

UD04 – Optimización y Documentación

CONTENIDO

1. Refactorización
2. Control de versiones
3. Documentación



OBJETIVOS

- Conocer el objetivo de la refactorización de código fuente.
- Introducir los principales patrones de refactorización.
- Describir el uso de las herramientas de control de versiones.
- Conocer el proceso de documentación de código fuente



1. Refactorización



“

**La refactorización
consiste en realizar una
transformación al software
preservando su
comportamiento,
modificando su estructura
interna para mejorarlo**

William F. Opdyke, 1992





```
package refactoring;

import java.util.ArrayList;


import comun.Persona;

public class Sangrado {

    private Persona[] participantes;

    Boolean esParticipante (Persona x) {
        for (int i=0; i<participantes.length; i++) {
            if (participantes[i].getDNI().equals(x.getDNI())) {
                return true;
            }
        }
        return false;
    }

    ArrayList<Persona> getHijosFromParticipantes (Persona p) {
        ArrayList<Persona> hijos = new ArrayList<>();
        if (participantes.length>0) {
            for (Persona aux:participantes) {
                if (aux.getPadre() == p) {
                    hijos.add(aux);
                }
            }
        }
        return hijos;
    }
}
```



```
package refactoring;

import java.util.ArrayList;

import comun.Persona;

public class Sangrado {

    private Persona[] participantes;

    Boolean esParticipante (Persona x) {
        for (int i=0; i<participantes.length; i++) {
            if (participantes[i].getDNI() == x.getDNI()) {
                return true;
            }
        }
        return false;
    }

    ArrayList<Persona> getHijosFromParticipantes (Persona p) {
        ArrayList<Persona> hijos = new ArrayList<>();
        if (participantes.length>0) {
            for (Persona aux:participantes) {
                if (aux.getPadre() == p) {
                    hijos.add(aux);
                }
            }
        }
        return hijos;
    }
}
```

ATAJO DE TECLADO EN ECLIPSE → Ctrl + I.

Extraer método

- Fragmento de código que puede agruparse.
- Crear método con el fragmento.

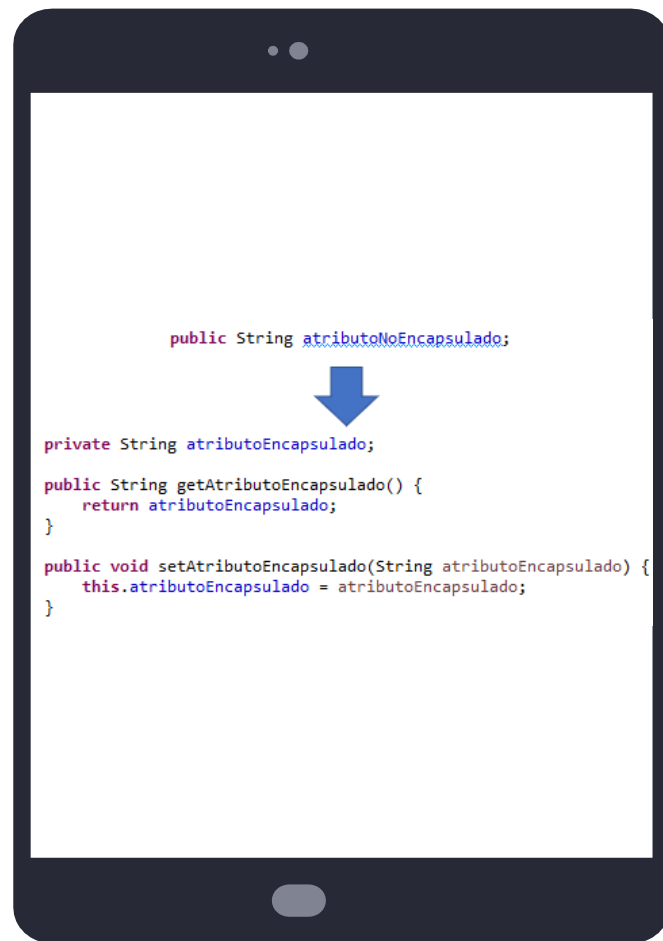
```
public void imprimirTodoNoRefactor() {  
    imprimirBanner();  
  
    //detalles de impresión  
    System.out.println("nombre: " + getNombre());  
    System.out.println("cantidad: " + getCargoPendiente());  
}
```



```
public void imprimirTodoRefactor() {  
    imprimirBanner();  
    imprimirDetalle(getNombre(), getCargoPendiente());  
}  
  
private void imprimirDetalle(String n, int cp) {  
    System.out.println("nombre: " + n);  
    System.out.println("cantidad: " + cp);  
}
```

Encapsular atributo

- Existe un atributo público.
- Convertir a privado y crear métodos de acceso.



Número mágico a constante

- Existe un literal u operación con un significado particular.
- Se crea una constante.

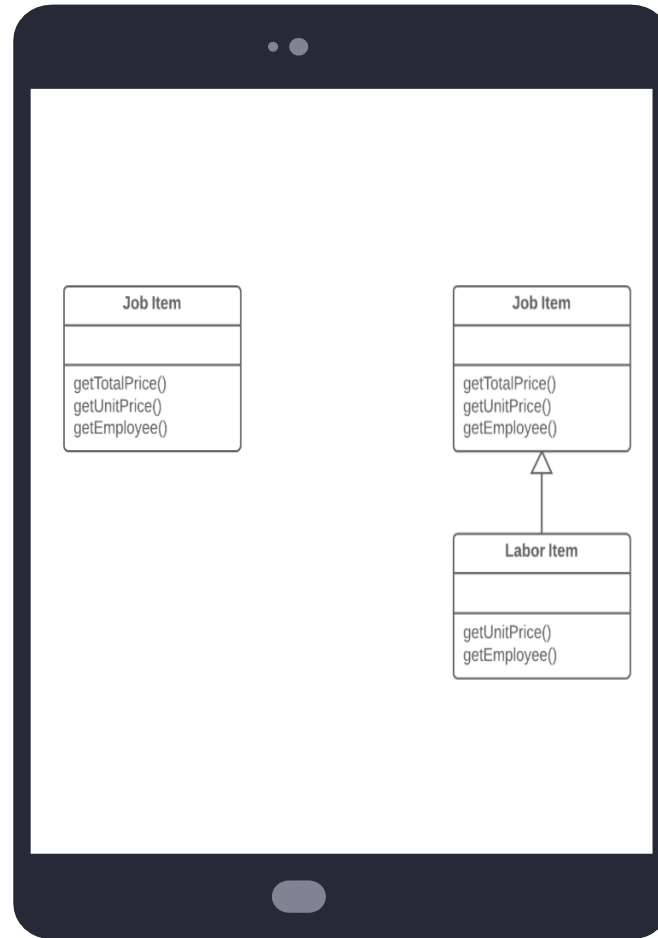
```
double energiaPotencial(double masa, double altura) {  
    return masa * altura * 9.81;  
}
```



```
private static final double _CONSTANTE_GRAVITACIONAL = 9.81;  
  
double energiaPotencialRefactor(double masa, double altura) {  
    return masa * altura * _CONSTANTE_GRAVITACIONAL;  
}
```

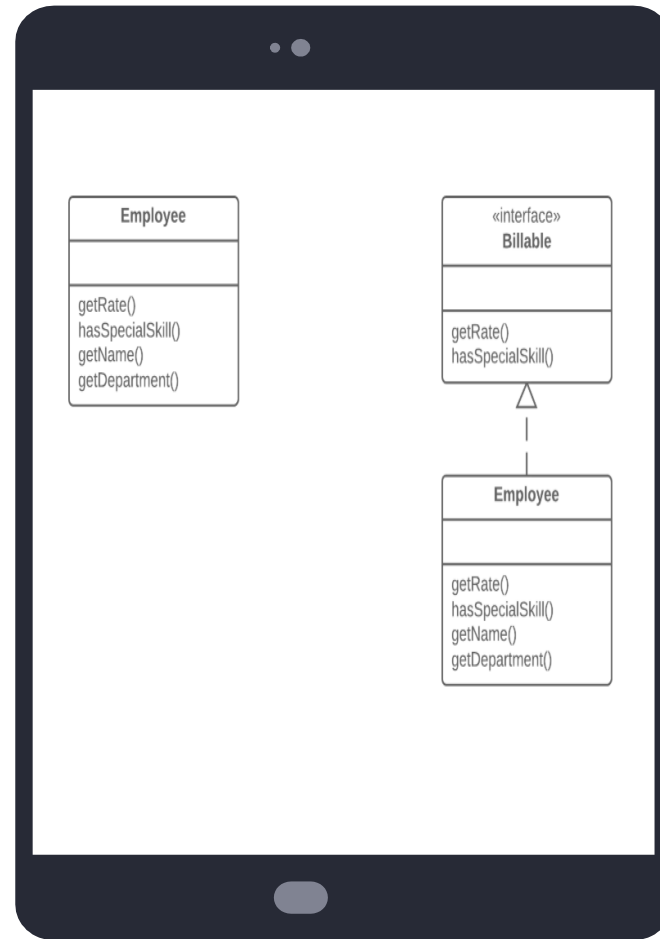
Extraer subclase

- Una clase tiene propiedades usadas por solo algunas instancias.
- Crear subclase con ese conjunto de propiedades



Extraer interfaz

- Múltiples usan métodos similares
- Crear interfaz con métodos comunes



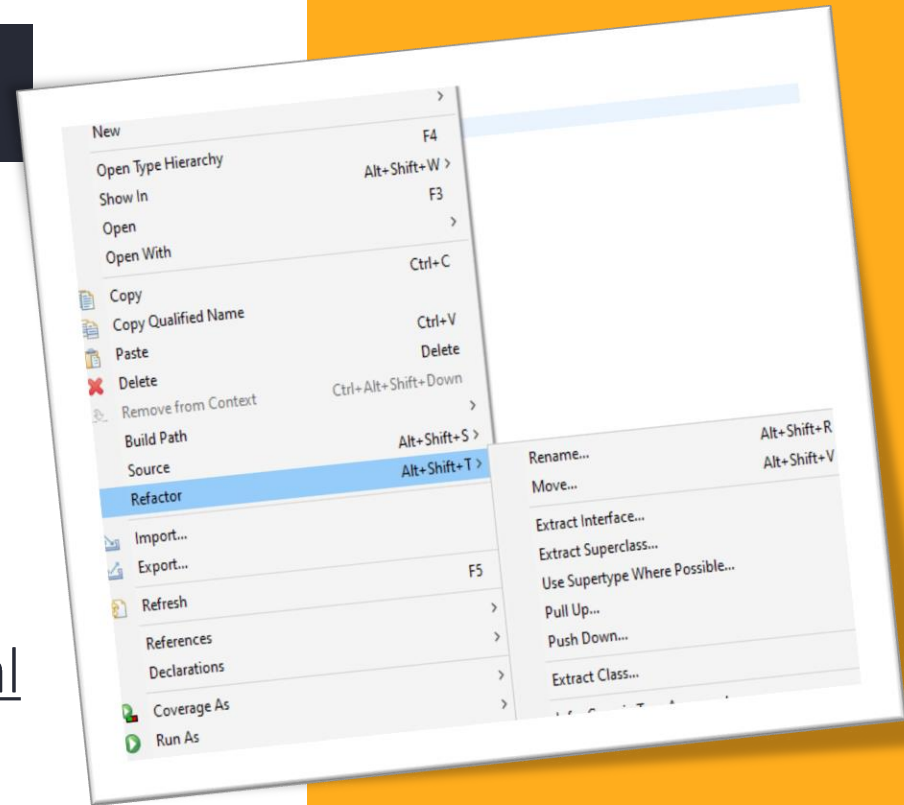
MALOS OLORES

Refactorización en Eclipse

Posee múltiples opciones de refactorización.

Se accede mediante menú contextual o Ctrl+Shift+T

Consultar [documentación oficial](#)





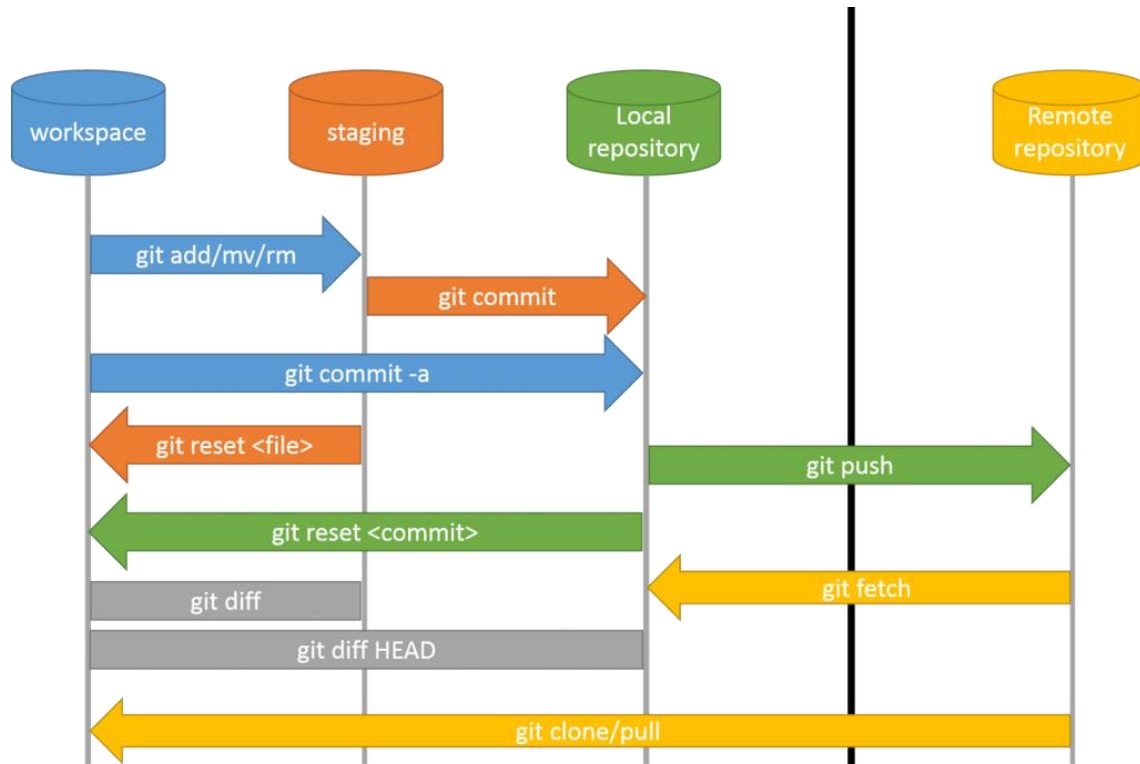
2. Control de versiones



Un **REPOSITORIO** es básicamente un servidor de archivos típico, con una gran diferencia: recuerdan todos los cambios que alguna vez se hayan escrito en ellos.

GIT

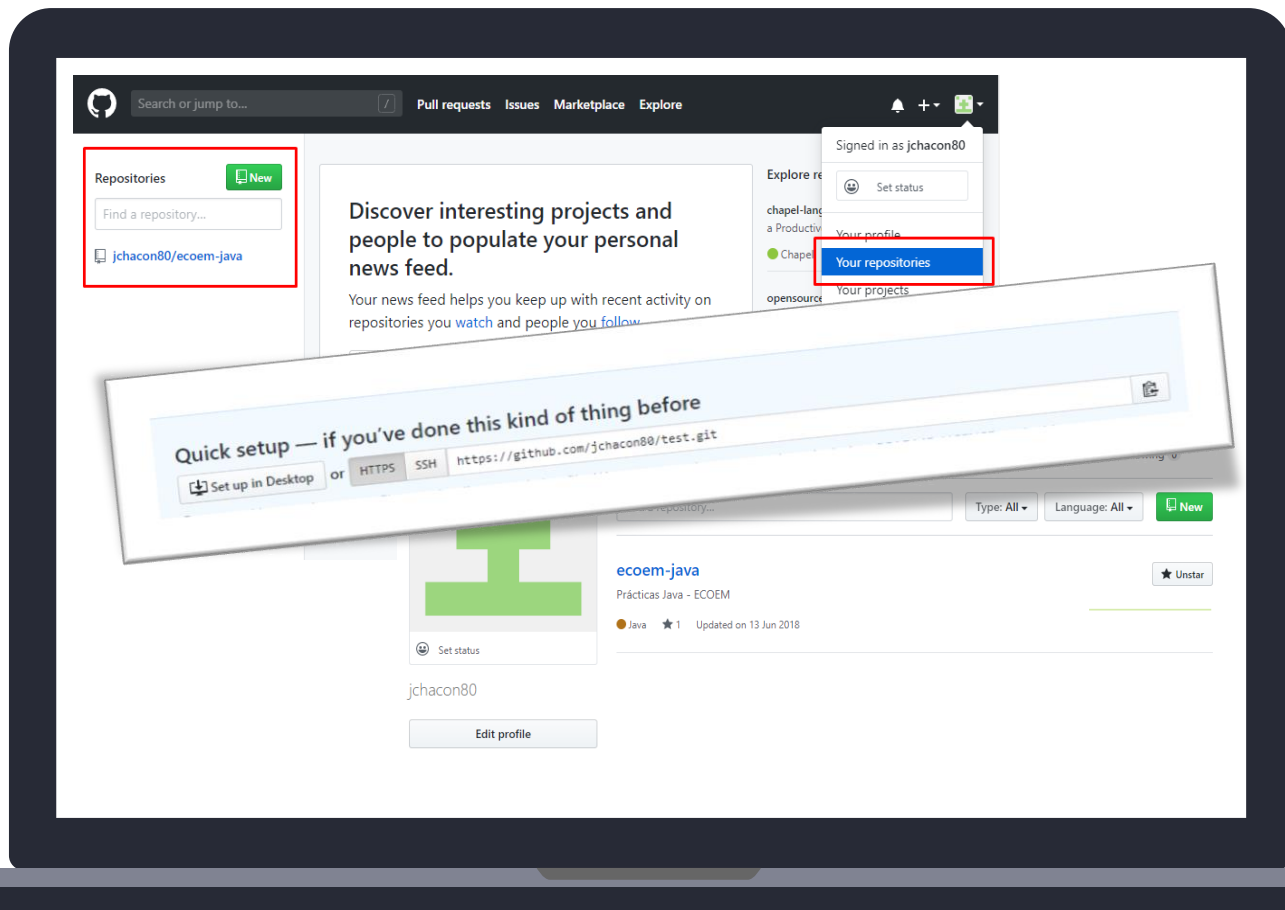
Creado en 2005 (Linus Torvalds). Licencia libre y distribuido.



GITHUB **(<https://github.com>)**

GitHub es una plataforma de desarrollo colaborativo donde se pueden crear repositorios de código para alojar proyectos utilizando el sistema de control de versiones GIT.





Herramientas de control de versiones

Permiten asociar nuestro código a un repositorio y trabajar actualizándolo y controlando las versiones de manera adecuada.



CLONAR UN REPOSITORIO

Select a URI

Source Git Repository

Enter the location of the source repository.

Location

URI: Local File...

Host:

Repository path:

Connection

Protocol:

Port:

Authentication

User:

Password:

☐ Store in Secure Store

Configure Push

Configure push for remote 'origin'

In order to use a remote for push, you must specify at least one URI and at least one ref mapping

URI:

Push URIs

Ref mappings

Configure Fetch

Configure fetch for remote 'origin'

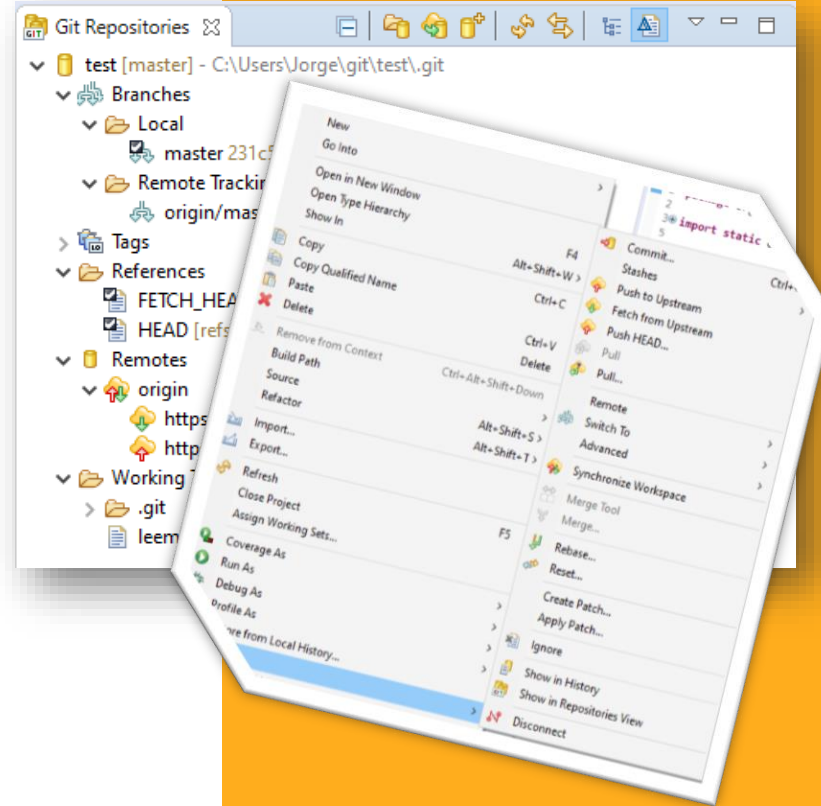
In order to use a remote for fetch, you must specify a URI and at least one ref mapping

URI:

Ref mappings

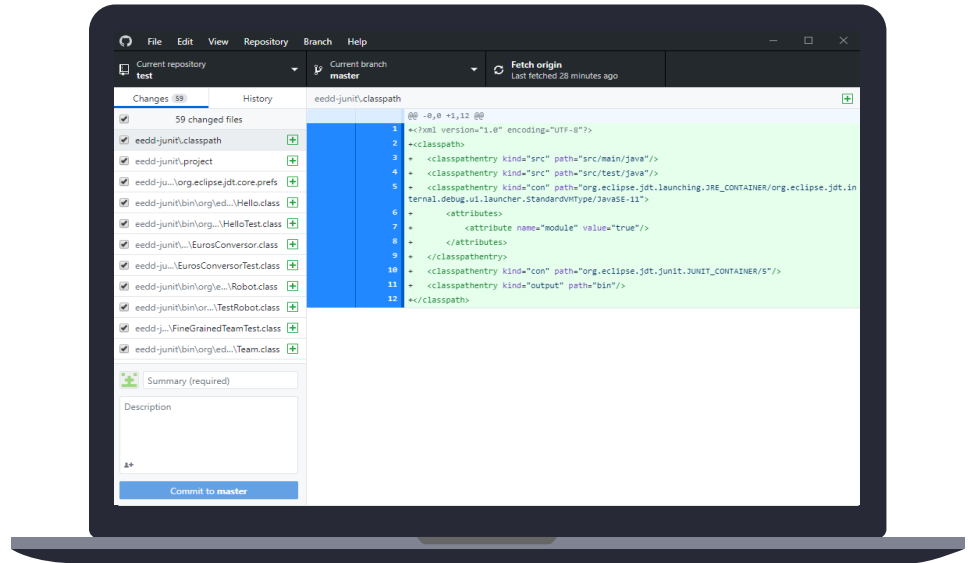
Operaciones de sincronización

- Habilitar vistas como Git Staging o History.
- Usar las acciones de la entrada Team de nuestro menú contextual.



Alternativa que permite trabajar de manera independiente a cualquier IDE.

Alternativa que permite trabajar de manera independiente a cualquier IDE.





3. Documentación



Los COMENTARIOS
clarifican el código fuente
y ayudan a mejorar el
entendimiento , algo
fundamental en el trabajo
en equipo.

JavaDoc

Tag	Descripción	Uso	Versión
@author	Nombre del desarrollador.	nombre_autor	1.0
@version	Versión del método o clase.	versión	1.0
@param	Definición de un parámetro de un método, es requerido para todos los parámetros del método.	nombre_parametro descripción	1.0
@return	Informa de lo que devuelve el método, no se puede usar en constructores o métodos "void".	descripción	1.0
@throws	Excepción lanzada por el método, posee un sinónimo de nombre @exception	nombre_clase descripción	1.2
@see	Asocia con otro método o clase.	referencia (#método()); clase#método(); paquete.clase#método(); paquete.clase#método()).	
@since	Especifica la versión del producto	indicativo numerico	
@serial	Describe el significado del campo y sus valores aceptables. Otras formas válidas son @serialField y @serialData	campo_descripcion	
@deprecated	Indica que el método o clase es antigua y que no se recomienda su uso porque posiblemente desaparecerá en versiones posteriores.	descripción	

```
/**
 * Suma un plus al salario del empleado si el empleado tiene mas de 40 años
 *
 * @param sueldoPlus valor del plus que se suma al salario
 * @return
 *
 *      <ul>
 *      <li>true: se suma el plus al sueldo</li>
 *      <li>false: no se suma el plus al sueldo</li>
 *      </ul>
 */
public boolean plus(double sueldoPlus) {
    boolean aumento = false;
    if (edad > 40 && compruebaNombre()) {
        salario += sueldoPlus;
        aumento = true;
    }
    return aumento;
}
```

Project Run Window Help

Open Project

Close Project

Build All

Build Project

Build Working Set

Clean...

Build Automatically

Generate Javadoc...

Properties

Generate Javadoc

Javadoc Generation

Javadoc generation may overwrite existing files

Javadoc command:

C:\Program Files\Java\jdk-11.0.5\bin\javadoc.exe

Configure...

Select types for which Javadoc will be generated:

- ☐ EDUD05
- ☒ eedd-javadoc
- ☐ eedd-junit
- ☐ HelloWorld
- ☐ SopaLetras

Create Javadoc for members with visibility:

☒ Private

☐ Package

☐ Protected

☐ Public

Private: Generate Javadoc for all classes and members.

☒ Use standard doclet

Destination: C:\Users\Jorge\eclipse-workspace\eedd-javadoc\doc

Browse...

☐ Use custom doclet

Doclet name:

Doclet class path:



< Back

Next >

Javadoc en Eclipse

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

ALL CLASSES

SEARCH: Search

Package main

Class Summary

Class	Description
Empleado	Clase Empleado Contiene informacion de cada empleado

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

ALL CLASSES