**CÓDIGO LIMPIO**

**Léete los siguientes capítulos del libro “Código limpio” de Robert C. Martin y responde a las siguientes preguntas**

**Capítulo 1: Código Limpio**

1. ¿Cuál es la importancia de escribir código limpio según el autor?

**Aumenta la productividad de mantener software.**

**Minimiza la posibilidad de que existan fallos**

**Entendible por lo que es más fácil de modificar.**

**Para el propio programador, el escribir código como un artesano.**

**Código modular y desacoplado, por tanto más fácil de probar.**

1. ¿Cómo se define el código limpio?

**Es aquel código que es fácil de entender, de modificar por parte del propio programador y del equipo, y ,mantener a lo largo del tiempo (añadir nuevas funcionalidades y corregir errores).**

**Capítulo 2: Nombres Significativos**

1. ¿Cuál es el impacto de elegir nombres significativos en la legibilidad del código?

**- Nombrar una variable u objeto en relación con la información que almacena.**

**- Hace que el código lo entiendan los miembros del equipo y el propio programador con el tiempo.**

1. ¿Cómo afecta la elección de nombres a la mantenibilidad del código?

**- Hace que el código lo entiendan los miembros del equipo y el propio programador con el tiempo.**

1. ¿Qué reglas se pueden inferir del capítulo?

**- Nombres que muestren la función de la variable / objeto.**

**- Comienzan por minúscula, el equipo tiene que llegar a acuerdos para que haya uniformidad en el código.**

**- Constantes todo con mayúscula.**

**- Clases primera en mayúscula.**

**Capítulo 3: Funciones**

1. ¿Por qué es crucial que las funciones hagan una sola cosa?

**- Descompone conceptos más amplios (reutilización de código).**

**- No tiene efectos secundarios (afectación sobre otras partes del código).**

**- Los métodos son setter (cambian un valor ) o getter (transformación de atributos).**

1. ¿Cuál es la relación entre las funciones pequeñas y la legibilidad del código?

**- Las funciones pequeñas son más fáciles de leer un vistazo**.

1. ¿Qué reglas se pueden inferir del capítulo?

**- Tamaño reducido (15 – 20 líneas de código y que no supere el ancho de la pantalla).**

**- Bloques de sangrado (tabular).**

**- Hacer una sola cosa (más sencilla de probar, más fácil de acotar el error).**

**- Como máximo 3 parámetros (hacer una clase POJO: estructura de datos y pasar un objeto por parámetro).**

**- Un parámetro se hace una pregunta o transformarlo.**

**- Nombre de la función (nombre descriptivo): verbo y sustantivo.**

**Capítulo 4: Comentarios**

1. Según el libro, ¿cuándo deberíamos escribir comentarios y cuándo no?

**- Hay que evitarlo, ya que hay que usar nombres descriptivos en funciones, variables, objetos.**

**- Solo cuando sea absolutamente necesario:**

**- Copyright.**

**- Expresiones regulares** (<https://medium.com/@jgratereaux/validar-correos-electr%C3%B3nicos-con-expresiones-regulares-7914751b6018>).

1. ¿Cuál es la diferencia entre un comentario y un nombre significativo?

**- El comentario sobra (normalmente) en caso de usar nombres significativos.**

1. ¿Qué reglas se pueden inferir del capítulo?

**- Tratar de evitar comentarios.**

**- Y código comentado.**

**- “El código debería hablar por si solo como si fuese un texto”.**

**SISTEMA DE CONTROL DE VERSIONES**

Un sistema de control de versiones es una **herramienta que se utiliza en el desarrollo de software y en la gestión de proyectos para realizar un seguimiento de los cambios en los archivos y el código fuente a lo largo del tiempo.**  Su función principal es **registrar, gestionar y facilitar la colaboración en el desarrollo de software o la gestión de archivos**, permitiendo a los equipos de trabajo mantener un historial de versiones, controlar quién realizó cambios y cuándo, y fusionar o revertir cambios de manera eficiente.

**BENEFICIOS:**  Historial de cambios: Permiten registrar todas las modificaciones realizadas en un proyecto a lo largo del tiempo. Esto facilita la revisión de versiones anteriores, la identificación de errores y la comprensión de cómo ha evolucionado el código. **Colaboración:** Facilitan la colaboración en equipo al permitir que varios desarrolladores trabajen en el mismo proyecto de manera simultánea. Los VCS gestionan los conflictos y fusionan los cambios de manera automática o manual, según sea necesario. Seguimiento de autores: Registran quién hizo cada cambio en el código o en los archivos, lo que ayuda a asignar responsabilidades y facilita la revisión de código.

**Ramificación y fusión:** Los VCS permiten crear ramas o bifurcaciones (branches) para trabajar en características (features) independientes sin afectar la rama principal (main). Luego, puedes fusionar (merge) estas ramas de nuevo en la rama principal una vez que estén listas.

**Reversión:** Puedes revertir los cambios a versiones anteriores en caso de que se detecten problemas o errores en el código actual.

**Respaldo y restauración:** Los VCS actúan como una forma de respaldo constante del código y los archivos del proyecto, lo que reduce el riesgo de pérdida de datos importantes.

**CASO 1**

2. Crear nuevo repositorio en Github

3. Eclipse: Crear proyecto

4. Github: crear token de acceso personal:

https://docs.github.com/es/authentication/keeping-your-account-and-datasecure/

managing-your-personal-access-tokens.

https://github.com/settings/tokens

Personal Access Token

5. Click derecho en el proyecto -> team -> create new git repository. Configure Git

Repository, seleccionar “use or create repository in parent folder of project”

6. Team -> commit

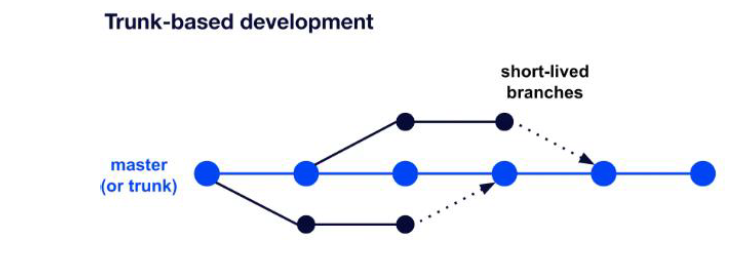
7. Team -> push to Brand in remote

8. Comprobar que se ha subido a git hub

9. Hacer cambio en un archivo .java de github

10. Team->pull

**CASO 2** Trabajo con ramas usando un sistema de control de versiones (modificación

estrategia de ramas git trunk).

- Sobre el caso de uso anterior, crear una nueva rama (Team-> Switch->New Brand):

algunos ejemplos de nombre son: tu identificación o nombre,

funcionalidad\_en\_desarrollo, bug, etc.

- Seleccionar check out new Branch, para que se conmute a la nueva rama.

- Añadir un cambio en un archivo.

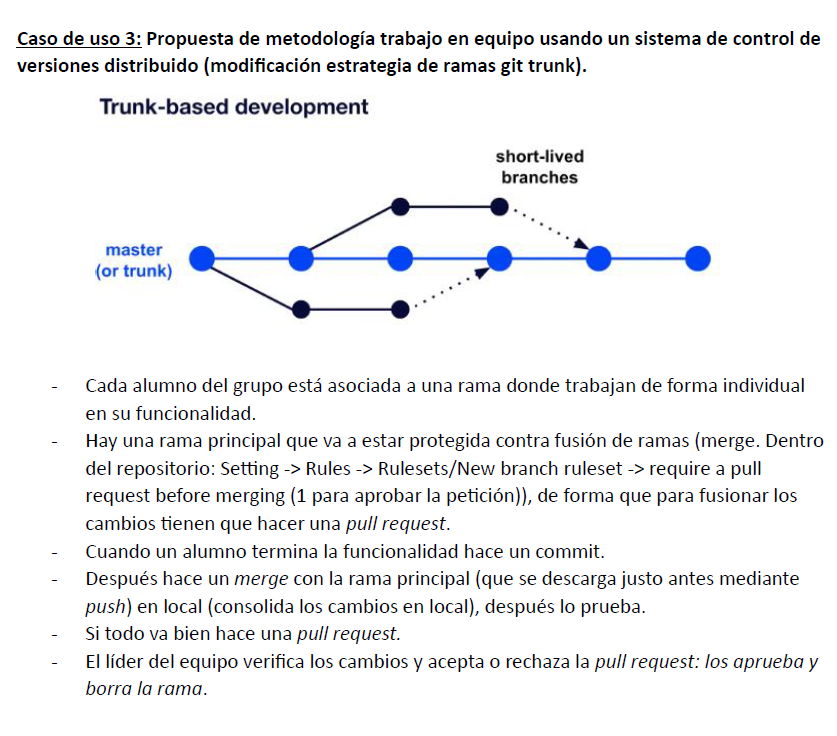
- Hacer un merge (o rebase, para eliminar el histórico de los cambios en la rama) para

combinar la rama secundaria en la principal (seleccionamos la principal y hacemos un

merge desde la rama secundaria): con todas las opciones por defecto.

- Team->Switch To-> rama master.

- Probar más cambios y opciones. Entre ellas el pull y push.

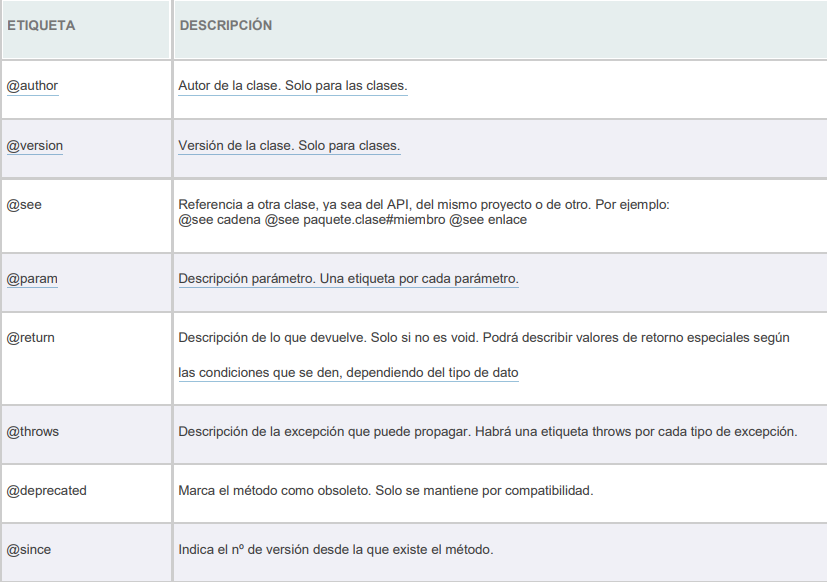


# **SMELL CODE Y REFACTORIZACIÓN**

1. Aplicar patrones de diseño(soluciones estándar a problemas recurrentes)
2. Dividir en funciones o clases
3. Cambiar nombres significativos(variables,métodos,clases).
4. Buscar duplicados y extraerlos en métodos o clases.
5. Reorganizar código (borrar código,etc)
6. Seguir principios de programación estructurada (usar 3 tipos de sentenecias: condicionales, bucles e instrucciones secuenciales) y programación orientada a objetos(encapsulación, abstracción, herencia, polimorfismo,etc) . Hay que hacerlo (no es una opción).
7. Simplificar el código: con estructuras de control (intentar evitar condiciones y bucles anidados).

# **JAVADOC**

Documentación en Java (Javadoc) El lenguaje java soporta 3 tipos de comentarios: - /\* texto \*/ : el compilador ignora desde /\* hasta \*/ - //texto : el compilador ignora todo desde // hasta el fin de línea - /\*\* documentación \*/: estos son los comentarios para generar documentación, a partir de los cuales se generará automáticamente la documentación. Javadoc es una herramienta incluida en el JDK (Java Development Kit) para generar documentación en formato HTML. Para generar APIs con Javadoc han de usarse etiquetas (tags) de HTML o ciertas palabras reservadas precedidas por el carácter "@" (anotaciones).



**Ejemplo:**

**/\*\* \* Clase Empleado**

**\***

**\* Contiene información de cada empleado**

**\***

**\* @author Mi nombre**

**\* @version 1.0**

**\*/**

**public class Empleado {**

**//Atributos**

**/\*\***

**\* Nombre del empleado**

**\*/**

**private String nombre;**

**/\*\***

**\* Apellido del empleado**

**\*/**

**private String apellido;**

**/\*\***

**\* Edad del empleado**

**\*/**

**private int edad;**

**/\*\***

**\* Salario del empleado**

**\*/**

**private double salario;**

**//Metodos publicos**

**/\*\***

**\* Suma un plus al salario del empleado si el empleado tiene mas de 40 años**

**\* @param sueldoPlus**

**\* @return <ul>**

**\* <li> true: se suma el plus al sueldo**

**\* <li> false: no se suma el plus al sueldo**

**\* </ul>**

**\*/**

**public boolean plus (double sueldoPlus)**

**{ boolean aumento=false;**

**if (edad>40 && compruebaNombre()){**

**salario+=sueldoPlus; aumento=true;**

**} return aumento;**

**}**

**//Metodos privados**

**/\*\***

**\* Comprueba que el nombre no este vacio**

**\* @return <ul>**

**\* <li> true: el nombre es una cadena vacia </li>**

**\* <li> false: el nombre no es una cadena vacia </li>**

**\* </ul>**

**\*/**

**private boolean compruebaNombre(){**

**if(nombre.equals("")){**

**return false;**

**}**

**return true;**

**}**

**//Constructores**

**/\*\***

**\* Constructor por defecto**

**\*/**

**public Empleado(){ this ("", "", 0, 0); }**

**/\*\***

**\* Constructor con 4 parametros**

**\* @param nombre nombre del empleado**

**\* @param apellido nombre del empleado**

**\* @param edad edad del empleado**

**\* @param salario salario del empleado**

**\*/**

**public Empleado(String nombre, String apellido, int edad, double salario){ this.nombre=nombre;**

**this.apellido=apellido;**

**this.edad=edad;**

**this.salario=salario; } }**

Referencias: 1.- Ayuda de Oracle para cómo escribir documentos para la herramienta Javadoc: https://www.oracle.com/technetwork/java/javase/documentation/index-137868.html#tag 2.- Cómo ejecutar la generación automática de Javadoc desde Eclipse:

1. Project > Generate Javadoc... .
2. En el campo Javadoc command: , busca el archivo javadoc.exe (normalmente está en [path\_to\_jdk\_directory]\bin\javadoc.exe) (por ejemplo: C:\Program Files\Java\jdk1.8.0\_144\bin).
3. En el campo Destination: , selecciona el destino deseado para la documentación.
4. Da click en Finish .