DAM
PROGRAMACIÓN

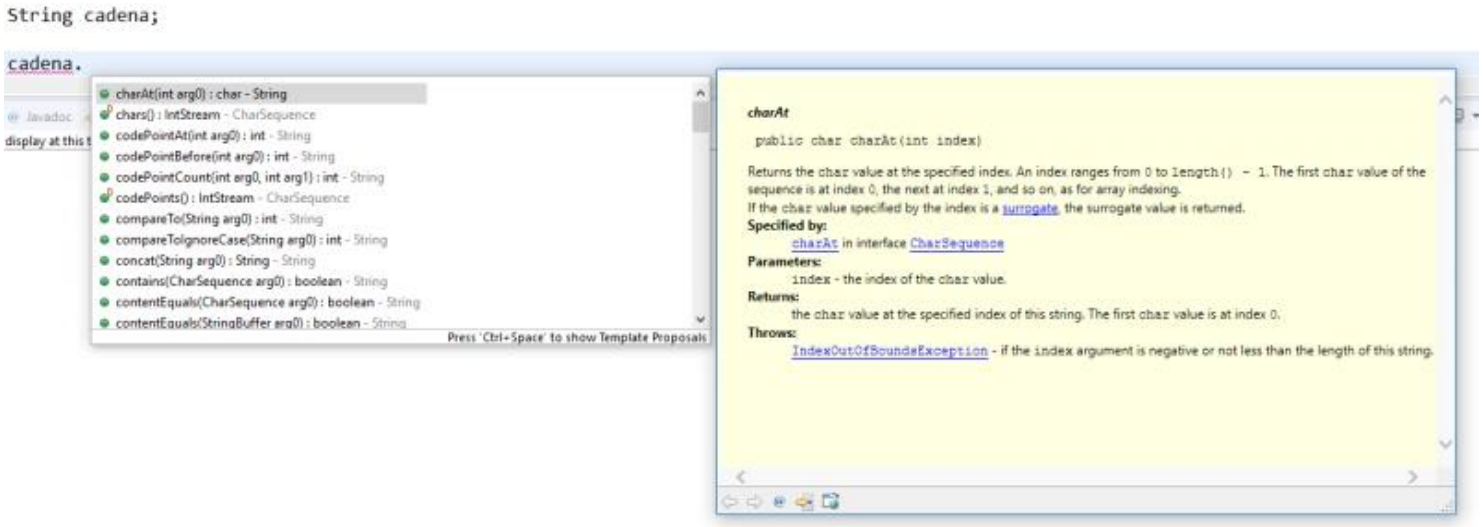
LIBRERÍAS
JAVADOC

Javadoc. Introducción

- **Javadoc** es la herramienta de Java para generar documentación de APIs en formato HTML.
- La documentación **Javadoc** se genera a partir de comentarios son la siguiente sintaxis:

```
/**
 *     Esto es un comentario para javadoc
 */
```

- La documentación facilita:
  - la comprensión del código por los programadores,
  - el mantenimiento del código.



Javadoc. Elementos a documentar

- Elementos a incluir al documentar una clase:
  - Nombre de la clase, descripción general, nº de versión, nombre de autores.
  - Documentación de cada constructor/método, que deben incluir:
    - nombre del constructor/método,
    - tipo de la variable de retorno,
    - descripción de los parámetros (nombre, tipo, significado, etc.).
- Interpretación de la documentación:
  - Si el comentario está justo antes de la declaración de clase se considera un comentario de clase.
  - Si el comentario está justo antes de un método/constructor se considera un comentario de ese método/constructor.

Javadoc. Ejemplo

```
/**
* Returns a new string that is a substring of this string. The
* substring begins with the character at the specified index and
* extends to the end of this string. <p>
* Examples:
* <blockquote><pre>
* "unhappy".substring(2) returns "happy"
* "Harbison".substring(3) returns "bison"
* "emptiness".substring(9) returns "" (an empty string)
* </pre></blockquote>
*
* @param     beginIndex   the beginning index, inclusive.
* @return    the specified substring.
* @exception  IndexOutOfBoundsException  if
*             <code>beginIndex</code> is negative or larger than the
*             length of this <code>String</code> object.
*/
```

Documentación Javadoc

```
public String substring(int beginIndex) {
        if (beginIndex < 0) {
            throw new StringIndexOutOfBoundsException(beginIndex);
        }
        int subLen = value.length - beginIndex;
        if (subLen < 0) {
            throw new StringIndexOutOfBoundsException(subLen);
        }
        return (beginIndex == 0) ? this : new String(value, beginIndex, subLen);
    }
```

Método que se documenta

substring

```
public String substring(int beginIndex)
```

Returns a string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.

Examples:

```
"unhappy".substring(2) returns "happy"
"Harbison".substring(3) returns "bison"
"emptiness".substring(9) returns "" (an empty string)
```

Parameters:

`beginIndex` - the beginning index, inclusive.

Returns:

the specified substring.

Throws:

`IndexOutOfBoundsException` - if `beginIndex` is negative or larger than the length of this `String` object.

Documentación HTML generada

Acceso a la documentación Javadoc del método **substring** mientras se implementa código Java

```
/*
 * To change this license header,
 * To change this template file, ch
 * and open the template in the ed
 */
package gal.teis.aaa.pgr.ejemplos;

/**
 *
 * @author Esther Ferreiro
 */
public class Principal_1 {

    /**
     * @param args the command lin
     */
    public static void main(String
        String nombreCompleto="And
        String nombre=nombreCompleto.su
    }

}
```

java.lang.String

```
public String substring(int beginIndex)
```

Returns a string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.

Examples:

```
"unhappy".substring(2) returns "happy"
"Harbison".substring(3) returns "bison"
"emptiness".substring(9) returns "" (an empty string)
```

Parameters:

beginIndex - the beginning index, inclusive

```
substring(int beginIndex)                    String
substring(int beginIndex, int endIndex)      String
subSequence(int beginIndex, int endIndex)    CharSequence
```

Instance Members; Press 'Ctrl+SPACE' Again for All Items

Javadoc. Elementos a documentar

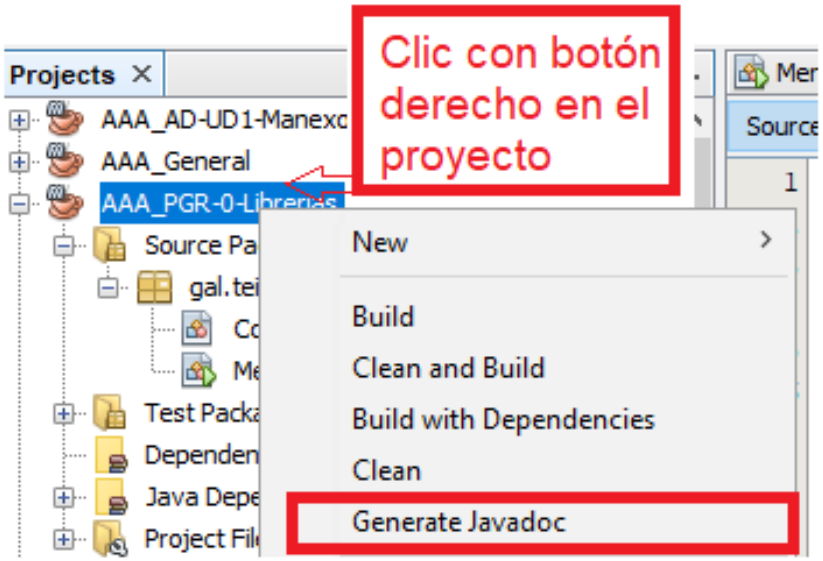
TAG	DESCRIPCIÓN	INCLUYE	TIPO
@autor	Nombre del desarrollador.	Nombre autor o autores	Resumen, paquete, clase, interfaz.
@deprecated	Indica que el método o clase es obsoleto y que no se recomienda su uso.	Descripción del elemento obsoleto.	Paquete, clase, interfaz, atributo, constructor, método.
@param	Definición de un parámetro de un método o constructor.	Nombre de parámetro y descripción	Constructor, método.
@return	Informa del valor que devuelve el método.	Descripción del valor de retorno	Constructor, método.
@see	Asocia con otro método o clase.	Referencia cruzada referencia (#método(); clase#método(); paquete.clase; paquete.clase#método()).	Paquete, clase, interfaz, atributo, constructor, método.
@throws	Nombre de la clase y descripción	Información sobre la excepción que se puede producir en el método.	Métodos
{@value}	Se utiliza con atributos estáticos.	Muestra los posibles valores del atributo.	Atributos estáticos
@version	Versión del método o clase.	Versión	Resumen, paquete, clase, interfaz

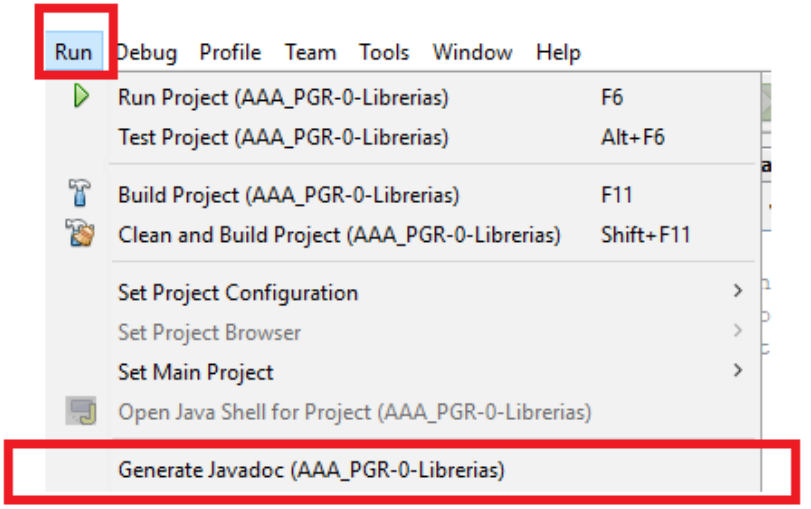
Dentro de estos tags se pueden usar HTML, p.e., para poner un link que aporte + información sobre el tag

Otra documentación

- A la hora de planificar la documentación de un proyecto deberíamos incluir un fichero **readme.txt** con la siguiente información:
  - título del proyecto,
  - descripción,
  - versión,
  - pasos para arrancar el proyecto,
  - autores del proyecto,
  - instrucciones para los usuarios relevantes a la hora de ejecutar o utilizar el proyecto.
- Ejemplo de fichero readme.txt de software aquí

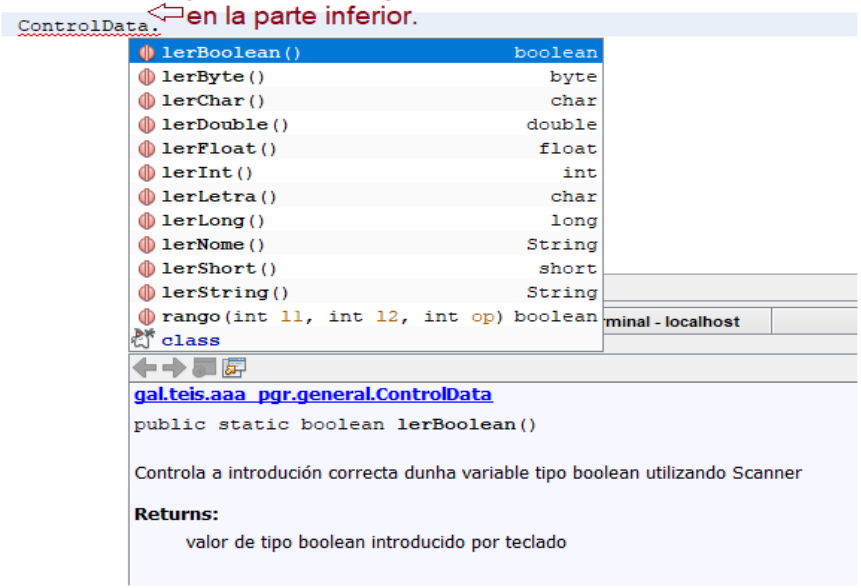
Javadoc. Generar desde NetBeans



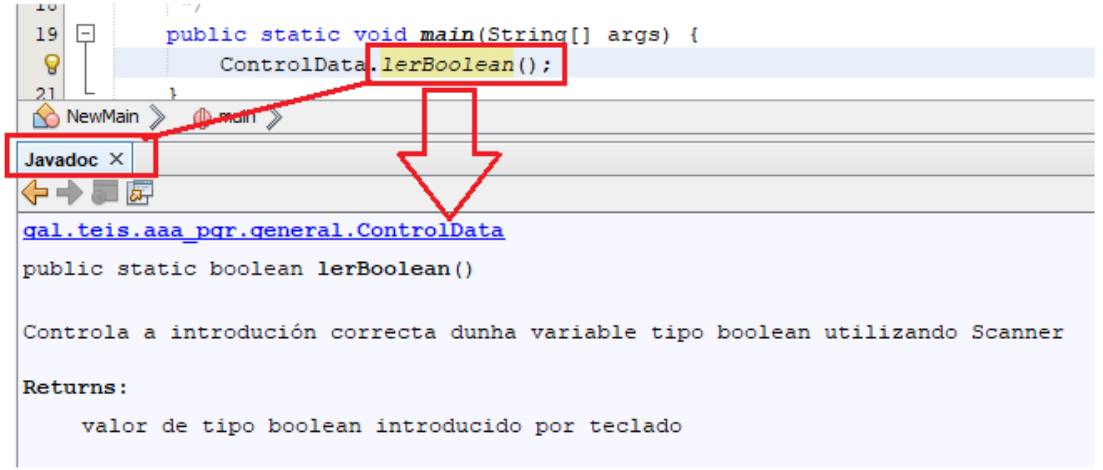


Javadoc. Acceso a documentación

Al escribir la clase se muestran los métodos a los que se puede acceder y la documentación de cada uno de ellos en la parte inferior.



Los fichero *.HTML se pueden visualizar con un navegador

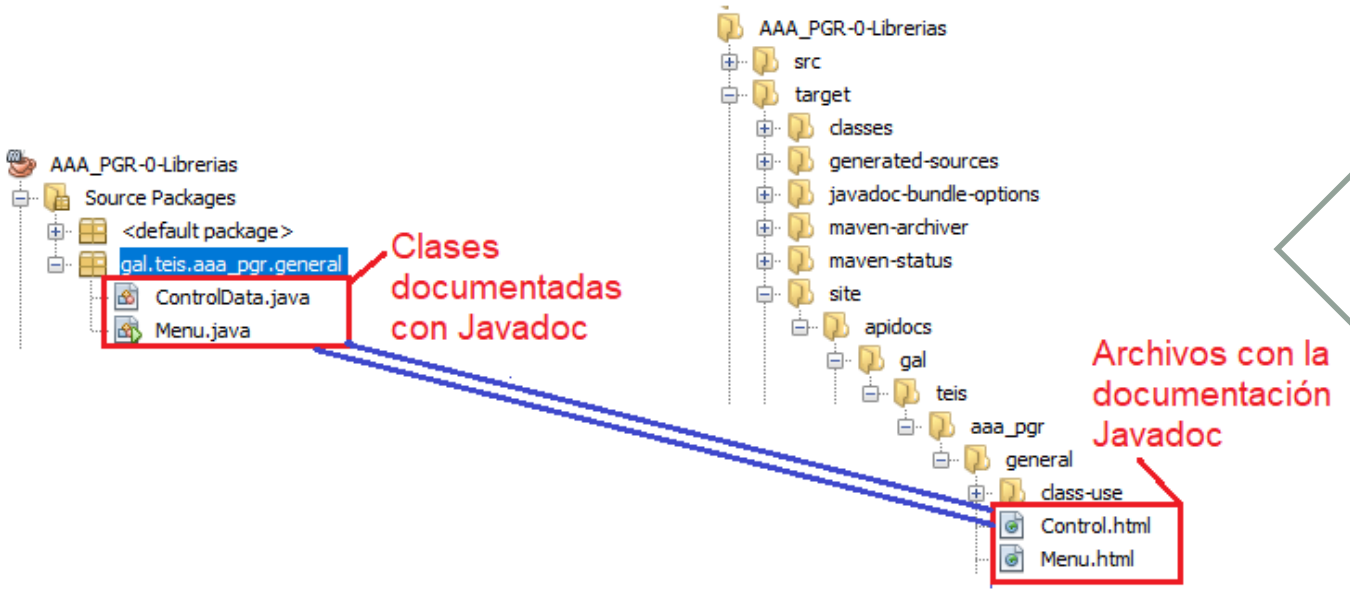


También se puede acceder a la documentación Javadoc de un proyecto desde un navegador.

Al generar la documentación se crean ficheros HTML en el proyecto que contienen la información de **Javadoc**.

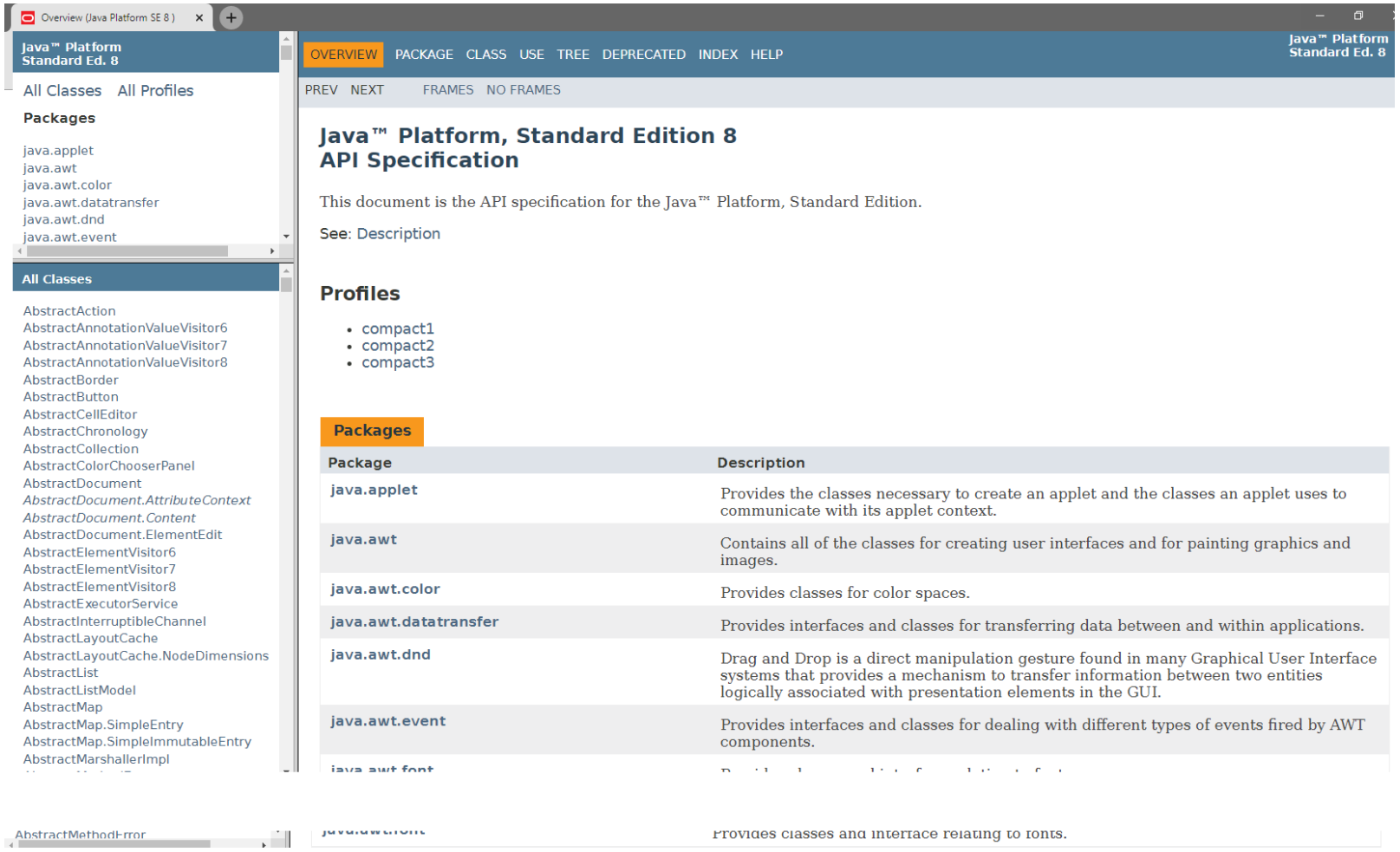
Los ficheros HTML se encuentran en la carpeta:

`nombreProyecto/target/site/apidocs/carpetas_paquete/class-use`



Los fichero *.HTML se pueden visualizar con un navegador

Acceso a la documentación oficial de Java

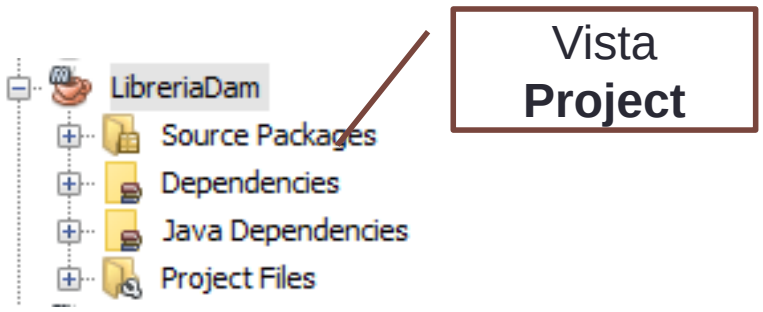


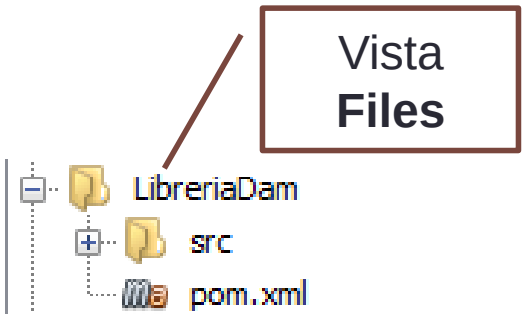
Ejercicio

- Crear un nuevo proyecto llamado "***GenerarJavadoc***"
- Descargar el fichero "Código fuente 1" y crear una nueva clase en el proyecto recién creado que incluya el código del fichero.
- Agregar valores de retorno a los métodos.
- Documentar los métodos, generar la documentación ***Javadoc*** y comprobar la visualización de la documentación desde ***main*** al utilizar los métodos documentados.

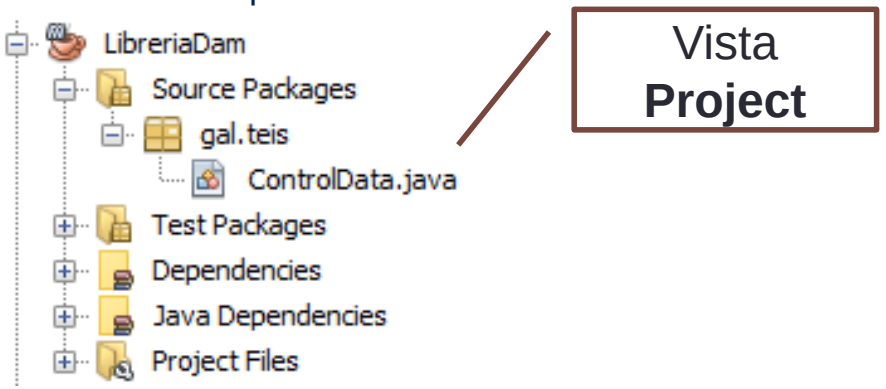
Crear librería JAR de un proyecto

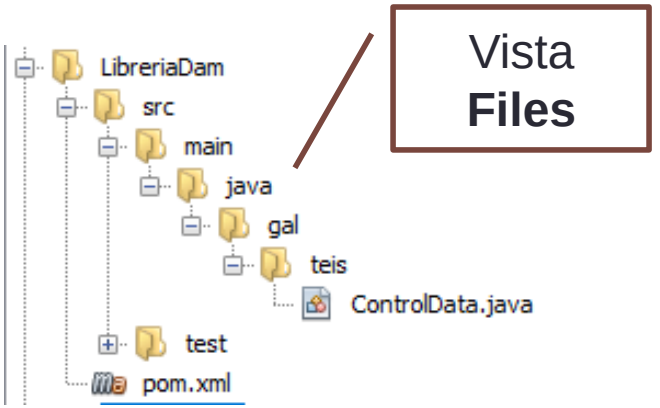
- Se puede crear una clase con métodos estáticos con el objetivo de utilizar sus métodos en diferentes proyectos.
- La clase de puede convertir en un fichero ***.jar** que podrá ser incorporado a distintos proyectos para utilizar, de la misma forma que se utiliza una clase de Java.
- **Para crear una librería JAR de un proyecto debemos seguir los siguientes pasos:**
  1. **Crear un proyecto que contendrá la/s clase/s que queremos utilizar como librería.**
     a) Creamos un proyecto denominado **LibreriaDam**.



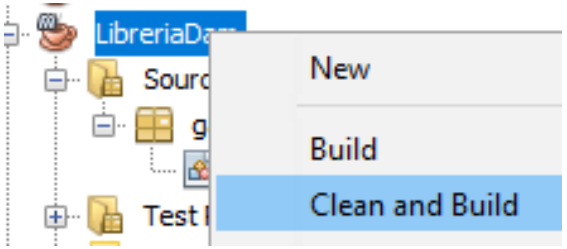


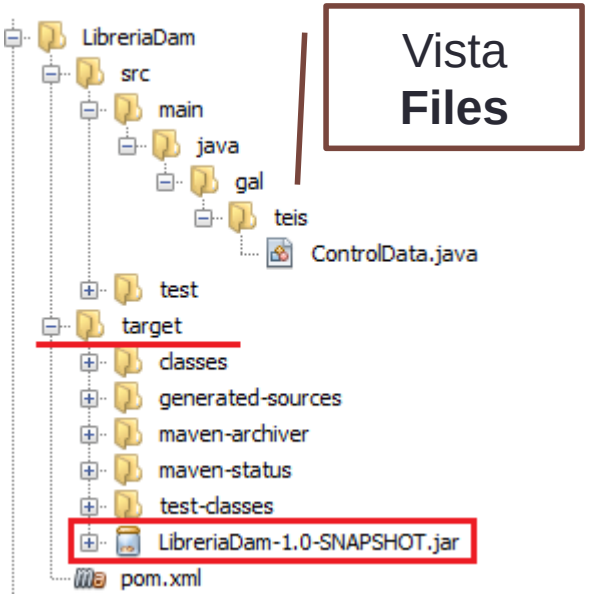
b) Creamos el paquete **gal.teis** y en su interior agregamos una clase **ControlData.java** que contendrá el contenido de la librería de métodos de control de tipo. El código de esta clase se suministrará en el aula. Debemos comprobar que no hay errores de compilación.





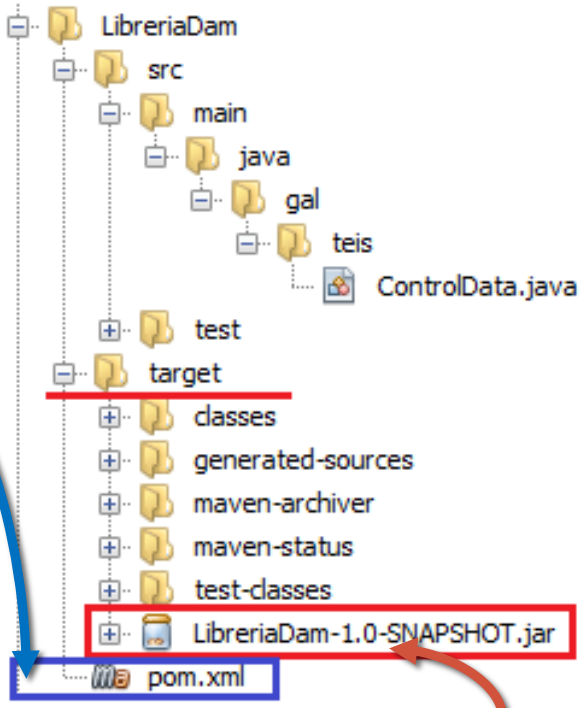
c) Construimos el proyecto mediante **Clean and Build** que tendrá como resultado la creación de la carpeta **LibreriaDam/target** que contiene el fichero **JAR** del proyecto.





```xml
<?xml version="1.0" encoding="UTF-8"?>
<!--Referencia al esquema MAVEN que interpreta las etiquetas de los fichero pom.xml-->
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <!--Versión del modelo de objetos que está usando este fichero pom-->
    <modelVersion>4.0.0</modelVersion>
    <!--Referencia del grupo que creo el proyecto, por defecto, es el nombre dominio especificado en los paquetes-->
    <groupId>gal.teis</groupId>
    <!--Parte del nombre del fichero JAR que se generará-->
    <artifactId>LibreriaDam</artifactId>
    <!--Versión del proyecto. SANPSHOT indica que está en desarrollo-->
    <version>1.0-SNAPSHOT</version>
    <!--Tipo de paquete que se va a producir-->
    <packaging>jar</packaging>
    <!--Propiedades relacionadas con el proyecto-->
    <properties>
        <!--Codificación. Para los idiomas latinos es UTF-8-->
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <!--Plataforma de Java desde la que se crea y ejecuta el proyecto-->
        <maven.compiler.source>15</maven.compiler.source>
        <maven.compiler.target>15</maven.compiler.target>
    </properties>
</project>
```

Fichero **pom.xml**



El fichero **JAR** tiene ese nombre debido a la información contenida en el fichero **pom.xml** que posee la información para crear proyectos y administrar las dependencias en **MAVEN**

2. Comprobar el repositorio local está él fichero JAR del proyecto que podré utilizar como una librería.

a) NetBeans crea un repositorio local de forma automática que podremos utilizar como un repositorio MAVEN para descargar las librerías a utilizar en un proyecto.
El repositorio se crea en la siguiente ruta:

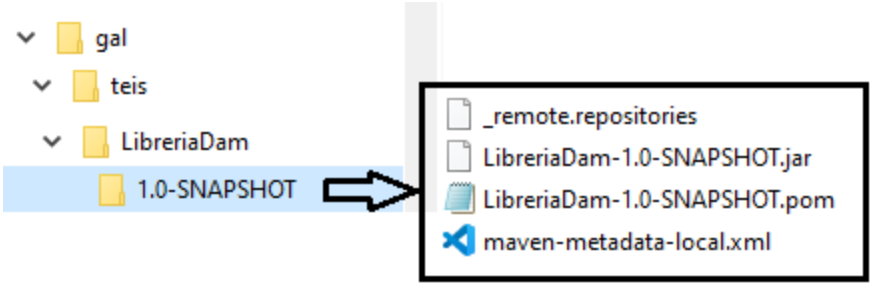
Nombre de paquete\nombre de proyecto

C:\Users\NombreUser\.m2\repository\gal\teis\LibreriaDam

carpeta del perfil del usuario

creado de forma automática por NetBeans

Donde se encuentra una carpeta que se denomina con el nombre de versión recogida en el fichero **pom.xml** y que contiene, entre otros, el fichero **JAR** del proyecto.



3. Agregar una dependencia al proyecto para poder usar una librería.

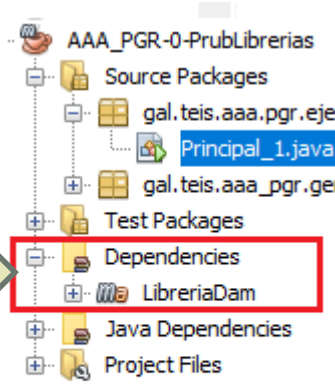
a) Editando el fichero pom.xml

i. Abrimos el fichero **pom.xml** del proyecto en el que queremos usar la librería.

ii. Incorporamos las siguientes etiquetas/valor al fichero *pom.xml*.

```
<dependencies>
    <dependency>
        <groupId>gal.teis</groupId>
        <artifactId>LibreriaDAM</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>
</dependencies>
```

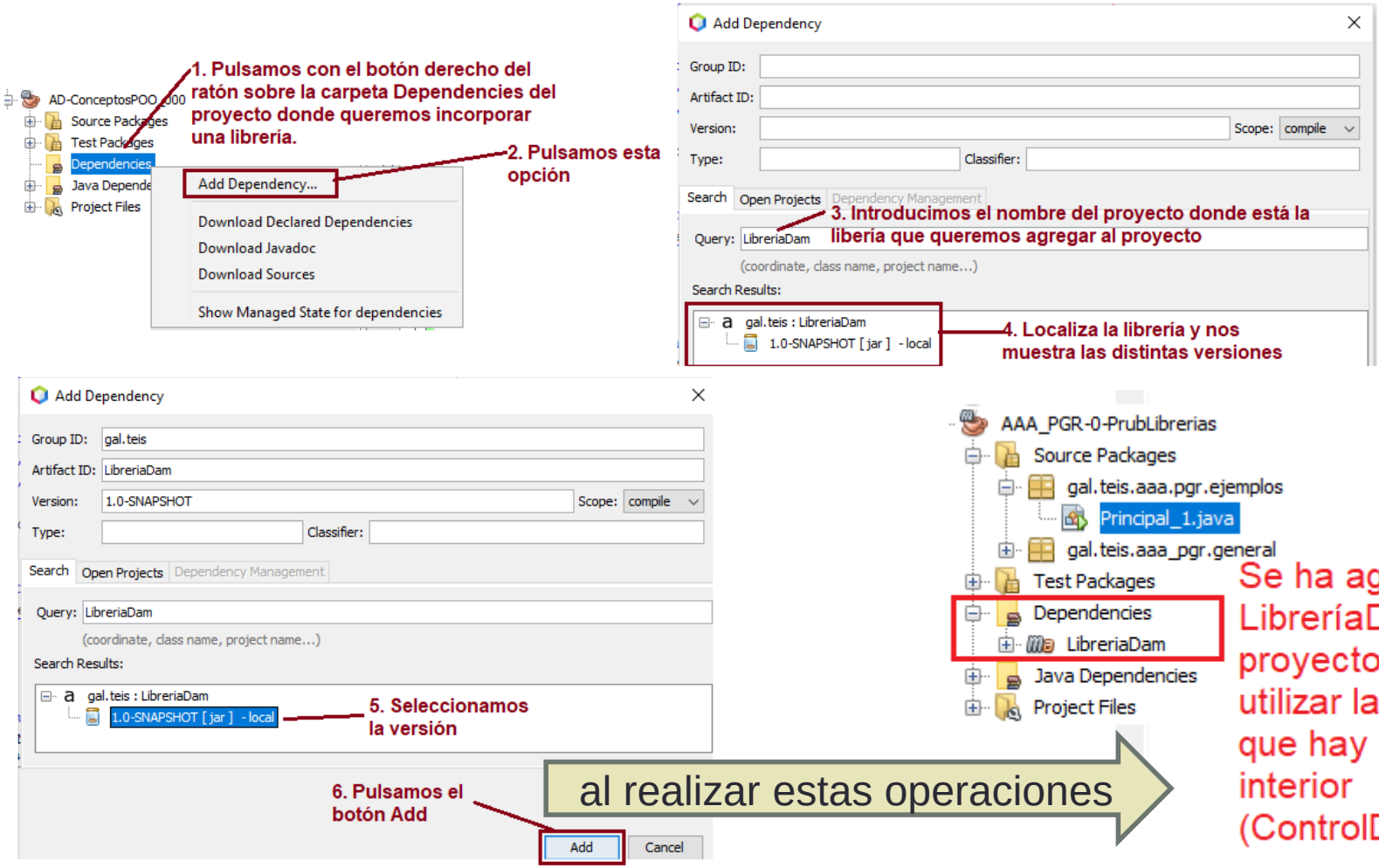
al modificar el **pom.xml**



Se ha agregado LibreríaDam al proyecto y puedo utilizar las clases que hay en su interior (ControlData)

b) Agregando una dependencia con la interfaz de NetBeans

i. Realizamos las operaciones que se detallan a continuación para agregar una librería a nuestro proyecto. Esto funciona para las librerías que están en el repositorio por defecto.



- En el caso de usar un repositorio que no sea el que corresponde a la ruta por defecto, por ejemplo, **C:\miRepositorio**, debemos realizar las siguientes operaciones:
  - Copiar en C:\miRepositorio el contenido que nos interese de la ruta por defecto de repositorios de NetBeans.
  - Modificar el fichero **pom.xml** con la información del nuevo repositorio.

```
<repositories>
    <repository>
        <id>mi-repositorio</id>
        <url>file://C:/miRepositorio</url>
    </repository>
</repositories>
```

- Al introducir la información del repositorio local en **pom.xml** podremos incorporar cualquier librería del mismo con los métodos estudiados.

4. Usar directamente la clase en el proyecto donde se ha referenciado el repositorio local.

```
import gal.teis.ControlData;
import java.util.Scanner;
/**
 *
 * @author Esther Ferreiro
 */
public class Principal {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);
        byte miByte;

        System.out.println("Introduce un valor de tipo Byte");

        miByte = ControlData.lerByte(sc);

        System.out.println("La variable leída es "+miByte);

    }
}
```

Importación de la librería **ControlData**

Uso de la librería **ControlData**