TAREA 1

Hibernate.Util.java

```
import java.io.File;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

/**
 *
 * @author alba_
 */

public class HibernateUtil {

    private static final SessionFactory SESSION_FACTORY;

    static {
        try {
          String hibernatePropsFilePath = "./src/hibernate.cfg.xml"; // Ruta al
fichero

          File hibernatePropsFile = new File(hibernatePropsFilePath);

          SESSION_FACTORY = new
Configuration().configure(hibernatePropsFile).buildSessionFactory();

      }catch(Throwable ex) {
          System.err.println("Error al crear la configuración de hibernate" +
ex.getMessage());
          throw new ExceptionInInitializerError();
      }
    }

    public static SessionFactory get() {
        return SESSION_FACTORY;
    }
}
```

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
3.0//EN" "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
```

```
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="connection.url">jdbc:mysql://localhost:3306/hospitaldb?
createDatabaseIfNotExist=true&amp;serverTimezone=UTC</property>
    <property name="connection.username">root</property>
    <property name="connection.password"></property>
    <property name="hbm2ddl.auto">validate</property>
    <property name="dialect">org.hibernate.dialect.MySQL8Dialect</property>
    <property name="hibernate.dialect.storage_engine">innodb</property>
    <property name="hibernate.show_sql">true</property>

    <mapping class="entidades.Doctor" />
    <mapping class="entidades.Hospital" />
    <mapping class="entidades.Paciente" />
    <mapping class="entidades.Tratamiento" />
    <mapping class="entidades.Cita" />
    <mapping class="entidades.Recibe" />
  </session-factory>
</hibernate-configuration>
```

pom.xml

```
<dependencies>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.6.15.Final</version>
            <type>jar</type>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.22</version>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.32</version>
            <scope>provided</scope>
    </dependency>
    </dependencies>
```

Principal.java

```
import entidades.Cita;
import entidades.Doctor;
import entidades.Hospital;
import entidades.Paciente;
import java.time.LocalDate;
```

```
import java.util.Scanner;
import org.hibernate.Session;
import repositorios.*;

/**
 *
 * @author alba_
 */
public class Principal {

    static public Scanner sc = new Scanner(System.in);
    static RepositorioDoctor repoDoc;
    static RepositorioPaciente repoPac;
    static RepositorioCita repoCit;
    static RepositorioRecibe repoRec;
    static RepositorioTratamiento repoTrat;
    static RepositorioHospital repoHosp;

    public static void main(String[] args) {

        Session sesion = HibernateUtil.get().openSession();
        repoDoc = new RepositorioDoctor(sesion);
        repoPac = new RepositorioPaciente(sesion);
        repoCit = new RepositorioCita(sesion);
        repoRec = new RepositorioRecibe(sesion);
        repoTrat = new RepositorioTratamiento(sesion);
        repoHosp = new RepositorioHospital(sesion);
        int opcion;
        do {

            System.out.println("------------------MENÚ------------------");
            System.out.println("1. Crear un nuevo doctor");
            System.out.println("2. Borrar por id un doctor");
            System.out.println("3. Modificar los datos de un doctor");
            System.out.println("4. Crear un nuevo paciente");
            System.out.println("5. Borrar por nombre un paciente");
            System.out.println("6. Modificar los datos de un paciente");
            System.out.println("7. Asignar un doctor a un paciente");
            System.out.println("8. Indicar la fecha de fin del tratamiento de un
paciente");
            System.out.println("9. Cambiar el hospital de un tratamiento");
            System.out.println("10. Mostrar los datos de un paciente");
            System.out.println("11. Mostrar los datos de los tratamientos y el
hospital en el que se realiza");
            System.out.println("12. Mostrar el número total de tratamientos que
tiene cada hospital");
            System.out.println("0. Salir");
            System.out.println("Escoge una opción: ");
            opcion = Integer.parseInt(sc.nextLine());

            switch (opcion) {
                case 0:
                    break;
                case 1:
```

```
                    crearDoctor();
                    break;
                case 2:
                    borrarDoctor();
                    break;
                case 3:
                    modificarDoctor();
                    break;
                case 4:
                    crearPaciente();
                    break;
                case 5:
                    borrarPacienteByName();
                    break;
                case 6:
                    modificarPaciente();
                    break;
                case 7:
                    asignarDoctor();
                    break;
                case 8:
                    asignarFinTratamiento();
                    break;
                case 9:
                    cambiarHospital();
                    break;
                case 10:
                    mostrarDatosPaciente();
                    break;
                case 11:
                    mostrarDatosTratamientosByHospital();
                    break;
                case 12:
                    mostrarTotalTratamientosByHospital();
                    break;
                default:
                    System.out.println("Escoge una opción correcta");
            }

        } while (opcion != 0);

        sesion.close();
    }

    /**
     * Crear los datos de un doctor.
     */
    private static void crearDoctor() {
        System.out.println("Ingrese el nombre del doctor: ");
        String nombre = sc.nextLine();
        System.out.println("Ingrese la especialidad: ");
        String especialidad = sc.nextLine();
        System.out.println("Ingrese el teléfono del doctor: ");
        String tlf = sc.nextLine();
```

```
        int idDoctor = repoDoc.devolverUltimoId() + 1;

        Doctor doctor = new Doctor(idDoctor, nombre, especialidad, tlf, null);
        int creado = repoDoc.crear(doctor);
        if (creado > 0) {
            System.out.println("Se ha creado el doctor correctamente");
        }
    }

    /**
     * borrar (por id) los datos de un doctor.
     */
    private static void borrarDoctor() {
        System.out.println("Ingrese el id del doctor a borrar: ");
        int id = Integer.parseInt(sc.nextLine());
        int idBorrado = repoDoc.borrar(id);
        if (idBorrado > 0) {
            System.out.println("Se ha borrado el doctor correctamente");
        } else {
            System.out.println("No se ha borrado el doctor");
        }
    }

    /**
     * modificar por id los datos de un doctor.
     */
    private static void modificarDoctor() {
        System.out.println("Ingrese el id del doctor a modificar: ");
        int id = Integer.parseInt(sc.nextLine());
        Doctor doctor = repoDoc.buscarById(id);
        if (doctor == null) {
            System.out.println("Ingrese un id correcto");
        } else {
            System.out.println("Ingrese el nuevo nombre del doctor: ");
            String nombre = sc.nextLine();
            System.out.println("Ingrese la nueva especialidad: ");
            String especialidad = sc.nextLine();
            System.out.println("Ingrese el nuevo teléfono del doctor: ");
            String tlf = sc.nextLine();
            doctor.setNombre(nombre);
            doctor.setEspecialidad(especialidad);
            doctor.setTelefono(tlf);
            int idModificado = repoDoc.modificar(doctor);
            if (idModificado > 0) {
                System.out.println("Se han modificado los datos del doctor con id
" + idModificado + " correctamente");
            } else {
                System.out.println("No se han podido modificar los datos del
doctor");
            }
        }
    }
```

```
    /**
     * Crear los datos de un paciente.
     */
    private static void crearPaciente() {
        System.out.println("Ingrese el nombre del paciente: ");
        String nombre = sc.nextLine();
        System.out.println("Ingrese la fecha de nacimiento (dd-mm-aaaa): ");
        String fecha = sc.nextLine();
        System.out.println("Ingrese la dirección del paciente: ");
        String direccion = sc.nextLine();

        int idPaciente = repoPac.devolverUltimoId() + 1; //creamos el id del
paciente
        // creamos la fecha a partir del string:
        int dia = Integer.parseInt(fecha.split("-")[0]);
        int mes = Integer.parseInt(fecha.split("-")[1]);
        int anho = Integer.parseInt(fecha.split("-")[2]);
        LocalDate fechaNac = LocalDate.of(anho, mes, dia);

        // creamos el paciente con los datos datos:
        Paciente paciente = new Paciente(idPaciente, nombre, fechaNac, direccion,
null, null);
        int creado = repoPac.crear(paciente);
        if (creado > 0) {
            System.out.println("Se ha creado el paciente correctamente");
        }
    }

    /**
     * modificar (por nombre) los datos de un paciente.
     */
    private static void modificarPaciente() {
        System.out.println("Ingrese el nombre del paciente a modificar: ");
        String nombre = sc.nextLine();
        Paciente paciente = repoPac.buscarByName(nombre);
        if (paciente == null) {
            System.out.println("Ingrese un nombre de paciente correcto");
        } else {
            System.out.println("Ingrese el nuevo nombre del paciente: ");
            String nombrePac = sc.nextLine();
            System.out.println("Ingrese la nueva fecha de nacimiento (dd-mm-aaaa):
");
            String fecha = sc.nextLine();
            System.out.println("Ingrese la nueva dirección del paciente: ");
            String direccion = sc.nextLine();

            int dia = Integer.parseInt(fecha.split("-")[0]);
            int mes = Integer.parseInt(fecha.split("-")[1]);
            int anho = Integer.parseInt(fecha.split("-")[2]);
            LocalDate fechaNac = LocalDate.of(anho, mes, dia);

            paciente.setNombre(nombrePac);
            paciente.setFecha_nacimiento(fechaNac);
            paciente.setDireccion(direccion);
```

```
            int idModificado = repoPac.modificar(paciente);
            if (idModificado > 0) {
                System.out.println("Se han modificado los datos del paciente con
nombre " + nombrePac + " correctamente");
            } else {
                System.out.println("No se han podido modificar los datos del
paciente");
            }
        }
    }

    /**
     * borrar los datos de un paciente.
     */
    private static void borrarPacienteByName() {
        System.out.println("Ingrese el nombre del paciente a borrar: ");
        String nombre = sc.nextLine();
        Paciente paciente = repoPac.buscarByName(nombre);
        int idBorrado = repoPac.borrar(paciente);
        if (idBorrado > 0) {
            System.out.println("Se ha borrado el paciente con nombre " + nombre +
" correctamente");
        } else {
            System.out.println("No se ha borrado el paciente");
        }
    }

    /**
     * Asignar un doctor a un paciente.
     * La asignación se hará a partir del nombre del doctor y del paciente.
     * Se pedirá por teclado introducir el nombre del doctor y del paciente.
     */
    private static void asignarDoctor() {
        System.out.println("Ingrese la fecha de la cita(dd-mm-aaaa): ");
        String fecha = sc.nextLine();
        System.out.println("Ingrese el nombre del paciente: ");
        String nombrePac = sc.nextLine();
        System.out.println("Ingrese el nombre del doctor: ");
        String nombreDoc = sc.nextLine();

        int dia = Integer.parseInt(fecha.split("-")[0]);
        int mes = Integer.parseInt(fecha.split("-")[1]);
        int anho = Integer.parseInt(fecha.split("-")[2]);
        LocalDate fechaCita = LocalDate.of(anho, mes, dia);

        Doctor doctor = repoDoc.buscarByName(nombreDoc);
        Paciente paciente = repoPac.buscarByName(nombrePac);

        //creamos una cita y modificamos datos:
        Cita cita = new Cita();
        cita.setDoctor(doctor);
        cita.setFecha(fechaCita);
        cita.setPaciente(paciente);
```

```
        int citaCreada = repoCit.crear(cita);
        if (citaCreada > 0) {
            System.out.println("Se ha asignado la cita con el doctor
correctamente");
        }
    }

    /**
     * Indicar la fecha de fin del tratamiento de un paciente.
     * El método recibirá el nombre del paciente, la fecha de inicio, el tipo y la
fecha de fin del tratamiento.
     */
    private static void asignarFinTratamiento() {
        System.out.println("Ingrese el nombre del paciente:");
        String nombre = sc.nextLine();
        System.out.println("Ingrese la fecha de inicio de tratamiento (dd-mm-
aaaa): ");
        String fechaIni = sc.nextLine();
        System.out.println("Ingrese el tipo del tratamiento: ");
        String tipo = sc.nextLine();
        System.out.println("Ingrese la fecha de fin de tratamiento (dd-mm-aaaa):
");
        String fechaFin = sc.nextLine();

        //creamos fecha inicio y fin:
        int dia = Integer.parseInt(fechaIni.split("-")[0]);
        int mes = Integer.parseInt(fechaIni.split("-")[1]);
        int anho = Integer.parseInt(fechaIni.split("-")[2]);
        LocalDate fechaInicio = LocalDate.of(anho, mes, dia);

        dia = Integer.parseInt(fechaFin.split("-")[0]);
        mes = Integer.parseInt(fechaFin.split("-")[1]);
        anho = Integer.parseInt(fechaFin.split("-")[2]);
        LocalDate fechaFinal = LocalDate.of(anho, mes, dia);

        repoRec.indicarFinTratamiento(nombre, tipo, fechaInicio, fechaFinal);

    }

    /**
     * Cambiar el hospital de un tratamiento.
     * El método recibirá el id del tratamiento, el nombre del hospital en donde
está ahora el tratamiento y
     * el nombre del hospital en dónde se va a realizar el tratamiento a partir de
ahora.
     */
    private static void cambiarHospital() {
        System.out.println("Ingrese el id del tratamiento:");
        int idTrat = Integer.parseInt(sc.nextLine());
        System.out.println("Ingrese el nombre del hospital actual que realiza el
tratamiento: ");
        String nombreActual = sc.nextLine();
        System.out.println("Ingrese el nombre del hospital nuevo que realizará el
```

```
tratamiento: ");
        String nombreNuevo = sc.nextLine();

        int cambiado = repoTrat.cambiarHospital(idTrat, nombreActual,
nombreNuevo);

        if (cambiado > 0) {
            System.out.println("Se ha modificado el hospital " + nombreActual + "
por el hospital " + nombreNuevo);
        }
    }

    /**
     * Mostrar los datos de un Paciente (id, nombre, fecha_nacimiento,
     * direccion, tratamientos que recibe y citas que tiene). La consulta se
     * hará a partir del nombre del Paciente que introduzca el usuario.
     */
    private static void mostrarDatosPaciente() {
        System.out.println("Introduza el nombre del paciente que quiere
visualizar: ");
        String nombre = sc.nextLine();
        Paciente paciente = repoPac.buscarByName(nombre);
        if (paciente == null) {
            //System.out.println("El paciente introducido no existe");
            return;
        }
        System.out.println("id: " + paciente.getId());
        System.out.println("nombre: " + paciente.getNombre());
        System.out.println("fecha nacimiento: " + paciente.getFecha_nacimiento());
        System.out.println("dirección: " + paciente.getDireccion());
        System.out.println("tratamientos que recibe: " + paciente.getRecibes());
        System.out.println("citas: " + paciente.getCitas());
    }

    /**
     * Mostrar los datos de los tratamientos y el hospital en el que se realiza.
     * La consulta se hará a partir del nombre del hospital que introduzca el
     * usuario.
     */
    private static void mostrarDatosTratamientosByHospital() {
        System.out.println("Introduce el nombre del hospital en el que quieres ver
sus tratamientos: ");
        String nombre = sc.nextLine();

        Hospital hospital = repoHosp.buscarByName(nombre);
        if (hospital != null) {
            System.out.println("Los tratamientos del hospital " +
hospital.getNombre() + "ubicado en " + hospital.getUbicacion() + " son:\n" +
hospital.getTratamientos());
        }
    }

    /**
     * Mostrar el número total de tratamientos que tiene cada hospital. La
```

```
     * consulta se hará a partir del nombre del hospital que introduzca el
     * usuario.
     */
    private static void mostrarTotalTratamientosByHospital() {
        System.out.println("Introduce el nombre del hospital en el que quieres ver
el total de sus tratamientos: ");
        String nombre = sc.nextLine();
        Hospital hospitalEncontrado = repoHosp.buscarByName(nombre);
        if (hospitalEncontrado == null) {
            //System.out.println("No se ha encontrado el hospital");
            return;
        }

        int totalTrat = repoTrat.totalTratamientosByHospital(nombre);

        if (totalTrat > 0) {
            System.out.println("El hospital " + nombre + " tiene " + totalTrat + "
tratamientos");
        } else {
            System.out.println("No se han encontrado tratamientos en el hospital "
+ nombre);
        }
    }
}
```

Entidades

1. **Hospital.java**

```
package entidades;

import java.util.ArrayList;
import java.util.List;
import javax.persistence.*;
import lombok.*;

/**
 * Clase de la tabla hospital de la base de datos hospitalbd
 * @author alba_
 */

@Entity //para decir que es una entidad (tabla bbdd)
@Data //lombok: getter, setter, constructores, toString
@AllArgsConstructor //incluye el constructor con todos los argumentos
@NoArgsConstructor //contructor vacío
@RequiredArgsConstructor //contructor con los argumentos que no puedan ser nulos
public class Hospital {
    @Id
    @NonNull
    private int id;
```

```
    @NonNull
    private String nombre;
    private String ubicacion;

    @OneToMany(mappedBy="hospital", cascade = CascadeType.ALL)
    private List<Tratamiento> tratamientos;

}
```

2. **Paciente.java**

```
package entidades;

import java.time.LocalDate;
import java.util.List;
import javax.persistence.*;
import lombok.*;

/**
 * Clase de la tabla paciente de la base de datos hospitalbd
 * @author alba_
 */

@Entity //para decir que es una entidad (tabla bbdd)
@Data //lombok: getter, setter, constructores, toString
@AllArgsConstructor //incluye el constructor con todos los argumentos
@NoArgsConstructor //contructor vacío
@RequiredArgsConstructor //contructor con los argumentos que no puedan ser nulos
public class Paciente {
    @Id
    @NonNull
    private int id;

    @NonNull
    private String nombre;

    private LocalDate fecha_nacimiento;
    private String direccion;

    @OneToMany(mappedBy="paciente", cascade = CascadeType.ALL)
    private List<Cita> citas;

    @OneToMany(mappedBy="paciente", cascade = CascadeType.ALL)
    private List<Recibe> recibes;

}
```

3. **Recibe.java**

```java
package entidades;

import java.io.Serializable;
import java.time.LocalDate;
import java.util.Date;
import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;

/**
 * Clase de la tabla recibe de la base de datos hospitalbd
 * @author alba_
 */

@Entity //para decir que es una entidad (tabla bbdd)
@Data //lombok: getter, setter, constructores, toString
@AllArgsConstructor //incluye el constructor con todos los argumentos
@NoArgsConstructor //contructor vacío
@RequiredArgsConstructor //contructor con los argumentos que no puedan ser nulos
public class Recibe implements Serializable{

    @Id
    @NonNull
    @ManyToOne(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    @JoinColumn(name="id_paciente")
    private Paciente paciente;

    @Id
    @NonNull
    @ManyToOne(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    @JoinColumn(name="tratamiento_id")
    private Tratamiento tratamiento;

    @Id
    @NonNull
    private LocalDate fecha_inicio;
    private LocalDate fecha_fin;

    @Override
    public String toString(){
        return "Recibe: " + tratamiento + ", fecha inicio: " + fecha_inicio + ",
fecha fin: " + fecha_fin + "\n";
    }
```

```
}
```

4. `Tratamiento.java`

```
package entidades;

import java.math.BigDecimal;
import java.util.List;
import javax.persistence.*;
import lombok.*;


/**
 * Clase de la tabla tratamiento de la base de datos hospitalbd
 * @author alba_
 */

@Entity //para decir que es una entidad (tabla bbdd)
@Data //lombok: getter, setter, constructores, toString
@AllArgsConstructor //incluye el constructor con todos los argumentos
@NoArgsConstructor //contructor vacío
@RequiredArgsConstructor //contructor con los argumentos que no puedan ser nulos
public class Tratamiento {
    @Id
    @NonNull
    private int id;

    @NonNull
    private String tipo;

    private BigDecimal costo;

    @ManyToOne(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    @JoinColumn(name="id_hospital")
    private Hospital hospital;

    @OneToMany(mappedBy="tratamiento", cascade = CascadeType.ALL)
    private List<Recibe> recibes;

    @Override
    public String toString(){
        return "tipo: " + tipo + ", costo: " + costo;
    }
}
```

Repositorios

1. Interfaz `Repositorio.java`

```java
package repositorios;

/**
 * Interfaz de métodos para implementar
 * @author alba_
 */
public interface Repositorio<T> {
    public int crear(T t);
    public int modificar(T t);
    public int borrar(int id);
    public void mostrarTodos();
    public int devolverUltimoId();
    public T buscarById(int id);
    public T buscarByName(String nombre);
}
```

2. **RepositorioCita.java**ç

```java
package repositorios;

import entidades.Cita;
import org.hibernate.Session;
import org.hibernate.Transaction;

/**
 *
 * @author alba_
 */
public class RepositorioCita implements Repositorio<Cita>{
    private Session sesion;

    //Creamos constructor con la sesion
    public RepositorioCita(Session sesion) {
        this.sesion = sesion;
    }

    @Override
    public int crear(Cita cita) {
        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        try {
            sesion.persist(cita);
            transacion.commit();
            return 1; // devuelve el id del doctor creado
        } catch (Exception ex) {
            transacion.rollback();//para que no lo guarde si hay un error
            return 0;
        }
    }
```

```
    @Override
    public int modificar(Cita t) {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public int borrar(int id) {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public void mostrarTodos() {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public int devolverUltimoId() {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public Cita buscarById(int id) {
        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        Cita cita = (Cita) sesion.createQuery("FROM Cita WHERE id = :idC
").setParameter("idC", id).uniqueResult();
        transacion.commit();
        if (cita == null) {
            System.out.println("El id no existe");
            return null;
        }
        return cita;
    }

    @Override
    public Cita buscarByName(String nombre) {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }
}
```

3. **RepositorioDoctor.java**

```java
package repositorios;

import entidades.Doctor;
import org.hibernate.Session;
import org.hibernate.Transaction;

/**
 * Clase que tendrá los métodos de la clase Doctor
 *
 * @author alba_
 */
public class RepositorioDoctor implements Repositorio<Doctor> {

    private Session sesion;

    //Creamos constructor con la sesion
    public RepositorioDoctor(Session sesion) {
        this.sesion = sesion;
    }

    @Override
    public int crear(Doctor doctor) {
        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        try {
            sesion.persist(doctor);
            transacion.commit();
            return doctor.getId(); // devuelve el id del doctor creado
        } catch (Exception ex) {
            transacion.rollback();//para que no lo guarde si hay un error
            return 0;
        }
    }

    @Override
    public int modificar(Doctor doctor) {
        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        try {
            sesion.merge(doctor);
            transacion.commit();
            return doctor.getId(); // devuelve el id del doctor modificado
        } catch (Exception ex) {
            transacion.rollback();//para que no lo guarde si hay un error
            return 0;
        }
    }

    @Override
    public int borrar(int id) {
        Doctor doctor = buscarById(id);
        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
```

```
        try {
            sesion.remove(doctor);
            transacion.commit();
            return doctor.getId(); // devuelve el id del doctor borrado
        } catch (Exception ex) {
            transacion.rollback();//para que no lo guarde si hay un error
            return 0;
        }
    }

    @Override
    public void mostrarTodos() {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public int devolverUltimoId() {
        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        Integer ultimoId = (Integer) sesion.createQuery("SELECT MAX(id) FROM
Doctor").uniqueResult();
        transacion.commit();
        if (ultimoId == null) {
            return 0;
        }
        return ultimoId;
    }

    @Override
    public Doctor buscarById(int idDoc) {
        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        Doctor doctor = (Doctor) sesion.createQuery("FROM Doctor WHERE id = :idDoc
").setParameter("idDoc", idDoc).uniqueResult();
        transacion.commit();
        if (doctor == null) {
            System.out.println("El id no existe");
            return null;
        }
        return doctor;
    }

    public Doctor buscarByName(String nombre){
    Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        Doctor doctor = (Doctor) sesion.createQuery("FROM Doctor WHERE nombre =
:nomDoc ").setParameter("nomDoc", nombre).uniqueResult();
        transacion.commit();
        if (doctor == null) {
            System.out.println("El nombre no existe");
            return null;
```

```
        }
        return doctor;
    }
}
```

4. **RepositorioHospital.java**

```
package repositorios;

import entidades.Hospital;
import org.hibernate.Session;
import org.hibernate.Transaction;

/**
 *
 * @author alba_
 */
public class RepositorioHospital implements Repositorio<Hospital> {

    private Session sesion;

    //Creamos constructor con la sesion
    public RepositorioHospital(Session sesion) {
        this.sesion = sesion;
    }

    @Override
    public int crear(Hospital hospital) {
        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        try {
            sesion.persist(hospital);
            transacion.commit();
            return hospital.getId(); // devuelve el id del hospital creado
        } catch (Exception ex) {
            transacion.rollback();//para que no lo guarde si hay un error
            return 0;
        }
    }

    @Override
    public int modificar(Hospital hospital) {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public int borrar(int id) {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
```

```
    }

    @Override
    public void mostrarTodos() {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public int devolverUltimoId() {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public Hospital buscarById(int id) {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public Hospital buscarByName(String nombre) {
        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        Hospital hospital = (Hospital) sesion.createQuery("FROM Hospital WHERE
nombre = :nomHos ").setParameter("nomHos", nombre).uniqueResult();
        transacion.commit();
        if (hospital == null) {
            System.out.println("El nombre del hospital no existe");
            return null;
        }
        return hospital;
    }
}
```

5. **RepositorioPaciente.java**

```
package repositorios;

import entidades.Paciente;
import org.hibernate.Session;
import org.hibernate.Transaction;

/**
 *
 * @author alba_
 */
public class RepositorioPaciente implements Repositorio<Paciente> {
```

```
    private Session sesion;

    //Creamos constructor con la sesion
    public RepositorioPaciente(Session sesion) {
        this.sesion = sesion;
    }

    @Override
    public int crear(Paciente paciente) {
        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        try {
            sesion.persist(paciente);
            transacion.commit();
            return paciente.getId(); // devuelve el id del paciente creado
        } catch (Exception ex) {
            transacion.rollback();//para que no lo guarde si hay un error
            return 0;
        }
    }

    @Override
    public int modificar(Paciente paciente) {
        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        try {
            sesion.merge(paciente);
            transacion.commit();
            return paciente.getId(); // devuelve el id del doctor modificado
        } catch (Exception ex) {
            transacion.rollback();//para que no lo guarde si hay un error
            return 0;
        }
    }

    @Override
    public int borrar(int id) {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public void mostrarTodos() {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public int devolverUltimoId() {
        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        Integer ultimoId = (Integer) sesion.createQuery("SELECT MAX(id) FROM
```

```
Paciente").uniqueResult();
        transacion.commit();
        if (ultimoId == null) {
            return 0;
        }
        return ultimoId;
    }

    @Override
    public Paciente buscarById(int idPac) {
        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        Paciente paciente = (Paciente) sesion.createQuery("FROM Paciente WHERE id
= :idPac ").setParameter("idPac", idPac).uniqueResult();
        transacion.commit();
        if (paciente == null) {
            System.out.println("El id no existe");
            return null;
        }
        return paciente;
    }

    public Paciente buscarByName(String nombre) {
        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        Paciente paciente = (Paciente) sesion.createQuery("FROM Paciente WHERE
nombre = :nomPac ").setParameter("nomPac", nombre).uniqueResult();
        transacion.commit();
        if (paciente == null) {
            System.out.println("El nombre no existe");
            return null;
        }
        return paciente;
    }

    public int borrar(Paciente paciente) {
        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        try {
            sesion.remove(paciente);
            transacion.commit();
            return paciente.getId(); // devuelve el id del paciente borrado
        } catch (Exception ex) {
            transacion.rollback();//para que no lo guarde si hay un error
            return 0;
        }
    }
}
```

6. **RepositorioRecibe.java**

```java
package repositorios;

import entidades.Paciente;
import entidades.Recibe;
import entidades.Tratamiento;
import java.time.LocalDate;
import org.hibernate.Session;
import org.hibernate.Transaction;

/**
 *
 * @author alba_
 */
public class RepositorioRecibe implements Repositorio<Recibe>{
    private Session sesion;

    //Creamos constructor con la sesion
    public RepositorioRecibe(Session sesion) {
        this.sesion = sesion;
    }

    public void indicarFinTratamiento(String paciente, String tipo, LocalDate
fechaInicio, LocalDate fechaFin ){
        RepositorioPaciente repPac = new RepositorioPaciente(sesion);
        RepositorioTratamiento repTra = new RepositorioTratamiento(sesion);
        Paciente pac = repPac.buscarByName(paciente);
        Tratamiento tra = repTra.buscarByName(tipo);

        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        try {
            Recibe rec=new Recibe(pac, tra, fechaInicio, fechaFin);
            sesion.merge(rec);
            transacion.commit();

        } catch (Exception ex) {
            transacion.rollback();//para que no lo guarde si hay un error
            System.out.println("No se pudo crear un registro en recibe");
        }
    }

    @Override
    public int crear(Recibe t) {
        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        try {
            sesion.persist(t);
            transacion.commit();
            return 1; // devuelve el id del recibe creado
        } catch (Exception ex) {
            transacion.rollback();//para que no lo guarde si hay un error
            return 0;
        }
```

```
    }

    @Override
    public int modificar(Recibe t) {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public int borrar(int id) {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public void mostrarTodos() {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public int devolverUltimoId() {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public Recibe buscarById(int id) {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public Recibe buscarByName(String nombre) {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }
}
```

7. `RepositorioTratamiento.java`

```
package repositorios;

import entidades.Hospital;
import entidades.Tratamiento;
```

```
import org.hibernate.Session;
import org.hibernate.Transaction;

/**
 *
 * @author alba_
 */
public class RepositorioTratamiento implements Repositorio<Tratamiento>{
    private Session sesion;

    //Creamos constructor con la sesion
    public RepositorioTratamiento(Session sesion) {
        this.sesion = sesion;
    }
    @Override
    public int crear(Tratamiento tratamiento) {
        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        try {
            sesion.persist(tratamiento);
            transacion.commit();
            return tratamiento.getId(); // devuelve el id del tratamiento creado
        } catch (Exception ex) {
            transacion.rollback();//para que no lo guarde si hay un error
            return 0;
        }
    }

    @Override
    public int modificar(Tratamiento tratamiento) {
        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        try {
            sesion.merge(tratamiento);
            transacion.commit();
            return tratamiento.getId(); // devuelve el id del tratamiento
modificado
        } catch (Exception ex) {
            transacion.rollback();//para que no lo guarde si hay un error
            return 0;
        }
    }

    @Override
    public int borrar(int id) {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public void mostrarTodos() {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
```

```
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public int devolverUltimoId() {
        throw new UnsupportedOperationException("Not supported yet."); //
Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }

    @Override
    public Tratamiento buscarById(int id) {
        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        Tratamiento tratamiento = (Tratamiento) sesion.createQuery("FROM
Tratamiento WHERE id = :idTr ").setParameter("idTr", id).uniqueResult();
        transacion.commit();
        if (tratamiento == null) {
            System.out.println("El id no existe");
            return null;
        }
        return tratamiento;
    }

    public Tratamiento buscarByName(String nombre) {
        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        Tratamiento tratamiento = (Tratamiento) sesion.createQuery("FROM
Tratamiento WHERE tipo = :tipo ").setParameter("tipo", nombre).uniqueResult();
        transacion.commit();
        if (tratamiento == null) {
            System.out.println("El tipo no existe");
            return null;
        }
        return tratamiento;

    }

    public int cambiarHospital(int idTrat, String hospitalActual, String
hospitalNuevo){
        RepositorioHospital repoHos = new RepositorioHospital(sesion);
        Tratamiento tratamiento = buscarById(idTrat);
        Hospital hospitalAct = repoHos.buscarByName(hospitalActual);
        Hospital hospitalNew = repoHos.buscarByName(hospitalNuevo);
        if(hospitalNew!=null){
            tratamiento.setHospital(hospitalNew);
        }else{
            repoHos.crear(hospitalNew);
            tratamiento.setHospital(hospitalNew);
        }
        return modificar(tratamiento);
    }

    public int totalTratamientosByHospital(String nombre){
```

```
        RepositorioHospital repoHos = new RepositorioHospital(sesion);
        Hospital hospital = repoHos.buscarByName(nombre);

        Transaction transacion = sesion.beginTransaction(); // para comenzar una
transación
        int tratamientos = (Integer) sesion.createQuery("SELECT COUNT(*) FROM
Tratamiento WHERE id_hospital =:idH GROUP BY id_hospital").setParameter("idH",
hospital.getId()).uniqueResult();
        transacion.commit();
        if (tratamientos == 0) {
            System.out.println("No hay tratamientos");
            return 0;
        }
        return tratamientos;
    }
}
```