

DESARROLLO DE APLICACIONES DE ESCRITORIO CON LENGUAJE



FAVA - Formación en Ambientes Virtuales de Aprendizaje

SENA - Servicio Nacional de Aprendizaje.

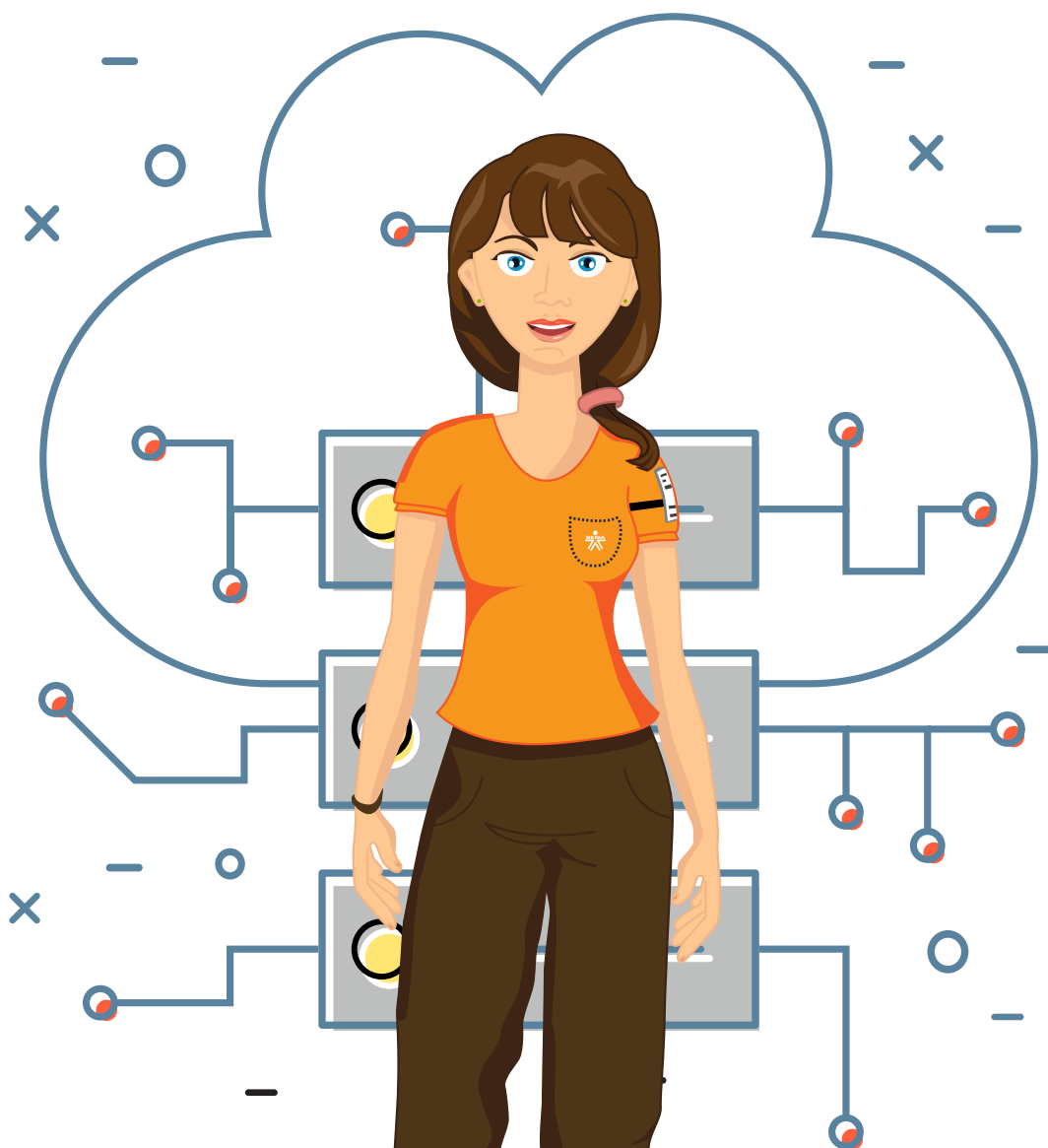
ESTRUCTURA DE CONTENIDOS

	Pág
Introducción.....	3
Mapa de contenido.....	4
1. Java y las bases de datos.....	5
2. JDBC (java database connectivity).....	6
3. JPA (java persistence api).....	6
4. Patrón de diseño MVC.....	7
5. Reportes en Java con jasperreports.....	7
6. Desarrollo de una aplicación en Java paso a paso.....	7
Glosario.....	143
Bibliografía.....	144
Control del documento.....	145

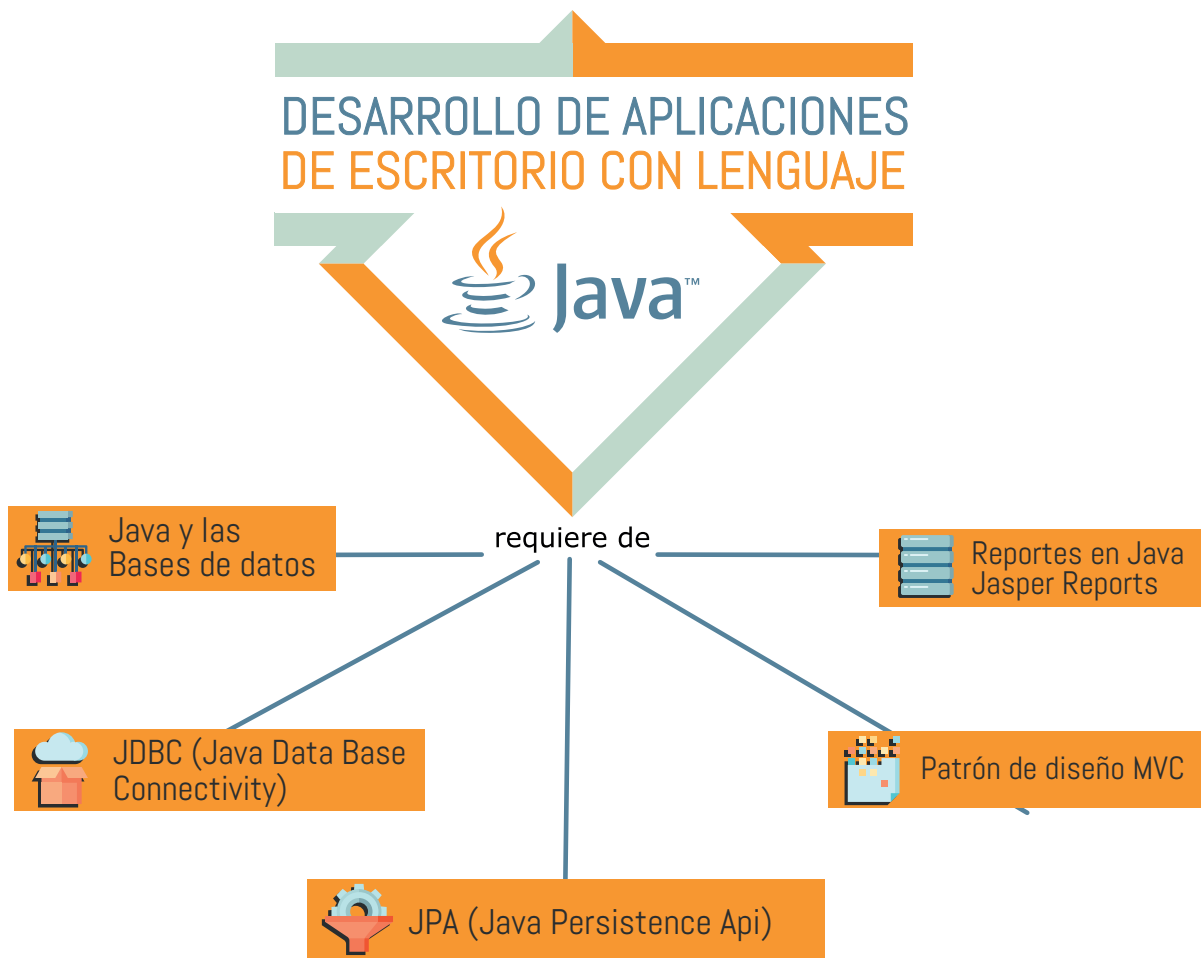
INTRODUCCIÓN A LA BASE DE DATOS ORACLE

INTRODUCCIÓN

El desarrollo de una aplicación en Java además de requerir los conocimientos básicos del lenguaje, el manejo de estructuras de almacenamiento y la aplicación de la programación orientada a objetos, requiere también de otros elementos necesarios para la funcionalidad y calidad profesional esperada, como el acceso a bases de datos, la aplicación de patrones como el MVC (Modelo Vista Controlador), la generación de reportes, entre otros. En este objeto de aprendizaje relacionado con el lenguaje de programación Java, encontrará un referente teórico que le permitirá el desarrollo completo de un caso de uso, desde la creación de la base de datos hasta la generación de reportes impresos utilizando Java.



MAPA DE CONTENIDO



GENERALIDADES

1. Java y las bases de datos

Los sistemas de información requieren un repositorio de almacenamiento de datos para la efectiva gestión de la información, a este repositorio se le denomina **Back-End** y generalmente se trata de una base de datos. Estas bases de datos no son accedidas directamente por el usuario final, en lugar de ello, los sistemas de información ofrecen otros mecanismos como formularios desarrollados con un lenguaje de programación, en los cuales se presentan los datos de la base de datos y/o se permite la actualización de los mismos, esos formularios que tienen comunicación directa con el usuario final se denominan **Front-End**.

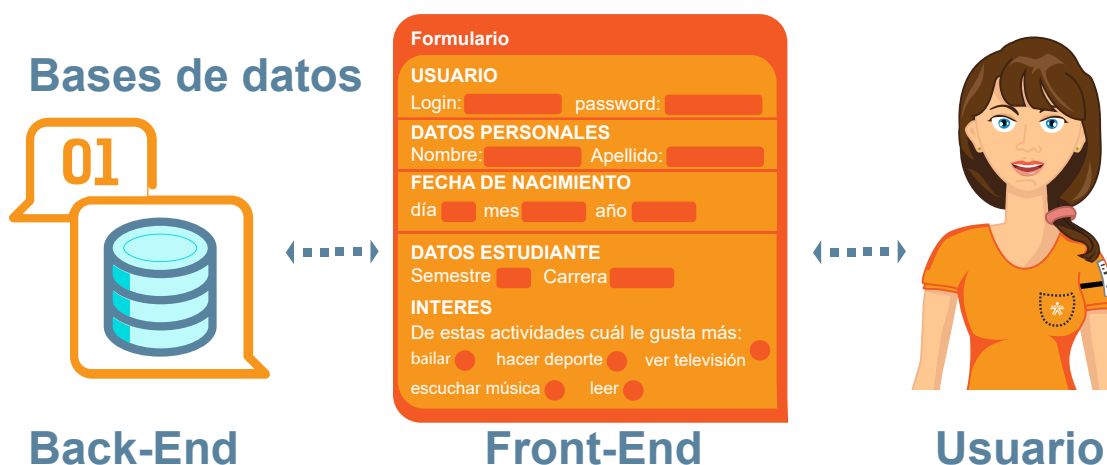


Figura 1. Estructura del programa a desarrollar.
Fuente: SENA

Mientras que el **Front-End** de una aplicación se puede desarrollar directamente en el lenguaje de programación seleccionado como Java a través de componentes tipo swing, el **Back-End** requiere el uso de una nueva herramienta conocida como el SGBD (Sistema Gestor de Bases de Datos).

En la actualidad existen diferentes SGBD, entre ellos, Oracle, MySQL, SQLServer, PostgreSQL, DB2, entre otros. Para lograr la comunicación entre estos SGBD y los lenguajes de programación como Java, deben existir unas interfaces y protocolos de comunicación entre ellos, en el caso de Java, existen dos mecanismos de acceso a los datos de una base de datos, estos son:

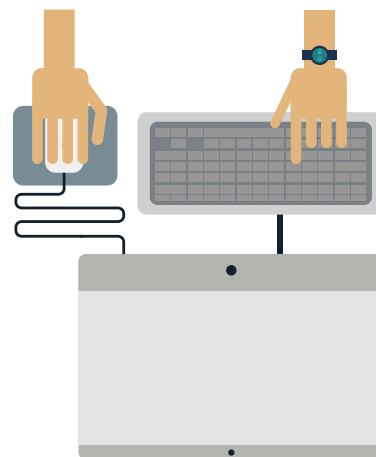
- JDBC - Java DataBase Connectivity
- JPA - (Java Persistence Api)

2. JDBC (Java DataBase Connectivity)

JDBC Java DataBase Connectivity es un estándar usado por Java para el acceso y la manipulación de datos en bases de datos relacionales a través de clases e interfaces, permitiendo embeber código SQL dentro del código Java.

Para el acceso a datos con JDBC se deben seguir los siguientes pasos:

1. Establecer y abrir la conexión a la base de datos.
2. Ejecutar las sentencias SQL.
3. Procesar los resultados.
4. Cerrar la conexión a la base de datos.



3. JPA (Java Persistence Api)

El mecanismo de **persistencia JPA** permite a los desarrolladores crear clases mapeando los atributos de las mismas con campos de una tabla en una base de datos relacional. Este mapeo, el cual se realiza a través de anotaciones dentro de la clase, permite abstraer el modelo de la base de datos a nivel de aplicación, logrando mayor transparencia en el acceso y la gestión de los datos, permitiendo que los objetos creados en una aplicación se puedan persistir en la base de datos. Para el acceso y gestión de los datos a través de JPA los pasos son los siguientes:



A Crear una unidad de persistencia: esta unidad es un archivo xml que contiene información asociada a la conexión con la base de datos.

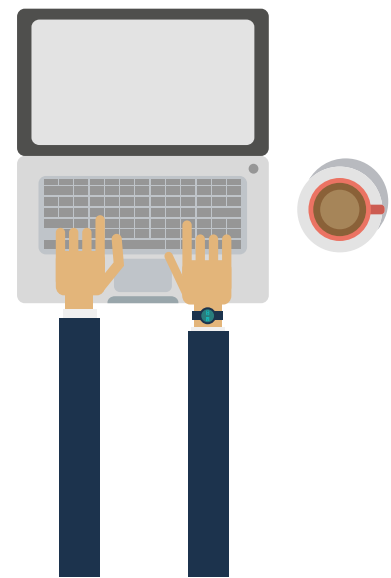
B Crear las clases entidad: una clase de entidad es una clase Java que está asociada a una tabla de la base de datos. En estas clases los atributos de la clase se mapean (vinculan) con los campos de la tabla asociada en la base de datos, de esta manera, la clase de entidad representa en memoria a la tabla de la base de datos.

C Crear las clases controladoras de las clases de entidad: las clases controladoras gestionan a los conjuntos de objetos de las clases de entidad, por ejemplo, si se tiene la clase de entidad Estudiante, la clase controladora tiene la funcionalidad para gestionar a un conjunto de estudiantes.

4. Patrón de diseño MVC

El **MVC (Modelo Vista Controlador)** es un patrón (buena práctica) que permite separar las capas de una aplicación, teniendo una capa de presentación (la vista), una capa de datos (el modelo) y una capa lógica (el controlador). Esta separación funcional de las tres capas permite la independencia entre ellas logrando que los cambios en una de ellas no afecten a las demás.

- **El Modelo:** el modelo está conformado por las clases que forman parte del dominio de la aplicación. Por ejemplo, en un sistema de información para una escuela el modelo podría estar conformado por las clases Estudiante, Profesor, Materia, Aula, Computador, entre otros.
- **La Vista:** la vista está conformada por los formularios con los que interactúa el usuario final, en la vista debe existir la menor cantidad de código posible, incluso la gestión de los eventos no debe ser responsabilidad de la vista.
- **El Controlador:** las clases controladoras son aquellas que reciben instancias de la vista y el modelo para interactuar con ellos y mantener la armonía entre las capas. Por ejemplo, para almacenar la información de un estudiante en la base de datos, la clase controladora debe recoger la información ingresada en el formulario (la vista), crear un objeto de tipo Estudiante (el modelo) y enviarlo para su almacenamiento en el modelo.



5. Reportes en Java con Jasper Reports

Un sistema de información completo requiere la generación de cierta información impresa como medio de soporte o análisis. Los reportes en Java se pueden crear mediante la incorporación de un plugin o complemento al entorno de desarrollo. **JasperReports** es un complemento de uso libre que facilita el diseño y la creación de reportes en Java y trabaja con formatos pdf, html, rtf, xls, csv y xml.

6. Desarrollo de una aplicación en Java paso a paso

A continuación, se presenta un ejemplo completo de un caso de uso desarrollado en lenguaje de programación JAVA, comprende las siguientes sesiones:

- Sesión 1. Construcción de la base de datos.
(Se parte de la base de datos creada de citas).
- Sesión 2. Desarrollo de la Interfaz Gráfica de Usuario (GUI).
- Sesión 3. Creación del Modelo de la Aplicación.
- Sesión 4. Enlace con el Controlador.
- Sesión 5. Pruebas del Modelo Vista Controlador MVC.
- Sesión 6. Acceso a la Base de Datos con JDBC.
- Sesión 7. Acceso a la Base de Datos con JPA.
- Sesión 8. Generación de reportes impresos.

Desarrollo de una aplicación paso a paso

El presente ejemplo es una guía paso a paso sobre el desarrollo completo de un caso de uso en el lenguaje de programación JAVA, comprende desde el desarrollo de la base de datos hasta la generación de reportes impresos.

El caso de uso que servirá de fuente para el desarrollo de la aplicación es “**Administrar Datos Pacientes**” que hace parte del caso de uso del Sistema Gestión Citas que se ha venido realizando.

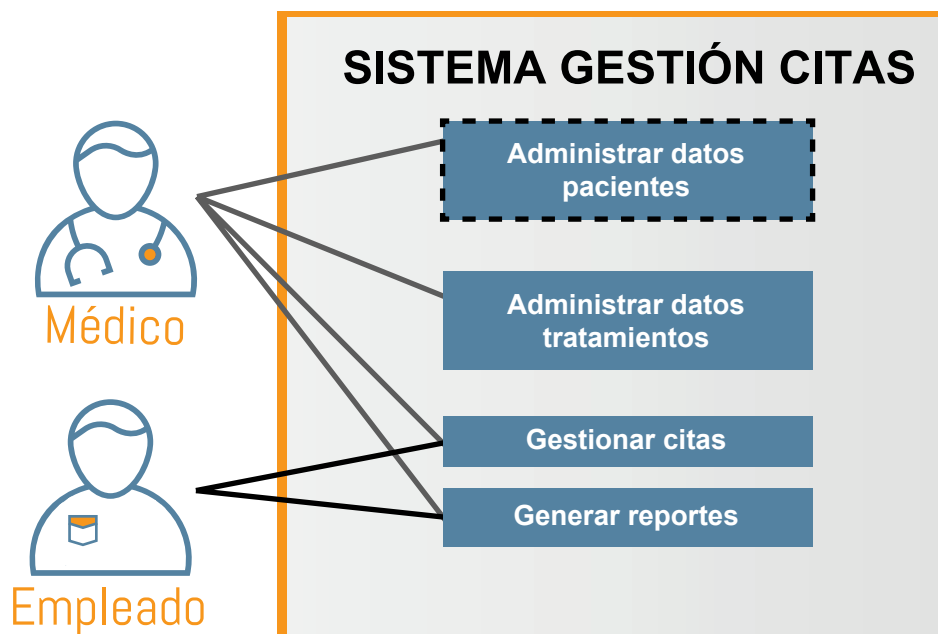


Figura 2. Sistema de gestión de citas caso de uso a desarrollar.

Fuente: SENA

El ejercicio realizado de la aplicación desarrollada puede ser revisado en sus archivos fuente:

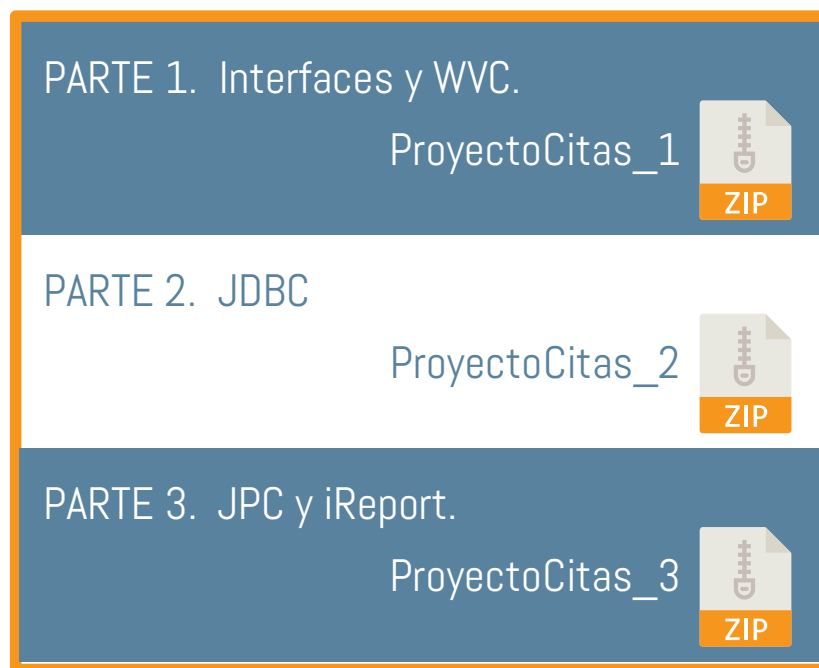


Figura 3. Guía de archivos fuente.
Fuente: SENA

Desarrollo de aplicaciones DE ESCRITORIO CON LENGUAJE



Sesión 1

Construcción
de la base
de datos.



En esta sesión, se debe tener en cuenta que en el material de estudio “Introducción a la base de datos en Oracle”, se creó la base de datos con la siguiente estructura:

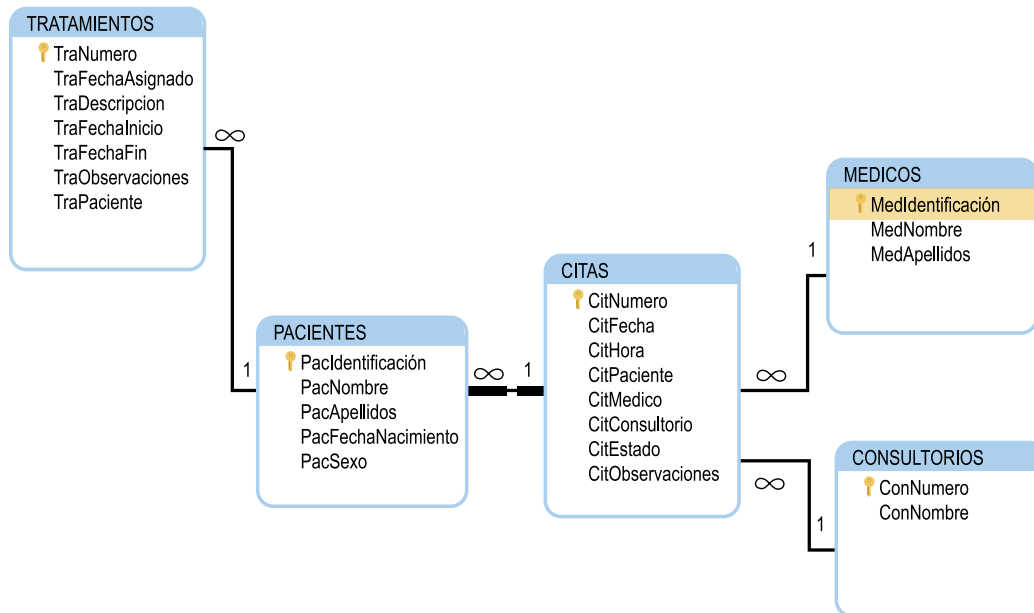


Figura 4. Estructura de la base de datos citas.
Fuente:SENA

De igual forma, en ese mismo material de estudio “Introducción a la base de datos en Oracle” se encuentra el usuario el esquema creado y la base de datos creada al igual que el usuario, se verifica que el servidor de base de datos esté funcionando adecuadamente y que esté la base de datos creada.



Se recuerda para tener en cuenta al momento de la conexión, el usuario y la clave.

Usuario: CITAS
Clave: citas

Figura 5. Revisión de la base de datos creada.
Fuente:SENA

Desarrollo de aplicaciones DE ESCRITORIO CON LENGUAJE



Sesión 2

Desarrollo de la
Interfaz Gráfica
de Usuario.



En la sesión anterior se creó la base de datos que servirá de repositorio para almacenar la información del sistema de gestión de citas. Sin embargo, un usuario de la aplicación como por ejemplo un médico, no debe tener acceso directo a la base de datos ya que le sería muy difícil entender cómo llevar a cabo sus funciones, además podría ser potencialmente peligroso para el sistema, debido a que la información o la base de datos estarían expuestas a probables daños.

Para evitar que esto ocurra y darle una mejor presentación a la aplicación, facilitando el uso de la misma, se requiere desarrollar una interface gráfica de usuario conocida como GUI por sus siglas en inglés **Graphical User Interface**, la GUI para el proyecto que se viene desarrollando será realizada en el ambiente de ejecución integrado **NETBEANS 8.2**.

NOTA. Se recuerda que el caso de uso que se desarrolla en el presente guía es **Administrar Datos Paciente**.

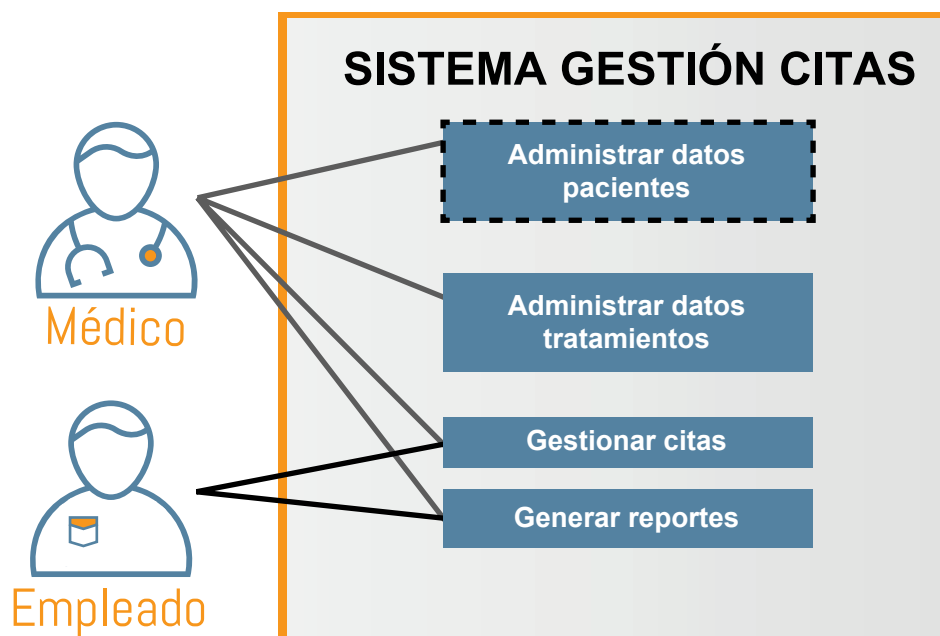


Figura 6. Estructura del programa a desarrollar.
Fuente:SENA

Crear nuevo proyecto citas.

Después de ejecutar netbeans, se debe crear un nuevo proyecto, para lo cual se sigue el menú: File → New Project

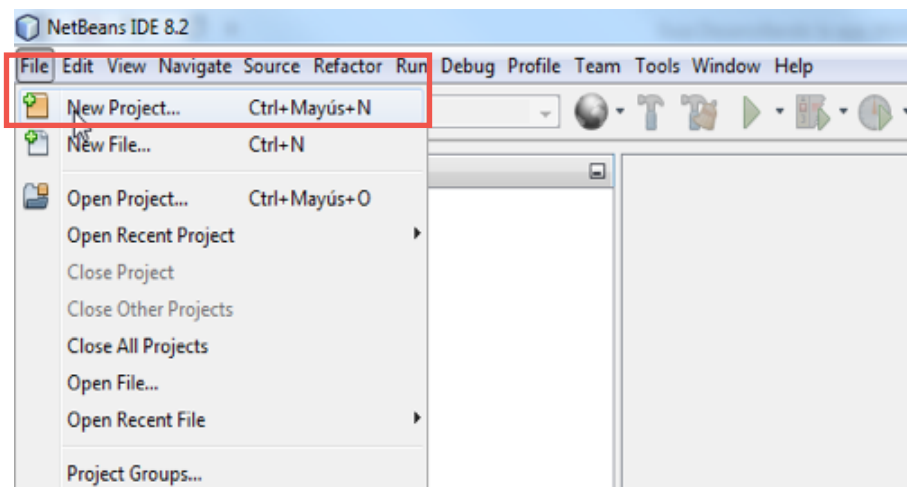


Figura 7. Creación del nuevo proyecto.
Fuente:SENA

Se selecciona la categoría de Java y el tipo de proyecto **Java Application** y se hace clic en **Next**.

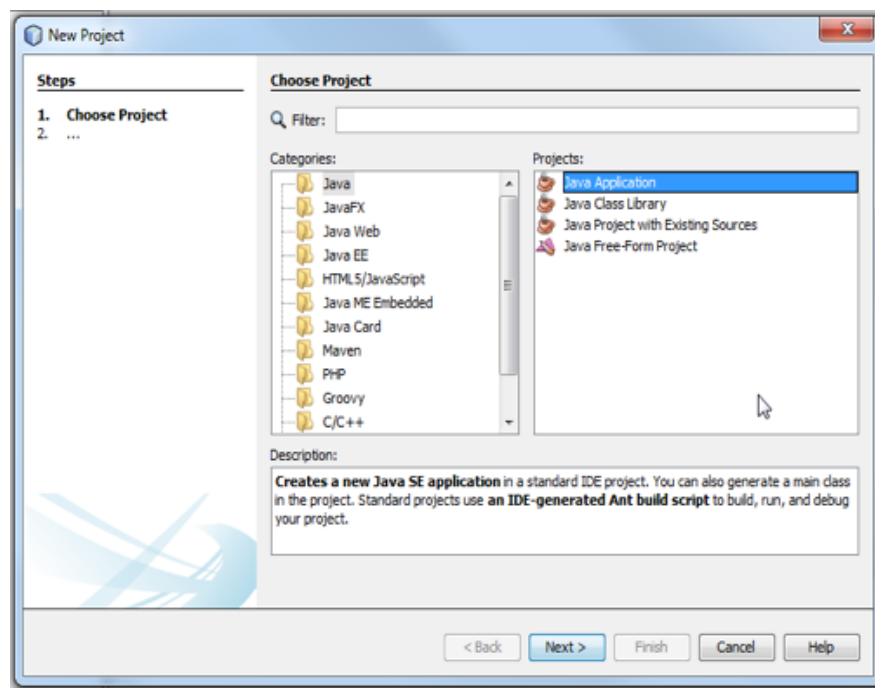


Figura 8. Estructura del programa a desarrollar Java Application.
Fuente:SENA

Se ingresa el nombre del proyecto, en este caso **ProyectoCitas**, se selecciona la ubicación donde se va a guardar y se establece como proyecto inicial y se hace clic en **Finish**.

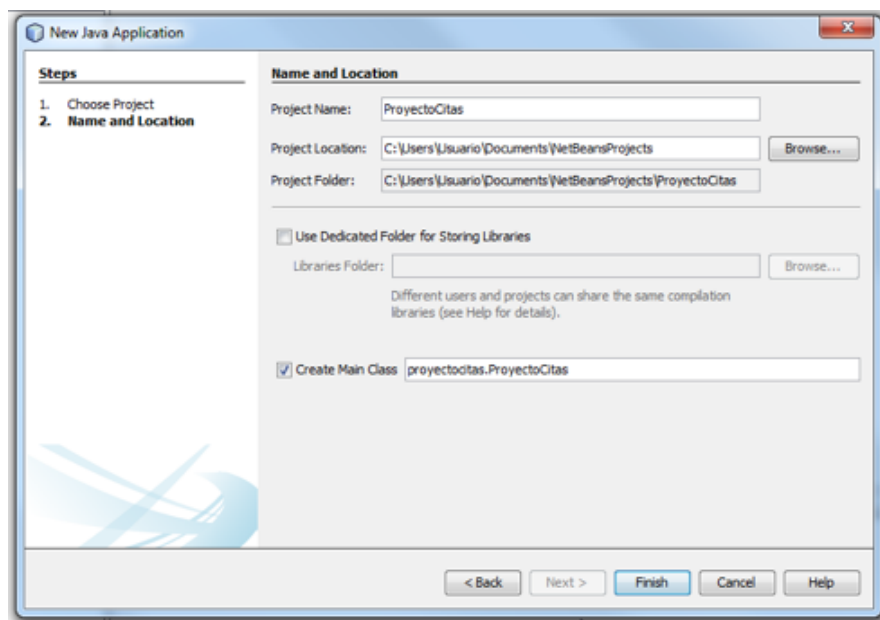


Figura 9. Nombre del Proyecto Citas.
Fuente:SENA

En la ventana de proyectos debe aparecer el nuevo proyecto, si la ventana no es visible, puede ser accedida a través del menú **window** → **projects**.

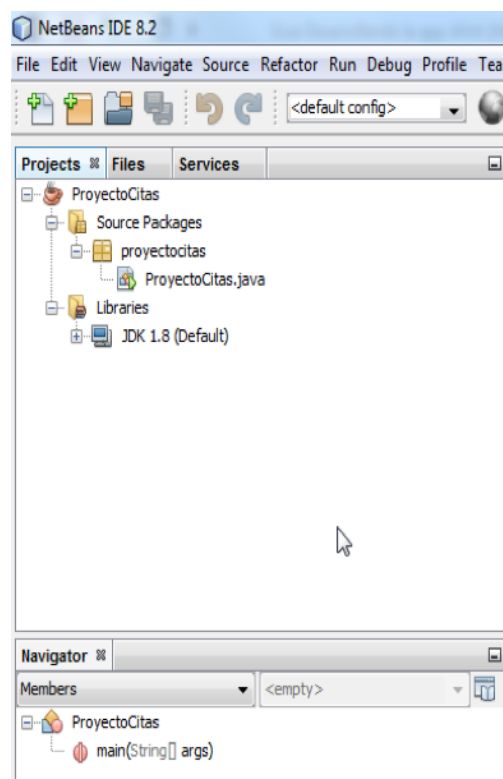


Figura 10. Estructura del programa ProyectoCitas.
Fuente:SENA

A continuación, se crea un nuevo paquete, haciendo clic derecho en **ProyectoCitas** y seleccionando del menú contextual **New** → **Java Package**.

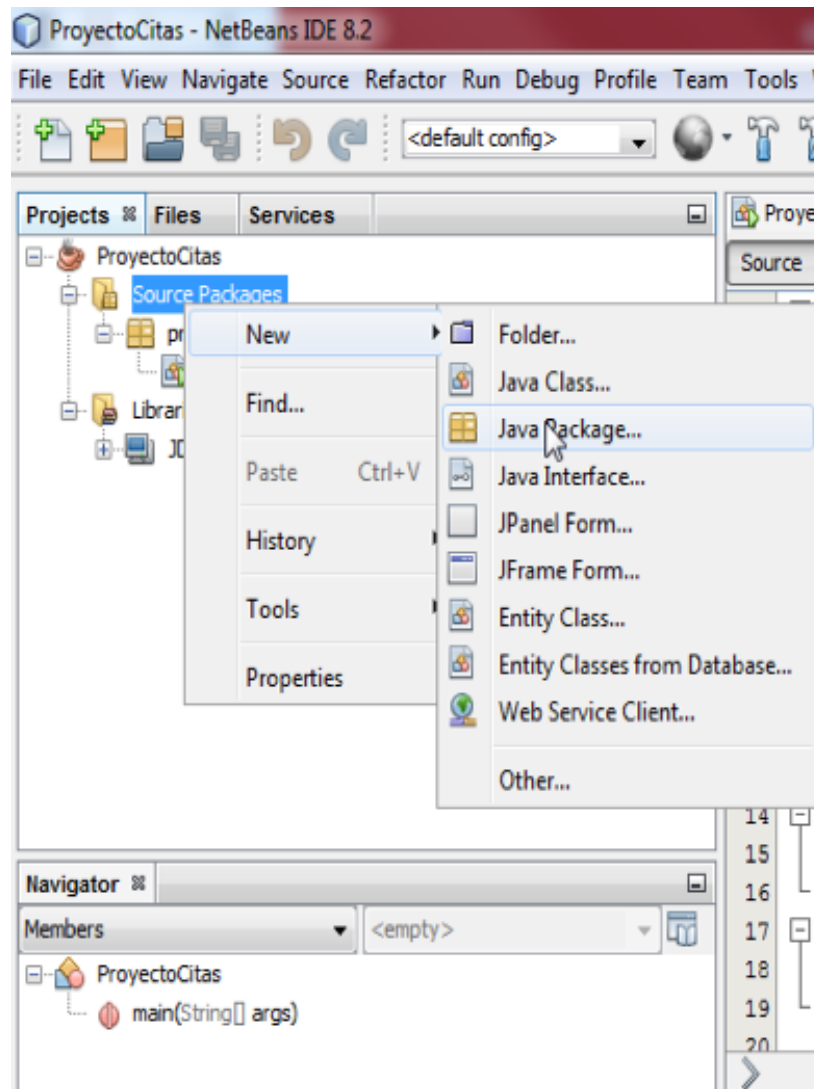


Figura 11. Agregando un paquete al proyecto.
Fuente: SENA

Se ingresa el nombre del paquete, en este caso vista y se hace clic en **Finish**.

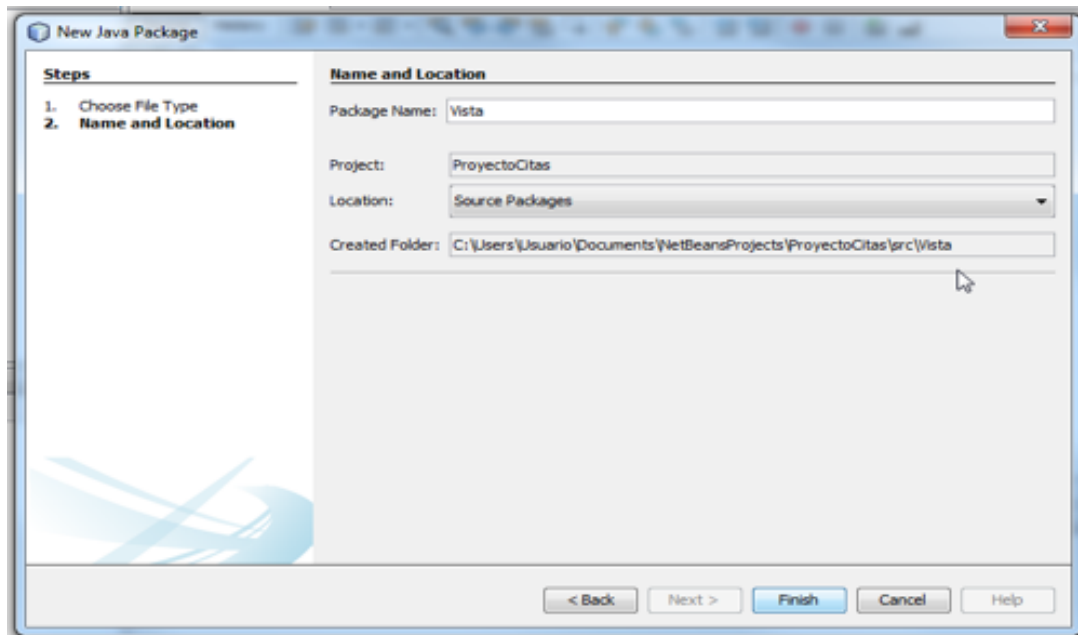


Figura 12. Configurando el nombre del paquete.

Fuente: SENA

El objetivo de crear este paquete es almacenar en él todos los formularios que formen parte de la interface gráfica de usuario como parte de un patrón conocido como el Modelo Vista Controlador, sobre el cual se profundizará más adelante.

- Diseño del formulario principal.

Hacer clic derecho sobre el nuevo paquete creado, luego seleccionar del menú contextual **New** → **Jframe Form**.

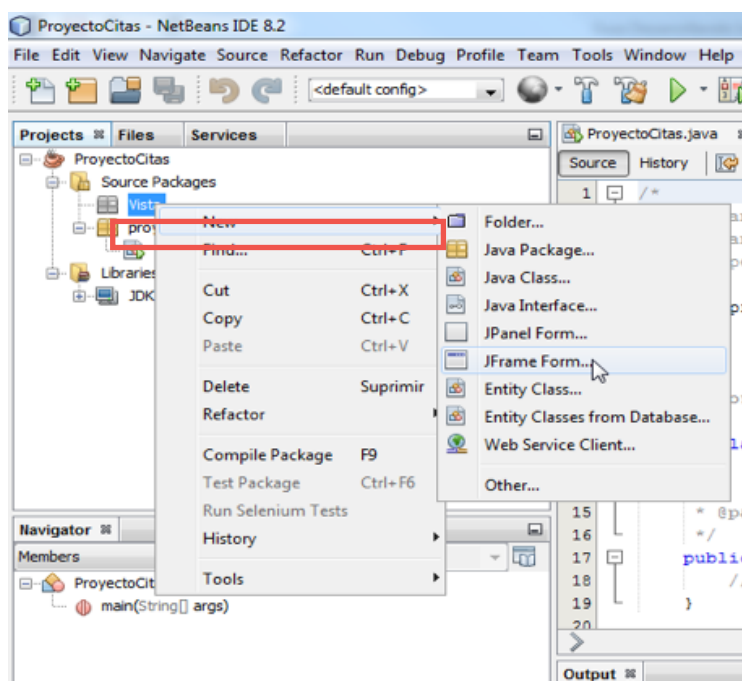


Figura 13. Agregando un JFrame Form.

Fuente: SENA

NOTA. La aplicación será desarrollada bajo un ambiente **MDI: Multiple Document Interfaz**, este tipo de aplicaciones permite tener un formulario principal y visualizar sobre otros formularios secundarios que pueden estar abiertos de manera simultánea pero que dependen del formulario principal. En este caso, el **Jframe form** será el formulario principal.

Se define el nombre del **Jframe**, en este caso **PrincipalJFrame** y se hace clic en **Finish**.

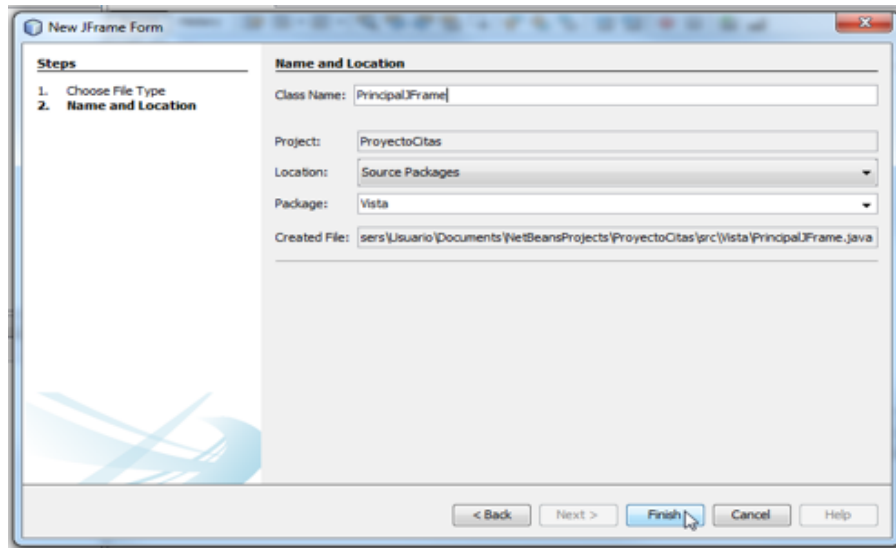


Figura 14. Configurando el nombre del JFrame.

Fuente: SENA

A continuación, aparecerá un formulario vacío y una paleta de controles, si dicha paleta no es visible puede obtenerla en el siguiente menú: **Window → Palette**.

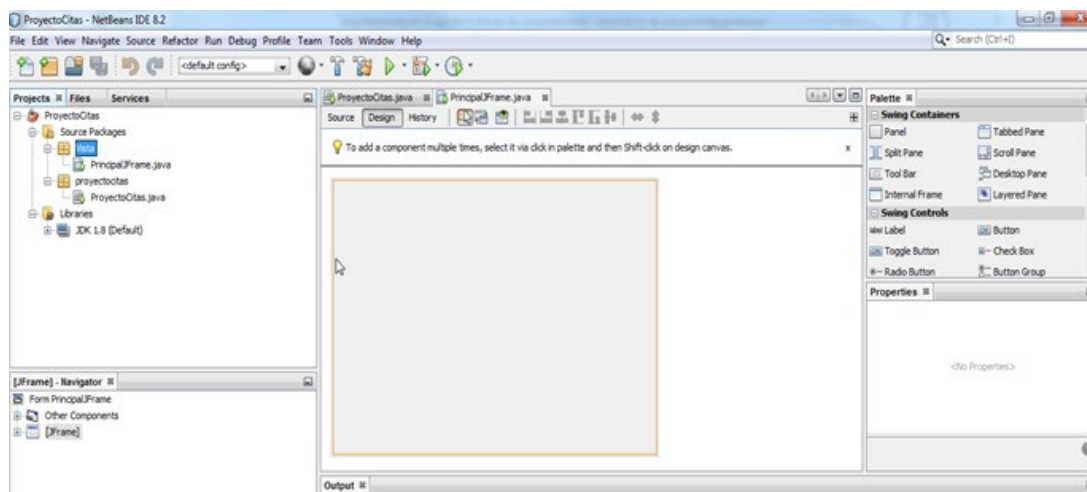


Figura 15. Formulario del JFrame.

Fuente: SENA

Al hacer clic en Siguiendo el menú **Window**→ **Properties** se tendrá acceso a la ventana.

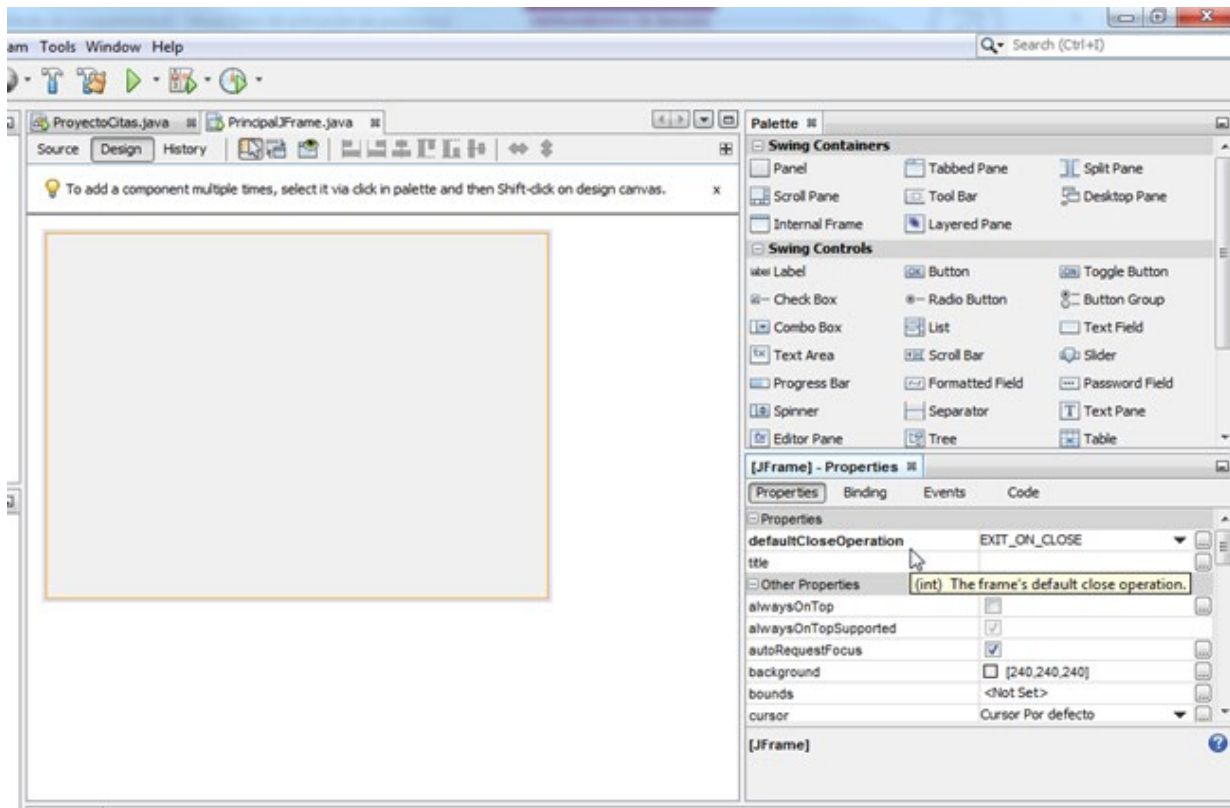


Figura 16. Formulario y paleta de controles.
Fuente: SENA

En la ventana de propiedades se debe realizar los siguientes cambios:

Tabla 1

Cambios en ventana propiedades.

Propiedad	Valor
Title	GESTIÓN DE CITAS
Name P	PrincipalJFrame

Nota: Producción SENA

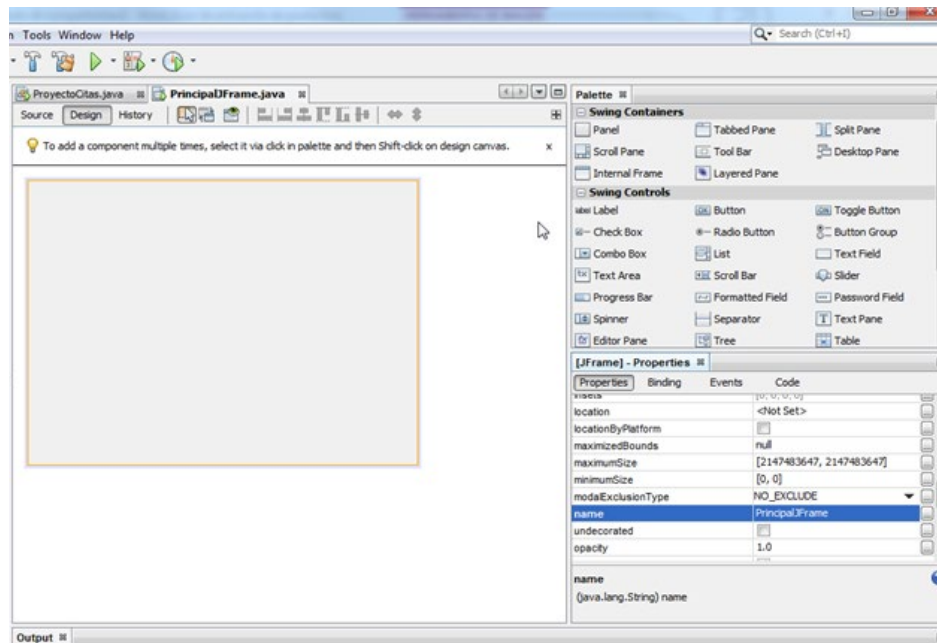


Figura 17. Configurando las propiedades del objeto.
Fuente: SENA

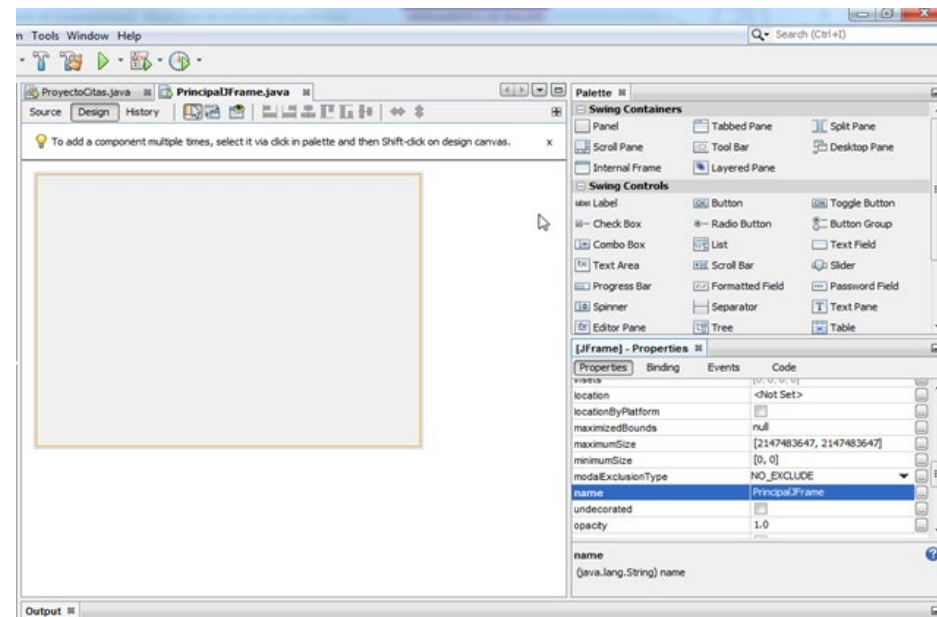


Figura 18. Configurando las propiedades del objeto.
Fuente: SENA

Se lleva desde la paleta de controles un control de tipo **menú Bar**, el cual se coloca sobre el **PrincipalJframe**.

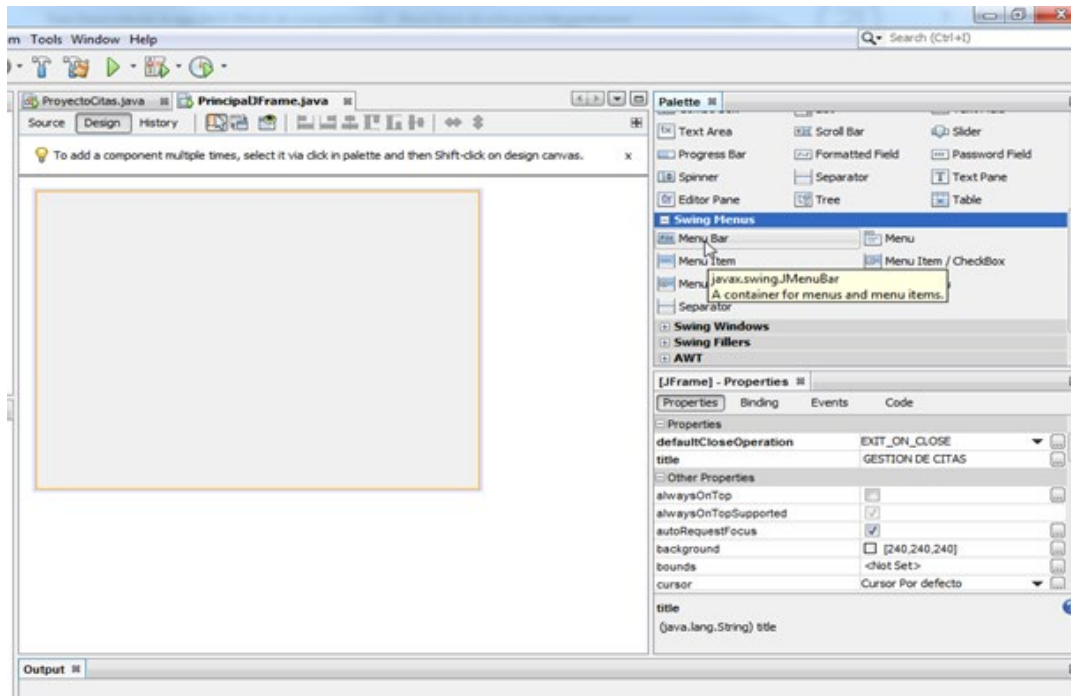


Figura 19. Configurando las propiedades del objeto.

Fuente: SENA

Se hace doble clic en el título **File** y se escribe **Archivo**, posteriormente se hace doble clic en el título **Edit** y se escribe **Pacientes**, de esta manera se va personalizando el menú de la aplicación.

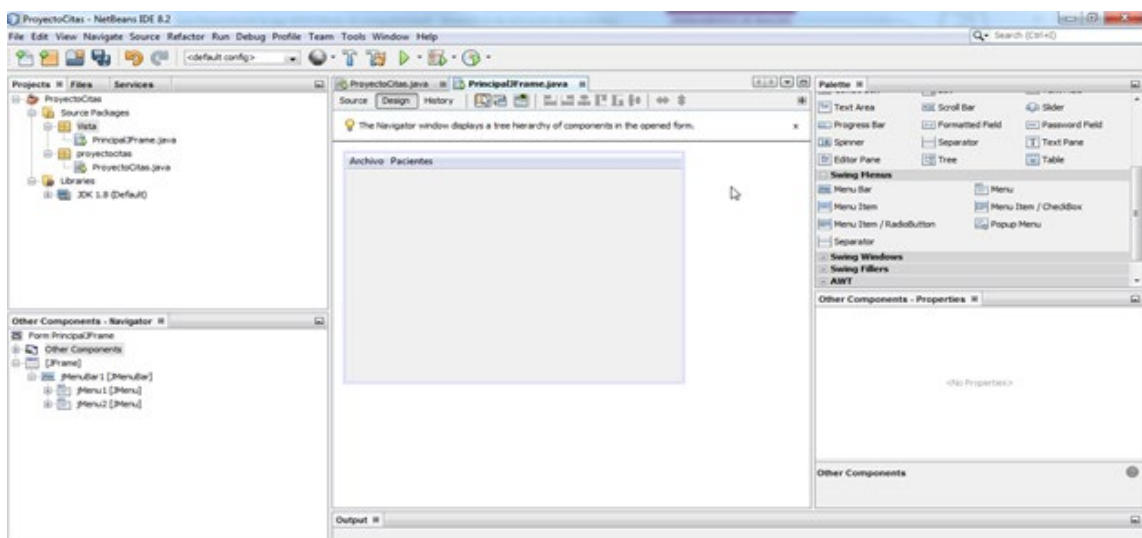


Figura 20. Configurando el menú del formulario.

Fuente: SENA

Se arrastra un control de tipo **Menú ítem** (jmenuItem 1= sobre el título **archivo** y dos **menú ítem** sobre el título **pacientes** (jmenutelm2 y jmenutelm3), cambiando sus títulos por los siguientes.

Tabla 2
Cambio de títulos.

Título actual	Nuevo título
jMenuItem1	Salir
jMenuItem2	Registrar
jMenuItem3	Consultar

Nota: Producción SENA

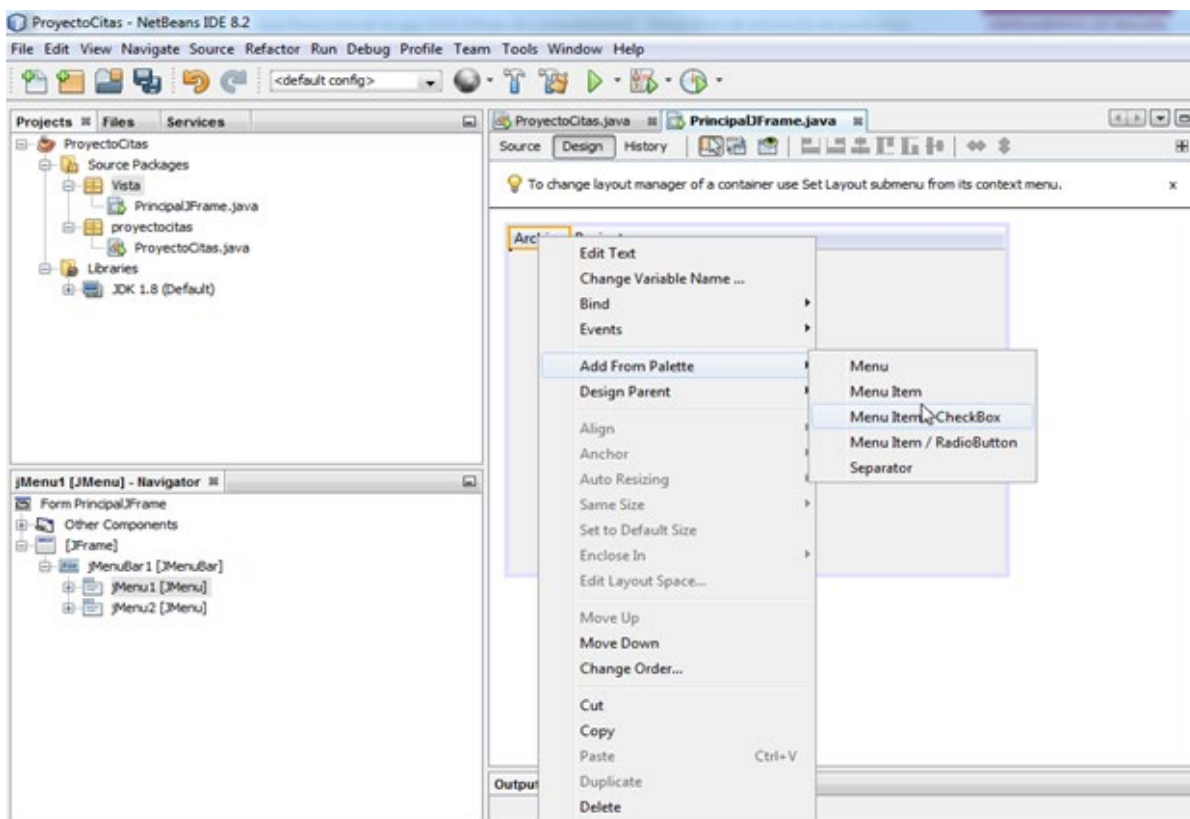


Figura 21. Agregando ítems al menú.
Fuente: SENA

De esta forma, al hacer clic sobre el menú **archivo**, aparecerá la opción **salir** y al hacer clic sobre el menú **pacientes**, aparecerán las opciones **registrar** y **consultar**.

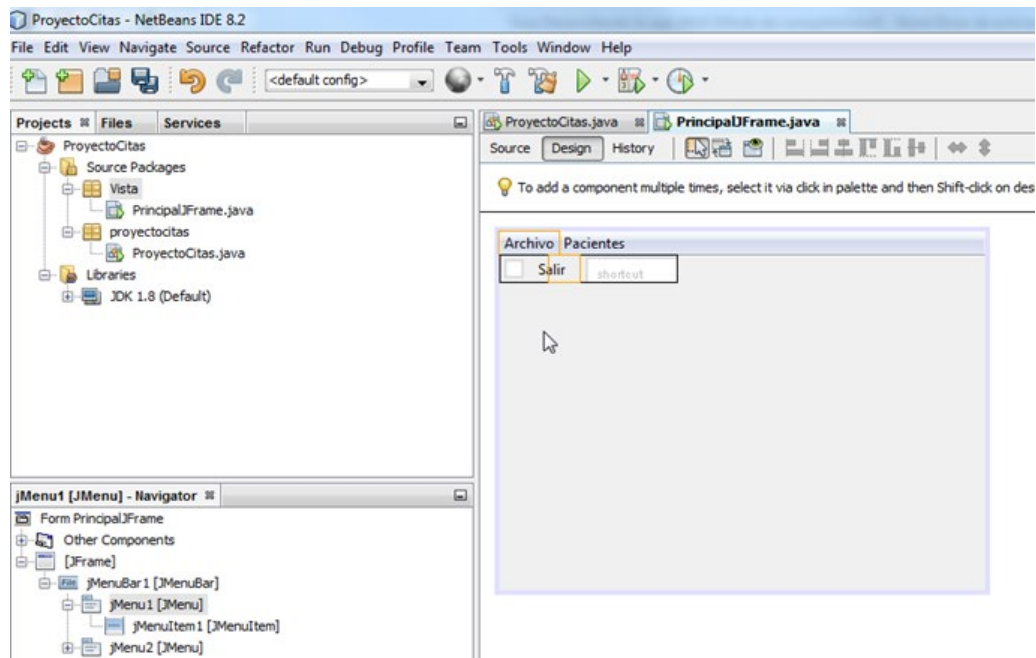


Figura 22. Configurando el menú del formulario.
Fuente: SENA

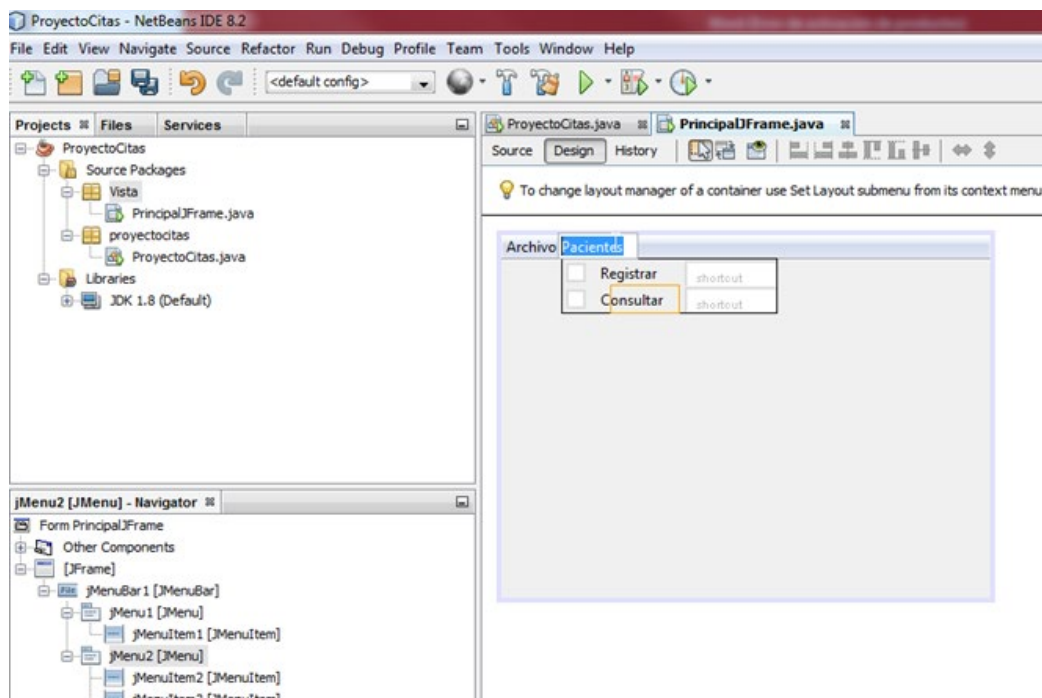


Figura 23. Agregando el menú paciente.
Fuente: SENA

De esta manera se termina de establecer los elementos gráficos del formulario principal de la aplicación, por ahora no se introduce ningún código hasta que se tenga los demás formularios necesarios. Para verificar el resultado parcial de la interface gráfica, se hace clic en el botón **Preview Design**.

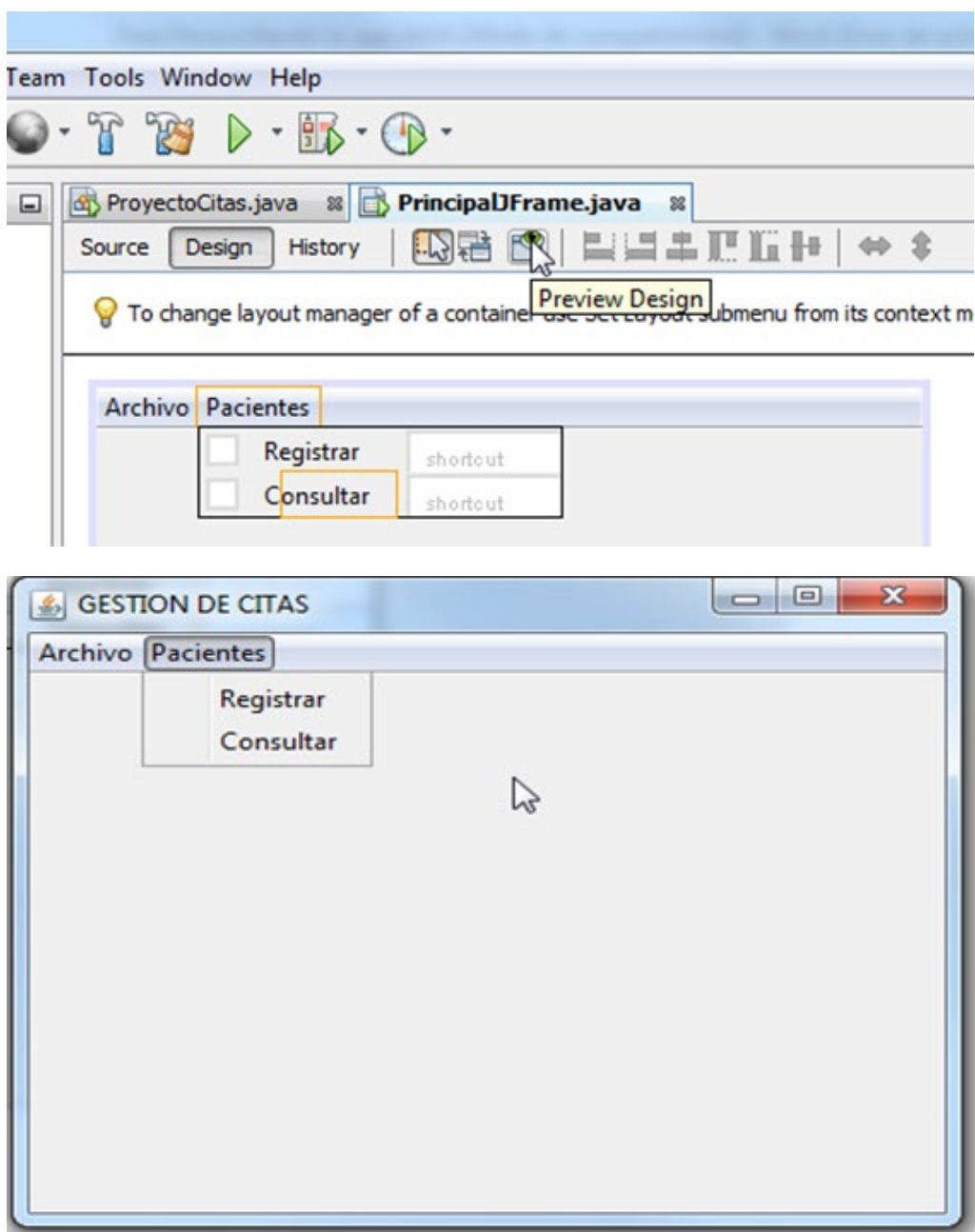


Figura 24. Preview del menú.
Fuente: SENA

• Diseño del formulario para el registro de pacientes

El formulario para el registro de pacientes y el formulario para consultar pacientes, deben ser formularios de tipo `Jinternalframe`, para lograr que dependan y se desplieguen dentro del **principaljframe**, que será el formulario principal de la aplicación.

En la ventana de proyectos, haga clic derecho sobre el paquete vista y en el menú contextual seleccione la opción **New** → **Jinternalframeform**.

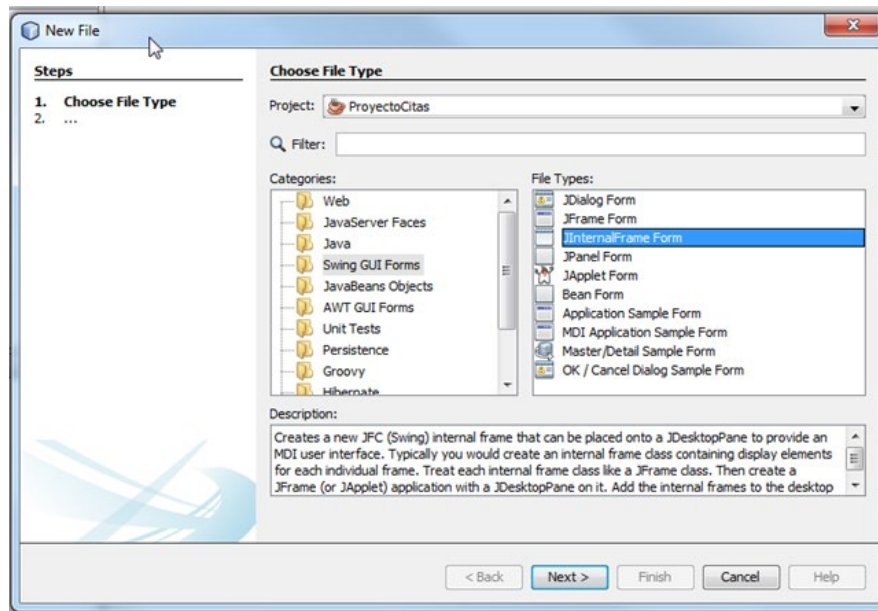


Figura 25. Agregando un `Jinternalframeform`.

Fuente: SENA

Se ingresa el nombre **RegpacienteInternalFrame** y se hace clic en **Finish**.

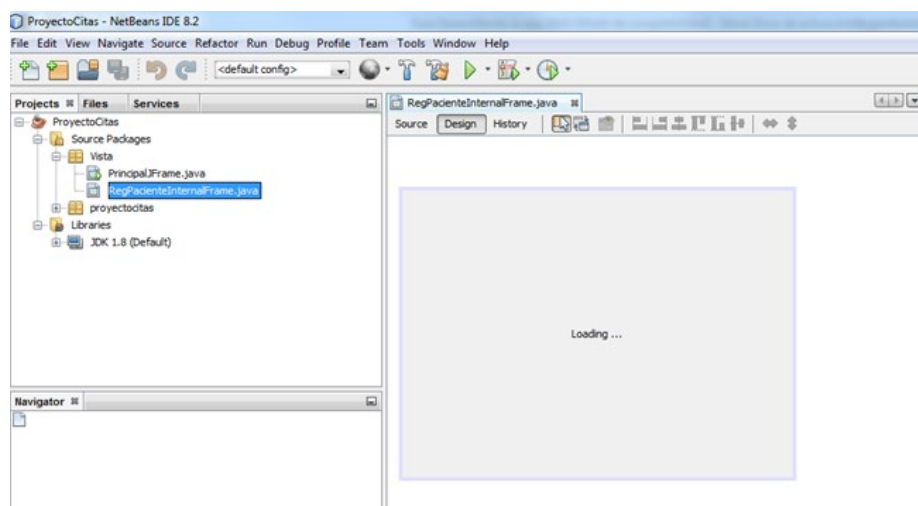


Figura 26. Agregando un `Jinternalframeform`.

Fuente: SENA

Al nuevo formulario creado se le cambia la propiedad **Title** por registro de pacientes a través de la ventana propiedades y se le adiciona los siguientes elementos.

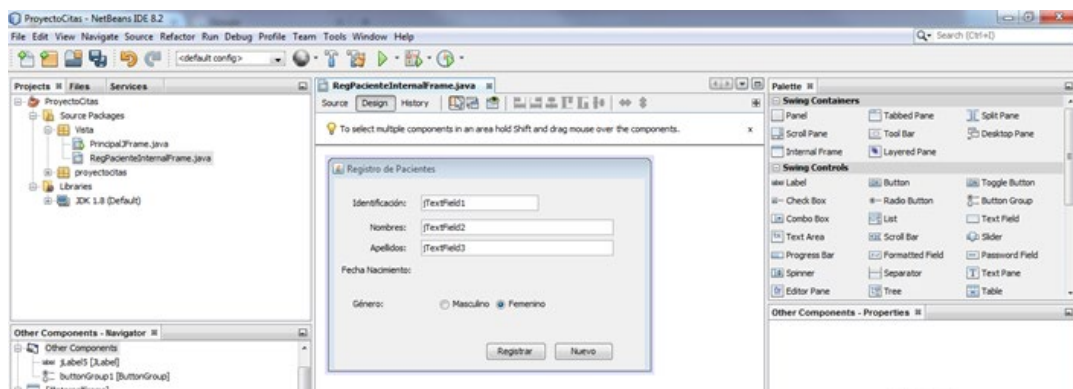


Figura 27. Agregando los objetos al formulario de registro de pacientes.

Fuente: SENA

Puede notarse que el título fecha de nacimiento no tiene un control asociado para capturar la información y aunque esta podría capturarse en un cuadro de texto, es más adecuado emplear un control específico de fecha, que en el caso de **Netbeans** no viene por defecto, razón por la cual se explicará cómo incorporarlo a este proyecto.

Tabla 3

Datos - Control asociado para capturar información.

Control	Propiedad	Valor
JLabel1	text	Identificación:
JLabel2	text	Nombres:
JLabel3	text	Apellidos:
JLabel4	text	Fecha de Nacimiento:
JLabel5	text	Sexo:
JTextField1	text	
	name	IdentificacionTxt
JTextField2	text	
	name	NombresTxt
JTextField3	text	
	name	ApellidosTxt
ButtonGroup1		
JRadioButton1	text	M
	buttonGroup	buttonGroup1
JRadioButton2	text	F
	buttonGroup	buttonGroup1
JButton1	text	Registrar
	name	RegistrarBtn
JButton2	text	Nuevo
	name	NuevoBtn

Nota: Producción SENA

Para descargar el archivo **jcalendar-1.4.zip** se refiere la siguiente dirección web: <https://toedter.com/jcalendar/>.

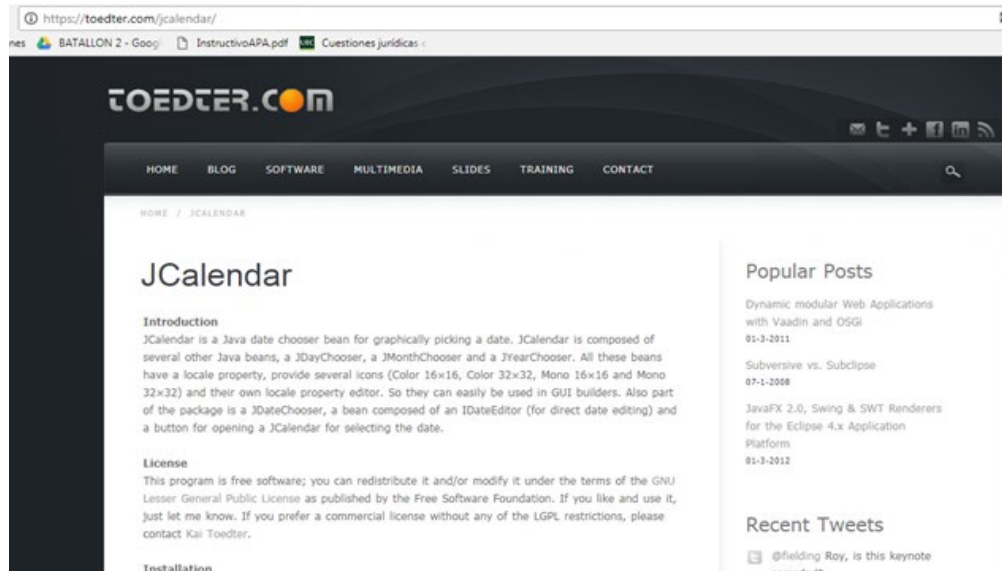


Figura 28. Página de descarga del componente JCalendar.
Fuente: SENA

Este archivo se descomprime en cualquier ruta y se observa que al interior de la subcarpeta **lib** se encuentre el archivo **jcalendar-1.4.jar**

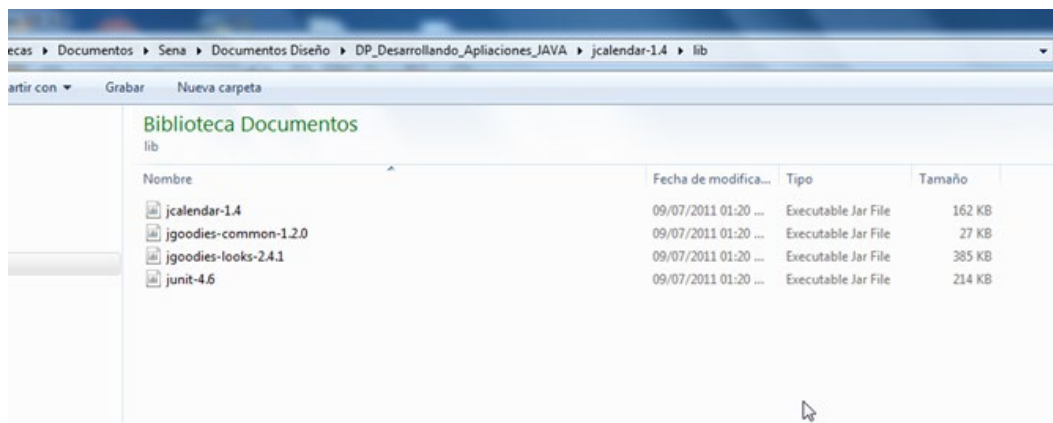


Figura 29. Descarga de los archivos .jar del calendar.
Fuente: SENA

Hacer clic derecho en cualquier espacio de la paleta de controles y seleccionar del menú contextual la opción **Palette Manager**.

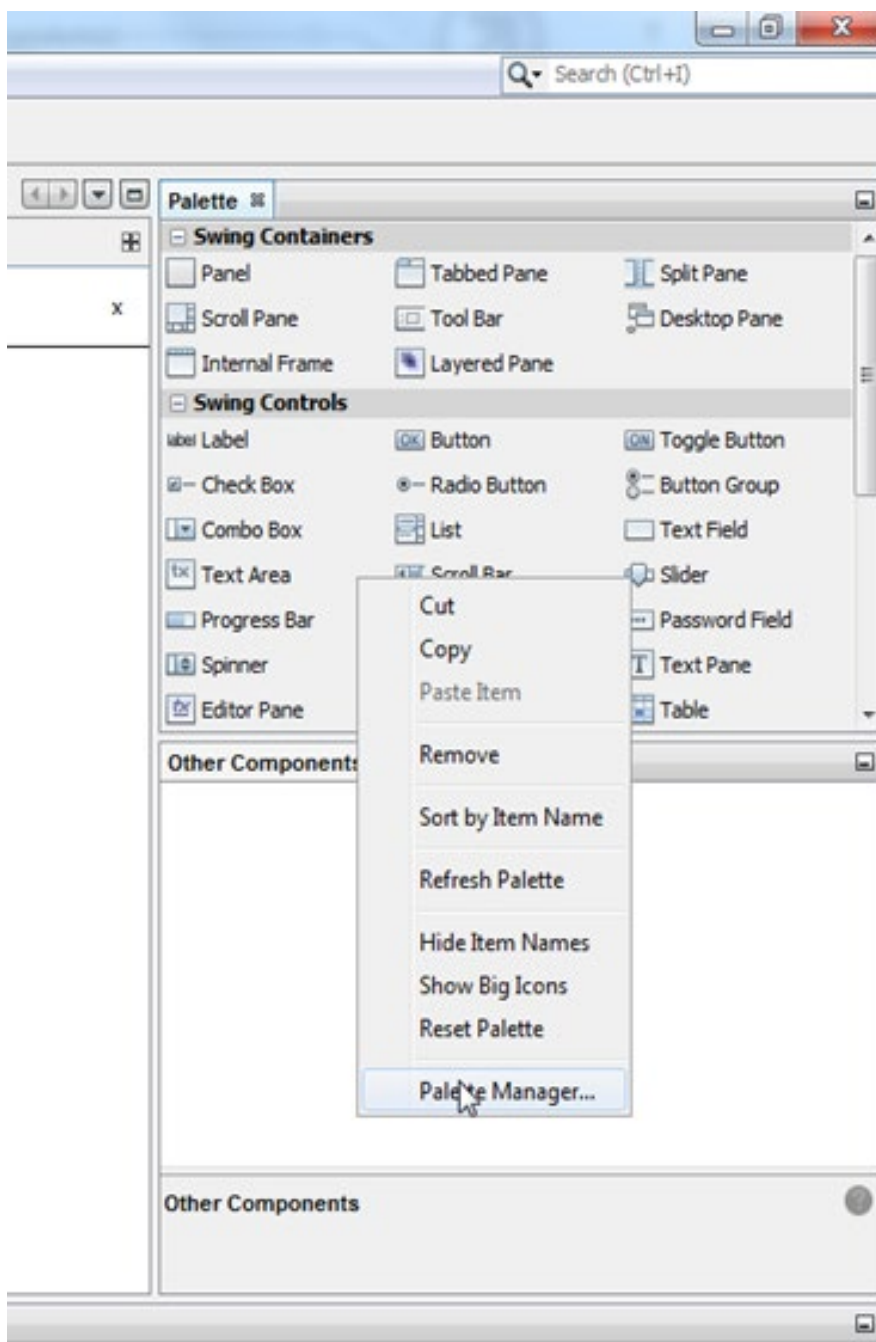


Figura 30. Opción de Palette Manager.
Fuente: SENA

Hacer clic en el botón **New Category**, ingresar un nombre para la categoría, en este caso **Swing Date** y hacer clic en el botón ok.

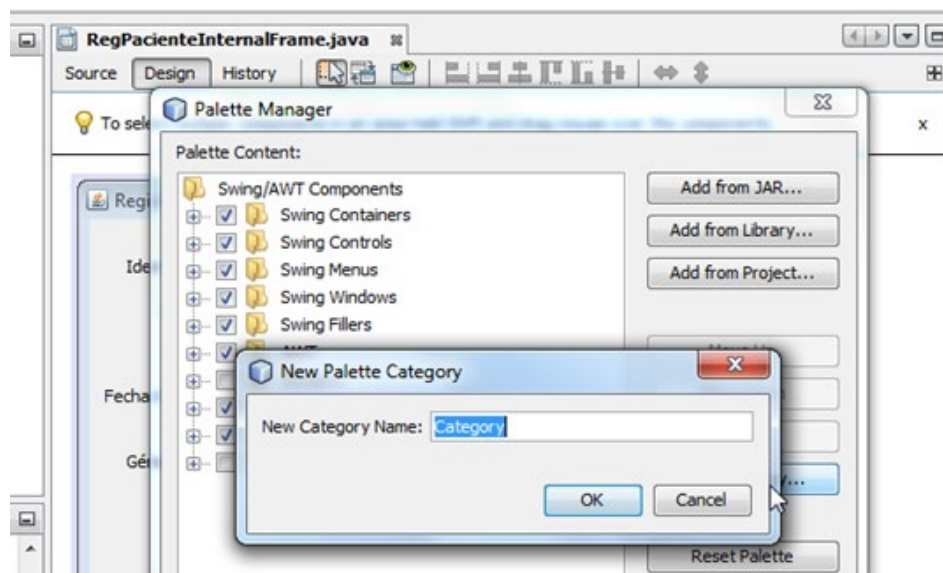


Figura 31. Creando la categoría swing date.
Fuente: SENA

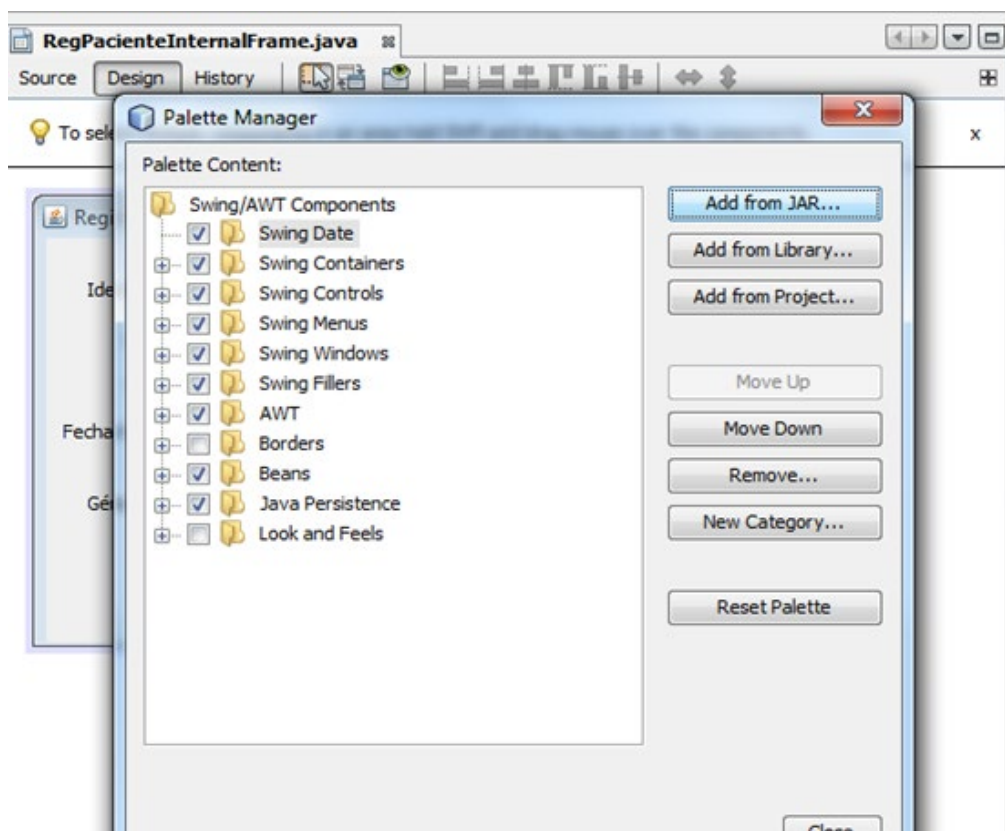


Figura 32. Adicionando a la categoría el objeto calendar.
Fuente: SENA

Hacer clic en el botón **Add From JAR** y buscar el archivo **jcalendar -1.4 jar**, hacer clic en el botón **Next**.

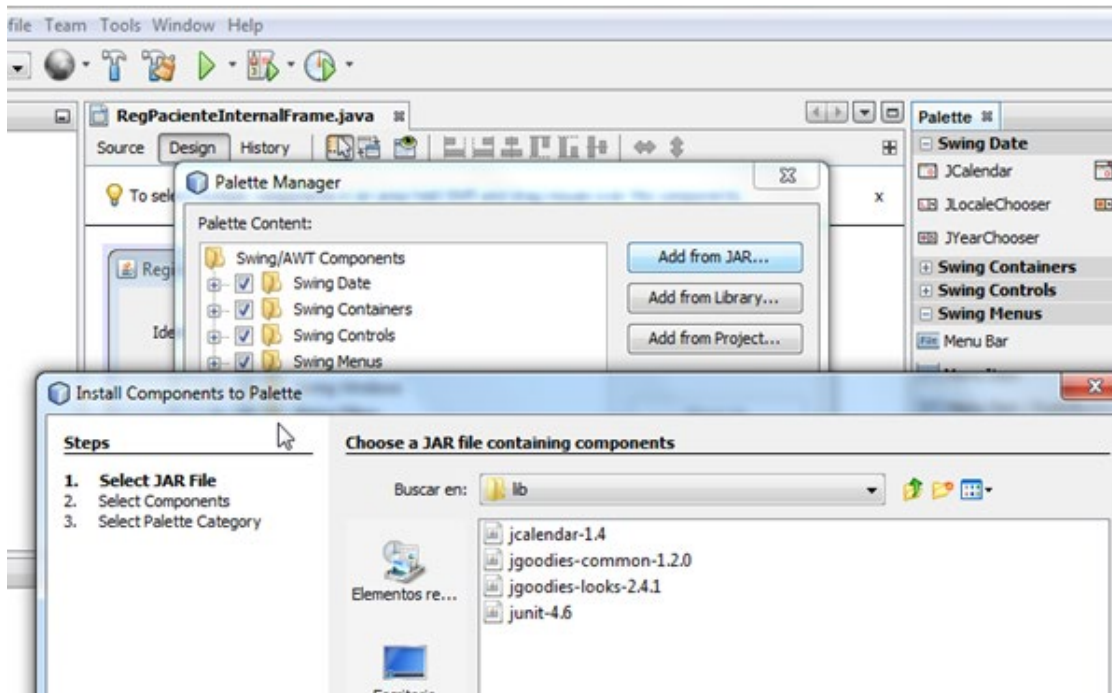


Figura 33. Adicionando el componente calendar a la paleta.
Fuente: SENA

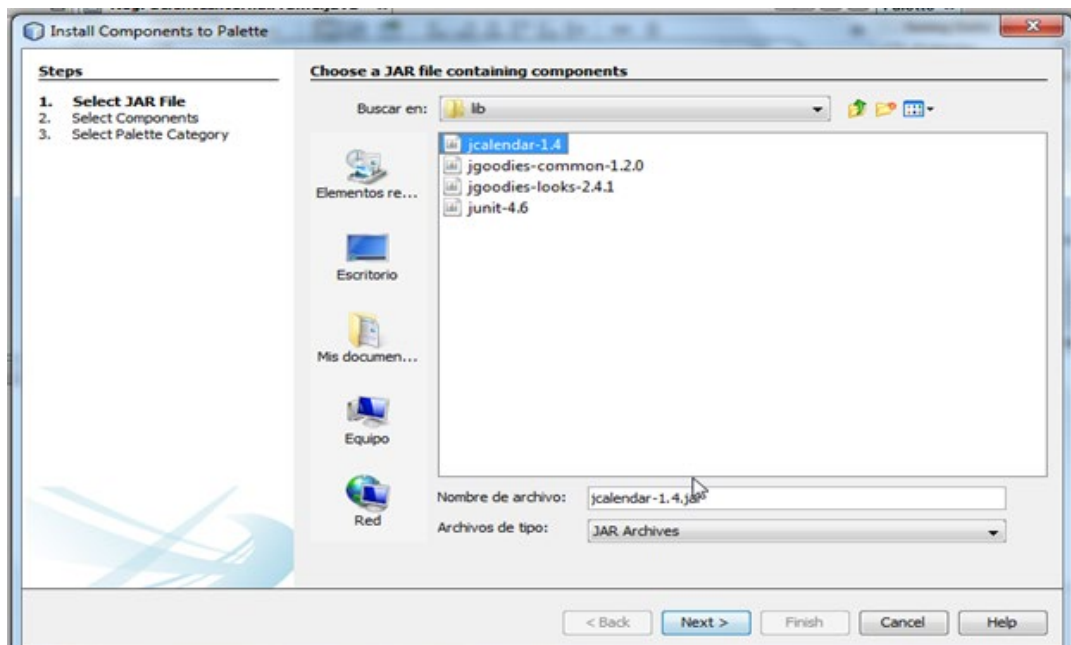


Figura 34. Seleccionando el componente Jcalendar.
Fuente: SENA

Se seleccionan los componentes que se quieren cargar en la paleta de controles, en este caso se seleccionan todo. Cada componente es un control que se agregará a la ficha swing date.

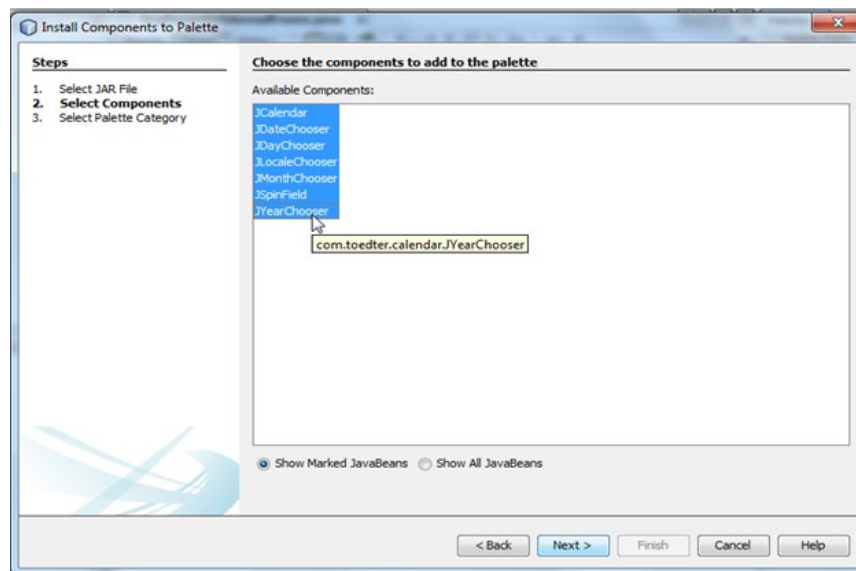


Figura 35. Seleccionando los componentes de Jcalendar.

Fuente: SENA

Ahora se selecciona la categoría donde se agregan los nuevos controles, en este caso la categoría es **swing date** y se hace clic en **Finish**.

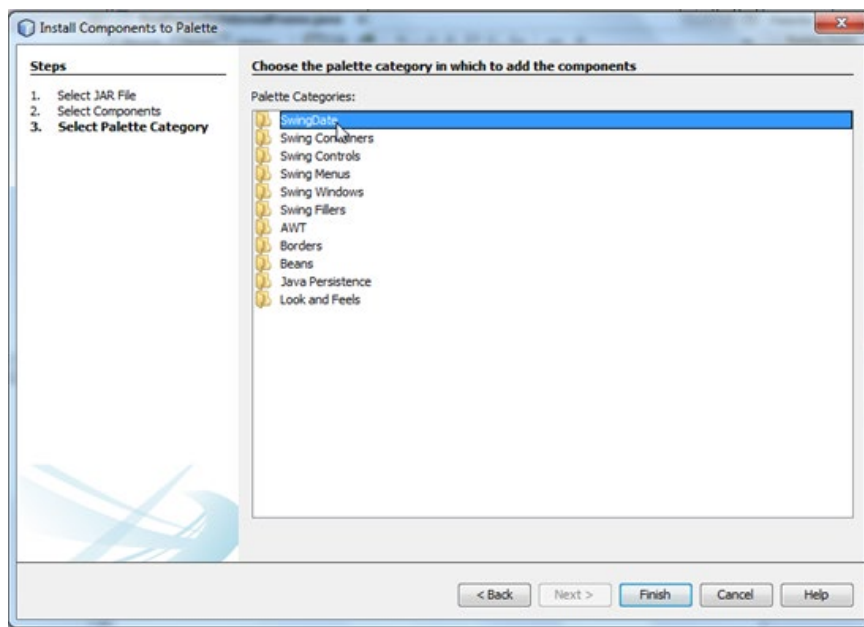


Figura 36. Seleccionando la categoría dentro de la paleta donde estará el objeto JCalendar.

Fuente: SENA

Finalmente hacer clic en **Close**.

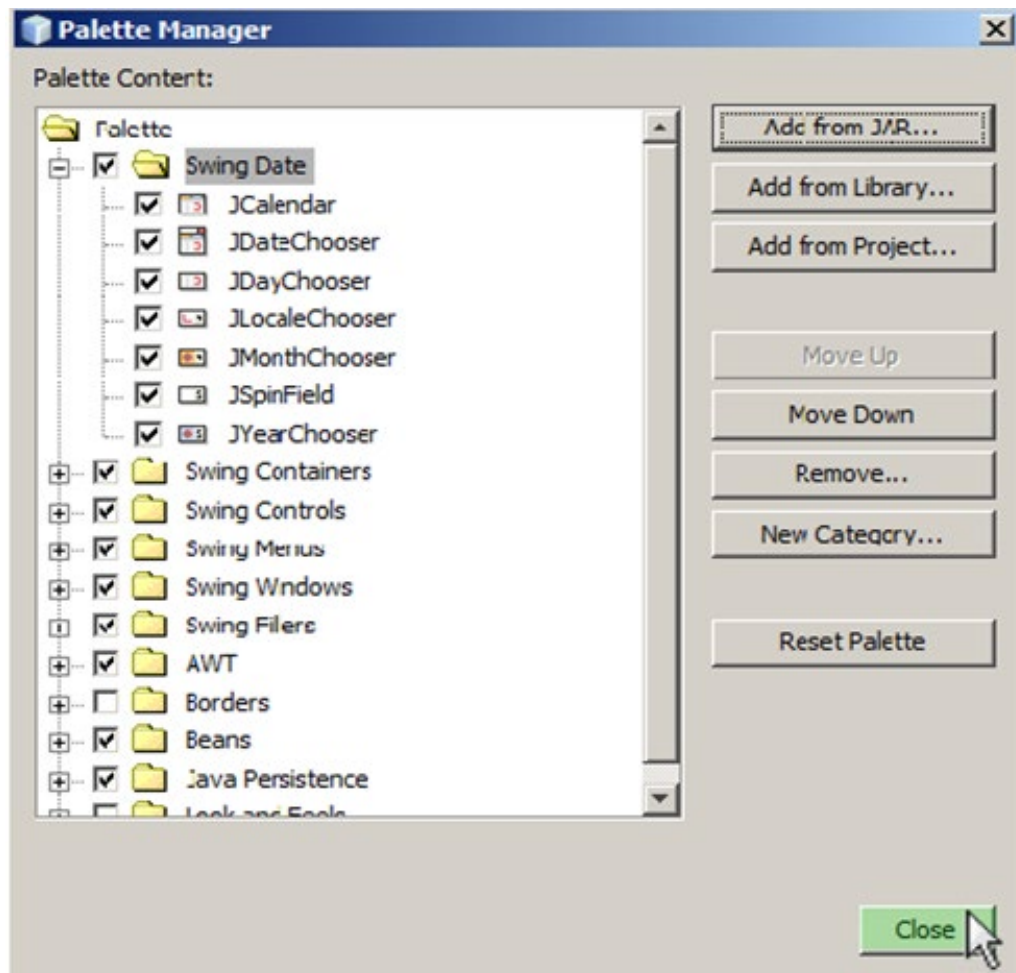


Figura 37. Cerrando el pallet manager.

Fuente: SENA

Se observa como la categoría swing date se ha agregado a la paleta de controles.

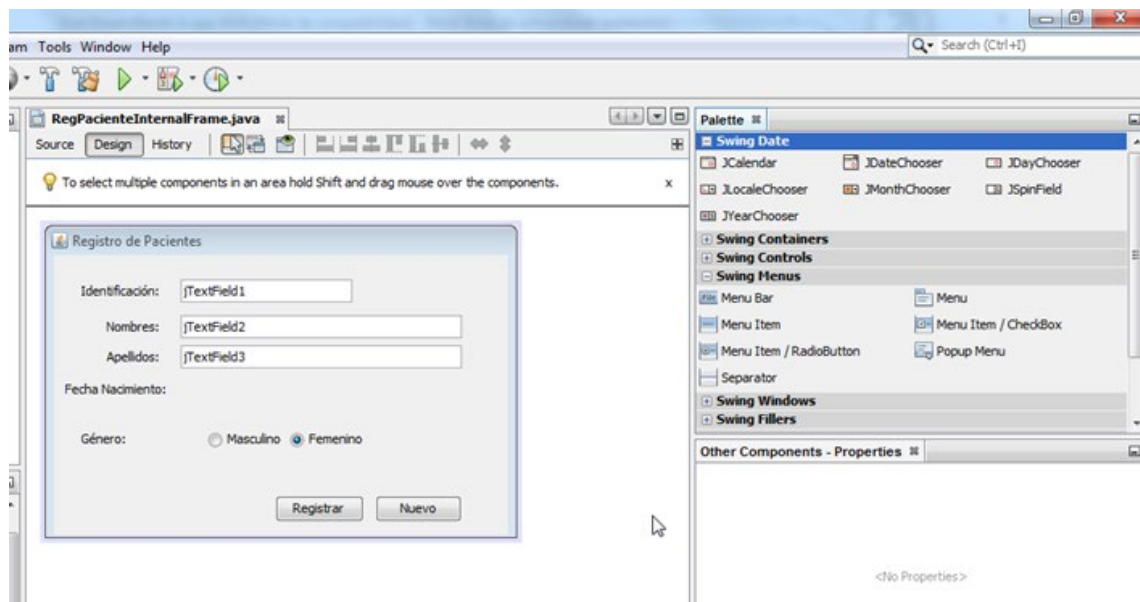


Figura 38. Visualizando la página de descarga del componente JCalendar.

Fuente: SENA

Ahora que se dispone de nuevos controles para el manejo de fechas, se arrastra uno de ellos, en este caso el control **JDateChooser** y se ubica en el formulario de registro clientes frente al título **fecha de nacimiento**.

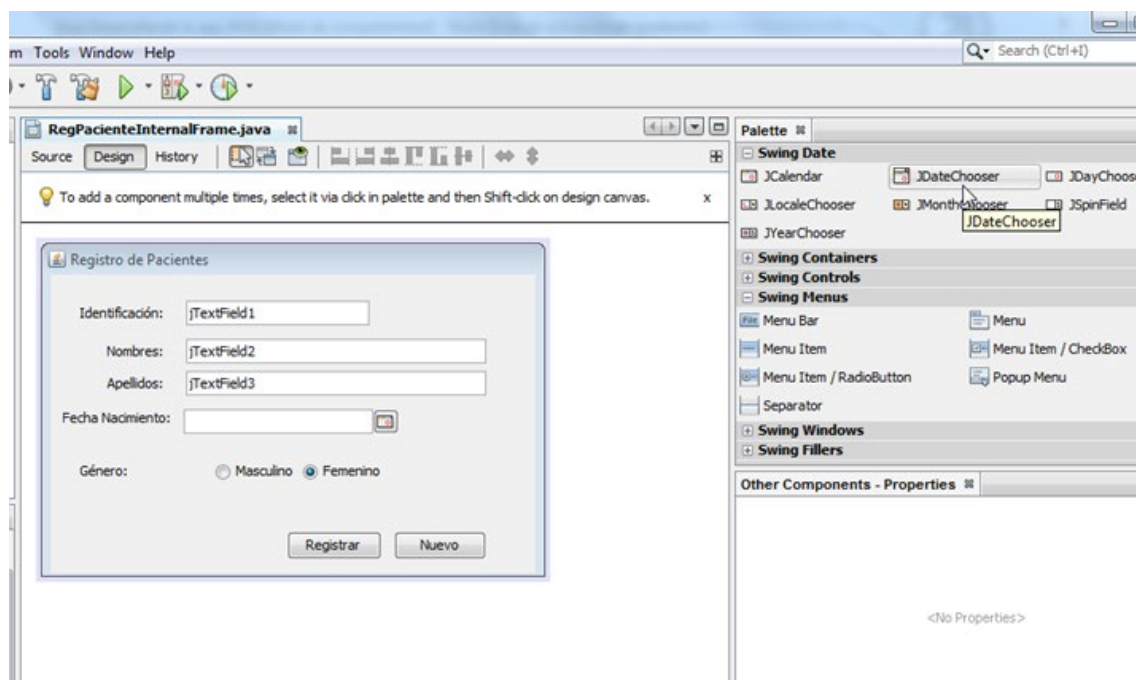
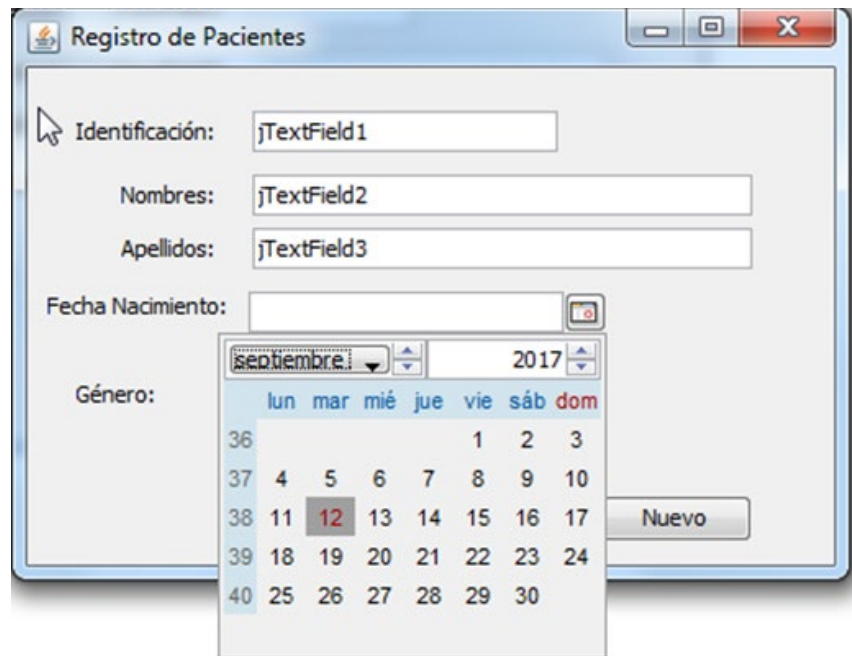


Figura 39. Agregando el página de descarga del componente JCalendar.

Fuente: SENA

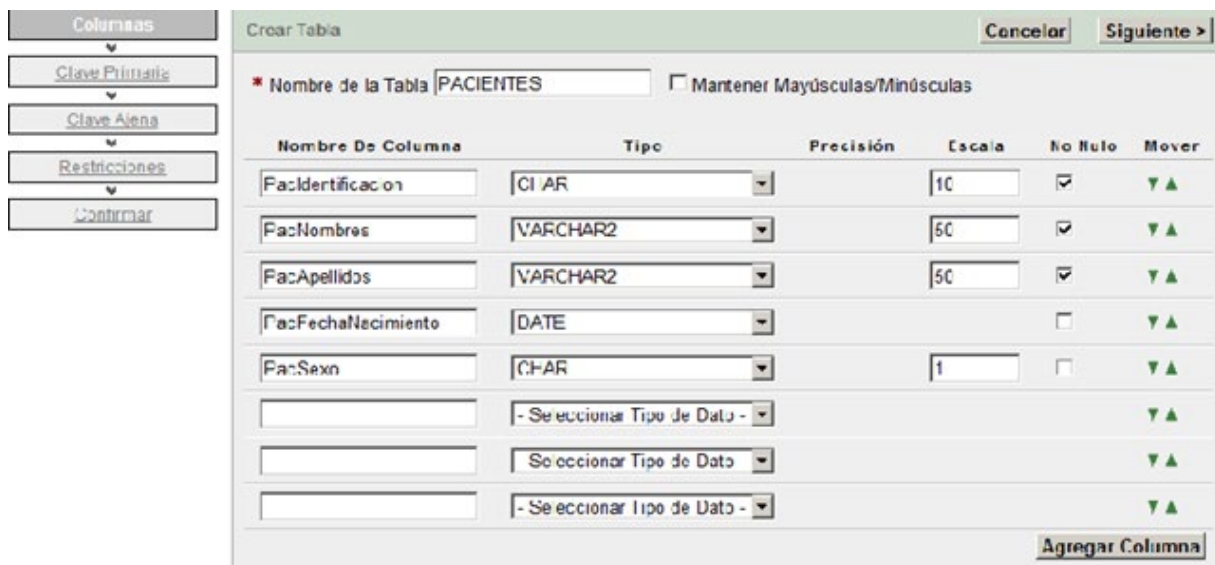
Para verificar el resultado parcial de la interface gráfica, se hace clic en el botón **Preview Design**.



The screenshot shows a Java Swing window titled "Registro de Pacientes". It contains several text input fields labeled "Identificación:", "Nombres:", and "Apellidos:", each followed by a text field (jTextField1, jTextField2, jTextField3). Below these is a "Fecha Nacimiento:" label followed by a date picker. The date picker shows "septiembre" and "2017". A calendar popup is visible, showing the month of September 2017. The calendar has days of the week (lun, mar, mié, jue, vie, sáb, dom) and dates (1-30). The date "12" is highlighted. To the right of the calendar is a "Nuevo" button.

Figura 40. Visualizando el componente Calendario.
Fuente: SENA

Se observa como los campos del formulario coinciden con los campos de la tabla pacientes en la base de datos.



The screenshot shows the "Crear Tabla" (Create Table) dialog box. The table name is "PACIENTES". The table structure is defined by the following columns:

Nombre De Columna	Tipo	Precisión	Escala	No Nulo	Mover
FacIdentificac on	CI AR		10	☒	▼ ▲
FacNombres	VARCHAR2		50	☒	▼ ▲
FacApellidos	VARCHAR2		50	☒	▼ ▲
FacFechaNacimiento	DATE			☐	▼ ▲
FacSexo	CHAR		1	☐	▼ ▲
	- Seleccionar Tipo de Dato -				▼ ▲
	- Seleccionar Tipo de Dato -				▼ ▲
	- Seleccionar Tipo de Dato -				▼ ▲

Buttons: "Cancelar", "Siguiente >", "Agregar Columna".

Figura 41. Verificación de campos de la tabla.
Fuente: SENA

• DISEÑO DEL FORMULARIO PARA LA CONSULTA DE PACIENTES

Otra de las funcionalidades que se implementan para la gestión de pacientes es la consulta de los mismos, para ello se desarrollará un nuevo formulario de tipo **jInternalFrame**. Se crea un nuevo formulario de tipo **jInternalframe**. Dentro del paquete vista y se le asigna el nombre **ConsPacienteInternalFrame**.

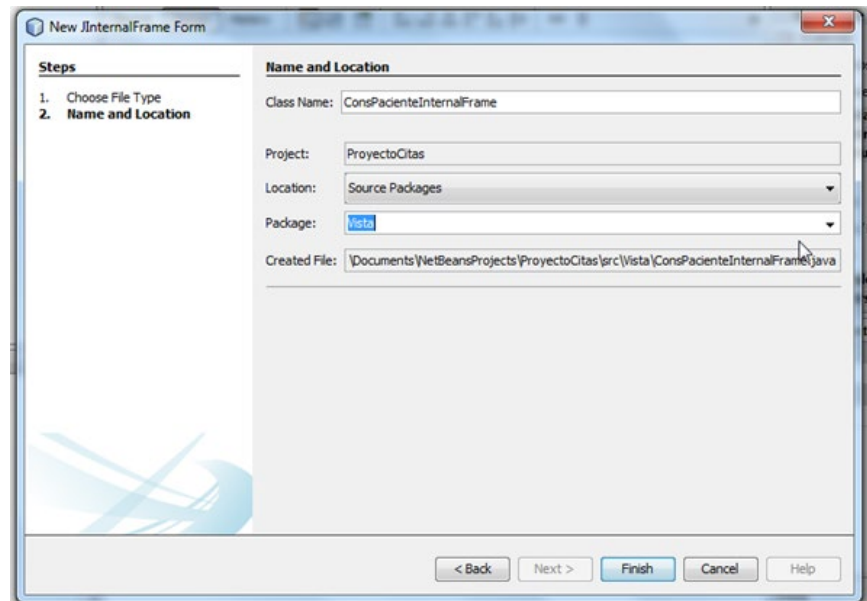


Figura 42. Agregando el formulario de consulta de pacientes.
Fuente: SENA

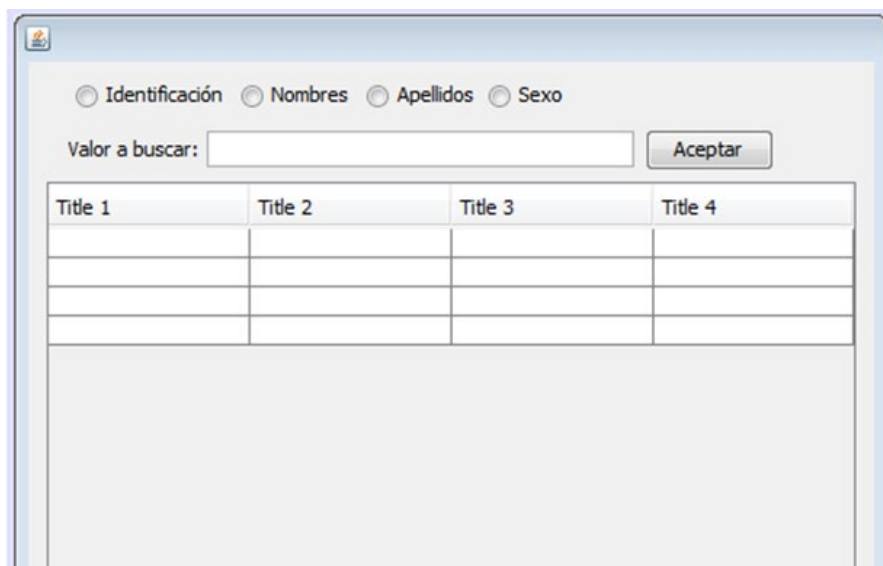
El diseño del nuevo formulario es el siguiente:

Tabla 4

Datos - Diseño del nuevo formulario.

Control	Propiedad	Valor
buttonGroup1		
JRadioButton1	Text I	dentificación
	buttonGroup	buttonGroup1
JRadioButton2	Text N	ombres
	buttonGroup	buttonGroup1
JRadioButton3	Text A	pellidos
	buttonGroup	buttonGroup1
JRadioButton4	Text S	exo
	buttonGroup	buttonGroup1
JLabel1	Text V	alor a buscar
JTextfield1	Text	
	Name V	alorTxt
JButton1 T	ext	Aceptar
	Name A	ceptarBtn
JTable1	Name R	esultadosTbl

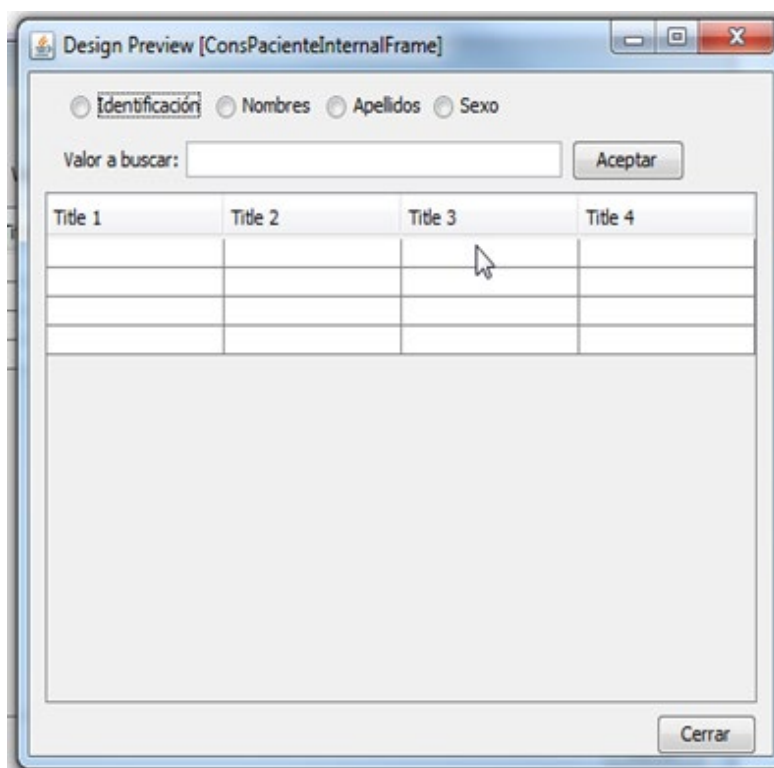
Nota: Producción SENA



Title 1	Title 2	Title 3	Title 4

Figura 43. Configurando los controles en el formulario de consulta.
Fuente: SENA

Para verificar el resultado final de la interface gráfica, se hace clic en el botón **Preview Design**.



Title 1	Title 2	Title 3	Title 4

Figura 44. Preview del formulario de consulta de pacientes.
Fuente: SENA

El paquete vista ha quedado conformado por tres formularios, el formulario principal **PrincipalJFrame** que contiene el menú y los formularios internos, **regpacienteinternalframe** en el cual se va a llevar a cabo el registro de pacientes y **conspacienteinternalframe** donde se podrá realizar consultas de pacientes.

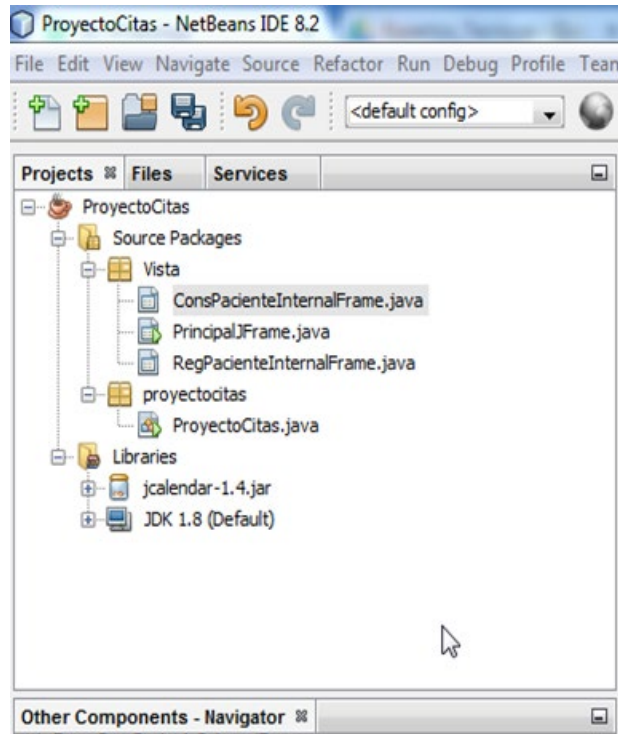


Figura 45. Visualización de los formularios en el proyecto.
Fuente: SENA

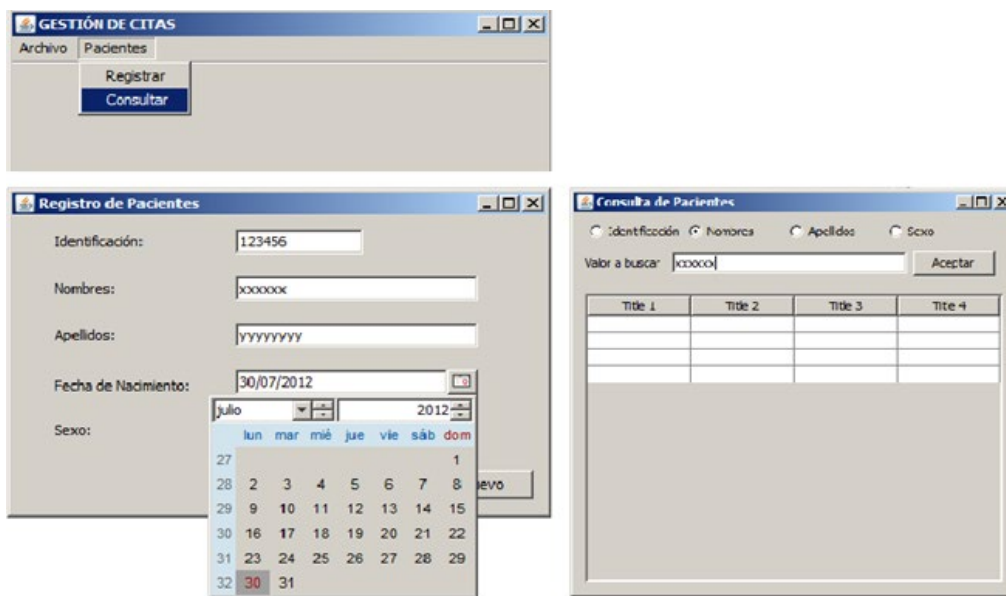


Figura 46. Ejecución de los formularios.
Fuente: SENA

Desarrollo de aplicaciones DE ESCRITORIO CON LENGUAJE



Sesión 3

Creación el
modelo de la
aplicación



Teniendo la base de datos y los formularios de la aplicación contruidos (la vista) es hora de pasar al modelo, el modelo está conformado por las clases que forman parte del dominio de la aplicación. Se Recuerda que el caso de uso que se desarrolla en el presente material es sobre “**Administrar Datos Paciente**”.

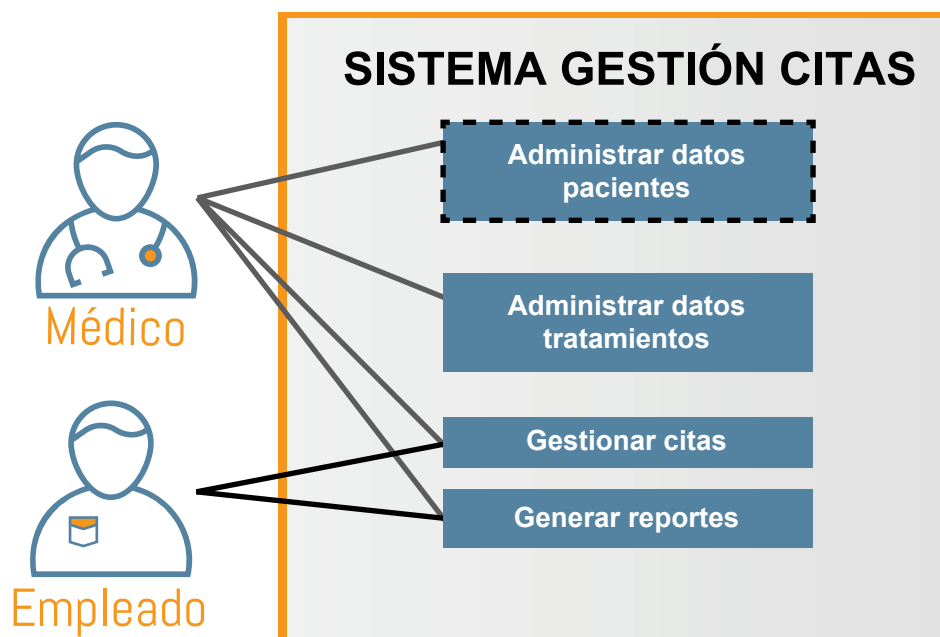


Figura 47. Caso de uso administrar datos pacientes.
Fuente: SENA

NOTA. Para identificar las clases que forman parte del modelo es necesario revisar el diagrama de clases realizado durante el análisis y refinado durante el diseño.

Para el desarrollo del sistema completo, seguramente se necesitarán varias clases formando parte del modo: paciente, cita, tratamiento, médico; sin embargo, para este caso particular, **administrar datos paciente** sólo se requiere la **clase paciente**, encargada de manejar la información de un determinado paciente y **clase gestor paciente** que será la encargada de ejecutar operaciones que afectan a varios pacientes.

• Creación clase “paciente”

Crear un nuevo paquete dentro del proyecto **ProyectoCitas**.

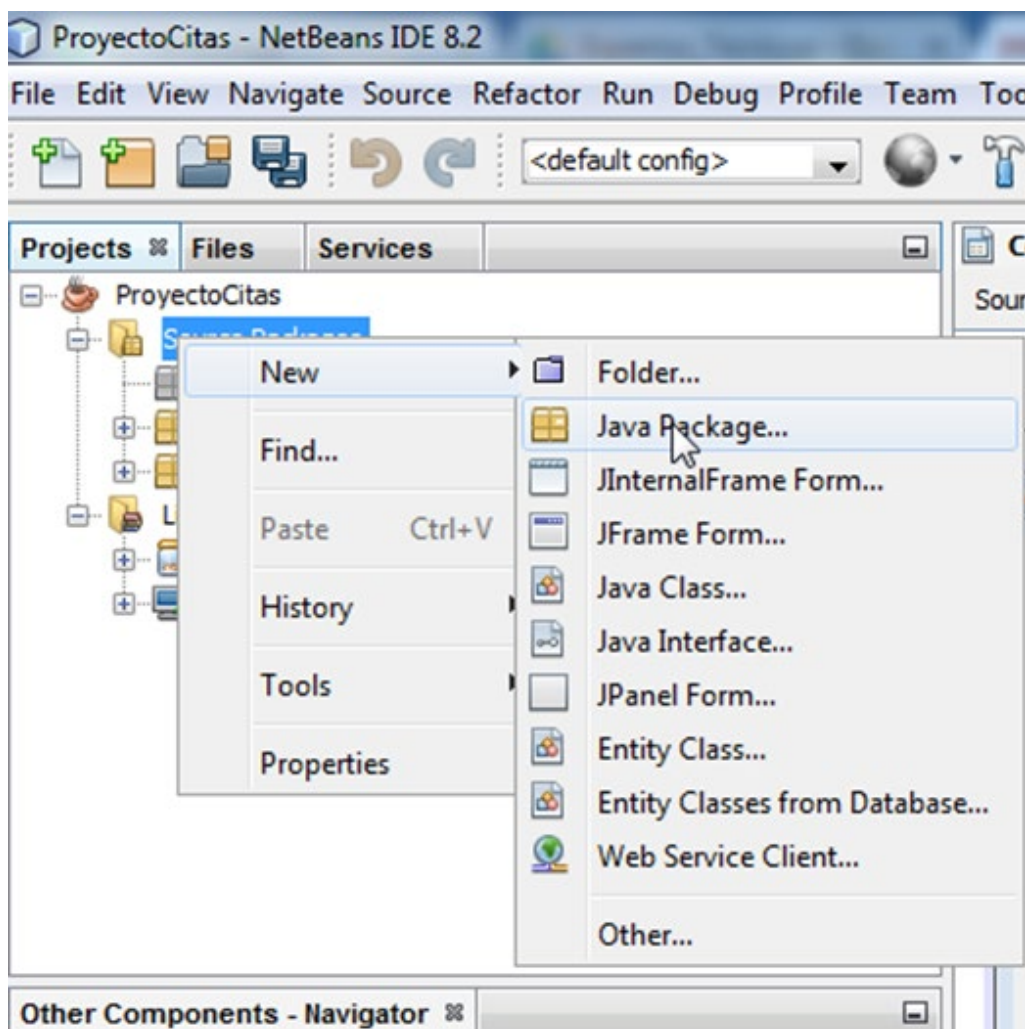


Figura 48. Creando un nuevo paquete.
Fuente: SENA

Asignarle nombre al paquete modelo y hacer clic en **Finish**.

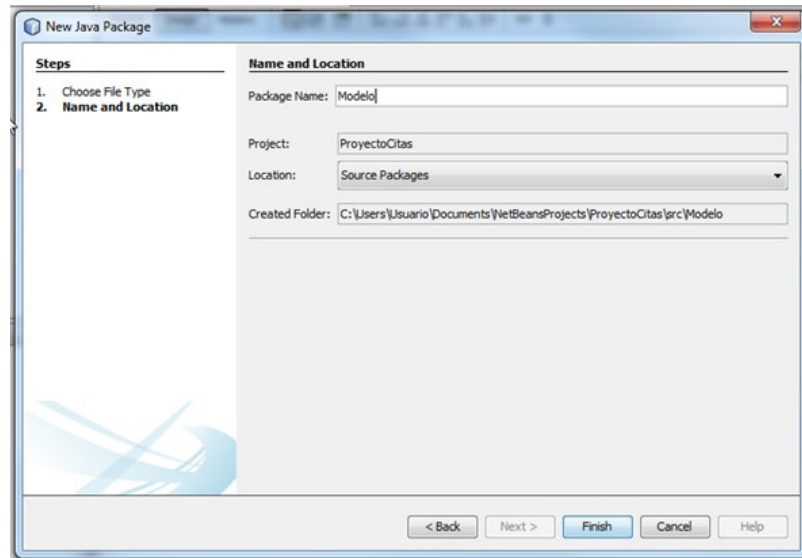


Figura 49. Nombrando el paquete Modelo.
Fuente: SENA

Sobre el nuevo paquete creado modelo, hacer clic derecho y seleccionar del menú contextual la opción **New→Java Class**

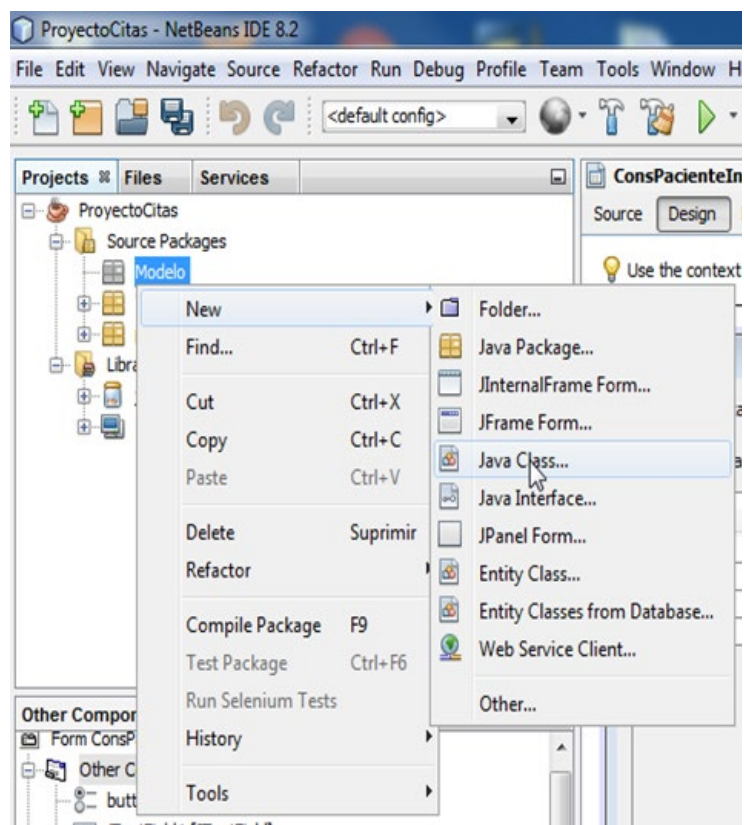


Figura 50. Página de descarga del componente JCalendar.
Fuente: SENA

Se ingresa el nombre de la clase paciente y se hace clic en **Finish**.

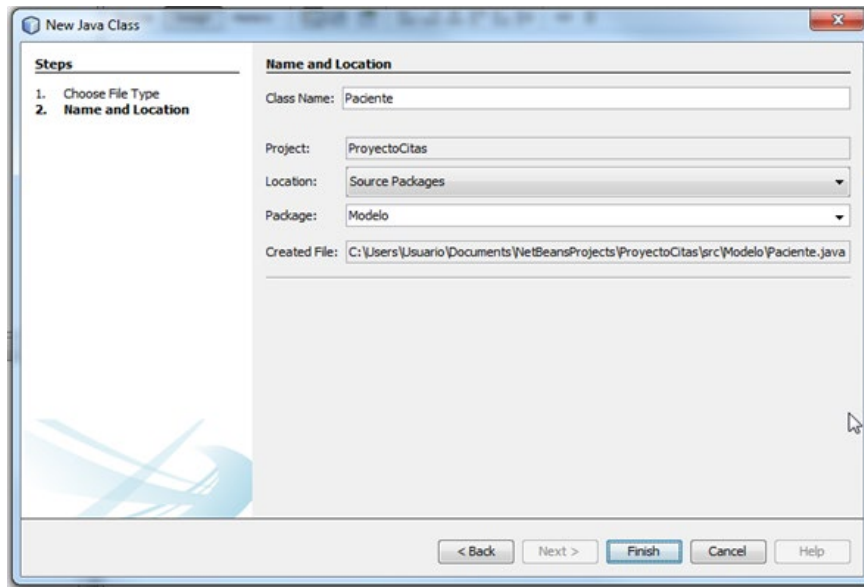


Figura 51. Agregando la clase paciente.
Fuente: SENA

Al hacer clic en **Finish** aparece una plantilla de clase para la clase paciente similar a la siguiente:

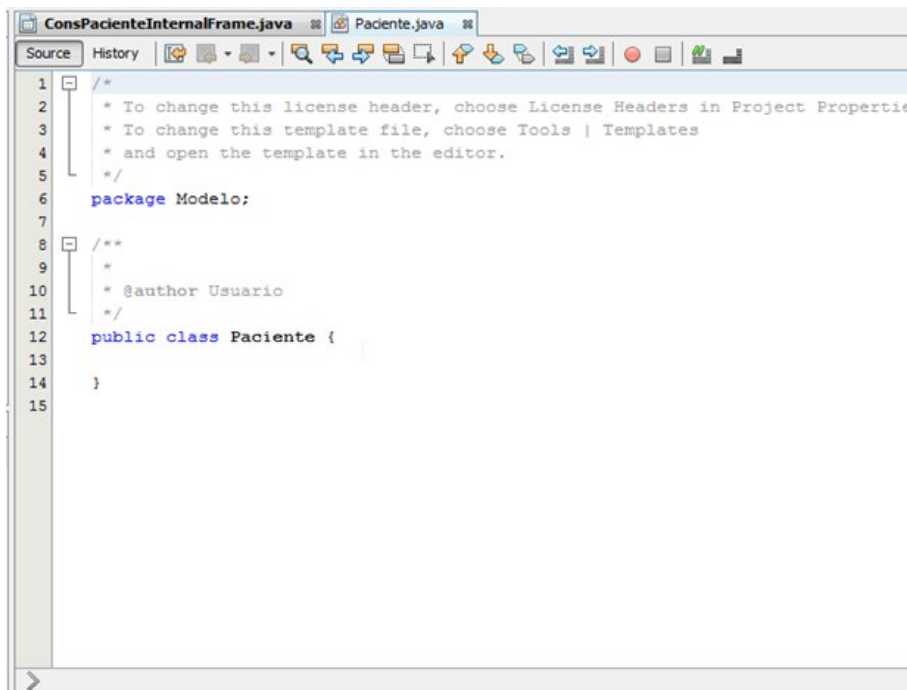
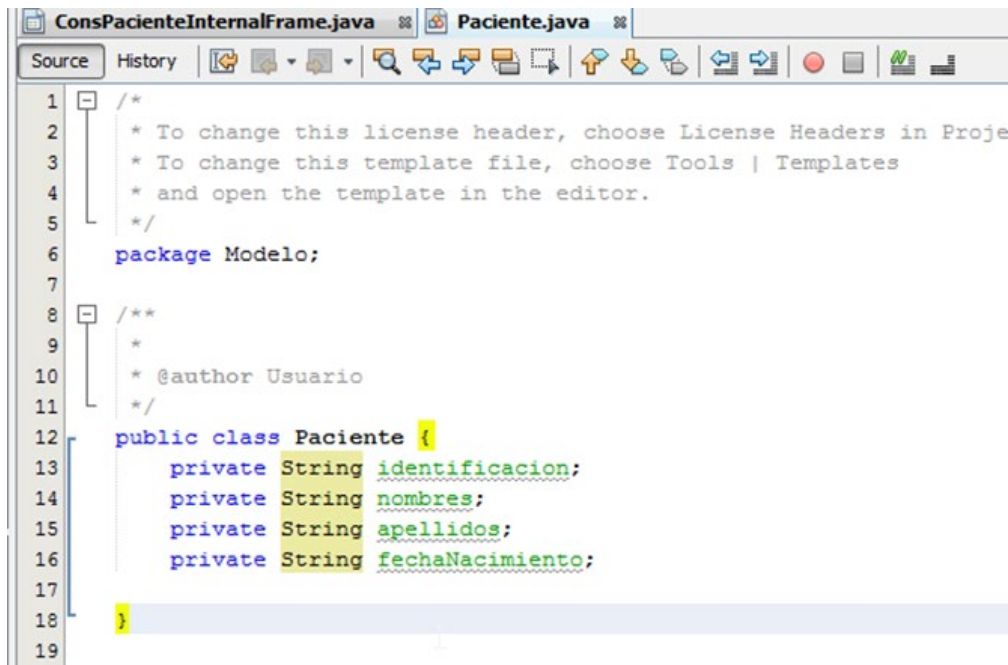


Figura 52. Estructura de la clase pacientes.
Fuente: SENA

Borrar los comentarios e ingresar los atributos de la clase.



```

1  /*
2  * To change this license header, choose License Headers in Project Properties
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package Modelo;
7
8  /**
9   *
10  * @author Usuario
11  */
12  public class Paciente {
13      private String identificacion;
14      private String nombres;
15      private String apellidos;
16      private String fechaNacimiento;
17  }
18
19

```

Figura 53. Atributos de la clase paciente.
Fuente: SENA

Hacer clic derecho sobre el código y seleccionar del menú:

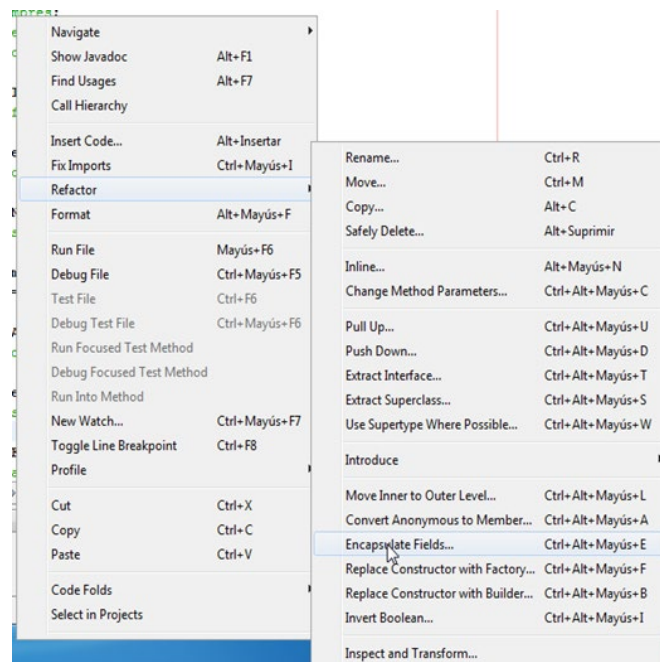


Figura 54. Encapsulando los atributos.
Fuente: SENA

Se selecciona todos los campos y se hace clic sobre el botón **Refactor**.

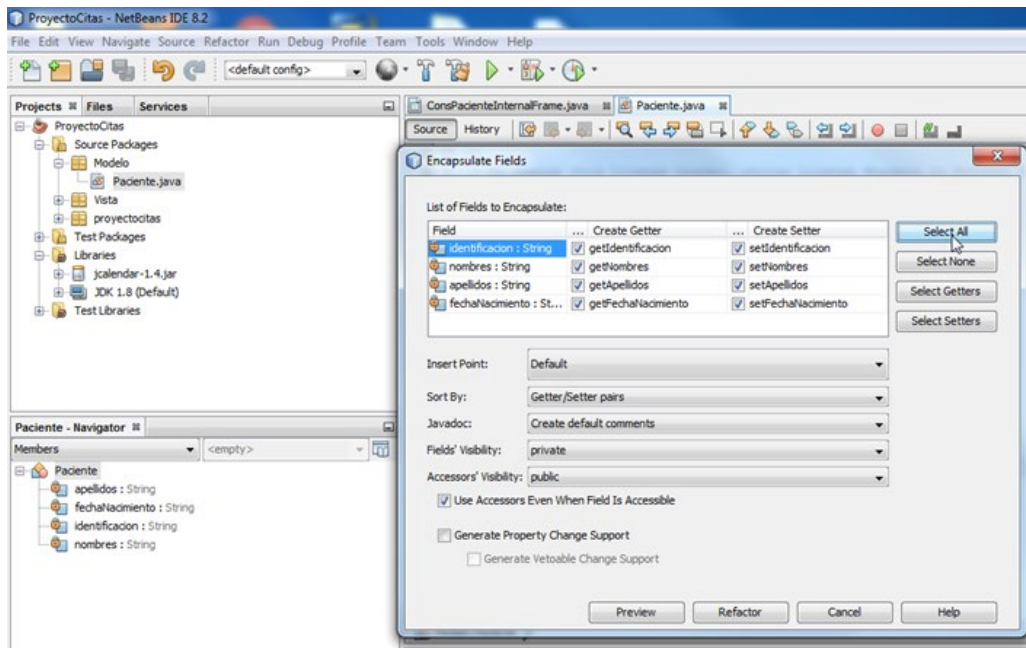


Figura 55. Configurando los Getters y los Setters de los atributos.

Fuente: SENA

De esta forma aparecen todos los métodos **Getters y Setters** asociados a los atributos definidos en la clase paciente.

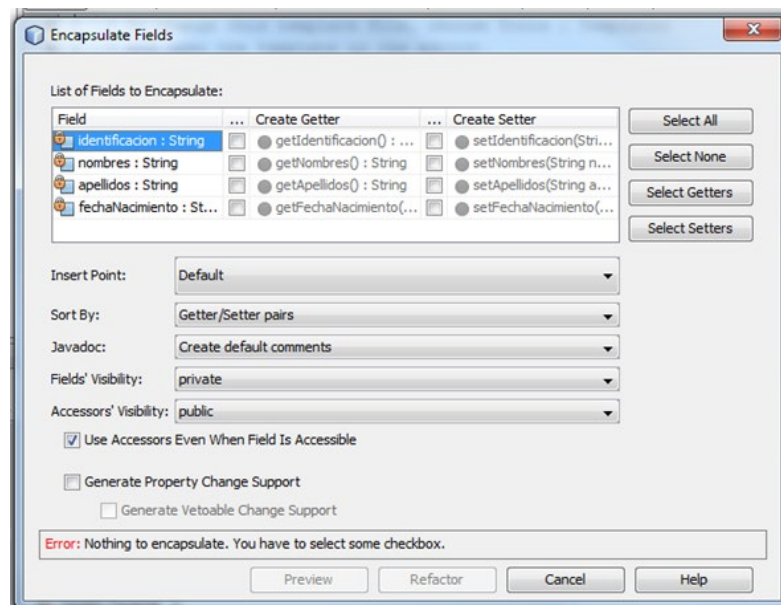
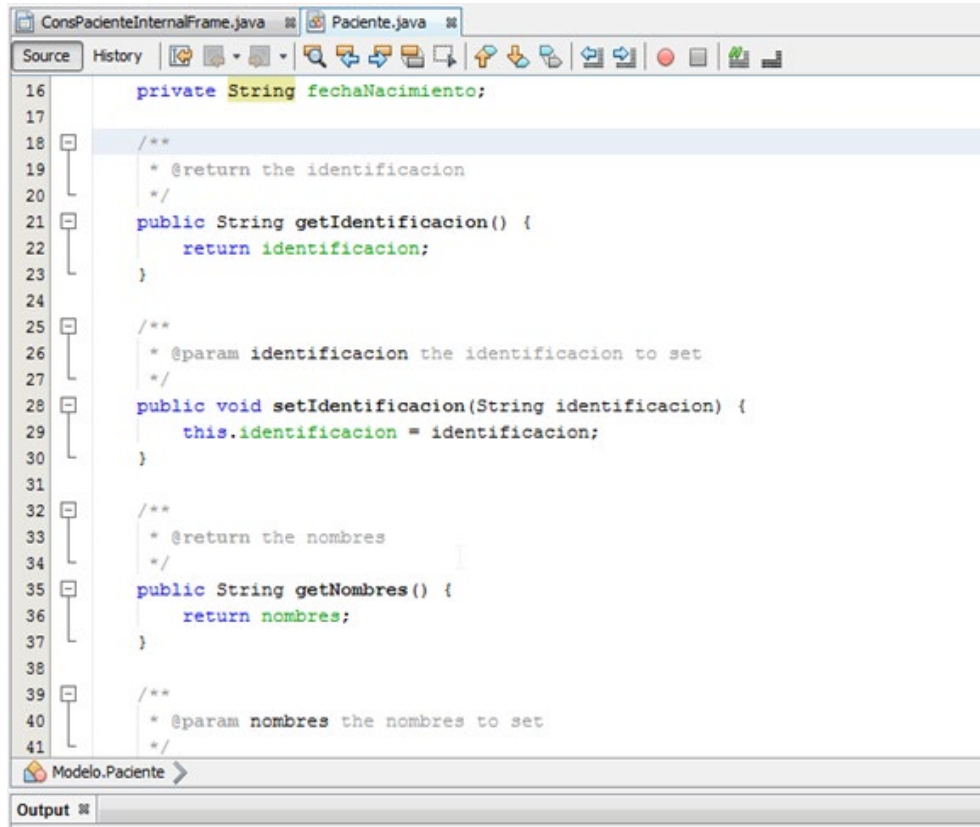


Figura 56. Configurando los Getters y los Setters de los atributos.

Fuente: SENA



```

16     private String fechaNacimiento;
17
18     /**
19      * @return the identificacion
20      */
21     public String getIdentificacion() {
22         return identificacion;
23     }
24
25     /**
26      * @param identificacion the identificacion to set
27      */
28     public void setIdentificacion(String identificacion) {
29         this.identificacion = identificacion;
30     }
31
32     /**
33      * @return the nombres
34      */
35     public String getNombres() {
36         return nombres;
37     }
38
39     /**
40      * @param nombres the nombres to set
41      */

```

Figura 57. Getters y los Setters de la clase Pacientes.
Fuente: SENA

NOTA. Un método **Getter** es aquel que retorna el valor de un atributo, por ejemplo, **Getnombres** es un método que retorna los nombres de un paciente. Un método **Setter** es aquel que se emplea para modificar el valor de un atributo, por ejemplo Setnombres un método que modifica el nombre de un paciente. El prefijo get y set forman parte de una convención empleada en java y conocida como **Javebeans**.

Apartir de la línea 11, se crea el constructor de la clase paciente y borrando los comentarios, el código queda de la siguiente manera.

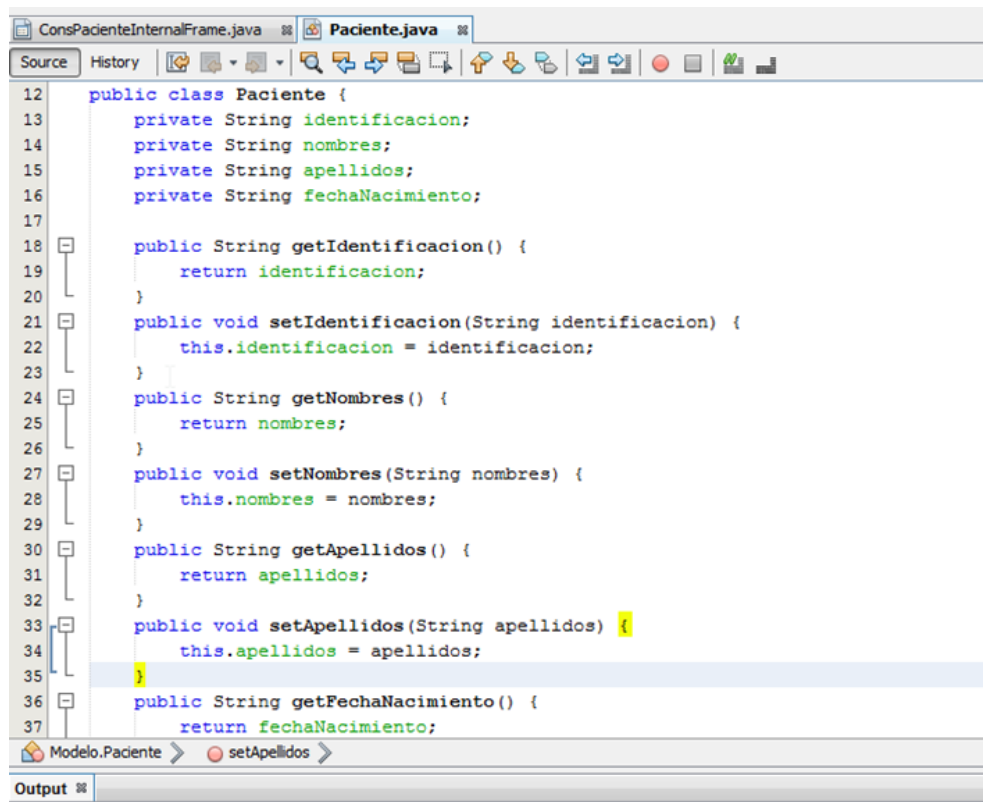


Figura 58. Configurando los Getters y los Setters de los atributos.
Fuente: SENA

• Creación de la clase “gestor paciente”

Se crea una nueva clase dentro del paquete modelo llamado **Gestorpaciente** y se hace clic en el botón **Finish**.

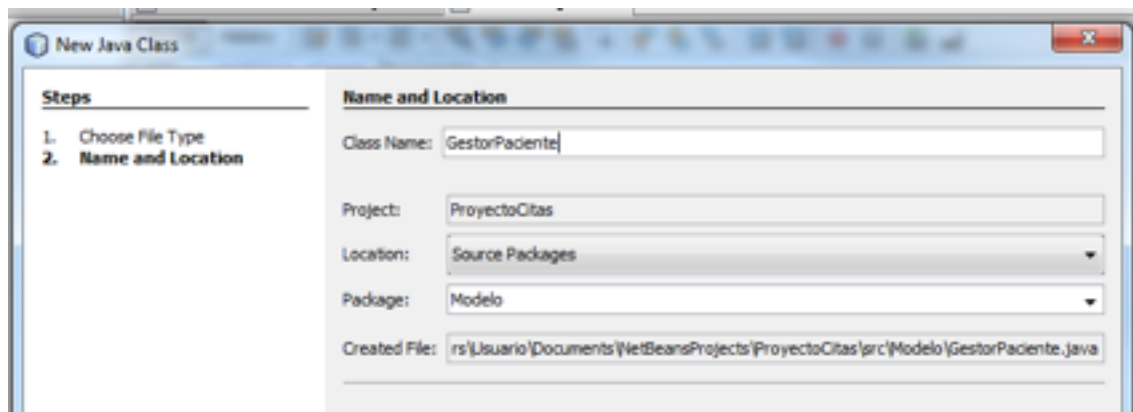


Figura 59. Creando la clase GestorPaciente. ibutos.
Fuente: SENA

NOTA. Los códigos expuestos en el tutorial no tienen comentarios ni siguen un estándar de codificación predefinido, sin embargo es necesario incluir estos aspectos de calidad en el código propio.

El código de esta nueva clase es el siguiente:

```
package Modelo;
import java.util.*;
/**
 *
 * @author Usuario
 */
public class GestorPaciente {
    private static LinkedList<Paciente> pacientes;

    public GestorPaciente(){
        pacientes=new LinkedList<Paciente>();
    }

    public void RegistrarPacientes(Paciente paciente){
        pacientes.add(paciente);
    }
    public static LinkedList<Paciente> getPacientebyParametro(int parametro, String valor)
    {
        LinkedList<Paciente> resultado=new LinkedList<Paciente>();
        for(Paciente pac:pacientes)
        {
            switch(parametro){

                case 1: if(pac.getIdentificacion().equals(valor))
                    resultado.add(pac);
                    break;

                case 2: if(pac.getNombres().equals(valor))
                    resultado.add(pac);
                    break;

                case 3: if(pac.getApellidos().equals(valor))
                    resultado.add(pac);
                    break;

                case 4: if(pac.getGenero().equals(valor))
                    resultado.add(pac);
                    break;

            }
        }
    }
}
```

Explicación del código.

Tabla 5

Explicación del código. .

Línea	Explicación
2	Importación de la clase LinkedList para el almacenamiento de Pacientes.
5	Declaración de la variable pacientes que será la responsable de almacenar todos los pacientes registrados.
7..10	En el constructor de la clase se instancia el LinkedList determinando que en cada posición se almacenará un objeto de tipo Paciente .
11..14	Método que recibe un objeto de tipo Paciente y lo guarda dentro del LinkedList .
15..37	Método que recibe un criterio de búsqueda y un valor y retorna una lista de pacientes que cumplen con los parámetros recibidos.
17	Se crea un LinkedList interno para almacenar los pacientes que cumplen con los criterios de la consulta.
18	Recorrido de cada uno de los pacientes registrados.
20	Evalúa el valor del parámetro para conocer si se desea hacer la búsqueda por Identificación 1, Nombres 2, Apellidos 3, Sexo 4 .
22..33	De acuerdo con el parámetro de búsqueda, verifica si coincide cada paciente con el valor recibido. En ese caso, se agrega al LinkedList interno resultado .
36	Retorna el LinkedList interno resultado con los pacientes encontrados.

Nota: Producción SENA

Desarrollo de aplicaciones DE ESCRITORIO CON LENGUAJE



Sesión 4

Enlace con el
controlador



Ahora que la aplicación cuenta con la vista que proporciona los formularios requeridos para interactuar con el usuario (**PrincipalJframe**, **RegPaciente**, **InternalFrame** y **ConsPacienteInternalFrame**) y cuenta con el modelo, conformado por las clases del dominio de la aplicación (paciente y gestor paciente), es el momento de crear las clases del controlador, estas clases gestionarán las solicitudes de la vista y darán respuesta a las mismas recurriendo a las clases de modelo este mecanismo de interacción multicapas se conoce como el patrón modelo vista controlador (MVC).

• Controlador paciente control

NOTA. Para poder acceder desde la clase controladora al contenido de los campos de los formularios, estos deben permitir su acceso bien sea a través de métodos públicos o modificando el nivel de acceso de estos campos pasándolos de **Private** a **Públicos**.

Se abre el formulario **RegpacienteInternalFrame** que se encuentra en el paquete vista y se hace clic derecho sobre el primer cuadro de texto, del menú contextual se selecciona la opción **Properties**.

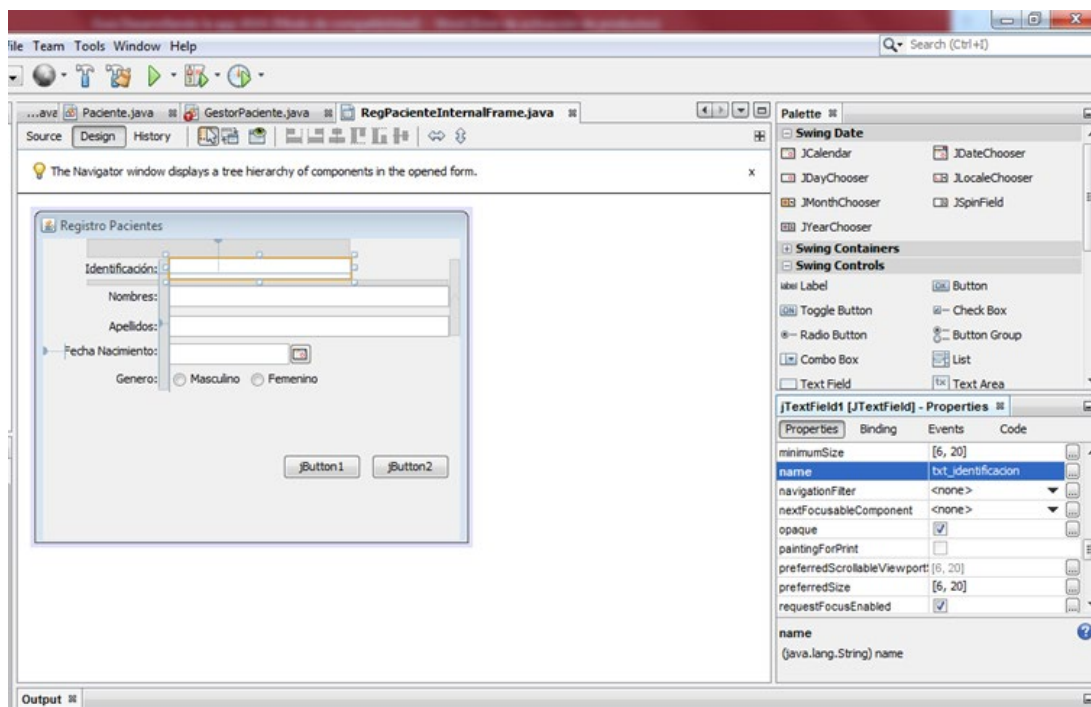


Figura 60. Creando el formulario de registro de paciente.
Fuente: SENA

De la ventana propiedades, seleccionar ficha **Code** y modificar los valores de las propiedades variable **Name** y variable **Modifiers** por identificación **txt_identificacion** y **public** respectivamente.

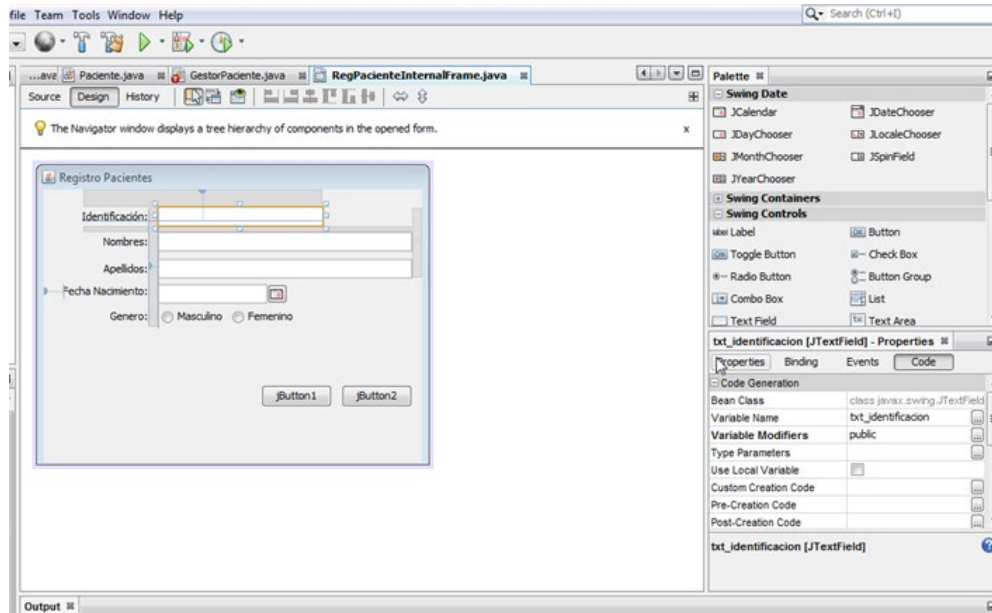


Figura 61. Creando el formulario de registro de pacientes.
Fuente: SENA

Hacer clic derecho en las demás cajas de texto, el control **JDateChooser** y los controles **JRadioButton** y repetir el proceso para los campos nombres, apellidos, fechas de nacimiento, genero, al igual que para los botones de registrar y nuevo tal como se presenta en siguiente cuadro.

Tabla 6
Datos - Modificación campo variable.

Nombre etiqueta	Variable Name	Variable Modifiers
identificación	txt_identificacion	public
nombres	txt_nombres	public
apellidos	txt_apellidos	public
Fecha Nacimiento	Dtd_fecha_nacimiento	public
Masculino	rdb_masculino	public
Femenino	rdb_femenino	public

Nota: Producción SENA

NOTA. No basta con seleccionar otra caja de texto para modificar sus propiedades, es necesario hacer clic derecho sobre ella y seleccionar la opción Properties del menú contextual. (Aplica para todos los objetos).

Crear un nuevo paquete llamado **Controlador** dentro del proyecto **ProyectoCitas**.

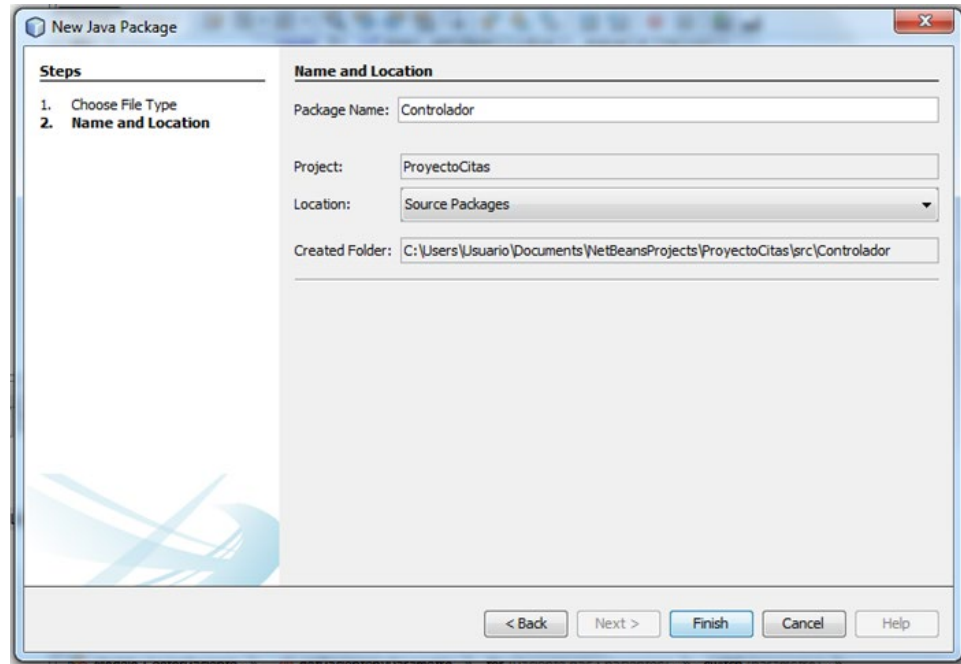


Figura 62. Creando la clase Controlador.
Fuente: SENA

Se hace clic derecho sobre el paquete controlador recién creado, se selecciona del menú contextual New → **Java Class**.

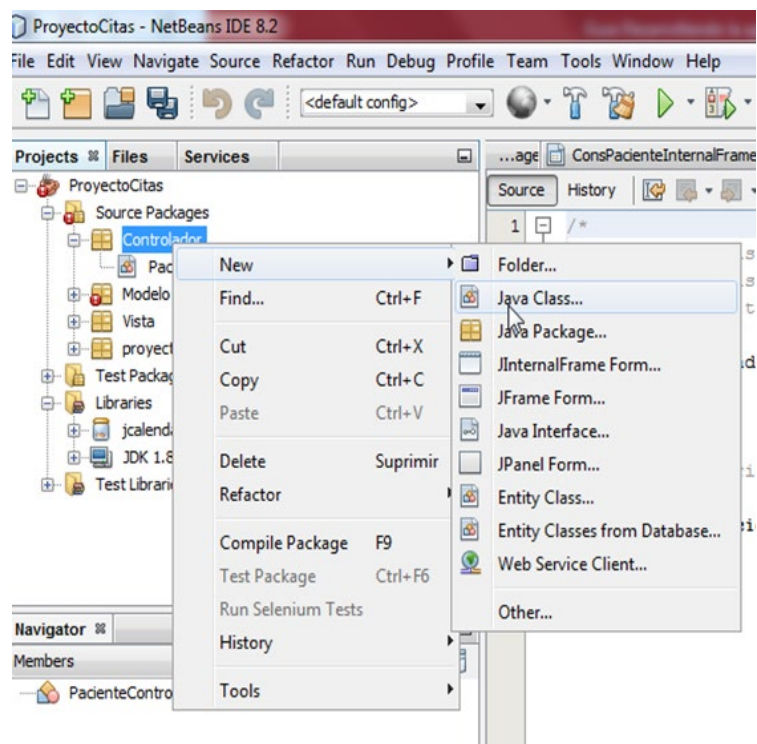


Figura 63. Creando la clase.
Fuente: SENA

Ingresa el nombre para la nueva clase **PacienteControl** y se hace clic sobre el botón **Finish**.

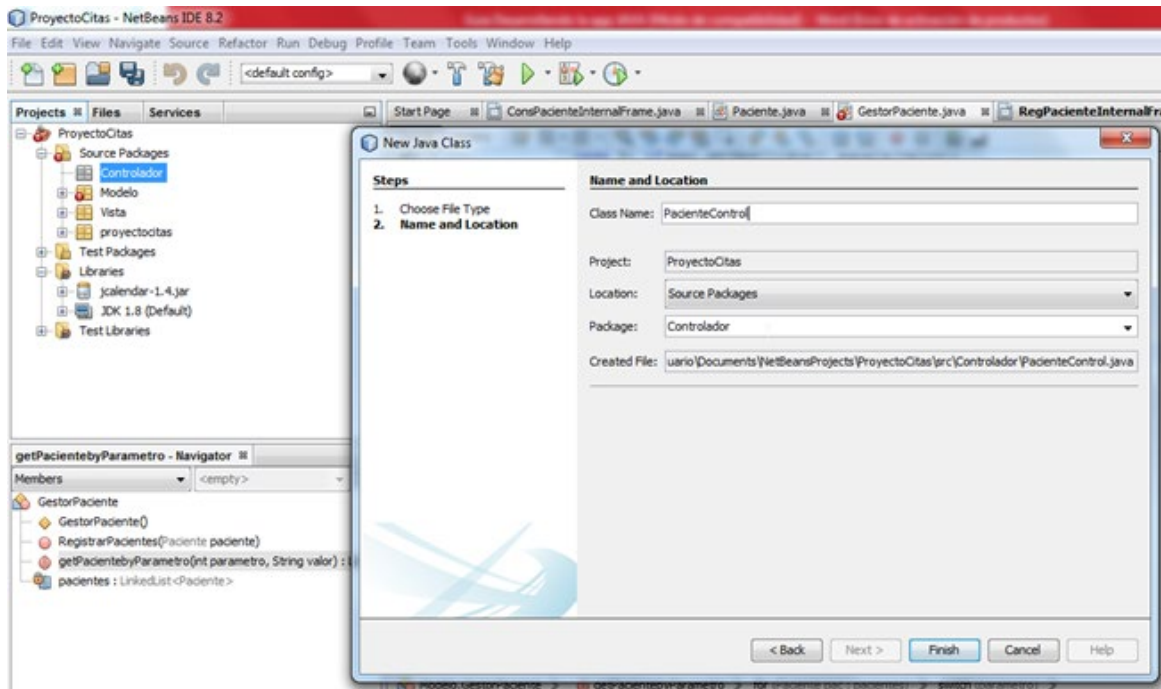


Figura 64. Creando la clase PacienteControl.
Fuente: SENA

El objetivo de esta clase es comunicarse con la vista para capturar los datos ingresados por el usuario en el formulario de registro de clientes y posteriormente con esos datos capturados comunicarse con el modelo para crear un objeto de la clase paciente y registrarlo a través de la clase **GestorPaciente** del modelo.

El código correspondiente a la clase **PacienteControl** es el siguiente:

```
package Controlador;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.SimpleDateFormat;

public class PacienteControl implements ActionListener {
    Vista.RegPacienteInternalFrame pacienteVista;
    Modelo.Paciente pacienteModelo;
    Modelo.GestorPaciente gestorpacienteModelo;

    public PacienteControl(Vista.RegPacienteInternalFrame pacienteVista){

        this.pacienteVista=pacienteVista;
        gestorpacienteModelo=new Modelo.GestorPaciente();

    }

    @Override
    public void actionPerformed(ActionEvent e){
        if(e.getSource().equals(pacienteVista.jButton1)){

            String identificacion=pacienteVista.txt_identificacion.getText();
            String nombres=pacienteVista.txt_nombres.getText();
            String apellidos=pacienteVista.txt_apellidos.getText();
            SimpleDateFormat formato=new SimpleDateFormat("dd/MM/yyyy");
            String fecha_nacimiento=formato.format(pacienteVista.Dtd_Fecha_Nacimiento.getDate());
            String genero="";
            if(pacienteVista.rdb_masculino.isSelected()){
                genero="M";
            }

            if(pacienteVista.rdb_femenino.isSelected()){
                genero="F";
            }
            pacienteModelo=new Modelo.Paciente(identificacion,nombres,apellidos,fecha_nacimiento,genero);
            gestorpacienteModelo.RegistrarPacientes(pacienteModelo);

        }

        if(e.getSource().equals(pacienteVista.jButton2)){

            pacienteVista.txt_identificacion.setText("");
            pacienteVista.txt_nombres.setText("");
            pacienteVista.txt_apellidos.setText("");
            pacienteVista.Dtd_Fecha_Nacimiento.setDate(null);
            pacienteVista.rdb_masculino.setSelected(true);
            pacienteVista.rdb_femenino.setSelected(false);
            pacienteVista.txt_identificacion.requestFocus();

        }
    }
}
```

```

1  package controlador;
2
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.text.SimpleDateFormat;
6
7  public class PacienteControl implements ActionListener
8  {
9      vista.RegPacienteInternalFrame pacienteVista;
10     modelo.Paciente pacienteModelo;
11     modelo.GestorPaciente gestorPacienteModelo;
12
13     public PacienteControl(vista.RegPacienteInternalFrame pacienteVista)
14     {
15         this.pacienteVista=pacienteVista;
16         gestorPacienteModelo = new modelo.GestorPaciente();
17     }
18     @Override
19     public void actionPerformed(ActionEvent e)
20     {
21         if(e.getSource().equals(pacienteVista.RegistrarBtn))
22         {
23             String identificacion=pacienteVista.IdentificacionTxt.getText();
24             String nombres=pacienteVista.NombresTxt.getText();
25             String apellidos=pacienteVista.ApellidosTxt.getText();
26             SimpleDateFormat formato = new SimpleDateFormat("dd/MM/yyyy");
27             String fechaNacimiento=formato.format(pacienteVista.FechaNacimientoDec.getDate());
28             String sexo=null;
29             if(pacienteVista.MasculinoOpt.isSelected())
30                 sexo="m";
31             else
32                 sexo="f";
33             pacienteModelo= new modelo.Paciente(identificacion, nombres, apellidos, fechaNacimiento, sexo);
34             gestorPacienteModelo.registrarPaciente(pacienteModelo);
35         }
36         if(e.getSource().equals(pacienteVista.NuevoBtn))
37         {
38             pacienteVista.IdentificacionTxt.setText(null);
39             pacienteVista.NombresTxt.setText(null);
40             pacienteVista.ApellidosTxt.setText(null);
41             pacienteVista.FechaNacimientoDec.setDate(null);
42             pacienteVista.MasculinoOpt.setSelected(false);
43             pacienteVista.FemeninoOpt.setSelected(false);
44             pacienteVista.IdentificacionTxt.requestFocus();
45         }
46     }
47 }

```

Figura 65. Código clase PacientControl.
Fuente: SENA

Explicación del código.

Tabla 7

Explicación código - clase PatientControl

Línea	Explicación
3,4	Importación de paquetes para el manejo del evento clic del botón Registrar del formulario Registro de Pacientes .
5	Importación de la clase SimpleDateFormat para dar formato a la fecha de nacimiento recibida desde la vista .
7	ActionListener es la interfaz que se debe implementar para escuchar eventos.
9,10,11	Declaración de variables asociadas a la vista y al modelo . El controlador debe conocer a la vista y al modelo para poder comunicarse con ellos.
13	El constructor de la clase controladora recibe como parámetro la vista que debe controlar.
15,16	Creación de las variables asociadas a la vista y el modelo .
18	Anotación que indica que el método actionPerformed se está sobrescribiendo.
19	Método de la interfaz ActionListener que se debe implementar. Este método se ejecuta cuando se hace clic en el botón Registrar del formulario Registro de Pacientes .
21	Verifica si fue el botón Registrar del formulario Registro de Pacientes el que generó el evento.
23..32	Captura de la información ingresada por el usuario en el formulario Registro de Pacientes .
33	Creación de un objeto de tipo Paciente a partir de los datos ingresados por el usuario.
34	Registro del paciente creado a través de la clase GestorPaciente del modelo.
36	Verifica si fue el botón Nuevo del formulario Registro de Pacientes el que generó el evento.
38..44	Borra los contenidos de los controles del formulario Registro de Pacientes .

Nota: Producción SENA

• Controlador **gestorPacienteControl**

Ahora que el controlador **pacientecontrol** intermedio entre la vista y el modelo para registrar pacientes, se creará otro controlador que se comunique con el modelo para extraer los datos de los pacientes registrados y visualizados en la vista.

El objetivo de la nueva clase controladora es capturar la solicitud que el usuario realiza a través del formulario consulta de pacientes y recibir del modelo la información pertinente para llevarla a la vista. Al igual que ocurrió con el formulario de registro pacientes, se debe tener acceso a los elementos del formulario consulta de pacientes desde el controlador.

Hacer clic derecho en el primer control del formulario (***jradiobutton1***). Se selecciona del menú contextual la opción **Properties**.

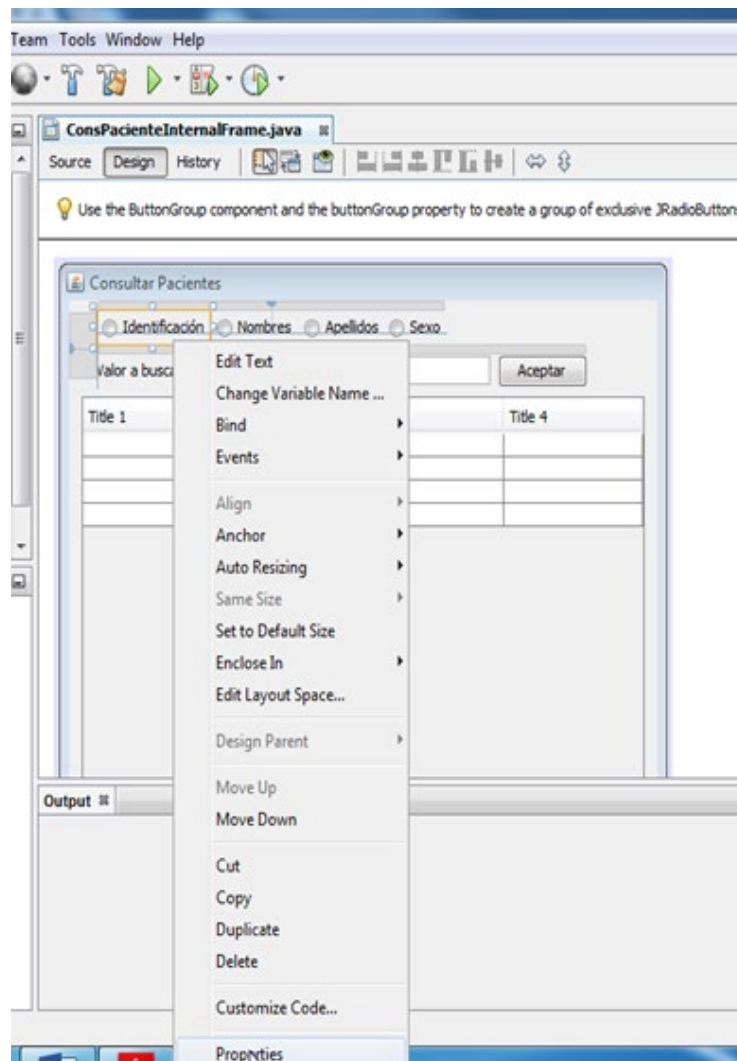


Figura 66. Configurando las propiedades de los objetos.
Fuente: SENA

En la venta de propiedades se selecciona la ficha **Code** y en ella se realizan los cambios a las propiedades variable **Name** y variable **Modifiers**, ingresando los valores **rdb_identificacion** y **Public** respectivamente.

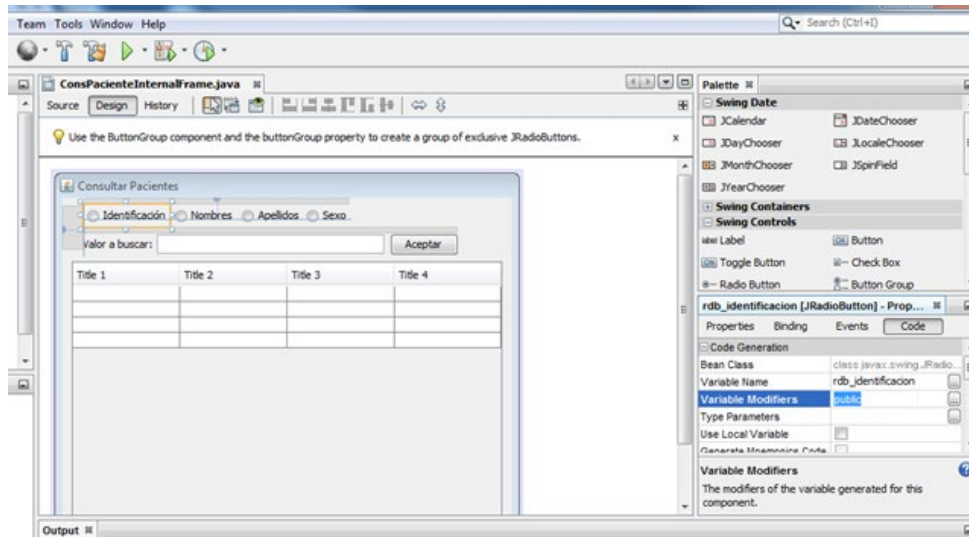


Figura 67. Configurando las propiedades de los objetos.
Fuente: SENA

Modificar la misma manera propiedades variable **Name** y variable **Modifiers** de los **Jradiobutton**, la caja de texto, el botón y el **JTable** tal como se presenta en la siguiente tabla.

Tabla 8
Modificación- variables de los Jradiobutton

Nombre etiqueta	Variable Name	Variable Modifiers
identificacion	rdb_identificacion	Public
nombres	rdb_nombres	Public
apellidos	rdb_apellidos	Public
genero	rdb_genero	Public
Valor a buscar	txt_valor	Public
Aceptar	btn_aceptar	Public
Tabla	Tbl_datos	Public

Nota: Producción SENA

NOTA. No basta con seleccionar otro control para modificar sus propiedades, es necesario hacer clic derecho sobre él y seleccionar la opción **Properties** del menú contextual. (Aplica para todos los pasos).

Ahora que ya se tiene acceso a los controles del formulario **Consultar Pacientes** por ser públicos, se procede a crear la clase controladora. Crear una nueva clase dentro del paquete controlador con el nombre **GestorPacienteControl** y se hace clic en **Finish**.

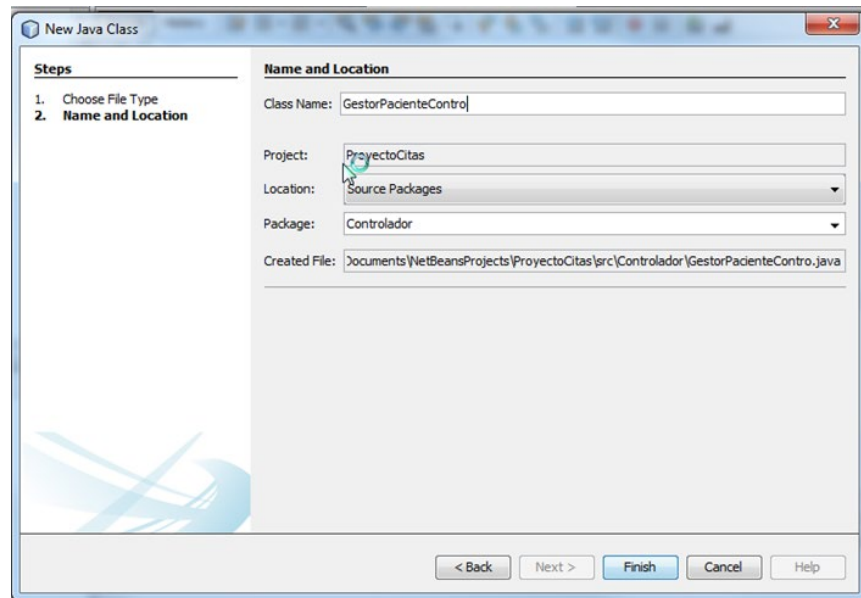


Figura 68. Creando la clase GestorPacienteControl.
Fuente: SENA

NOTA. El código expuesto no posee comentarios ni responde a un estándar de codificación establecida, es recomendable tener en cuenta estos aspectos en los proyectos de acuerdo con las sugerencias del instructor que orienta el proceso de formación.

Explicación del código.

Tabla 9
Explicación del código

Línea	Explicación
7	La clase debe poder escuchar eventos de la vista, en este caso el evento clic del botón Aceptar del formulario Consulta de Pacientes.
9,1	Declaración de variables asociadas a la vista y al modelo. El controlador debe conocer a la vista y al modelo para poder comunicarse con ellos.
12	El constructor de la clase controladora recibe como parámetro la vista que debe controlar.
14,15	Creación de las variables asociadas a la vista y el modelo.
17	Anotación que indica que el método ActionPerformed se está sobrescribiendo.
18	Método que se ejecuta cuando el usuario hace clic en el botón Aceptar del formulario Consulta de Pacientes.
20	Captura del valor introducido por el usuario en la caja de texto del formulario Consulta Pacientes.

Nota: Producción SENA

```

package Controlador;
import java.awt.event.*;
import java.util.LinkedList;
import javax.swing.table.DefaultTableModel;

public class GestorPacienteControl implements ActionListener {

    Modelo.GestorPaciente pacientesModelo;
    Vista.ConsPacienteInternalFrame consultarPacienteVista;

    public GestorPacienteControl(Vista.ConsPacienteInternalFrame consultarPacienteVista)
    {
        this.consultarPacienteVista=consultarPacienteVista;
        pacientesModelo=new Modelo.GestorPaciente();
    }

    public void actionPerformed(ActionEvent e){

        DefaultTableModel tmodelo;
        String valor=consultarPacienteVista.txt_valor.getText();
        int parametro=0;

        if(consultarPacienteVista.rdb_identificacion.isSelected()){
            parametro=1;
        }
        if(consultarPacienteVista.rdb_nombres.isSelected()){
            parametro=2;
        }
        if(consultarPacienteVista.rdb_apellidos.isSelected()){
            parametro=3;
        }
        if(consultarPacienteVista.rdb_genero.isSelected()){
            parametro=4;
        }

        LinkedList<Modelo.Paciente>
        pacientes=pacientesModelo.getPacientebyParametro(parametro, valor);
        String registro[]=new String[5];

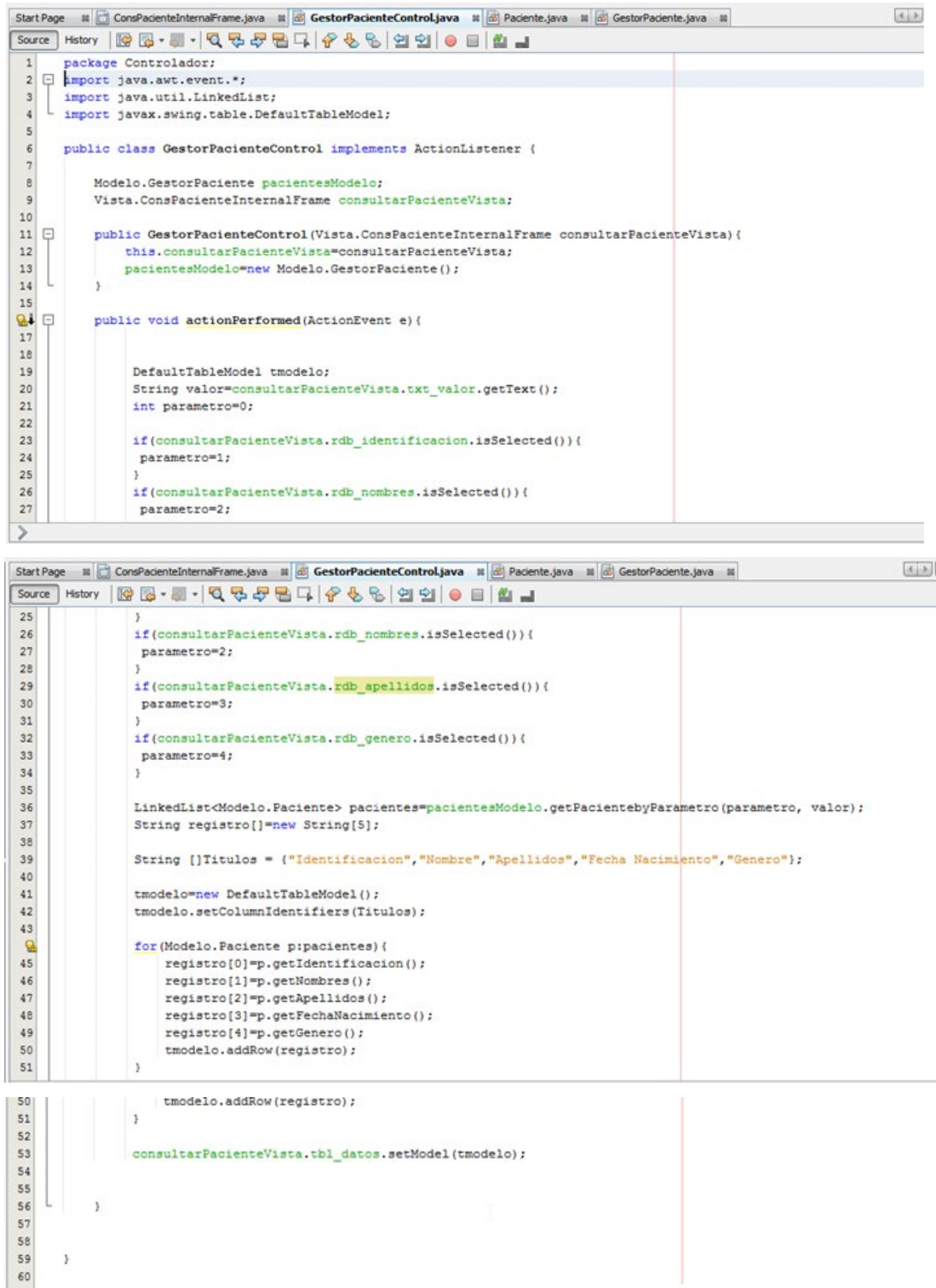
        String []Titulos = {"Identificacion","Nombre","Apellidos","Fecha Nacimiento","Genero"};

        tmodelo=new DefaultTableModel();
        tmodelo.setColumnIdentifiers(Titulos);

        for(Modelo.Paciente p:pacientes){
            registro[0]=p.getIdentificacion();
            registro[1]=p.getNombres();
            registro[2]=p.getApellidos();
            registro[3]=p.getFechaNacimiento();
            registro[4]=p.getGenero();
            tmodelo.addRow(registro);
        }

        consultarPacienteVista.tbl_datos.setModel(tmodelo);
    }
}

```



```

1 package Controlador;
2 import java.awt.event.*;
3 import java.util.LinkedList;
4 import javax.swing.table.DefaultTableModel;
5
6 public class GestorPacienteControl implements ActionListener {
7
8     Modelo.GestorPaciente pacientesModelo;
9     Vista.ConsPacienteInternalFrame consultarPacienteVista;
10
11     public GestorPacienteControl(Vista.ConsPacienteInternalFrame consultarPacienteVista){
12         this.consultarPacienteVista=consultarPacienteVista;
13         pacientesModelo=new Modelo.GestorPaciente();
14     }
15
16     public void actionPerformed(ActionEvent e){
17
18         DefaultTableModel tmodelo;
19         String valor=consultarPacienteVista.txt_valor.getText();
20         int parametro=0;
21
22         if(consultarPacienteVista.rdb_identificacion.isSelected()){
23             parametro=1;
24         }
25
26         if(consultarPacienteVista.rdb_nombres.isSelected()){
27             parametro=2;
28
29             if(consultarPacienteVista.rdb_apellidos.isSelected()){
30                 parametro=3;
31             }
32             if(consultarPacienteVista.rdb_genero.isSelected()){
33                 parametro=4;
34             }
35
36             LinkedList<Modelo.Paciente> pacientes=pacientesModelo.getPacientebyParametro(parametro, valor);
37             String registro[]=new String[5];
38
39             String []Titulos = {"Identificacion","Nombre","Apellidos","Fecha Nacimiento","Genero"};
40
41             tmodelo=new DefaultTableModel();
42             tmodelo.setColumnIdentifiers(Titulos);
43
44             for(Modelo.Paciente p:pacientes){
45                 registro[0]=p.getIdentificacion();
46                 registro[1]=p.getNombres();
47                 registro[2]=p.getApellidos();
48                 registro[3]=p.getFechaNacimiento();
49                 registro[4]=p.getGenero();
50                 tmodelo.addRow(registro);
51             }
52
53             tmodelo.addRow(registro);
54
55             consultarPacienteVista.tbl_datos.setModel(tmodelo);
56         }
57     }
58 }
59
60

```

Figura 69. Código - clase controladora.
Fuente: SENA

Explicación del código (a continuación):

Tabla 10
Explicación del código

Línea	Explicación
21	Declaración e inicialización de la variable que vas a indicar el criterio de búsqueda seleccionado por el usuario(identificación, nombres, apellidos, genero).
41	Actualización del JTable , esto hace que el JTable quede vacío para poder ser cargado posteriormente con los resultados de la consulta.
44	Captura de la opción de búsqueda selecciona por el usuario en el formulario consulta de pacientes, de acuerdo a esa opción se asigna un valor a la variable parámetro.
45	Captura de los pacientes devueltos por el modelo de acuerdo con los criterios de búsqueda proporcionados por el usuario.
45	Recorrido de cada uno de los pacientes retornados por el modelo a partir de la consulta realizada.
53	Asignación de los valores de cada paciente al arreglo que se cargará en el TableModel .
50	Adición de una fila al TableModel cuyo contenido es la información del arreglo.

Nota: Producción SENA

El paquete controlador finalmente queda conformado por dos clases paciente control que recoge la información queda conformado por dos clases paciente control que recoge la información de cada paciente ingresado por el usuario en la vista y lo registra en el modelo y la clase **GestorPaciente** control que se encarga de recibir de la vista y lo registra en el modelo y la clase **GestorPacienteControl** que se encarga de recibir de la vista los parámetros de búsqueda, consultar en el modelo los pacientes que cumplen con esos parámetros y regresar a la vista los resultados obtenidos, la estructura del proyecto debe ser el siguiente:

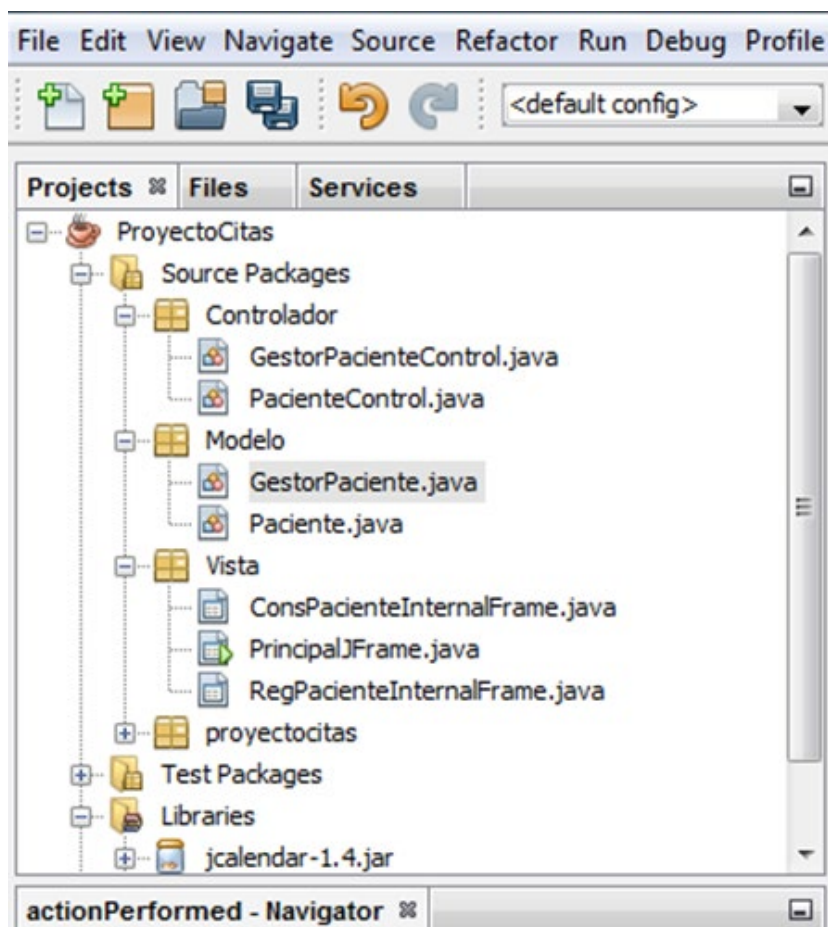


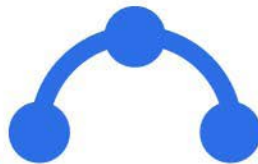
Figura 70. Estructura del proyecto con el MVC.
Fuente: SENA

Desarrollo de aplicaciones DE ESCRITORIO CON LENGUAJE



Sesión 5

Pruebas del
modelo vista
controlador MVC



Durante la sesión 2,3 y 4 se crearon los diferentes componentes que hacen parte del modelo vista controlador (MVC) sesión2: la vista, sesión 3: el modelo, sesión 4; el controlador.

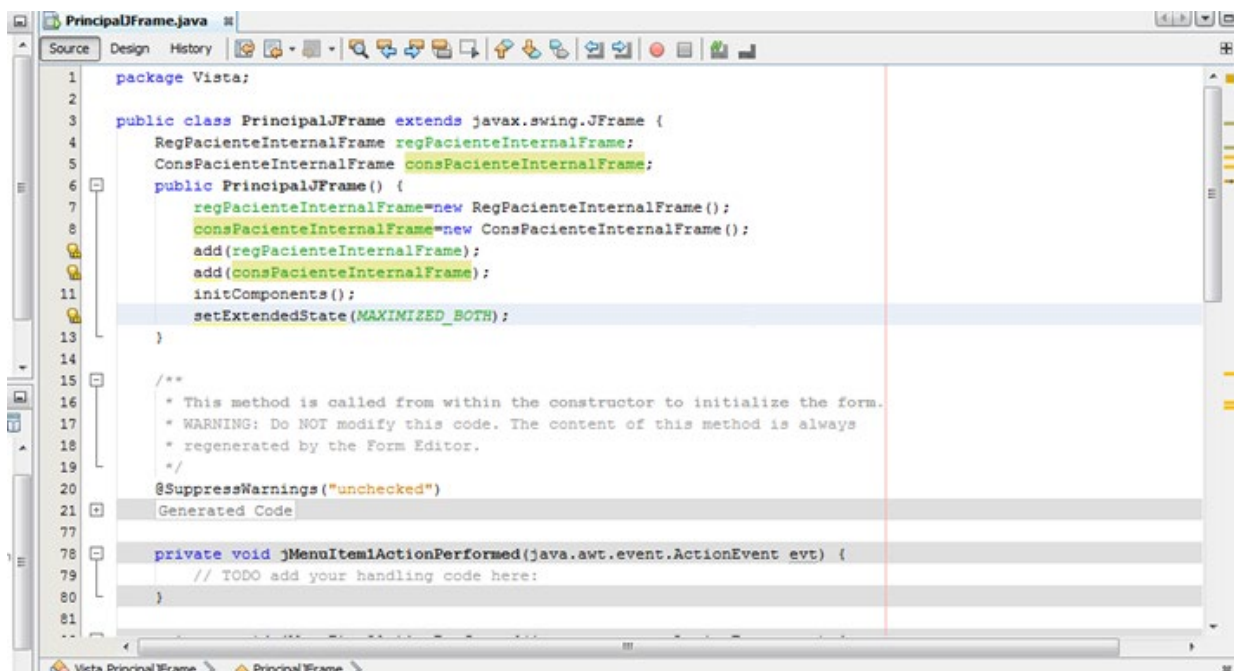
NOTA. El **MVC** es un patrón de diseño que separa la interfaz de usuario de la lógica y los datos.

El desarrollo de este patrón dentro de una aplicación trae como consecuencia el uso de arquitectura multicapa, ya que la capa de presentación está abordada por la vista, la capa de la lógica es responsabilidad del controlador y la capa de datos está a cargo del modelo.

Las ventajas de una arquitectura multicapa son la separación de las responsabilidades y la facilidad de mantenimiento y adaptación de las aplicaciones.

En esta sesión se van a realizar los ajustes finales de la vista para vincularla con el controlador y así poder ejecutar la aplicación. Inicialmente se va a adicionar los formularios internos (registro de pacientes y consulta de pacientes) al formulario principal.

En el código del formulario principal se incluyen las siguientes líneas:



```

1 package Vista;
2
3 public class PrincipalJFrame extends javax.swing.JFrame {
4     RegPacienteInternalFrame regPacienteInternalFrame;
5     ConsPacienteInternalFrame consPacienteInternalFrame;
6     public PrincipalJFrame() {
7         regPacienteInternalFrame=new RegPacienteInternalFrame();
8         consPacienteInternalFrame=new ConsPacienteInternalFrame();
9         add(regPacienteInternalFrame);
10        add(consPacienteInternalFrame);
11        initComponents();
12        setExtendedState(MAXIMIZED_BOTH);
13    }
14
15    /**
16     * This method is called from within the constructor to initialize the form.
17     * WARNING: Do NOT modify this code. The content of this method is always
18     * regenerated by the Form Editor.
19     */
20    @SuppressWarnings("unchecked")
21    Generated Code
22
23    private void jMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
24        // TODO add your handling code here:
25    }
26
27 }

```

Figura 71. Programando el constructor clase principal.

Fuente: SENA

Explicación del código

Tabla 11
Explicación del código

Línea	Explicación
4,5	Declaración de objetos asociados a los formularios internos.
8,9	Instanciación de los objetos asociados a los formularios internos.
10,11	Adición de los formularios internos al formulario principal.
13	Instrucción para que el formulario principal se presente maximizado.

Nota: Producción SENA

Ahora se escribe el código necesario para que al hacer clic en los menús del formulario principal se presenten los formularios internos correspondientes.

Abrir el formulario principal en vista de diseño hacer clic en el menú Archivo y sobre esta hacer clic derecho y seguir el menú **Events**→**actionactionperformed**.

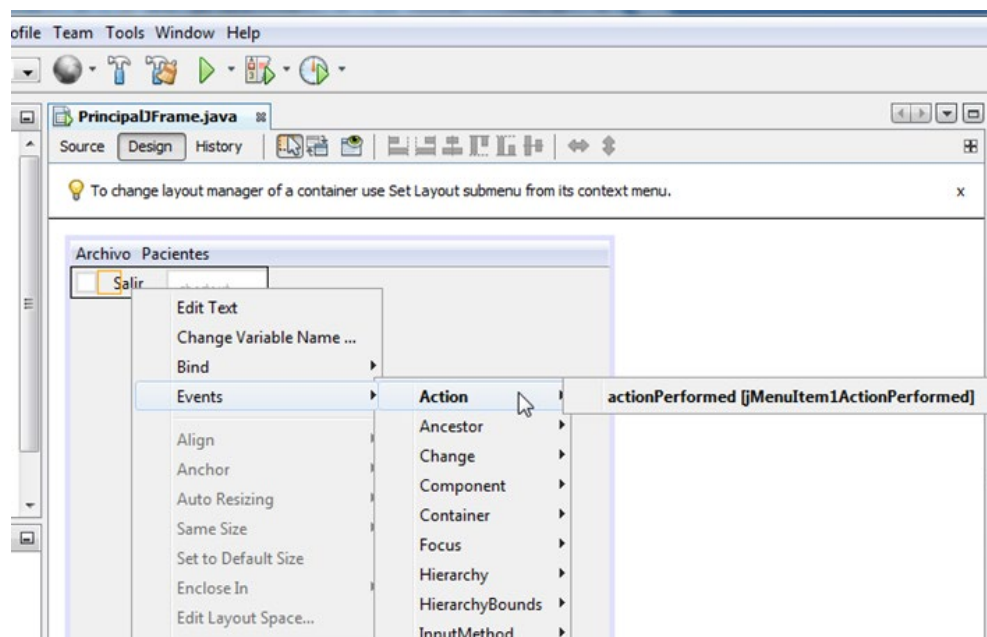


Figura 72. Programando el action performed.
Fuente: SENA

De esa manera se puede ubicar en el método que se ejecuta al hacer clic sobre el menú salir, en ese espacio ingresar el código para salir de una aplicación en JAVA.

```

77
78 private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt) {
79     // TODO add your handling code here:
80     System.exit(0);
81
82 }

```

Figura 73. Programando la salida del sistema.
Fuente: SENA

Repetir el proceso para la opción del menú registrar.

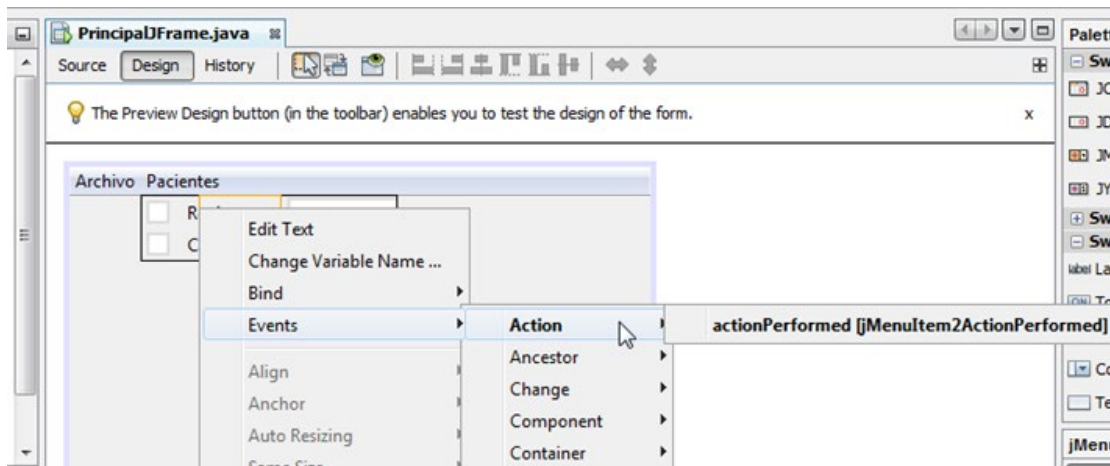


Figura 74. Programando el action performed.
Fuente: SENA

```

private void jMenuItem3ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    regPacienteInternalFrame.setVisible(true);
}

```

Figura 75. Programando el Action Performed.
Fuente: SENA

Se repite el proceso para opción de menú consultar.
Ingresar el código para hacer visible el formulario de consulta de pacientes.

```

84 private void jMenuItem2ActionPerformed(java.awt.event.ActionEvent evt) {
85     // TODO add code here:
86     consPacienteInternalFrame.setVisible(true);
87 }
88
89 /**

```

Figura 76. Clase JProgramando el Action Performed para la consulta de pacientes.
Fuente: SENA

El código del formulario principal debe tener un aspecto semejante al siguiente:

```

public class PrincipalJFrame extends javax.swing.JFrame {
    RegPacienteInternalFrame regPacienteInternalFrame;
    ConsPacienteInternalFrame consPacienteInternalFrame;
    public PrincipalJFrame() {
        regPacienteInternalFrame=new RegPacienteInternalFrame();
        consPacienteInternalFrame=new ConsPacienteInternalFrame();
        add(regPacienteInternalFrame);
        add(consPacienteInternalFrame);
        initComponents();
        setExtendedState(MAXIMIZED_BOTH);
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jMenuBar1 = new javax.swing.JMenuBar();
        jMenu1 = new javax.swing.JMenu();
        jMenuItem1 = new javax.swing.JMenuItem();
        jMenu2 = new javax.swing.JMenu();
        jMenuItem2 = new javax.swing.JMenuItem();
        jMenuItem3 = new javax.swing.JMenuItem();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("GESTION DE CITAS");
        setName("PrincipalJFrame"); // NOI18N

        jMenu1.setText("Archivo");

        jMenuItem1.setText("Salir");
        jMenuItem1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItem1ActionPerformed(evt);
            }
        });
        jMenu1.add(jMenuItem1);

        jMenuBar1.add(jMenu1);

        jMenu2.setText("Pacientes");

        jMenuItem2.setText("Registrar");
        jMenuItem2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItem2ActionPerformed(evt);
            }
        });
    }
}

```

```

    }
    });
    jMenu2.add(jMenuItem2);

    jMenuItem3.setText("Consultar");
    jMenuItem3.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jMenuItem3ActionPerformed(evt);
        }
    });
    jMenu2.add(jMenuItem3);

    jMenuBar1.add(jMenu2);

    setJMenuBar(jMenuBar1);

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 400, Short.MAX_VALUE)
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 279, Short.MAX_VALUE)
    );

    pack();
} // </editor-fold>

private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    System.exit(0);
}

private void jMenuItem2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    consPacienteInternalFrame.setVisible(true);
}

private void jMenuItem3ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    regPacienteInternalFrame.setVisible(true);
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {

```

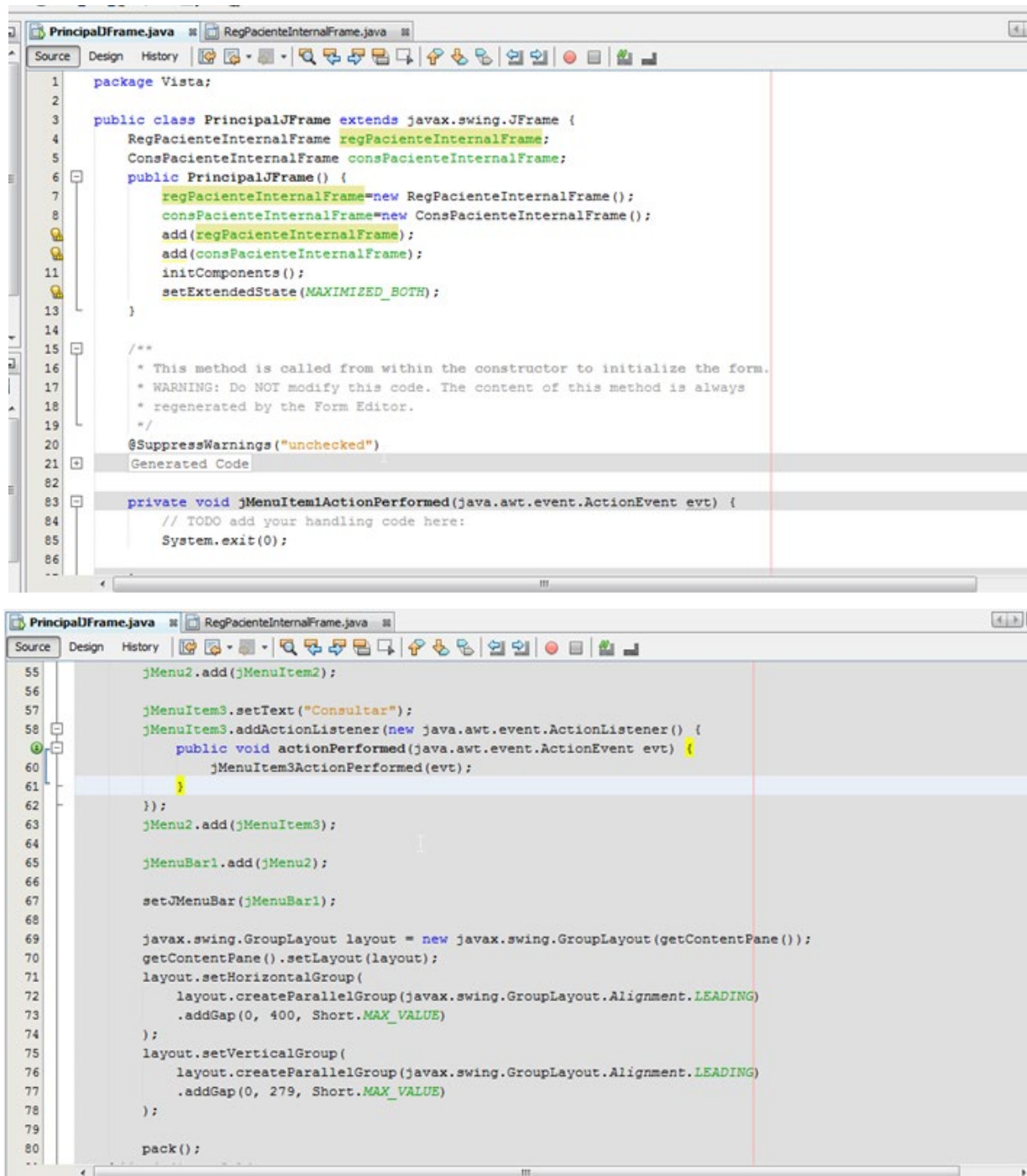
```

/* Set the Nimbus look and feel */
//<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
/* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and
feel.
* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.
html
*/
try {
    for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.
getInstalledLookAndFeels()) {
        if ("Nimbus".equals(info.getName())) {
            javax.swing.UIManager.setLookAndFeel(info.getClassName());
            break;
        }
    }
} catch (ClassNotFoundException ex) {
    java.util.logging.Logger.getLogger(PrincipalJFrame.class.getName()).log(java.util.
logging.Level.SEVERE, null, ex);
} catch (InstantiationException ex) {
    java.util.logging.Logger.getLogger(PrincipalJFrame.class.getName()).log(java.util.
logging.Level.SEVERE, null, ex);
} catch (IllegalAccessException ex) {
    java.util.logging.Logger.getLogger(PrincipalJFrame.class.getName()).log(java.util.
logging.Level.SEVERE, null, ex);
} catch (javax.swing.UnsupportedLookAndFeelException ex) {
    java.util.logging.Logger.getLogger(PrincipalJFrame.class.getName()).log(java.util.
logging.Level.SEVERE, null, ex);
}
//</editor-fold>

/* Create and display the form */
java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
        new PrincipalJFrame().setVisible(true);
    }
});
}

// Variables declaration - do not modify
private javax.swing.JMenu jMenuItem1;
private javax.swing.JMenu jMenuItem2;
private javax.swing.JMenuBar jMenuItemBar1;
private javax.swing.JMenuItem jMenuItem1;
private javax.swing.JMenuItem jMenuItem2;
private javax.swing.JMenuItem jMenuItem3;
// End of variables declaration
}

```



```

package Vista;

public class PrincipalJFrame extends javax.swing.JFrame {
    RegPacienteInternalFrame regPacienteInternalFrame;
    ConsPacienteInternalFrame consPacienteInternalFrame;
    public PrincipalJFrame() {
        regPacienteInternalFrame=new RegPacienteInternalFrame();
        consPacienteInternalFrame=new ConsPacienteInternalFrame();
        add(regPacienteInternalFrame);
        add(consPacienteInternalFrame);
        initComponents();
        setExtendedState(MAXIMIZED_BOTH);
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code

    private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        System.exit(0);
    }
}

```

```

jMenuItem2.add(jMenuItem2);

jMenuItem3.setText("Consultar");
jMenuItem3.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jMenuItem3ActionPerformed(evt);
    }
});
jMenuItem2.add(jMenuItem3);
jMenuBar1.add(jMenuItem2);

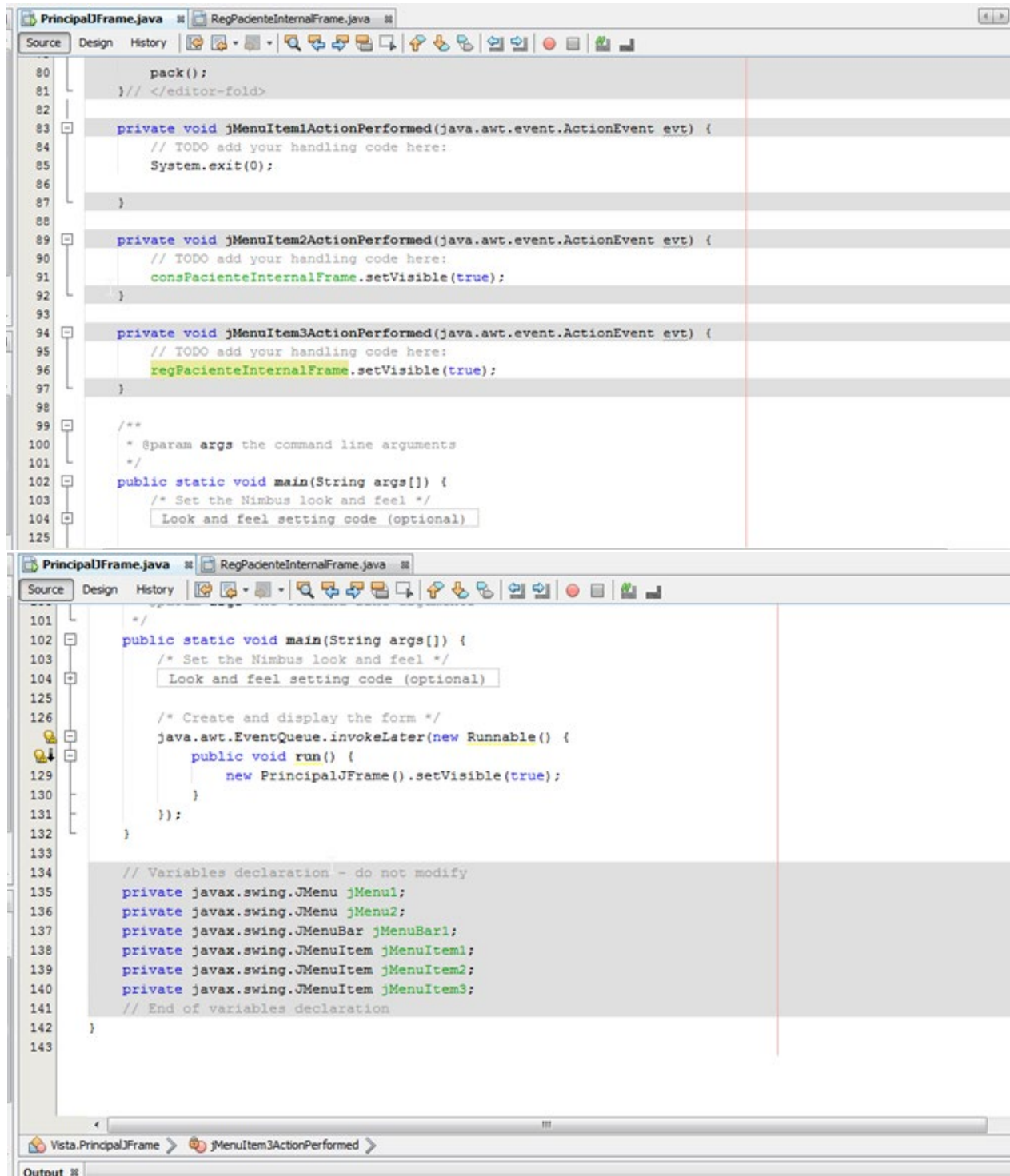
setJMenuBar(jMenuBar1);

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGap(0, 400, Short.MAX_VALUE)
);
layout.setVerticalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGap(0, 279, Short.MAX_VALUE)
);

pack();

```

Figura 77. Código del formulario principal - Parte 1
Fuente: SENA



```

PrincipalJFrame.java
RegPacienteInternalFrame.java

Source Design History
80 pack();
81 } // </editor-fold>
82
83 private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt) {
84     // TODO add your handling code here:
85     System.exit(0);
86 }
87
88
89 private void jMenuItem2ActionPerformed(java.awt.event.ActionEvent evt) {
90     // TODO add your handling code here:
91     consPacienteInternalFrame.setVisible(true);
92 }
93
94 private void jMenuItem3ActionPerformed(java.awt.event.ActionEvent evt) {
95     // TODO add your handling code here:
96     regPacienteInternalFrame.setVisible(true);
97 }
98
99 /**
100  * @param args the command line arguments
101  */
102 public static void main(String args[]) {
103     /* Set the Nimbus look and feel */
104     Look and feel setting code (optional)
105 }
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134 // Variables declaration - do not modify
135 private javax.swing.JMenu jMenuItem1;
136 private javax.swing.JMenu jMenuItem2;
137 private javax.swing.JMenuBar jMenuItem3;
138 private javax.swing.JMenuItem jMenuItem4;
139 private javax.swing.JMenuItem jMenuItem5;
140 private javax.swing.JMenuItem jMenuItem6;
141 // End of variables declaration
142 }
143

```

Figura 78. Código del formulario principal - Parte 2
Fuente: SENA

El código del formulario **Registro de Pacientes** debe registrar a la clase controladora que atenderá los eventos que en él se generen. A continuación se presenta el código que debe agregarse:

```

12 public class RegPacienteInternalFrame extends javax.swing.JInternalFrame {
13
14     /**
15      * Creates new form RegPacienteInternalFrame
16      */
17     private Controlador.PacienteControl pacienteControlador;
18     public RegPacienteInternalFrame() {
19         initComponents();
20         pacienteControlador=new Controlador.PacienteControl(this);
21         btn_registrar.addActionListener(pacienteControlador);
22         btn_nuevo.addActionListener(pacienteControlador);
23     }
24
25     /**
26      * This method is called from within the constructor to initialize the form.
27      * WARNING: Do NOT modify this code. The content of this method is always
28      * regenerated by the Form Editor.
29      */
30     @SuppressWarnings("unchecked")

```

Figura 79. Programando el registro del paciente.

Fuente:SENA

Explicación del código.

El código del formulario **Consulta de Pacientes** de igual manera debe registrar a la clase que la entenderá sus eventos, además de incluir el código de configuración inicial del **JTable** donde se presentarán los datos de las consultas realizadas. A continuación, se presenta el código que debe agregarse:

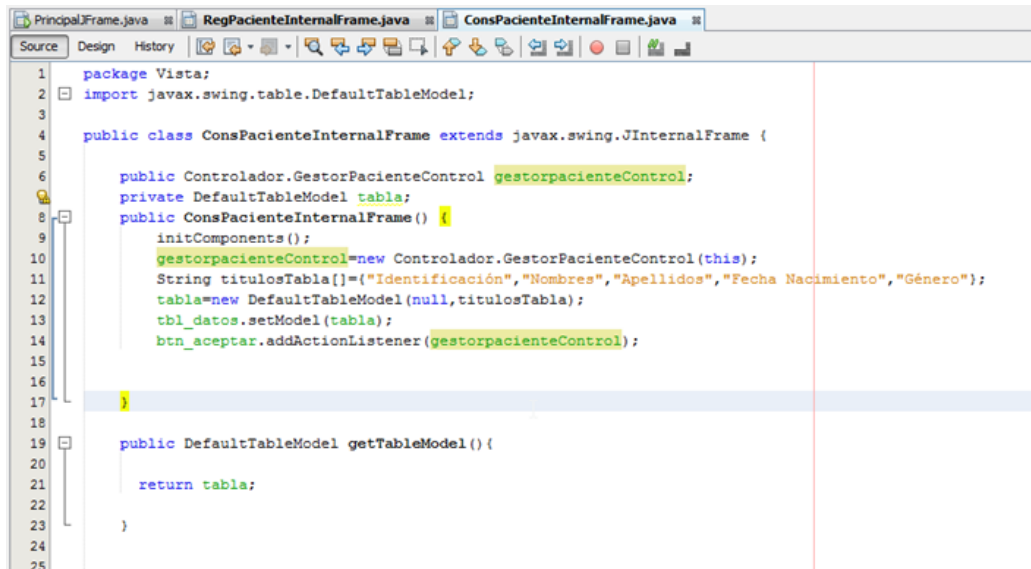
Tabla 12

Explicación del código

Línea	Explicación
5	Declaración del objeto que controla el formulario.
10	Instanciación del objeto que controla el formulario y gestiona sus eventos.
11,12	Indica que el objeto pacienteControlador es el encargado de gestionar los eventos que se generen en los botones Registrar y Nuevo .

Nota: Producción SENA

El código del formulario **Consulta de Pacientes** de igual manera debe registrar a la clase que le atenderá sus eventos, además de incluir el código de configuración inicial del **JTable** donde se presentarán los datos de las consultas realizadas. A continuación, se presenta el código que debe agregarse:



```

1 package Vista;
2 import javax.swing.table.DefaultTableModel;
3
4 public class ConsPacienteInternalFrame extends javax.swing.JInternalFrame {
5
6     public Controlador.GestorPacienteControl gestorpacienteControl;
7     private DefaultTableModel tabla;
8     public ConsPacienteInternalFrame() {
9         initComponents();
10        gestorpacienteControl=new Controlador.GestorPacienteControl(this);
11        String titulosTabla[]={"Identificación","Nombres","Apellidos","Fecha Nacimiento","Género"};
12        tabla=new DefaultTableModel(null,titulosTabla);
13        tbl_datos.setModel(tabla);
14        btn_aceptar.addActionListener(gestorpacienteControl);
15
16    }
17
18
19    public DefaultTableModel getTableModel() {
20
21        return tabla;
22    }
23
24
25

```

Figura 80. Programando el JTable.
Fuente: SENA

NOTA. **Jtable** representa un elemento gráfico en forma de cuadrícula comúnmente usado para presentar registros de datos.

Este objeto gráfico no es manipulado directamente sino a través de un objeto no gráfico llamado **TableModel**, el **TableModel** es el que define los títulos y contenido del **Jtable**, así mismo operaciones como insertar filas, acceder a un determinado dato de la tabla o eliminar los datos de la tabla, se realizarán a través del **TableModel** que al estar asociado al objeto **Jtable**, refleja visualmente estas modificaciones en el mismo.

Explicación del código .

Tabla 13
Explicación del código.

Línea	Explicación
3	Importación de la clase DefaultTableModel que se emplea para establecer la configuración del objeto JTable .
7	Declaración del objeto que controla el formulario Consulta de Pacientes .
8	Declaración del objeto que configura y gestiona la información del JTable .
13	Instanciación del objeto que controla el formulario y gestiona sus eventos
14	Creación del arreglo que contiene los títulos del JTable donde se presentan los resultados de las consultas.
15	Instanciación del TableModel con los títulos requeridos.
16	Asignación del TableModel al JTable .

17	Indica que el objeto gestorpacienteControl es el encargado de gestionar los eventos que se generen en el botón Aceptar del formulario Consulta de Pacientes .
19..22	Método que retorna el objeto TableModel asociado al JTable para ser manipulado desde la clase controladora.

Nota: Producción SENA

Finalmente, después de establecer desde la vista los objetos controladores que se encargan de gestionar los eventos sólo queda faltando ejecutar la aplicación se hace clic derecho sobre el proyecto **ProyectoCitas** y se selecciona del menú contextual la opción **Clean and Build**.

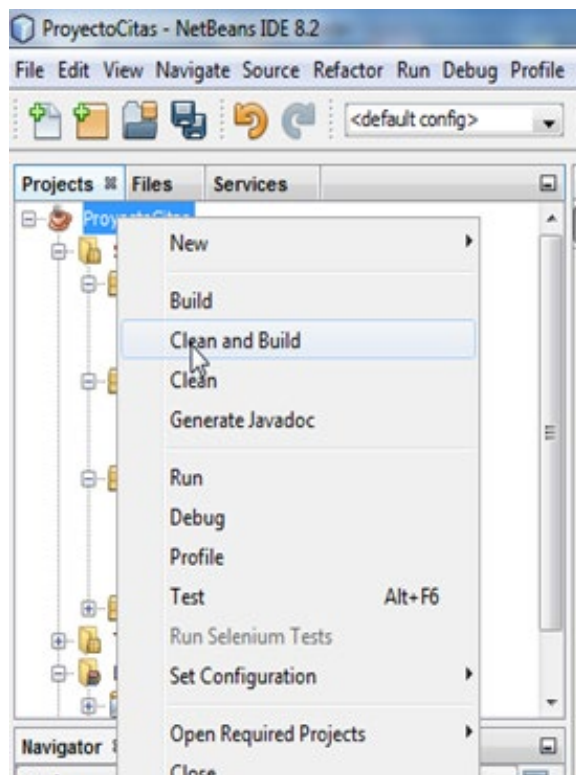


Figura 81. Compilando el código.

Fuente: SENA

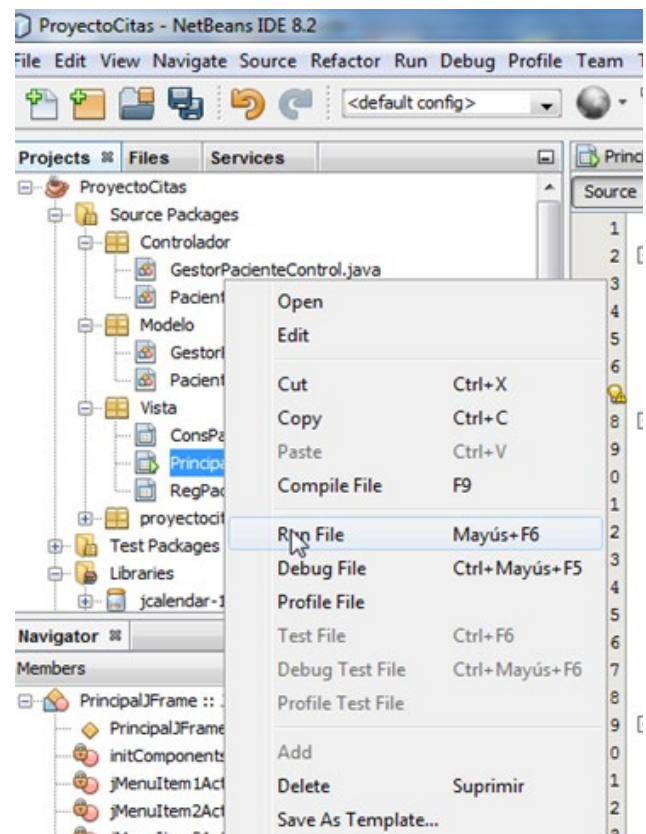


Figura 82. Ejecutando el programa.

Fuente: SENA

Por último, se hace clic derecho sobre la clase **PrincipalFrame** y se selecciona del menú contextual la opción **Runfile**.

Desarrollo de aplicaciones DE ESCRITORIO CON LENGUAJE



Sesión 6

Acceso a la base
de datos con
JDBC



En el objeto de aprendizaje introducción a la bases de datos Oracle la sesión se construyó la base de datos en Oracle, a partir de la sesión 2 de esta guía se desarrolló la aplicación basada en el patrón **MVC Modelo Vista Controlador**, sin embargo los datos ingresados en el formulario de registro de pacientes se pierden cuando la aplicación se cierra, esto se da porque los datos de los pacientes registrados se están almacenando en variables y listas pero no en un medio persistente como la base de datos.

Durante la presente sesión se conectará la aplicación que se desarrolla con la base de datos para que la información de los pacientes registrados no se pierda, aunque se cierre la aplicación incluso aunque se apague el equipo.

Se utilizará para la conexión y gestión de la base de datos el **API JDBC** de Java.

NOTA. **JDBC Java DataBase Connectivity** es un estándar usado por JAVA para el acceso y la manipulación de datos de bases de datos relacionales a través de clases e interfaces, permitiendo embeber código **SQL** dentro del código JAVA.

Descargar el driver JDBC para Oracle ojdbc7.jar de la siguiente dirección:
<http://www.oracle.com/technetwork/database/features/jdbc/default-2280470.html>

El archivo **ojdbc7.jar** puede ser descargado en cualquier ruta.

NOTA. Es importante que el aprendiz se registre en la plataforma de **Oracle** para descargar cualquier componente que se requiera, en la opción crear una cuenta.

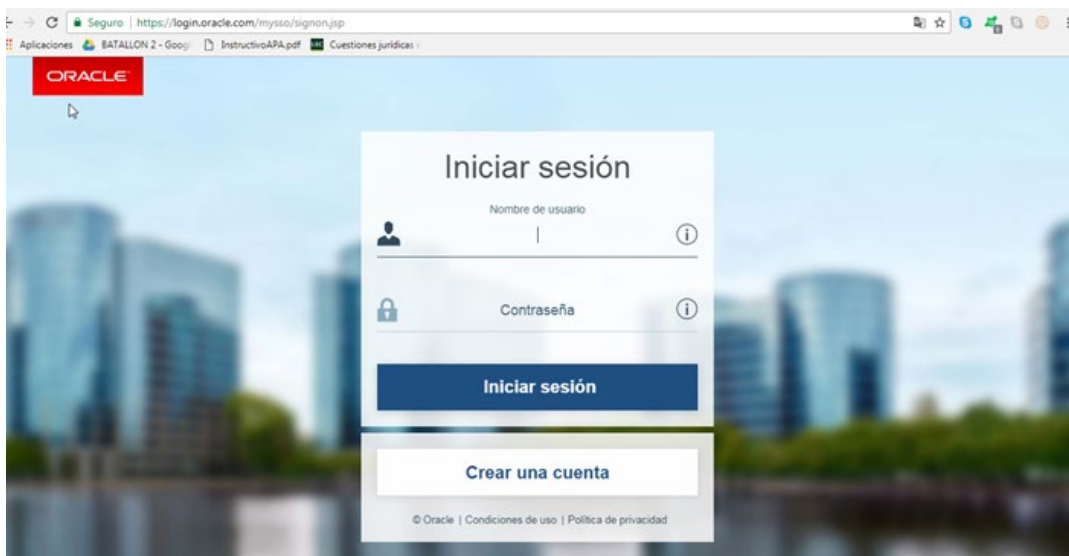


Figura 83. Descargando el JDBC driver jdbc Oracle.

Fuente: SENA

- **Conexión con la base de datos desde NETBEANS**

En **Netbeans**, hacer clic en el menú **Window**→ **Services**, aparecerá la siguiente ventana.

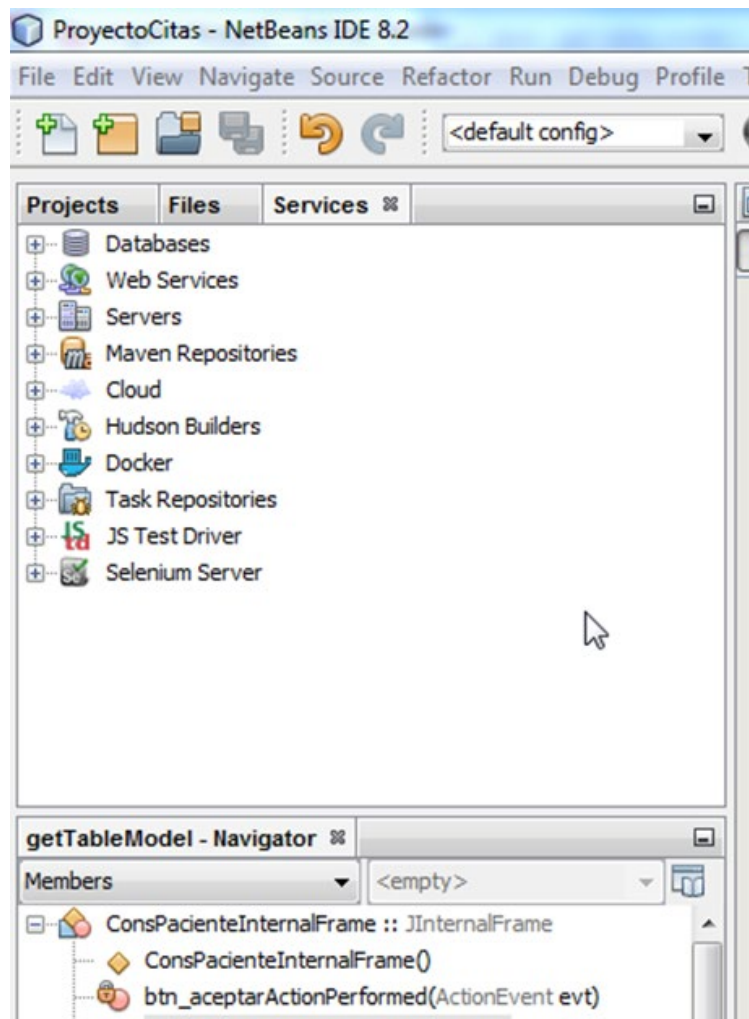


Figura 84. Servicios disponibles en NETBEANS.
Fuente: SENA

Hacer clic derecho sobre la opción **Databases** y en ella seleccionar del menú contextual la opción **New Connection**.

En la siguiente ventana, seleccionar el driver **Oracle Thin**.

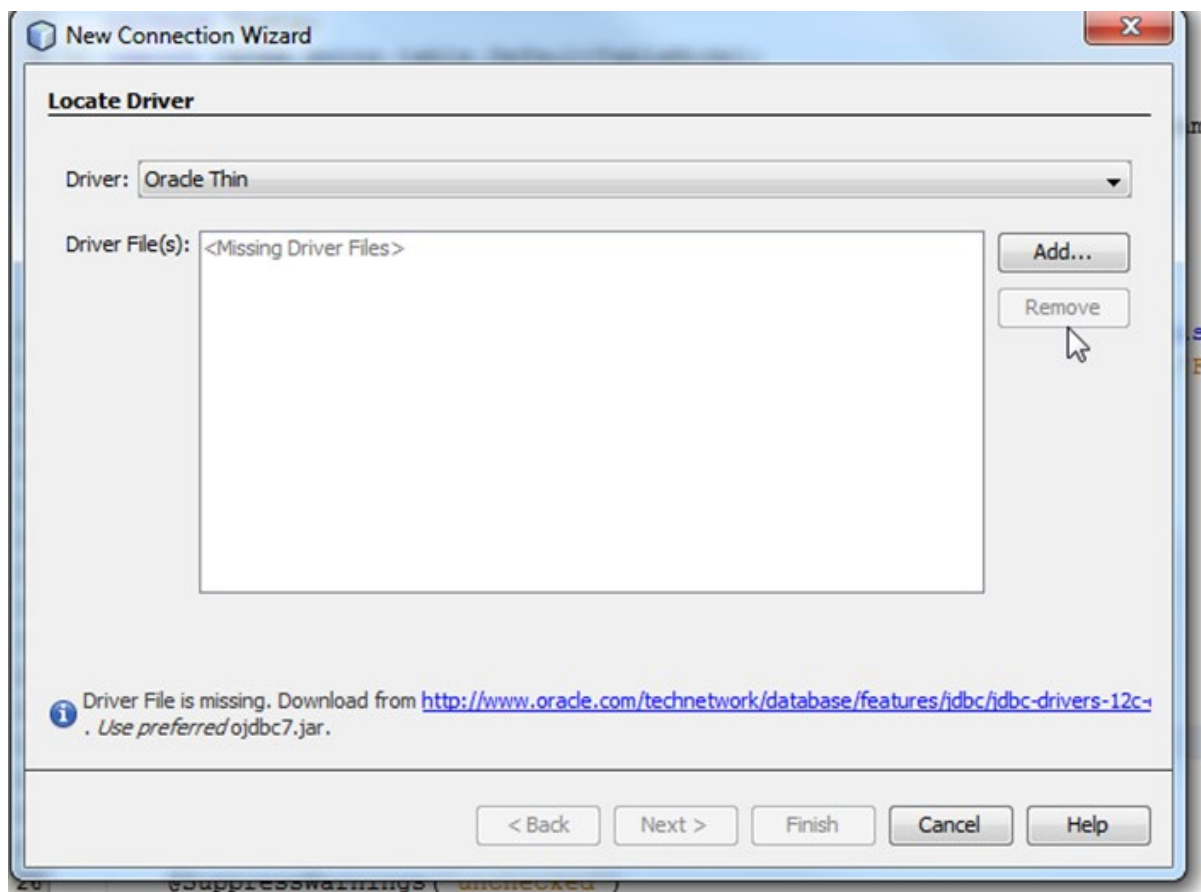


Figura 85. Conectándose con la opción de Oracle Thin.
Fuente: SENA

Hacer clic en el botón **add**, buscar y seleccionar el driver **ojdbc6.jar** previamente descargado y hacer clic en Abrir, luego hacer clic en **Next**.

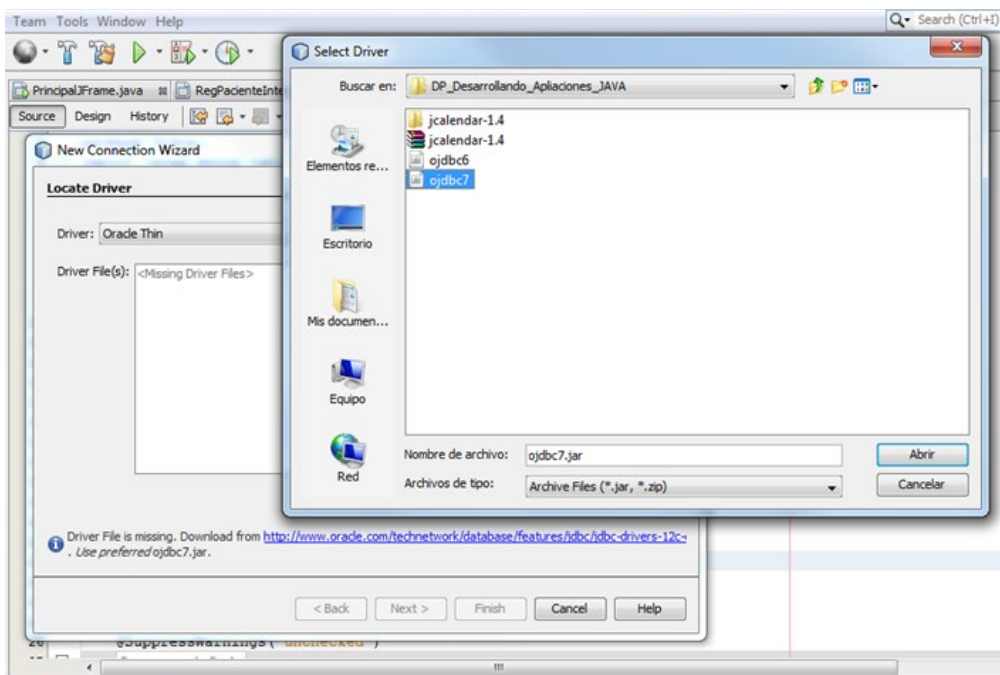


Figura 86. Seleccionando el driver de Oracle.

Fuente: SENA

En la siguiente ventana, ingresar los siguientes valores de configuración.

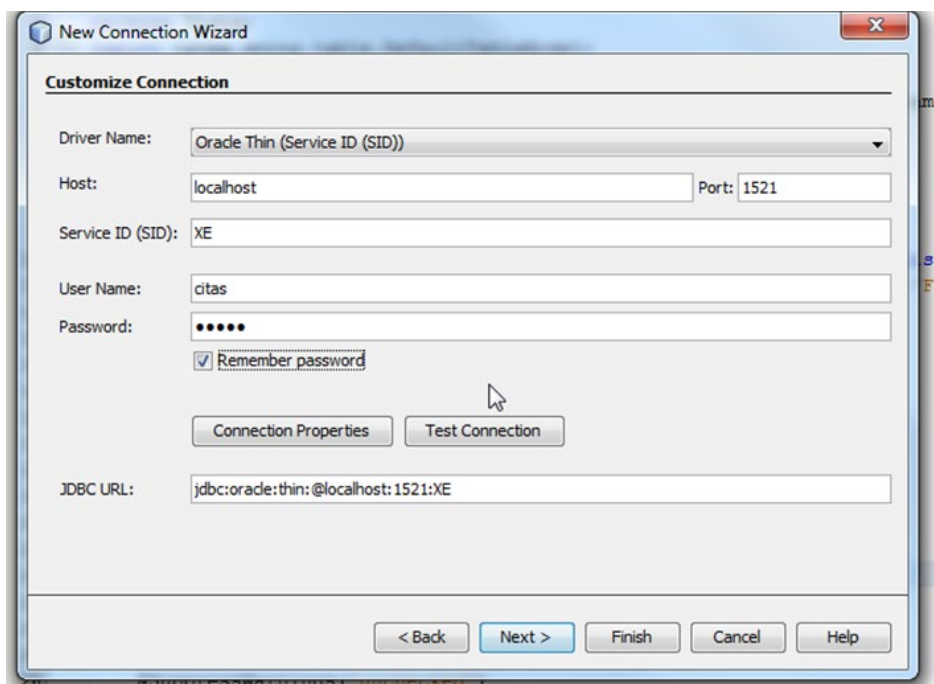


Figura 87. Configurando la conexión.

Fuente: SENA

NOTA. Ingresar el username: citas y el password: citas, los cuales son los datos de conexión con la base de datos creados en el material de estudio “Introducción a las bases de datos de ORACLE”; si los datos de conexión son diferentes, ingresar los propios.

Para comprobar la conexión hacer clic en el botón **test connection** y verificar que aparece el mensaje **connection succeeded**, hacer clic en **Next**.

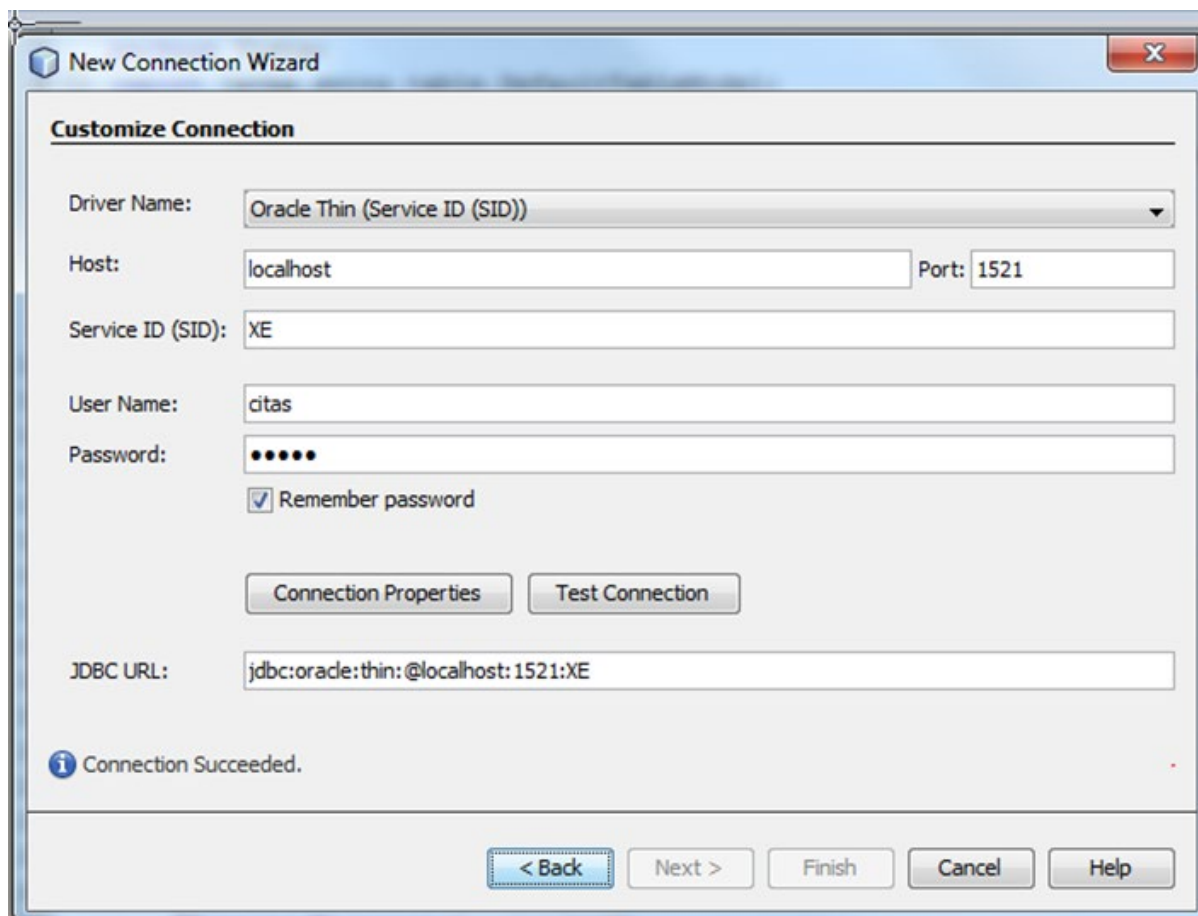


Figura 88. Probando la conexión con la base de datos.
Fuente: SENA

Se selecciona el esquema citas y se hace clic en el botón Finish.

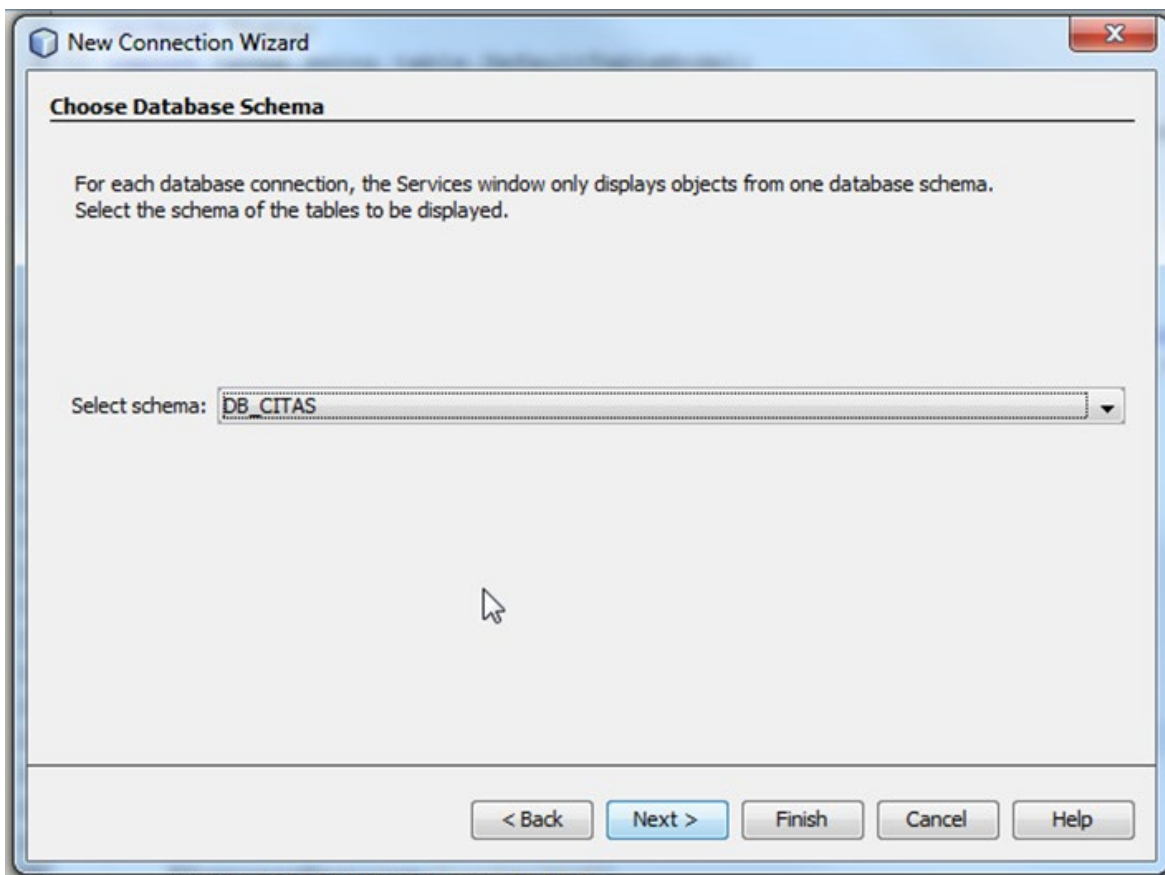


Figura 89. Seleccionando el esquema de la base de datos creada.
Fuente: SENA

Al abrir la conexión recién configurada y el esquema citas, se puede observar que desde nuestra aplicación JAVA en NETBEANS se tiene acceso a las tablas del esquema DB_CITAS configurado en el OA introducción a las bases de datos en ORACLE.

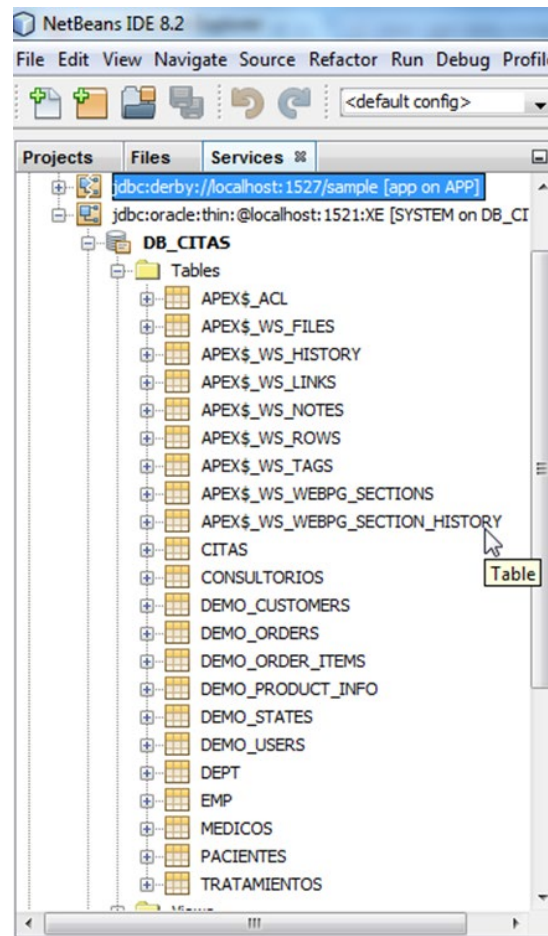


Figura 90. Vista de la base de datos desde NETBEANS.
Fuente: SENA

Lo anterior permite comprobar que efectivamente la aplicación que se está desarrollando en JAVA tiene conexión con la base de datos ORACLE y reconoce sus diferentes objetos del esquema.

Cada operación o transacción que se desee realizar sobre una base de datos desde una aplicación externa, requiere establecer una conexión, en el caso de java, esta conexión, está configurada a través de un objeto de la clase connection que forma parte del paquete java.sql.

Debido a que la creación de conexiones es una actividad recurrente cuando se trabaja con acceso a bases de datos, en el presente proyecto se creará una clase que se encargue de configurar, crear y retornar la conexión con la base de datos, esta clase encapsula toda la lógica de la conexión y solo requiere recibir unos parámetros para crear la conexión.

Crear un nuevo paquete dentro del proyecto ProyectoCitas llamado recurso y sobre este nuevo paquete, crear un java class con el nombre de conexión.

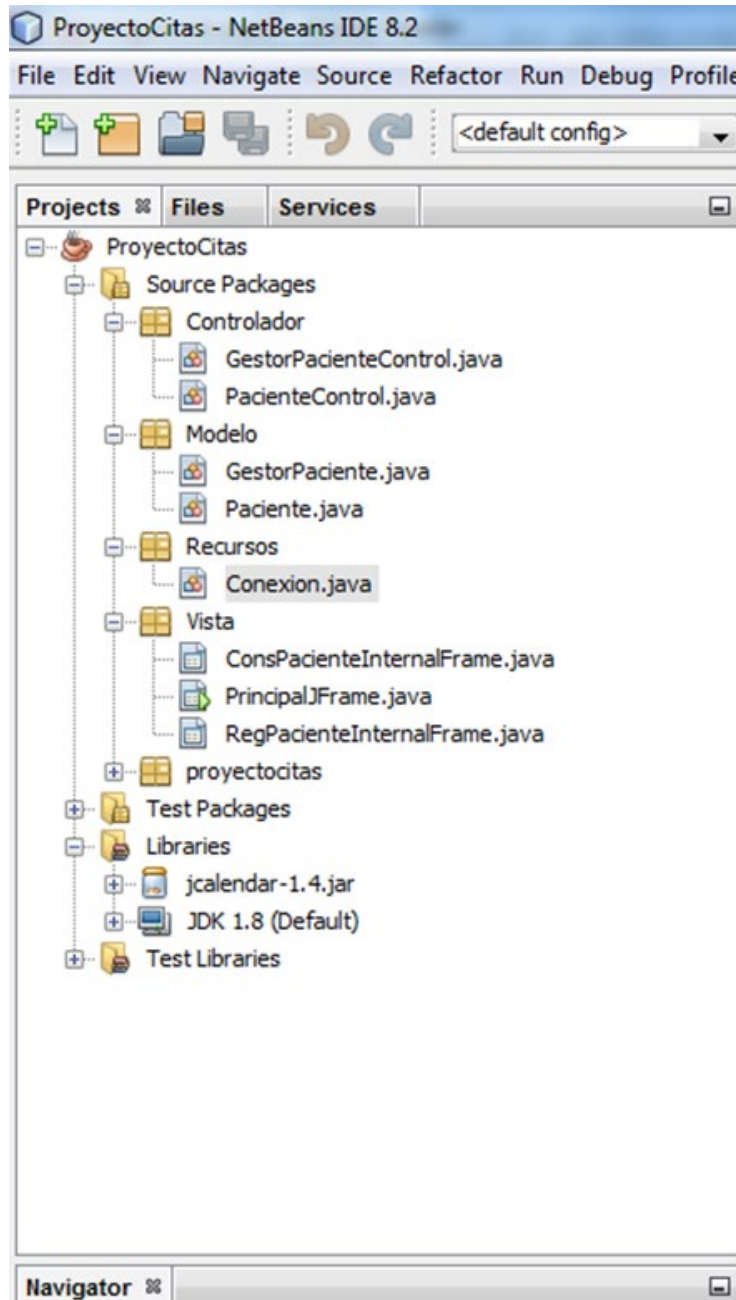


Figura 91. Creando la clase conexión.
Fuente: SENA

El código de la clase de conexión es el siguiente:

```
package Recursos;
import java.sql.*;
import javax.swing.JOptionPane;

public class Conexion {

    private String driver,url, ip, bd, usr,pass;
    public Connection conexion;

    public Conexion(String ip, String bd, String usr,String pass){

        driver="oracle.jdbc.driver.OracleDriver";
        this.bd=bd;
        this.usr=usr;
        this.pass=pass;
        url="jdbc:oracle:thin:@"+ip+":1521:"+bd;
        try{

            Class.forName(driver).newInstance();
            conexion=DriverManager.getConnection(url,usr,pass);

        }catch(Exception ex){

            JOptionPane.showMessageDialog(null,"Error de conexion a la base de datos ");

        }

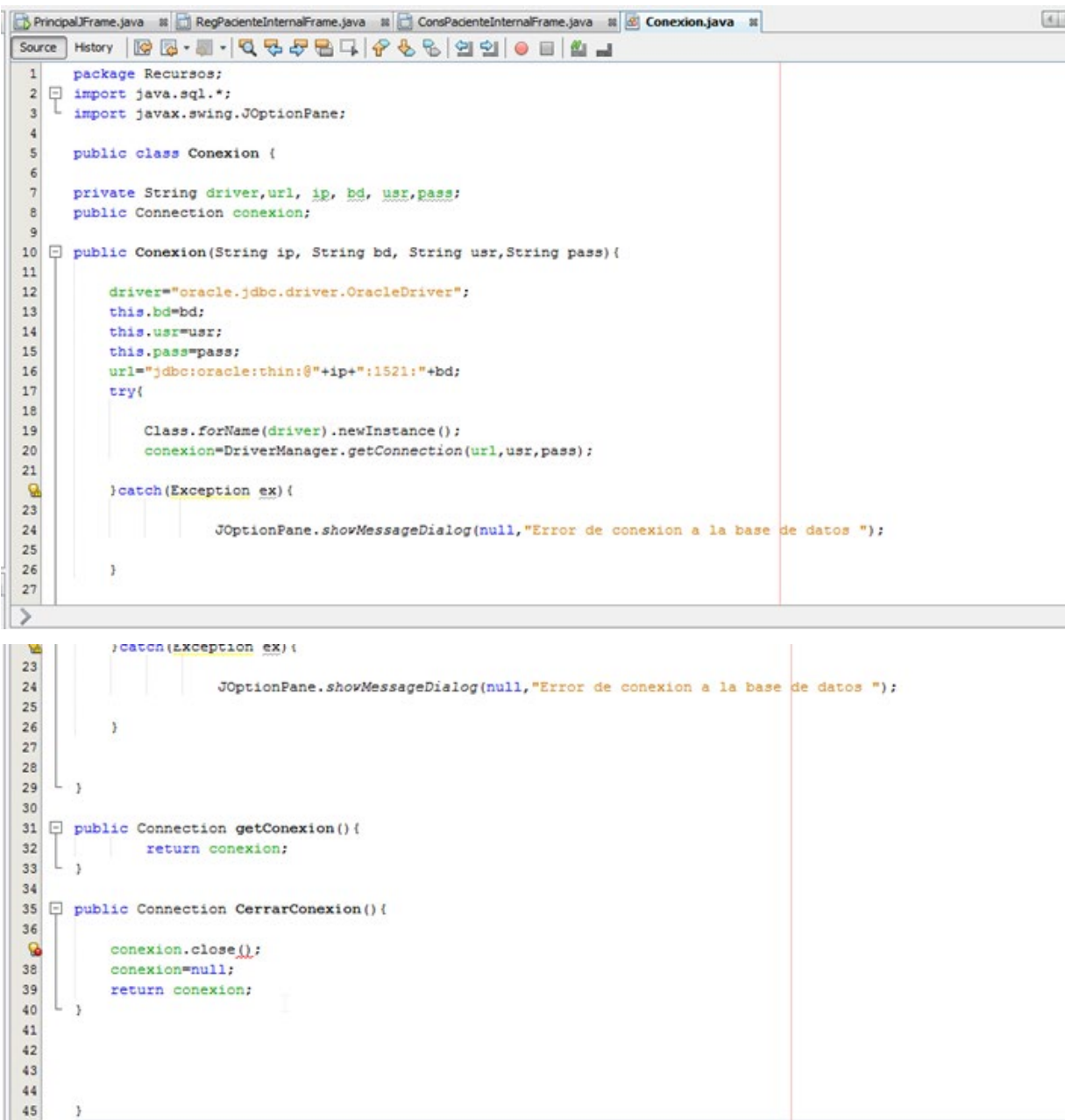
    }

    public Connection getConexion(){
        return conexion;
    }

    public Connection CerrarConexion(){

        conexion.close();
        conexion=null;
        return conexion;
    }

}
```



```

1  package Recursos;
2  import java.sql.*;
3  import javax.swing.JOptionPane;
4
5  public class Conexion {
6
7      private String driver,url, ip, bd, usr,pass;
8      public Connection conexion;
9
10     public Conexion(String ip, String bd, String usr,String pass){
11
12         driver="oracle.jdbc.driver.OracleDriver";
13         this.bd=bd;
14         this.usr=usr;
15         this.pass=pass;
16         url="jdbc:oracle:thin:@"+ip+":1521:"+bd;
17         try{
18
19             Class.forName(driver).newInstance();
20             conexion=DriverManager.getConnection(url,usr,pass);
21
22         }catch(Exception ex){
23
24             JOptionPane.showMessageDialog(null,"Error de conexion a la base de datos ");
25
26         }
27     }
28
29
30
31     public Connection getConexion(){
32         return conexion;
33     }
34
35     public Connection CerrarConexion(){
36
37         conexion.close();
38         conexion=null;
39         return conexion;
40     }
41
42
43
44
45 }

```

Figura 92. Código clase conexión
Fuente: SENA

Explicación del código

Tabla 14

Explicación del código.

Línea	Explicación
3	Importación del paquete sql para hacer uso de las clases que permiten hacer la conexión con la base de datos.
4	Importación de la clase JOptionPane empleada para presentar un mensaje al usuario cuando la conexión es incorrecta .
8	Atributos requeridos para hacer la conexión con la base de datos.
13..17	Se reciben los parámetros ip , que indica la dirección ip donde reside el servidor de bases de datos, si es local, se define como “ localhost ”, los parámetros usr y pass se refieren a el usuario y password de acceso al esquema, driver y url son cadenas de conexión usadas de manera particular para conexiones con bases de datos Oracle.
20	Registro del driver en la aplicación.
21	Creación de la conexión con la base de datos.
28..31	Retorna la conexión establecida.
32..37	Cierre de la conexión creada.

Nota: Producción SENA

NOTA. Una de las principales bondades del modelo vista controlador MVC es la independencia entre sus componentes, esta independencia se evidenciará a continuación, haciendo un cambio en el modelo dentro de la clase **gestorpaciente**, el cual emplea una lista de tipo **linkedlist** para almacenar los pacientes registrados. Esta clase será modificada para que el almacenamiento de los pacientes ya no sea una lista sino la base de datos. A pesar de cambiar el modelo para que responda a esta nueva necesidad, el controlador y la vista permanecerán intactos.

Abrir la clase de gestor paciente y realizar los cambios en el código que se refleja a continuación:

```
package Modelo;
import java.util.*;
import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;
```

```
public class GestorPaciente {
    private static LinkedList<Paciente> pacientes;
```

```

private static Connection conn;
public GestorPaciente(){
    Recursos.Conexion conexion=new Recursos.
Conexion("localhost","XE","SYSTEM","123");
    conn=conexion.getConexion();
    //pacientes=new LinkedList<Paciente>();
}

public void RegistrarPacientes(Paciente paciente){

    PreparedStatement pst;
    try {
        pst = conn.prepareStatement("insert into PACIENTE values(?,?,?,?,?)");
        pst.setString(1,paciente.getIdentificacion());
        pst.setString(2,paciente.getNombres());
        pst.setString(3,paciente.getApellidos());
        pst.setString(4,paciente.getFechaNacimiento());
        pst.setString(5,paciente.getGenero());
        pst.executeUpdate();
        JOptionPane.showMessageDialog(null,"Paciente Registrado");
    } catch (SQLException ex) {
        Logger.getLogger(GestorPaciente.class.getName()).log(Level.SEVERE, null, ex);
    }

    //pacientes.add(paciente);
}
public LinkedList<Paciente> getPacientebyParametro(int parametro, String valor){
    LinkedList<Paciente> resultado=new LinkedList<Paciente>();
    String sql="";
    //for(Paciente pac:pacientes)
    //{
        switch(parametro){

            case 1: //if(pac.getIdentificacion().equals(valor))
                sql="select * from pacientes where PACIDENTIFICACION='"+valor+"'";
                //resultado.add(pac);
                break;

            case 2: //if(pac.getNombres().equals(valor))
                sql="select * from pacientes where PACNOMBRES='"+valor+"'";
                //resultado.add(pac);
                break;

```

```

        case 3: //if(pac.getApellidos().equals(valor))
            sql="select * from pacientes where PACAPELLIDOS='"+valor+"'";
            //resultado.add(pac);
            break;

        case 4: //if(pac.getGenero().equals(valor))
            sql="select * from pacientes where PACGENERO='"+valor+"'";
            //resultado.add(pac);
            break;
    }
    //}

    try{

        Statement st=conn.createStatement();
        ResultSet rs=st.executeQuery(sql);
        while(rs.Next()){

            resultado.add(new Paciente(rs.getString("PACIDENTIFICACION"),rs.
getString("PACNOMBRES"),rs.getString("PACAPELLIDOS"),rs.
getString("PACFECHANACIMIENTO"),rs.getString("PACGENERO")));

        }

        st.close();

    }catch(Exception e){

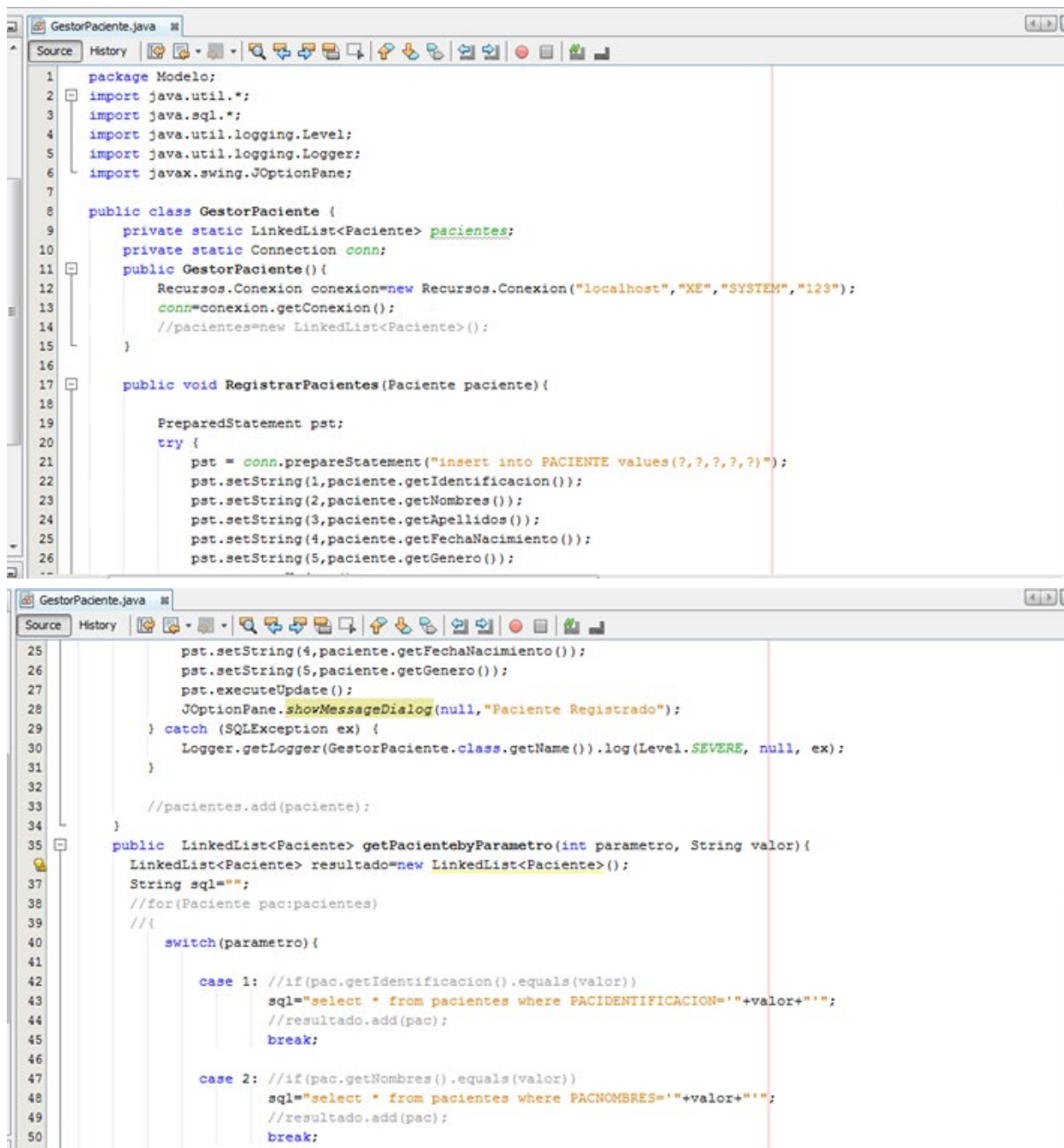
        JOptionPane.showMessageDialog(null, e.getMessage());

    }

    return resultado;
}

}

```

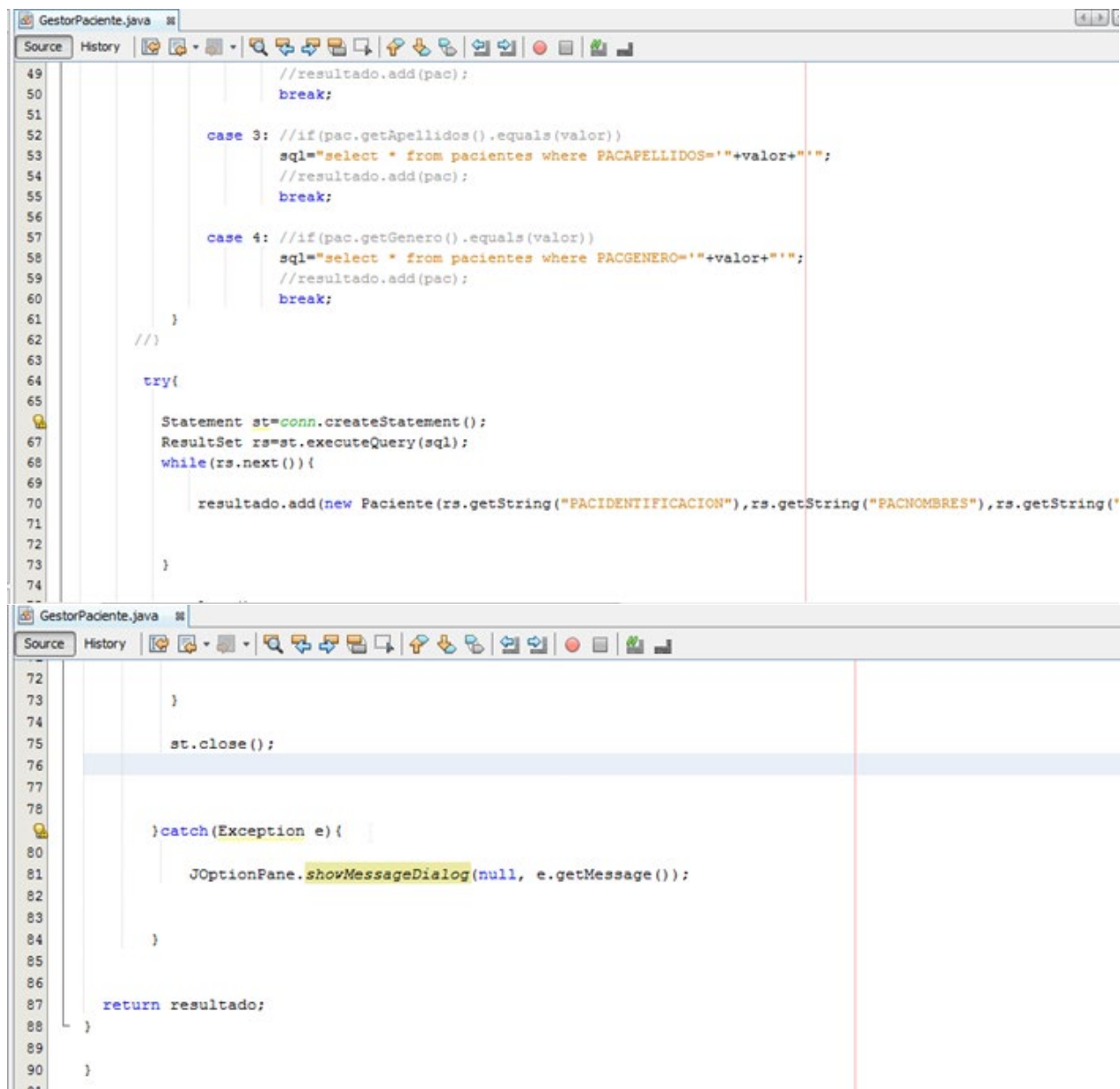


```

1 package Modelo;
2 import java.util.*;
3 import java.sql.*;
4 import java.util.logging.Level;
5 import java.util.logging.Logger;
6 import javax.swing.JOptionPane;
7
8 public class GestorPaciente {
9     private static LinkedList<Paciente> pacientes;
10    private static Connection conn;
11    public GestorPaciente() {
12        Recursos.Conexion conexion=new Recursos.Conexion("localhost","XE","SYSTEM","123");
13        conn=conexion.getConnection();
14        //pacientes=new LinkedList<Paciente>();
15    }
16
17    public void RegistrarPacientes(Paciente paciente){
18
19        PreparedStatement pst;
20        try {
21            pst = conn.prepareStatement("insert into PACIENTE values(?, ?, ?, ?, ?)");
22            pst.setString(1, paciente.getIdentificacion());
23            pst.setString(2, paciente.getNombres());
24            pst.setString(3, paciente.getApellidos());
25            pst.setString(4, paciente.getFechaNacimiento());
26            pst.setString(5, paciente.getGenero());
27
28            pst.executeUpdate();
29            JOptionPane.showMessageDialog(null, "Paciente Registrado");
30        } catch (SQLException ex) {
31            Logger.getLogger(GestorPaciente.class.getName()).log(Level.SEVERE, null, ex);
32        }
33
34        //pacientes.add(paciente);
35    }
36
37    public LinkedList<Paciente> getPacientebyParametro(int parametro, String valor){
38        LinkedList<Paciente> resultado=new LinkedList<Paciente>();
39        String sql="";
40        //for(Paciente pac:pacientes)
41        //{
42            switch(parametro){
43                case 1: //if(pac.getIdentificacion().equals(valor))
44                    sql="select * from pacientes where PACIDENTIFICACION='"+valor+"'";
45                    //resultado.add(pac);
46                    break;
47                case 2: //if(pac.getNombres().equals(valor))
48                    sql="select * from pacientes where PACNOMBRES='"+valor+"'";
49                    //resultado.add(pac);
50                    break;
51            }
52        }
53        return resultado;
54    }
55 }

```

Figura 93. Cambios en el código de clase de gestor paciente - Parte 1.
Fuente: SENA



```

49         //resultado.add(pac);
50         break;
51
52         case 3: //if(pac.getApellidos().equals(valor))
53             sql="select * from pacientes where PACAPELIDOS='"+valor+"'";
54             //resultado.add(pac);
55             break;
56
57         case 4: //if(pac.getGenero().equals(valor))
58             sql="select * from pacientes where PACGENERO='"+valor+"'";
59             //resultado.add(pac);
60             break;
61     }
62     //}
63
64     try{
65
66         Statement st=conn.createStatement();
67         ResultSet rs=st.executeQuery(sql);
68         while(rs.next()){
69
70             resultado.add(new Paciente(rs.getString("PACIDENTIFICACION"),rs.getString("PACNOMBRES"),rs.getString("
71
72         }
73     }
74
75     st.close();
76
77
78     }catch(Exception e){
79
80         JOptionPane.showMessageDialog(null, e.getMessage());
81
82     }
83
84
85
86
87     return resultado;
88 }
89
90 }
91

```

Figura 94. Cambios en el código de clase de gestor paciente - Parte 2.
Fuente: SENA

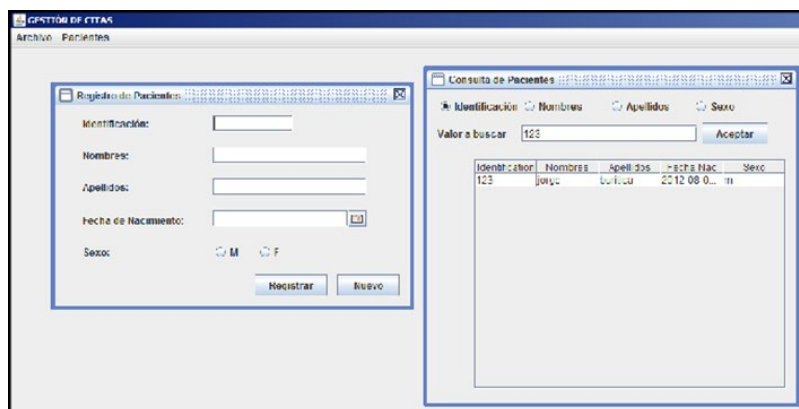
Tabla 15

Explicación del código.

Línea	Explicación
9	Declaración de un objeto de tipo Connection que recibe la conexión con la base de datos.
11..15	Constructor de la clase que envía a la clase Conexion los parámetros específicos de la base de datos y obtiene la conexión con la misma.
20	Se crea un objeto de tipo PreparedStatement , a partir de la conexión creada en el constructor y se establece la sentencia insert , dejando los valores de la misma para ser establecidos posteriormente.
21..25	Definición de los valores que se deben insertar a través de la sentencia insert .
26	Ejecuta la sentencia insert sobre la base de datos.
36	LinkedList para retornar los pacientes consultados de la base de datos.
38..48	De acuerdo con el parámetro de búsqueda se configura la consulta específica (buscar por la identificación, por el nombre, por el apellido o por el sexo).
52	Se ejecuta la consulta.
53..60	Recorre los resultados de la consulta, con cada registro devuelto se crea un paciente nuevo, el cual se almacena en el LinkedList que será retornado.
70	Retorno del LinkedList que posee a todos los pacientes que cumplen con el criterio de búsqueda enviado al método.

Nota: Producción SENA

Después de realizado los cambios en la clase **gestorpaciente** que corresponde al modelo se puede compilar y ejecutar la aplicación, se notará que no es necesario realizar los cambios en las otras capas de la aplicación y que ahora el almacenamiento de los pacientes registrados y las consultas de los mismos se realiza sobre la base de datos.


Figura 95. Ejecutando el programa.
Fuente: SENA

Desarrollo de aplicaciones DE ESCRITORIO CON LENGUAJE



Sesión 7

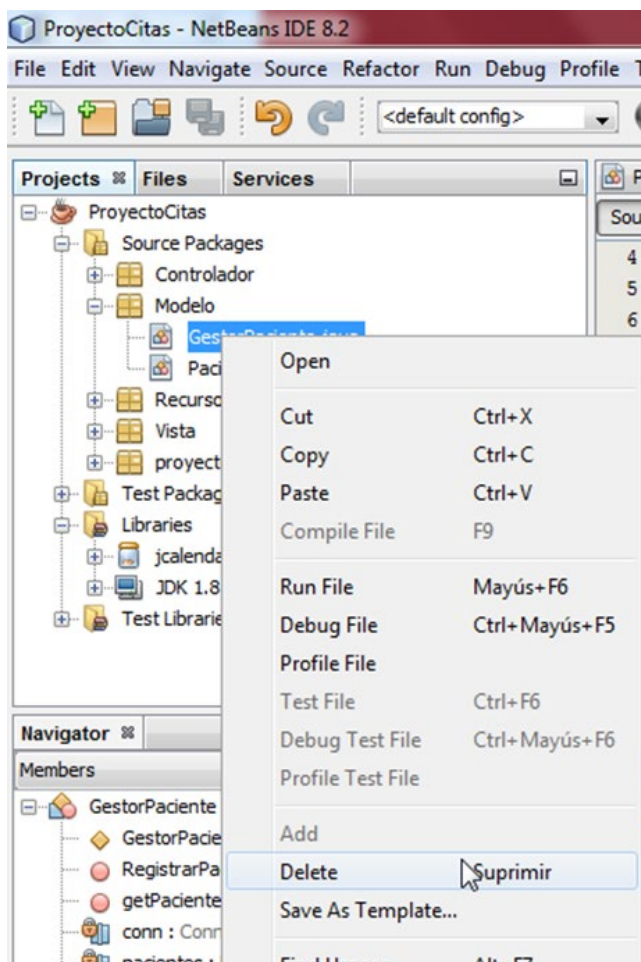
Acceso a la base
de datos con
JPA



En la sesión anterior, el almacenamiento y la consulta de información sobre la base de datos se realizó a través de JDBC java **Database Connectivity**. A través de JDBC se pudo observar que se hace necesario emplear sentencias **SQL Structured Query Language** como la sentencia **insert** para registrar un nuevo paciente en la base de datos, y la sentencia **select** para consultar los pacientes de la base de datos que cumplirán con un determinado criterio de búsqueda.

En esta sesión se empleará un mecanismo diferente de acceso y gestión de los datos en una base de datos conocido como **JPA Java Persistence API**.

NOTA. Antes de iniciar, es recomendable realizar una copia de la carpeta del proyecto **projectocitas**, renombrar la copia como **projectocitas2** y abrir esta última desde **netbeans**, con esto, los cambios efectuados en el proyecto no afectarán el proyecto respaldo.



Para iniciar se abre el proyecto copia desde NETBEANS y en el eliminar las clases **paciente** y **gestorpaciente** del paquete modelo, haciendo clic derecho sobre ellas y seleccionando la opción **delete**.

Figura 96. Eliminando la clase gestor paciente.

Fuente: SENA

NOTA. Al eliminar las clases **paciente** y **gestorpaciente** aparecerán errores en clases del paquete **controlador** y en el **modelo**, más adelante se solucionarán dichos errores.

A continuación, se crea una unidad de persistencia, se hace clic derecho sobre el proyecto **projectocitas** y se selecciona del menú contextual la opción **New→other**.

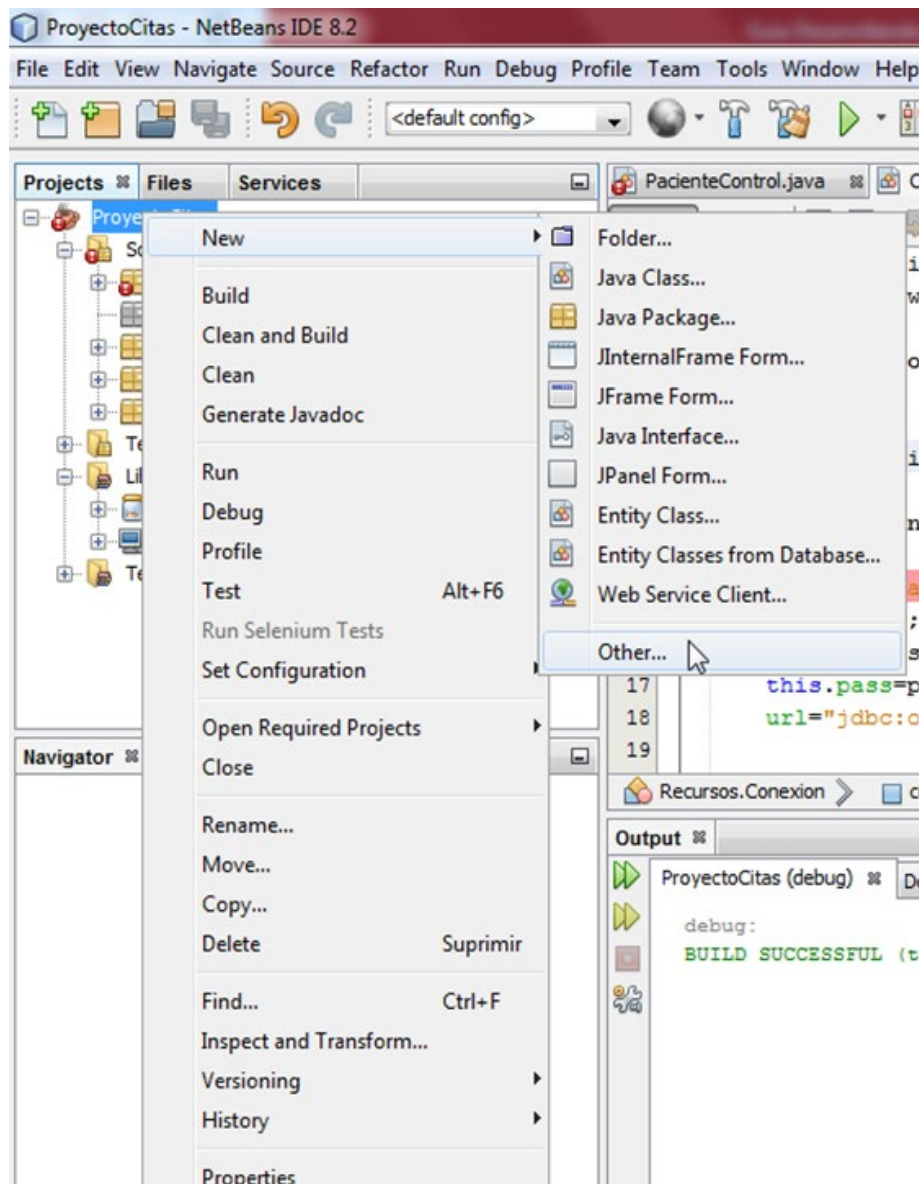


Figura 97. Agregando una nueva clase gestor paciente.
Fuente: SENA

Se selecciona de la ventana la categoría **persistence** y el tipo de archivo **persistence unit** y se hace clic en **Next**.

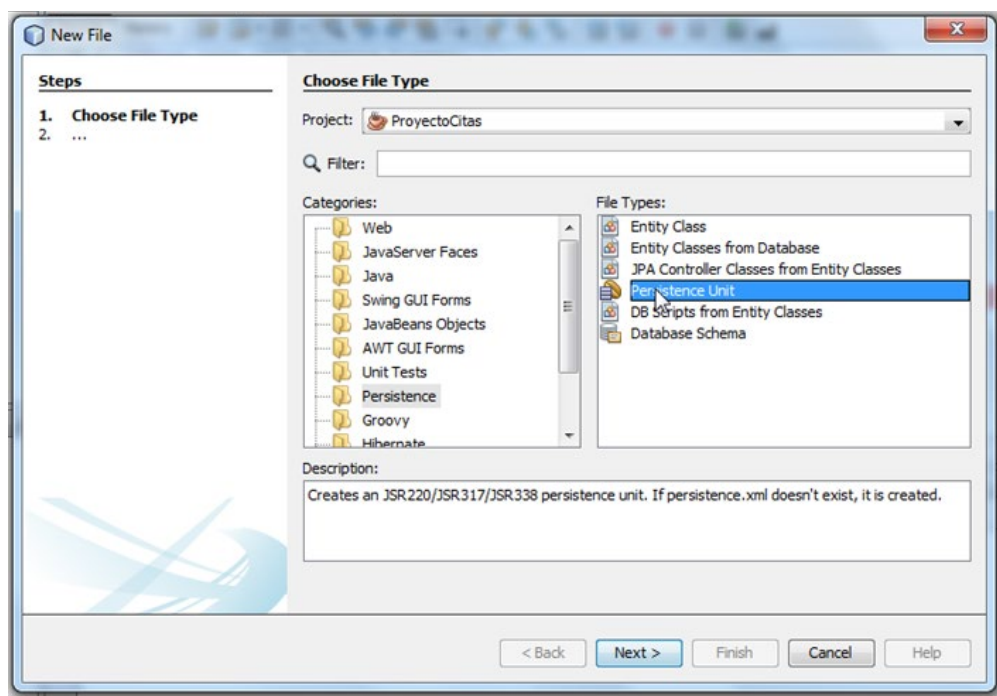


Figura 98. Seleccionando el persistence Unit.
Fuente: SENA

Se indica la librería de persistencia a utilizar y la conexión de base de datos que se emplea, en este caso, la conexión ya existe, y se hace clic en **Finish**.

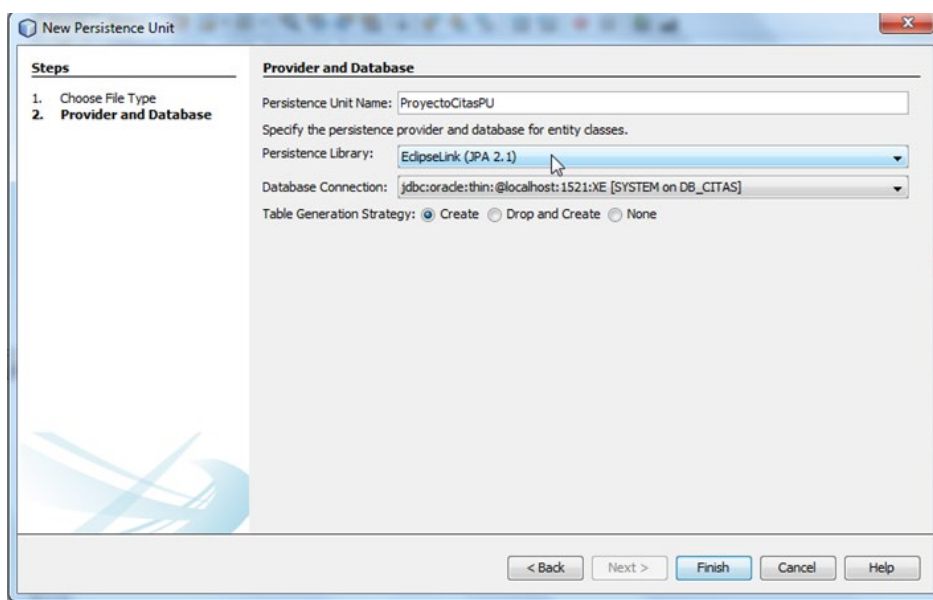


Figura 99. Seleccionando el proveedor de datos.
Fuente: SENA

Automáticamente se crea la unidad de persistencia conformada por un archivo xml llamado ***persistence.xml*** dentro de una carpeta META-INF, este archivo ***persistence.xml*** contiene información relacionada con la base de datos y las librerías JPA.

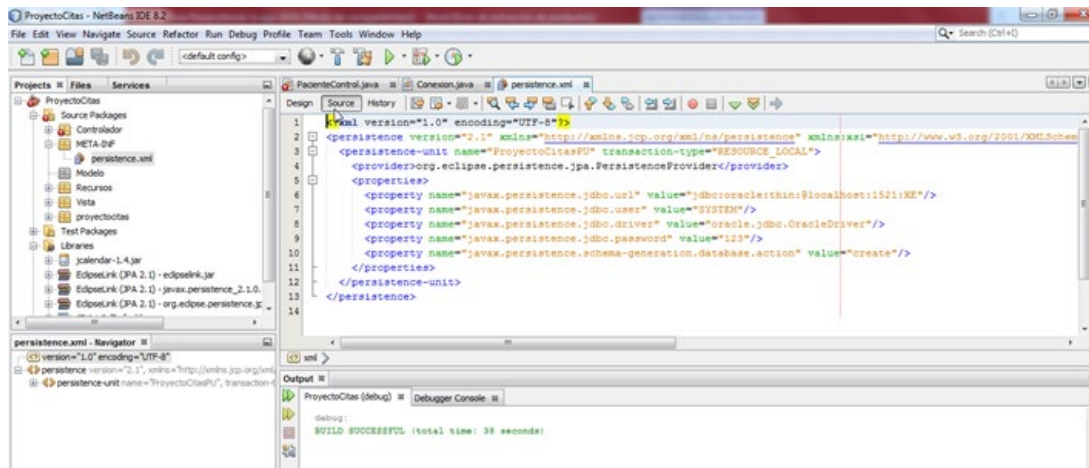


Figura 100. Visualizando el META-INF.
Fuente: SENA

Luego de creada la unidad de persistencia, se procede a crear las clases de entidad estas clases son clases java que se mapean a tablas de la base de datos.

Hacer clic derecho sobre el paquete modelo, el cual debe estar vacío y se selecciona del menú contextual **New→other**.

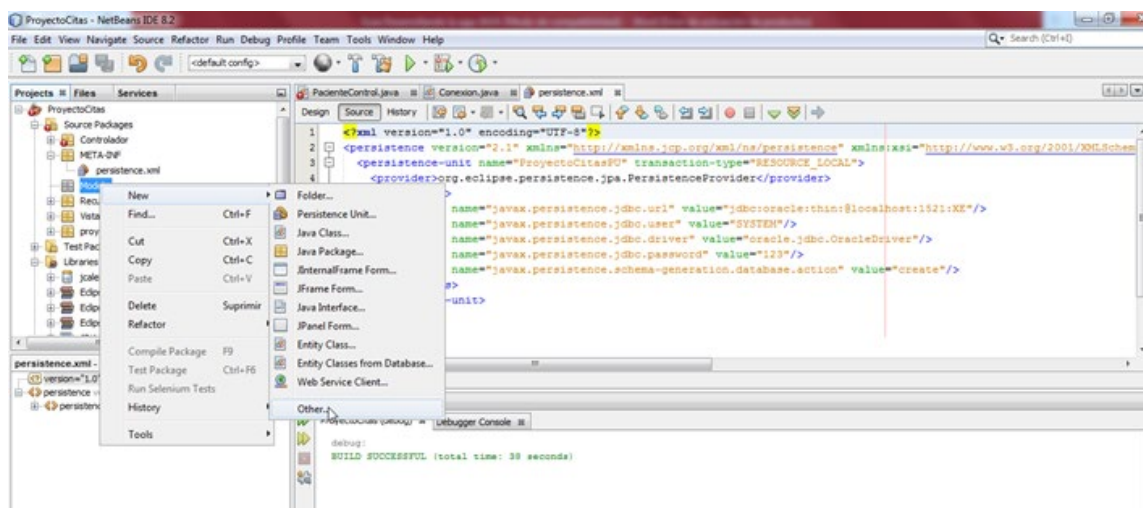


Figura 101. Creando las nuevas clases entidad.
Fuente: SENA

De la ventana emergente, se selecciona la categoría persistente y el tipo de archivo **Entity Classes From Database**.

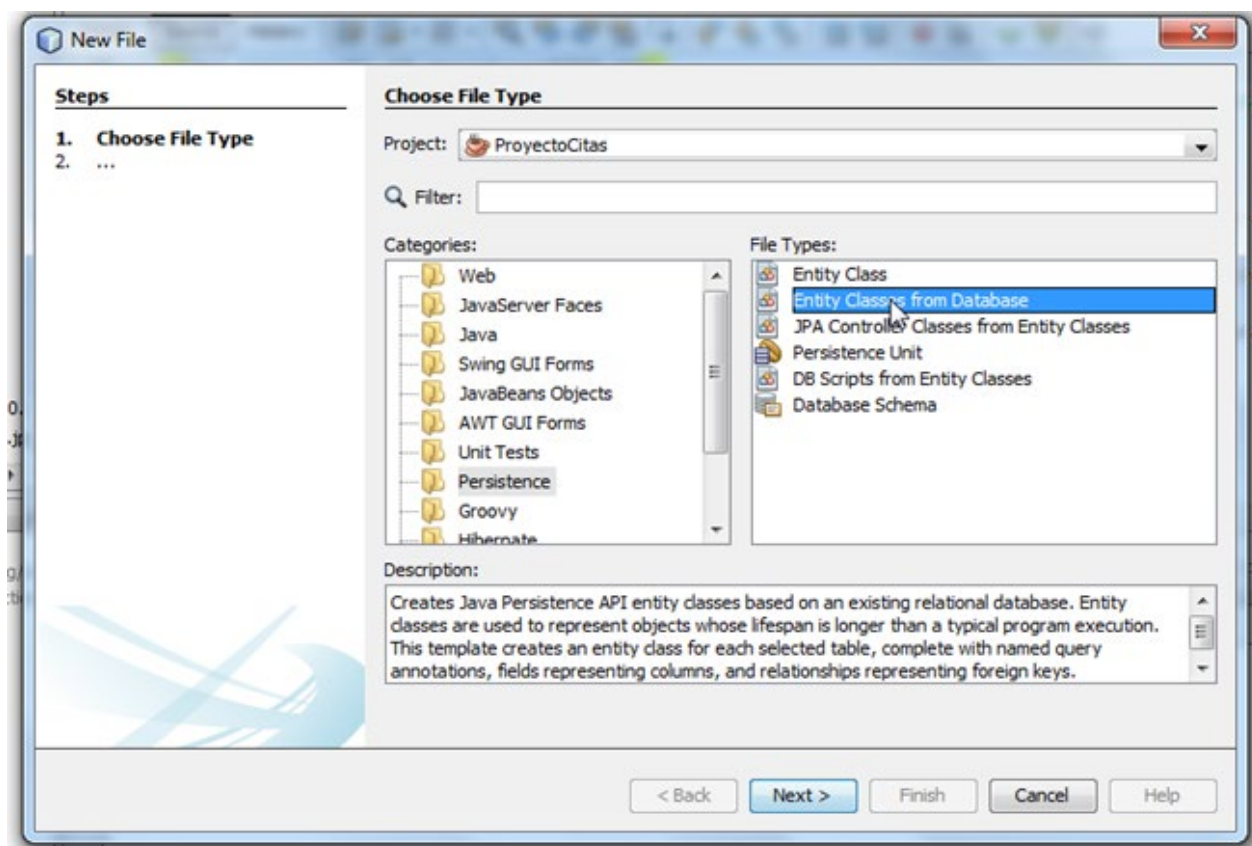


Figura 102. Seleccionando el tipo de clases Entity Classes From Database.

Fuente: SENA

En la siguiente ventana se seleccionan la conexión de base de datos ya establecida anteriormente, de esta forma se cargan las tablas de la base de datos, se selecciona la tabla que interesa en la aplicación que es la de pacientes y se hace clic en Finish.

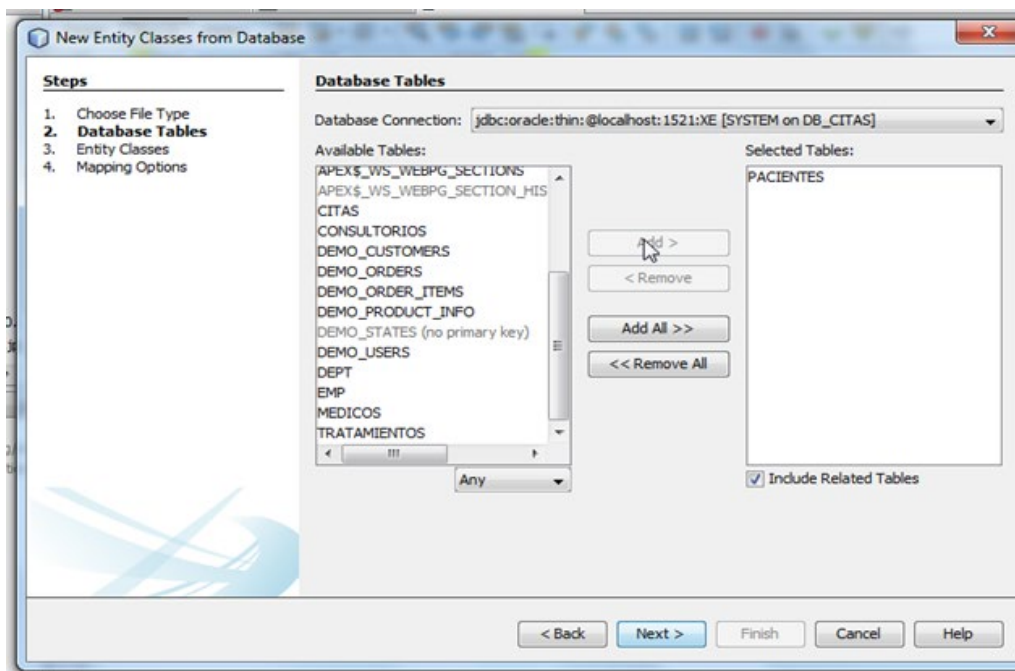


Figura 103. Cargando las tablas de datos al modelo.
Fuente: SENA

En la siguiente ventana, se quita la selección de los **checkbox** y se hace clic en Next.

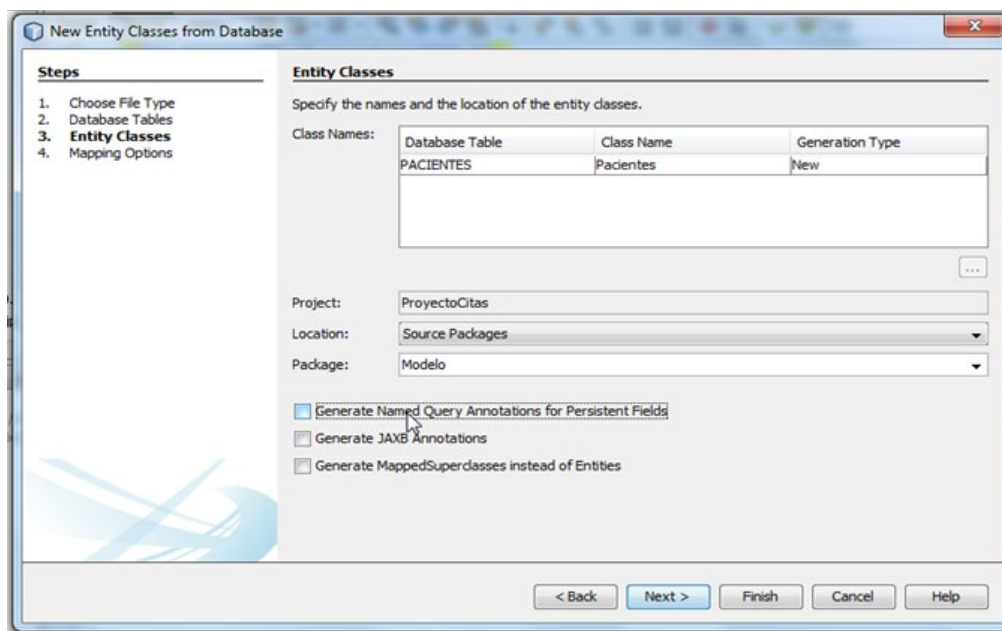


Figura 104. Especificando el nombre de la clase entidad.
Fuente: SENA

En la siguiente ventana se dejan seleccionadas las opciones por defecto y se hace clic en Finish.

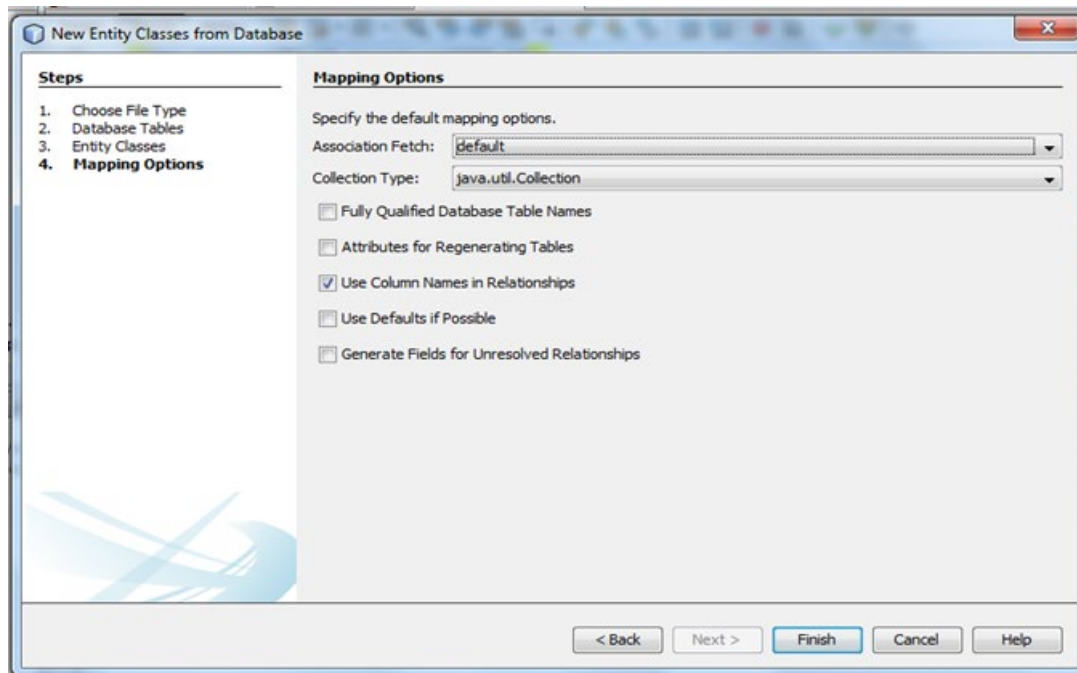
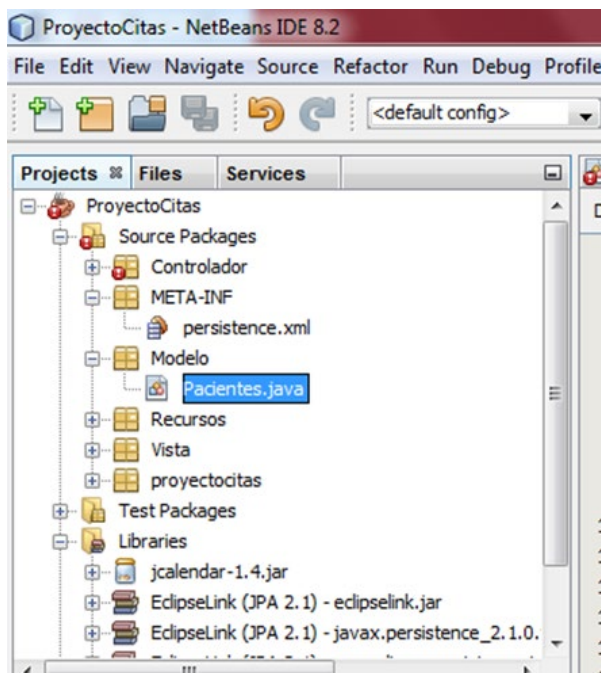


Figura 105. Mapaeando la clase entidad.
Fuente: SENA

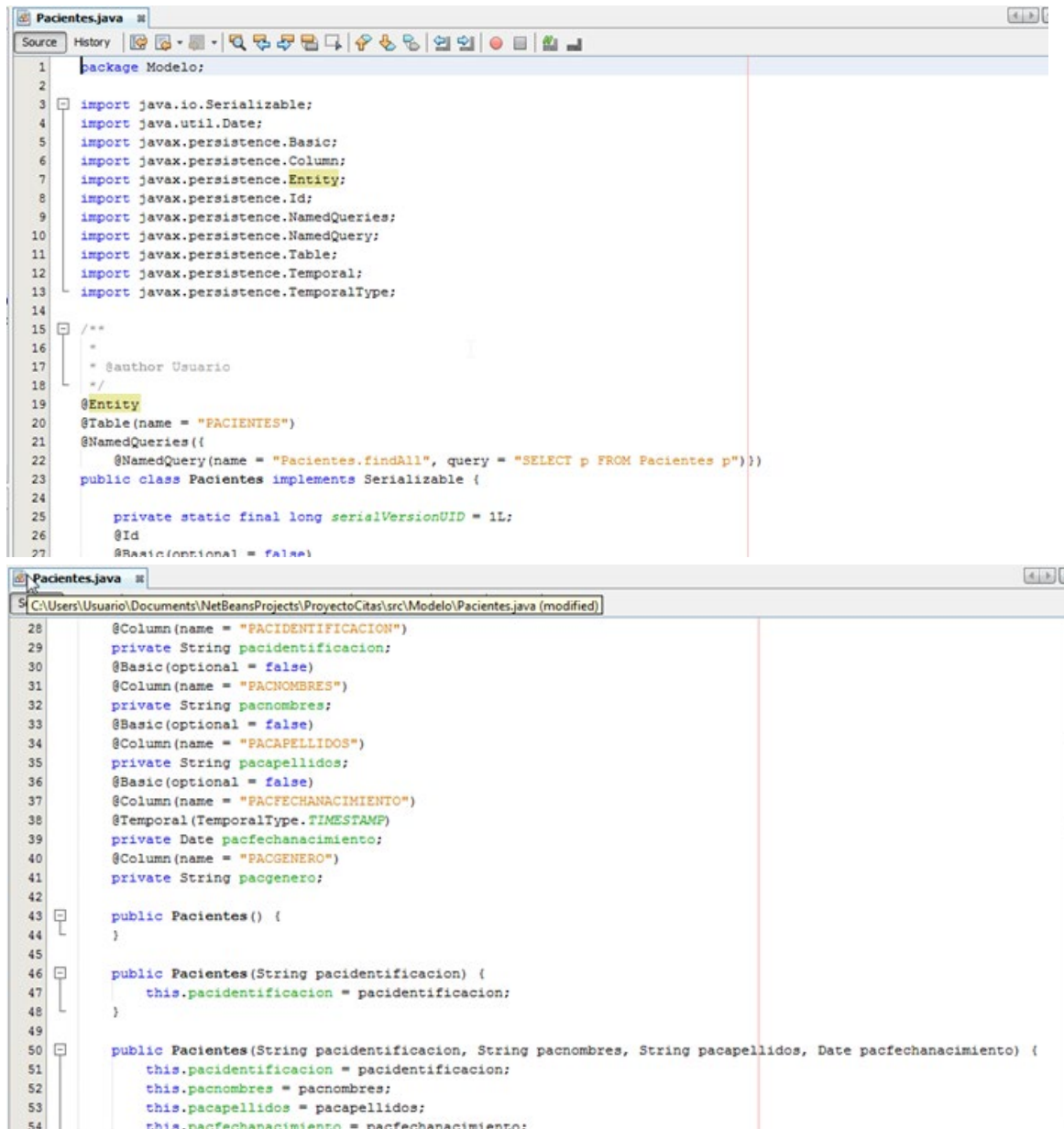


Se observa que la clase pacientes ha sido creada dentro del paquete Modelo.

Figura 106. Visualizando el modelo.
Fuente: SENA

Al abrir la clase paciente se puede observar todo el código que se crea de manera automática. Este código contiene los atributos, los constructores y los métodos **getters** y **setters** de la clase pacientes basados en los campos de la tabla pacientes de la base de datos.

Además, estos elementos de la clase, también se observan una serie de clases del paquete **javax, persistence** importadas.



```

1 package Modelo;
2
3 import java.io.Serializable;
4 import java.util.Date;
5 import javax.persistence.Basic;
6 import javax.persistence.Column;
7 import javax.persistence.Entity;
8 import javax.persistence.Id;
9 import javax.persistence.NamedQueries;
10 import javax.persistence.NamedQuery;
11 import javax.persistence.Table;
12 import javax.persistence.Temporal;
13 import javax.persistence.TemporalType;
14
15 /**
16  * @author Usuario
17  */
18 @Entity
19 @Table(name = "PACIENTES")
20 @NamedQueries({
21     @NamedQuery(name = "Pacientes.findAll", query = "SELECT p FROM Pacientes p")})
22 public class Pacientes implements Serializable {
23
24     private static final long serialVersionUID = 1L;
25     @Id
26     @Basic(optional = false)
27     @Column(name = "PACIDENTIFICACION")
28     private String pacidentificacion;
29     @Basic(optional = false)
30     @Column(name = "PACNOMBRES")
31     private String pacnombres;
32     @Basic(optional = false)
33     @Column(name = "PACAPPELLIDOS")
34     private String pacapellidos;
35     @Basic(optional = false)
36     @Column(name = "PACFECHANACIMIENTO")
37     @Temporal(TemporalType.TIMESTAMP)
38     private Date pacfechanacimiento;
39     @Column(name = "PACGENERO")
40     private String pacgenero;
41
42     public Pacientes() {
43     }
44
45     public Pacientes(String pacidentificacion) {
46         this.pacidentificacion = pacidentificacion;
47     }
48
49     public Pacientes(String pacidentificacion, String pacnombres, String pacapellidos, Date pacfechanacimiento) {
50         this.pacidentificacion = pacidentificacion;
51         this.pacnombres = pacnombres;
52         this.pacapellidos = pacapellidos;
53         this.pacfechanacimiento = pacfechanacimiento;
54     }
55
56 }

```

Figura 107. Clase paciente - Parte 1.
Fuente: SENA

```

Pacientes.java
Source History
55 }
56
57 public String getPacidentificacion() {
58     return pacidentificacion;
59 }
60
61 public void setPacidentificacion(String pacidentificacion) {
62     this.pacidentificacion = pacidentificacion;
63 }
64
65 public String getPacnombres() {
66     return pacnombres;
67 }
68
69 public void setPacnombres(String pacnombres) {
70     this.pacnombres = pacnombres;
71 }
72
73 public String getPacapellidos() {
74     return pacapellidos;
75 }
76
77 public void setPacapellidos(String pacapellidos) {
78     this.pacapellidos = pacapellidos;
79 }
80
81 public Date getPacfechanacimiento() {
82     return pacfechanacimiento;
83 }
84
85 public void setPacfechanacimiento(Date pacfechanacimiento) {
86     this.pacfechanacimiento = pacfechanacimiento;
87 }
88
89 public String getPacgenero() {
90     return pacgenero;
91 }
92
93 public void setPacgenero(String pacgenero) {
94     this.pacgenero = pacgenero;
95 }
96
97 @Override
98 public int hashCode() {
99     int hash = 0;
100     hash += (pacidentificacion != null ? pacidentificacion.hashCode() : 0);
101     return hash;
102 }
103
104 @Override
105 public boolean equals(Object object) {
106     // TODO: Warning - this method won't work in the case the id fields are not set
107     if (!(object instanceof Pacientes)) {
108         return false;
109     }
110     Pacientes other = (Pacientes) object;
111     if ((this.pacidentificacion == null && other.pacidentificacion != null) || (this.pacidentificacion != null &
112         return false;
113     }
114     return true;
115 }
116
117 @Override
118 public String toString() {
119     return "Modelo.Pacientes[ pacidentificacion=" + pacidentificacion + " ]";
120 }
121
122 }
123

```

Figura 108. Clase paciente - Parte 2.
Fuente: SENA

Una vez creada la clase de identidad se debe crear la clase de contraste en esta entidad, es decir, se requiere de una clase que permita ejecutar operaciones sobre los pacientes como por ejemplo agregarlos modificarlos o consultarlos.

Se hace clic derecho sobre el paquete **modelo** y se selecciona del menú new- **other**, de la ventana emergente se selecciona la categoría **persistence** y el tipo de archivo JPA controller **classes from entity classes from** y se da clic en **Next**.

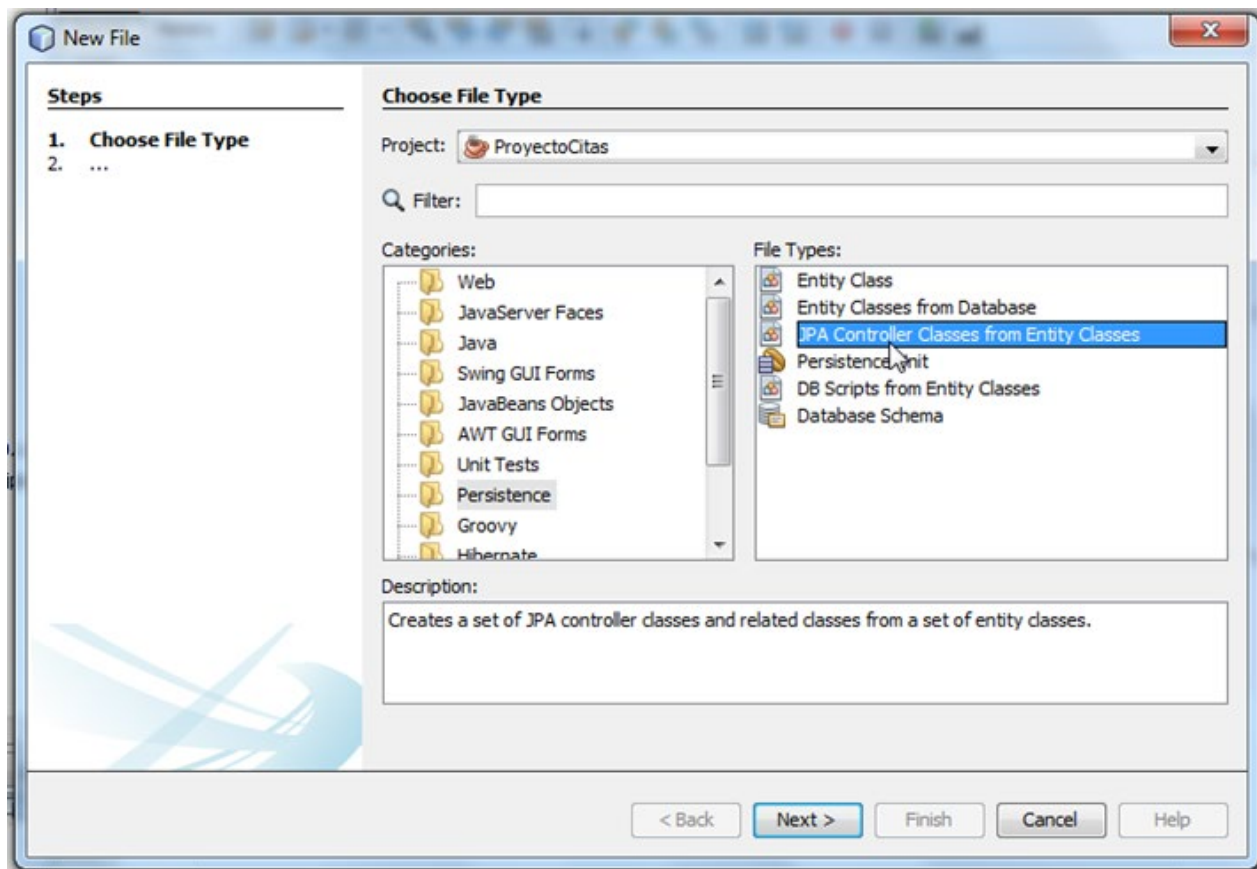


Figura 109. Configurando el controlador JPA.
Fuente: SENA

Estas clases son las que forman parte de la JPA, por último, sobresale un conjunto de signos de arroba @, estos elementos son anotaciones las cuales permiten mantener el mapeo (vínculo) entre los atributos de la clase y los campos de la tabla de pacientes

Se selecciona la clase de la entidad que existe en el proyecto pacientes y se hace clic en el botón **Next**.

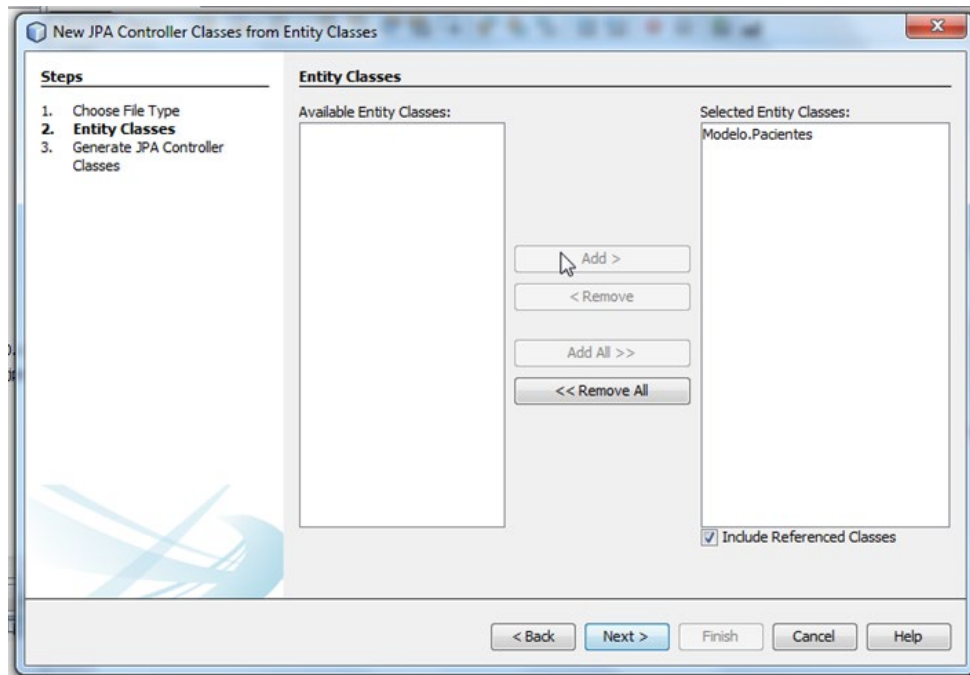


Figura 110. Seleccionando el modelo.
Fuente: SENA

Por último, se selecciona el paquete modelo y se hace clic en **Finish**.

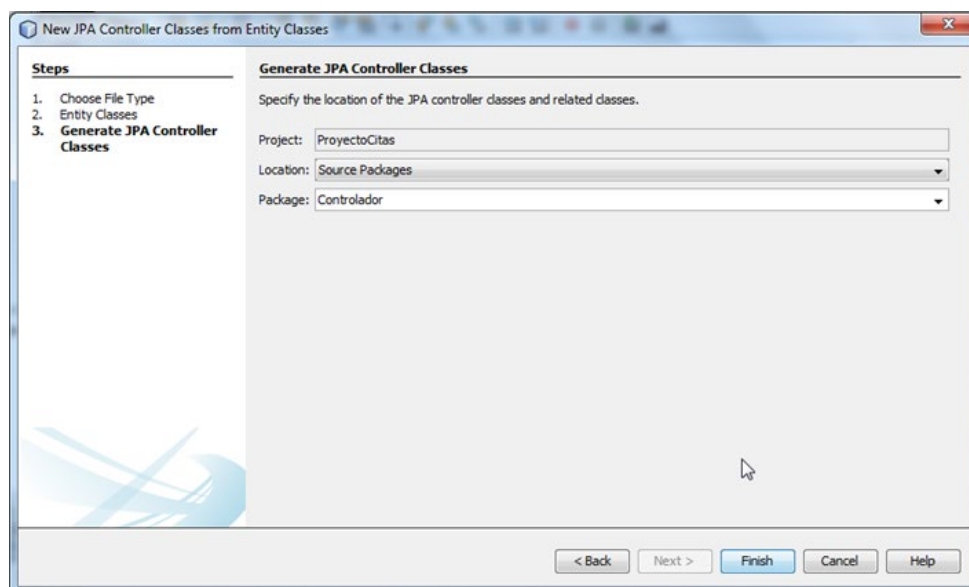


Figura 111. Generando JPA Controller clases.
Fuente: SENA

Se observa como la nueva clase se agrega al paquete modelo y se crea automáticamente un nuevo paquete llamado modelo **exceptions** el cual contiene 3 clases de tipo excepción.

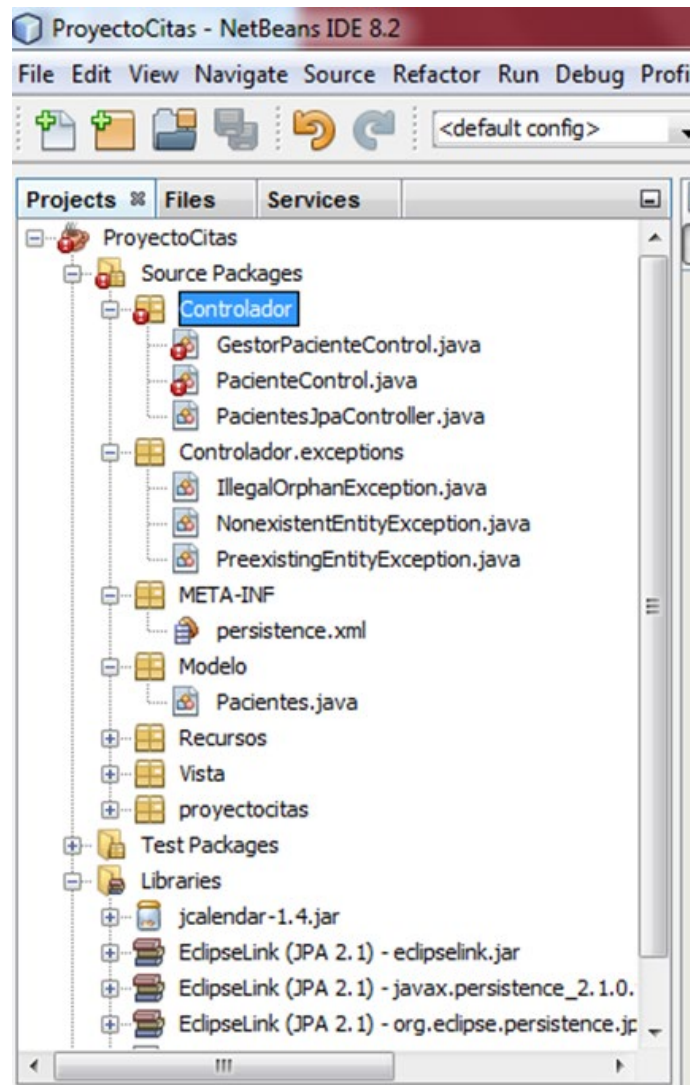
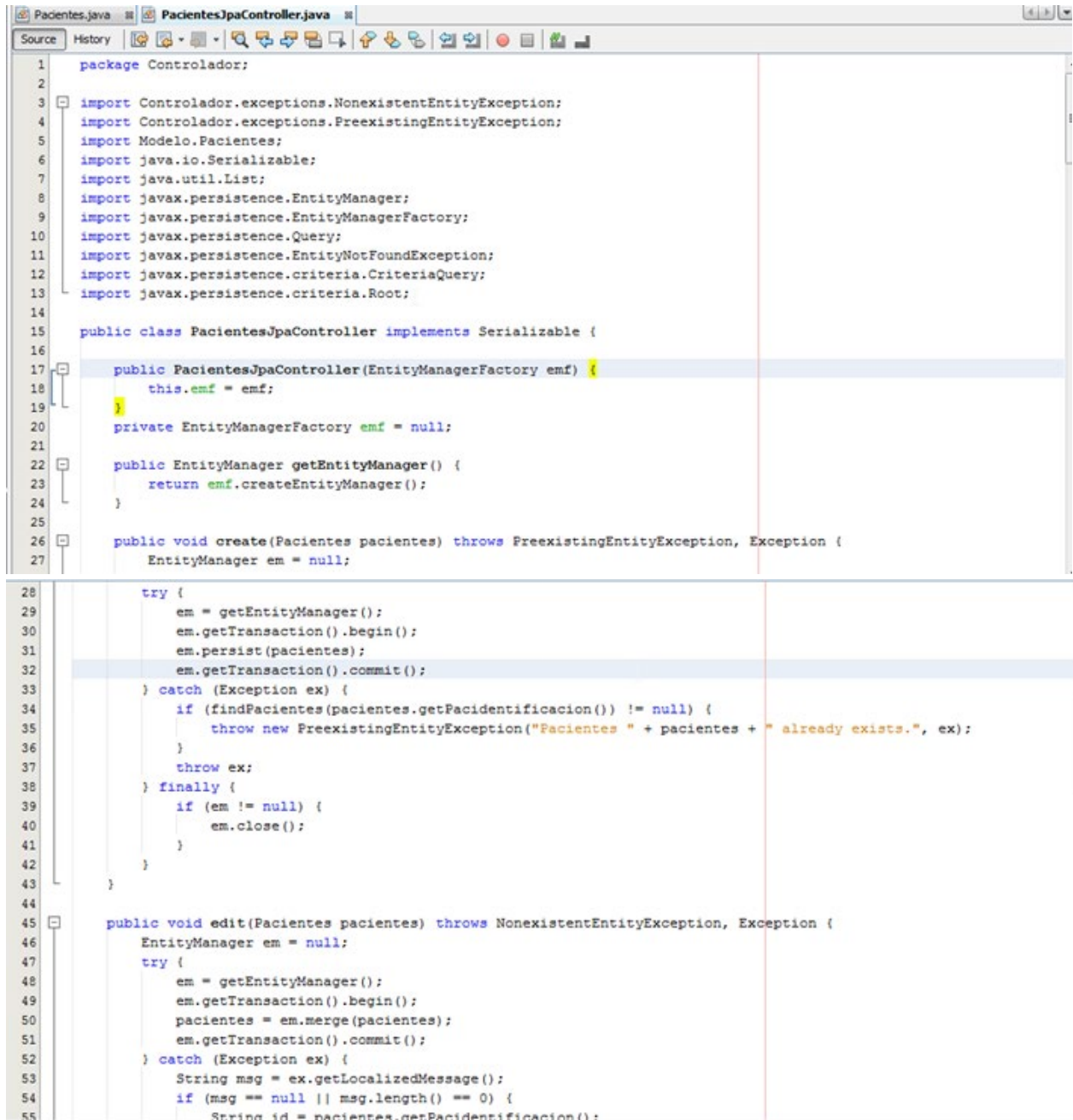


Figura 112. Visualizando la estructura controlador.
Fuente: SENA

De esta manera se crea de forma automática la clase, la cual contiene métodos como **create, edit, destroy & find** que se emplean para insertar, modificar, eliminar, y consultar **pacientes** de la base de datos respectivamente.



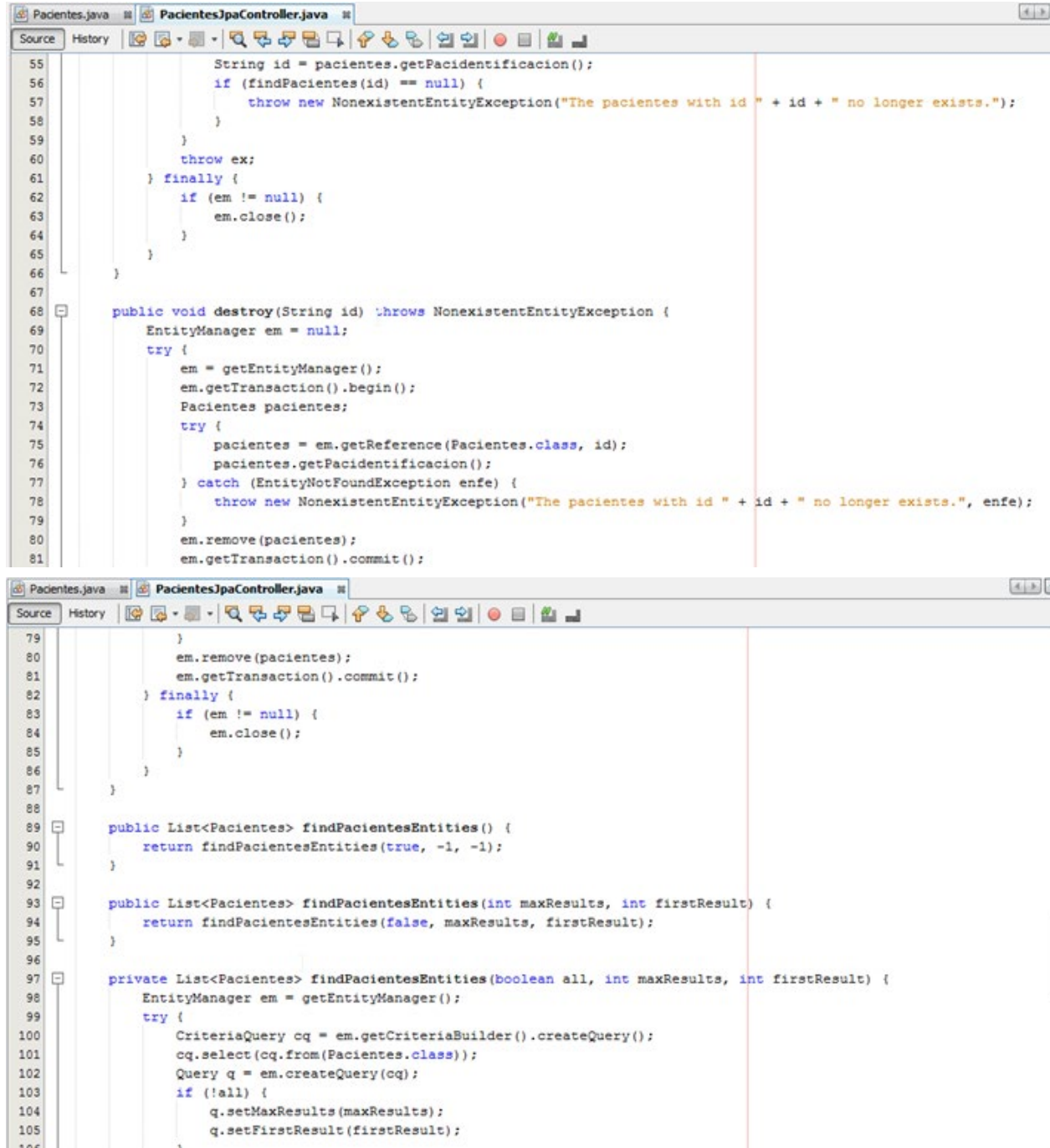
```

1 package Controlador;
2
3 import Controlador.exceptions.NonexistentEntityException;
4 import Controlador.exceptions.PreexistingEntityException;
5 import Modelo.Pacientes;
6 import java.io.Serializable;
7 import java.util.List;
8 import javax.persistence.EntityManager;
9 import javax.persistence.EntityManagerFactory;
10 import javax.persistence.Query;
11 import javax.persistence.EntityNotFoundException;
12 import javax.persistence.criteria.CriteriaQuery;
13 import javax.persistence.criteria.Root;
14
15 public class PacientesJpaController implements Serializable {
16
17     public PacientesJpaController(EntityManagerFactory emf) {
18         this.emf = emf;
19     }
20     private EntityManagerFactory emf = null;
21
22     public EntityManager getEntityManager() {
23         return emf.createEntityManager();
24     }
25
26     public void create(Pacientes pacientes) throws PreexistingEntityException, Exception {
27         EntityManager em = null;
28
29         try {
30             em = getEntityManager();
31             em.getTransaction().begin();
32             em.persist(pacientes);
33             em.getTransaction().commit();
34         } catch (Exception ex) {
35             if (findPacientes(pacientes.getPacientificacion()) != null) {
36                 throw new PreexistingEntityException("Pacientes " + pacientes + " already exists.", ex);
37             }
38             throw ex;
39         } finally {
40             if (em != null) {
41                 em.close();
42             }
43         }
44     }
45
46     public void edit(Pacientes pacientes) throws NonexistentEntityException, Exception {
47         EntityManager em = null;
48         try {
49             em = getEntityManager();
50             em.getTransaction().begin();
51             pacientes = em.merge(pacientes);
52             em.getTransaction().commit();
53         } catch (Exception ex) {
54             String msg = ex.getLocalizedMessage();
55             if (msg == null || msg.length() == 0) {
56                 String id = pacientes.getPacientificacion();

```

Figura 113. Código forma automática de la clase - Parte 1.

Fuente: SENA



```

55      String id = pacientes.getPacientificacion();
56      if (findPacientes(id) == null) {
57          throw new NonexistentEntityException("The pacientes with id " + id + " no longer exists.");
58      }
59      }
60      throw ex;
61      } finally {
62          if (em != null) {
63              em.close();
64          }
65      }
66  }
67
68  public void destroy(String id) throws NonexistentEntityException {
69      EntityManager em = null;
70      try {
71          em = getEntityManager();
72          em.getTransaction().begin();
73          Pacientes pacientes;
74          try {
75              pacientes = em.getReference(Pacientes.class, id);
76              pacientes.getPacientificacion();
77          } catch (EntityNotFoundException enfe) {
78              throw new NonexistentEntityException("The pacientes with id " + id + " no longer exists.", enfe);
79          }
80          em.remove(pacientes);
81          em.getTransaction().commit();
82      } finally {
83          if (em != null) {
84              em.close();
85          }
86      }
87  }
88
89  public List<Pacientes> findPacientesEntities() {
90      return findPacientesEntities(true, -1, -1);
91  }
92
93  public List<Pacientes> findPacientesEntities(int maxResults, int firstResult) {
94      return findPacientesEntities(false, maxResults, firstResult);
95  }
96
97  private List<Pacientes> findPacientesEntities(boolean all, int maxResults, int firstResult) {
98      EntityManager em = getEntityManager();
99      try {
100          CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
101          cq.select(cq.from(Pacientes.class));
102          Query q = em.createQuery(cq);
103          if (!all) {
104              q.setMaxResults(maxResults);
105              q.setFirstResult(firstResult);
106          }

```

Figura 114. Código forma automática de la clase - Parte 2.
Fuente: SENA

```

103         if (!all) {
104             q.setMaxResults(maxResults);
105             q.setFirstResult(firstResult);
106         }
107         return q.getResultList();
108     } finally {
109         em.close();
110     }
111 }
112
113 public Pacientes findPacientes(String id) {
114     EntityManager em = getEntityManager();
115     try {
116         return em.find(Pacientes.class, id);
117     } finally {
118         em.close();
119     }
120 }
121
122 public int getPacientesCount() {
123     EntityManager em = getEntityManager();
124     try {
125         CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
126         Root<Pacientes> rt = cq.from(Pacientes.class);
127         cq.select(em.getCriteriaBuilder().count(rt));
128         Query q = em.createQuery(cq);
129         return ((Long) q.getSingleResult()).intValue();
130     } finally {
131         em.close();
132     }
133 }
134
135 }

```

Figura 115. Código forma automática de la clase - Parte 3.
Fuente: SENA

Ahora que se ha creado las clases pacientes y **pacientescontroller** con la api JPA, sólo resta hacer uso de ellas desde las clases del paquete del controlador de este proyecto. Para eso, se debe modificar algunas partes del código de las clases paciente control y **gestorpacientecontrol** del paquete controlador para que usen los métodos de las clases pacientes y **pacientesjpacontroller** creadas a través del asistente.

Se abre la clase paciente control y sobre ella las modificaciones respectivas en su código.

```

1  package controlador;
2
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.text.SimpleDateFormat;
6  import java.util.Date;
7  import javax.persistence.EntityManagerFactory;
8  import javax.persistence.Persistence;
9  import javax.swing.JOptionPane;
10 import modelo.exceptions.PreexistingEntityException;
11
12 public class PacienteControl implements ActionListener
13 {
14     vista.RegPacienteInternalFrame pacienteVista;
15     modelo.Pacientes pacienteModelo;
16     modelo.PacientesJpaController gestorPacienteModelo;
17
18     public PacienteControl(vista.RegPacienteInternalFrame pacienteVista)
19     {
20         this.pacienteVista=pacienteVista;
21         EntityManagerFactory emf= Persistence.createEntityManagerFactory("ProyectoCitasPU");
22         gestorPacienteModelo = new modelo.PacientesJpaController(emf);
23     }
24
25     @Override

```

Figura 116. Modificación a código clase paciente control.
Fuente: SENA

Explicación del código

Tabla 16

Explicación del código.

Línea	Explicación
16	Declaración del objeto de tipo JpaController que permite realizar operaciones contra la base de datos.
21	Definición de la unidad de persistencia empleada.
22	Creación del objeto de tipo JpaController que realizará operaciones contra la base de datos.
40	Creación de un objeto de la clase Pacientes que se encuentra mapeada con la tabla Pacientes de la base de datos.
41.... 45	Asignación de valores a los atributos de la clase Pacientes .
48	Uso del objeto de tipo PacientesJpaController para insertar un nuevo paciente en la tabla PACIENTES a través del método créate .

Nota: Producción SENA

```

25 public void actionPerformed(ActionEvent e)
26 {
27     if(e.getSource().equals(pacienteVista.RegistrarBtn))
28     {
29         String identificacion=pacienteVista.IdentificacionTxt.getText();
30         String nombres=pacienteVista.NombresTxt.getText();
31         String apellidos=pacienteVista.ApellidosTxt.getText();
32         SimpleDateFormat formato = new SimpleDateFormat("dd/MM/yyyy");
33         String fechaNacimiento=formato.format(pacienteVista.FechaNacimientoDtc.getDate());
34         char sexo='\0';
35         if(pacienteVista.MasculinoOpt.isSelected())
36             sexo='m';
37         else
38             sexo='f';
39
40         pacienteModelo= new modelo.Pacientes();
41         pacienteModelo.setPacidentificacion(identificacion);
42         pacienteModelo.setPacapellidos(apellidos);
43         pacienteModelo.setPacnombres(nombres);
44         pacienteModelo.setPacfechanacimiento(new Date(fechaNacimiento));
45         pacienteModelo.setPacsexo(sexo);
46         try
47         {
48             gestorPacienteModelo.create(pacienteModelo);
49             JOptionPane.showMessageDialog(pacienteVista, "Paciente registrado correctamente");
50         }
51         catch (PreexistingEntityException ex)
52         {
53             JOptionPane.showMessageDialog(pacienteVista, "El paciente ya existe");
54         }
55         catch (Exception ex)
56         {
57             JOptionPane.showMessageDialog(pacienteVista, ex.getMessage());
58         }
59     }
60     if(e.getSource().equals(pacienteVista.NuevoBtn))
61     {
62         pacienteVista.IdentificacionTxt.setText(null);
63         pacienteVista.NombresTxt.setText(null);
64         pacienteVista.ApellidosTxt.setText(null);
65         pacienteVista.FechaNacimientoDtc.setDate(null);
66         pacienteVista.MasculinoOpt.setSelected(false);
67         pacienteVista.FemeninoOpt.setSelected(false);
68         pacienteVista.IdentificacionTxt.requestFocus();
69     }
70 }
71 }

```

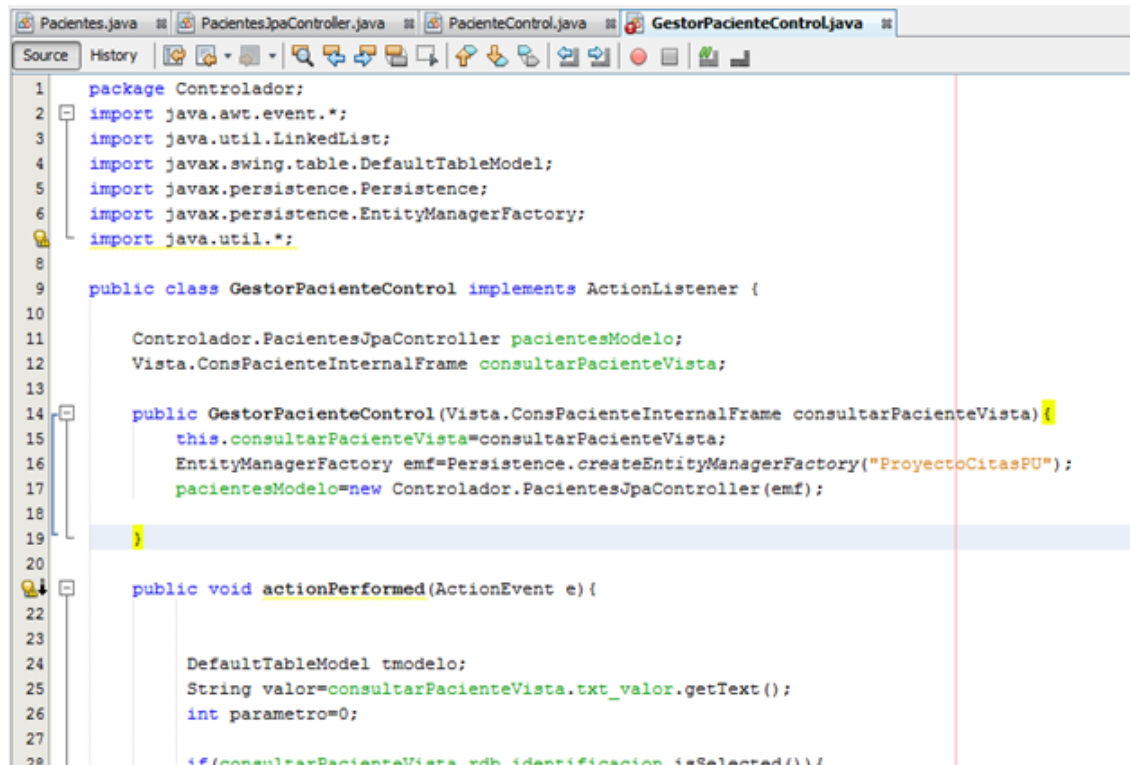
Figura 117. Código clase paciente control.

Fuente: SENA

La **clase paciente control** cambió un poco su código para hacer uso de la clase **pacientejpacontroller** y gracias a esto la inserción de pacientes en la base de datos se realiza de manera más directa y sin utilizar el lenguaje SQL.

Ahora para presentar el resultado de la consulta que el usuario realiza en el formulario **consulta de pacientes**, se debe modificar, también el código de la clase **gestorpacientescontrol** del paquete controlador para que utilice la clase **pacientesjpacontroller** del paquete modelo y así.

Se abre la clase **gestorpacientecontrol** y se realizan los siguientes cambios en su código.



```

1 package Controlador;
2 import java.awt.event.*;
3 import java.util.LinkedList;
4 import javax.swing.table.DefaultTableModel;
5 import javax.persistence.Persistence;
6 import javax.persistence.EntityManagerFactory;
7 import java.util.*;
8
9 public class GestorPacienteControl implements ActionListener {
10
11     Controlador.PacientesJpaController pacientesModelo;
12     Vista.ConsPacienteInternalFrame consultarPacienteVista;
13
14     public GestorPacienteControl(Vista.ConsPacienteInternalFrame consultarPacienteVista) {
15         this.consultarPacienteVista=consultarPacienteVista;
16         EntityManagerFactory emf=Persistence.createEntityManagerFactory("ProyectoCitasPU");
17         pacientesModelo=new Controlador.PacientesJpaController(emf);
18     }
19
20     public void actionPerformed(ActionEvent e){
21
22         DefaultTableModel tmodelo;
23         String valor=consultarPacienteVista.txt_valor.getText();
24         int parametro=0;
25
26         if(consultarPacienteVista.rdb_identificacion.isSelected()){

```

Figura 118. Código clase gestorpacientecontrol.
Fuente: SENA

Explicación del código.

Tabla 17
Explicación del código.

Línea	Explicación
11	Declaración del objeto de tipo JpaController que permite realizar operaciones contra la base de datos.
17	Definición de la unidad de persistencia empleada.
18	Creación del objeto de tipo JpaController que realizará operaciones contra la base de datos.
35	Se obtienen todos los pacientes de la base de datos haciendo uso del método findPacientesEntities() de la clase PacientesJpaController .
36..53	Recorrido de cada paciente obtenido y si coincide con el criterio y valor de búsqueda, se envía al método mostrarEnTabla para que sea presentado en el JTable del formulario Consulta de Pacientes .
55..65	Método que recibe un paciente y presenta su información en el formulario Consulta de Pacientes .

Nota: Producción SENA

```

20
21
22 public void actionPerformed(ActionEvent e){
23     DefaultTableModel tmodelo;
24     String valor=consultarPacienteVista.txt_valor.getText();
25     int parametro=0;
26
27     if(consultarPacienteVista.rdb_identificacion.isSelected()){
28         parametro=1;
29     }
30     if(consultarPacienteVista.rdb_nombres.isSelected()){
31         parametro=2;
32     }
33     if(consultarPacienteVista.rdb_apellidos.isSelected()){
34         parametro=3;
35     }
36     if(consultarPacienteVista.rdb_genero.isSelected()){
37         parametro=4;
38     }
39
40     List<Modelo.Pacientes> pacientes=pacientesModelo.findPacientesEntities();
41     String registro[]=new String[5];
42
43     String []Titulos = {"Identificacion","Nombre","Apellidos","Fecha Nacimiento","Genero"};
44
45     tmodelo=new DefaultTableModel();
46     tmodelo.setColumnIdentifiers(Titulos);
47
48     tmodelo=new DefaultTableModel();
49     tmodelo.setColumnIdentifiers(Titulos);
50
51     for(Modelo.Pacientes p:pacientes){
52         registro[0]=p.getPacidentificacion();
53         registro[1]=p.getPacnombres();
54         registro[2]=p.getPacapellidos();
55         registro[3]=p.getPacfechanacimiento().toString();
56         registro[4]=p.getPacgenero();
57         tmodelo.addRow(registro);
58     }
59
60     consultarPacienteVista.tbl_datos.setModel(tmodelo);
61
62 }
63

```

Figura 119. Cambios código clase gestorpacientecontrol.
Fuente: SENA

Antes de ejecutar la aplicación es necesario agregar el driver de conexión ORACLE al proyecto, con el fin de ser reconocido por la unidad de persistencia. Se hace clic derecho sobre la carpeta **libraries** del proyecto y selecciona del menú contextual opción **AddJAR/ Folder**.

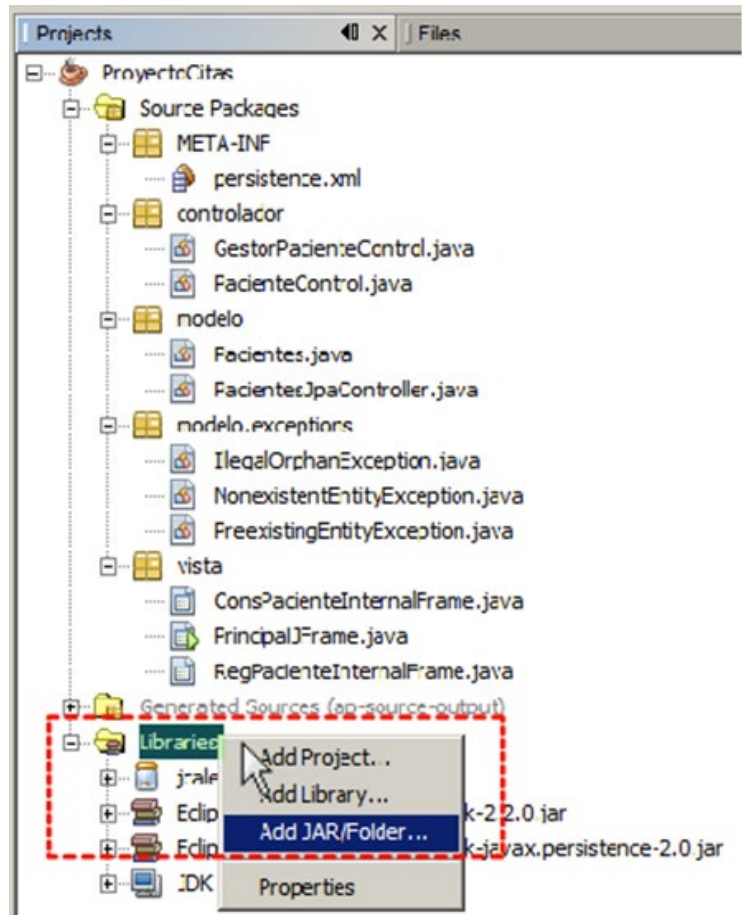


Figura 120. Adicionando el driver de conexión jdbc
Fuente: SENA

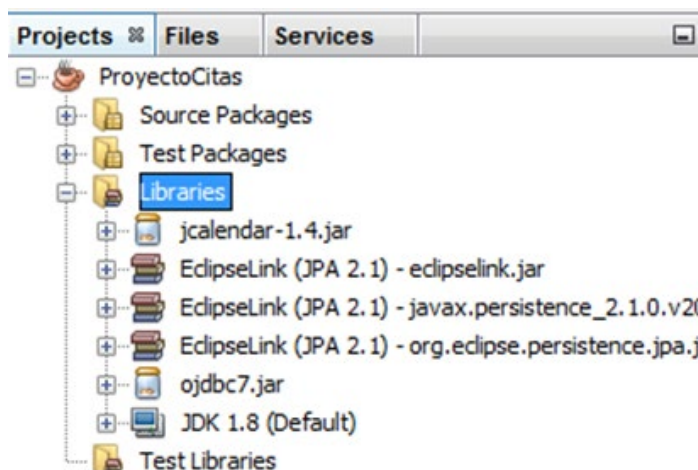


Figura 121. Visualizando el odbc driver de Oracle.
Fuente: SENA

El proyecto ya ha quedado listo, al ejecutarlo el funcionamiento debe ser el mismo que se obtuvo en la sesión 6, con acceso a la base de datos, pero en el lugar de usar JDBC, se usa JPA.

Desarrollo de aplicaciones DE ESCRITORIO CON LENGUAJE



Sesión 8

Generación de
reportes
impresos



Para finalizar el proyecto, se creará un reporte impreso que mostrará información obtenida de la base de datos. El reporte presenta la información de todos los pacientes registrados. Las herramientas que se emplean para la creación y visualización del reporte son: **iReport** y **jasperreports**, estos plugins se agregan a **netbeans** formando parte de su entorno de desarrollo integrado.

Para conocer más detalles sobre iReport , puede consultar en <http://es.scribd.com/doc/37388195/manual-de-iReport>

Los plugins necesarios están conformados por los siguientes archivos:

-  `{distro.zipfilename.extensions}.nbm`
-  `iReport-5.5.0.nbm`
-  `jasperreports-components-plugin-5.5.0.nbm`
-  `jasperserver-plugin-5.5.0.nbm`

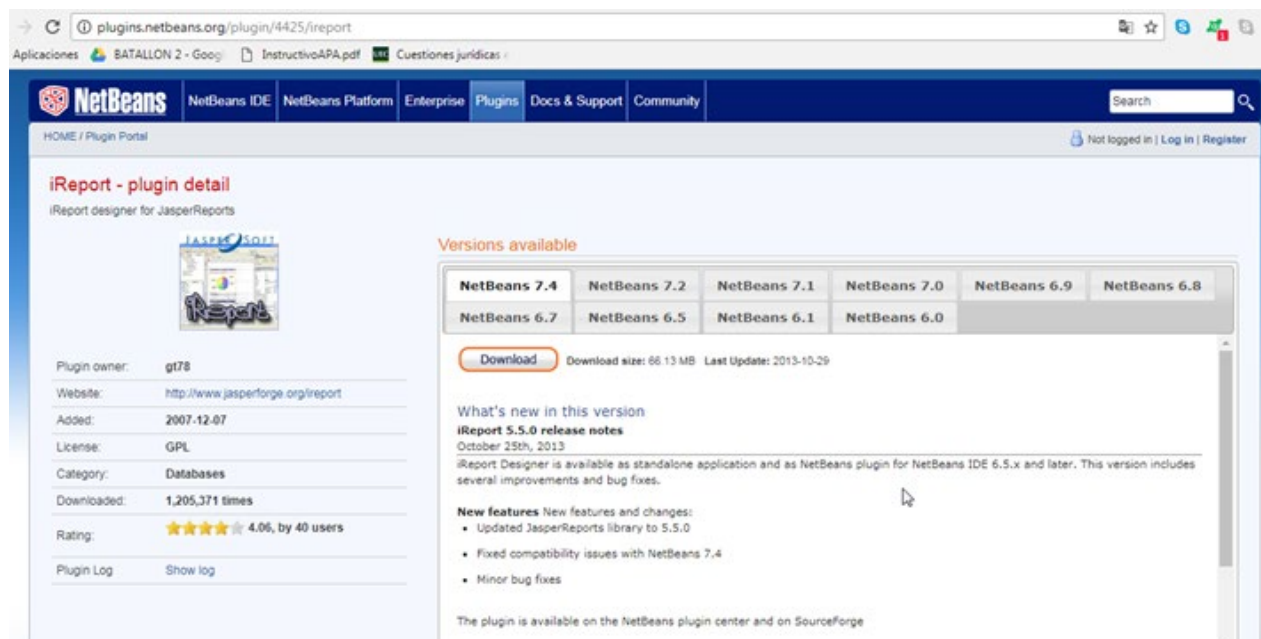


Figura 122. Descargando iReport.

Fuente: SENA

Una vez descargados los archivos en cualquier ruta del equipo desde Netbeans se selecciona en el menú **tools- plugins**.

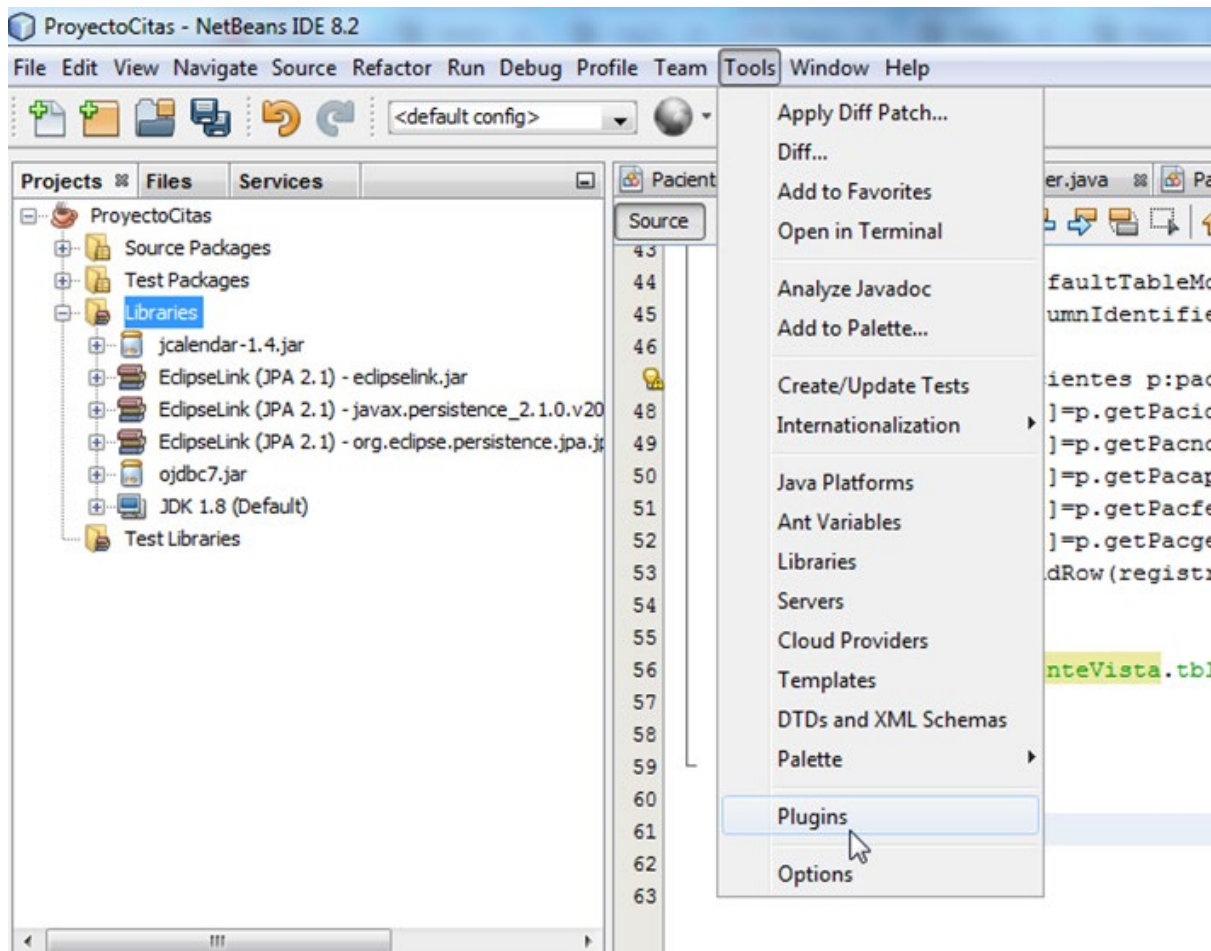


Figura 123. Agregando eliReport.
Fuente: SENA

Se selecciona la pestaña downloaded (descargados) y luego se hace clic en el botón **add plugins**, posteriormente se buscan los archivos descargados y se hace clic en el botón **abrir**.

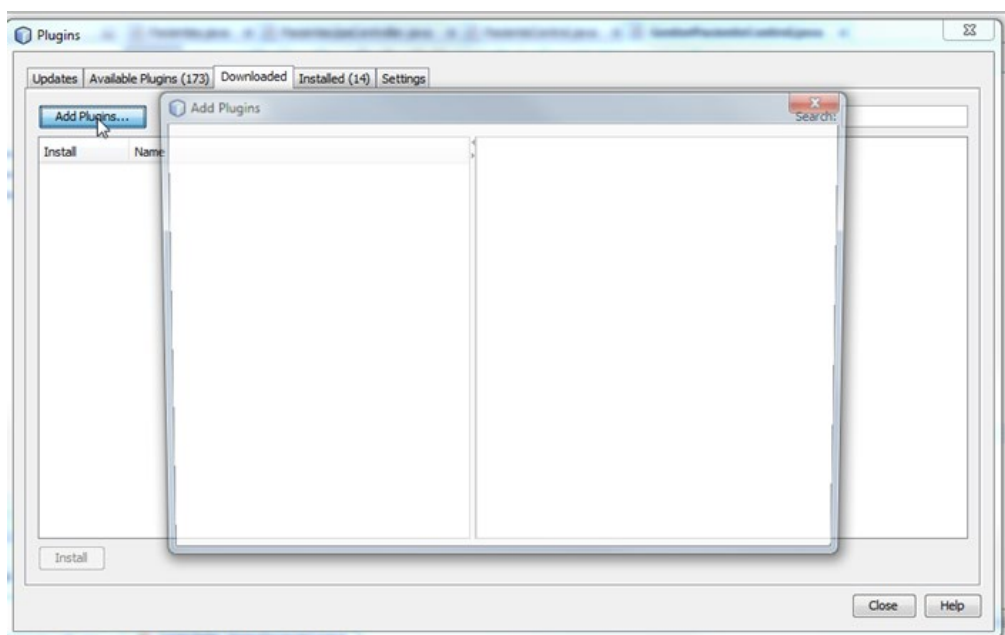


Figura 124. Agregando iReport.
Fuente: SENA

Se hace clic en el botón **install** y sigue los pasos del asistente para la instalación, durante la instalación, es posible que se deba deshabilitar el firewall desde el panel de control para que se lleve a cabo correctamente.

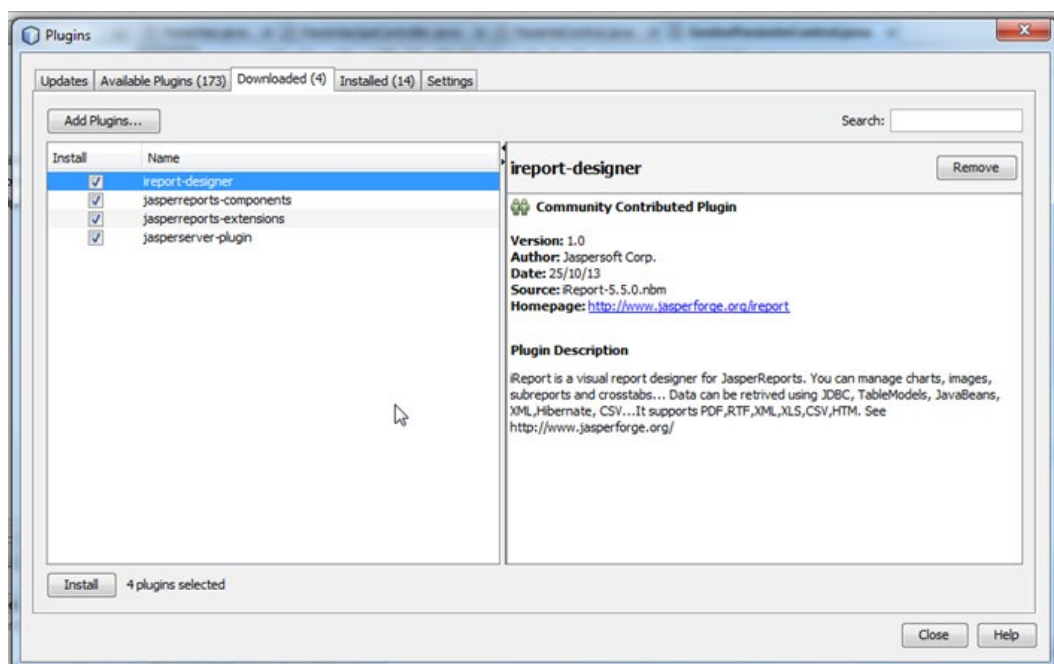


Figura 125. Importando los componentes de iReport.
Fuente: SENA

Al terminar la instalación se hace clic en el botón **Finish**.

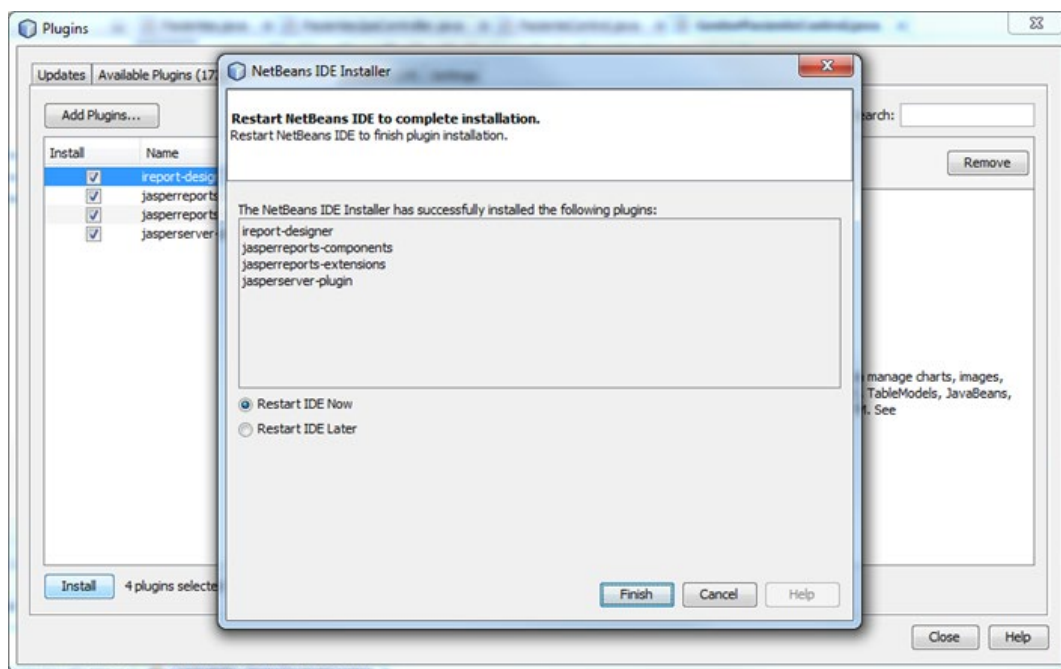


Figura 126. Reiniciando el IDE.

Fuente: SENA

Para que el nuevo plugin instalado tome efecto se recomienda cerrar y volver a abrir el **netbeans**, al ingresar a netbeans se notarán algunos elementos nuevos en su entorno, se trata de los componentes asociados a los reportes de **iReport**, ellos son el origen de datos y la ventana de salida **iReport**.

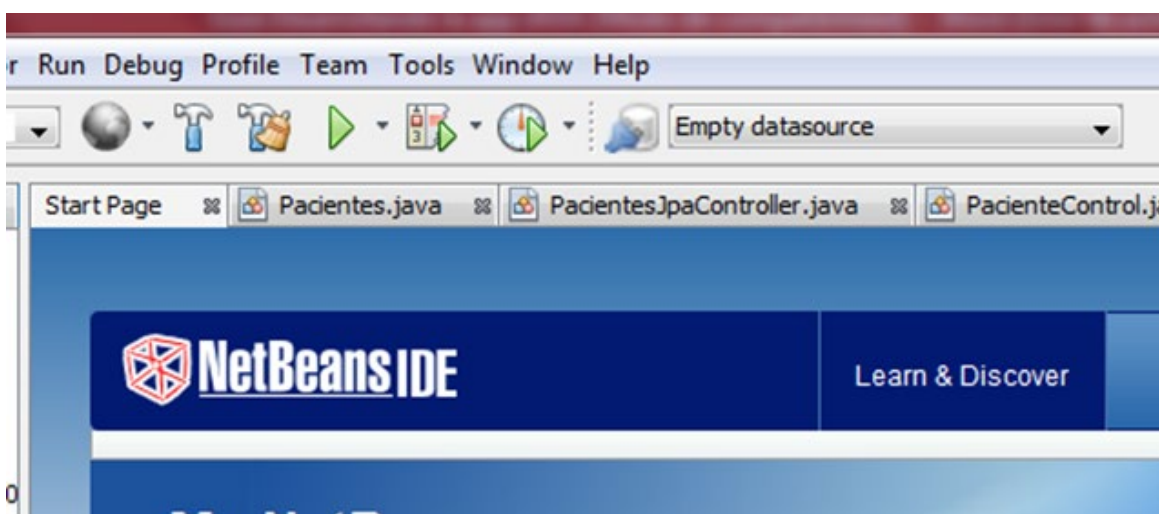


Figura 127. Verificando la instalación de iReport.

Fuente: SENA

Se ingresa al menú **tools libraries**. Allí se notará la librería **jasperreports** con todos sus archivos.

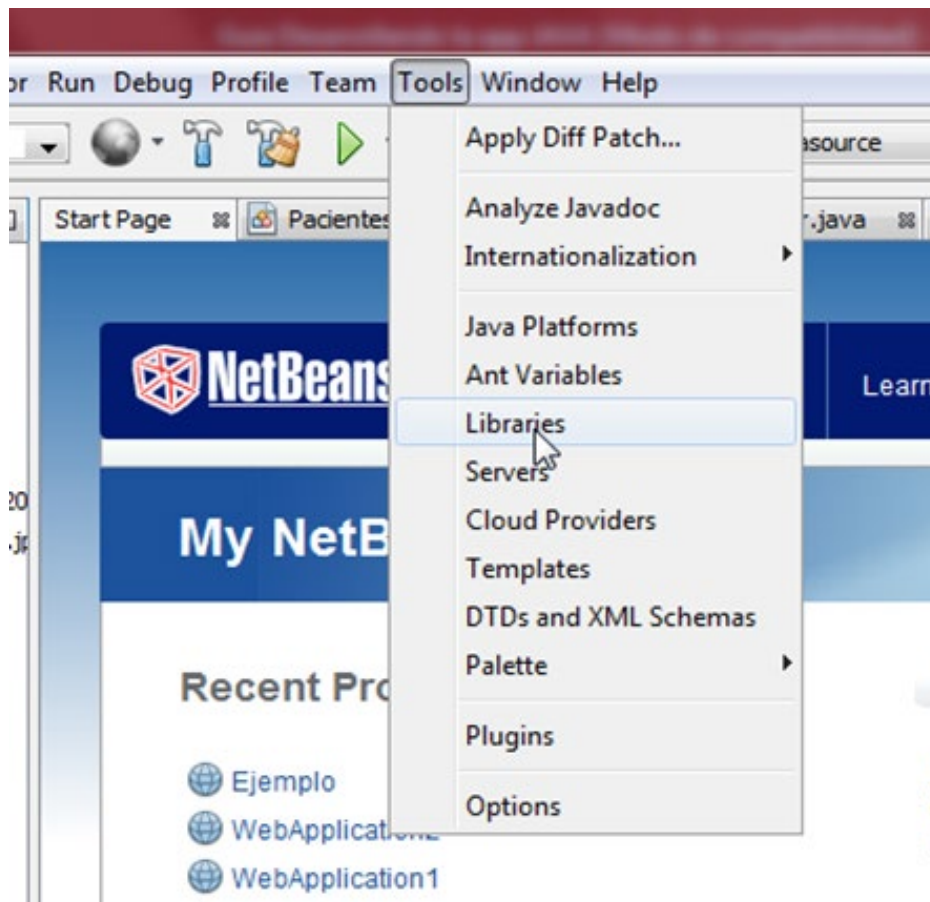


Figura 128. Verificando las librerías de iReport.
Fuente: SENA

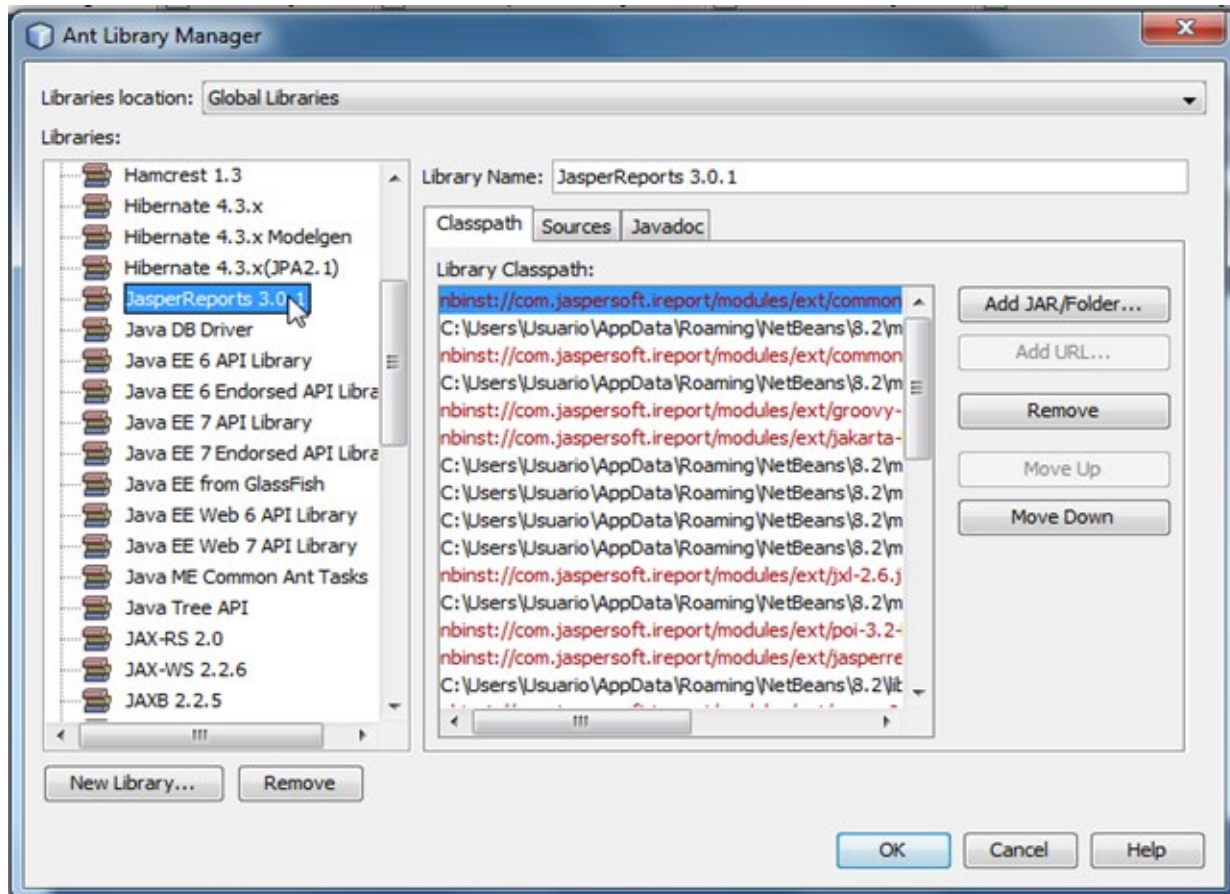


Figura 129. Visualizando las librerías de iReport.

Fuente: SENA

Luego de verificar lo anterior, se pasa a realizar una prueba de conexión del componente **iReport** contra la base de datos, para ello, se hace clic en el botón de **report data sources**.

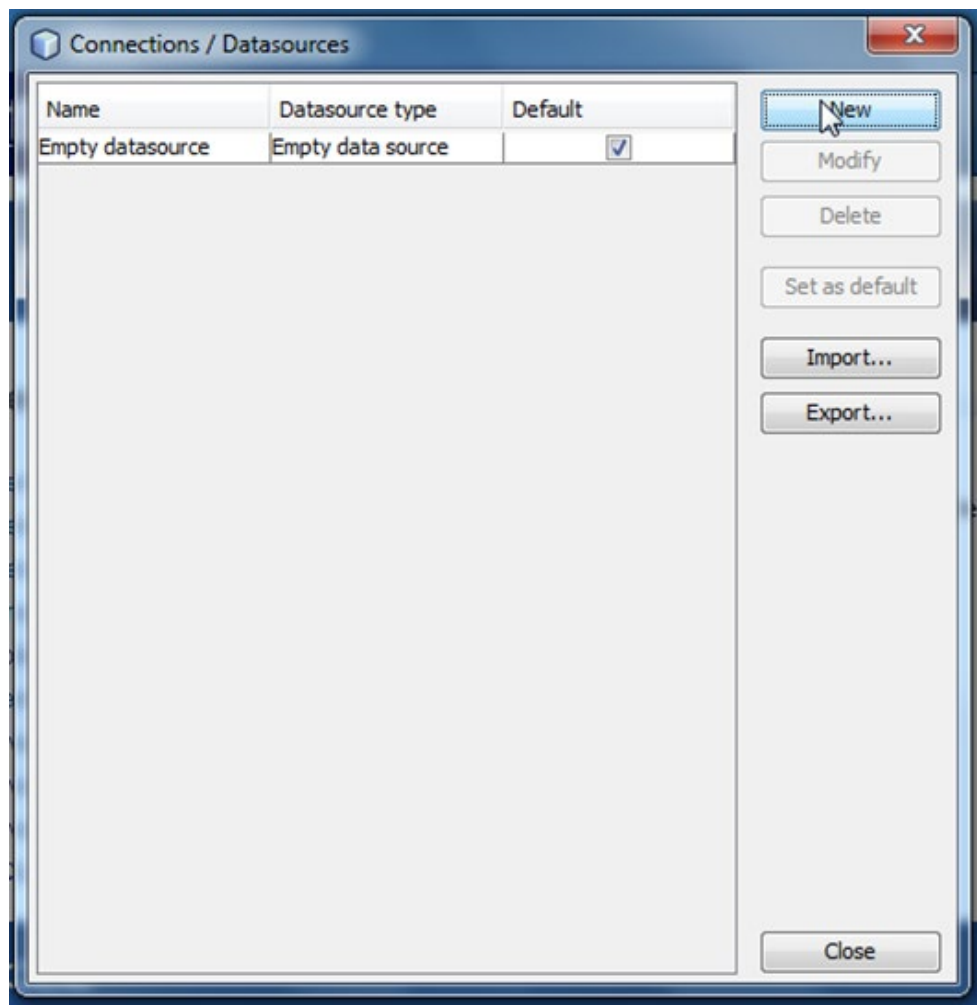


Figura 130. Agregando la conexión de la base de datos.
Fuente: SENA

Se selecciona **Database JDBC connection** y se hace clic en el botón **Next**.

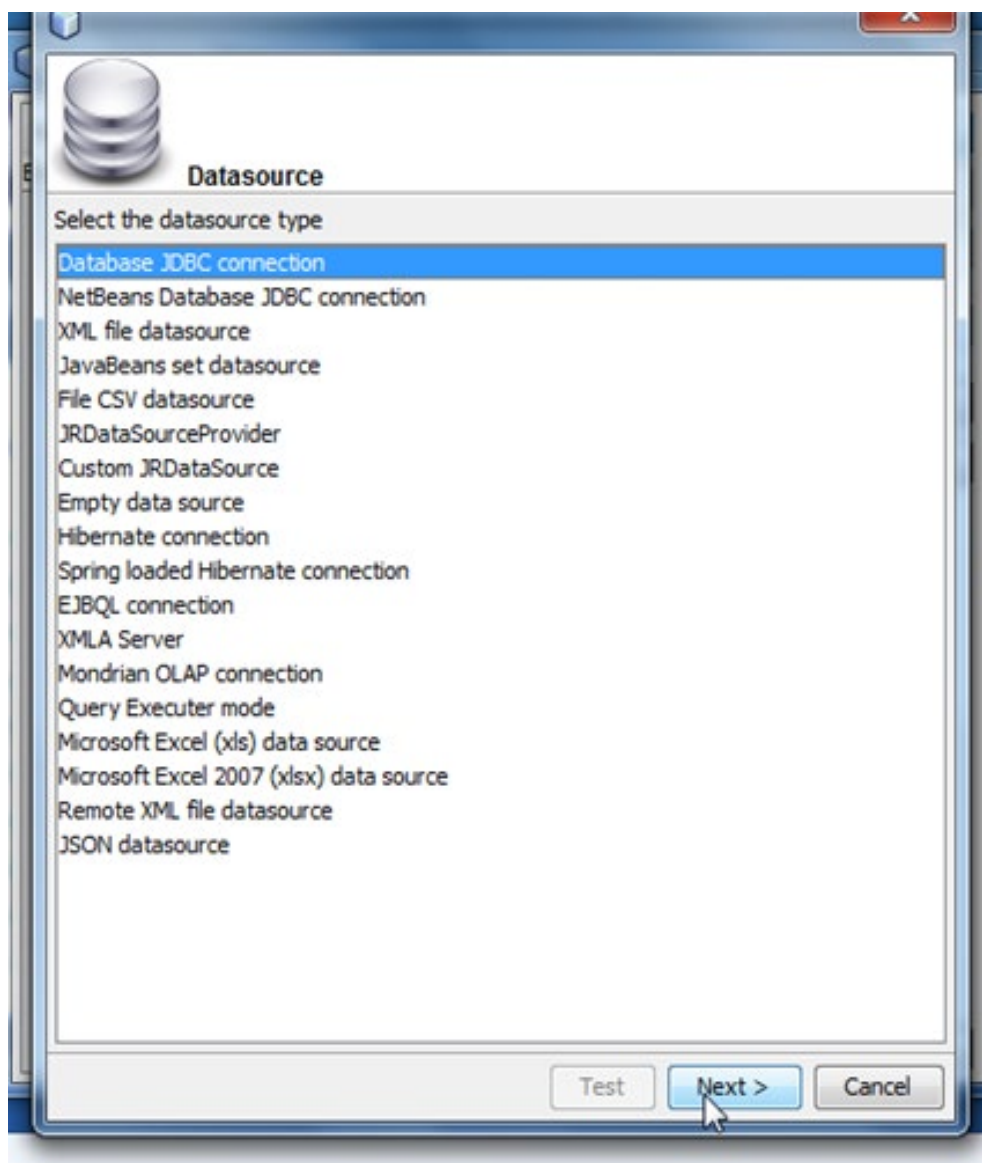


Figura 131. Agregando el Data Source.
Fuente: SENA

En la siguiente ventana, se ingresa un nombre para la conexión que puede ser cualquiera, en este caso **ReportesCitas**. Se selecciona el JDBC Driver de Oracle Oracle (**oracle.jdbc.driver.OracleDriver**), se ajusta el JDBC URL **jdbc:oracle:thin:@localhost:1521:XE**.

Se ingresan el **Username y Password** citas y se hace clic en el botón **Test**, luego de esto el sistema preguntará nuevamente el password, se ingresa la clave citas, posterior a esto aparecerá que la conexión fue exitosa **Connection test successful**

Finalmente se hace clic en el botón **Save** para guardar la conexión creada.

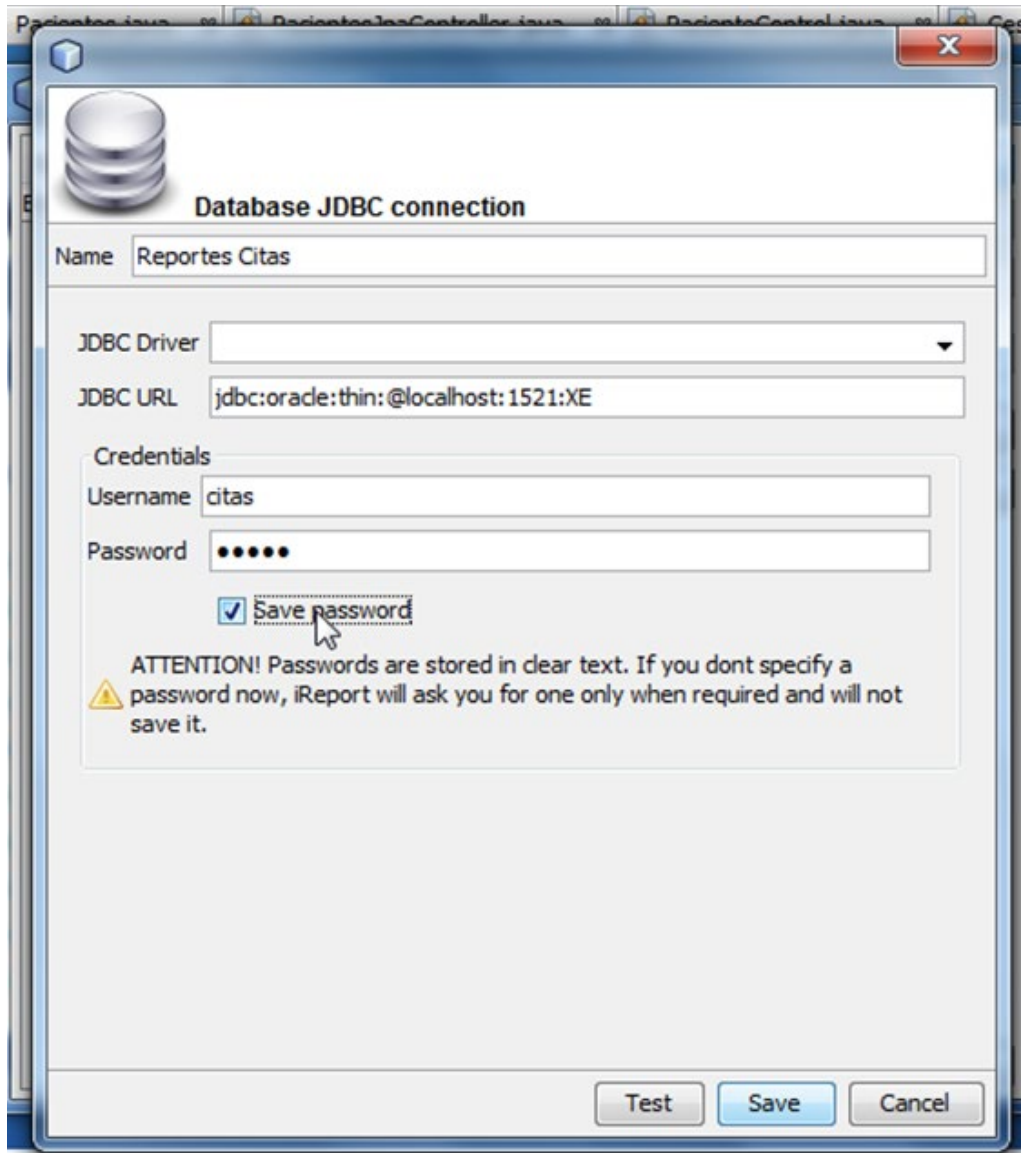


Figura 132. Realizando la conexión.

Fuente: SENA

NOTA. Se observa que en el **JDC URL** se cambia al final **MYDATABASE** por **XE**, indicando con ello que se conectará a una base de datos Oracle Express.

Si al momento de seleccionar el **JDBC Driver** la opción de Oracle aparece en color rojo, esto significa que se debe asociar el driver de Oracle en **iReport**, esto se hace siguiendo el menú **tools -> options**, seleccionando la pestaña **iReport** y dentro de esta la ficha **Classpath** y allí seleccionar el archivo **jar** del driver Oracle **ojdbc7.jar** o **ojdbc14.jar**.

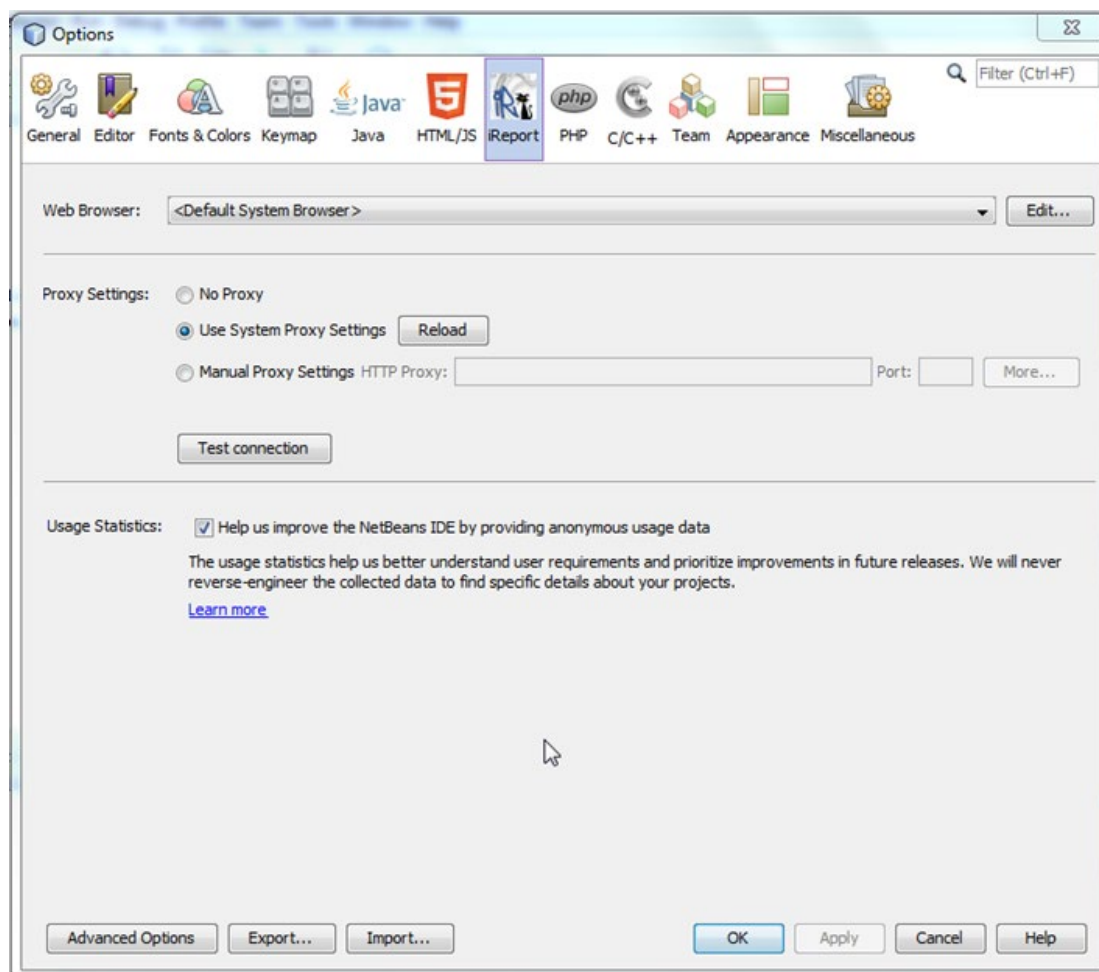


Figura 133. Asociando el driver al iReport.

Fuente: SENA

Una vez creada la conexión desde iReport con base de datos, se agregan algunas librerías adicionales al proyecto, estas librerías son necesarias para que el reporte finalmente pueda ser visualizado e impreso a través de algún medio externo como una impresora.

Las librerías adicionales son las siguientes:



Estas librerías pueden ser descargadas desde internet.

Figura 134. Librerías Adicionales.

Fuente: SENA

Una vez descargadas las librerías, se agregan al proyecto como se ha hecho antes con otras librerías, se hace clic derecho sobre la carpeta **libraries** y se selecciona del menú contextual la opción **Add JAR/Folder**, se selecciona y se agrega.

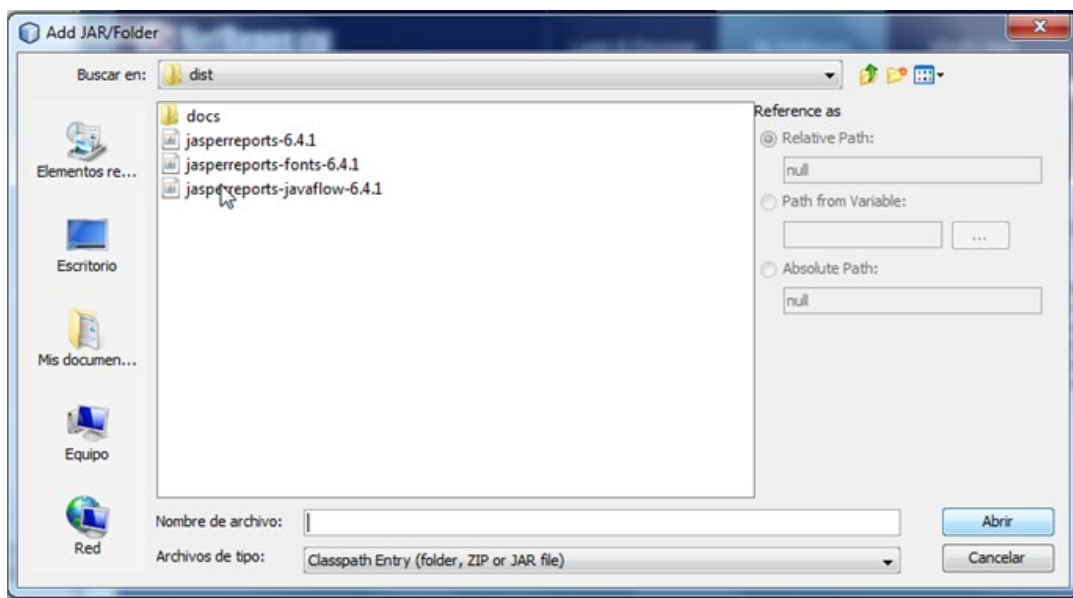


Figura 135. Agregando las librerías al proyecto.

Fuente: SENA

Se puede observar que las librerías quedan formando parte del proyecto.

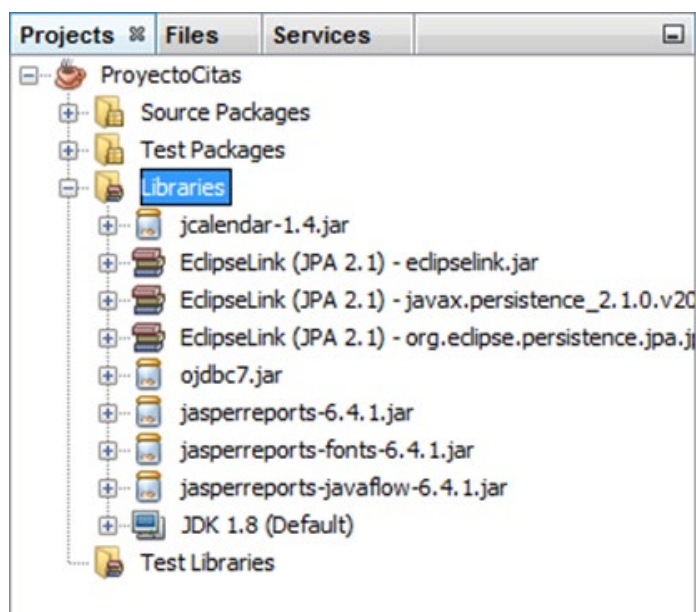


Figura 136. Visualizando las librerías agregadas.

Fuente: SENA

Después de agregar los componentes, probar la conexión con la base de datos y agregar las librerías necesarias, es hora de crear el reporte. Dentro del proyecto se crea un nuevo paquete llamado reportes, se hace clic derecho sobre el mismo y se selecciona del menú contextual la opción **New -> Other**.

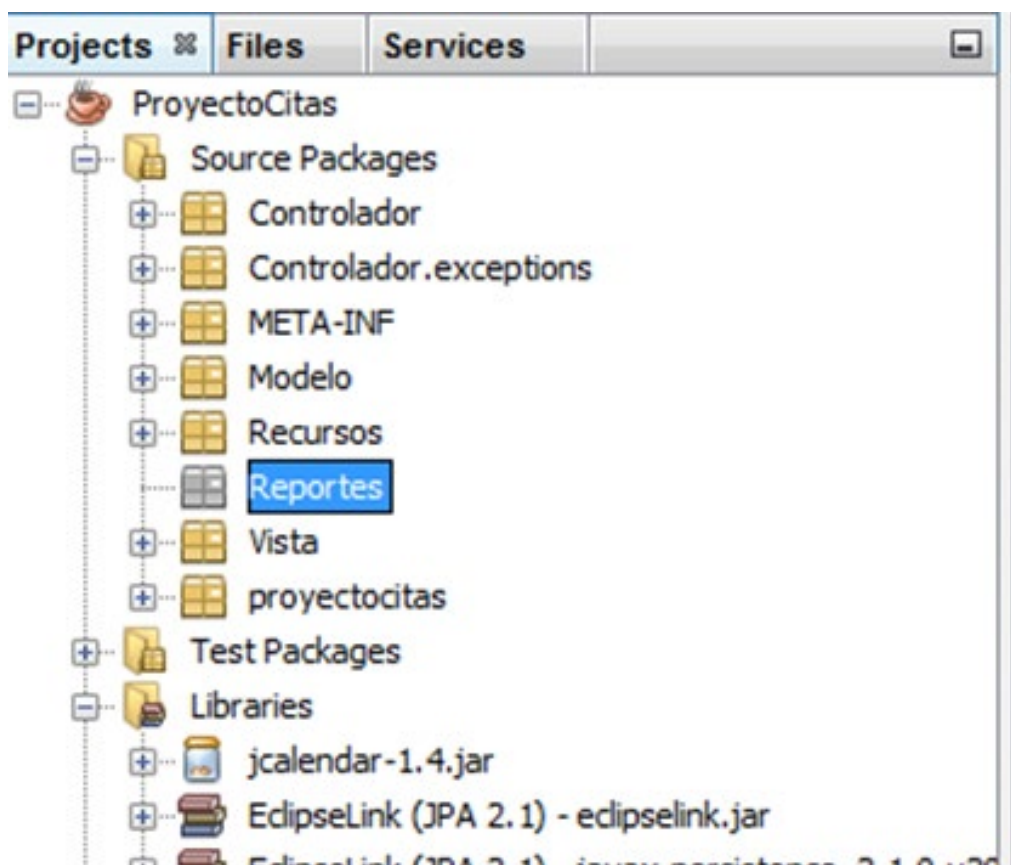


Figura 137. Creando el paquete Reportes.
Fuente: SENA

De la ventana emergente, se selecciona la categoría **Report** y el tipo de archivo **Report Wizard** y se hace clic en **Next**.

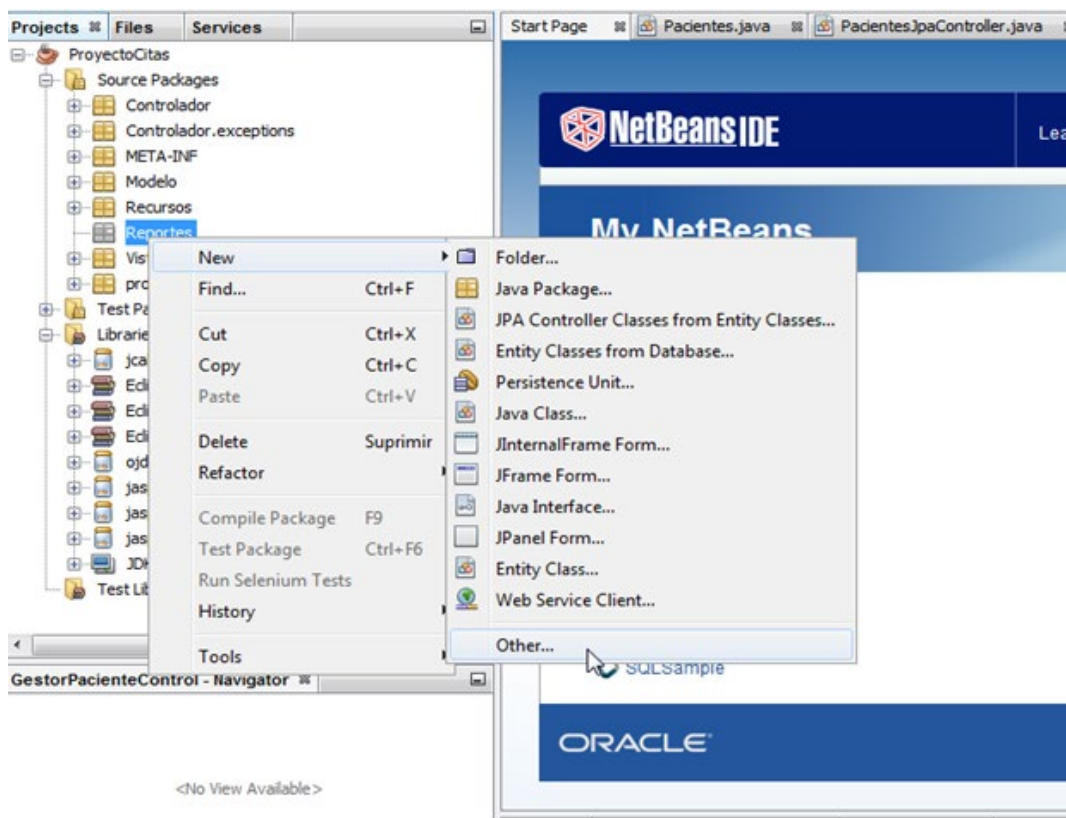


Figura 138. Agregando otro tipo de componente.
Fuente: SENA

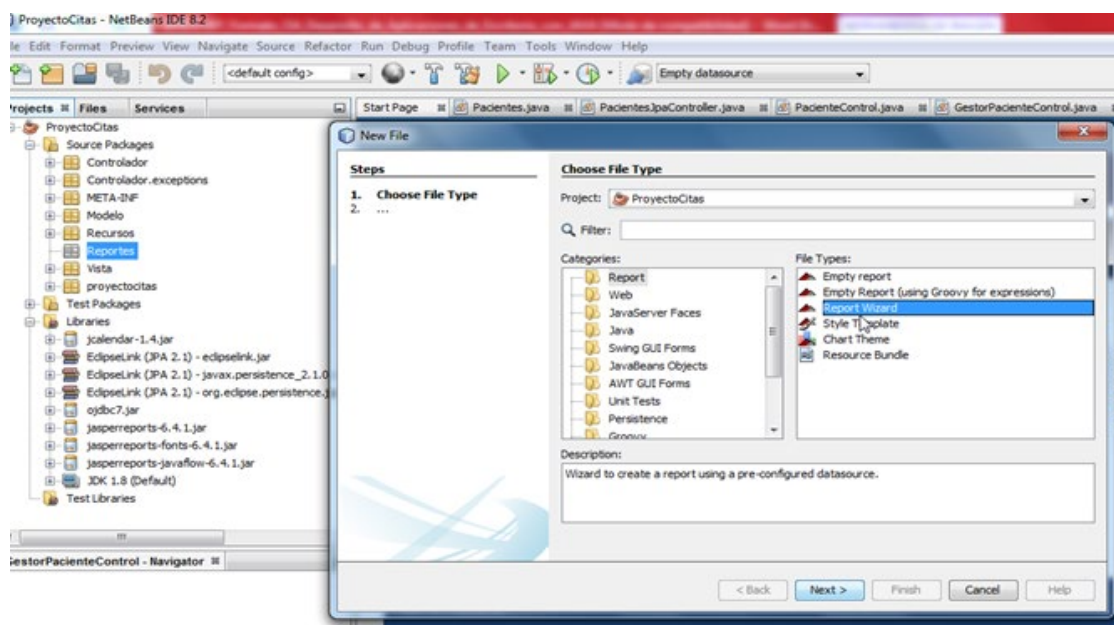


Figura 139. Agregando el Report Wizard.
Fuente: SENA

Se ingresa un nombre para el reporte **PacientesReport** y se hace clic en el botón Next.

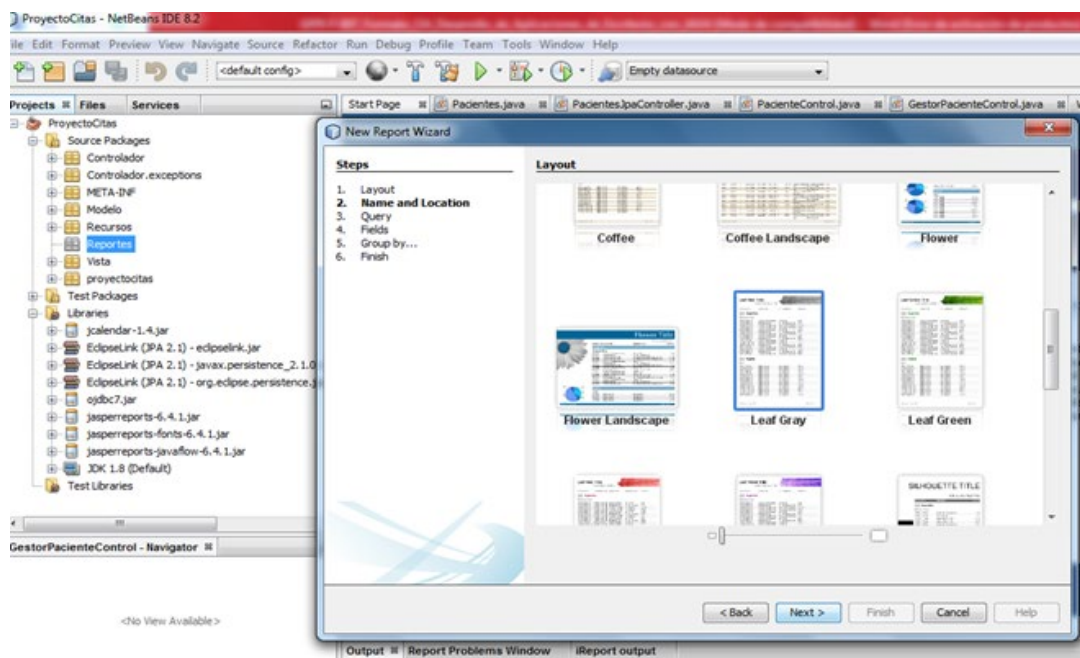


Figura 140. Seleccionando el estilo del reporte.

Fuente: SENA

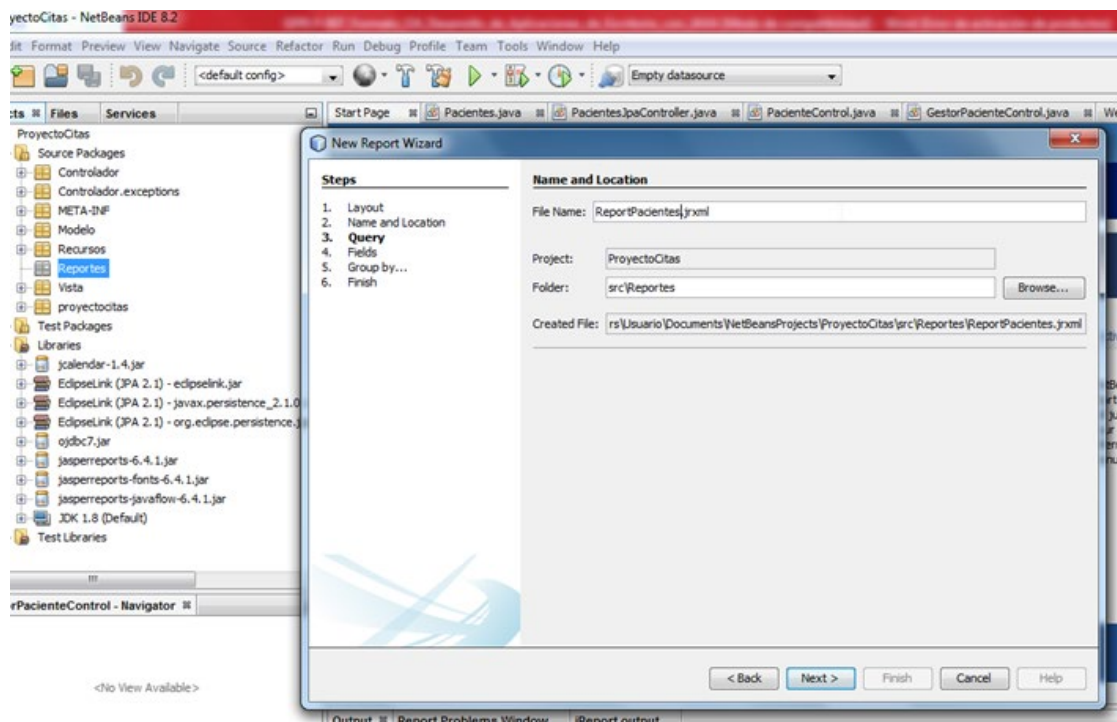


Figura 141. Agregando la conexión de la base de datos.

Fuente: SENA

Una vez creado el formato se procede a establecer la conexión con la base de datos. Para ello se selecciona en la opción **fields** la opción **new** de nueva conexión.

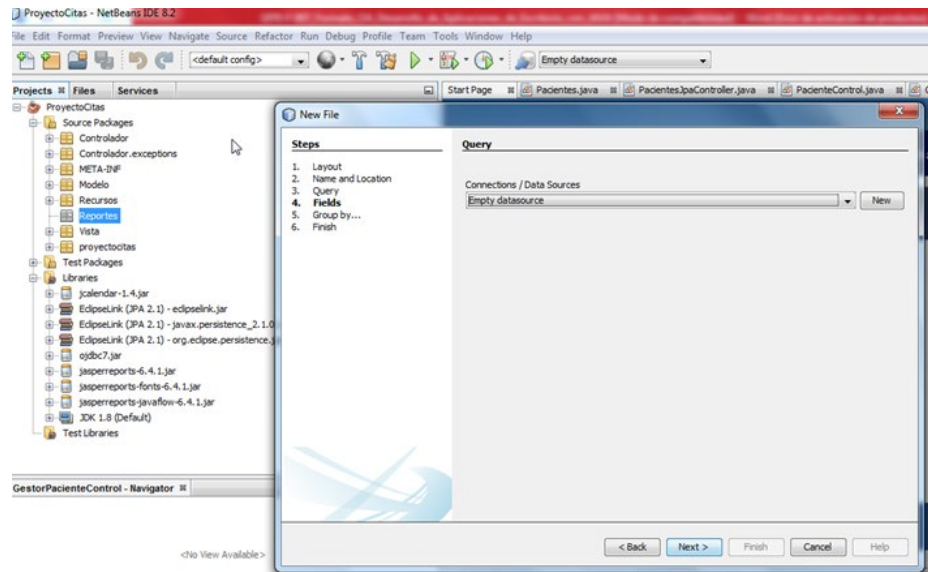


Figura 142. Agregando la conexión de la base de datos.
Fuente: SENA

Al seleccionar nuevo se despliega una interface de donde se selecciona la opción de **Netbeans Database JDBC Connection**.

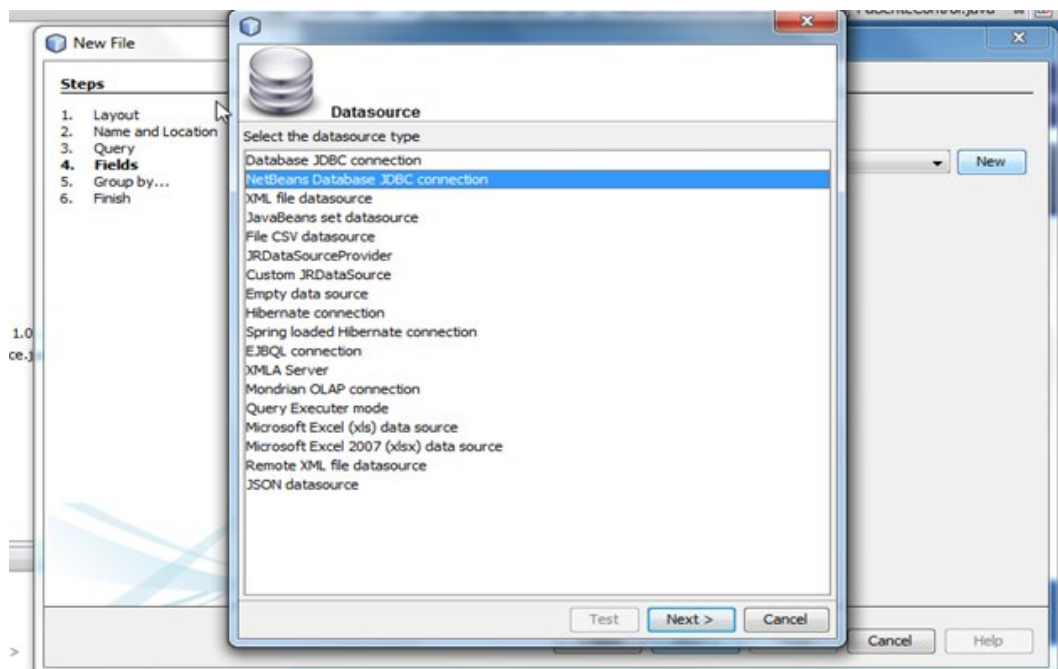


Figura 143. Agregando la conexión de la base de datos.
Fuente: SENA

En la siguiente pantalla se debe mostrar la conexión ya establecida en la sesión de conexión de bases de datos **jdbc**

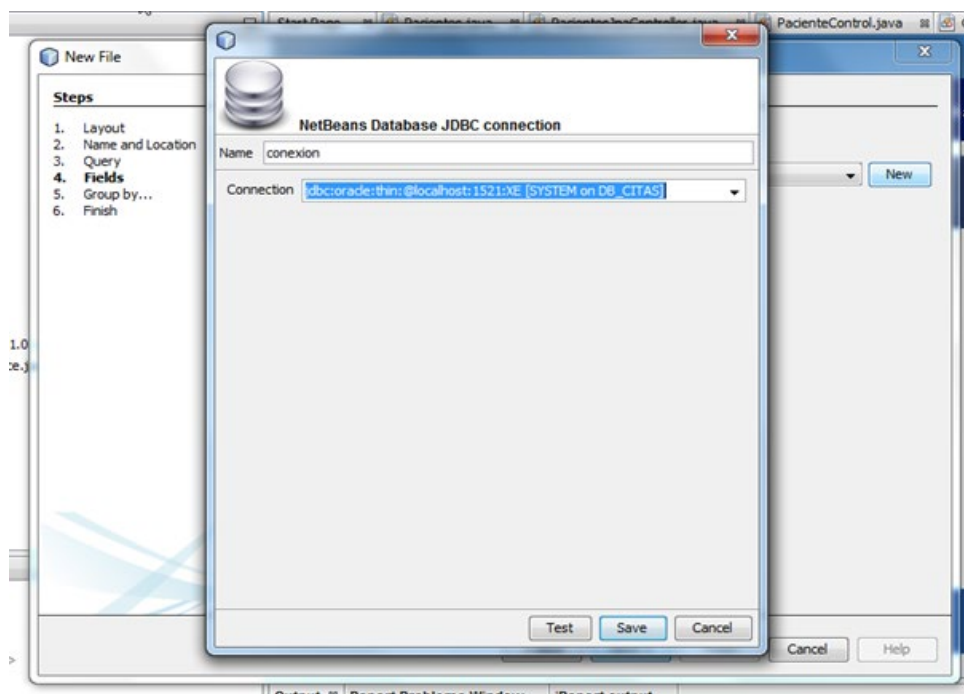


Figura 144. Agregando la conexión de la base de datos.

Fuente: SENA

Se hace clic en la opción test y se verifica la conexión.

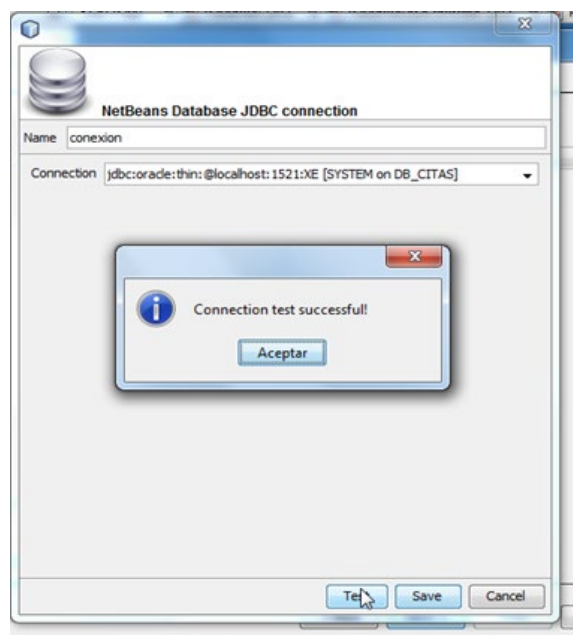


Figura 145. Probando la conexión.

Fuente: SENA

En la siguiente pantalla se selecciona la conexión anteriormente creada **reportescitas** y se escribe la consulta `SELECT*FROM PACIENTES order by PacApellidos`. Ya que en el reporte se desea mostrar la información de todos los pacientes registrados en la base de datos.

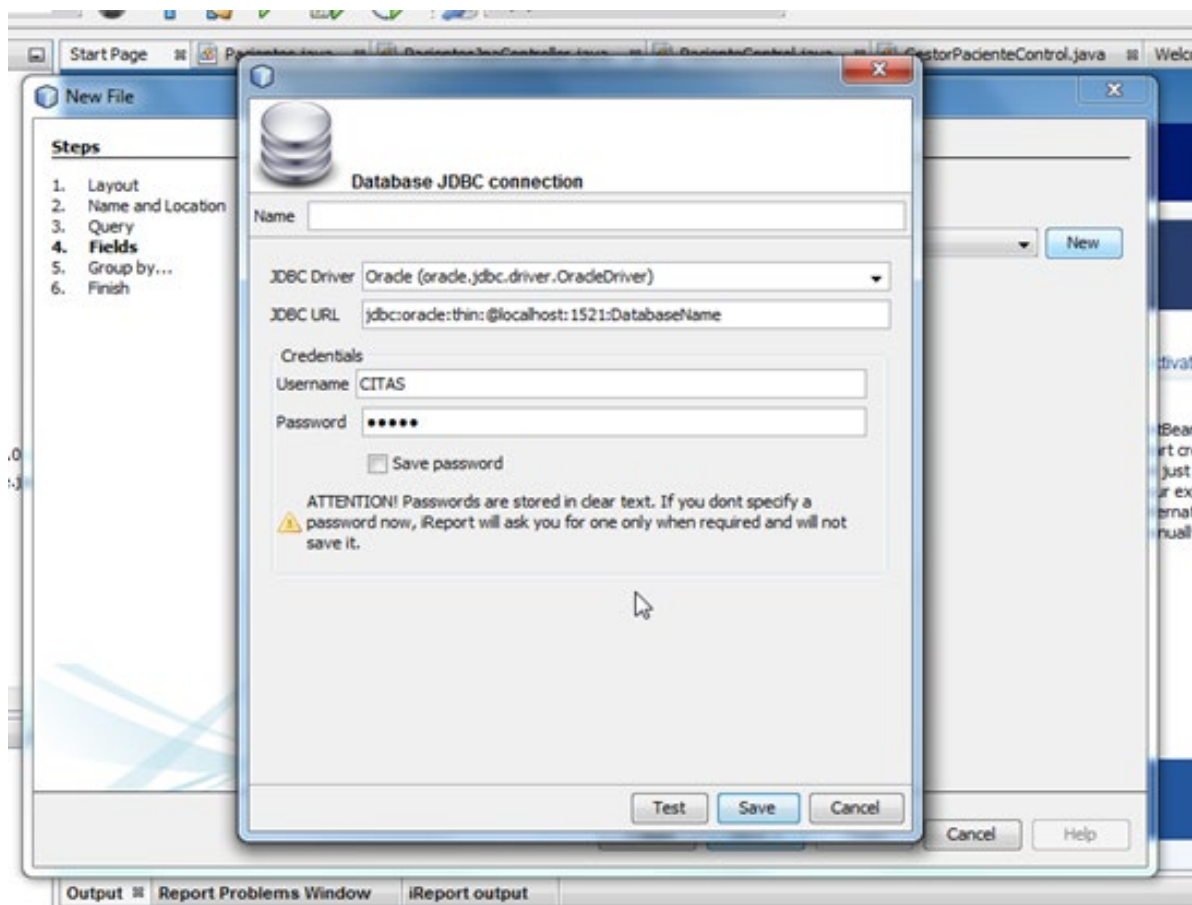


Figura 146. Probando la conexión.

Fuente: SENA

Se hace clic en el botón **Next**.

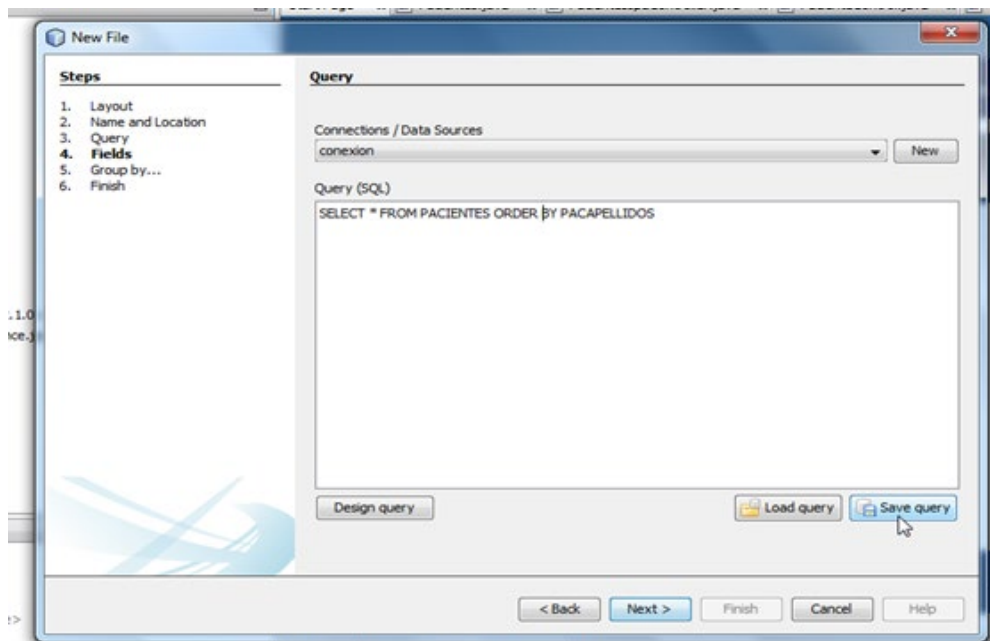


Figura 147. Incluyendo los datos en la conexión.
Fuente: SENA

En la siguiente ventana se seleccionan todos los campos que se desean presentar en el reporte, para este caso todo, se hace clic en el botón **Next**.

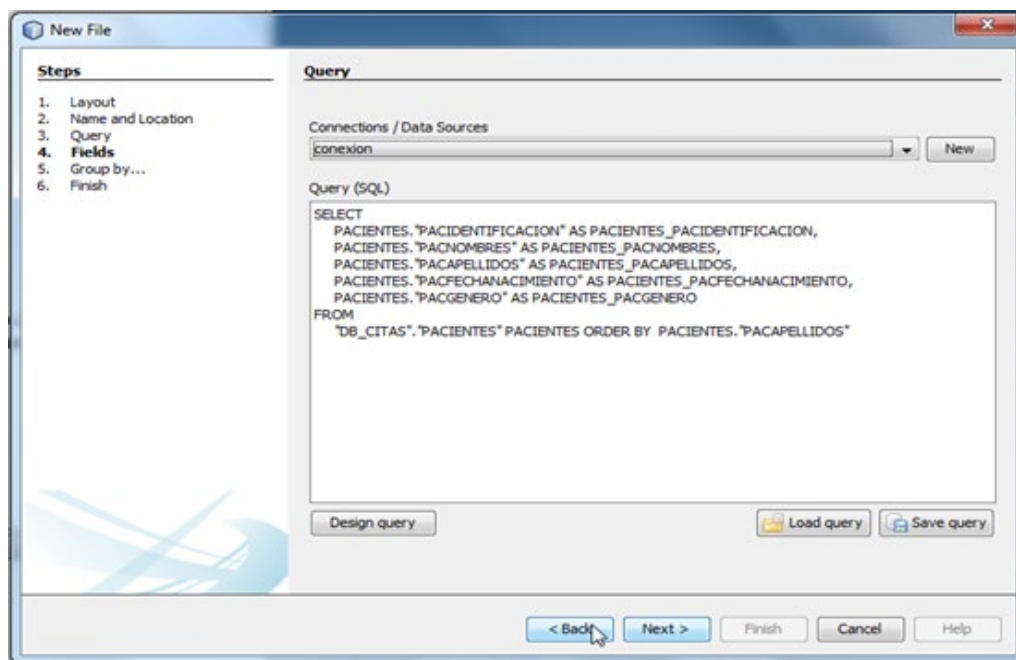


Figura 148. Incluyendo los datos en la conexión.
Fuente: SENA

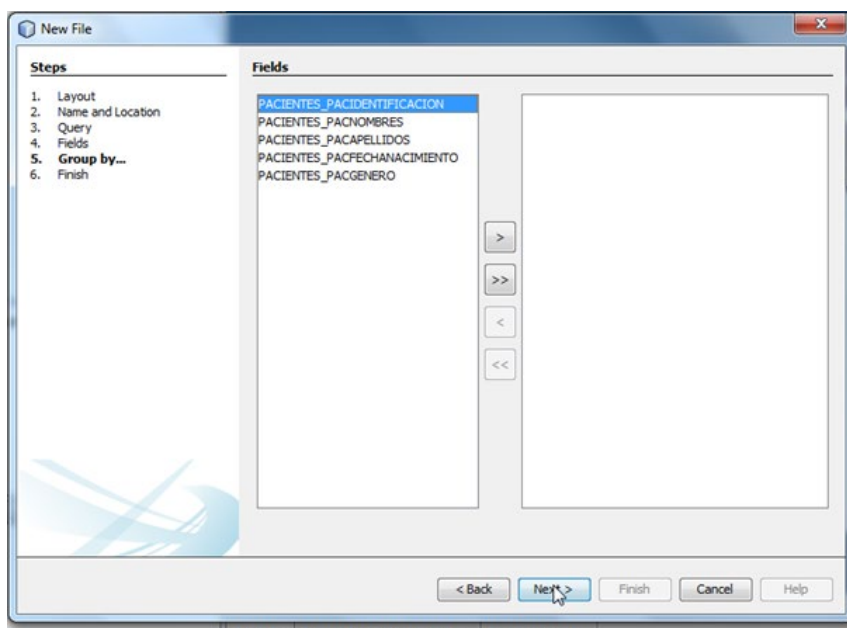


Figura 149. Incluyendo los datos en la conexión.
Fuente: SENA

En la ventana siguiente simplemente, se hace clic en **Next** ya que en este reporte no se desea agrupar por ningún campo.

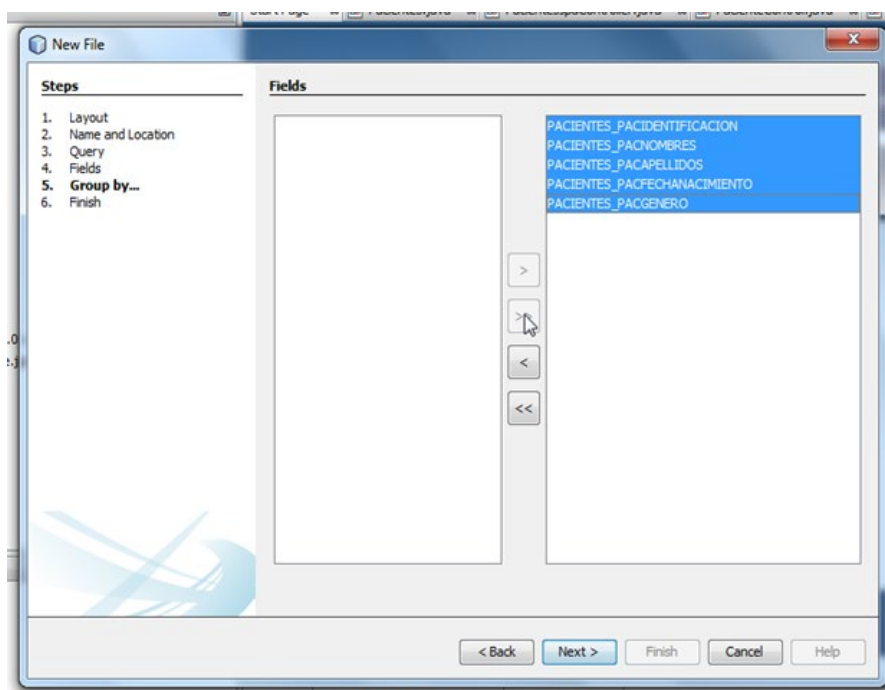


Figura 150. Incluyendo los datos en la conexión.
Fuente: SENA

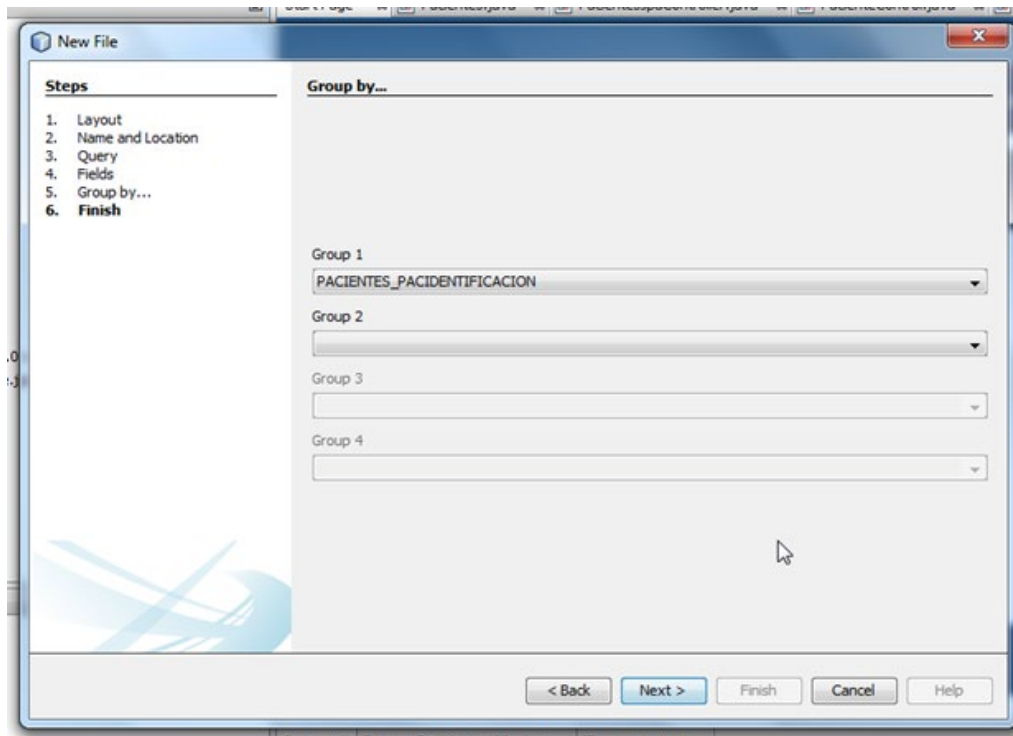


Figura 151. Incluyendo los datos en la conexión.
Fuente: SENA

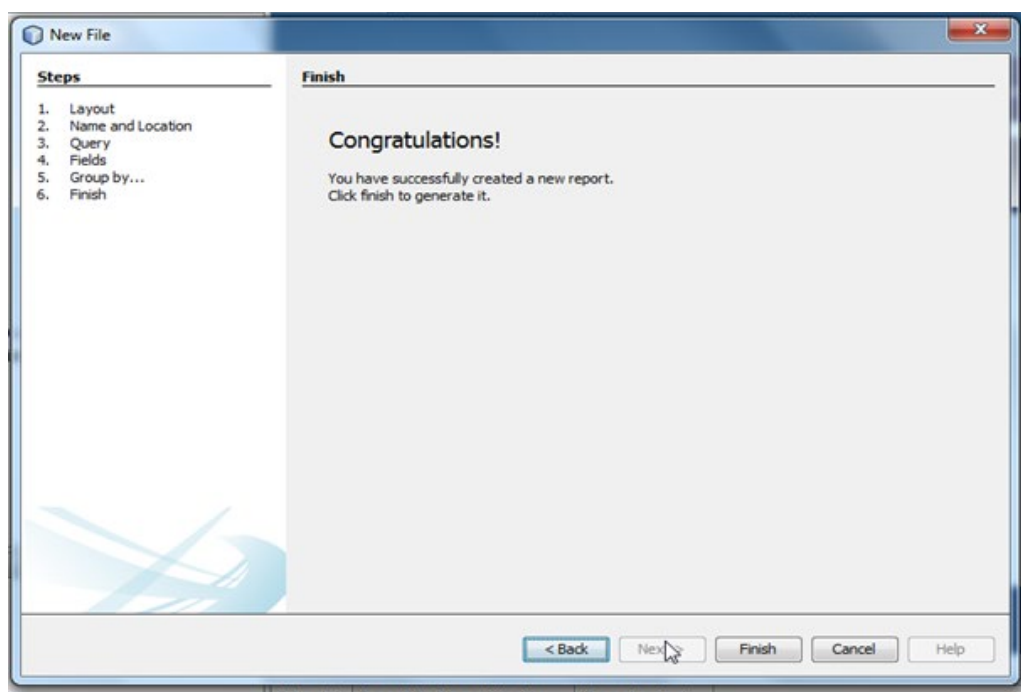


Figura 152. Finalizando la generación del reporte.
Fuente: SENA

Para verificar el estado parcial del reporte se hace clic en el botón **Preview**.

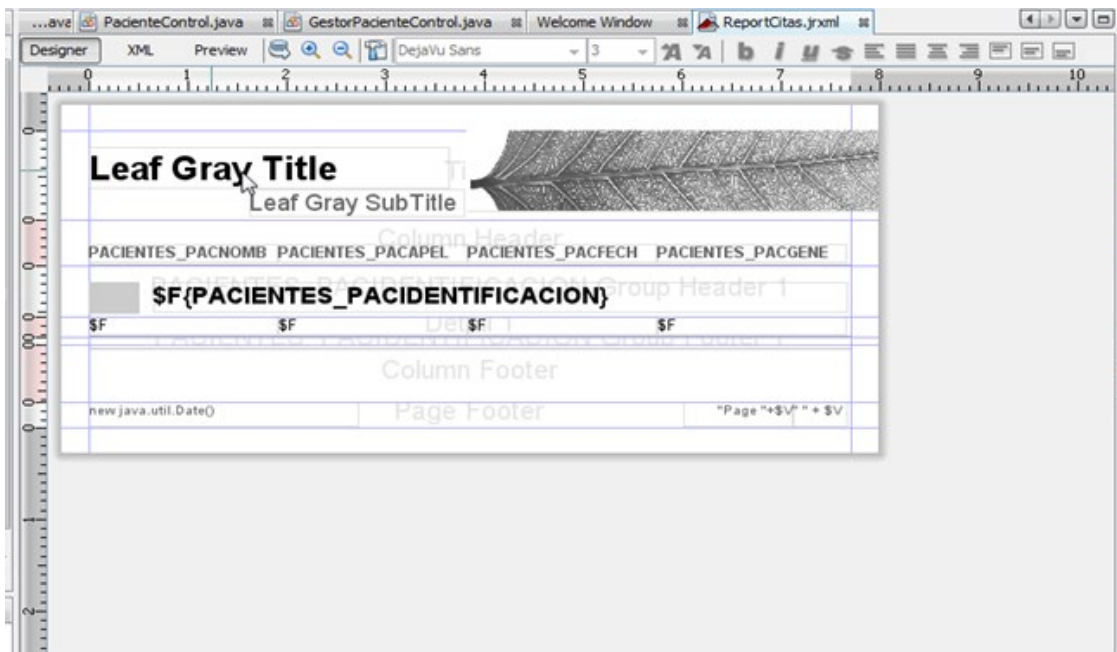


Figura 153. Preview del diseño del reporte.

Fuente: SENA

Haciendo clic en el botón **Designer** se puede personalizar el diseño del reporte, para este ejemplo, solamente se cambia el título del reporte y el título de las columnas.

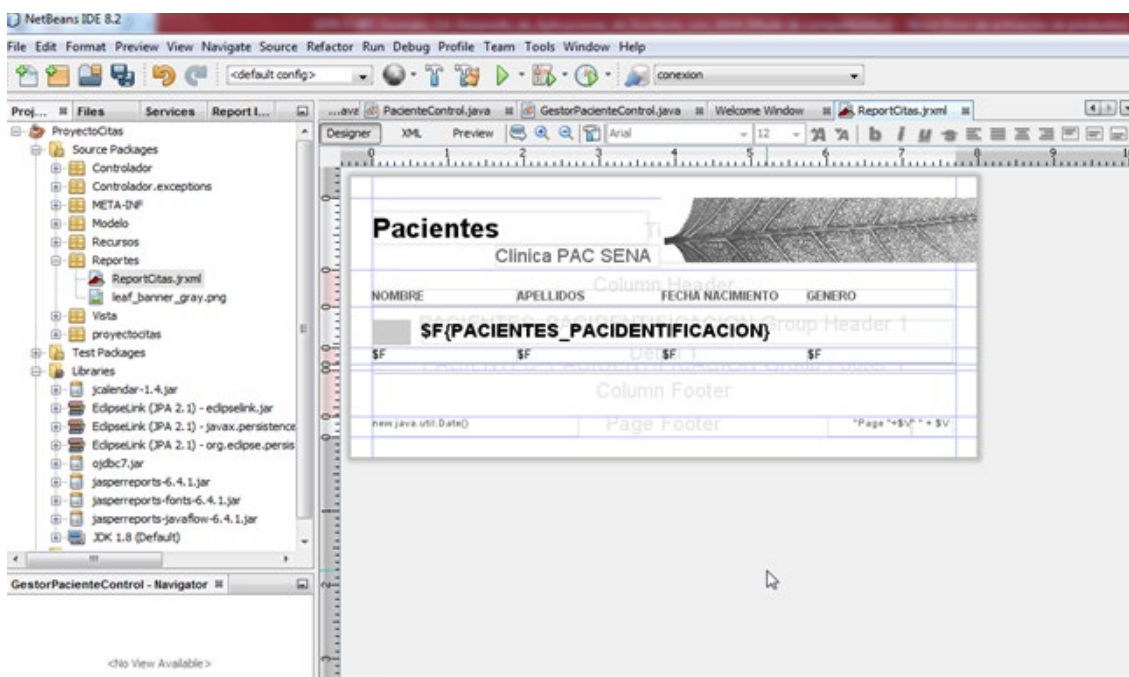


Figura 154. Modificando el reporte.

Fuente: SENA

Luciendo finalmente se genera el reporte.

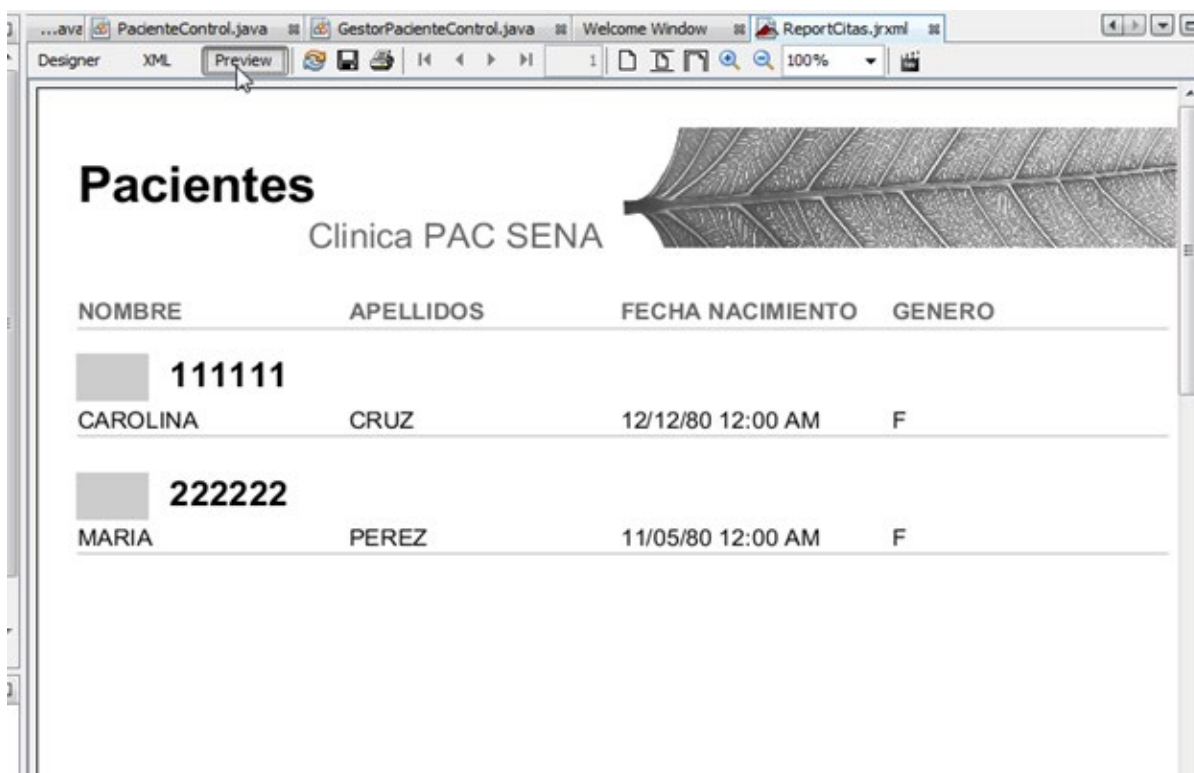


Figura 155. Desplegando el reporte.
Fuente: SENA

Al tener el diseño del reporte finalizado y probado de manera independiente, es el momento de enlazarlo con la aplicación, para ello, lo primero que se hace es crear una clase dentro del paquete reportes que se encargue de gestionar y mostrar los reportes. Hacer clic derecho sobre el paquete reportes y seleccionar del menú contextual la opción **new- Java Class**.

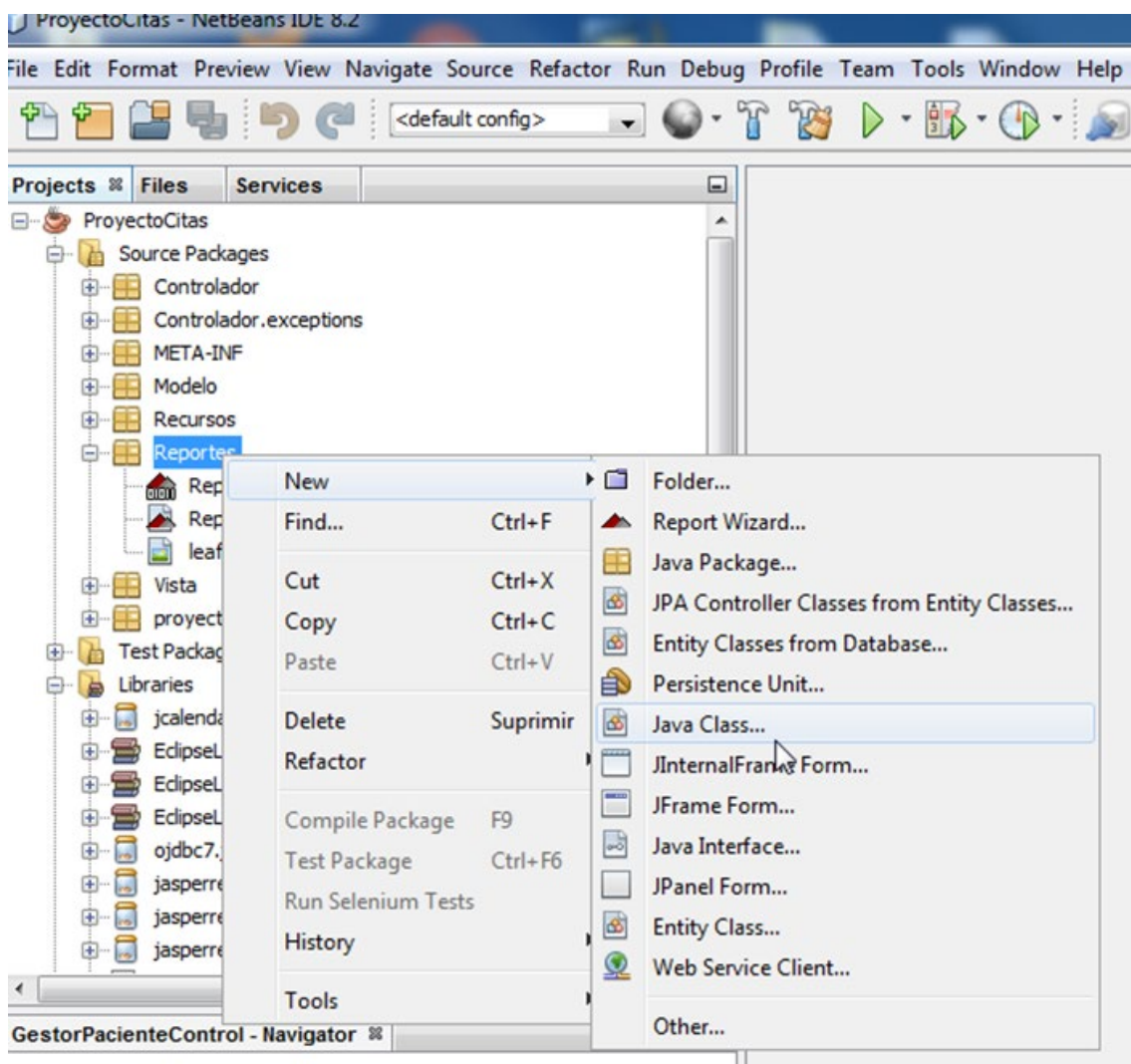


Figura 156. Creando la clase gestión reporte.
Fuente: SENA

Se asigna el nombre **GestorReportes** y se hace clic en el botón **Finish**.

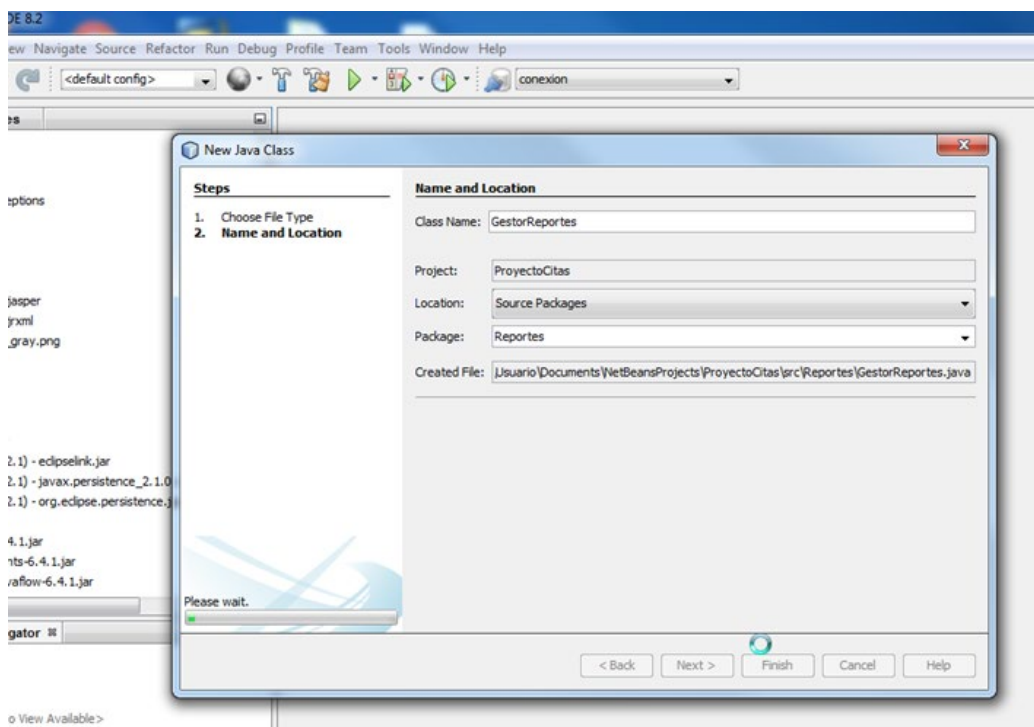


Figura 157. Creando la clase gestor reporte.
Fuente: SENA

El código de esta clase es el siguiente:

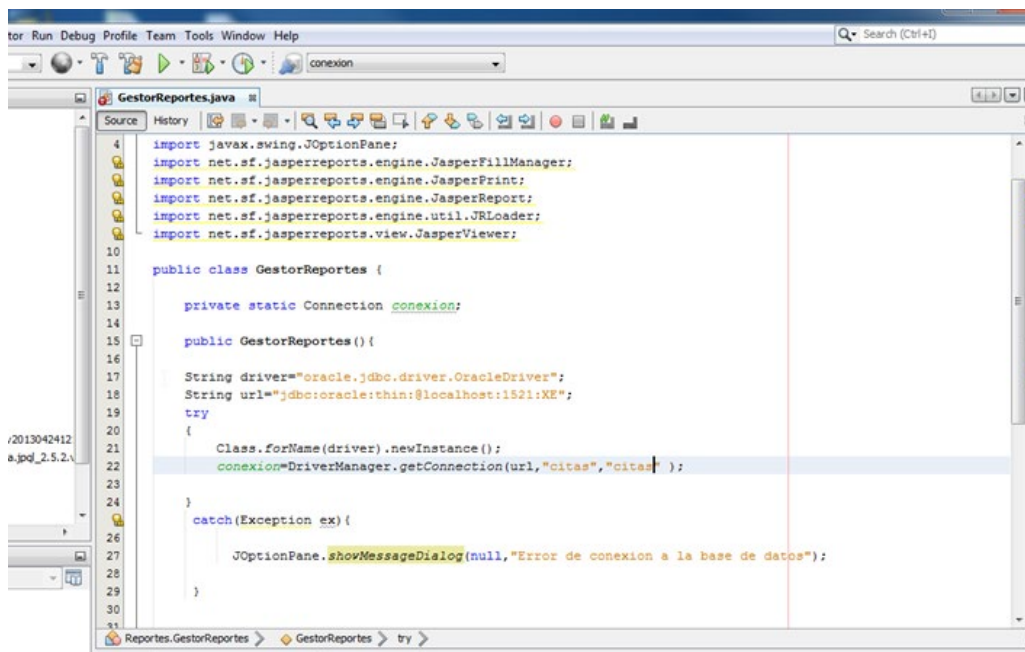


Figura 158. Código clase gestor reporte.
Fuente: SENA

Explicación del código.

Tabla 18

Explicación del código.

Línea	Explicación
5..9	Importación de paquetes para la gestión y visualización de los reportes.
15..28	Constructor de la clase, donde se crea la conexión con la base de datos, la cual queda almacenada en la variable conexión.

Nota: Producción SENA

El código de esta clase es el siguiente:

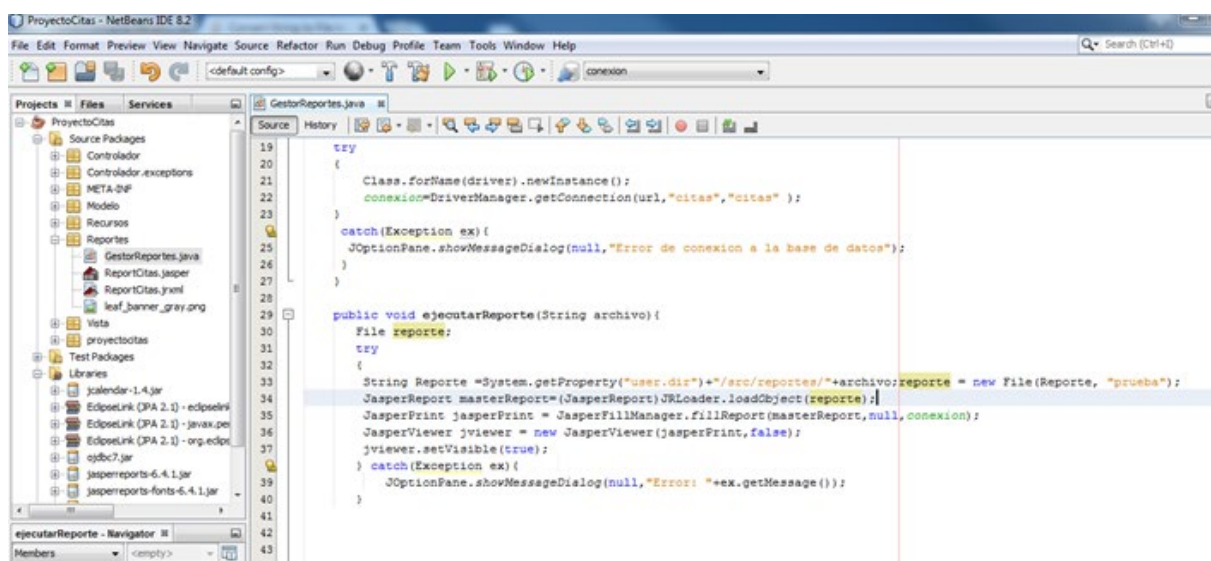


Figura 159. Código de la clase gestor reporte.

Fuente: SENA

Explicación del código.

Tabla 19

Explicación del código.

Línea	Explicación
33	Se obtiene la ubicación y el nombre del reporte que se desea presentar.
34	Carga del reporte seleccionado.
35	Se llenan los datos en el reporte con la información obtenida de la base de datos y de acuerdo a algún parámetro, en este caso no existen parámetros null.
36	Se llama al visor del reporte, asignando el reporte y sus datos.
37	Se hace visible el reporte.

Nota: Producción SENA

Después de crear la clase **GestorReportes** con su Método **EjecutarReporte** se debe modificar la vista para que desde el menú archivo del formulario principal se tenga la opción de imprimir el listado de pacientes.

Se abre el formulario **PrincipalJFramey**, al menú **Archivo** se le agrega un nuevo menú ítem con el título **Imprimir Pacientes Registrados**.

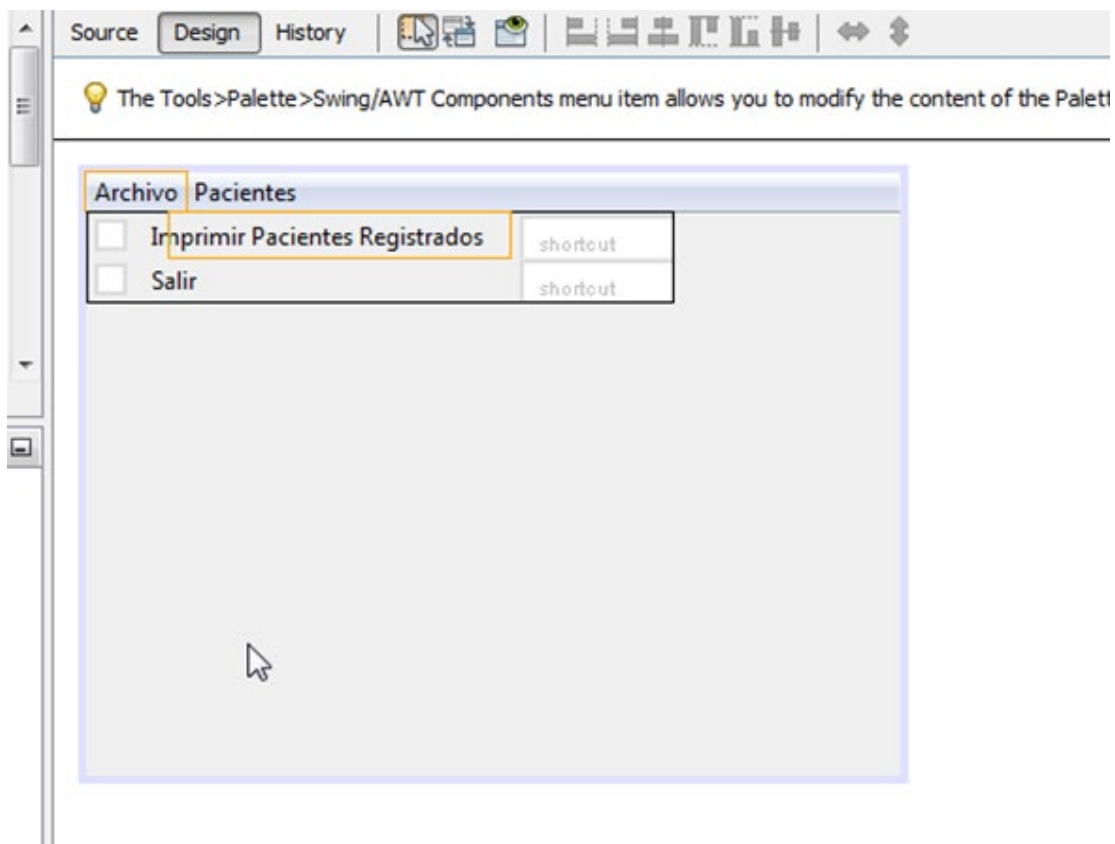


Figura 160. Creando la opción del menú para el reporte.
Fuente: SENA

Se hace clic derecho en el menú ítem **Imprimir Pacientes Registrados** y del menú contextual se selecciona la opción **event-action-actionperformed**.

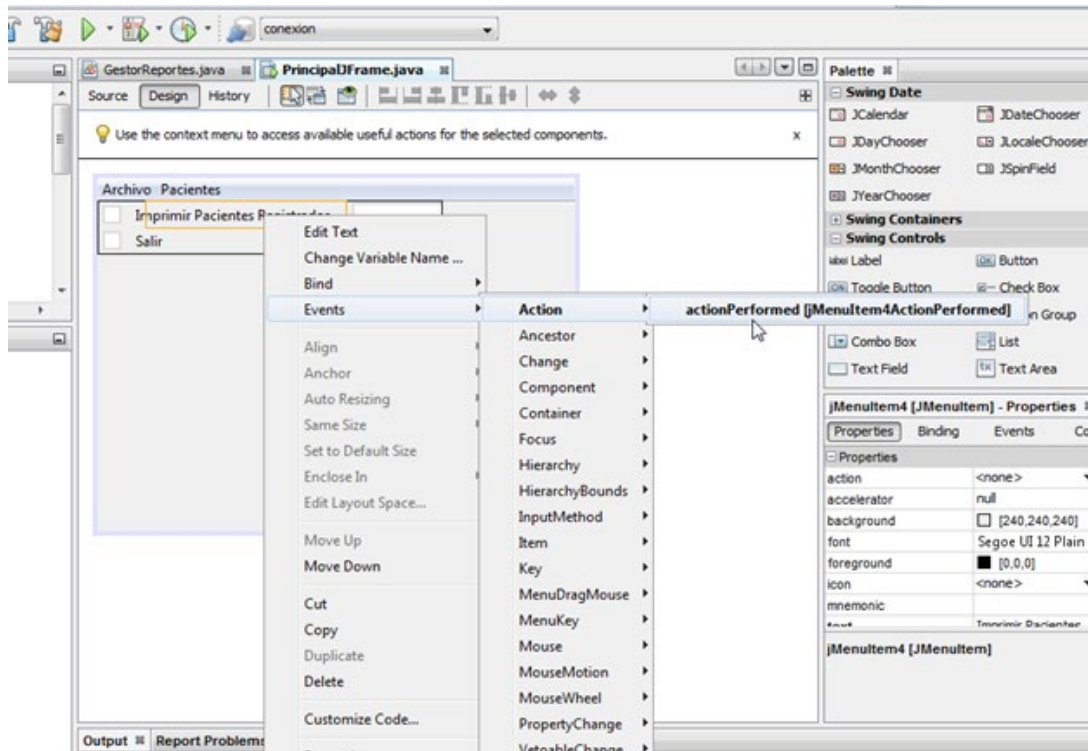


Figura 161. Programando el evento action performed.

Fuente: SENA

De esta forma se llega al evento, allí se escribe el siguiente código.

```

108
109 private void jMenuItem4ActionPerformed(java.awt.event.ActionEvent evt) {
110     // TODO add your handling code here
111     Reportes.GestorReportes gr=new Reportes.GestorReportes();
112     gr.ejecutarReporte("PaciesReport.jasper");
113
114 }
115

```

Figura 162. Código evento action performed.

Fuente: SENA

En el código anterior se instancia la clase de **GestorReportes** invocando posteriormente al método **EjecutarReporte**, donde se envía el nombre del reporte, que se desea ejecutar.

Ahora se prueba la aplicación y al hacer clic en el menú Imprimir.

Pacientes

Clinica PAC SENA



NOMBRE	APELLIDOS	FECHA NACIMIENTO	GENERO
 111111			
CAROLINA	CRUZ	12/12/80 12:00 AM	F
 222222			
MARIA	PEREZ	11/05/80 12:00 AM	F

Figura 163. Resultados del reporte.
Fuente: SENA

GLOSARIO

Back-End: hace referencia a las estructuras de almacenamiento de datos en un sistema de información.

Front-End: hace referencia a las interfaces que permiten la comunicación con los usuarios de un sistema de información.

JDBC: Java DataBase Connectivity: estándar de Java para la conexión y gestión de bases de datos relacionales.

JPA: Java Persistence Api: mecanismo en Java que permite la persistencia de los datos a partir del mapeo o vínculo entre tablas de la base de datos y clases de la aplicación.

MVC: Modelo Vista Controlador. Patrón de diseño que divide una aplicación en capas, facilitando el mantenimiento y la adaptabilidad de una aplicación.

Patrón: en informática un patrón es una buena práctica para la solución de un problema recurrente.

SGBD: Sistema Gestor de Bases de Datos.

BIBLIOGRAFÍA

Harvey M. Deitel, Paul J. (2004). *Deitel. Como programar en JAVA (2004)*. Pearson Educación, México.

Netbeans org IDE (2017). *Características IDE de Netbeans*. Recuperado de <https://netbeans.org/features/index.html>

CONTROL DEL DOCUMENTO

CONSTRUCCIÓN OBJETO DE APRENDIZAJE



DESARROLLO DE APLICACIONES DE ESCRITORIO CON LENGUAJE JAVA

Centro Industrial de Mantenimiento Integral - CIMI
Regional Santander

Líder línea de producción:	Santiago Lozada Garcés
Asesores pedagógicos:	Rosa Elvia Quintero Guasca Claudia Milena Hernández Naranjo
Líder expertos temáticos:	Rita Rubiela Rincón Badillo
Experto temático:	Andrés Julián Valencia Osorio (V1)
Experto temático:	Edward José Beltrán Lozano (V2)
Diseño multimedia:	Catalina Martínez Ávila
Programador:	Francisco José Lizcano Reyes
Producción de audio:	Víctor Hugo Tabares Carreño

**creative
commons**



Este material puede ser distribuido, copiado y exhibido por terceros si se muestra en los créditos. No se puede obtener ningún beneficio comercial y las obras derivadas tienen que estar bajo los mismos términos de la licencia que el trabajo original.



Registered trademark

Oracle, Netbeans y Java son marcas comerciales registradas de Oracle y / o de sus filiales. Otros nombres pueden ser marcas comerciales de sus respectivos propietarios.