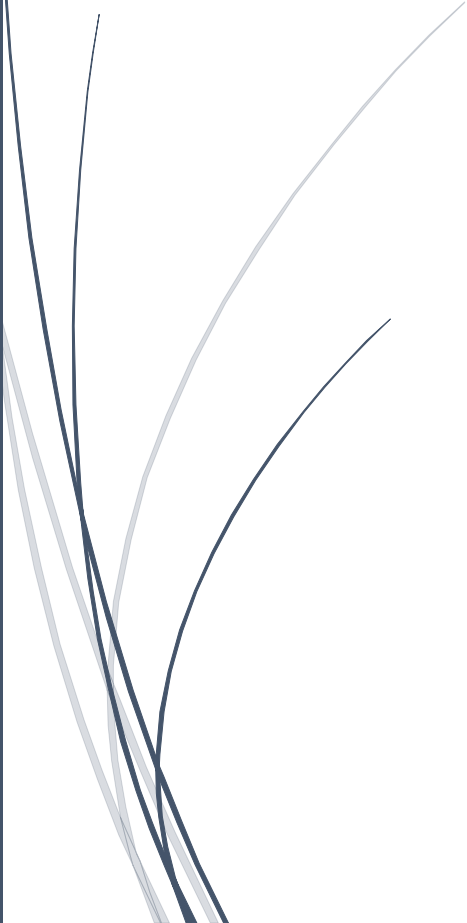


A dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from this bar, containing the date.

15-5-2018

# Clínica Privada Zaidín

Proyecto final EDD

Several thin, curved lines in dark blue and light grey originate from the bottom left and sweep upwards and to the right.

Mario Pérez Domínguez, José Miguel Yáñez  
Martínez, Plácido Ramírez Calvero, Laura Ruiz  
Sánchez, Alberto Privado Moreno, David Serrano  
Alonso.

# Índice

<b>Enunciado.....</b>	<b>2</b>
<b>Diagrama de Clases .....</b>	<b>3</b>
<b>JForms .....</b>	<b>4</b>
<b>Ficheros.....</b>	<b>14</b>
<b>Medicamentos .....</b>	<b>17</b>
<b>Pacientes.....</b>	<b>18</b>
<b>MySQL.....</b>	<b>22</b>
<b>Pruebas del Software .....</b>	<b>25</b>
<b>Control de Versiones .....</b>	<b>28</b>
<b>Unificación .....</b>	<b>29</b>
<b>Conclusiones .....</b>	<b>34</b>

## Enunciado

Una clínica privada de nuestra ciudad nos ha encargado que desarrollemos una aplicación informática que gestione toda la información de los pacientes y la de propia clínica.

Dicho software, tiene asociada una base de datos de pacientes, para los cuales, hay una asociación con la seguridad social y en la que el NUSS de la esta, corresponderá con el identificador de cada paciente de dicha clínica.

La base de datos, tendrá las siguientes tablas:

- Consulta.
- Medicamento.
- Medico.
- Paciente.
- Receta

En la tabla consulta, se guarda toda la información referente a la consulta. En la tabla paciente, se guardará toda la información referente a los pacientes. En la tabla Medicamento, se guardará toda la información referente a los medicamentos. En la tabla Médico se guarda toda la información referente al médico. En la tabla Receta se guardan todas las recetas emitidas para el paciente con el NUSS asociado. En la tabla historial, se guardará toda la información clínica del paciente y será mostrado en un cuadro de texto.

Cuando inicie la aplicación solicitará introducir el NUSS el cual nos mostrará todos los datos del paciente. Si es la primera vez que asiste un paciente a la consulta, la aplicación debe permitir la inserción de un nuevo paciente.

El formulario, deberá constar con un botón de guardar, un botón volver y un botón para emitir una receta:

- Guardar: Guardará todas las modificaciones que se realicen en el formulario en la base de datos asociada a este proyecto.
- Volver: Retrocedemos hasta la selección de un paciente.
- Recetar: En una nueva ventana, permitirá elegir medicamento, periodicidad y la toma de ese medicamento.

La nueva ventana de recetas constará de los siguientes botones y los siguientes campos:

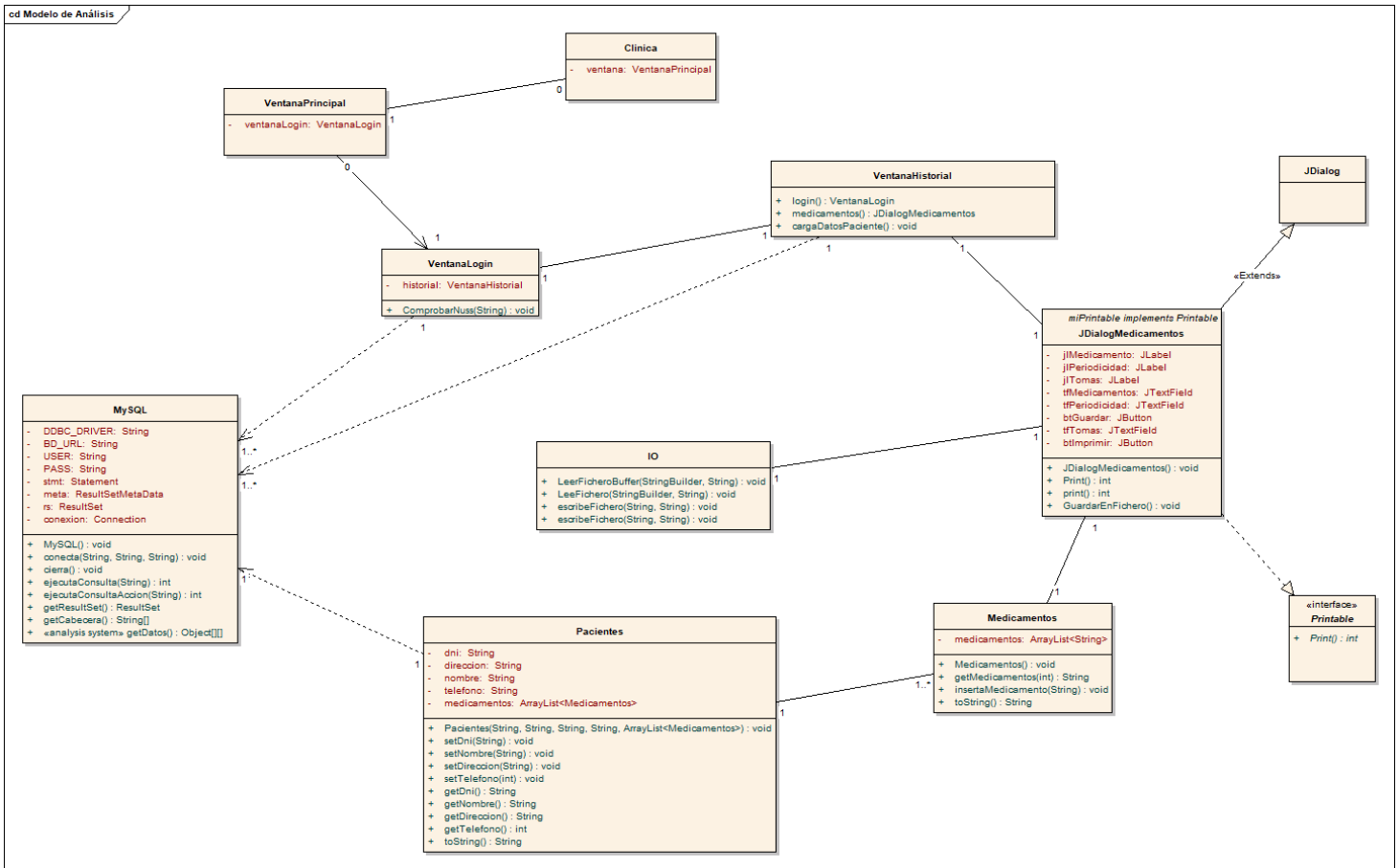
Campos:

- Medicamento:
- Periodicidad:
- Tomas:

Botones:

- Guardar, el cual guarda la receta en la base de datos.
- Imprimir, que nos permitirá imprimir la receta por la impresora predeterminada o en un PDF.

# Diagrama de Clases



## JForms

Para la creación de la interfaz gráfica, hemos creado tres JFrame Form: VentanaPrincipal, VentanaEntrada y VentanaFormulario.

Para crear esta clase, hay que seleccionar el Java Package y con el botón derecho daremos a 'New' y por último, a 'JFrame Form'.

La VentanaPrincipal está formada por un botón con una imagen, que es el logo de la clínica privada, que al clicar nos lleva a la siguiente ventana.



El código de esta ventana es el siguiente:

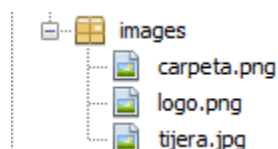
- 1) Las funciones básicas de esta ventana:

```
8 public VentanaPrincipal() {  
9     initComponents();  
10    setVisible(true);  
11    setLocationRelativeTo(null);  
12 }
```

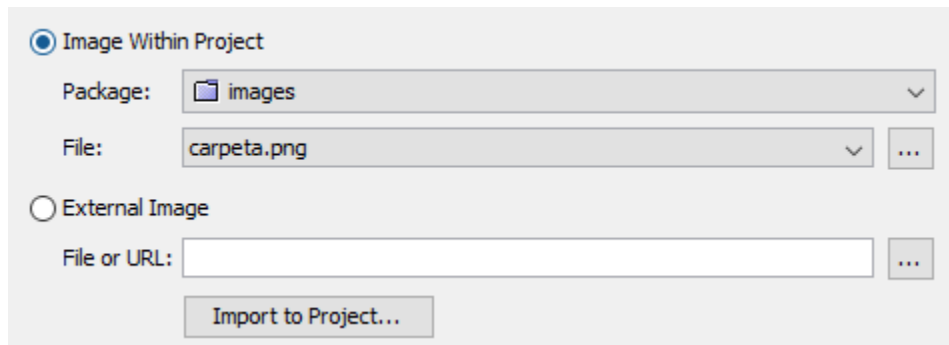
- 2) De la línea 44-46 definimos la función del botón que es abrir la siguiente ventana, que a continuación explicaremos.

```
44 private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
45     VentanaEntrada ven = new VentanaEntrada();  
46 }
```

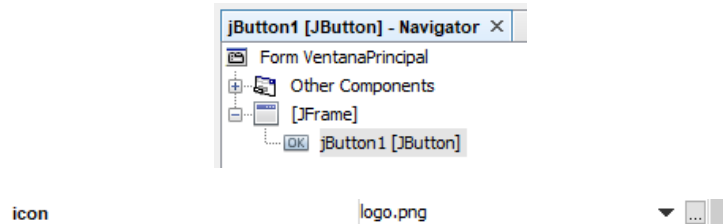
Para insertar la imagen tenemos que crear un nuevo Java Package y lo llamaremos imágenes.



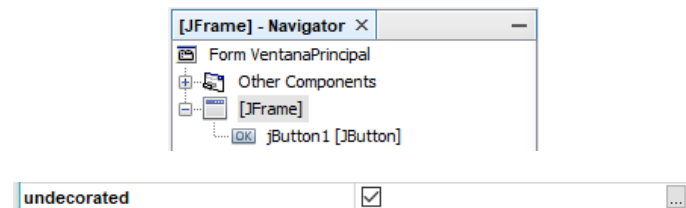
Una vez creamos el Package, tenemos que insertar las fotos que es a partir de las propiedades del botón, en este caso. Para insertarla, daríamos a 'Import to Project' y la meteríamos en el package que acabamos de crear 'Imágenes'. Una vez insertada, seleccionaremos 'Image Within Project', el paquete donde lo metimos anteriormente y por último seleccionamos la imagen que queremos que salga.



Si luego quisieramos cambiar la imagen debemos ir a las propiedades del botón, con el botón derecho sobre el nombre del botón, y buscar la opción 'ícon':

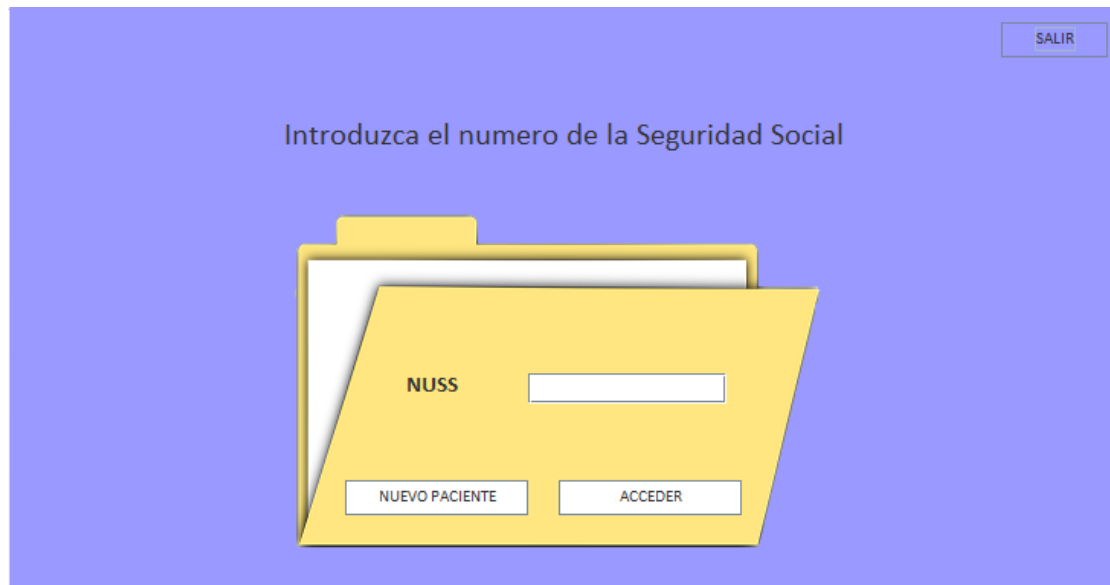


Para quitar la barra de arriba donde nos encontramos los botones para minimizar, restaurar tamaño y cerrar, se hace a partir del JFrame. Haríamos lo mismo que hicimos con el botón, nos meteríamos en sus propiedades y seleccionamos la opción 'undecorated':



Al clicar, nos llevaría a la siguiente ventana que es VentanaEntrada. Esta ventana es un poco más complicada que la anterior, ya que necesitamos de la base de datos para coger el NUS de los pacientes. Los elementos de los que consta son:

- Tres botones, uno de ellos es el de 'Salir' situado arriba a la derecha, otro es el de 'Nuevo Paciente' que sirve como su nombre indica, para insertar nuevos pacientes en la base de datos y, por último, el botón 'Acceder' que solo funcionará cuando el campo de texto del NUS este relleno con uno de los NUS de la base de datos, llevándonos a otro JFrame con la información de ese paciente.
- Un campo de texto JTextField, donde insertaremos el NUS del paciente del que queramos la información.



Además, para insertar la imagen, haríamos igual que en el JFrame anterior pero en vez de en un botón sería en un JLabel.

En esta ventana, empezamos a utilizar la base de datos por lo que tenemos que empezar a usar una nueva clase, inicializamos MySQL para poder usar dicha base de datos, utilizar los ResultSet y Connection para poder conectarnos. También usaremos un ArrayList de pacientes y el NUS.

```

18 public class VentanaEntrada extends javax.swing.JFrame {
19     private MySQL mysql;
20     private ResultSet rs;
21     private ResultSet resultSet;
22     private ResultSet resultSetPacientes;
23     private ArrayList<Pacientes> pacientes;
24     private Connection conexion;
25     private int nuss;

```

Una vez instanciamos empezamos a darle funcionalidad.

```

27 public VentanaEntrada() {
28     mysql = new MySQL();
29     conexion = mysql.conecta("root", "1234", "clinicaprivada");
30     pacientes = new ArrayList<>();
31     initComponents();
32     setVisible(true);
33
34     setLocationRelativeTo(null);
35
36
37     try {
38         llenarArrayList();
39     } catch (UnsupportedEncodingException ex) {
40         JOptionPane.showMessageDialog(null, "Problema al llenar el ArrayList");
41     }
42
43 }

```

Como vemos en la línea 38, utilizamos un método llamado 'llenarArrayList', miremos su funcionamiento.

Este método servirá para capturar todos los datos de un nuevo paciente. Y como vemos, al principio todos los campos están vacíos.

```

186 public void llenarArrayList() throws UnsupportedEncodingException{
187     resultSet = mysql.ejecutaConsulta("SELECT * FROM paciente");
    int nuss = 0;
    String dni = "";
    String nombre = "";
    String primerApellido = "";
    String segundoApellido = "";
    String calle = "";
    String localidad = "";
    int telf = 0;
    Date fechaNacimiento = null;
    Date fechaAlta = null;
    int medico = 0;
    String historial = "";
    Object medicamento = 0;
200

```

Cuando capturamos todos los datos del paciente, creamos un objeto paciente y lo introducimos en el arrayList de pacientes, de esta manera tendremos todos los pacientes cargados en memoria.

Antes de meternos con la funcionalidad de los botones, explicaremos algunos métodos que vamos a necesitar para que funcionen.

```

202 try {
203     while(resultSet.next()){
204         nuss = resultSet.getInt(1);
205         dni = resultSet.getString(2);
206         nombre = resultSet.getString(3);
207         primerApellido = resultSet.getString(4);
208         segundoApellido = resultSet.getString(5);
209         telf = resultSet.getInt(6);
210         calle = resultSet.getString(7);
211         localidad = resultSet.getString(8);
212         fechaNacimiento = resultSet.getDate(9);
213         fechaAlta = resultSet.getDate(10);
214         medico = resultSet.getInt(11);
215         historial = resultSet.getString(12);
216
217         Pacientes paciente = new Pacientes(nuss, dni, nombre, primerApellido,
218             segundoApellido, telf, calle, localidad, fechaNacimiento, fechaAlta, medico, historial);
219         this.nuss = nuss;
220         pacientes.add(paciente);
221     }
222 } catch (SQLException ex) {
223     JOptionPane.showMessageDialog(null, ex.getMessage());
224 }
225

```

Primero, crearemos un método que compruebe que el NUS que estamos insertando si está insertado en el arrayList.

```

165 public boolean comprobarNuss() throws SQLException{
166     boolean correcto = false;
167     for(Pacientes p : pacientes){
168         if(Integer.parseInt(tfNuss.getText()) == p.getNus()){
169             correcto = true;
170             nuss = p.getNus();
171         }
172     }
173     return correcto;
174 }

```



si el método devuelve “true”, nos permite acceder a los datos del paciente, que como se explicó anteriormente están en un arrayList. Primero el botón comprobará que el campo de texto del NUS esta relleno con los datos correctos, si lo está, abrirá el siguiente y último JFrame con los datos personales del paciente. Si no, nos avisará con un mensaje dependiendo de cual es el error por el que no nos deja acceder. Uno de los errores puede ser, que este mal introducido el numero y otro que se haya colado algún simbolo o letra. (137-151)

Siguiente botón al que le daremos funcionalidad es el de ‘Salir’. Cuando cliquemos el botón se cerrará el programa completamente. Y por último, el botón ‘Nuevo Paciente’ que nos abrirá el siguiente JFrame vacio para poder rellenarlo con los datos del paciente.

```

136
137 private void accederJActionPerformed(java.awt.event.ActionEvent evt) {
138     try {
139         if(comprobarNuss() == true){
140             VentanaFormulario ven = new VentanaFormulario(pacientes, nuss, conexion);
141             this.setEnabled(false);
142             this.setVisible(false);
143         }
144     } catch (SQLException ex) {
145         JOptionPane.showMessageDialog(null, "MAL en ventanaEntrada");
146     } catch (NumberFormatException e) {
147         tfNuss.setBackground(Color.red);
148         JOptionPane.showMessageDialog(null, "Solo números por favor");
149     }
150 }
151
152
153 private void salirJActionPerformed(java.awt.event.ActionEvent evt) {
154     System.exit(0);
155 }
156
157 private void btNuevoActionPerformed(java.awt.event.ActionEvent evt) {
158     new VentanaFormulario();
159 }
160

```

Para concluir, la última ventana (VentanaFormulario). Esta ventana tiene dos versiones, dependiendo del botón al que le demos. Si en la ventana anterior le damos al botón de ‘Nuevo Paciente’ nos encontraremos la ventana sin datos, vacía para poder insertar los datos de este nuevo paciente. Y, si le damos a ‘Acceder’ ya nos apareceran todos los campos llenos, ya que ese paciente si que existe en nuestra base de datos.

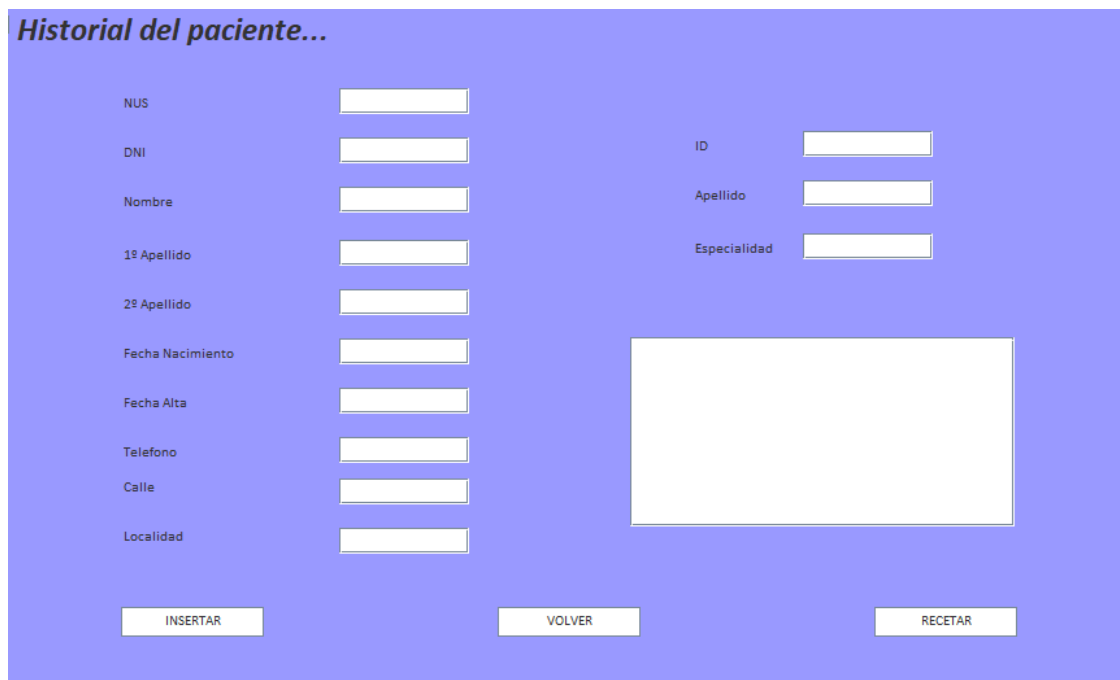
Al igual que en la ventana anterior, necesitamos de la base de datos para coger o insertar datos. Para ello, necesitaremos un ArrayList de pacientes, el NUS, la clase MySQL, el botón de insertar y la conexión con la base de datos (Connection).

```

31 public class VentanaFormulario extends javax.swing.JFrame{
32
33
34     private static ArrayList<Pacientes> pacientes;
35     static int nus = 0;
36     int nusa = 0;
37     private MySQL mysql;
38     private JButton btInsertar;
39     static Connection conexion;

```

Primero, explicaremos la ventana cuando clicamos en el botón 'Nuevo Paciente'. La apariencia de esta es:



Los campos de texto nos aparecen vacíos y editables, para poder rellenarlos con los datos del próximo paciente. Y abajo del todo nos encontramos tres botones, el de 'Insertar', 'Volver' y 'Recetar'.

Los botones 'Volver' y 'Recetar' aparecen tanto en esta ventana como en la que ya tiene los datos de los pacientes, por lo que lo explicaremos lo último.

Al clicar en el botón insertar, llamamos a un método llamado 'insertarBd()'.

```
467 private void jbInsertarActionPerformed(java.awt.event.ActionEvent evt) {  
468     insertarBd();  
469 }
```

El método 'insertarBd()' lo que hace es, coger los valores que hayamos escrito en los campos de texto:

```
562 public void insertarBd(){  
563     PreparedStatement prepared;  
564  
565     String nus = tfNus.getText();  
566     String dni = tfDni.getText();  
567     String nombre = nombreTF.getText();  
568     String primerApellido = apeTF.getText();  
569     String segundoApellido = ape2TF.getText();  
570     int telf = Integer.parseInt(tlfTF.getText());  
571     String calle = calleTF.getText();  
572     String localidad = localidadTF.getText();  
573     Date fechaNacimiento = Date.valueOf(fecnacTF.getText());  
574     Date fechaAlta = Date.valueOf(tfFechaAlta.getText());  
575     int medico = Integer.parseInt(idMedicoTF.getText());  
576     String historial = medicamentosTA.getText();
```

```

577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605

```

```

try {
    prepared = mysql.getConnection().prepareStatement("INSERT INTO paciente (NUS, DNI, nombre, "
        + "primer_apellido, segundo_apellido, tlf, "
        + "Calle, Localidad, fecha_nacimiento, fecha_alta, Medico, Historial)"
        + " VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?)");

    // Parámetros para el preparedStatement
    prepared.setString(1,nus);
    prepared.setString(2,dni);
    prepared.setString(3,nombre);
    prepared.setString(4,primerApellido);
    prepared.setString(5, segundoApellido);
    prepared.setInt(6,telf);
    prepared.setString(7,calle);
    prepared.setString(8,localidad);
    prepared.setDate(9,fechaNacimiento);
    prepared.setDate(10, fechaAlta);
    prepared.setInt(11, medico);
    prepared.setString(12, historial);

    JOptionPane.showMessageDialog(null, "Paciente insertado Correctamente");

    // llamada al método ejecuta update el cual actualiza
    prepared.executeUpdate();
    prepared.close();
}

```

E intentará insertarlo en nuestra base de datos. Si todo ha ido bien, nos saldrá el mensaje ‘Paciente insertado correctamente’.

Con este método ya habríamos explicado todo lo relacionado con el JFrame vacío. Ahora explicaremos el JFrame con datos de la base de datos. La apariencia es esta:

### Historial del paciente...

Datos del Paciente		Médico Asociado	
NUS	12345678	ID	1122
DNI	45921676z	Apellido	Soto
Nombre	Luisa	Especialidad	Maternidad
1º Apellido	Garrido		
2º Apellido	Sanchez		
Fecha Nacimiento	1980-02-03		
Fecha Alta	2000-01-01		
Telefono	958412036		
Calle	C/ Aurora		
Localidad	Granada		

**Medicamentos e Historial**  
 Fiebre

GUARDAR

VOLVER

RECETAR

Para esta ventana, hemos necesitado coger los datos de un paciente en concreto de nuestra base de datos. Como vemos en la siguiente fotografía, hemos necesitado conectar con la base de datos, con el arraylist donde estan la informacion de los pacientes y claramente, con el NUS correspondiente de dicho paciente.

```

40
41 public VentanaFormulario(ArrayList<Pacientes> pacientes, int nuss, Connection conexion) {
42
43     this.conexion = conexion;
44     mysql = new MySQL();
45     initComponents();
46     btInsertar = new JButton("Insertar");
47     this.pacientes = pacientes;
48     setVisible(true);
49     setLocationRelativeTo(null);
50     this.nus = nuss;
51
52     btInsertar.setVisible(false);
53
54     nusa = nuss;
55
56     jpMedico.setBorder(javax.swing.BorderFactory.createTitledBorder("Médico Asociado"));
57     jpMedicamentosHistorial.setBorder(javax.swing.BorderFactory.createTitledBorder("Medicamentos e Historial"));
58     jpDatosPaciente.setBorder(javax.swing.BorderFactory.createTitledBorder("Datos del Paciente"));
59
60     llenarFormulario(nus);
61
62     idMedicoTF.setEditable(false);
63     apeMTF.setEditable(false);
64     tfeEspecialidad.setEditable(false);
65
66
67 }

```

Como vemos, hemos necesitado crear otro método 'llenarFormulario()' al que le pasamos el nus que hemos escrito en la anterior ventana.

Este método lo que hace es llenar los campos de texto con los datos de la base de datos.

```

472 public void llenarFormulario(int nuss){
473     for(int i=0; i<pacientes.size();i++){
474         if(nusa == pacientes.get(i).getNus()){
475             tfNus.setText(String.valueOf(pacientes.get(i).getNus()));
476             tfDni.setText(pacientes.get(i).getDni());
477             nombreTF.setText(pacientes.get(i).getNombre());
478             apeTF.setText(pacientes.get(i).getPrimerApellido());
479             ape2TF.setText(pacientes.get(i).getSegundoApellido());
480             fecnacTF.setText(String.valueOf(pacientes.get(i).getFechaNacimiento()));
481             tfeFechaAlta.setText(String.valueOf(pacientes.get(i).getFechaAlta()));
482             tlfTF.setText(String.valueOf(pacientes.get(i).getTelefono()));
483             calleTF.setText(pacientes.get(i).getCalle());
484             localidadTF.setText(pacientes.get(i).getLocalidad());
485             idMedicoTF.setText(String.valueOf(pacientes.get(i).getMedico()));
486             medicamentosTA.setText(pacientes.get(i).getHistorial());
487         }
488     }
489     ResultSet rsMedico = mysql.ejecutaConsulta("SELECT apellido FROM medico WHERE idM =" +
490                                             "(SELECT Medico FROM paciente where NUS =" + nuss + ")");
491     String medico = "";
492
493     try {
494         while(rsMedico.next()){
495             medico = rsMedico.getString(1);
496         }
497         apeMTF.setText(medico);
498     } catch (SQLException ex) {
499         Logger.getLogger(VentanaFormulario.class.getName()).log(Level.SEVERE, null, ex);
500     }
501
502     ResultSet rsEspecialidad = mysql.ejecutaConsulta("SELECT especialidad FROM medico WHERE idM =" +
503                                             "(SELECT Medico FROM paciente WHERE NUS='"+nusa+"'");
504     String especialidad = "";
505     try {
506         while(rsEspecialidad.next()){
507             especialidad = rsEspecialidad.getString(1);
508         }
509     } catch (SQLException ex) {
510         Logger.getLogger(VentanaFormulario.class.getName()).log(Level.SEVERE, null, ex);
511     }
512     tfeEspecialidad.setText(especialidad);
513 }

```

Y el último método del que hablaremos es el de 'ActualizarBd()' que lo utilizaremos en el botón 'Guardar'.

```

451 private void guardarJBActionPerformed(java.awt.event.ActionEvent evt) {
452     ActualizarBD();
453 }

```

Este método lo que hace es coger los datos que hayamos cambiado en nuestra ventana para guardarlo en la base de datos.

```
514 public void ActualizarBD() {
515     PreparedStatement ps;
516     String nus = tfNus.getText();
517     String dni = tfDni.getText();
518     String nombre = nombreTF.getText();
519     String primerApellido = apeTF.getText();
520     String segundoApellido = ape2TF.getText();
521     int telf = Integer.parseInt(tlfTF.getText());
522     String calle = calleTF.getText();
523     String localidad = localidadTF.getText();
524     Date fechaNacimiento = Date.valueOf(fecnacTF.getText());
525     Date fechaAlta = Date.valueOf(tfFechaAlta.getText());
526     int medico = Integer.parseInt(idMedicoTF.getText());
527     String historial = medicamentosTA.getText();
528
529     System.out.println("pepe"+nus);
```

```
531     try {
532
533         ps = mysql.getConnection().prepareStatement("UPDATE paciente SET NUS= ?, DNI= ?, " + "nombre= ?, primer_apellido= ?, " +
534             "segundo_apellido= ?, tlf= ?, " + "Calle= ?, Localidad= ?, " + "fecha_nacimiento= ?, fecha_alta= ?, " +
535             "Medico= ?, Historial= ?" + "WHERE NUS= ?");
536
```

```
556     } catch (SQLException ex) {
557         Logger.getLogger(VentanaFormulario.class.getName()).log(Level.SEVERE, null, ex);
558     }
559 }
560
```

```
538 // Parámetros para el preparedStatement
539 ps.setString(1,nus);
540 ps.setString(2,dni);
541 ps.setString(3,nombre);
542 ps.setString(4,primerApellido);
543 ps.setString(5, segundoApellido);
544 ps.setInt(6,telf);
545 ps.setString(7,calle);
546 ps.setString(8,localidad);
547 ps.setDate(9,fechaNacimiento);
548 ps.setDate(10, fechaAlta);
549 ps.setInt(11, medico);
550 ps.setString(12, historial);
551 ps.setString(13, nus);
552
553 // llamada al método ejecuta update el cual actualiza
554 ps.executeUpdate();
555 ps.close();
556
```

Y ya por último, la función de los botones ‘Volver y ‘Recetar’. El de ‘Volver’ cierra esta ventana y vuelve a la anterior.

```
455 private void volverJBActionPerformed(java.awt.event.ActionEvent evt) {
456     this.dispose();
457     VentanaEntrada ven = new VentanaEntrada();
458 }
```

Y el de 'Recetar' abre un JDialog donde nos pide una serie de datos, para poder recetar un medicamento a un paciente.

```
460 private void btRecetasActionPerformed(java.awt.event.ActionEvent evt) {  
461  
462     new JDialogMedicamentos(apeMIF.getText(), tfEspecialidad.getText(), nombreTF.getText(),  
463         apeIF.getText(), ape2TF.getText(), tfDni.getText(), nus, mysql.getConnection(), mysql);  
464  
465 }
```

## Ficheros

La clase IOFicheros se encargará de leer y escribir en un fichero de texto los datos que introduce el usuario.

El primer método de la clase es el *"leeFicheroBuffer"*, el cual recibe el nombre del fichero de texto y lee todo el contenido de este, aunque al final solo nos muestra el contenido que le hemos pedido, y el resto lo almacena en memoria.

```
public class IOFicheros {  
  
    public static String leeFicheroBuffer(String fichero, StringBuilder estado) {  
        String lineaActual;  
  
        BufferedReader archivo = null;  
        try {  
            archivo = new BufferedReader(new FileReader(fichero));  
  
            lineaActual = archivo.readLine();  
  
            String resultado = "";  
            while (lineaActual != null) {  
                resultado += lineaActual + "\r\n";  
                lineaActual = archivo.readLine();  
            }  
  
            return resultado;  
        } catch (FileNotFoundException ex) {  
  
        } catch (IOException ex) {  
  
        } finally {  
            try {  
                archivo.close();  
            } catch (Exception ex) {  
            }  
        }  
        return "";  
    }  
}
```

A su vez disponemos del método *"escribeFicheroBuffer"*, la cual recibe unos datos por parte del usuario y los almacena en memoria hasta que tenga bastantes datos como para realizar una escritura eficiente

```

public static int escribeFicheroBuffer(String fichero, String datos) {
    BufferedReader input = null;
    try {
        String linea;
        input = new BufferedReader(new FileReader(fichero));
        linea = input.readLine();
        String linea2 = "";
        while (linea != null) {
            linea2 += linea;
            linea = input.readLine();
        }
    } catch (FileNotFoundException ex) {
        Logger.getLogger(IOFicheros.class.getName()).log(Level.SEVERE, null, ex);
    } catch (IOException ex) {
        Logger.getLogger(IOFicheros.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        try {
            input.close();
        } catch (IOException ex) {
            Logger.getLogger(IOFicheros.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    return 0;
}

```

Por último tenemos los métodos "*leeFichero*" y "*escribeFichero*", las cuales realizan las mismas funciones que las clases anteriores, con la peculiaridad de que estas no almacenan datos en memoria, sino que leen y escriben todo el contenido del fichero de texto

```

public static int escribeFicheroBuffer(String fichero, String datos) {
    BufferedReader input = null;
    try {
        String linea;
        input = new BufferedReader(new FileReader(fichero));
        linea = input.readLine();
        String linea2 = "";
        while (linea != null) {
            linea2 += linea;
            linea = input.readLine();
        }
    } catch (FileNotFoundException ex) {
        Logger.getLogger(IOFicheros.class.getName()).log(Level.SEVERE, null, ex);
    } catch (IOException ex) {
        Logger.getLogger(IOFicheros.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        try {
            input.close();
        } catch (IOException ex) {
            Logger.getLogger(IOFicheros.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    return 0;
}

```



```

public static String leeFichero(String fichero, StringBuilder estado) {
    FileReader reader = null;
    try {
        reader = new FileReader(fichero);
        int c = reader.read();
        StringBuilder sb = new StringBuilder();
        while (c != -1) {
            sb.append((char) c);
            c = reader.read();
        }
        reader.close();
        return sb.toString();
    } catch (FileNotFoundException ex) {
        estado.append("Error 001: El fichero no puede ser encontrado o no existe");
        return "";
    } catch (IOException ex) {
        estado.append("Error 002: El fichero no puede ser leído");
        return "";
    } finally {
        try {
            reader.close();
        } catch (Exception x) {
        }
    }
}
}

```

## Medicamentos

La clase medicamentos está compuesta por un constructor, tres métodos y un atributo de clase.

```
public class Medicamentos {  
    ArrayList<String> medicamentos;
```

En el constructor es donde instanciamos el arraylist

```
public Medicamentos() {  
    medicamentos = new ArrayList<String>();  
}
```

A continuación tenemos los tres métodos, el primero de ellos se encarga de coger los medicamentos del arraylist

```
public String getMedicamentos(int n) {  
    return medicamentos.get(n);  
}
```

El segundo método se encarga de añadir medicamentos al arraylist donde le pasamos el nombre de un medicamento

```
public void insertaMedicamentos(String nomMed) {  
    medicamentos.add(nomMed);  
}
```

Y el tercer método se encarga de imprimir el arraylist de medicamentos

```
public String toString() {  
  
    String cadena = "Medicamentos: ";  
    for(int i=0;i<medicamentos.size();i++) {  
        cadena+=medicamentos.get(i) + " ";  
    }  
    return cadena;  
}
```

## Pacientes

La clase Pacientes va a ser la encargada de trabajar con los datos de los pacientes de la clínica y algunos datos relacionados con estos.

Lo primero que contiene esta clase, es una serie de declaraciones de atributos.

El primer atributo es un ArrayList de Pacientes con el nombre de pacientes. Después se declaran todos los datos del paciente. Estos son el nus, dni, el nombre, el primer apellido, el segundo apellido, la calle, la localidad, el teléfono, la fecha de nacimiento, la fecha de alta, el médico de este y el historial del paciente. Entre ellos está el nus, que va a ser número de identificación del paciente de la seguridad social, el cual se va a usar para acceder a los datos del paciente en el inicio de la aplicación.

```
public class Pacientes {  
  
    private ArrayList<Pacientes> pacientes;  
    private int nus;  
    private String dni;  
    private String nombre;  
    private String primerApellido;  
    private String segundoApellido;  
    private String calle;  
    private String localidad;  
    private int telefono;  
    private Date fechaNacimiento;  
    private Date fechaAlta;  
    private int medico;  
    private String historial;  
}
```

En la siguiente imagen tenemos el constructor de los atributos anteriormente explicados, en el cual, asignamos un valor a estos objetos recién declarados. Una vez declarados y

```
public Pacientes(int nus, String dni, String nombre, String primerApellido, String segundoApellido, int telf, String calle, String localidad, Date fechaNacimiento, Date fechaAlta, int medico, String historial){  
  
    this.nus = nus;  
    this.dni = dni;  
    this.nombre = nombre;  
    this.primerApellido = primerApellido;  
    this.segundoApellido = segundoApellido;  
    this.calle = calle;  
    this.localidad = localidad;  
    this.telefono = telf;  
    this.fechaNacimiento = fechaNacimiento;  
    this.fechaAlta = fechaAlta;  
    this.medico = medico;  
    this.historial = historial;  
    this.pacientes = new ArrayList<>();  
}
```

construidos estos atributos, utilizamos los Setters y Getters, los cuales son métodos de acceso lo que indica que son siempre declarados públicos.

- Los setters nos sirven para asignar un valor inicial a un atributo, pero de forma explícita, además este no retorna nada, por eso siempre es void y solo nos permite dar acceso público a ciertos atributos que deseemos que el usuario pueda modificar.
- Los getters nos sirven para obtener el valor ya asignado a un atributo y utilizarlo para cierto método.

Los métodos que utilizamos a continuación son los que se encargan asignar un valor inicial y obtenerlo de los datos del paciente.

```
public int getNus() {  
    return nus;  
}  
  
public String getDni() {  
    return dni;  
}  
  
public String getNombre() {  
    return nombre;  
}  
  
public String getCalle() {  
    return calle;  
}  
  
public int getTelefono() {  
    return telefono;  
}  
  
public Date getFechaNacimiento() {  
    return fechaNacimiento;  
}  
  
public void setFechaAlta(Date fechaAlta) {  
    this.fechaAlta = fechaAlta;  
}  
  
public Date getFechaAlta() {  
    return fechaAlta;  
}
```

```

public void setNus(int nus) {
    this.nus = nus;
}

public void setDni(String dni) {
    this.dni = dni;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

public void setCalle(String calle) {
    this.calle = calle;
}

public void setTelefono(int telefono) {
    this.telefono = telefono;
}

}

public void getPacientes() {
    for(int i=0; i<pacientes.size();i++){
        pacientes.get(i);
    }
}

public void setPacientes(Pacientes paciente) {
    pacientes.add(paciente);
}

public String getPrimerApellido() {
    return primerApellido;
}

public void setPrimerApellido(String primerApellido) {
    this.primerApellido = primerApellido;
}

public String getSegundoApellido() {
    return segundoApellido;
}

public void setSegundoApellido(String segundoApellido) {
    this.segundoApellido = segundoApellido;
}

public String getLocalidad() {
    return localidad;
}
}

```

```

public void setLocalidad(String localidad) {
    this.localidad = localidad;
}

```

```

public int getMedico() {
    return medico;
}

```

```

public void setMedico(int medico) {
    this.medico = medico;
}

```

```

public String getHistorial() {
    return historial;
}

```

```

public void setHistorial(String historial) {
    this.historial = historial;
}

```

Por último tenemos un método llamado ToString, el cual devuelve una cadena que representa al conjunto de objetos con los cuales hemos trabajado anteriormente.

```

public String toString(){
    return "\n DNI: " + dni + "\n Nombre: " + nombre + "\n Direccion " + calle + "\n Localidad " + localidad + "\n Telef. " + telefono + "\n";
}

```

# MySQL

La clase MySQL realiza una conexión con una base de datos en el lenguaje MySQL.

Lo primero que contiene esta clase son declaraciones de objetos necesarios para el funcionamiento de los métodos que contiene dicha clase, además del controlador necesario para el funcionamiento de ella y el puerto de conexión de la base de datos.

```
public class MySQL {  
  
    static final String DDBC_DRIVER = "com.mysql.jdbc.Driver";  
    static String DB_URL = "jdbc:mysql://localhost:3306/";  
  
    // Credenciales de usuario  
    //     static String USER = "pepe";  
    //     static String PASS = "pepa";  
    static Statement stmt;  
    static private ResultSet rs;  
    static private ResultSetMetaData meta;  
    static private Connection conex;  
    //int numColumnas;
```

La clase "*conecta*" se encarga de realizar la conexión con la base de datos, e informa al usuario en caso de que haya un error de conexión.

```
public Connection conecta(String user, String pass, String db) {  
    try {  
  
        Class.forName("com.mysql.jdbc.Driver");  
        Class.forName(DBDBC_DRIVER);  
  
        conex = DriverManager.getConnection(DB_URL + db, user, pass);  
  
        stmt = conex.createStatement(  
            ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);  
    } catch (Exception e) {  
        e.printStackTrace();  
        JOptionPane.showMessageDialog(null, "Error al conectar con la base de datos");  
    }  
    return conex;  
}
```

Las clase "*getResultSet*", "*getCabecera*" y "*getDatos*" cogen los datos que contiene la base de datos y los nombres de las tablas.

```

public ResultSet getResultSet() {

    return rs;
}

public String[] getCabecera() {

    String[] cabecera = null;
    try {
        int numColumnas = meta.getColumnCount();
        cabecera = new String[numColumnas];
        for (int i = 1; i <= numColumnas; i++) {
            cabecera[i - 1] = meta.getColumnName(i);
        }

        return cabecera;

    } catch (SQLException se) {
        se.printStackTrace();
        return cabecera;
    }
}

public Connection getConexion(){
    return conex;
}

```

```

public Object[][] getDatos() {

    Object[][] datos = null;
    int nFila = 0;

    try {
        rs.last();
        int cuantasFilas = rs.getRow();
        rs.beforeFirst();
        int cuantasColumnas = meta.getColumnCount();
        datos = new Object[cuantasFilas][cuantasColumnas];
        while (rs.next()) {
            // Se crea y rellena la fila para el modelo de la tabla.

            for (int i = 1; i <= cuantasColumnas; i++) {
                datos[nFila][i - 1] = rs.getObject(i);
            }

            nFila++;
        }
        //rs.close();
    } catch (Exception e) {
        e.printStackTrace();
    }

    return datos;
}

```



Las siguientes clases, "*ejecutaConsulta*" y "*ejecutaConsultaAccion*" permiten la ejecución de consultas de inserción, actualización y borrado en la base de datos.

```
public ResultSet ejecutaConsulta(String sql) {
    try {
        rs = stmt.executeQuery(sql);
        meta = rs.getMetaData();
    } catch (SQLException se) {
        JOptionPane.showMessageDialog(null, "Error en la ejecución de la consulta" + se.getMessage());
    }
    return rs;
}

public int ejecutaConsultaAccion(String sql) {
    try {
        Statement stmt = conex.createStatement();
        int filas = 0;

        filas = stmt.executeUpdate(sql);

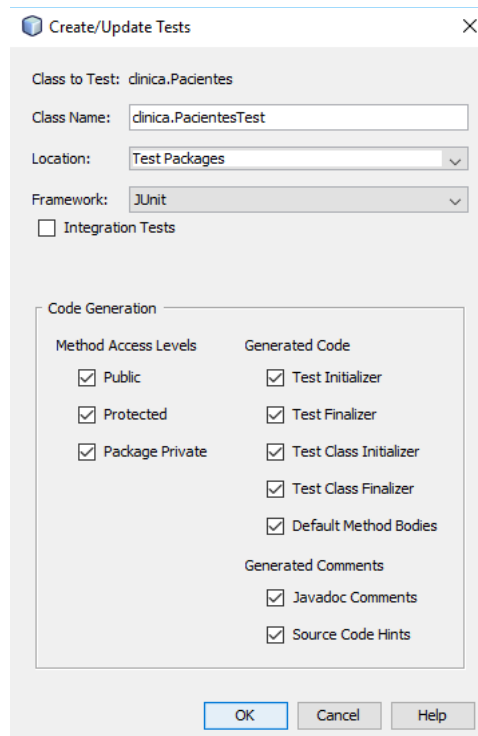
        return filas;
    } catch (SQLException se) {
        JOptionPane.showMessageDialog(null, "Error en la ejecución de la actualización");
        return -1;
    }
}
```

Por último, la clase "*cierra*" finaliza la conexión con la base de datos.

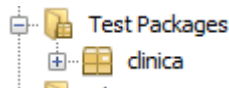
```
public void cierra() {
    try {
        if (conex != null) {
            conex.close();
        }
    } catch (SQLException se) {
        se.printStackTrace();
        JOptionPane.showMessageDialog(null, "Error al cerrar la conexión con la base de datos");
    }
}
```

## Pruebas del Software

Para crear una clase de Test, hay que seleccionar una clase del proyecto pulsar botón derecho del ratón, navegar hasta el menú **Test** y posteriormente **Create/Update tests** y nos aparecerá esta ventana:



Seleccionaremos todas las casillas y al pulsar sobre ok, nos creará un paquete nuevo en nuestro proyecto



En el cual se encuentra nuestra clase con las pruebas con tantos métodos anteceditos de `@Test` como método tuviéramos en nuestra clase original.

Hay que configurar la clase para ejecutar los distintos métodos; En este caso primeramente vamos a crear varios objetos `Pacientes`.

También en todos los métodos que vayamos a probar hay que eliminar la línea que empieza con `Fail` porque si no provocará que nuestras pruebas fallen.

```
fail("The test case is a prototype.");|
```

Cuando hayamos modificado todos los métodos de nuestra clase test, debería quedar algo similar:

```
@Test
public void testGetNus() {
    System.out.println("getNus");
    Pacientes instance = paciente;
    int expectedResult = 14526;
    int result = instance.getNus();
    assertEquals(expResult, result);
}

@Test
public void testGetDni() {
    System.out.println("getDni");
    Pacientes instance = paciente;
    String expectedResult = "45921676z";
    String result = instance.getDni();
    assertEquals(expResult, result);
}

@Test
public void testGetNombre() {
    System.out.println("getNombre");
    Pacientes instance = paciente;
    String expectedResult = "pepe";
    String result = instance.getNombre();
    assertEquals(expResult, result);
}
```

Ahora un objeto Paciente e instanciaremos el arrayList:

```
public class PacientesTest {
    ArrayList<Pacientes> pacientes;
    Pacientes paciente = new Pacientes(14526, "45921676z", "pepe", "serrano", "ramirez", 123456789, "Ciorca", "Granada", Date.valueOf(LocalDate.of(1993, Month.MAY, 31)), Date.valueOf(LocalDate.now()), 0, "Alergia");

    public PacientesTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

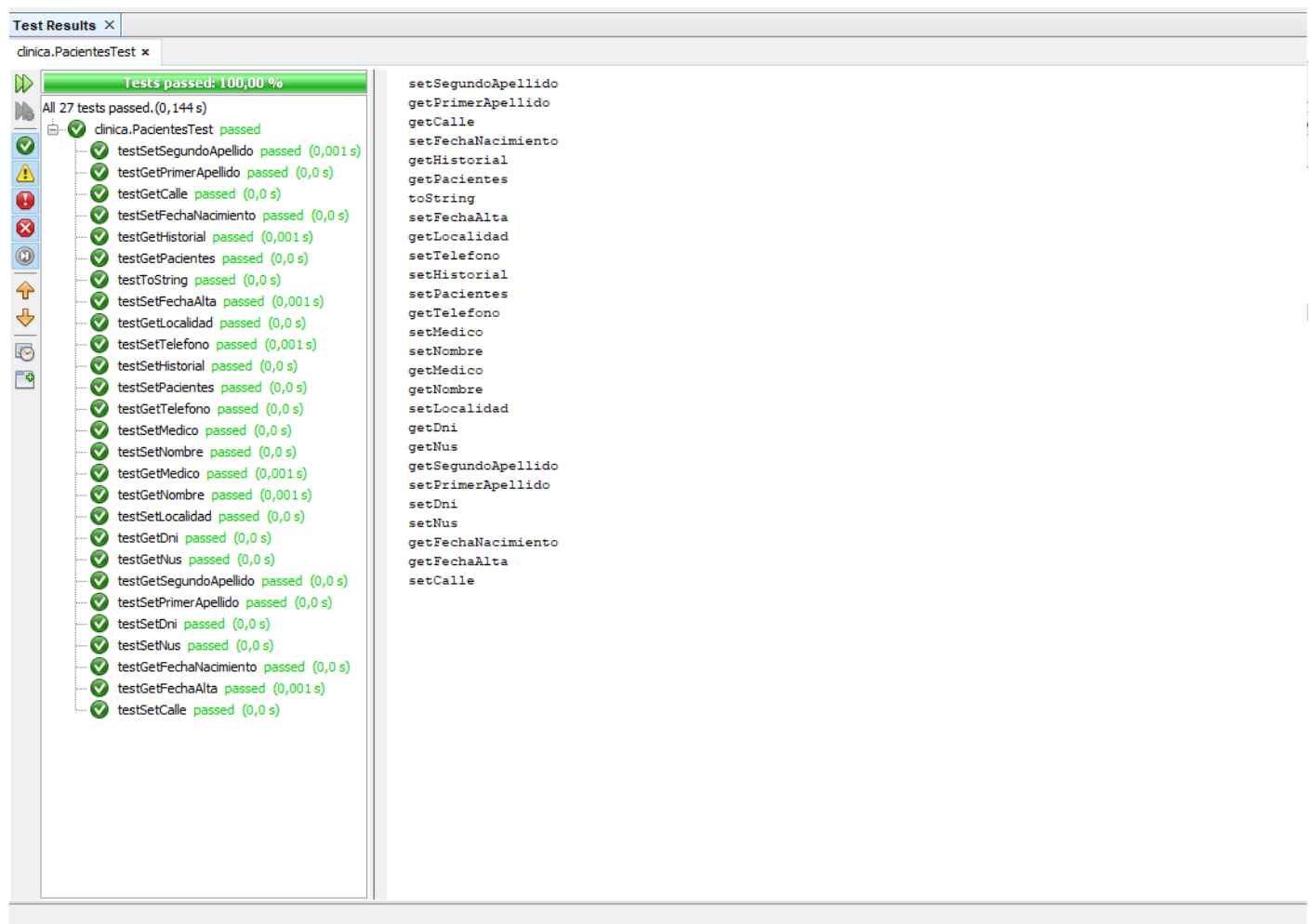
    @Before
    public void setUp() {
        pacientes = new ArrayList<>();
    }

    @After
    public void tearDown() {
    }
}
```

Con nuestro objeto Paciente creado iremos probando los distintos métodos, como por ejemplo el setPacientes() que lo que hará será insertar un paciente en el arrayList:

```
@Test
public void testSetPacientes() {
    System.out.println("setPacientes");
    pacientes.add(paciente);
}
```

Seguidamente iremos probando y configurando los distintos métodos hasta que terminemos de evaluar satisfactoriamente nuestro test:



Este sería nuestro resultado final

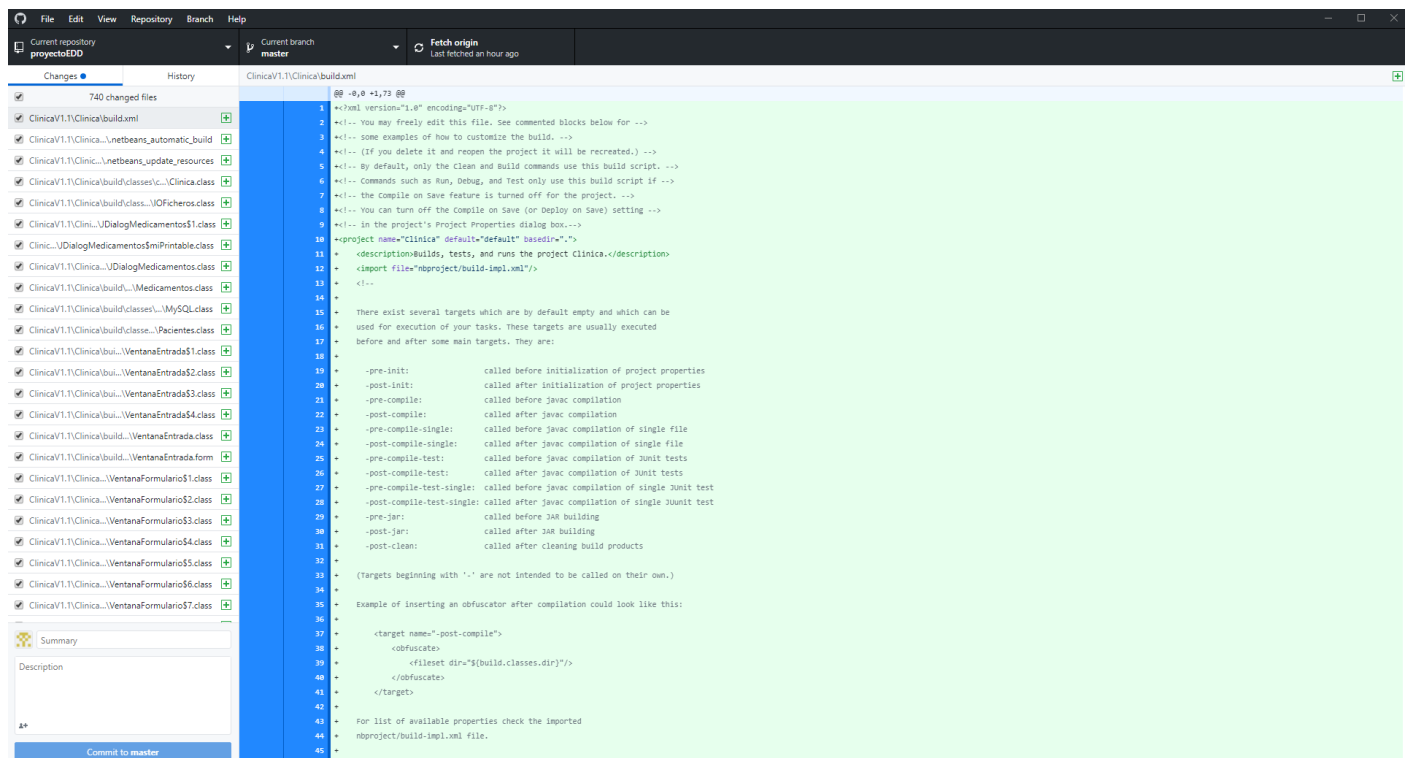
## Control de Versiones

Para el control de versiones, hemos usado gitHub, ya que era lo más cómodo gracias a su interfaz gráfica. En total en el proyecto, hemos realizado 17 versiones diferentes desde la 0.0 a la 1.6

ClinicaV0.0	07/04/2018 13:11	Carpeta de archivos
ClinicaV1.0	07/04/2018 14:23	Carpeta de archivos
ClinicaV1.0.1	11/04/2018 18:17	Carpeta de archivos
ClinicaV1.0.2	12/04/2018 2:24	Carpeta de archivos
ClinicaV1.0.3	17/04/2018 17:37	Carpeta de archivos
ClinicaV1.0.4	18/04/2018 22:08	Carpeta de archivos
ClinicaV1.0.5	20/04/2018 21:22	Carpeta de archivos
ClinicaV1.0.6	26/04/2018 16:49	Carpeta de archivos
ClinicaV1.0.7	03/05/2018 0:23	Carpeta de archivos
ClinicaV1.0.8	03/05/2018 23:54	Carpeta de archivos
ClinicaV1.0.9	04/05/2018 0:10	Carpeta de archivos
ClinicaV1.1	04/05/2018 0:40	Carpeta de archivos
ClinicaV1.2	05/05/2018 11:51	Carpeta de archivos
ClinicaV1.3	05/05/2018 13:03	Carpeta de archivos
ClinicaV1.4	09/05/2018 16:48	Carpeta de archivos
ClinicaV1.5	09/05/2018 17:24	Carpeta de archivos
ClinicaV1.6	05/05/2018 11:04	Carpeta de archivos
.gitignore	21/04/2018 23:19	Archivo GITIGNORE 1 KB
clinicaprivada	04/05/2018 0:01	Archivo SQL 7 KB

Para usar gitHub es necesario registrarse y descargar su aplicación. Una vez descargada e iniciada la sesión

Tendremos esta interfaz que mostrará si hay algún cambio:



Abajo a la izquierda, introducimos un título y una descripción y se habilitará el botón “commit to master” pinchamos ahí y se habilitará arriba “Push origin” pulsamos en push y ya hemos compartido nuestros cambios.

## Unificación

El proyecto consta de 9 clases una principal que instancia VentanaPrincipal, la cual es un botón que instancia VentanaEntrada que muestra un JComboBox rellenándolo con un ResultSet y obteniendo los NUS de los pacientes. Una vez seleccionado el NUS podremos acceder a ver los datos de dicho paciente, instanciando VentanaFormulario y pasando como parámetro un ArrayList cargado con todos los datos de los pacientes a través de un resultSet, en VentanaEntrada:

```
package clinica;

public class Clinica {

    public static void main(String[] args) {
        new VentanaPrincipal();
    }

}

public class VentanaEntrada extends javax.swing.JFrame {
    private MySQL mysql;
    private ResultSet rs;
    private ResultSet resultSet;
    private ResultSet resultSetPacientes;
    private ArrayList<Pacientes> pacientes;
    private Connection conexion;

    public VentanaEntrada() {
        mysql = new MySQL();
        //conexion = mysql.conecta("u5768522_root", "Clinia*123456", "db5768522_clinicaprivada");
        conexion = mysql.conecta("root", "1234", "clinicaprivada");
        pacientes = new ArrayList<>();
        initComponents();
        setVisible(true);
        //setSize(500,600);
        setLocationRelativeTo(null);

        llenarCombo();

        try {
            llenarArrayList();
        } catch (UnsupportedEncodingException ex) {
            Logger.getLogger(VentanaEntrada.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

```
public class VentanaPrincipal extends javax.swing.JFrame {

    public VentanaPrincipal() {
        initComponents();
        setVisible(true);
        setLocationRelativeTo(null);
    }

    @SuppressWarnings("unchecked")
    Generated Code

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        VentanaEntrada ven = new VentanaEntrada();
    }

    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new VentanaPrincipal().setVisible(true);
            }
        });
    }

}

public class VentanaFormulario extends javax.swing.JFrame {

    private static ArrayList<Pacientes> pacientes;
    static String nus = "";
    String nusa = "";
    private MySQL mysql;
    private JButton btInsertar;
    static Connection conexion;

    public VentanaFormulario(ArrayList<Pacientes> pacientes, String nusa, Connection conexion) {

        this.conexion = conexion;
        mysql = new MySQL();
        initComponents();
        btInsertar = new JButton("Insertar");
        this.pacientes = pacientes;
        setVisible(true);
        setLocationRelativeTo(null);
        this.nus = nusa;

        btInsertar.setVisible(false);

        nusa = nus;

        System.out.println("nuss: " + nusa);

        jpMedico.setBorder(javax.swing.BorderFactory.createTitledBorder("Médico Asociado"));
        jpMedicamentosHistorial.setBorder(javax.swing.BorderFactory.createTitledBorder("Medicamentos e Historial"));
        jpDatosPaciente.setBorder(javax.swing.BorderFactory.createTitledBorder("Datos del Paciente"));

        llenarFormulario(Integer.parseInt(nus));

        idMedicoTF.setEnabled(false);
        apeMTF.setEnabled(false);
        tfEspecialidad.setEnabled(false);
    }
}
```

Al haber pasado como parámetro el `ArrayList<Paciente>` tenemos acceso a los distintos métodos que tiene la clase `Paciente` detallada anteriormente. Gracias a estos métodos, nos permite rellenar todo el formulario con los datos personales y clínicos de cada paciente:

### Historial del paciente...

#### Datos del Paciente

NUS	<input type="text" value="12345678"/>
DNI	<input type="text" value="45921676z"/>
Nombre	<input type="text" value="Luisa"/>
1º Apellido	<input type="text" value="Garrido"/>
2º Apellido	<input type="text" value="Sanchez"/>
Fecha Nacimiento	<input type="text" value="1980-02-03"/>
Fecha Alta	<input type="text" value="2000-01-01"/>
Telefono	<input type="text" value="958412036"/>
Calle	<input type="text" value="C/ Aurora"/>
Localidad	<input type="text" value="Granada"/>

#### Médico Asociado

ID	<input type="text" value="1122"/>
Apellido	<input type="text" value="Soto"/>
Especialidad	<input type="text" value="Maternidad"/>

#### Medicamentos e Historial

Fiebre

GUARDAR

VOLVER

RECETAR

Desde este `JFrame` podremos modificar cualquier dato del paciente, excepto su médico ya que esto es una llave Externa de `MySQL` que apunta a la tabla `médico` y de la cual extraemos los datos con un `ResultSet`. Si estos datos se pudieran modificar, podría causar un conflicto en la llave externa que produciría un error fatal en nuestra aplicación.

Todas las inserciones y modificaciones que se hacen en la `BD`, se llevan a cabo con `PreparedStatement`, esto nos permite más flexibilidad y seguridad a la hora de ejecutar nuestras consultas.

```
PreparedStatement ps;
String nus = tfNus.getText();
String dni = tfDni.getText();
String nombre = nombreTF.getText();
String primerApellido = ape1TF.getText();
String segundoApellido = ape2TF.getText();
int telf = Integer.parseInt(tlfTF.getText());
String calle = calleTF.getText();
String localidad = localidadTF.getText();
Date fechaNacimiento = Date.valueOf(fecnacTF.getText());
Date fechaAlta = Date.valueOf(tfFechaAlta.getText());
int medico = Integer.parseInt(idMedicoTF.getText());
String historial = medicamentosTA.getText();

System.out.println("pepe"+nusa);

try {

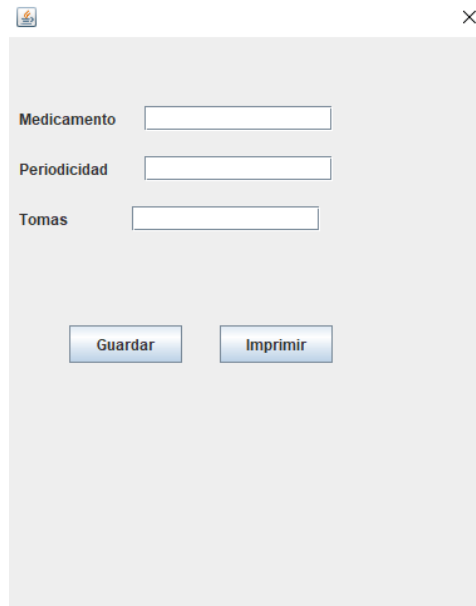
    ps = mysql.getConnection().prepareStatement("UPDATE paciente SET NUS= ?, DNI= ?, " + "nombre= ?, primer_apellido= ?, " +
        "segundo_apellido= ?, tlf= ?, " + "Calle= ?, Localidad= ?, " + "fecha_nacimiento= ?, fecha_alta= ?, " +
        "Medico= ?, Historial= ?" + "WHERE NUS= ?");

    // Parámetros para el preparedStatement
    ps.setString(1,nus);
    ps.setString(2,dni);
    ps.setString(3,nombre);
    ps.setString(4,primerApellido);
    ps.setString(5, segundoApellido);
    ps.setInt(6,telf);
    ps.setString(7,calle);
    ps.setString(8,localidad);
    ps.setDate(9,fechaNacimiento);
    ps.setDate(10, fechaAlta);
    ps.setInt(11, medico);
    ps.setString(12, historial);
    ps.setString(13, nus);
```

El botón volver, simplemente no devuelve a la ventanaEntrada

```
private void volverJActionPerformed(java.awt.event.ActionEvent evt) {  
    this.dispose();  
    VentanaEntrada ven = new VentanaEntrada();  
}
```

Por último el botón recetar, instancia un JDialog en el cual podremos recetar un medicamento a nuestro paciente e imprimir dicha receta:



El botón de guardar, nos genera un archivo de texto con el contenido de los textField usando para ello la clase IOFicheros explicada anteriormente.

También el botón de guardar, hacer una inserción en la base de datos en la tabla recetas y dicha receta estará asociada a un médico y a un paciente.

```
if(e.getSource() == btGuardar){  
    //public void actionPerformed(ActionEvent e) {  
        PreparedStatement prepared;  
        ResultSet rs = mysql.ejecutaConsulta("SELECT * FROM medico");  
  
        int medicamento = 0;  
        int idMedico = 0;  
        int paciente = 0;  
        Date fechaReceta = null;  
  
        medicamento = Integer.valueOf(tfMedicamentos.getText());  
        try {  
            while(rs.next()){  
                idMedico = rs.getInt(1);  
            }  
        } catch (SQLException ex) {  
            Logger.getLogger(JDialogMedicamentos.class.getName()).log(Level.SEVERE, null, ex);  
        }  
        paciente = Integer.parseInt(nus);  
        fechaReceta = Date.valueOf(LocalDate.now());  
  
        try {  
            prepared = mysql.getConexion().prepareStatement("INSERT INTO receta (idMedicamento, idMedico, Paciente, fecha_receta)"  
                + " VALUES(?, ?, ?, ?)");  
  
            // Parámetros para el preparedStatement  
            prepared.setInt(1, medicamento);  
            prepared.setInt(2, idMedico);  
            prepared.setInt(3, paciente);  
            prepared.setDate(4, fechaReceta);  
        }
```



En el botón imprimir creamos un objeto PageFormat y un PrinterJob que es el que será enviado a la impresora seleccionada.

Establecemos que sea imprimible, le pasamos como parámetro nuestra clase interna en la cual redefinimos el método print y el PageFormat y seleccionamos el dialogo de impresión. Por último imprimimos.

```
try {
    PrinterJob job = PrinterJob.getPrinterJob();
    PageFormat pf = job.defaultPage();
    Paper paper = new Paper();

    job.setPrintable(new miPrintable(), pf);
    job.printDialog();
    job.print();
} catch (PrinterException ex) {
    Logger.getLogger(JDialogMedicamentos.class.getName()).log(Level.SEVERE, null, ex);
}
```

En la clase interna donde redefinimos el método printable lo primero que hacemos es un If si la página es igual a 0 para saber si es la primera. Si lo es, creamos un objeto Graphics2D y le asignamos el valor del parámetro Graphics que se nos pasa haciendo un casting.

Tras ello, usamos el método Translate() de Graphics2D para traducir (mover) cada punto desde Graphics2D al punto correspondiente del formato de página (PageFormat) pasándole los valores del eje X y el eje Y.

```
public class miPrintable implements Printable {
    @Override
    public int print(Graphics graphics, PageFormat pageFormat, int pageIndex) throws PrinterException {
        String cadena = "";
        if (pageIndex == 0) {
            Graphics2D g2d = (Graphics2D) graphics;

            //Punto donde empezará a imprimir dentro la pagina (100, 50)
            g2d.translate(pageFormat.getImageableX() + 50,
                pageFormat.getImageableY() + 70);
```

Una vez hecho esto, cambiamos la escala de la página un 50% con el método scale() y comenzamos a escribir en la página con los métodos drawLine() para hacer líneas, drawString() para escribir Strings en la página y drawImage() para colocar una imagen.

```
g2d.scale(0.50, 0.50); //Reducción de la impresión al 50%
//Líneas información paciente
g2d.drawLine(410, -100, 800, -100);
g2d.drawLine(410, -100, 410, 70);
g2d.drawLine(800, -100, 800, 70);
g2d.drawLine(410, 70, 800, 70);
//Líneas información prescripcion
g2d.drawLine(20, 190, 780, 190);
g2d.drawLine(20, 190, 20, 395);
g2d.drawLine(780, 190, 780, 395);
g2d.drawLine(20, 395, 780, 395);
//Líneas firma y sello
g2d.drawLine(590, 202, 770, 202);
g2d.drawLine(590, 202, 590, 383);
g2d.drawLine(770, 202, 770, 383);
g2d.drawLine(590, 383, 770, 383);
//Líneas tijeras
for (int i = 0; i < 24; i++) {
    g2d.drawLine(-60+x*i, 523, -40+x*i, 523);
}
```

```

//Pie
g2d.drawImage(imagenTijera, -100, 510, null);
g2d.setFont(fuenteFirma);
g2d.setColor(Color.BLUE);
g2d.drawString("Clínica privada de especialidades médicas", 200,440);
g2d.drawString("C/Primavera, 26-28, 18008 Granada", 225,460);
g2d.drawString("Teléfono: 958 89 38 50", 285,480);

/*Con esta línea imprimimos el JDialog completo*/
//jdMedicamentos.printAll(graphics);

return PAGE_EXISTS;

} else {
    return NO_SUCH_PAGE;
}

```

Terminamos de redefinir el método haciendo que devuelva PAGE\_EXISTS en caso de que se escriba la página o NO\_SUCH\_PAGE en caso de que no se haga nada.

Una vez compilado el software, para ejecutarlo necesitamos Java Enviroment y nuestro archivo Jar por defecto Netbeans lo crea en la carpeta dist dentro del proyecto

## Conclusiones

El trabajo me ha ayudado bastante a coger nuevas perspectivas a la hora de trabajar en un proyecto, también me ha ayudado a corregir algunos errores muy básicos que tenía y me ha enseñado que la realización de un proyecto lleva su tiempo y trabajo

Este proyecto me ha servido para afianzar conocimientos en general de toda la asignatura así como la asignatura de programación. Como estructurar un proyecto con varios desarrolladores y como organizar el trabajo entre todos.

En mi opinión hacer este proyecto en grupo nosotros desde 0 sin ayuda ha sido una idea bastante acertada ya que nos ha ayudado mucho a pensar y ver cómo crear y estructurar nosotros mismos un proyecto totalmente nuevo, y en lo personal, sobre todo en mi parte me ha servido mucho para aprender más sobre el printable.

Este trabajo en grupo me ha ayudado a entender mejor algunos conceptos que no entendía muy bien. Además, el grupo de trabajo, en mi opinión, se ha coordinado muy bien y ha trabajado de forma muy compenetrada y sin tensiones, así que la sensación es que hemos realizado un buen trabajo y que hemos hecho un programa que puede tener utilidad real perfectamente con un par de retoques.

Me ha ayudado a trabajar en grupo y mejorar mi nivel de programación, además no he tenido ningún problema con los demás a la hora de organizar el trabajo y he estado conforme con mi parte.

El trabajo en grupo no ha estado mal, nos hemos dividido bastante bien las tareas y nos hemos ayudado unos a otros. En mi opinión, el trabajo nos ha quedado bastante bien y creo que todos hemos aprendido bastante.