

## Tarea para AD07.

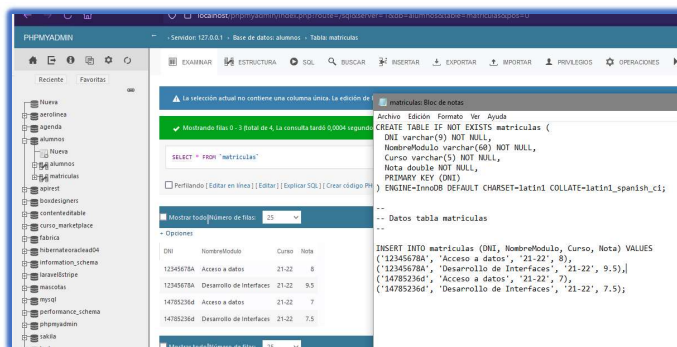
### Enunciado.

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:

- ✓ Debes añadir una tabla a la base de datos alumnos que represente las matrículas de los alumnos. 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 guion, por ejemplo 11-12.
  - **Nota:** double.

Recuerda rellenar la tabla con algunos datos para que puedas hacer pruebas.

Este primer apartado nos pide crear una nueva tabla para la base de datos **alumnos** con la que he estado haciendo las pruebas en el temario. Uso por tanto el mismo motor y tipo de caracteres que con la tabla **alumnos** y para los datos cogeré los dos primeros **dni** de esa primera tabla (por ello, no lo usaré finalmente como **primary key**), y las dos asignaturas que estoy cursando este año. Copio el resto de consulta en la ventana SQL de **PhpMyAdmin** según está y la ejecuto.

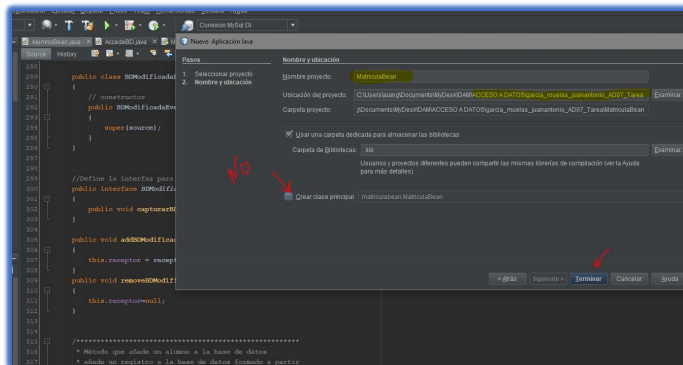


Detalle creación tabla matriculas

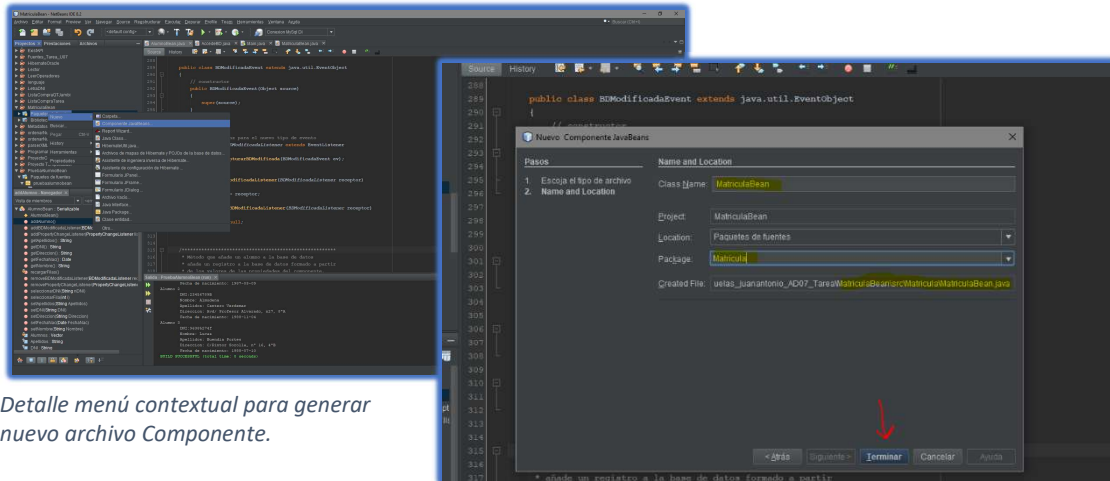
Con esto, ya podemos seguir con el siguiente punto.

- ✓ Crea un componente nuevo en el proyecto Alumno que para gestionar toda esta información. Además del código necesario para gestionar las propiedades del componente y mantener la información de la base de datos en un vector interno, es preciso que incluyas los siguientes métodos:
  - **seleccionarFila(i):** recupera en las propiedades del componente el registro número **i** del vector.
  - **RecargarDNI():** recarga la estructura interna del componente con las matrículas de un DNI en particular.
  - **AddMatricula():** añade un registro nuevo a la base de datos con la información almacenada en las propiedades del componente.
  - 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.

Para el segundo apartado, creo un nuevo proyecto **Java Application**, llamado **MatriculaBean** (a imagen del ejemplo trabajado en el tema), sin clase principal y dentro un nuevo archivo de tipo **Componente JavaBeans** con el mismo nombre, y dentro de una carpeta que llamaré **Matricula**.



Creación de proyecto.



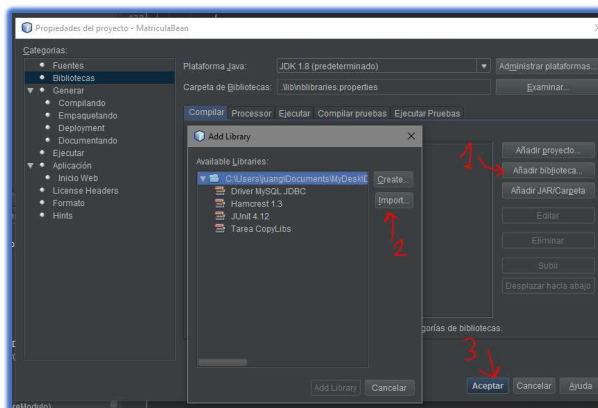
Detalle menú contextual para generar nuevo archivo Componente.

Detalle creación archivo Componente JavaBeans.

Nos genera de forma automática un código standard, del que podemos borrar buena parte, como los ejemplos.

Con la clase creada, tomaré como base el anterior componente para trabajar sobre él, modificando lo necesario para cumplir con lo solicitado para este ejercicio.

Antes de dar más pasos, añadimos las librerías que ya utilizaba el ejemplo del temario, para conectarnos con nuestra base de datos.



Añadiendo librerías para nuestro proyecto.

Como en tareas anteriores, intentaré dejar lo mejor documentado el código, y resaltaré aspectos puntales de la clase creada.

### MatriculaBean.java

```
/*
 * TAREA AD07.
 * Crea un componente nuevo en el proyecto Alumno que para gestionar
 * toda esta información.
 * Además del código necesario para gestionar las propiedades del
 * componente y mantener la información de la base de datos en un
 * vector interno, es preciso que incluyas los siguientes métodos:
 * seleccionarFila(i): recupera en las propiedades del componente
 * el registro número i del vector.
 * RecargarDNI(): recarga la estructura interna del componente con
 * las matrículas de un DNI en particular.
 * AddMatricula(): añade un registro nuevo a la base de datos con
 * la información almacenada en las propiedades del componente.
 * 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.
 *
 * Esta clase contiene toda la configuración relativa a la
 * creación del componente.
 */

package Matricula;

import java.beans.*;
import java.io.Serializable;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.EventListener;
import java.util.Vector;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author juang <juangmuelas@gmail.com>
 * @since 14/04/2022
 * @version 1
 */
public class MatriculaBean implements Serializable {

    /**
     * *****
     * Propiedades comunes de conexión para los métodos.
     * Evitaremos así posibles errores, tanto en las
     * llamadas a la BBDD como ante posibles cambios.
     * Si tu usuario o password son diferentes, recuerda modificarlos
     * y volver a compilar.
     */

    static final String url = "jdbc:mysql://localhost:3306/alumnos";
    static final String user="root";
    static final String pass="";
```

```

/*****
 * Propiedades del Bean.
 * Crearemos una propiedad por cada campo de la tabla de
 * la base de datos del siguiente modo:
 *
 * DNI: String
 * NombreModulo: String
 * Curso: String
 * Nota: double
 */

protected String DNI;
/**
 * 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;
}

protected String NombreModulo;

/**
 * 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;
}

protected String Curso;

/**
 * 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;
}

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;
}


/**
 * Nuestro componente ya nos ha creado este acceso a la
 * clase PropertyChangeSupport que manejará posibles
 * cambios en nuestras propiedades.
 *
 */
private PropertyChangeSupport propertySupport;

/**
 * Constuctor que manteniendo la escucha de eventos,
 * llama al método recargar filas.
 *
 */
public MatriculaBean() {
    propertySupport = new PropertyChangeSupport(this);

    try {
        recargarFilas();
    } catch (ClassNotFoundException ex) {
        this.DNI = "";
        this.NombreModulo = "";
        this.Curso = "";
        this.Nota = 0.0;

        Logger.getLogger(MatriculaBean.class.getName()).log(Level.SEVERE,
            null, ex);
    }
}
} // fin constructor

```

```

/*****
 * Tras varios errores en la salida del ejemplo,
 * decido generar una nueva propiedad con la que poder
 * manejar el tamaño de nuestro array de datos
 * cuando se maneje desde otro proyecto.
 */
protected int size;

/**
 * Get the value of size
 * @return the value of size
 */
public int getSize() {
    return size;
}

/**
 * Set the value of size
 * @param size new value of size
 */
public void setSize(int size) {
    this.size = size;
}

/**
 * @param modoDni boolean discrimina según petición
 */
boolean modoDni;

/*****
 * Clase auxiliar que usaremos para crear un vector privado
 * de alumnos.
 */
private class Matricula{
    /**
     * Recogemos las propiedades para nuestra clase auxiliar.
     *
     * @param DNI String
     * @param NombreModulo String
     * @param Curso String
     * @param Nota double
     */
    String DNI;
    String NombreModulo;
    String Curso;
    double Nota;

    //constructor vacio
    public Matricula() {
    }

    //Constructor con argumentos
    public Matricula(String DNI, String NombreModulo, String
        Curso, double Nota) {
        this.DNI = DNI;
        this.NombreModulo = NombreModulo;
        this.Curso = Curso;
        this.Nota = Nota;
    }
}

} //fin clase Matricula

```

```

/*****
 * Usaremos un vector auxiliar para cargar la información
 * de la tabla de forma que tengamos acceso a los datos
 * sin necesidad de estar conectados constantemente
 */
private Vector Matriculas = new Vector();

```

Hasta este punto cumple casi de forma idéntica con lo desarrollado en el ejemplo del temario, salvo por la propiedad `size` que nos ayudará en el futuro a controlar la salida de nuestras peticiones y `modoDni` para discriminar como disparar el evento según pidamos por `Dni` o por `matrícula`.

Prosigo con los métodos requeridos, que tras evaluar como funcionaban en el ejemplo del tema, he decidido añadirle siempre al comienzo, un control sobre el `vector` de recogida de datos, empezando siempre por un reseteo del mismo.

```

/*****
 * Actualiza el contenido de la tabla en el vector de matriculas
 * Las propiedades contienen el valor del primer elemento de la
 * tabla.
 * @throws java.lang.ClassNotFoundException
 */
public void recargarFilas() throws ClassNotFoundException {
    /**
     * Aseguro la estricta recogida de datos en el vector.
     */
    if(!Matriculas.isEmpty())
    {
        Matriculas.removeAllElements();
    }
    try {
        Class.forName("com.mysql.jdbc.Driver");
        Connection con =
            DriverManager.getConnection(url,user,pass);
        java.sql.Statement s = con.createStatement();
        ResultSet rs = s.executeQuery("select * from matriculas");
        while (rs.next())
        {
            Matricula a = new Matricula(rs.getString("DNI"),
                                         rs.getString("NombreModulo"),
                                         rs.getString("Curso"),
                                         rs.getDouble("Nota"));

            Matriculas.add(a);
        }
        Matricula mat = new Matricula();
        mat = (Matricula) Matriculas.elementAt(1);
        this.DNI = mat.DNI;
        this.NombreModulo = mat.NombreModulo;
        this.Curso = mat.Curso;
        this.Nota = mat.Nota;

        /**
         * Tras recoger los datos recibidos recojo el
         * tamaño final y cierro conexiones.
         */

        size = Matriculas.size();
        rs.close();
    }
}

```

```

        con.close();
    } catch (SQLException ex) {
        this.DNI = "";
        this.NombreModulo = "";
        this.Curso = "";
        this.Nota = 0;
        Logger.getLogger(MatriculaBean.class.getName()).log(Level.SEVERE,
            null, ex);
    }
} //fin recargarFilas()

/*****
 *
 * @param i numero de la fila a cargar en las
 * propiedades del componente.
 *
 */
public void seleccionarFila(int i)
{
    if(i < Matriculas.size())
    {
        Matricula mat = new Matricula();
        mat = (Matricula) Matriculas.elementAt(i);
        this.DNI = mat.DNI;
        this.NombreModulo = mat.NombreModulo;
        this.Curso = mat.Curso;
        this.Nota = mat.Nota;
    } else {
        this.DNI = "";
        this.NombreModulo = "";
        this.Curso = "";
        this.Nota = 0.0;
    }
} //Fin seleccionarFila(i)

/*****
 *
 * @param DNI DNI A buscar, se carga en las propiedades del
 * componente
 *
 */
public void recargarDNI(String DNI) throws ClassNotFoundException
{
    /**
     * Aseguro la estricta recogida de datos en el vector.
     */
    if(!Matriculas.isEmpty())
    {
        Matriculas.removeAllElements();
    }
    try {
        Class.forName("com.mysql.jdbc.Driver");
        Connection con =
            DriverManager.getConnection(url, user, pass);
        PreparedStatement s = con.prepareStatement(
            "select * from matriculas where DNI = ?");
        s.setString(1, DNI);
        ResultSet rs = s.executeQuery();
        while (rs.next())
        {
            Matricula a = new Matricula(rs.getString("DNI"),
                rs.getString("NombreModulo"),

```



```

        rs.getString("Curso"),
        rs.getDouble("Nota"));

        Matriculas.add(a);
    }
    Matricula mat = new Matricula();
    mat = (Matricula) Matriculas.elementAt(1);
    this.DNI = mat.DNI;
    this.NombreModulo = mat.NombreModulo;
    this.Curso = mat.Curso;
    this.Nota = mat.Nota;
    /**
     * Dando un valor de true al booleano, permitimos
     * un mensaje concreto al lanzar el evento.
     */
    modoDni = true;
    receptor.capturarBDModificada(
        new BDModificadaEvent(this, modoDni));

    /**
     * Tras recoger los datos recibidos recojo el
     * tamaño final y cierro conexiones.
     */
    size = Matriculas.size();
    rs.close();
    con.close();
} catch (SQLException ex) {
    this.DNI = "";
    this.NombreModulo = "";
    this.Curso = "";
    this.Nota = 0;

    Logger.getLogger(MatriculaBean.class.getName()).log(Level.SEVERE,
        null, ex);
}
} //fin recargarDNI()

```

La parte que maneja los cambios (inserciones) en la tabla `matriculas`, he creído que podía mantener los nombres de los métodos que controlan las escuchas y respuestas a eventos del ejemplo, por ser cambios en la propia BD. Sólo ampliamos el constructor de la clase que hereda de `EventObject`, para poder elegir el modo.

```

/*****
 * Código para añadir una nueva matrícula la base de datos.
 * cada vez que se modifica el estado de la BD se genera un evento
 * para que se recargue el componente.
 */

private BDModificadaListener receptor;

public class BDModificadaEvent extends java.util.EventObject
{
    // constructor
    public boolean modoDni;
    public BDModificadaEvent(Object source, boolean modoDNI)
    {
        super(source);
        modoDni = modoDNI;
    }
}

```

```

//Define la interfaz para el nuevo tipo de evento
public interface BDModificadaListener extends EventListener
{
    public void capturarBDModificada(BDModificadaEvent ev);
}

public void addBDModificadaListener(BDModificadaListener receptor)
{
    this.receptor = receptor;
}
public void removeBDModificadaListener(BDModificadaListener
receptor)
{
    this.receptor=null;
}

/*****
 * Método que añade una matrícula a la base de datos
 * añade un registro a la base de datos formado a partir
 * de los valores de las propiedades del componente.
 *
 * Se presupone que se han usado los métodos set para configurar
 * adecuadamente las propiedades con los datos del nuevo registro.
 */
public void addMatricula() throws ClassNotFoundException
{
    try {
        Class.forName("com.mysql.jdbc.Driver");

        Connection con =
        DriverManager.getConnection(url, user, pass);
        PreparedStatement s = con.prepareStatement(
            "insert into matriculas values (?, ?, ?, ?)");

        s.setString(1, DNI);
        s.setString(2, NombreModulo);
        s.setString(3, Curso);
        s.setDouble(4, Nota);

        s.executeUpdate ();
        recargarFilas ();

        /**
         * Dando un valor de false al booleano, permitimos
         * un mensaje concreto al lanzar el evento.
         */
        modoDni = false;
        receptor.capturarBDModificada(
            new BDModificadaEvent(this, modoDni));

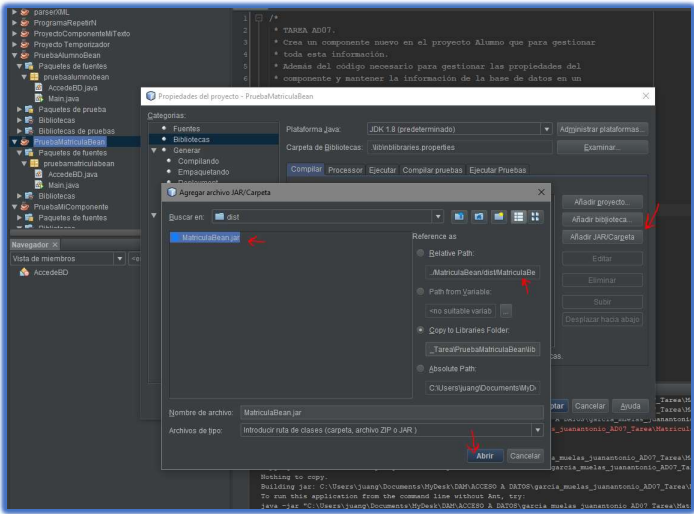
        receptor.capturarBDModificada(
            new BDModificadaEvent(this));
    }
    catch(SQLException ex)
    {
        Logger.getLogger(MatriculaBean.class.getName()).log(Level.SEVERE,
        null, ex);
    }
}
} // fin addMatricula()

```

Con el código desarrollado, sólo nos queda limpiar y construir el proyecto, para así obtener el `.jar` con el que poder probarlo, por ejemplo, en nuestro proyecto de prueba del próximo punto.

- ✓ 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.

Tras crear un nuevo proyecto, que siguiendo el ejemplo mantendrá dos clases (`main` y `accedeBD`), le añadimos el componente y la librería con el conector como en la creación del componente.



*Añadiendo archivo .jar con nuestro componente.*

La clase `AccedeBD` se encargará de la gestión de los eventos, por medio de varios métodos creados para recoger los datos según las directrices dadas en la tarea y que tienen como base, el ejercicio del tema. Podremos aquí ver la función concreta de las variables `size` y `modoDni`, que permiten ajustar mejor las salidas a nuestros propósitos:

## AccedeBD.java

```
/*
 * TAREA AD07.
 * Desde esta clase, generamos los métodos que trabajarán con
 * nuestro componente.
 */
```

```

package pruebamatriculabean;

import Matricula.MatriculaBean;
import Matricula.MatriculaBean.BDModificadaEvent;
import Matricula.MatriculaBean.BDModificadaListener;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author juang <juangmuelas@gmail.com>
 * @since 14/04/2022
 * @version 1
 */
public class AccedeBD implements BDModificadaListener{
    MatriculaBean matriculas;

    /**
     * Constructor indicando que vamos a estar
     * escuchando si hay un evento.
     */
    AccedeBD(){
        matriculas = new MatriculaBean();
        matriculas.addBDModificadaListener(
            (BDModificadaListener)this );
    }

    /**
     * Método para listar todas las matrículas de nuestra BD.
     * Recorremos mediante un for que hemos ajustado al tamaño
     * de nuestro array.
     */
    public void listado()
    {
        System.out.println("\033[36m***** LISTADO GENERAL DE
MATRÍCULAS *****");
        for(int i=0; i< matriculas.getSize(); i++)
        {
            matriculas.seleccionarFila(i);
            System.out.println("Alumno con DNI:" + matriculas.getDNI()
+ " matriculado en:");
            System.out.println("\tNombre Modulo: "
+ matriculas.getNombreModulo());
            System.out.println("\tCurso: " + matriculas.getCurso());
            System.out.println("\tNota: " + matriculas.getNota());
        }
    }

    /**
     * Método para listar por un dni que le indicaremos desde
     * la llamada en el main.
     * Recorremos mediante un for que hemos ajustado al tamaño
     * de nuestro array.
     * @param DNI
     * @throws ClassNotFoundException
     */
    public void listadoDNI(String DNI) throws ClassNotFoundException
    {
        matriculas.recargarDNI(DNI);
        System.out.println(
            "\033[36m***** LISTADO DE MATRÍCULAS PARA EL DNI "

```

```

        + matriculas.getDNI()+" *****");
    for(int i= 0; i < matriculas.getSize(); i++){
        matriculas.seleccionarFila(i);
        System.out.println("\tNombre Modulo: "
            + matriculas.getNombreModulo());
        System.out.println("\tCurso: " + matriculas.getCurso());
        System.out.println("\tNota: " + matriculas.getNota());
    }
}

```

- ✓ Cuando cargues la matrícula del usuario concreto deberás capturar el evento generado al cambiar de modo.
- ✓ Añade el código necesario para añadir una matrícula nueva a la base de datos.

```

/**
 * Método para añadir una nueva matrícula.
 */
void anade ()
{
    matriculas.setDNI("98765432B");
    matriculas.setNombreModulo("Programación");
    matriculas.setCurso("21-22");
    matriculas.setNota(7.5);

    try {
        matriculas.addMatricula();
    } catch (ClassNotFoundException ex) {

        Logger.getLogger(AccedeBD.class.getName()).log(Level.SEVERE,
            null, ex);
    }
}

/**
 * Método para sobrecribir y lanzar un mensaje por consola al
 * producirse el evento.
 * @param ev
 */

public void capturarBDModificada(BDModificadaEvent ev)
{
    if(ev.mododni)
        System.out.println("\033[33mSe ha pedido el listado con
            DNI a la base de datos");
    else
        System.out.println("\033[33mSe ha añadido una nueva
            matrícula a la base de datos");
}

}
}

```

Por último, sólo nos queda la clase `main`, con la que observar el funcionamiento por la consola.

#### Main.java

```

/*
 * TAREA AD07.
 * Desde esta clase, accedemos a los métodos de consulta a BBDD
 * creados en la clase AccedeBD, para mostrar resultados.
 */

```

```

package pruebamatriculabean;

import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author juang <juangmuelas@gmail.com>
 * @since 14/04/2022
 * @version 1
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // Creamos un objeto de AccedeBD
        AccedeBD gestion = new AccedeBD();

        //llamamos al método listado para mostrar resultado por consola
        gestion.listado();

        /**
         * llamamos al método listadoDNI para mostrar resultado
         * por consola.
         * Lo envolvemos en un try-catch que recoge la excepción
         * común en este tipo de llamada.
         */
        try {
            gestion.listadoDNI("12345678A");
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE,
null, ex);
        }

        /**
         * llamamos al método anade para mostrar resultado
         * por consola.
         * Lo dejo comentado para evitar grabados posteriores con
         * los mismos datos.
         */
        // gestion.anade();
    } //fin método main()
} //Fin clase Main

```

Si ejecutamos, dejando comentados los dos últimos métodos, nos muestra el listado completo:

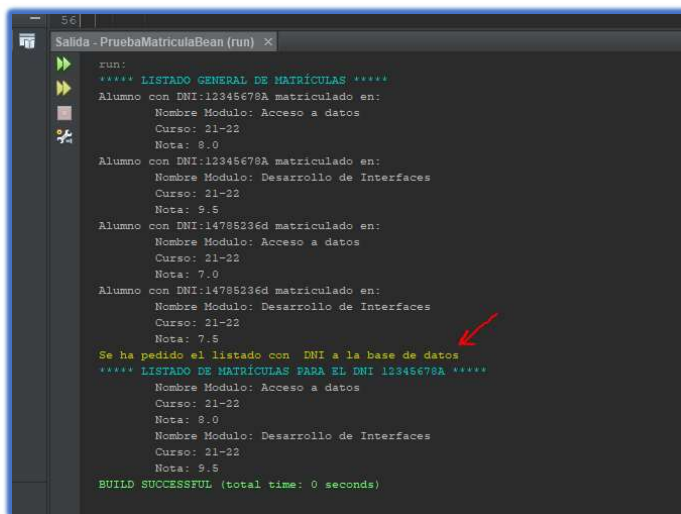
```

run:
***** LISTADO GENERAL DE MATRÍCULAS *****
Alumno con DNI:12345678A matriculado en:
Nombre Modulo: Acceso a datos
Curso: 21-22
Nota: 8.0
Alumno con DNI:12345678A matriculado en:
Nombre Modulo: Desarrollo de Interfaces
Curso: 21-22
Nota: 9.5
Alumno con DNI:14785236d matriculado en:
Nombre Modulo: Acceso a datos
Curso: 21-22
Nota: 7.0
Alumno con DNI:14785236d matriculado en:
Nombre Modulo: Desarrollo de Interfaces
Curso: 21-22
Nota: 7.5
BUILD SUCCESSFUL (total time: 0 seconds)

```

*Detalle listado.*

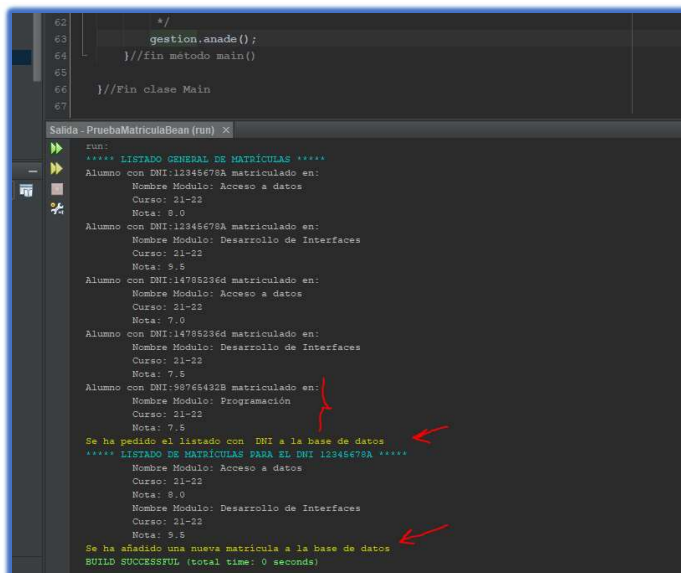
Si añadimos el listado por dni, nos añadirá además la respuesta con el evento programado.



```
run:
**** LISTADO GENERAL DE MATRÍCULAS ****
Alumno con DNI:12345678A matriculado en:
Nombre Modulo: Acceso a datos
Curso: 21-22
Nota: 8.0
Alumno con DNI:12345678A matriculado en:
Nombre Modulo: Desarrollo de Interfaces
Curso: 21-22
Nota: 9.5
Alumno con DNI:14785236d matriculado en:
Nombre Modulo: Acceso a datos
Curso: 21-22
Nota: 7.0
Alumno con DNI:14785236d matriculado en:
Nombre Modulo: Desarrollo de Interfaces
Curso: 21-22
Nota: 7.5
Se ha pedido el listado con DNI a la base de datos
**** LISTADO DE MATRÍCULAS PARA EL DNI 12345678A ****
Nombre Modulo: Acceso a datos
Curso: 21-22
Nota: 8.0
Nombre Modulo: Desarrollo de Interfaces
Curso: 21-22
Nota: 9.5
BUILD SUCCESSFUL (total time: 0 seconds)
```

Detalle con respuesta evento modoDni.

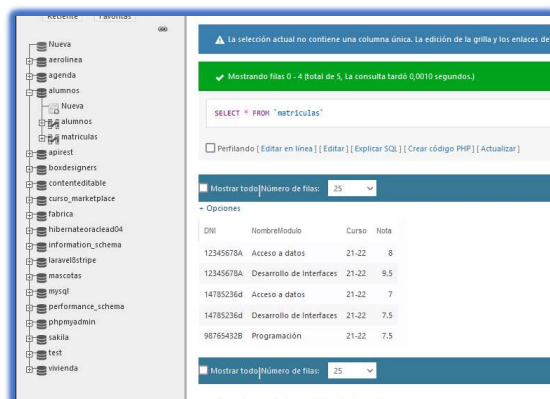
Por último, pruebo el añadir una nueva matrícula, que debe también lanzar su propia respuesta.



```
run:
**** LISTADO GENERAL DE MATRÍCULAS ****
Alumno con DNI:12345678A matriculado en:
Nombre Modulo: Acceso a datos
Curso: 21-22
Nota: 8.0
Alumno con DNI:12345678A matriculado en:
Nombre Modulo: Desarrollo de Interfaces
Curso: 21-22
Nota: 9.5
Alumno con DNI:14785236d matriculado en:
Nombre Modulo: Acceso a datos
Curso: 21-22
Nota: 7.0
Alumno con DNI:14785236d matriculado en:
Nombre Modulo: Desarrollo de Interfaces
Curso: 21-22
Nota: 7.5
Alumno con DNI:98765432B matriculado en:
Nombre Modulo: Programación
Curso: 21-22
Nota: 7.5
Se ha pedido el listado con DNI a la base de datos
**** LISTADO DE MATRÍCULAS PARA EL DNI 12345678A ****
Nombre Modulo: Acceso a datos
Curso: 21-22
Nota: 8.0
Nombre Modulo: Desarrollo de Interfaces
Curso: 21-22
Nota: 9.5
Se ha añadido una nueva matrícula a la base de datos
BUILD SUCCESSFUL (total time: 0 seconds)
```

Detalle de nueva inserción.

Si refrescamos, nuestro **PhpMyAdmin**, podemos ver la nueva fila en nuestra tabla.



Detalle de nueva fila en nuestra tabla matriculas.