

# Tarea 7 para Acceso a Datos

```
PruebaAlumnoBean - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects Files Services Start Page AccedeBD.java MatriculaBean.java Main.java
Source History <default config>
Output - PruebaAlumnoBean (run)
run:
Todas las matriculas
-----
Alumno 1
DNI:12345678A
Nombre del modulo: Acceso a Datos
Curso: 23-24
Nota: 8.5
Alumno 2
DNI:12345678B
Nombre del modulo: Desarrollo de Interfaces
Curso: 23-24
Nota: 9.0
Alumno 3
DNI:12345678C
Nombre del modulo: Programación de Servicios y Procesos
Curso: 23-24
Nota: 8.0
Alumno 4
DNI:12345678F
Nombre del modulo: Sistemas de Gestión Empresarial
Curso: 23-24
Nota: 9.2
Se han cargado todas las matriculas
-----
Solo la matricula del DNI 12345678A
-----
Alumno con DNI:12345678A
Nombre del modulo: Acceso a Datos
Curso: 23-24
Nota: 8.5
Se ha cargado la matricula del alumno con DNI 12345678A
-----
Adadir matricula a la base de datos
Se ha añadido una nueva matricula a la base de datos
BUILD SUCCESSFUL (total time: 3 seconds)
```

```
PruebaAlumnoBean - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects Files Services Start Page AccedeBD.java MatriculaBean.java Main.java
Source History <default config>
Output - PruebaAlumnoBean (run)
run:
Alumno 1
DNI:12345678A
Nombre del modulo: Acceso a Datos
Curso: 23-24
Nota: 8.5
Alumno 2
DNI:12345678B
Nombre del modulo: Desarrollo de Interfaces
Curso: 23-24
Nota: 9.0
Alumno 3
DNI:12345678C
Nombre del modulo: Programación de Servicios y Procesos
Curso: 23-24
Nota: 8.0
Alumno 4
DNI:12345678F
Nombre del modulo: Sistemas de Gestión Empresarial
Curso: 23-24
Nota: 9.2
Alumno 5
DNI:12345678G
Nombre del modulo: Programación Multimedia y Dispositivos Móviles
Curso: 23-24
Nota: 9.8
Se han cargado todas las matriculas
-----
Solo la matricula del DNI 12345678A
-----
Alumno con DNI:12345678A
Nombre del modulo: Acceso a Datos
Curso: 23-24
Nota: 8.5
Se ha cargado la matricula del alumno con DNI 12345678A
-----
BUILD SUCCESSFUL (total time: 1 second)
```

Diego Manuel Carrasco Castañares

Enunciado.

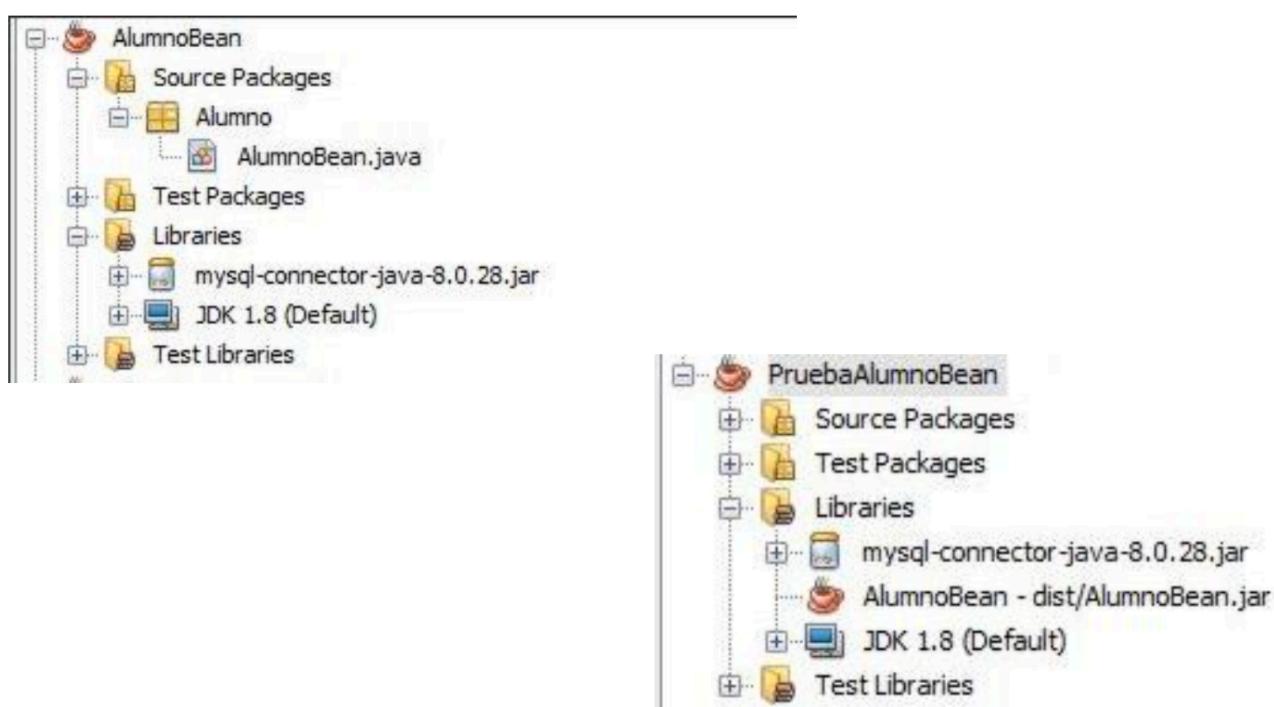
OBLIGATORIO Hay que entregar un documento en pdf, con explicación y los pantallazos de la ejecución de la tarea.

Los chicos de BK están aprendiendo a hacer componentes de acceso a datos. Están practicando con los datos de la matrícula de los alumnos de la base de datos con la que has estado trabajando durante esta unidad y necesitan que les eches una mano, en concreto te piden que hagas lo siguiente:

1.- Crea un script matricula.sql (mira en el apartado 11.1 de la unidad que se creó la base de datos alumnos) y adjúntalo al proyecto. Este script debe añadir una tabla llamada matriculas a la base de datos alumnos con 4 registros, que represente las matrículas de los alumnos . La tabla consta de los siguientes campos:

- DNI: varchar(9).
- NombreMódulo: varchar(60).
- Curso: varchar(5), el curso se forma con los dos años que lo componen separados por un guión, por ejemplo 23-24.
- Nota: double.

2.- Crea un paquete nuevo en el proyecto AlumnoBean (está en la unidad en el apartado 11.8 en el enlace Código de ejemplo) llamado Matricula para crear en él un componente nuevo llamado MatriculaBean. La imagen siguiente muestra el componente de ejemplo y la aplicación que lo maneja.



3.- Debes hacer lo siguiente

The screenshot shows a Java IDE interface with two projects:

- AlumnoBean** (selected):
  - Source Packages:
    - Alumno (containing AlumnoBean.java)
    - Matricula (containing MatriculaBean.java)
  - Test Packages
  - Libraries
  - Test Libraries
- PruebaMatriculaBean**:
  - Source Packages:
    - pruebamatriculabean (containing AccedeBDMatriculas.java and Main.java)
  - Test Packages
  - Libraries
  - Test Libraries

Below the projects, there is a file list:

- El mes pasado**
  - PruebaMatriculaBean (15/03/2024 16:49) - Carpeta de archivos
  - AlumnoBean (15/03/2024 16:49) - Carpeta de archivos
- hace mucho tiempo**
  - alumnos.sql (13/02/2012 20:26) - Archivo SQL (2 KB)

4.- En MatriculaBean el componente gestionará la información de la matrícula. Conexión a la base de datos. Además del código necesario para gestionar las propiedades del componente (getter y setter) y mantener la información de la base de datos en un vector interno, es preciso que incluyas los siguientes métodos (mira el componente AlumnoBean):

- o recargarFilas() para añadir los registros obtenidos de la tabla matriculas al vector
- o seleccionarFila(i): recupera en las propiedades del componente el registro número i del vector.
- o RecargarDNI(): recarga la estructura interna del componente con las matrículas de un DNI en particular.
- o AddMatricula(): añade un registro nuevo a la base de datos con la información almacenada en las propiedades del componente.
- o Dado que el componente puede funcionar en dos modos diferentes (todos los alumnos o un alumno concreto) se generará un evento cada vez que se cambie de modo, es decir, cuando se carguen todas las matrículas se lanzará un evento que lo señale y cuando se carguen las matrículas para un solo alumno también.

5.- Tendrás que crear un proyecto de prueba del componente en el que hagas un listado de todas las matrículas que hay en el sistema, y luego hagas un listado de las matrículas de un

alumno concreto. Listado de todos y también otro de un alumno (DNI).

- 6.- Cuando cargues la matricula del usuario concreto deberás capturar el evento generado al cambiar de modo. Un modo (variable) para listar todos los alumnos y otro modo para listar los del DNI 12345678A.
- 7.- Añade el código necesario para añadir una matrícula nueva a la base de datos.

Criterios de puntuación.

Total 10 puntos.

- 1.- Añadir tabla a la base de datos y rellenarla. El script .sql. 1 punto.
- 2.- Conexión 1 punto
- 3.- Adición de las propiedades de una matrícula con sus correspondientes getters y setters. 1 punto.
- 4.- Añadir las estructuras internas para gestionar los datos. clase auxiliar Matricula que usaremos para crear el vector privado y declaración del Vector y método recargarFilas(). 1 punto.
- 5.- Método SeleccionarFila() e Implementar la diferencia entre el modo de listado normal y listado por DNI. 1 punto.
- 6.- Añadir los métodos para hacer listados (completos y por DNI). 1 punto.
- 7.- Crear la clase que hereda de **EventObject** para que se puedan crear los eventos a lanzar. 1 punto.
- 8.- Añadir el código necesario para generar los eventos cuando se cargan las matrículas. 1 puntos.
- 9.- Generar el ejemplo de prueba que realice un listado completo, otro por DNI y añadir un registro a la tabla. 1 punto.
- 10.- Añadir el código necesario para programar la captura de los eventos que se generan. 1 punto.

Recursos necesarios para realizar la Tarea. No necesitas recursos adicionales para realizar esta tarea. Consejos y recomendaciones.

Repasa el ejemplo que has visto en la teoría, ya que te servirá como base para hacer esta tarea.

La realización de esta práctica puede incluir múltiples variantes, puesto que depende en gran medida de cómo te sea más cómodo programar.

Es aconsejable usar una clase para gestionar las matrículas.

No olvides construir el proyecto e insertar el archivo .jar que se construye en las bibliotecas del proyecto de prueba.

Indicaciones de entrega.

Una vez realizada la tarea tendrás que comprimir los archivos que has generado y subirlos a la plataforma. La estructura de archivos a entregar dentro del archivo comprimido es como sigue:

- 1.- Un directorio llamado leeme donde incluyas un archivo de texto con las observaciones que le quieras hacer al profesor o profesora.
- 2.- Un directorio llamado proyectos en el que incluyas:
  - o Un documento de texto (con extensión .sql) con el código para crear la tabla de alumnos y rellenarla.
  - o El proyecto NetBeans en el que hayas creado el componente.
  - o El proyecto NetBeans de prueba del componente.

El envío se realizará a través de la plataforma de la forma establecida para ello, y el archivo se nombrará siguiendo las siguientes pautas:

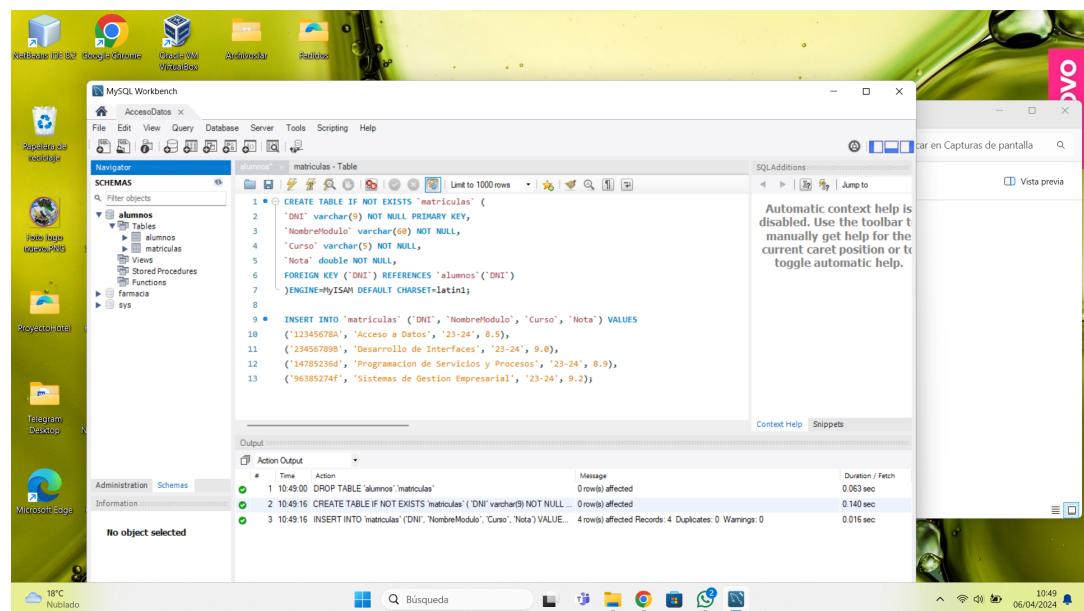
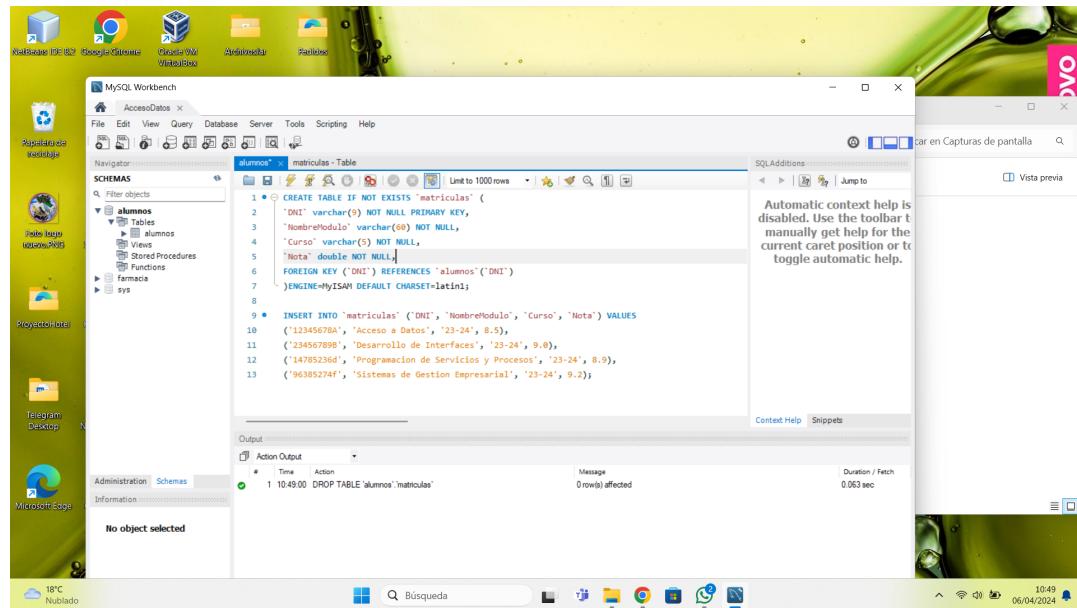
apellido1\_apellido2\_nombre\_ADxx\_TareaXX\_EntregaXX

Asegúrate que el nombre no contenga la letra ñ, tildes ni caracteres especiales extraños. Así por ejemplo la alumna Begoña Sánchez Mañas para la primera unidad del MP de AD, debería nombrar esta tarea como...

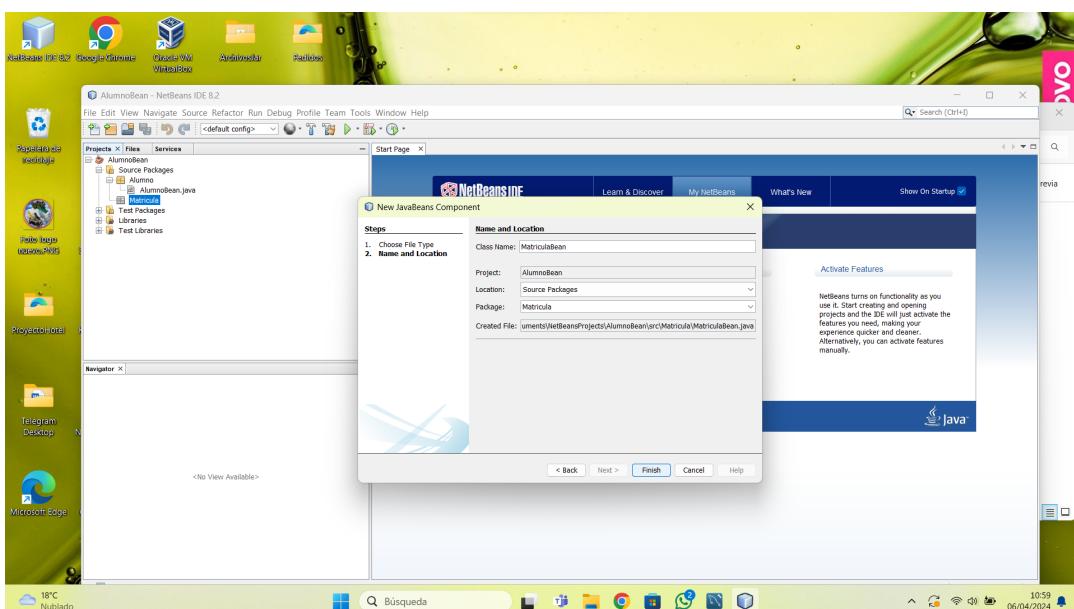
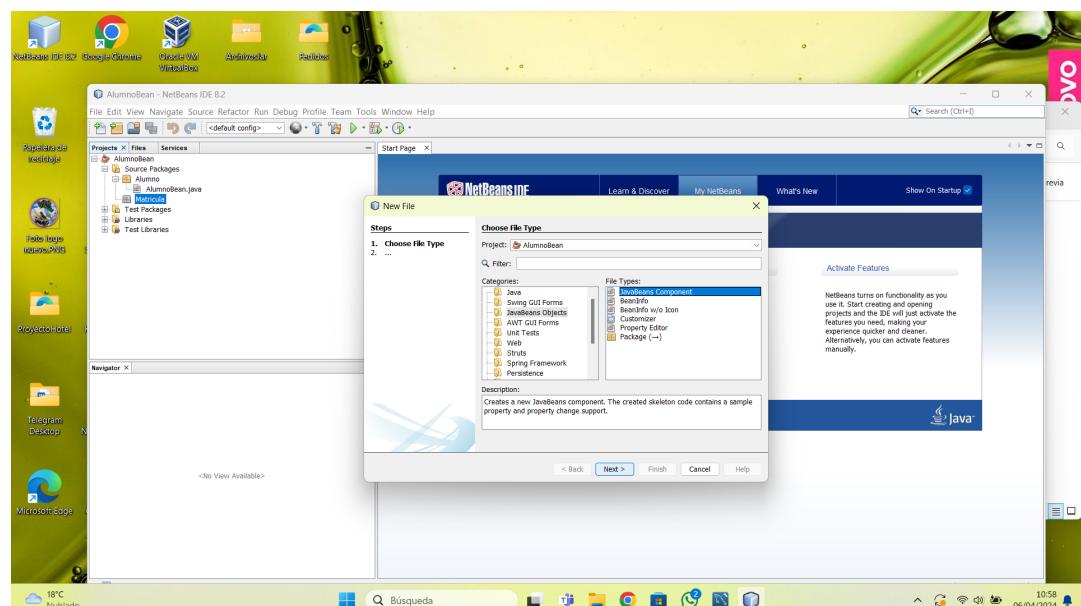
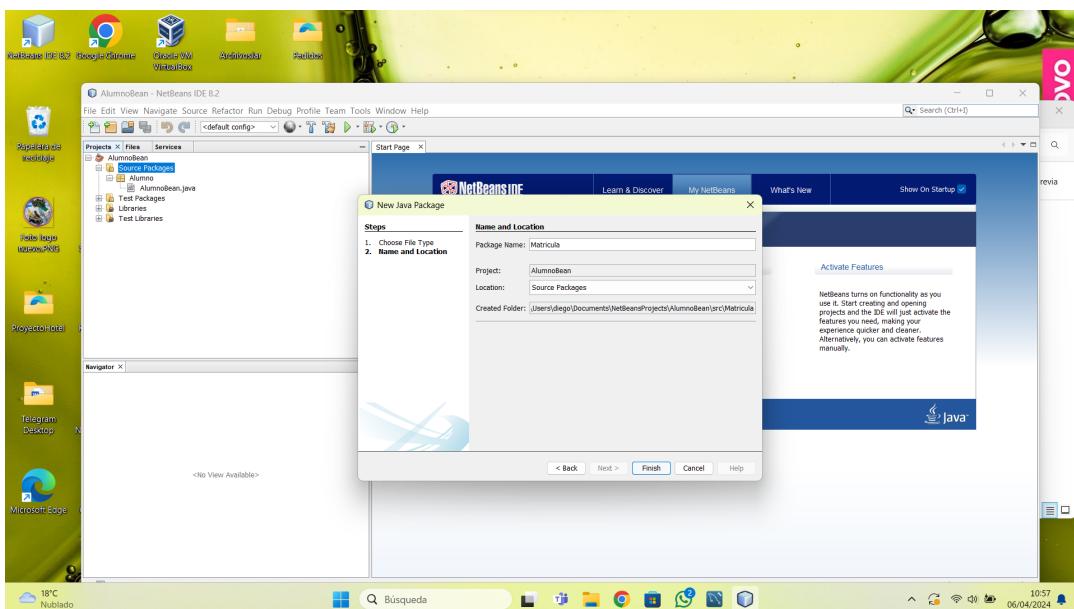
sanchez\_manas\_begona\_AD07\_Tarea07\_Entrega01

Empezaremos creando el script, que compondremos en MySQL Workbench y posteriormente guardaremos.

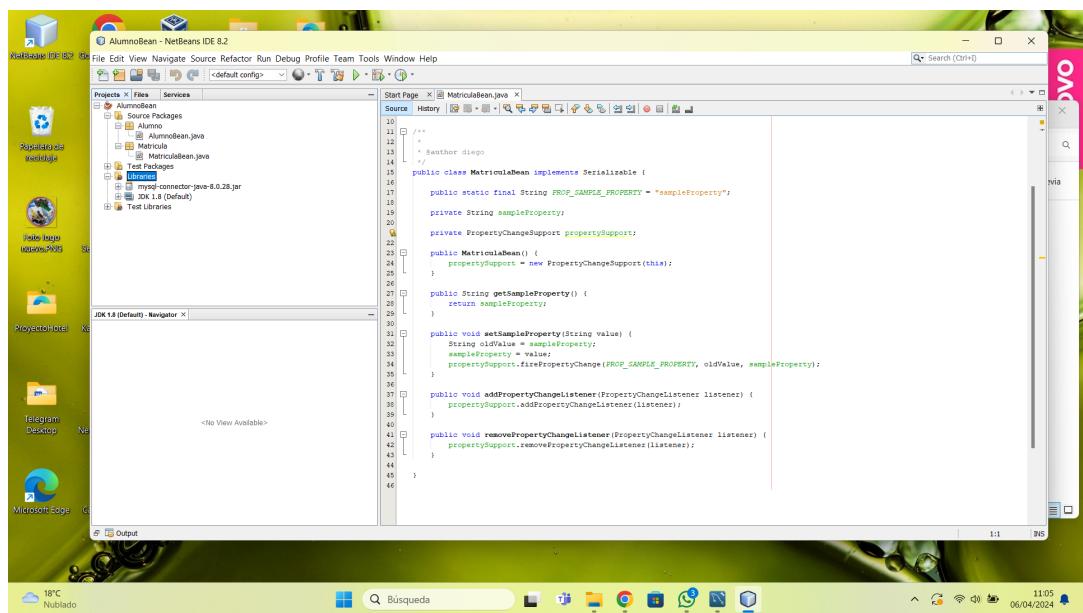
En el mismo script creamos la tabla tal como indica el enunciado de la tarea, asignamos las claves primarias y las foráneas y hacemos los insert into de las diferentes matrículas.



Tras ellos descargaremos el proyecto de ejemplo del apartado 11.8, como indica la tarea, crearemos un nuevo paquete llamado Matricula, y un nuevo JavaBeans Component llamado MatriculaBean.



El proyecto de ejemplo incorpora un conector java antiguo, por lo que lo he reemplazado por uno mas moderno (8.0.28).



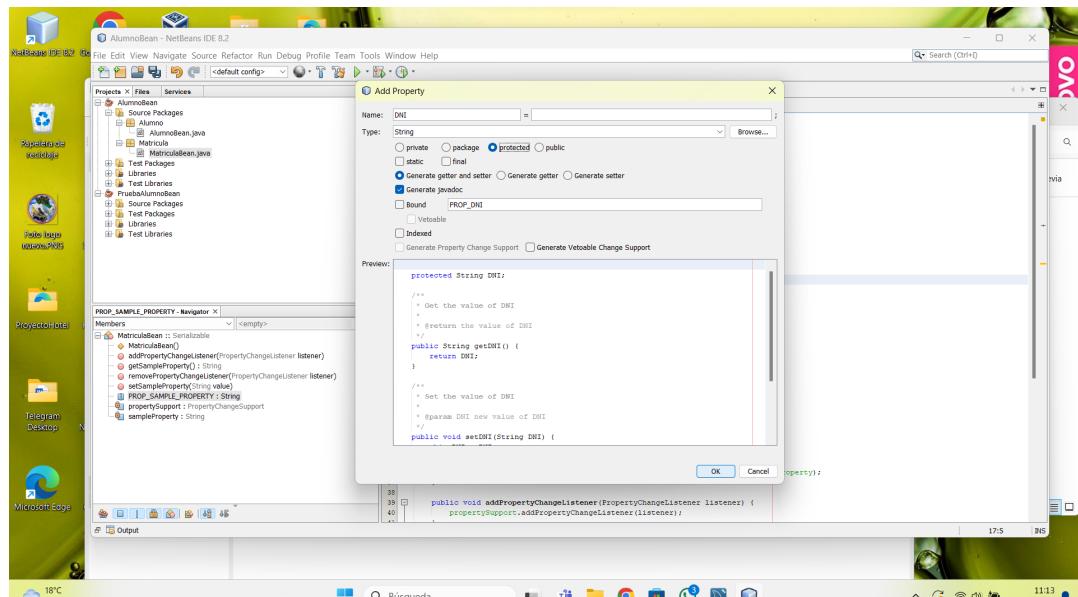
Posteriormente vamos a crear los cuatro atributos con los métodos getter and setter.

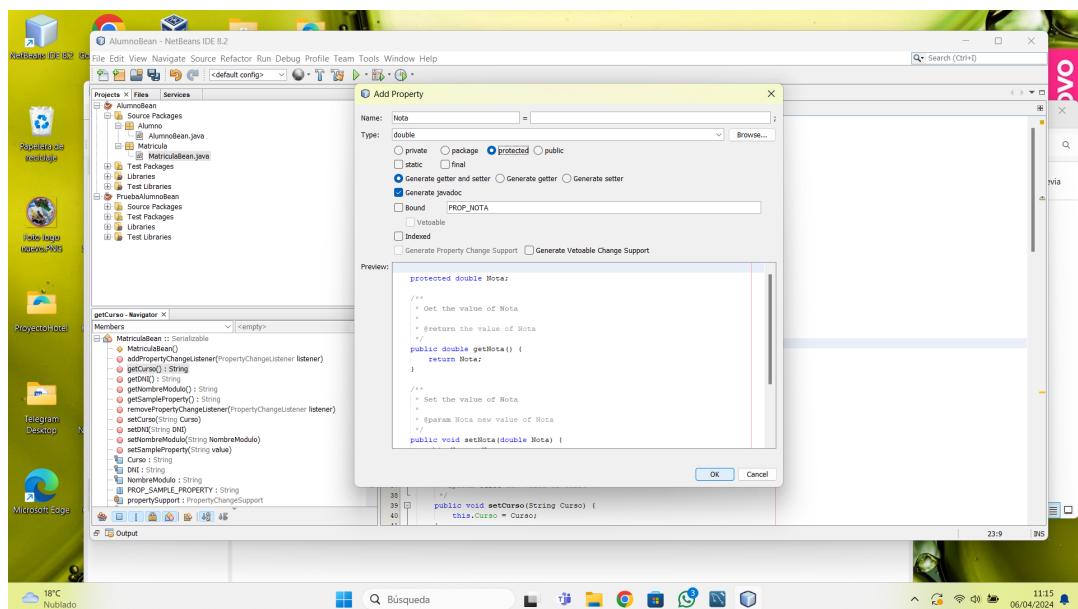
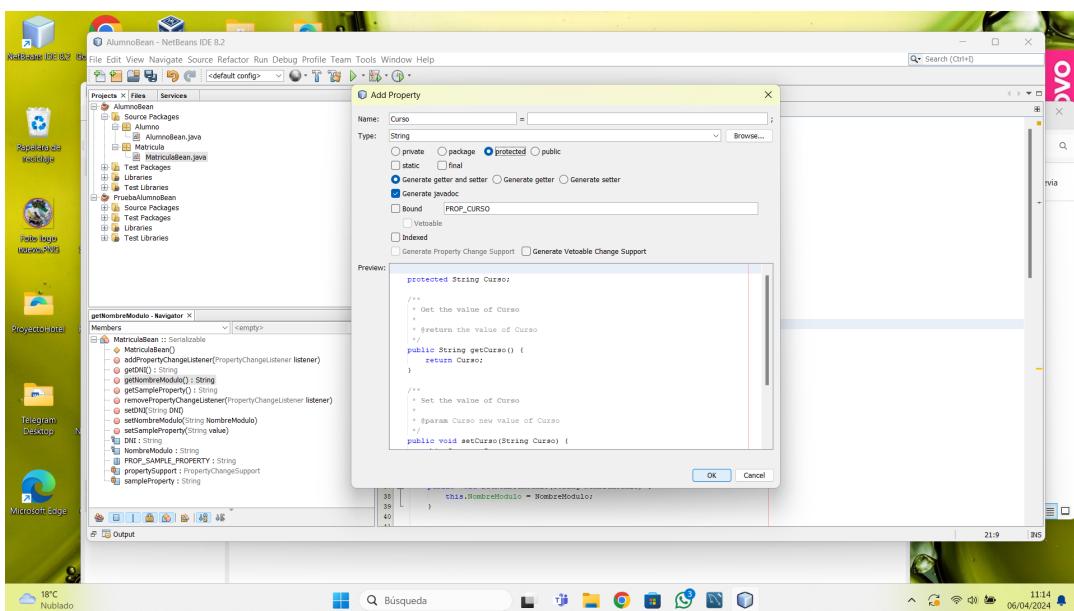
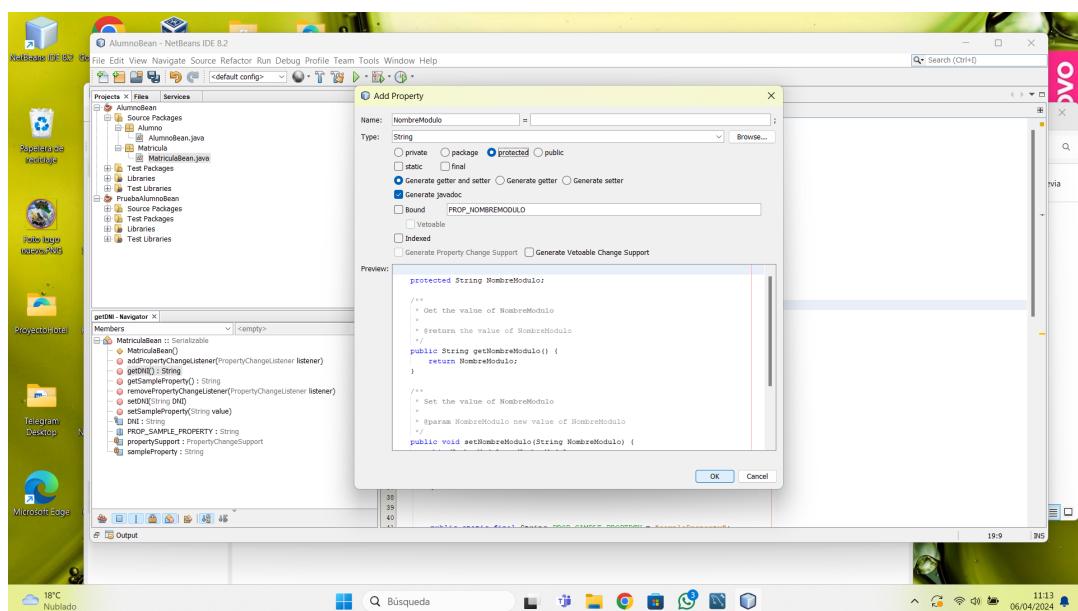
Para ello clicaremos en botón derecho → insert code → add property.

En la ventana que nos sale le daremos un nombre, el tipo (String para DNI, NombreModulo y Curso y double para nota), seleccionaremos protected, generate getter and setter y generate javadoc si no están seleccionados.

Por último le daremos a ok.

Todo ello lo tendremos que repetir con cada atributo que queramos añadir.





```

public class MatriculaBean implements Serializable {
    protected String DNI;
    protected String NombreModulo;
    protected String Curso;
    protected double Nota;

    /**
     * Get the value of Nota
     */
    @return the value of Nota
    public double getNota() {
        return Nota;
    }

    /**
     * Set the value of Nota
     *
     * @param Nota new value of Nota
     */
    public void setNota(double Nota) {
        this.Nota = Nota;
    }

    /**
     * Get the value of Curso
     */
    @return the value of Curso
    public String getCurso() {
        return Curso;
    }

    /**
     * Set the value of Curso
     *
     * @param Curso new value of Curso
     */
    public void setCurso(String Curso) {
        this.Curso = Curso;
    }

    /**
     * Get the value of NombreModulo
     */
    @return the value of NombreModulo
    public String getNombreModulo() {
        return NombreModulo;
    }

    /**
     * Set the value of NombreModulo
     *
     * @param NombreModulo new value of NombreModulo
     */
    public void setNombreModulo(String NombreModulo) {
        this.NombreModulo = NombreModulo;
    }

    /**
     * Get the value of DNI
     */
    @return the value of DNI
    public String getDNI() {
        return DNI;
    }

    /**
     * Set the value of DNI
     *
     * @param DNI new value of DNI
     */
    public void setDNI(String DNI) {
        this.DNI = DNI;
    }
}

```

```

private class Matricula {
    String DNI;
    String NombreModulo;
    String Curso;
    double Nota;

    public Matricula() {
    }

    public Matricula(String nDNI, String sNombreModulo, String sCurso, double dNota) {
        this.DNI = nDNI;
        this.NombreModulo = sNombreModulo;
        this.Curso = sCurso;
        this.Nota = dNota;
    }
}

```

Posteriormente vamos a crear, dentro de esta clase, una clase nueva que será privada y se llamará Matricula.

Tendrá los cuatro atributos mencionados antes (String para DNI, NombreModulo y Curso y double para nota), un constructor vacío y otro por parámetros con todos los atributos, como podemos apreciar en la imagen anterior.

Tras ello crearemos un vector para guardar los datos extraídos de la base de datos.

Posteriormente crearemos un método que nos devolverá el tamaño del vector, para usar posteriormente en el proyecto de prueba.

A continuación vamos a crear el método recargar filas.

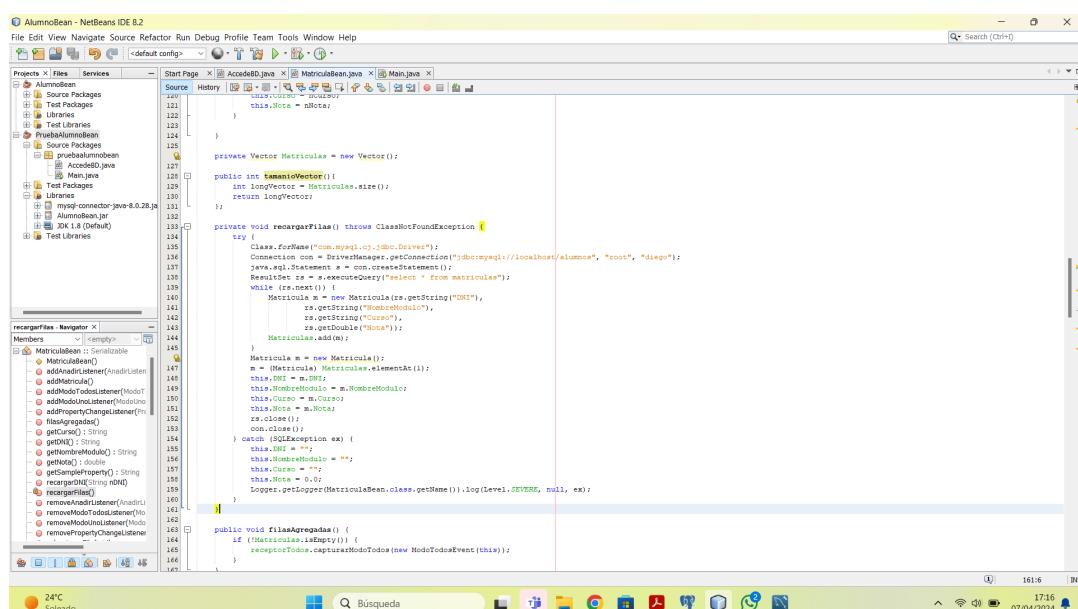
Este método será el encargado de acceder a la base de datos, mediante el conector java, crear la conexión con la dirección de la base de datos, el usuario y la contraseña, y lanzar la consulta (select \* from matriculas) mediante el método executeQuery.

Tras ello crearemos un while, que mientras queden datos por leer del resulset, cree una nueva instancia de la clase matrícula y guarde en ella cada uno de los datos extraídos mediante el método get... (string y double en nuestro caso) y la añada al vector.

Tras el while cerraremos el resulset y la conexión a la base de datos.

Todo ello envuelto en un try - catch.

A continuación crearemos un método que comprobara si el vector no esta vacío, en cuyo caso lanzara el evento para indicar que se han cargado todos los alumnos.



```

AlumnoBean - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
AlumnoBean - AlumnoBean.java - MatriculaBean.java - Main.java - <default config>
Projects Files Services Start Page AlumnoBean.java MatriculaBean.java Main.java
Source History < AlumnoBean.java > < MatriculaBean.java > < Main.java >
121     }
122     }
123     }
124     }
125     }
126     }
127     private Vector<Matricula> matriculas = new Vector();
128     public int tamanoVector() {
129         int longVector = matriculas.size();
130         return longVector;
131     }
132     public void recargarFilas() throws ClassNotFoundException {
133         try {
134             Class.forName("com.mysql.jdbc.Driver");
135             Connection con = DriverManager.getConnection("jdbc:mysql://localhost/alumnos", "root", "diego");
136             java.sql.Statement s = con.createStatement();
137             ResultSet rs = s.executeQuery("Select * from matriculas");
138             while (rs.next()) {
139                 Matricula m = new Matricula(rs.getString("DNI"),
140                     rs.getString("NombreModulo"),
141                     rs.getDouble("Nota"),
142                     rs.getDouble("Curso"));
143                 matriculas.add(m);
144             }
145             Matricula m = new Matricula();
146             m = (Matricula) matriculas.elementAt(1);
147             this.DNI = m.DNI;
148             this.NombreModulo = m.NombreModulo;
149             this.Curso = m.Curso;
150             this.Nota = m.Nota;
151             rs.close();
152             con.close();
153         } catch (SQLException ex) {
154             this.DNI = "";
155             this.NombreModulo = "";
156             this.Curso = "";
157             this.Nota = 0.0;
158             Logger.getLogger(MatriculaBean.class.getName()).log(Level.SEVERE, null, ex);
159         }
160     }
161     public void filtraAgregadas() {
162         if (!matriculas.isEmpty()) {
163             receptorTodos.capturarModoTodos(new ModoTodosEvent(this));
164         }
165     }
166 }

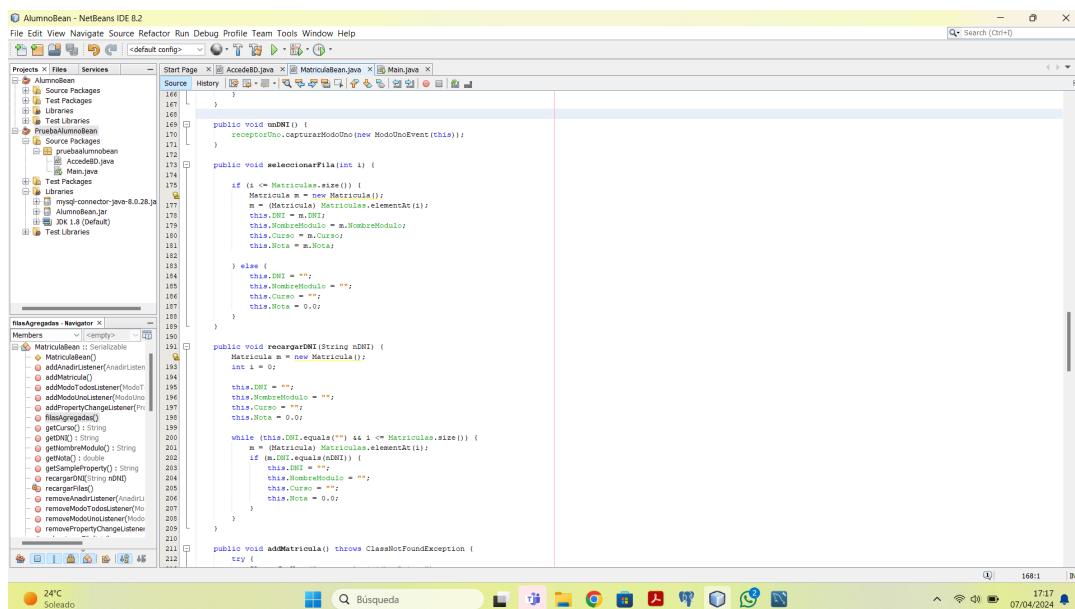
```

Posteriormente crearemos un método que lanzará el evento de cargar solo un alumno.

Tras ello crearemos un método que recorrerá el vector y se encargará de seleccionar las filas del mismo par luego devolver cada uno de los campos leidos.

Seguidamente crearemos un método que nos devolverá los datos del vector que coincidan con el dni que le pasemos.

Para ello creará una instancia de la clase matrícula y tras ello, mediante un while, recorrerá el vector buscando que el dni pasado coincida con el dni encontrado en el vector, en cuyo caso devolverá dicha matrícula.



A continuación crearemos un método para añadir una matrícula a la base de datos.

Para ello conectaremos, a través del conector, con la base de datos pasándole la dirección, el usuario y la contraseña.

Crearemos un preparedStatement con la sentencia (insert into matriculas values ????, donde cada símbolo de interrogación corresponde a uno de los parámetros pasados.

Mediante los métodos set(string y double en nuestro caso) y la posición que ocupan dentro de los parámetros ejecutaremos la consulta mediante el método executeUpdate para posteriormente llamar al método recargarFilas.

Todo ellos estará envuelto en un try - catch.

Ya fuera del try - catch lanzaremos el evento que nos mostrará cuando se añada una nueva matrícula a la base de datos.

Lo siguiente que haremos será crear una clase y su constructor para capturar los eventos necesarios, un interface que gestionara el evento, los método de añadir y eliminar los eventos y la instancia a la clase creada.

Esto lo haremos para cada uno de los eventos, que en mi caso serán tres, uno para capturar cuando se listen todos los clientes, otro para cuando se liste un cliente según el dni pasado y otro para cuando se actualice alguna matricula en la base de datos.

AlumnoBean - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help <default config>

Projects X Files Services Start Page AccedeBD.java MatriculaBean.java Main.java

```
    this.receptorAnadir = receptorAnadir;
}
}

public void removeAnadirListener(AnadirListener receptorAnadir) {
    this.receptorAnadir = null;
}

public static final String PROP_SAMPLE_PROPERTY = "sampleProperty";
private String sampleProperty;
private PropertyChangeSupport propertySupport;

public MatriculaBean() {
    propertySupport = new PropertyChangeSupport(this);
    try {
        recaerarFichero();
    } catch (ClassNotFoundException ex) {
        this.DNI = "";
        this.matriculaDNI = "";
        this.Curso = "";
        this.Nota = 0.0;
    }
}

public String getSampleProperty() {
    return sampleProperty;
}

public void setSampleProperty(String value) {
    String oldValue = sampleProperty;
    sampleProperty = value;
    propertySupport.firePropertyChange(PROP_SAMPLE_PROPERTY, oldValue, sampleProperty);
}

public void addPropertyChangeListener(PropertyChangeListener listener) {
    propertySupport.addPropertyChangeListener(listener);
}

public void removePropertyChangeListener(PropertyChangeListener listener) {
    propertySupport.removePropertyChangeListener(listener);
}
```

ModulosEvent Navigator X

Members <empty>

receptorTodos : ModoTodosListener  
receptorTodosUn : ModoTodosUnListener  
sampleProperty : String  
AnadirEvent :: EventObject  
AnadirEvent : EventObject  
AnadirListener : Listener<EventObject, AnadirEvent>  
Matricula  
Matricula[]  
Matricula : String  
DNI : String  
NombreModule : String  
ModoTodosEvent : EventObject  
ModoTodosEvent : EventObject  
ModoTodosListener : Listener<EventObject, ModuloEvent>  
ModuloEvent : EventObject

24°C Sol y nubes Búsquedas 17:19 07/04/2024

Por último crearemos el constructor la propia clase que llamará al método de recargar filas, como podemos apreciar en la imagen anterior y haremos un clean and build para que nos genere el archivo jar para poder usarlo en otros proyectos.

Una vez llegados a este punto hemos terminado de crear el componente.

Ahora crearemos un proyecto de prueba para probar el componente.

En mi caso, como el ejemplo descargado ya tenía un proyecto para probar el componente lo he usado y modificado para adaptarlo a la tarea.

Lo primero que haremos será importar el archivo jar del proyecto creado anteriormente y, al igual que en el proyecto anterior, cambiar el conector java por uno más moderno.

Tras ello le diremos a la clase AccedeBD que implementa las clases de los tres eventos creados.

Incluirá un instancia a la clase MatriculaBean, el constructor que incluirá una nueva instancia de MatriculasBean y añadirá los tres eventos mediante el método creado a tal fin.

Lo siguiente que haremos será crear el método listado, que será el encargado de listarnos todas las matriculas guardadas en la base de datos.

Para ello recorrerá el vector mediante un for, llamará al método seleccionarFila e imprimirá por pantalla cada una de las matriculas guardadas en el vector.

Ya fuera del for llamará al método que nos lanzará el evento indicando que se han listado todas las matriculas.

El siguiente método será undNI, que se encargará de listarnos un DNI en concreto (el que le pasemos por parámetro).

Para ello, contendrá el mismo código que el método anterior pero solo mostrará el DNI que corresponda al parámetro pasado.

Para ello comprobará mediante un if si el DNI pasado es igual al DNI que ha capturado de la base de datos.

Posteriormente llamará al método que lanzara el evento cuando se muestra un solo DNI.

```

public class AccedeBD implements ModoTodosListener, ModoUnoListener, AnadirListener {
    private Matriculas matriculas;
    ...
}

AccedeBD() {
    matriculas = new MatriculaBean();
    matriculas.addModoTodosListener(this);
    matriculas.addModoUnoListener(this);
    matriculas.addAnadirListener(this);
}

public void listaMatricula() {
    for (int i = 0; i < matriculas.tamanoVector(); i++) {
        matriculas.seleccionarFila(i);
        System.out.println("[" + i + "] " + matriculas.getDNI());
        System.out.println("Nombre del modulo: " + matriculas.getNombreModulo());
        System.out.println("Cursos: " + matriculas.getCurso());
        System.out.println("Nota: " + matriculas.getNota());
    }
    matriculas.filasAgregadas();
}

public void unMatricula(String nDNI) {
    for (int i = 0; i < matriculas.tamanoVector(); i++) {
        matriculas.seleccionarFila(i);
        String DNI = matriculas.getDNI().toString();
        if (DNI.equals(nDNI)) {
            System.out.println("Alumno con DNI: " + matriculas.getDNI());
            System.out.println("Nombre del modulo: " + matriculas.getNombreModulo());
            System.out.println("Cursos: " + matriculas.getCurso());
            System.out.println("Nota: " + matriculas.getNota());
        }
    }
    matriculas.unMatricula();
}

public void adddeMatricula() {
    matriculas.setDNI("76764321X");
    matriculas.setNombreModulo("Programacion Multimedia y Dispositivos Moviles");
    matriculas.setCurso("1º");
    matriculas.setNota(9.0);
    try {
        matriculas.sacarMatricula();
    } catch (Exception ex) {
        Logger.getLogger(AccedeBD.class.getName()).log(Level.SEVERE, null, ex);
    }
}

```

El siguiente método que crearemos será el encargado de añadir una matrícula a la base de datos.

Para ello, mediante los métodos set instanciará los atributos para posteriormente llamar al método addMatricula y al método que lanza el evento en caso de añadir una matrícula como puede verse en la imagen anterior.

Lo último que haremos será crear los tres métodos de los eventos para que muestren los mensajes pertinentes en cada caso.

```

public void listaMatricula() {
    for (int i = 0; i < matriculas.tamanoVector(); i++) {
        matriculas.seleccionarFila(i);
        System.out.println("[" + i + "] " + matriculas.getDNI());
        System.out.println("Nombre del modulo: " + matriculas.getNombreModulo());
        System.out.println("Cursos: " + matriculas.getCurso());
        System.out.println("Nota: " + matriculas.getNota());
    }
    matriculas.filasAgregadas();
}

public void unMatricula(String nDNI) {
    for (int i = 0; i < matriculas.tamanoVector(); i++) {
        matriculas.seleccionarFila(i);
        String DNI = matriculas.getDNI().toString();
        if (DNI.equals(nDNI)) {
            System.out.println("Alumno con DNI: " + matriculas.getDNI());
            System.out.println("Nombre del modulo: " + matriculas.getNombreModulo());
            System.out.println("Cursos: " + matriculas.getCurso());
            System.out.println("Nota: " + matriculas.getNota());
        }
    }
    matriculas.unMatricula();
}

public void adddeMatricula() {
    matriculas.setDNI("76764321X");
    matriculas.setNombreModulo("Programacion Multimedia y Dispositivos Moviles");
    matriculas.setCurso("1º");
    matriculas.setNota(9.0);
    try {
        matriculas.sacarMatricula();
    } catch (ClassNotFoundException ex) {
        Logger.getLogger(AccedeBD.class.getName()).log(Level.SEVERE, null, ex);
    }
}

public void capturarModoTodos(ModoTodosEvent mue) {
    System.out.println("Se han cargado todas las matriculas");
}

public void capturarModoUno(ModoUnoEvent mue) {
    System.out.println("Se ha cargado la matricula del alumno con DNI 12345678A");
}

public void capturarAnadir(AnadirEvent avi) {
    System.out.println("Se ha añadido una nueva matricula a la base de datos");
}

```

En las siguientes imágenes podemos ver la salida por consola del proyecto.

```

PruebaAlumnoBean - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects Services Start Page AccesoBD.java MatriculaBean.java Main.java
Source History <default config> Run Stop Build Clean Reload All
Output PruebaAlumnoBean (run)
Todas las matrículas
Alumno 1
DNI:12345678A
Nombre del modulo: Acceso a Datos
Cursos: 23-24
Nota: 8.5
Alumno 2
DNI:12345678B
Nombre del modulo: Desarrollo de Interfaces
Cursos: 23-24
Nota: 9.0
Alumno 3
DNI:12345678C
Nombre del modulo: Programación de Servicios y Procesos
Cursos: 23-24
Nota: 8.9
Alumno 4
DNI:12345678D
Nombre del modulo: Sistemas de Gestión Empresarial
Cursos: 23-24
Nota: 9.2
Se han cargado todas las matrículas
Solo la matrícula del DNI 12345678A
Alumno con DNI:12345678A
Nombre del modulo: Acceso a Datos
Cursos: 23-24
Nota: 8.5
Nota: 8.5
Se ha cargado la matrícula del alumno con DNI 12345678A
Además matrícula a la base de datos
Se ha añadido una nueva matrícula a la base de datos
BUILD SUCCESSFUL (total time: 2 seconds)

```

24°C Soleado      Búsqueda      17:45 07/04/2024

```

PruebaAlumnoBean - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects Services Start Page AccesoBD.java MatriculaBean.java Main.java
Source History <default config> Run Stop Build Clean Reload All
Output PruebaAlumnoBean (run)
Todas las matrículas
Alumno 1
DNI:12345678A
Nombre del modulo: Acceso a Datos
Cursos: 23-24
Nota: 8.5
Alumno 2
DNI:12345678B
Nombre del modulo: Desarrollo de Interfaces
Cursos: 23-24
Nota: 9.0
Alumno 3
DNI:12345678C
Nombre del modulo: Programación de Servicios y Procesos
Cursos: 23-24
Nota: 8.9
Alumno 4
DNI:12345678D
Nombre del modulo: Sistemas de Gestión Empresarial
Cursos: 23-24
Nota: 9.2
Alumno 5
DNI:12345678E
Nombre del modulo: Programación Multimedia y Dispositivos Móviles
Cursos: 23-24
Nota: 8.8
Alumno 6
DNI:12345678F
Nombre del modulo: Acceso a Datos
Cursos: 23-24
Nota: 8.7
Alumno 7
DNI:12345678G
Nombre del modulo: Desarrollo de Interfaces
Cursos: 23-24
Nota: 9.1
Alumno 8
DNI:12345678H
Nombre del modulo: Programación de Servicios y Procesos
Cursos: 23-24
Nota: 8.6
Se han cargado todas las matrículas
Solo la matrícula del DNI 12345678A
Alumno con DNI:12345678A
Nombre del modulo: Acceso a Datos
Cursos: 23-24
Nota: 8.5
Nota: 8.5
Se ha cargado la matrícula del alumno con DNI 12345678A
BUILD SUCCESSFUL (total time: 1 second)

```

24°C Soleado      Búsqueda      17:23 07/04/2024

Y con esto damos por finalizada la tarea.