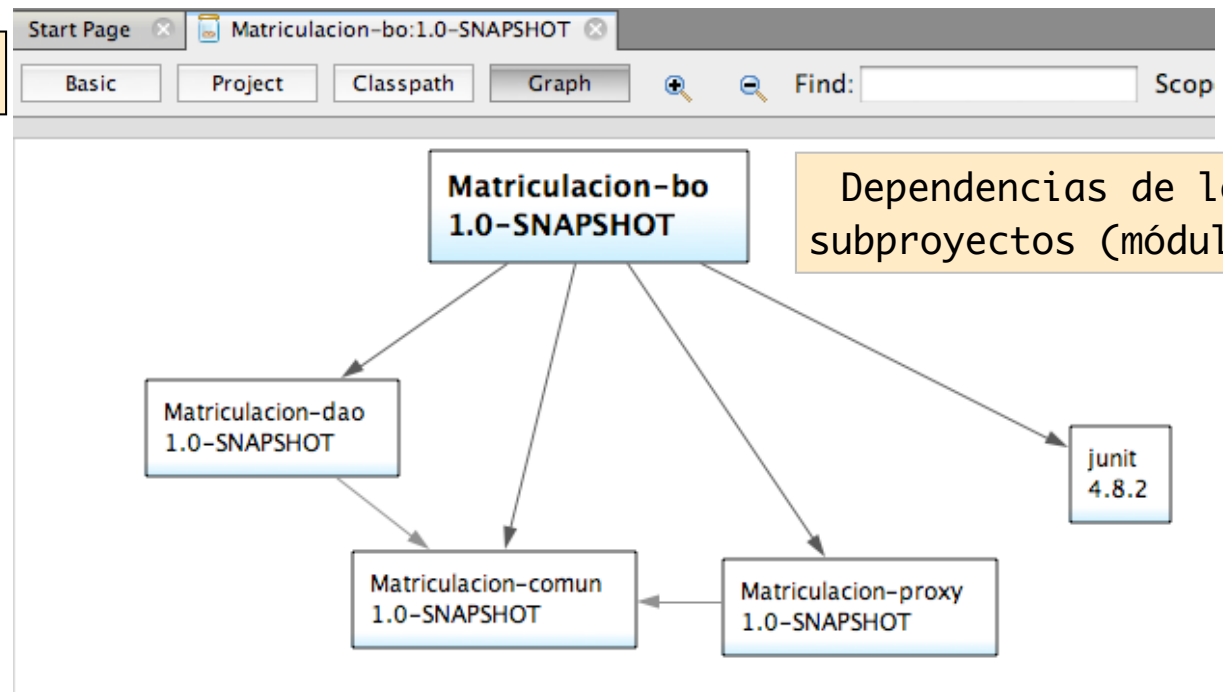
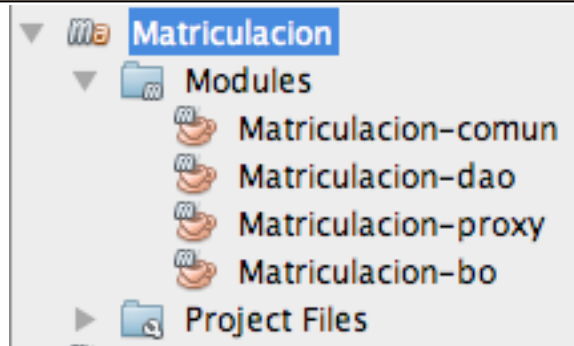
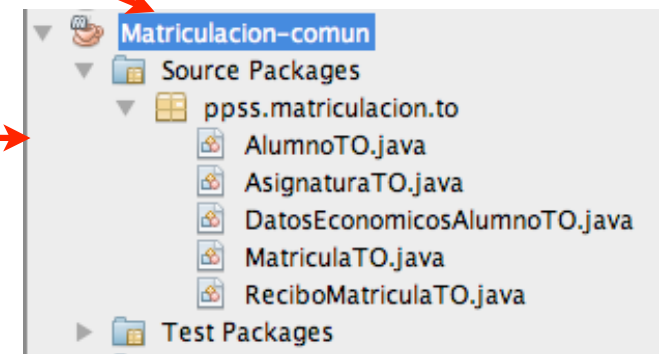
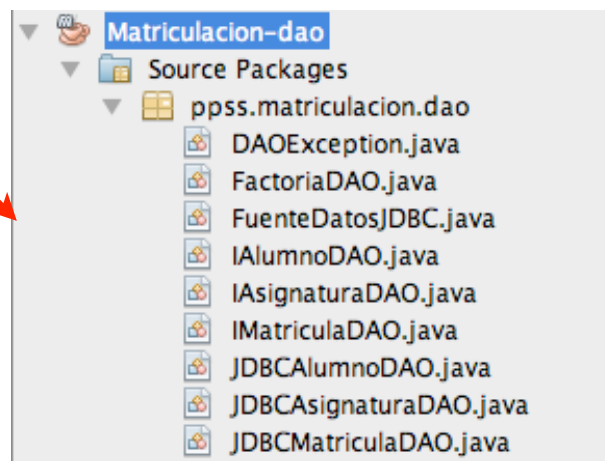
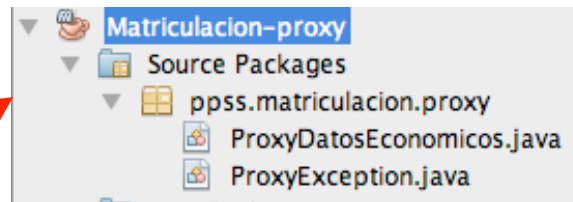
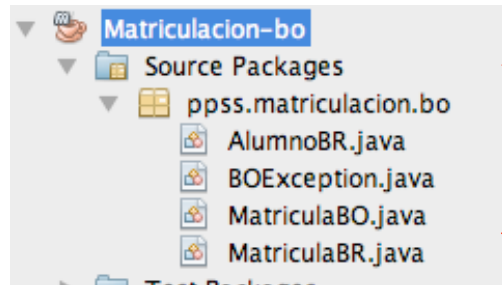
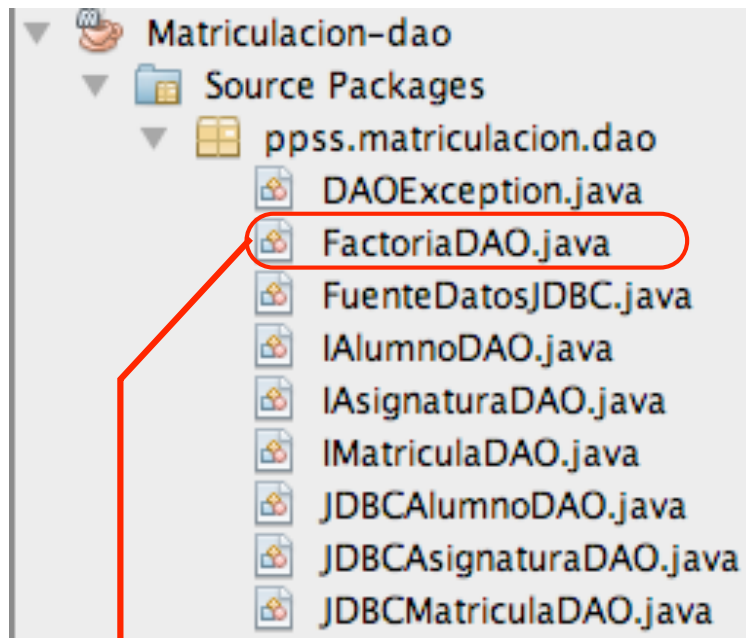


# Proyecto Matriculacion



Dependencias de los subproyectos (módulos)



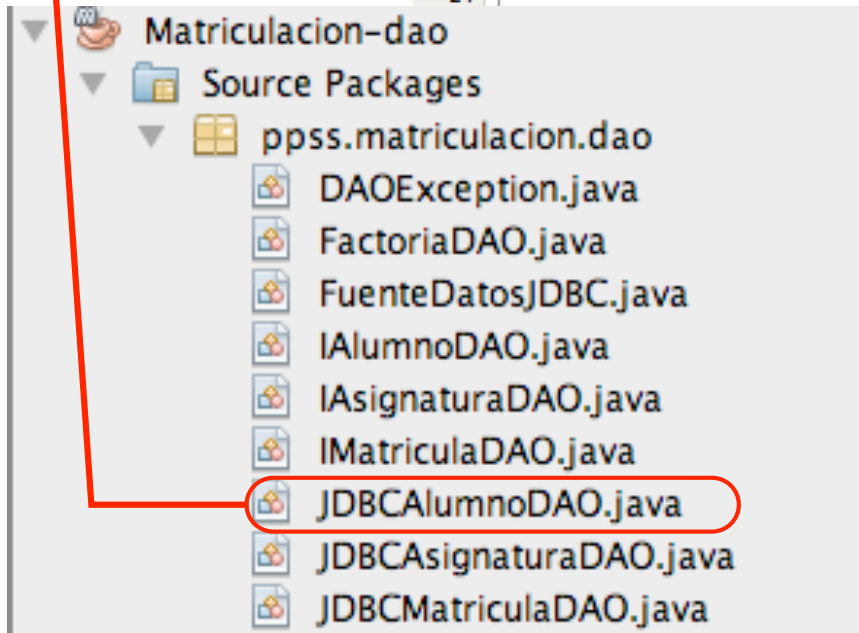


FactoriaDAO proporciona  
DAOS JCBC  
(Patrón Factory Method)

```
Start Page x FactoriaDAO.java x
Source History
1 package ppss.matriculacion.dao;
2
3 /**
4  * La clase <code>GestorDAO</code> se utilizará como factoría de los ob
5  * a datos de la aplicación.
6  *
7  */
8
9 public class FactoriaDAO {
10     /**
11      * Devuelve al DAO para acceder a los datos de los alumnos.
12      * @return DAO que da acceso a los alumnos.
13      */
14     public IAlumnoDAO getAlumnoDAO() {
15         return new JDBCAlumnoDAO();
16     }
17
18     /**
19      * Devuelve al DAO para acceder a los datos de las asignaturas.
20      * @return DAO que da acceso a las asignaturas.
21      */
22     public IAsignaturaDAO getAsignaturaDAO() {
23         return new JDBCAsignaturaDAO();
24     }
25
26     /**
27      * Devuelve al DAO para acceder a los datos de matriculación.
28      * @return DAO que da acceso a las matriculaciones.
29      */
30     public IMatriculaDAO getMatriculaDAO() {
31         return new JDBCMatriculaDAO();
32     }
33 }
```

```
Start Page x FactoriaDAO.java x JDBCAlumnoDAO.java x FuenteDatosJDBC.java x
Source History
1 package ppss.matriculacion.dao;
2
3 import java.sql.Connection;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7 import java.util.ArrayList;
8 import java.util.List;
9
10 import ppss.matriculacion.to.AlumnoTO;
11
12 /**
13  * Implementación del acceso a los datos de los alumnos almacenados en una base de
14  * datos.
15  *
16  */
17
18 public class JDBCAlumnoDAO implements IAlumnoDAO {
19
20     public void addAlumno(AlumnoTO a) throws DAOException {
21         Connection con = null;
22         PreparedStatement ps = null;
23
24         if(a==null) {
25             throw new DAOException("El alumno a insertar no puede ser nulo");
26         }
27
28         con = FuenteDatosJDBC.getInstance().getConnection();
29         ps = con.prepareStatement("INSERT INTO alumnos(nif, nombre, direccion, email, fechaNacimiento) VALUES(?, ?, ?, ?, ?)");
30         ps.setString(1, a.getNif());
31         ps.setString(2, a.getNombre());
32         ps.setString(3, a.getDireccion());
33         ps.setString(4, a.getEmail());
34         ps.setDate(5, new java.sql.Date(a.getFechaNacimiento().getTime()));
35         ps.executeUpdate();
36     }
37
38     public List<AlumnoTO> getAllAlumnos() throws DAOException {
39         List<AlumnoTO> alumnos = new ArrayList<>();
40         Connection con = null;
41         PreparedStatement ps = null;
42         ResultSet rs = null;
43
44         con = FuenteDatosJDBC.getInstance().getConnection();
45         ps = con.prepareStatement("SELECT * FROM alumnos");
46         rs = ps.executeQuery();
47         while(rs.next()) {
48             AlumnoTO alumno = new AlumnoTO(rs.getString(1), rs.getString(2), rs.getString(3), rs.getString(4), rs.getDate(5));
49             alumnos.add(alumno);
50         }
51         return alumnos;
52     }
53
54     public AlumnoTO getAlumnoById(int id) throws DAOException {
55         AlumnoTO alumno = null;
56         Connection con = null;
57         PreparedStatement ps = null;
58         ResultSet rs = null;
59
60         con = FuenteDatosJDBC.getInstance().getConnection();
61         ps = con.prepareStatement("SELECT * FROM alumnos WHERE id = ?");
62         ps.setInt(1, id);
63         rs = ps.executeQuery();
64         if(rs.next()) {
65             alumno = new AlumnoTO(rs.getString(1), rs.getString(2), rs.getString(3), rs.getString(4), rs.getDate(5));
66         }
67         return alumno;
68     }
69
70     public void deleteAlumno(int id) throws DAOException {
71         Connection con = null;
72         PreparedStatement ps = null;
73
74         con = FuenteDatosJDBC.getInstance().getConnection();
75         ps = con.prepareStatement("DELETE FROM alumnos WHERE id = ?");
76         ps.setInt(1, id);
77         ps.executeUpdate();
78     }
79
80     public void updateAlumno(AlumnoTO a) throws DAOException {
81         Connection con = null;
82         PreparedStatement ps = null;
83
84         con = FuenteDatosJDBC.getInstance().getConnection();
85         ps = con.prepareStatement("UPDATE alumnos SET nombre = ?, direccion = ?, email = ?, fechaNacimiento = ? WHERE id = ?");
86         ps.setString(1, a.getNombre());
87         ps.setString(2, a.getDireccion());
88         ps.setString(3, a.getEmail());
89         ps.setDate(4, new java.sql.Date(a.getFechaNacimiento().getTime()));
90         ps.setInt(5, a.getId());
91         ps.executeUpdate();
92     }
93 }
```

JDBCAlumnoDAO implementa la interfaz IAlumnoDAO



JDBCAlumnoDAO implementa operaciones CRUD de los objetos TO

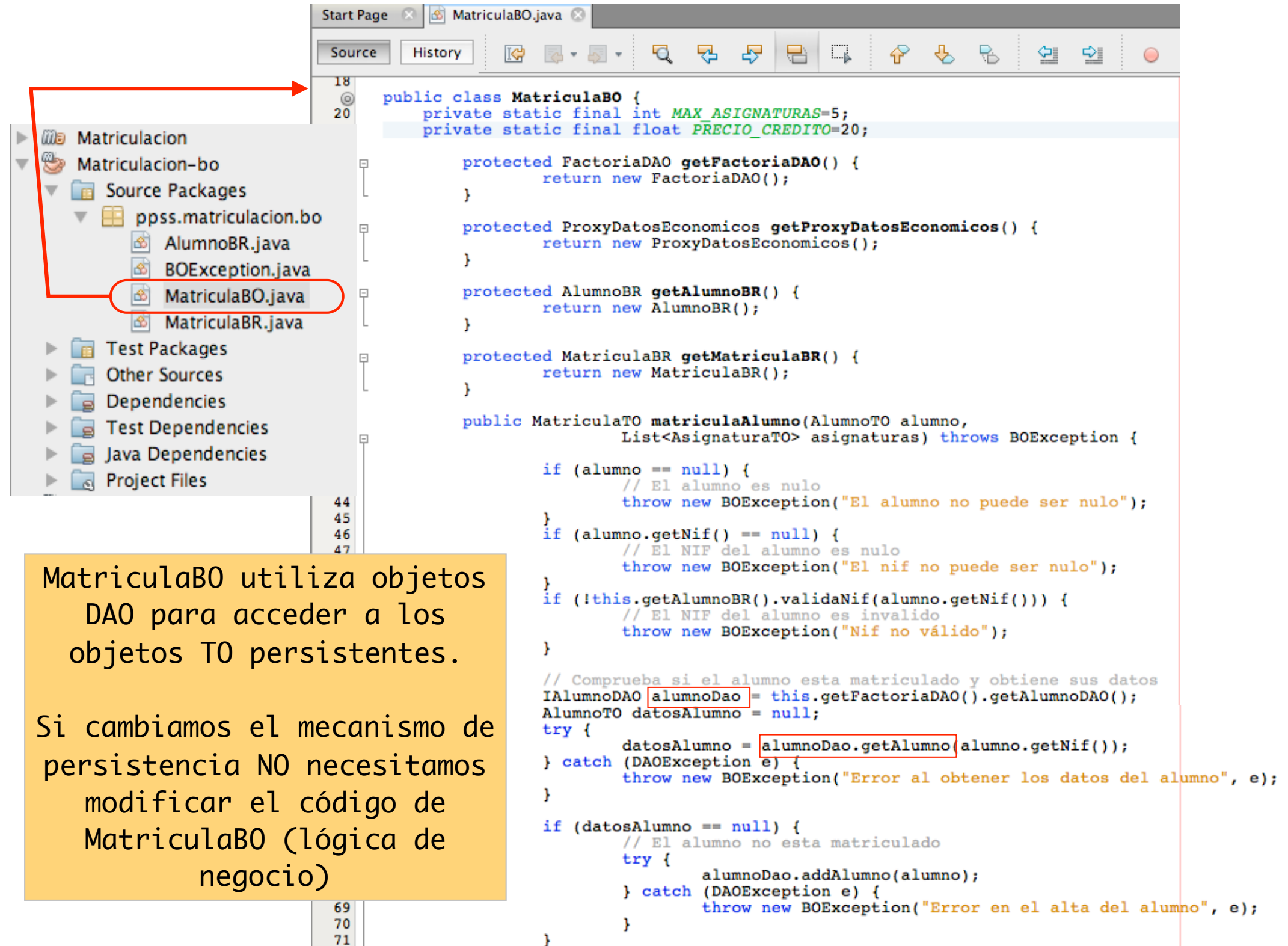
The image shows an IDE window with the file `IAlumnoDAO.java` open. The file is part of the `ppss.matriculacion.dao` package. The code defines an interface `IAlumnoDAO` with three methods: `addAlumno`, `delAlumno`, and `getAlumno`. The project structure on the left shows the file's location within the `Matriculacion-dao` source package.

```
1 package ppss.matriculacion.dao;
2
3 import java.util.List;
4 import ppss.matriculacion.to.AlumnoTO;
5
6
7 /**
8  * Interfaz común de los objetos que dan acceso a los datos de los alumnos.
9  */
10 public interface IAlumnoDAO {
11
12     /**
13      * De de alta una alumno.
14      * @param a Alumno que se añadirá. Se producirá un error si el alumno
15      * ya existe, o si el parámetro o su campo nif es null.
16      * @throws DAOException Si ocurre un error al añadir al alumno
17      */
18     public abstract void addAlumno(AlumnoTO a) throws DAOException;
19
20     /**
21      * De de baja un alumno.
22      * @param nif Nif del alumno a eliminar. Se producirá un error si el alumno no
23      * existe, o si el parámetro es null.
24      * @throws DAOException Si ocurre un error al eliminar al alumno
25      */
26     public abstract void delAlumno(String nif) throws DAOException;
27
28     /**
29      * Obtiene los datos de un alumno.
30      * @param nif Nif del alumno del cual queremos obtener los datos
31      * @return Datos del alumno, o null si el alumno no existe.
32      * @throws DAOException Si ocurre un error al recuperar los datos
33      */
34     public abstract AlumnoTO getAlumno(String nif) throws DAOException;
35 }
```

Matriculacion-dao  
Source Packages  
ppss.matricula  
DAOExcept  
FactoriaDA  
FuentesDatosJDBC.java  
IAlumnoDAO.java  
IASignaturaDAO.java  
IMatriculaDAO.java  
JDBCAlumnoDAO.java  
JDBCAsignaturaDAO.java  
JDBCMatriculaDAO.java

IAlumnoDAO especifica operaciones CRUD  
(Create, Retrieve, Update, Delete)





```
18 public class MatriculaBO {
19     private static final int MAX_ASIGNATURAS=5;
20     private static final float PRECIO_CREDITO=20;

    protected FactoriaDAO getFactoriaDAO() {
        return new FactoriaDAO();
    }

    protected ProxyDatosEconomicos getProxyDatosEconomicos() {
        return new ProxyDatosEconomicos();
    }

    protected AlumnoBR getAlumnoBR() {
        return new AlumnoBR();
    }

    protected MatriculaBR getMatriculaBR() {
        return new MatriculaBR();
    }

    public MatriculaTO matriculaAlumno(AlumnoTO alumno,
        List<AsignaturaTO> asignaturas) throws BOException {

        if (alumno == null) {
            // El alumno es nulo
            throw new BOException("El alumno no puede ser nulo");
        }
        if (alumno.getNif() == null) {
            // El NIF del alumno es nulo
            throw new BOException("El nif no puede ser nulo");
        }
        if (!this.getAlumnoBR().validaNif(alumno.getNif())) {
            // El NIF del alumno es invalido
            throw new BOException("Nif no válido");
        }

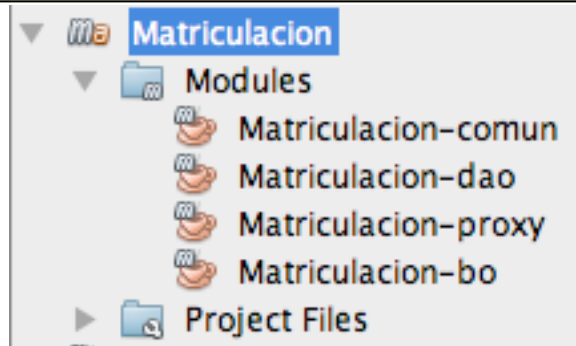
        // Comprueba si el alumno esta matriculado y obtiene sus datos
        IAlumnoDAO alumnoDao = this.getFactoriaDAO().getAlumnoDAO();
        AlumnoTO datosAlumno = null;
        try {
            datosAlumno = alumnoDao.getAlumno(alumno.getNif());
        } catch (DAOException e) {
            throw new BOException("Error al obtener los datos del alumno", e);
        }

        if (datosAlumno == null) {
            // El alumno no esta matriculado
            try {
                alumnoDao.addAlumno(alumno);
            } catch (DAOException e) {
                throw new BOException("Error en el alta del alumno", e);
            }
        }
    }
}
```

MatriculaBO utiliza objetos DAO para acceder a los objetos TO persistentes.

Si cambiamos el mecanismo de persistencia NO necesitamos modificar el código de MatriculaBO (lógica de negocio)

# Proyecto Matriculacion



## Posible estrategia de integración

1

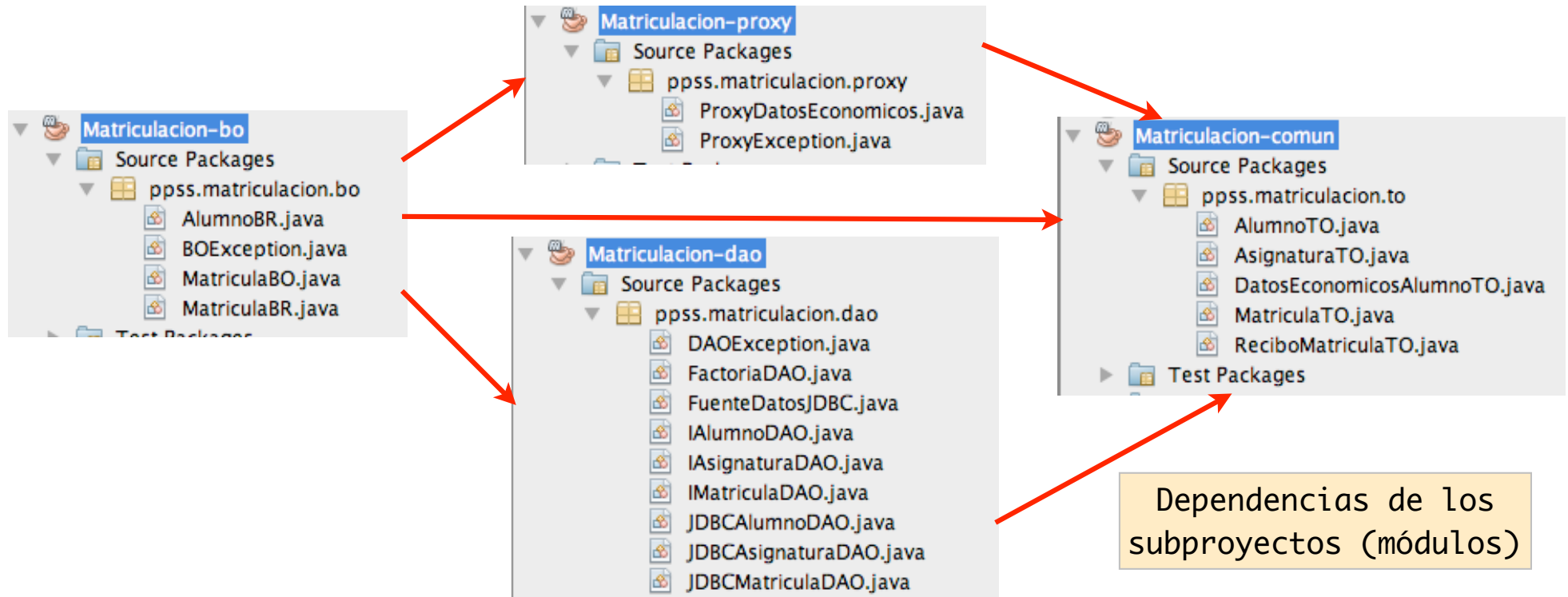
Matriculacion-dao

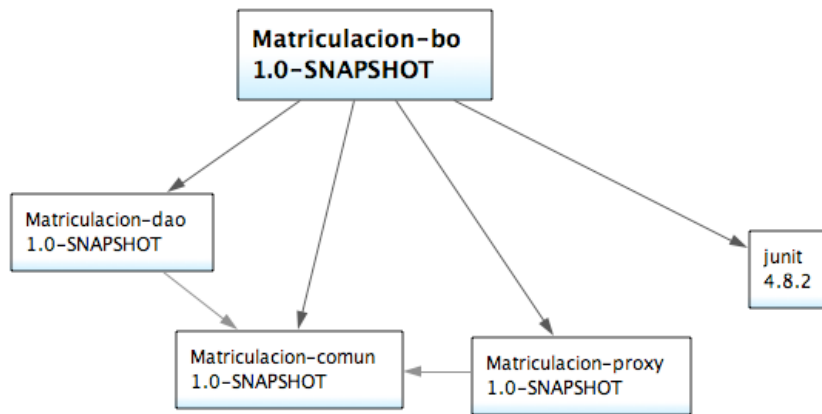
2

Matriculacion-bo + Matriculacion-dao

3

Matriculacion-bo + Matriculacion-dao +  
Matriculacion-proxy





El orden de ejecución de los tests debería ser: 1, 2, 3  
Para ello necesitaríamos agrupar en categorías los tests de Matriculación-bo

Para automatizar las pruebas necesitaremos implementar en cada una de las fases de la integración:

- 1 **Matriculacion-dao**

Utilizamos DbUnit para implementar Tests de integración con la BD:  
`/src/test/java/AlumnoDAOIT.java, /src/test/java/AsignaturaDAOIT.java, /src/test/java/MatriculaDAOIT.java`  
*implementados en el proyecto Matriculacion-dao* ↗ mvn verify
- 2 **Matriculacion-bo + Matriculacion-dao**

*implementados en el proyecto Matriculacion-bo* ↗ Implementamos:

  - stubs para Matriculación-proxy
  - tests DBUnit:
    - `/src/test/java/MatriculaBO-daoIT.java`  
categoría Integracion-fase2  
mvn verify -D.groups="Integracion-fase2"
- 3 **Matriculacion-bo + Matriculacion-dao + Matriculacion-proxy**

*implementados en el proyecto Matriculacion-bo* ↗ Implementamos:

  - tests DBUnit:
    - `/src/test/javaMatriculaBO-dao-proxyIT.java`  
categoría Integracion-fase3  
mvn verify -D.groups="Integracion-fase3"