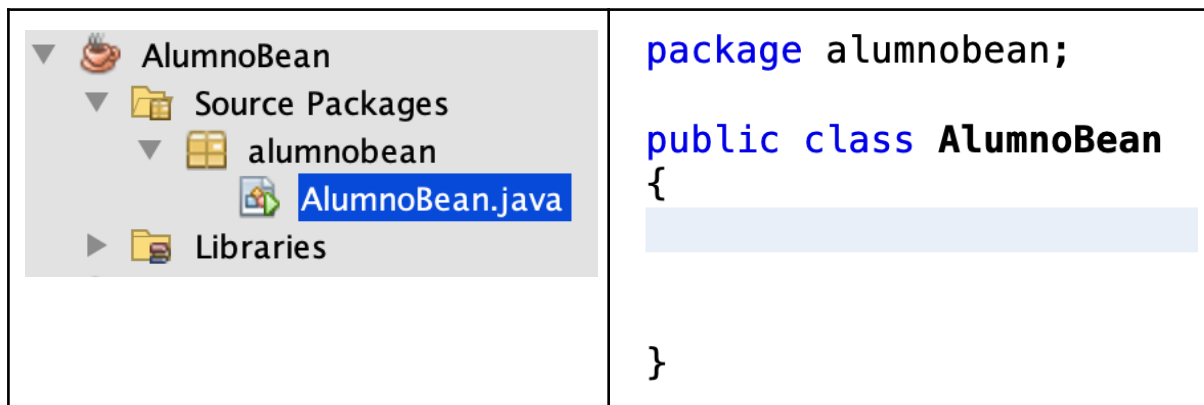


Elaboración de un componente Java (JavaBean)

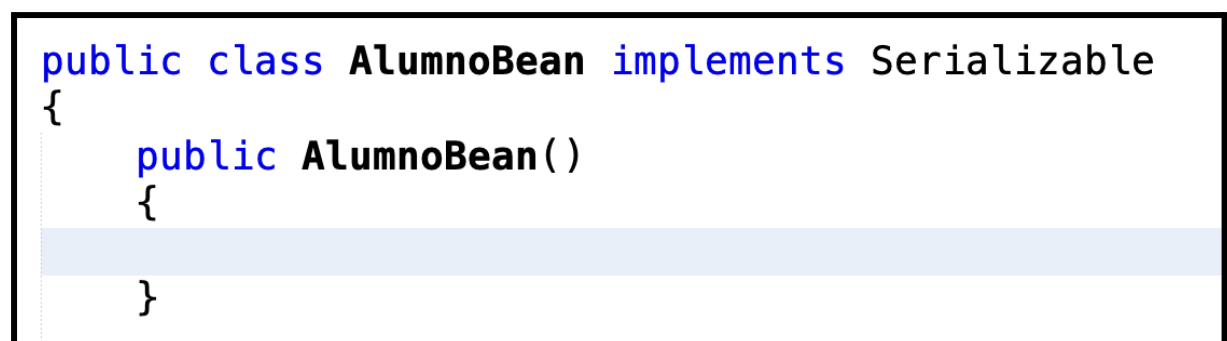
1. Creación del componente.
2. Adición de propiedades.
3. Implementación de su comportamiento.
4. Gestión de los eventos.
5. Uso de componentes ya creados en NetBeans.

1. Creación del componente

Creamos un nuevo proyecto, pero sin método main en la clase principal (AlumnoBean).



Para que una clase se pueda considerar un componente debe implementar la interfaz `Serializable` y, además, tener un **constructor sin argumentos**.



2. Adición de propiedades.

El componente tendrá como propiedades las mismas que los campos de la base de datos: DNI, nombre, apellidos, dirección y fecha de nacimiento.

En el ejemplo concreto que vamos a realizar, como nuestra intención es leer las filas de una tabla de alumnos de la BBDD y almacenarlas en un vector, vamos a necesitar una clase auxiliar llamada Alumno, que contendrá las mismas propiedades que AlumnoBean y las columnas de la tabla Alumno.

2.1 Creación de la clase auxiliar Alumno

```
package alumnobean;
import java.sql.Date;
public class Alumno
{
    private String DNI;
    private String Nombre;
    private String Apellidos;
    private String Direccion;
    private Date FechaNac;

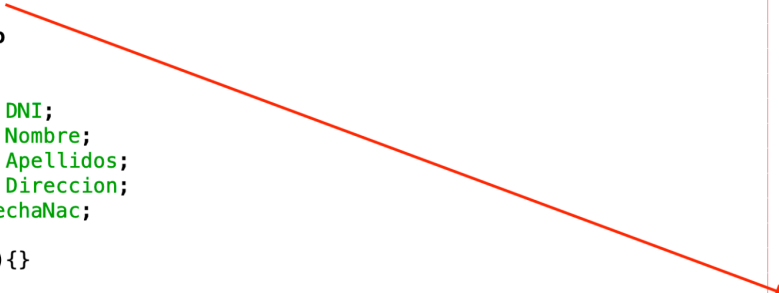
    public Alumno(){}

    public Alumno(String DNI, String Nombre, String Apellidos, String Direccion, Date FechaNac)
    {
        this.DNI = DNI;
        this.Nombre = Nombre;
        this.Apellidos = Apellidos;
        this.Direccion = Direccion;
        this.FechaNac = FechaNac;
    }

    public String getDNI() {
        return DNI;
    }

    public void setDNI(String DNI) {
        this.DNI = DNI;
    }

    public String getNombre() {
        return Nombre;
    }
}
```



**** No se ha incluido la clase entera.**

2.2 Creación de las propiedades en la clase AlumnoBean

Para crear las propiedades, simplemente podemos usar la opción de NetBeans **Source -> Insert Code -> Getter and Setter**.

```
/* Propiedades */

private String DNI;
private String Nombre;
private String Apellidos;
private String Direccion;
private Date FechaNac;

public String getDNI() {
    return DNI;
}

public void setDNI(String DNI) {
    this.DNI = DNI;
}

public String getNombre() {
    return Nombre;
}

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






public String getApellidos() {
    return Apellidos;
}

public void setApellidos(String Apellidos) {
    this.Apellidos = Apellidos;
}

public String getDireccion() {
    return Direccion;
}
```

**** No se ha incluido la clase entera.**

Y aquí podemos ver la tabla en la BBDD Postgre.

Columns							
	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
	DNI	character varying v			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
	Nombre	character varying v			<input type="checkbox"/>	<input type="checkbox"/>	
	Apellidos	character varying v			<input type="checkbox"/>	<input type="checkbox"/>	
	Direccion	character varying v			<input type="checkbox"/>	<input type="checkbox"/>	
	FechaNac	date v			<input type="checkbox"/>	<input type="checkbox"/>	

3. Implementación de su comportamiento.

Con las propiedades listas, tenemos que programar el comportamiento del componente.

En principio la idea es que al crear el componente se cargue el contenido de la **tabla alumno** de la base de datos en un vector de uso interno que nos va a servir para no tener que estar conectándonos constantemente (recuerda que estos componentes se usan en entornos cliente servidor con gran cantidad de accesos por parte de múltiples usuarios que pueden llegar a saturar la base de datos).

```
private void recargarFilas()
{
    try
    {
        Connection con = DriverManager.getConnection("jdbc:postgresql://localhost/ad07",
            "postgres", "sanpedro");

        Statement s = con.createStatement();

        ResultSet rs = s.executeQuery ("select * from alumno");

        while (rs.next())
        {
            Alumno a = new Alumno(rs.getString("DNI"),
                                   rs.getString("Nombre"),
                                   rs.getString("Apellidos"),
                                   rs.getString("Direccion"),
                                   rs.getDate("FechaNac"));

            alumnos.add(a);
        }

        rs.close();
        con.close();
    }
    catch (SQLException ex)
    {
        System.out.println("Error al conectar con la BBDD");
    }
}
```

Por este motivo programaremos al **constructor** para que realice la carga de datos. En un primer momento los valores de las propiedades serán los del primer alumno recuperado.

```
public AlumnoBean()  
{  
    recargarFilas();  
}
```

También generaremos un par de métodos para recuperar información de un registro según su **posición** o según su **clave**.

<pre>/* * @param i numero de la fila a cargar en las propiedades del componente */ public Alumno seleccionarFila(int i) { if(i < alumnos.size()) { return alumnos.get(i); } else { return null; } }</pre>	<pre>/* * @param DNI DNI A buscar, se carga en las propiedades del componente */ public Alumno seleccionarDNI(String DNI) { int i=0; while((i< alumnos.size()) && (!DNI.equals(alumnos.get(i).getDNI()))) { i++; } if (i == alumnos.size()) { return null; } else { return alumnos.get(i); } }</pre>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

4. Gestión de los eventos

Los componentes Java utilizan el modelo de delegación de eventos para gestionar la comunicación entre objetos.

Para ilustrar el uso de los eventos en un componente JavaBean le añadiremos un comportamiento extra que añadirá un nuevo alumno a la base de datos. Tendrá como especial característica que cada vez que se inserte un alumno nuevo se generará un evento que alerte de esta modificación.

Esto que en principio puede parecer redundante, no lo es en un entorno multiusuario en el que varios clientes se conecten simultáneamente a la base de datos.

Para poder implementar esto necesitarás varias cosas:

1. Una **clase** que implemente los eventos. Esta clase hereda de `java.util.EventObject`. En el ejemplo se llamará `BDModificadaEvent`.

```
package alumnobean;

public class BDModificadaEvent extends java.util.EventObject
{
    //Constructor
    public BDModificadaEvent(Object source)
    {
        super(source);
    }
}
```

2. Una **interfaz** que defina los métodos a usar cuando se genere el evento. Implementa `java.util.EventListener`. En este caso la gestión del evento se hará a través del método `capturarBDModificada` de la interfaz `BDModificadaListener`.

```
package alumnobean;

import java.util.EventListener;

public interface BDModificadaListener extends EventListener
{
    public void capturarBDModificada(BDModificadaEvent ev);
}
```

- 2.1. En la clase **AlumnoBean**, crearemos un objeto de tipo **BDModificadaListener** llamado **receptor** que representa aquellos programas susceptibles de recibir el evento.

```
/* Objeto que recibirá la notificación de que la BD ha sido actualizada */  
private BDModificadaListener receptor;
```

3. Dos **métodos**, **addEventListener** y **removeEventListener** que permitan al componente añadir oyentes y eliminarlos. En principio se deben encargar de que pueda haber varios oyentes. En nuestro caso sólo vamos a tener un oyente, pero se suele implementar para admitir a varios.

```
/* Métodos para asignar o eliminar el objeto que recibirá la notificación de que la BD,  
ha sido actualizada */  
public void addBDModificadaListener(BDModificadaListener receptor)  
{  
    this.receptor = receptor;  
}  
  
public void removeBDModificadaListener(BDModificadaListener receptor)  
{  
    this.receptor=null;  
}
```

4. Implementar el **método** que lanza el evento, asegurándonos de que todos los oyentes reciban el aviso. En este caso lo que se ha hecho es lanzar el método que se creó en la interfaz que describe al oyente.

```

/*****
 * Método que añade un alumno 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 addAlumno(Alumno a)
{
    try
    {
        Connection con = DriverManager.getConnection("jdbc:postgresql://localhost/ad07",
            "postgres", "sanpedro");

        PreparedStatement s = con.prepareStatement("insert into alumno values (?, ?, ?, ?, ?)");

        s.setString(1, a.getDNI());
        s.setString(2, a.getNombre());
        s.setString(3, a.getApellidos());
        s.setString(4, a.getDireccion());
        s.setDate(5, a.getFechaNac());

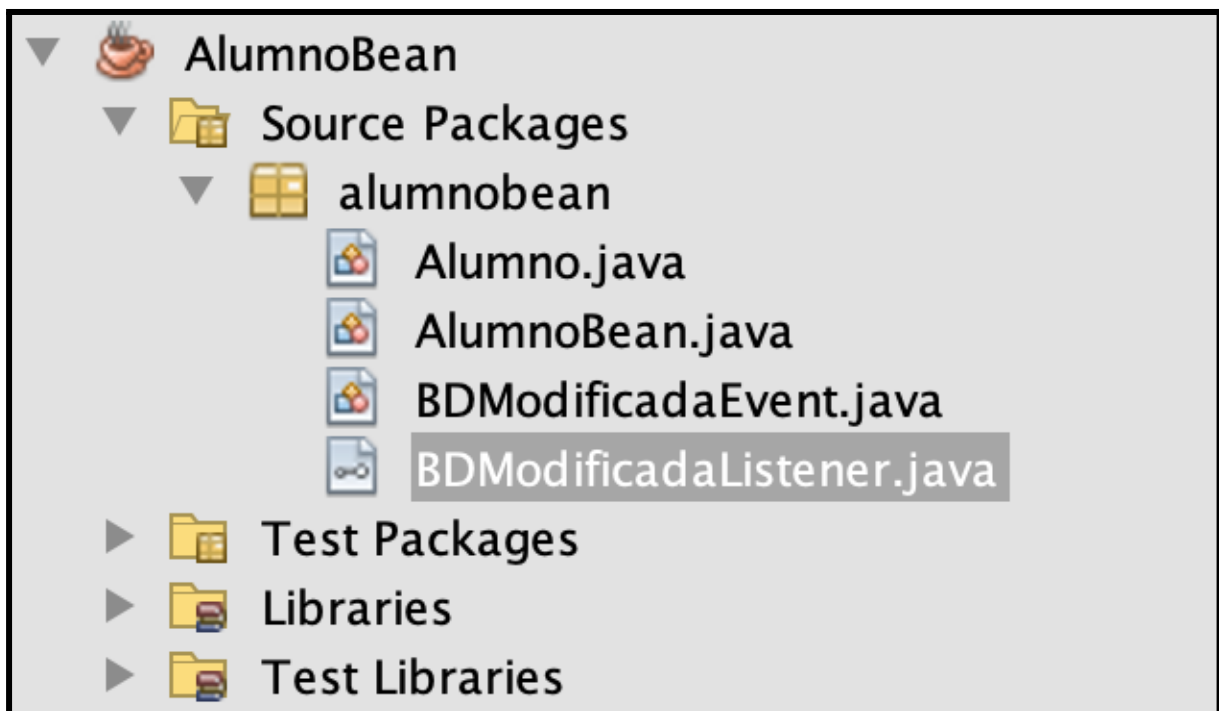
        s.executeUpdate ();

        recargarFilas();

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

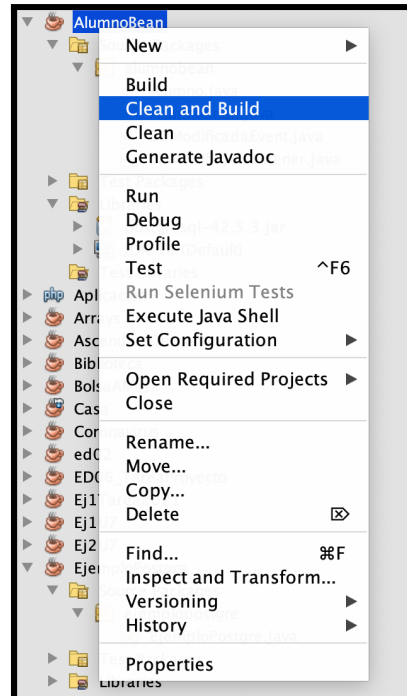
```

El proyecto queda de la siguiente manera:

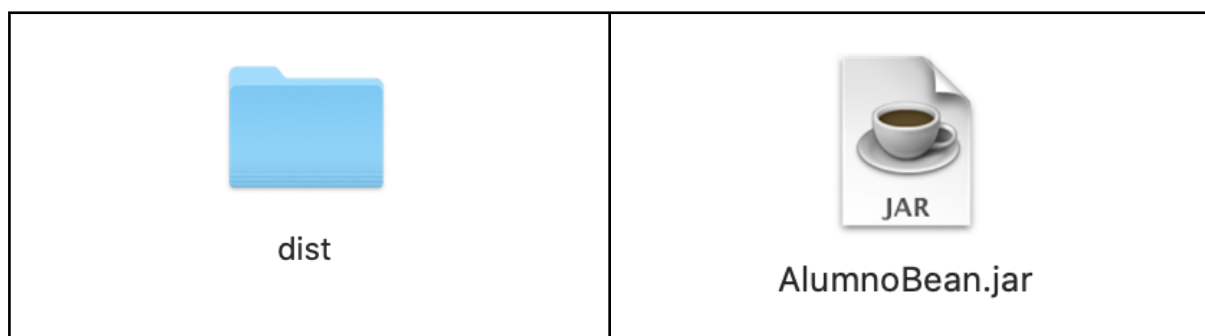


5. Uso de componentes ya creados en NetBeans.

Para poder usar este componente creado, debemos lanzar la orden “*Clean and Build*” desde el proyecto de *NetBeans*.

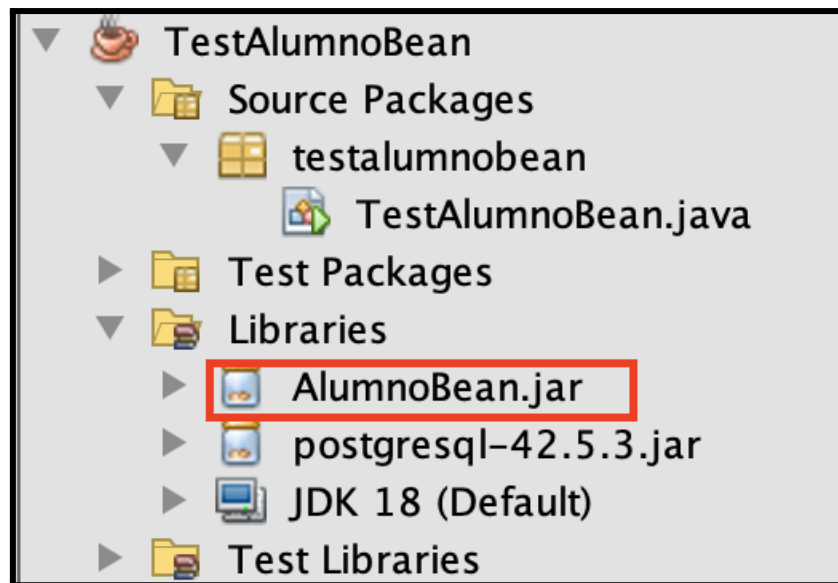


Después, debemos irnos a la carpeta del proyecto de NetBeans (AlumnoBean), y accederemos a la carpeta **dist**, donde se encuentra el archivo **jar** generado en el paso anterior.



Ahora vamos a crear un proyecto nuevo para probar el componente. Creamos un nuevo proyecto llamado **TestAlumnoBean**.

Añadimos el fichero **AlumnoBean.jar** como una librería del proyecto.



El código del **TestAlumnoBean** es el siguiente:

```
import alumnobean.Alumno;
import alumnobean.AlumnoBean;
import alumnobean.BDModificadaEvent;
import alumnobean.BDModificadaListener;
import java.sql.Date;
import java.time.LocalDate;

public class TestAlumnoBean implements BDModificadaListener 1
{
    static AlumnoBean alumnos;

    public static void main(String[] args)
    {
        TestAlumnoBean tab = new TestAlumnoBean(); 3.1
        tab.run(args);
    }

    @Override
    public void capturarBDModificada(BDModificadaEvent bdme) 2
    {
        System.out.println("Se ha añadido un elemento a la base de datos");
    }

    public void run (String[] args)
    {
        //Se crea el componente
        alumnos = new AlumnoBean();
        //Se establece al objeto tab como escuchador 3.2
        alumnos.addBDModificadaListener(this);

        String dni="11111111A";
        System.out.printf("Alumno con DNI:%s --> %s\n",dni,alumnos.seleccionarDNI(dni));
        System.out.printf("Alumno en la posición 0 --> %s\n",alumnos.seleccionarFila(0));

        //Actualizamos los datos de un alumno
        Alumno a = new Alumno();
        a.setDNI("33333333C");
        a.setNombre("Juan");
        a.setApellidos("Jiménez");
        a.setDireccion("Alameda de San Antón, 12");
        a.setFechaNac(Date.valueOf(LocalDate.now()));

        //Añadimos un alumno a la BBDD
        alumnos.addAlumno(a);
    }
}
```

1 - La clase **TestAlumnoBean** debe implementar la interfaz **BDModificadaListener** para poder escuchar cuándo se actualiza la BD.

2 - Al implementar la interfaz **BDModificadaListener**, la clase está obligada a implementar el método **capturarBDModificada**.

Los apartados 3.1 y 3.2 son necesarios como consecuencia del problema de no poder usar el puntero **this**, dentro del método **main**. Ya que este puntero pertenece a un objeto y no a la clase, y por lo tanto, no es posible usar una variable no estática en un entorno estático como es el método **main**.

3.1 - Creamos un objeto del mismo tipo que la clase principal, y llamamos al método run, que es el que hace el trabajo, que no es posible hacer directamente desde el main.

3.2 - Se asigna a la clase principal como escuchador del evento ***BDModificadaEvent***.