

Prácticas de Laboratorio

Práctica 5 Interfaz Gráfica de Usuario

Ingeniería
del Software
ETS Ingeniería
Informática
DSIC - UPV

*(Guía de la
práctica)*

0. Objetivo

El objetivo de esta práctica es aplicar los conocimientos adquiridos en asignaturas relacionadas sobre el diseño y construcción de IGU para construir la capa de presentación o interacción con el usuario en aplicaciones Java.

En las prácticas anteriores se ha diseñado e implementado la capa lógica y de persistencia de una parte del sistema **“Servicio de emergencia”**. En esta práctica se propone la construcción de la capa de presentación de esa parte del sistema, mediante el diseño e implementación de una pequeña IGU, que incluye la comunicación con la capa lógica:

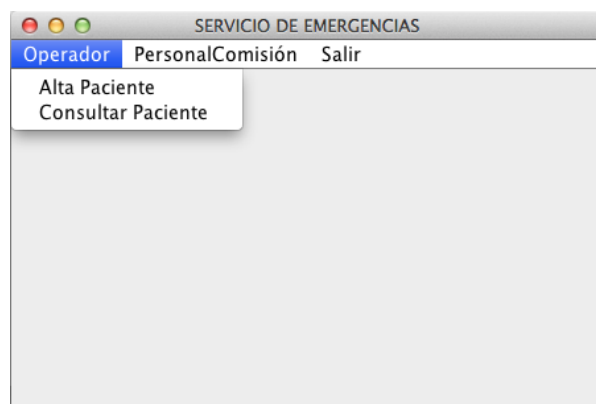


Figura 1. Ventana principal de la aplicación

En esta práctica se asume que el alumno tiene los conocimientos necesarios sobre construcción de IGU en Java, en concreto de la librería Swing y el modelo de gestión de eventos de Java. En caso contrario, se recomienda realizar antes el tutorial sobre WindowBuilder.

Descripción del trabajo a realizar

Partiendo del proyecto desarrollado en las sesiones anteriores, se va a implementar la capa de presentación (IGU). La aplicación resultante proporcionará la funcionalidad para dos casos usos: Alta Paciente y Listar Pacientes. Los pasos que se van a seguir desde la construcción de la IGU a la implementación de la capa de persistencia son los siguientes:

1. Creación de la ventana principal.
2. Creación del formulario Alta Paciente.
3. Creación de la clase Data Access Layer (DAL) que proporcionar los servicios para insertar un nuevo paciente en la base de datos.

4. Creación de la clase Controlador de la capa lógica que proporciona los servicios para dar de alta un nuevo paciente.
5. Gestión de eventos en la IGU.
6. Repetir los pasos para implementar el caso de uso Listar Pacientes
 - a. Creación del formulario Listar Pacientes.
 - b. Extender la clase DAL con los servicios para recuperar los pacientes almacenados en la base de datos.
 - c. Extender la clase Controlador con los servicios para listar los pacientes existentes.
 - d. Añadir los manejadores de eventos en la IGU.

1. Creación de la ventana principal

En el proyecto construido en las prácticas anteriores añada un nuevo paquete llamado *presentacion* que contendrá las clases que implementan la IGU.

Se creará un menú para cada uno de los actores o usuarios de la aplicación. Las opciones de cada menú se corresponderán con las funciones o casos de uso que puede iniciar el actor correspondiente:

- Menú Operador: Alta Paciente, Consultar Paciente, etc.
- Menú PersonalComisión: Listar Pacientes, etc.
- Menú Salir

Cree una nueva clase visual para diseñar la ventana principal de la aplicación. Para ello utilice el asistente: File | New | Other ... | WindowBuilder | Swing Designer. Escoja como estilo *Application Window*. Llame a dicha clase *EmergenciasApp*. La clase contiene el método `main()` desde donde arrancará la aplicación.

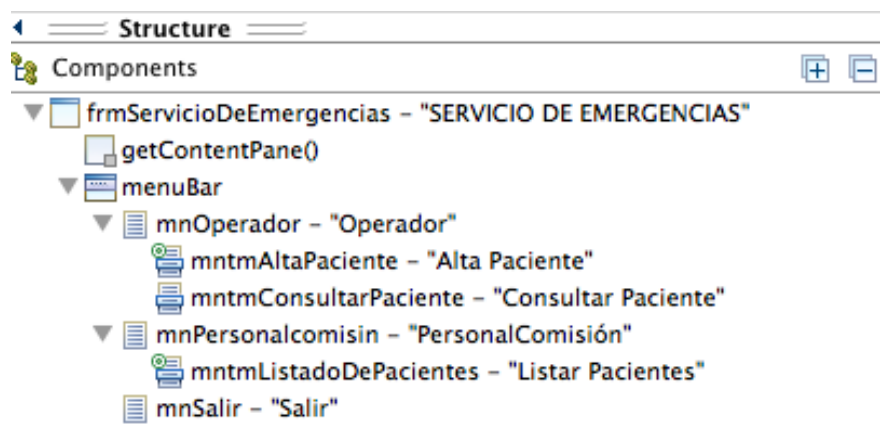


Figura 2. Estructura de componentes de la IGU

Añada la barra de menús, un menú por cada actor y las correspondientes opciones de menú por cada caso de uso. En la figura 2 puede ver la estructura de componentes resultantes.

2. Creación del formulario Alta Paciente.

Cree una ventana (JDialog) para solicitar los datos necesarios para poder da de alta un paciente. Darle como nombre *AltaPacienteJDialog*. Marque la opción *Generate JDialog with OK and Cancel buttons*.

Renombre los botones y modifique el layout de contentPanel para indicar *Absolute*. Distribuya los componentes gráficos, etiquetas (JLabel), cuadros de texto (JTextField), botones (JRadioButton), etc. en las posiciones que se desee. El formulario debe ser similar al de la figura 3:

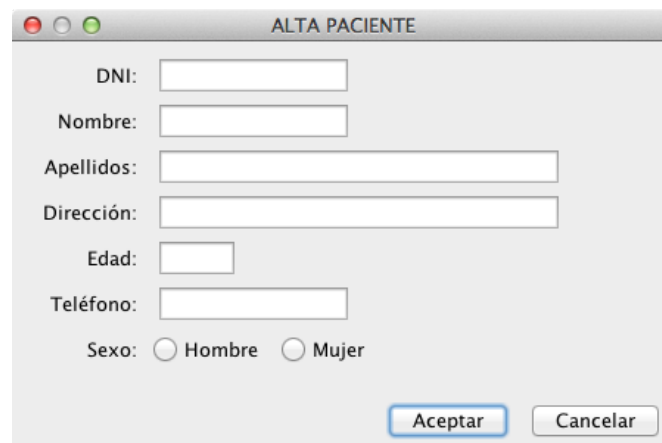
The image shows a Java Swing dialog box titled "ALTA PACIENTE". It has a standard Mac OS-style title bar with red, yellow, and green window control buttons. The dialog contains several text input fields: "DNI:", "Nombre:", "Apellidos:", "Dirección:", "Edad:", and "Teléfono:". Below these fields are two radio buttons labeled "Sexo: Hombre" and "Sexo: Mujer". At the bottom right of the dialog are two buttons: "Aceptar" (highlighted with a blue border) and "Cancelar".

Figura 3. Formulario Alta Paciente

3. Creación de la clase Data Access Layer (DAL) que proporcionar los servicios para insertar un nuevo paciente en la base de datos.

El objeto DAL actúa como interfaz entre la capa lógica (en particular, la clase *ServicoEmergencia*) y los objetos DAO que implementan la capa de persistencia de la aplicación. Los objetos DAL proporcionan servicios a la capa lógica que implementan parte de la funcionalidad de los casos de uso. Para aglutinar todo el acceso a la capa de datos se creará un único objeto DAL, que gestione todas las peticiones de servicio que tengan que ver con el acceso a datos y que trabajará con los DAO.

El objeto DAL implementa el patrón *singleton*: restringe la instanciación de la clase a una única instancia. Esto es útil cuando es necesario únicamente un objeto para coordinar una serie de acciones en el sistema, a través de un único punto de acceso. El patrón *singleton* se implementa así:

- Definiendo un constructor privado. De esta forma no pueden crearse instancias desde fuera de la clase.
- Proporcionando un método público estático, *dameDAL()*, que instancia el objeto DAL únicamente la primera vez que se invoca.

Defina una nueva clase llamada DAL en la capa de acceso a datos con el código siguiente.

```
/**
//Data Access Layer. Capa de indirección para gestionar el acceso a datos
//
public class DAL {
    private static DAL dal;

    // Declaración de los DAO
    IPacienteDAO pacienteDAO;

    // constructor privado
    private DAL() {

    }

    // Patrón Singleton
    public static DAL dameDAL() {
        if(dal==null)
            dal = new DAL();
        return dal;
    }

    // Servicios para el C.U. Alta Paciente
    public Paciente buscarPaciente(String dni) throws DAOExcepcion {

        pacienteDAO = new PacienteDAOImp();
        return pacienteDAO.buscarPaciente(dni);

    }
    public void crearPaciente(Paciente p) throws DAOExcepcion {

        pacienteDAO = new PacienteDAOImp();
        pacienteDAO.crearPaciente(p);

    }

}
```

Como se puede observar, en la implementación de los distintos servicios se inicializa el objeto DAO necesario para gestionar la persistencia.

4. Creación de la clase Controlador de la capa lógica que proporciona los servicios para dar de alta un nuevo paciente.

Para controlar la interacción entre los objetos de la lógica del negocio y la IGU ha que utilizarse un objeto controlador por cada caso de uso. Para simplificar la implementación, se va a definir una única clase que englobe todos los servicios que la capa lógica ofrece a la capa de presentación.

La clase *Controlador* se creará en el paquete *lógica*. Se implementará también utilizando el patrón *singleton*. La clase *Controlador* implementará los servicios que necesita la capa de presentación haciendo uso de la clase *ServicioEmergencia*.

```
public class Controlador {

    private ServicioEmergencia servicioEmergencia;
    private static Controlador controlador = null;

    /**
     * Creación del controlador
     * Un único controlador para todos los C.U.
     * Los servicios se ejecutan a través de la clase ServicioEmergencia
     */
    private Controlador() {
        servicioEmergencia = new ServicioEmergencia();
    }

    public static Controlador dameControlador(){
        if (controlador==null) controlador = new Controlador();
        return controlador;
    }

    public void AltaPaciente(Paciente pa) throws LogicaExcepcion{

        // Los servicios se implementaron en la clase ServicioEmergencia en las sesiones
        previas
        servicioEmergencia.añadirPaciente(pa);

    }

}
```

Una vez implementado esta parte del controlador, la clase *ServicioEmergencia* tiene que ser modificada: la comunicación con la capa de persistencia se realizará mediante el DAL, en lugar de utilizar los

objetos DAO. Así la capa lógica es independiente de una implementación particular de los DAO, de manera que cualquier cambio en la fuente de datos (de una BD relacional a una BD objetual, por ejemplo) es transparente a la capa lógica. El código de la clase *ServicioEmergencia* quedará así:

```
public class ServicioEmergencia {
    private HashMap<String, Paciente> listaPacientes;

    public ServicioEmergencia() {
        this.listaPacientes = new HashMap<String, Paciente>();
    }

    public void añadirPaciente(Paciente paciente) throws LogicaExcepcion
    {
        //Buscamos el paciente
        if (buscarPaciente(paciente.getDni())==null)
        {
            try {

                //Si no lo tenemos lo añadimos
                this.listaPacientes.put(paciente.getDni(), paciente);
                DAL.dameDAL().crearPaciente(paciente);
            }
            catch (DAOExcepcion e){
                e.printStackTrace();
            }
        }
        else throw new LogicaExcepcion("El paciente ya existe.");
    }

    public Paciente buscarPaciente(String dni)
    {
        //Busco el paciente en memoria
        Paciente paciente = this.listaPacientes.get(dni);

        //El paciente no esta en memoria, se busca en la BD
        if (paciente==null)
        {
            try {
                paciente = DAL.dameDAL().buscarPaciente(dni);

                //Si lo he encontrado en la BD, lo añado a memoria porque no lo
                tenía
                if (paciente != null) this.listaPacientes.put(dni, paciente);
            }
            catch (DAOExcepcion e){
                e.printStackTrace();
            }
        }

        //Devuelvo el paciente que he encontrado o null en caso contrario
        System.out.println(paciente);
        return paciente;
    }

    public List<Paciente> ListarPacientes() throws LogicaExcepcion
    {
        //Se obtienen los pacientes de la BD, porque no hay ninguna garantía de tenerlos
        todos en memoria
    }
}
```

```

    try {
        List<Paciente> lista = DAL.dameDAL().listarPacientes();

        //Para cada paciente de la BD, si no esta en memoria lo añadimos
        for(Paciente p:lista)
            if (!this.listaPacientes.containsKey(p.getDni()))
                this.listaPacientes.put(p.getDni(), p);
    }
    catch (DAOExcepcion e) {
        e.printStackTrace();
        throw new LogicaExcepcion("La lista de pacientes no puede ser
recuperada.");
    }
    return new ArrayList<Paciente>(listaPacientes.values());
}
}

```

El código captura excepciones de la capa de persistencia, *DAOExcepcion*, y las convierte en excepciones de la capa lógica, *LogicaExcepcion*.

Añada la clase *LogicaExcepcion* en el paquete *Excepciones*:

```

public class LogicaExcepcion extends Exception{

    public LogicaExcepcion(String message) {
        super(message);
        // TODO Auto-generated constructor stub
    }

    public LogicaExcepcion(Exception e) {
        super(e.getMessage());
        // TODO Auto-generated constructor stub
    }
}

```

5. Gestión de eventos en la IGU.

La última parte a implementar es la gestión de eventos en la capa de presentación y la comunicación con la capa lógica a través del objeto *Controlador*.

Asocie al botón *Aceptar* el manejador del evento, método *ActionPerformed()* del evento *ActionListener()*, que se encargará de leer la información introducida por el operador en el formulario e invocar al servicio *añadirPaciente(Paciente p)* del *Controlador*.

Puede añadir el manejador del evento desde la ventana de diseño del formulario en *WindowBuilder*. Pulse con el botón derecho del ratón y escoja la opción *Add event handler > Action > actionPerformed*.

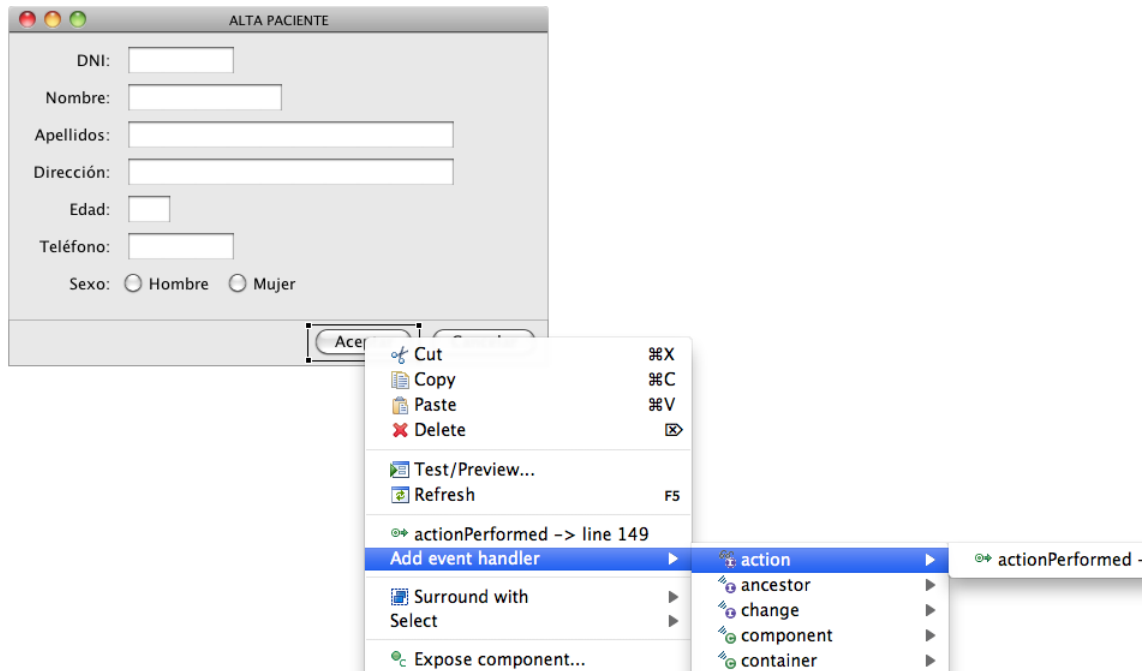


Figura 4. Añadir el manejador del evento actionPerformed al botón Aceptar.

Observe que el código espera unos nombres concretos para los componentes en los que se captura la información (*textFieldCodigo*, *textFieldDireccion*, ...). Una vez creada la instancia de la clase *Paciente*, se invoca al servicio de la capa lógica para añadirlo a la base de datos (*Controlador.dameControlador().AltaPaciente(pa)*).

```
okButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        try {
            Paciente pa = new Paciente (textFieldDNI.getText(),
                textFieldNombre.getText(),
                textFieldApellidos.getText(),
                textFieldDireccion.getText(),
                Integer.parseInt(textFieldTelefono.getText()),
                Integer.parseInt(textFieldEdad.getText()),
                rdbtnHombre.isSelected() ? (char) rdbtnHombre.getText().charAt(0):
                (char) rdbtnMujer.getText().charAt(0));
            Controlador.dameControlador().AltaPaciente(pa);
            JOptionPane.showMessageDialog(null, "Paciente añadido a la BD", "",
                JOptionPane.INFORMATION_MESSAGE);
            dispose();
        }
        catch (Exception e){
            JOptionPane.showMessageDialog(null, e.getMessage(),
                "Error",JOptionPane.ERROR_MESSAGE);
        }
    }
});
```

Igualmente puede asociar un manejador al botón Cancelar que cierre la ventana del formulario.

Finalmente, para abrir el formulario desde la ventana principal, tiene que añadir el correspondiente evento a la opción de menú Alta Paciente. El manejador creará una instancia de *AltaPacienteJDialog* y hará visible la ventana, como muestra el código:

```
mntmAltaPaciente.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        AltaPacienteJDialog altaPaciente = new AltaPacienteJDialog();  
        altaPaciente.setModal(true);  
        altaPaciente.setVisible(true);  
    }  
});
```

En este punto, pruebe el código implementado y verifique que el caso de uso Alta Paciente funciona correctamente.

6. Repetir los pasos para implementar el caso de uso Listar Pacientes

Para implementar el caso de uso Listar Pacientes tiene que repetir los pasos anteriores como se detallará a continuación. Para hacer el acceso a los pacientes más fácil la clase *ServicioEmergencias* se ha extendido con una **agregación de Pacientes** (ya utilizada en la práctica anterior). De esta forma no es necesario acceder a los pacientes a través de la clase *RegistroEmergencia*.

a) Creación del formulario Listar Pacientes.

En esta ventana, ver figura 5, se desean mostrar todos los pacientes que estén dados de alta en el sistema. Cree una nueva ventana *JDialog* llamada *ListarPacientesJDialog* que contendrá una tabla (*JTable*) llamada *tablePacientes*. Recuerde que la tabla debe estar incluida en un panel *JScrollPane*.

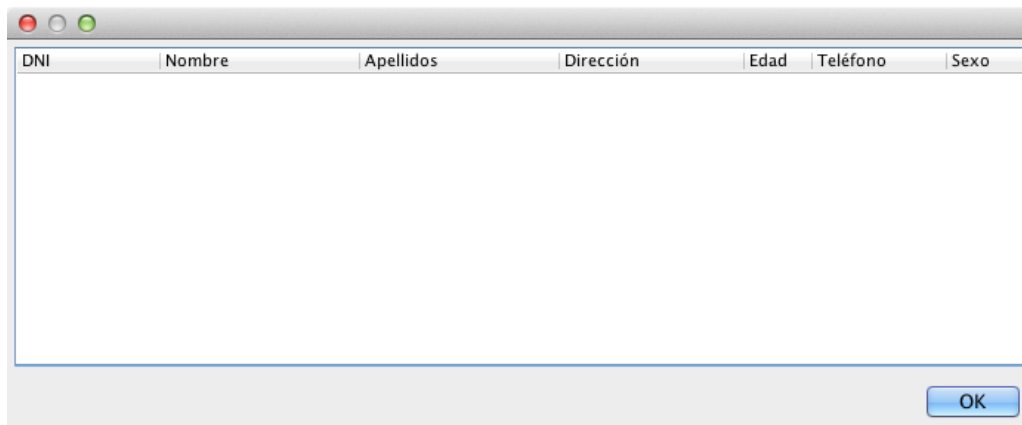


Figura 5. Formulario Listar Pacientes

Para que la tabla pueda mostrar de forma adecuada la información de cada columna, es necesario cambiar el modelo de tabla. Para ello defina la siguiente clase interna dentro de la clase *ListarPacientesJDialog*:

```

/*****
// Clase interna para del Modelo de la tabla
*****/
class PacienteTableModel extends AbstractTableModel {

    private static final long serialVersionUID = 1L;

    private String[] columnNames = { "DNI", "Nombre", "Apellidos", "Dirección",
    "Edad", "Teléfono", "Sexo"};

    private ArrayList<Paciente> data=new ArrayList<Paciente>();

    public int getColumnCount() {
        return columnNames.length;
    }

    public int getRowCount() {
        return data.size();
    }

    public String getColumnName(int col) {
        return columnNames[col];
    }

    public Object getValueAt(int row, int col) {
        Paciente in =data.get(row);
        switch(col){
            case 0: return in.getDni();
            case 1: return in.getNombre();
            case 2: return in.getApellidos();
            case 3: return in.getDireccion();
            case 4: return in.getEdad();
            case 5: return in.getTelefono();
            case 6: return in.getSexo();
            default: return null;
        }
    }

    public void clear(){
        data.clear();
    }
}

```

```

    }

    public Class<? extends Object> getColumnClass(int c) {
        return getValueAt(0, c).getClass();
    }

    public void addRow(Paciente row) {
        data.addRow();
        this.fireTableDataChanged();
    }

    public void delRow(int row) {
        data.remove(row);
        this.fireTableDataChanged();
    }
}

```

Los objetos JTable y ScrollPane se inicializan en el constructor del cuadro de diálogo así:

```

public ListarPacientesJDialog() {
    ...

    JScrollPane scrollPane = new JScrollPane();
    contentPanel.add(scrollPane);

    tablePacientes = new JTable(new PacienteTableModel());
    scrollPane.setViewportView(tablePacientes);
    ...
}

```

b) Extender la clase DAL con los servicios para recuperar los pacientes almacenados en la base de datos.

Añada el siguiente método en la clase DAL, que recupera la lista de pacientes a través del correspondiente objeto DAO:

```

//*****
// Servicios para el C.U. Listar Pacientes
//*****

    public List<Paciente> listarPacientes() throws DAOExcepcion {

        pacienteDAO = new PacienteDAOImp();
        return pacienteDAO.listarPacientes();
    }

```

c) Extender a la clase Controlador con los servicios para listar los pacientes existentes.

Añada el siguiente método en la clase Controlador, que accede a la lista de pacientes a través de la clase *ServicioEmergencia*:

```
public List<Paciente> ListarPacientes() throws LogicaExcepcion{
    return servicioEmergencia.ListarPacientes();
}
```

d) Añadir lo manejadores de eventos en la IGU

Finalmente, falta añadir el código para llenar la tablas con los datos de los pacientes para que puedan ser mostrados e implementar el manejador que muestra la tabla.

Incluya en la clase *ListarPacientesJDialog* el siguiente método que será invocado cuando se visualice la tabla:

```
public void cargaPacientes(){
    try{
        ArrayList<Paciente> listaPacientes= (ArrayList<Paciente>)
Controlador.dameControlador().ListarPacientes();
        Iterator<Paciente>it= listaPacientes.iterator();

        Paciente pa;
        PacienteTableModel model=(PacienteTableModel) tablePacientes.getModel();
        model.clear();
        while (it.hasNext()){
            pa=it.next();
            model.addRow(pa);
        }
    }catch (Exception e){
        JOptionPane.showMessageDialog(this,e.getMessage(),"ERROR",
JOptionPane.ERROR_MESSAGE);
    }
}
```

Sólo falta implementar el manejador asociado a la opción de menú PersonalComisión > Listar Pacientes, que incluirá el siguiente código:

```
ListarPacientesJDialog listarPacientes = new ListarPacientesJDialog();
listarPacientes.setModal(true);
listarPacientes.cargaPacientes();
listarPacientes.setVisible(true);
```

En este punto, compruebe que el caso de uso se ha implementado correctamente. La tabla mostrará los pacientes dados de alta en el sistema:

DNI	Nombre	Apellidos	Dirección	Edad	Teléfono	Sexo
11111111A	María	Martínez Sánchez	Avenida del Puerto 32	67	111111111	M
12345678Y	Luis	Pérez Gómez	Gran vía 12	32	888888888	H

OK

Figura 6. Tabla con el listado de pacientes