio: 1
Descripción: Cura de aguas
Coste: 65.0 euros.

Código de servicio: 2
Descripción: Tratamiento facial con pepino
Coste: 75.0 euros.

Código de servicio: 3
Descripción: Masaje con aceite de pomelo
Coste: 80.0 euros.

1. Listado de todos los servicios
2. Información de un servicio
3. Añadir un servicio
4. Eliminar un servicio
5. Guardar cambios
0. Volver

Elige una opción: 4
Introduce el código del servicio:
1
1. Listado de todos los servicios
2. Información de un servicio
3. Añadir un servicio
4. Eliminar un servicio
5. Guardar cambios
0. Volver

Elige una opción: 3
Introduce una descripción del servicio:

Cura de aguas
Introduce el coste del servicio (usar coma):

60
Servicio añadido.

1. Listado de todos los servicios
2. Información de un servicio
3. Añadir un servicio
4. Eliminar un servicio
5. Guardar cambios
0. Volver

Elige una opción: 1
Código de servicio: 0
Descripción: Masaje relajante
Coste: 50.5 euros.

Código de servicio: 2
Descripción: Tratamiento facial con pepino
Coste: 75.0 euros.

Código de servicio: 3
Descripción: Masaje con aceite de pomelo
Coste: 80.0 euros.

Código de servicio: 4
Descripción: Cura de aguas
Coste: 60.0 euros.
```

Probamos que las funcionalidades del menú de Reservas funcionan correctamente.

```
Datos cargados
1. Habitaciones
2. Clientes
3. Servicios
4. Reservas
5. Facturas
0. Salir

Elige una opción: 4
1. Listado de todas las reservas
2. Información de una reserva
3. Añadir una reserva
4. Eliminar una reserva
5. Guardar cambios
0. Volver

Elige una opción: 1
Identificador de reserva: 0
Número de habitación: 111
Día de inicio: 2021-01-01
Coste de la habitación: 115.0
Coste total de habitación + servicios: 115.0
Día de fin de reserva: 2021-02-01
Servicios de Spa asociados:

Identificador de reserva: 1
Número de habitación: 222
Día de inicio: 2021-01-01
Coste de la habitación: 50.0
Coste total de habitación + servicios: 100.5
Día de fin de reserva: 2021-03-01
Servicios de Spa asociados:
Identificador de reserva: 3
Código del servicio: 0
Descripción del servicio: Masaje relajante
Coste base del servicio: 50.5
Coste total del servicio (coste base * número de personas): 151.5
Día servicio: 2021-01-10
Número de personas: 3

Identificador de reserva: 2
Número de habitación: 333
Día de inicio: 2021-04-01
Coste de la habitación: 130.5
Coste total de habitación + servicios: 130.5
Día de fin de reserva: 2021-05-01
Servicios de Spa asociados:

Identificador de reserva: 3
Código del servicio: 0
Descripción del servicio: Masaje relajante
Coste base del servicio: 50.5
Coste total del servicio (coste base * número de personas): 151.5
Día servicio: 2021-01-10
Número de personas: 3


Elige una opción: 2
Introduce el número de la reserva:
3
Identificador de reserva: 3
Código del servicio: 0
Descripción del servicio: Masaje relajante
Coste base del servicio: 50.5
Coste total del servicio (coste base * número de personas): 151.5
Día servicio: 2021-01-10
Número de personas: 3
```

```
1. Listado de todas las reservas
2. Información de una reserva
3. Añadir una reserva
4. Eliminar una reserva
5. Guardar cambios
0. Volver

Elige una opción: 3
Introduce un número de habitación:
333
Introduce la fecha de inicio de la reserva (formato aaaa-mm-dd):

2021-05-02
Introduce el tipo de reserva:
1. Reserva de habitación.
2. Reserva de servicio de Spa.

1
¿A nombre de quién se hace la reserva?

Introduce el DNI del cliente:
12345678a
Introduce la fecha de finalización de la reserva (formato aaaa-mm-dd):

2021-06-01
Reserva añadida.


Elige una opción: 3
Introduce un número de habitación:
333
Introduce la fecha de inicio de la reserva (formato aaaa-mm-dd):

2021-05-02
Introduce el tipo de reserva:
1. Reserva de habitación.
2. Reserva de servicio de Spa.

2
Introduce el código del servicio:
0
Introduce la fecha para reservar servicio de spa (formato aaaa-mm-dd):

2021-05-05
Introduce el número de personas que atenderán al servicio:

2
Se ha agregado Reserva de Spa: 5 a la Reserva de Habitación: 4, con número de habitación: 333

Reserva añadida.

1. Listado de todas las reservas
2. Información de una reserva
3. Añadir una reserva
4. Eliminar una reserva
5. Guardar cambios
0. Volver

Elige una opción: 2
Introduce el número de la reserva:
4
Identificador de reserva: 4
Número de habitación: 333
Día de inicio: 2021-05-02
Coste de la habitación: 130.5
Coste total de habitación + servicios: 181.0
Día de fin de reserva: 2021-06-01
Servicios de Spa asociados:
Identificador de reserva: 5
Código del servicio: 0
Descripción del servicio: Masaje relajante
Coste base del servicio: 50.5
Coste total del servicio (coste base * número de personas): 101.0
Día servicio: 2021-05-05
Número de personas: 2
```

```
1. Listado de todas las reservas
2. Información de una reserva
3. Añadir una reserva
4. Eliminar una reserva
5. Guardar cambios
0. Volver

Elige una opción: 4
ATENCIÓN: eliminar una reserva de habitación eliminará todos los servicio
Introduce el número de la reserva:
4
1. Listado de todas las reservas
2. Información de una reserva
3. Añadir una reserva
4. Eliminar una reserva
5. Guardar cambios
0. Volver

Elige una opción: 1
Identificador de reserva: 0
Número de habitación: 111
Día de inicio: 2021-01-01
Coste de la habitación: 115.0
Coste total de habitación + servicios: 115.0
Día de fin de reserva: 2021-02-01
Servicios de Spa asociados:

Identificador de reserva: 1
Número de habitación: 222
Día de inicio: 2021-01-01
Coste de la habitación: 50.0
Coste total de habitación + servicios: 100.5
Día de fin de reserva: 2021-03-01
Servicios de Spa asociados:
Identificador de reserva: 3
Código del servicio: 0
Descripción del servicio: Masaje relajante
Coste base del servicio: 50.5
Coste total del servicio (coste base * número de personas): 151.5
Día servicio: 2021-01-10
Número de personas: 3

Identificador de reserva: 2
Número de habitación: 333
Día de inicio: 2021-04-01
Coste de la habitación: 130.5
Coste total de habitación + servicios: 130.5
Día de fin de reserva: 2021-05-01
Servicios de Spa asociados:

Identificador de reserva: 3
Código del servicio: 0
Descripción del servicio: Masaje relajante
Coste base del servicio: 50.5
Coste total del servicio (coste base * número de personas): 151.5
Día servicio: 2021-01-10
Número de personas: 3
```

Probamos que las funcionalidades del menú de Facturas funcionan correctamente.

```
Datos cargados
1. Habitaciones
2. Clientes
3. Servicios
4. Reservas
5. Facturas
0. Salir

Elige una opción: 5
1. Listado de todas las facturas
2. Información de una factura
3. Generar una factura
4. Guardar cambios
0. Volver

Elige una opción: 1
Código de factura: 00000000
DNI de cliente: 12345678a
Fecha de facturación: 2021-02-02

Código de factura: 00000001
DNI de cliente: 87654321z
Fecha de facturación: 2021-03-03

1. Listado de todas las facturas
2. Información de una factura
3. Generar una factura
4. Guardar cambios
0. Volver

Elige una opción: 2
Introduce el ID de la factura:
00000001
Código de factura: 00000001
Información del cliente: DNI: 87654321z
Nombre y Apellidos: Marco Machado
Teléfono Móvil: 987654321

Listado de reservas: Identificador de reserva: 1
Número de habitación: 222
Día de inicio: 2021-01-01
Coste de la habitación: 50.0
Coste total de habitación + servicios: 100.5
Día de fin de reserva: 2021-03-01
Servicios de Spa asociados:
Identificador de reserva: 3
Código del servicio: 0
Descripción del servicio: Masaje relajante
Coste base del servicio: 50.5
Coste total del servicio (coste base * número de personas): 151.5
Día servicio: 2021-01-10
Número de personas: 3

Coste total: 100.5
Fecha de facturación: 2021-03-03

1. Listado de todas las facturas
2. Información de una factura
3. Generar una factura
4. Guardar cambios
0. Volver

Elige una opción: 3
Introduce un número de habitación:
333
Introduce la fecha de inicio de la reserva cuya factura se desea:

2021-04-01
Introduce la fecha de facturación (formato aaaa-mm-dd). La fecha
de facturación debe ser posterior a la fecha de salida de la habitación:


2021-05-02
Factura generada con éxito.

1. Listado de todas las facturas
2. Información de una factura
3. Generar una factura
4. Guardar cambios
0. Volver

Elige una opción: 2
Introduce el ID de la factura:
00000002
Código de factura: 00000002
Información del cliente: DNI: 11223344b
Nombre y Apellidos: Hugo Hierro
Teléfono Móvil: 999999999

Listado de reservas: Identificador de reserva: 2
Número de habitación: 333
Día de inicio: 2021-04-01
Coste de la habitación: 130.5
Coste total de habitación + servicios: 130.5
Día de fin de reserva: 2021-05-01
Servicios de Spa asociados:

Coste total: 130.5
Fecha de facturación: 2021-05-02

1. Listado de todas las facturas
2. Información de una factura
3. Generar una factura
4. Guardar cambios
0. Volver

Elige una opción:
```

Probamos a introducir parámetros incorrectos en los distintos menús y observamos que en cada caso se rechazan las entradas incorrectas y se ofrece una explicación del error. Para entradas rechazadas por error de formato, se vuelve a pedir la entrada. En otros casos, cuando se hace referencia a un recurso no registrado en el sistema, se devuelve al usuario al menú para que pueda crear dicho recurso (ej: intentar reservar un servicio de spa que no existe. Se devuelve al usuario al menú para que pueda crear dicho servicio desde el menú de servicios).

Comprobación de errores para el menú principal:

```
Datos cargados
1. Habitaciones
2. Clientes
3. Servicios
4. Reservas
5. Facturas
0. Salir

Elige una opción: 8

Por favor, introduce una opción válida.

e

Por favor, introduce una opción válida.

1
1. Listado de todas las habitaciones
2. Información de una habitación
3. Añadir una habitación
4. Eliminar una habitación
5. Guardar cambios
0. Volver

Elige una opción:
```

Comprobación de errores para el menú de clientes:

```
1. Listado de todos los clientes
2. Información de un cliente
3. Añadir un cliente
4. Guardar cambios
0. Volver

Elige una opción: 2
Introduce el DNI del cliente:
a
Error de formato, vuelve a intentarlo (un DNI consiste de 8 dígitos y una letra).
Introduce el DNI del cliente:
123456789a
Error de formato, vuelve a intentarlo (un DNI consiste de 8 dígitos y una letra).
Introduce el DNI del cliente:
1234567ab
Error de formato, vuelve a intentarlo (un DNI consiste de 8 dígitos y una letra).
Introduce el DNI del cliente:
98765432g
No existe ningún cliente con el DNI indicado.

1. Listado de todos los clientes
2. Información de un cliente
3. Añadir un cliente
4. Guardar cambios
0. Volver

Elige una opción: 3
Introduce el DNI del cliente:
a
Error de formato, vuelve a intentarlo (un DNI consiste de 8 dígitos y una letra).
Introduce el DNI del cliente:
98765432g
Introduce el nombre y apellidos del cliente:

Pepe Porras
Introduce el número de teléfono móvil del cliente:

a
Formato incorrecto. El número de móvil debe tener 9 cifras.

123
Formato incorrecto. El número de móvil debe tener 9 cifras.
```

Comprobación de errores para el menú de servicios:

```
1. Listado de todos los servicios
2. Información de un servicio
3. Añadir un servicio
4. Eliminar un servicio
5. Guardar cambios
0. Volver

Elige una opción: 2
Introduce el código del servicio:
a

El valor introducido debe ser un número.

Introduce el código del servicio:
55
No existe ningún servicio con el código indicado.

1. Listado de todos los servicios
2. Información de un servicio
3. Añadir un servicio
4. Eliminar un servicio
5. Guardar cambios
0. Volver

Elige una opción: 3
Introduce una descripción del servicio:

Masaje deluxe
Introduce el coste del servicio (usar coma):

a

El coste del servicio debe ser un dato de tipo float. Inténtalo otra vez:

-1

El precio debe ser superior a 0.

1. Listado de todos los servicios
2. Información de un servicio
3. Añadir un servicio
4. Eliminar un servicio
5. Guardar cambios
0. Volver

Elige una opción: 4
Introduce el código del servicio:
a

El valor introducido debe ser un número.

Introduce el código del servicio:
77
No existe ningún servicio con el código indicado.
```

Comprobación de errores para el menú habitaciones:

```
1. Listado de todas las habitaciones
2. Información de una habitación
3. Añadir una habitación
4. Eliminar una habitación
5. Guardar cambios
0. Volver

Elige una opción: 2
Introduce un número de habitación:
e

El valor introducido no es un numero

Introduce un número de habitación:
6

El numero debe ser de tres cifras

Introduce un número de habitación:

555

La habitación no está registrada
```

```
1. Listado de todas las habitaciones
2. Información de una habitación
3. Añadir una habitación
4. Eliminar una habitación
5. Guardar cambios
0. Volver

Elige una opción: 3
Introduce un número de habitación:
a

El valor introducido no es un numero

Introduce un número de habitación:
1

El numero debe ser de tres cifras

Introduce un número de habitación:
555
Introduce el precio de la habitación (usar coma)
a
El valor introducido no es un numero
-1
El precio tiene que ser superior a 0
100
Introduce el tipo de habitación:
1. Habitación individual
2. Habitación doble
3. Habitación familiar

a

Por favor, introduce una opción correcta

7

Por favor, introduce una opción correcta

1
Habitación añadida.
```

```
1. Listado de todas las habitaciones
2. Información de una habitación
3. Añadir una habitación
4. Eliminar una habitación
5. Guardar cambios
0. Volver

Elige una opción: 4
Introduce un número de habitación:
a

El valor introducido no es un numero

Introduce un número de habitación:
1

El numero debe ser de tres cifras

Introduce un número de habitación:
555
Habitacion individual
Numero de habitación: 555
Precio de la habitación: 100.0

¿Seguro que quieres borrar esta habitación? (s/n)
h
Operación cancelada
1. Listado de todas las habitaciones
2. Información de una habitación
3. Añadir una habitación
4. Eliminar una habitación
5. Guardar cambios
0. Volver

Elige una opción: 4
Introduce un número de habitación:
555
Habitacion individual
Numero de habitación: 555
Precio de la habitación: 100.0

¿Seguro que quieres borrar esta habitación? (s/n)
s
Se ha eliminado la habitación
```

Comprobación de errores para el menú facturas:

```
1. Listado de todas las facturas
2. Información de una factura
3. Generar una factura
4. Guardar cambios
0. Volver

Elige una opción: 2
Introduce el ID de la factura:
a
Error de formato, vuelve a intentarlo (un código de factura tiene 8 dígitos).
Introduce el ID de la factura:
123
Error de formato, vuelve a intentarlo (un código de factura tiene 8 dígitos).

1. Listado de todas las facturas
2. Información de una factura
3. Generar una factura
4. Guardar cambios
0. Volver

Elige una opción: 3
Introduce un número de habitación:
333
Introduce la fecha de inicio de la reserva cuya factura se desea:

2000-01-01
La habitación indicada no tiene una reserva en vigencia o la fecha de
inicio de la reserva no coincide. Imposible generar factura.

La factura NO se ha creado. Por favor verifique que las habitaciones, clientes y reservas indicadas estén registradas previamente en el sistema.

1. Listado de todas las facturas
2. Información de una factura
3. Generar una factura
4. Guardar cambios
0. Volver

Elige una opción: 3
Introduce un número de habitación:
145
La habitación indicada no está registrada en el sistema. Imposible generar factura.

La factura NO se ha creado. Por favor verifique que las habitaciones, clientes y reservas indicadas estén registradas previamente en el sistema.
```

Comprobación de errores para el menú de reservas:

```
1. Listado de todas las reservas
2. Información de una reserva
3. Añadir una reserva
4. Eliminar una reserva
5. Guardar cambios
0. Volver

Elige una opción: 2
Introduce el número de la reserva:
a

El valor introducido debe ser un número.

Introduce el número de la reserva:
50
No existe ninguna reserva con el número indicado.

1. Listado de todas las reservas
2. Información de una reserva
3. Añadir una reserva
4. Eliminar una reserva
5. Guardar cambios
0. Volver

Elige una opción:


1. Listado de todas las reservas
2. Información de una reserva
3. Añadir una reserva
4. Eliminar una reserva
5. Guardar cambios
0. Volver

Elige una opción: 3
Introduce un número de habitación:
a

El valor introducido no es un numero

Introduce un número de habitación:
500
La habitación indicada no está registrada en el sistema. Regístrela primero antes de usarla.
```

```
1. Listado de todas las reservas
2. Información de una reserva
3. Añadir una reserva
4. Eliminar una reserva
5. Guardar cambios
0. Volver

Elige una opción: 3
Introduce un número de habitación:
333
Introduce la fecha de inicio de la reserva (formato aaaa-mm-dd):

a

Error, fecha inválida. El formato de la fecha debe ser aaaa-mm-dd (Ejemplo: 2000-10-25).

2000-01-01
Introduce el tipo de reserva:
1. Reserva de habitación.
2. Reserva de servicio de Spa.

a

Por favor, introduce una opción válida.

6

Por favor, introduce una opción válida.

1
¿A nombre de quién se hace la reserva?

Introduce el DNI del cliente:
a
Error de formato, vuelve a intentarlo (un DNI consiste de 8 dígitos y una letra).
Introduce el DNI del cliente:
5555555t
Error de formato, vuelve a intentarlo (un DNI consiste de 8 dígitos y una letra).
Introduce el DNI del cliente:
55555555t
El DNI del cliente indicado no está registrado en el sistema. Regístrelo antes de hacer la reserva a su nombre.

La reserva NO se ha creado. Por favor verifique que las habitaciones, clientes y servicios necesarios estén registrados previamente en el sistema.

1. Listado de todas las reservas
2. Información de una reserva
3. Añadir una reserva
4. Eliminar una reserva
5. Guardar cambios
0. Volver

Elige una opción: 3
Introduce un número de habitación:
333
Introduce la fecha de inicio de la reserva (formato aaaa-mm-dd):

2000-01-01
Introduce el tipo de reserva:
1. Reserva de habitación.
2. Reserva de servicio de Spa.

1
¿A nombre de quién se hace la reserva?

Introduce el DNI del cliente:
12345678a
Introduce la fecha de finalización de la reserva (formato aaaa-mm-dd):

1999-01-01

La fecha de fin de reserva debe ser superior a la fecha de inicio. El formato de la fecha debe ser aaaa-mm-dd (Ejemplo: 2000-10-25).
```

```
1. Listado de todas las reservas
2. Información de una reserva
3. Añadir una reserva
4. Eliminar una reserva
5. Guardar cambios
0. Volver

Elige una opción: 3
Introduce un número de habitación:
333
Introduce la fecha de inicio de la reserva (formato aaaa-mm-dd):

2000-01-01
Introduce el tipo de reserva:
1. Reserva de habitación.
2. Reserva de servicio de Spa.

2
No se puede reservar servicio de spa para esa habitación en ese periodo. (Debes asegurarte de que la habitación está reservada y que
la fecha de inicio indicada para el servicio de spa coincide con la fecha de inicio de la habitación).

La reserva NO se ha creado. Por favor verifique que las habitaciones, clientes y servicios necesarios estén registrados previamente en el sistema.

1. Listado de todas las reservas
2. Información de una reserva
3. Añadir una reserva
4. Eliminar una reserva
5. Guardar cambios
0. Volver

Elige una opción: 3
Introduce un número de habitación:
111
Introduce la fecha de inicio de la reserva (formato aaaa-mm-dd):

2021-01-01
Introduce el tipo de reserva:
1. Reserva de habitación.
2. Reserva de servicio de Spa.

2
Introduce el código del servicio:
0
Introduce la fecha para reservar servicio de spa (formato aaaa-mm-dd):

2021-01-10

El servicio no puede ser reservado ese día porque ya existe una reserva previa. Elija una fecha distinta.

La reserva NO se ha creado. Por favor verifique que las habitaciones, clientes y servicios necesarios estén registrados previamente en el sistema.

1. Listado de todas las reservas
2. Información de una reserva
3. Añadir una reserva
4. Eliminar una reserva
5. Guardar cambios
0. Volver

Elige una opción: 4
ATENCIÓN: eliminar una reserva de habitación eliminará todos los servicios de Spa asociados a esa reserva.
Introduce el número de la reserva:
a

El valor introducido debe ser un número.

Introduce el número de la reserva:
60
No existe ninguna reserva con el código indicado.
```