EJERCICIO HOSPITAL

1. Conexión.

En este caso se utiliza 1 clase inmutable (para repasar) para la conexión. En este ejercicio se realiza una única conexión al principio, se utiliza para todos los accesos a la BD, y se cierra al final. Cada conexión implica retardos y consumo de recursos por eso utilizamos una sola. Otra opción podría ser utilizar el patrón Singleton (aparece el ejemplo en los apuntes de OW).

2. Resto de Clases (modelo)

Fijaos que la fecha de nacimiento de Personas está en la BD como DATE pero en la aplicación se trata como String. Esto no nos permite realizar cálculos de fechas. La fecha de operación, sin embargo, se toma como LocalDate.

```
private void jButtonInsertarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    try {
        String consulta = "INSERT INTO operaciones (cod_operacion, descripcion, fecha, tipo, dni, cod_qui
        PreparedStatement ps = connection.prepareStatement(consulta);

        if (comprobarCajasOperaciones()) {
            ps.setString(1, jTextFieldCodigo.getText());
            ps.setString(2, jTextFieldDescripcion.getText());
            SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
            String addDate = dateFormat.format(jDateChooserFecha.getDate());
           //ps.setString(3, addDate);
            //una forma de guardar la fecha tomándola como String o como tipo fecha,
            //en el segundo caso hay que utilizar java.sql.Date para cambiar de tipo Date a sql.date
            ps.setDate(3, java.sql.Date.valueOf(addDate));
```

Fijaos que en este caso, no se pasa como cadena sino como Fecha (setDate), pero hay que hacer una conversión antes porque el componente jDateChooser me devuelve un valor de tipo Date de java.Date y en los prepared Statement el setDate utiliza uno de java.sql.Date. (por eso lo pasamos a cadena y luego a sq.Date)

También cuando accedemos a su valor en un ResultSet al recoger el valor no puede ser tomado como String sino como fecha pero en este caso con LocalDate que es como lo hemos definido en la clase Operaciones. Aquí podemos hacerlo de dos formas utilizando getDate o getObject como genérico y luego especificando la clase.

```
public ArrayList<Operacion> getListaOperaciones() {
 ArrayList<Operacion> listaOperaciones = new ArrayList<>();

 String query = "SELECT * FROM operaciones";
 Statement st;
 ResultSet rs;

 try {
     st = connection.createStatement();
     rs = st.executeQuery(query);
     Operacion operacion;


     while (rs.next()) {
             operacion = new Operacion(rs.getString("cod_operacion"), rs.getString("descripcion"),
             //Con la fecha podemos:
             //rs.getObject ( "fecha", LocalDate.class ), //indico la clase en el genérico de Object
             //rs.getObject ( 3, LocalDate.class ); puede ser por posición o por nombre
             rs.getDate("fecha").toLocalDate(),
             rs.getString("tipo"), rs.getString("dni"), rs.getString("cod_quirofano"));
         listaOperaciones.add(operacion);
```

3. DAO

Solo estamos utilizando dos paquetes, uno para las clases y otro para la interfaz. Si quisiésemos diferenciar aún más la parte de los métodos que utilizamos para conectar a la BD, podríamos crear un nuevo paquete: DAO en el que pusiésemos una clase por cada una de las que tenemos en el modelo con todos los métodos asociados. Por ejemplo, del modelo de la clase Personas que ya tenemos en /clases podríamos tener en el paquete DAO una clase que fuese PersonasDAO y que tuviese todos los métodos necesarios para añadir, borrar, modificar… personas. (Ver el ejemplo de OW)

4. ORM y JPA

Existen otras formas de persistencia en Java, no corresponden a este curso, pero sí podemos comentar que por ejemplo en ORM se trabaja con la BD relacional desde Java como si fuesen objetos. Así se trabaja con la BD relacional como si fuese una BD de objetos con lo que no se hacen consultas con SQL. Ejemplos: Hibernate, OpenJPA, …

Podéis echarle un vistazo a: https://www.tutorialspoint.com/es/jpa/jpa_orm_components.htm

5. Try con recursos.
   También pueden utilizarse con los preparedStatement

6. Recogida de parámetros
   - Con resultSet:
     rs.getString(1);
     ó
     rs.getString(“nombre”);

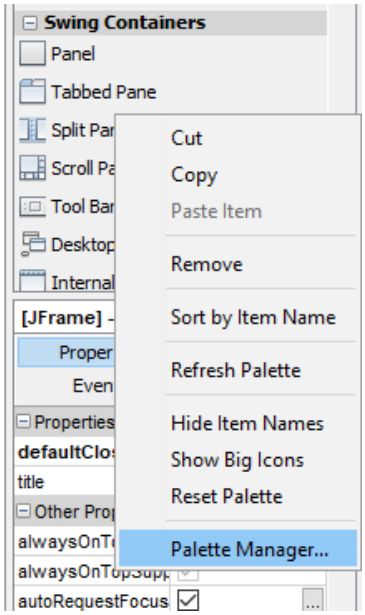
   - Con preparedStatement:

     ps.setString(1,variable que contiene el valor a guardar);

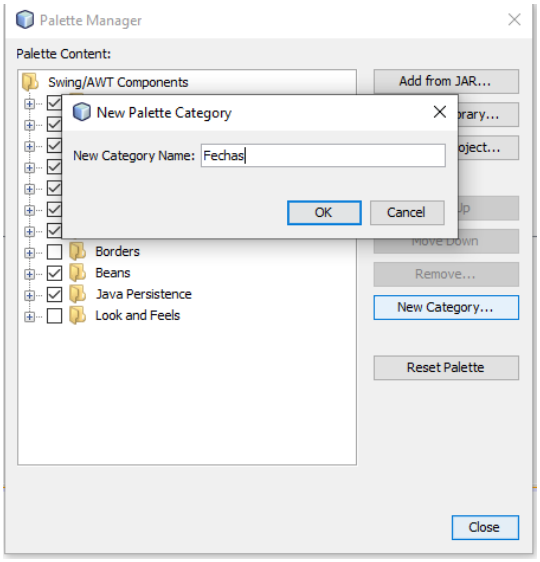
7. jCalendar

A continuación se indican los pasos para introducir nuevos componentes, los pasos son similares cuando queráis añadir nuevos .jar con componentes externos.

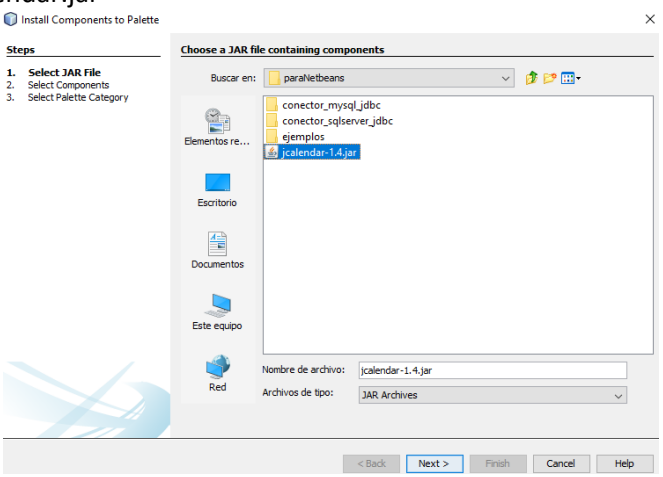
a. Añadimos en la Paleta de Componentes un nuevo elemento:



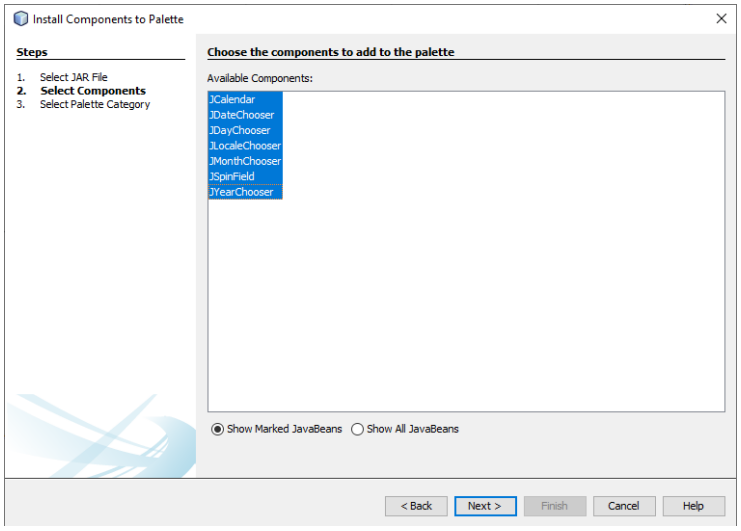
b. Añadimos una nueva Categoría:



c. Pulsamos OK y pinchamos en Add from JAR y especificamos ahí donde tenemos el fichero jcalendar.jar



d. Pulsamos Next. Y en la siguiente pantalla seleccionamos todos los componentes



e. Ahora en nuestra paleta ya aparecen los nuevos componentes: