

CLEAN CODE

Vamos a ver técnicas erróneas muy comunes a la hora de programar y cómo solucionarlas de manera sencilla para mejorar nuestros desarrollos, para ello, vamos a utilizar una técnica llamada "Clean Code".

Clean Code es una filosofía que se centra en la escritura de código software fácil de leer, entender y mantener. La diferencia entre un desarrollo mediocre y uno bien ejecutado y estructurado es mucho más importante de lo que parece, aunque los dos desarrollos cumplan la misma función, no lo hacen de la misma manera y a la hora de cambiar el código o trabajar con él, notamos esas diferencias y nos dificulta el trabajo.

A continuación van a aparecer algunas capturas en las que veremos errores en nuestro código y cómo solucionarlos con muy poco esfuerzo.

```
public static void main(String[] args) {
    int i = 3, j = 5, k = 4;
    double d = 1.2;

    System.out.println( "Linea 01 : " + i++ + "-" + ++j + "-" + k);
    System.out.printf( "Linea 02 : %02d - %02d - %02d\n", i++, --j, k+k);
    System.out.println( "Linea 03 : " + (i++ > --j) + "-" + (k+=k) );
    System.out.printf( "Linea 04 : %02d / %02d / %04d / %02d\n", i, (~i+1), -i, ~(~i+1) );
    System.out.println( "Linea 05 : " + ( (true || (false && true)) || true) && false );
    System.out.printf( "Linea 06 : %.2f / %.2f / %.4f / %.2f\n", d, d*=1.1, -d, d*2+1 );
}
```

Empezamos con el error más común y probablemente el más importante.

El nombre de las variables: Como vemos en la imagen, las cuatro variables declaradas no tienen ningún significado, son letras singulares que no aportan nada. Si fuéramos los programadores que estaban realizando este desarrollo, pensaríamos "No pasa nada, yo sé lo que significa", pero lo más normal es que cuando trabajemos en un proyecto, no lo vamos a hacer solos.

Es muy común que se unan varios grupos en paralelo con los que casi no vas a tener contacto durante el proceso, por tanto, es crucial que algo tan vital como las variables tenga un nombre coherente y que aporte información útil a nuestro código.

```
public static void main(String[] args) {
    int salario = 3, bonus = 5, gasto = 4;
    double impuesto = 1.2;

    System.out.println( "Linea 01 : " + salario++ + "-" + ++bonus + "-" + gasto);
    System.out.printf( "Linea 02 : %02d - %02d - %02d\n", salario++, --bonus, gasto+gasto);
    System.out.println( "Linea 03 : " + (salario++ > --bonus) + "-" + (gasto+=gasto) );
    System.out.printf( "Linea 04 : %02d / %02d / %04d / %02d\n", salario, (~salario+1), -salario, ~(~salario+1) );
    System.out.println( "Linea 05 : " + ( (true || (false && true)) || true) && false );
    System.out.printf( "Linea 06 : %.2f / %.2f / %.4f / %.2f\n", impuesto, impuesto*=1.1, -impuesto, impuesto*2+1 );
}
```

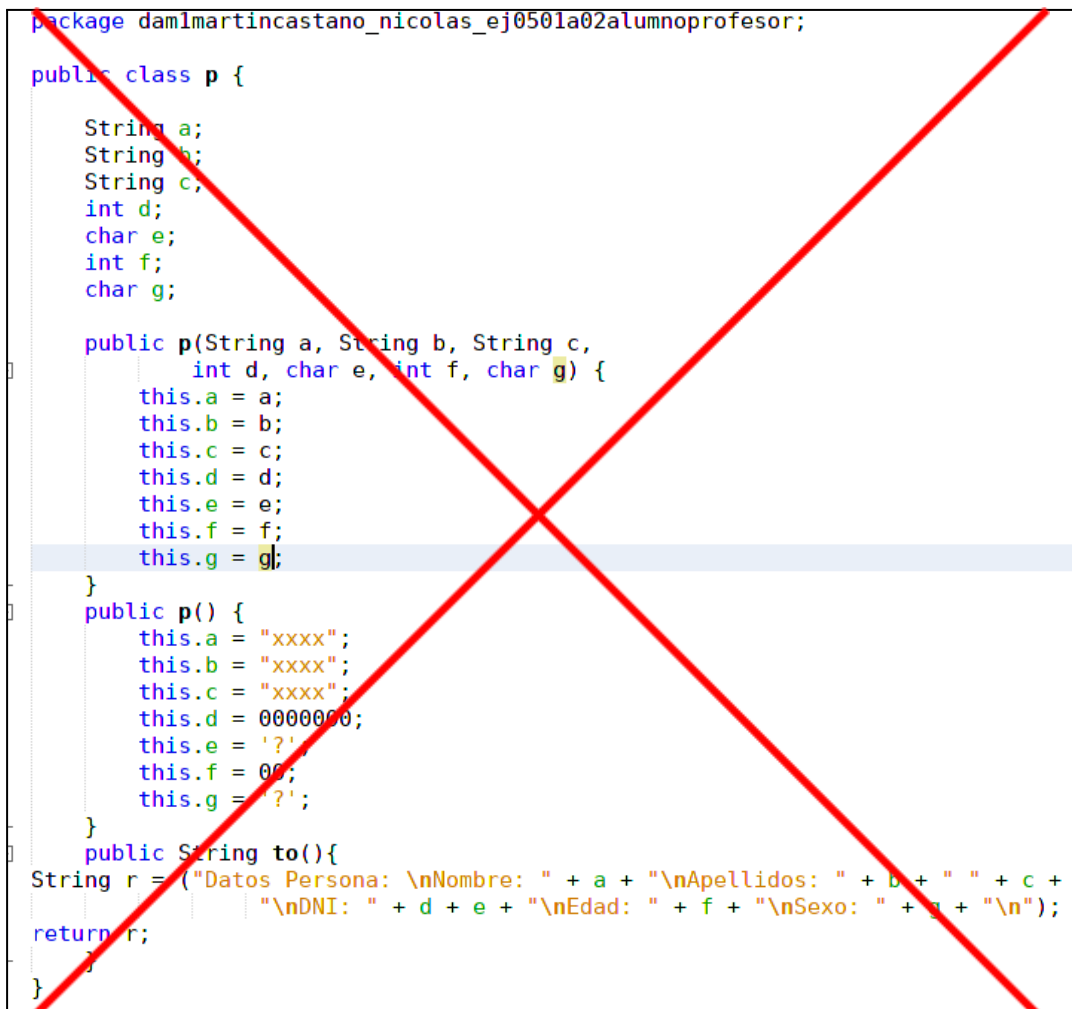
Con unos minutos vemos que el significado del código ha cambiado drásticamente y va a ser mucho más fácil saber al menos de lo que estamos hablando al hacer el desarrollo.

Aparte de utilizar nombres con sentido y significado, es importante sean fáciles de pronunciar ya que cuando trabajamos con proyectos de gran tamaño y personal es muy común que las distintas partes del proyecto estén localizadas en diferentes sitios ciudades e incluso países, con lo cual aparte de utilizar el inglés siempre como lenguaje universal tenemos que asegurarnos de poder pronunciarlos fácilmente.

Clases y Métodos: Si utilizamos clases o métodos, estas reglas también se aplican a ellos, y aparte debemos seguir las guías de mayúscula y minúscula dependiendo de lo que estemos declarando:

Clases y Constantes = mayúscula.

Funciones, Atributos y Variables locales = minúscula.



```
package dam1martincaetano_nicolas_ej0501a02alumnoprofesor;

public class p {

    String a;
    String b;
    String c;
    int d;
    char e;
    int f;
    char g;

    public p(String a, String b, String c,
            int d, char e, int f, char g) {
        this.a = a;
        this.b = b;
        this.c = c;
        this.d = d;
        this.e = e;
        this.f = f;
        this.g = g;
    }

    public p() {
        this.a = "xxxx";
        this.b = "xxxx";
        this.c = "xxxx";
        this.d = 0000000;
        this.e = '?';
        this.f = 00;
        this.g = '?';
    }

    public String to(){
        String r = ("Datos Persona: \nNombre: " + a + "\nApellidos: " + b + " " + c +
                "\nDNI: " + d + e + "\nEdad: " + f + "\nSexo: " + g + "\n");
        return r;
    }
}
```

```

package damlmartincastano_nicolas_ej0501a02alumnoprofesor;

public class Persona {

    String nombre;
    String apellido1;
    String apellido2;
    int nroDNI;
    char letraDNI;
    int edad;
    char sexo;

    public Persona(String nombre, String apellido1, String apellido2,
        int nroDNI, char letraDNI, int edad, char sexo) {
        this.nombre = nombre;
        this.apellido1 = apellido1;
        this.apellido2 = apellido2;
        this.nroDNI = nroDNI;
        this.letraDNI = letraDNI;
        this.edad = edad;
        this.sexo = sexo;
    }

    public Persona() {
        this.nombre = "xxxx";
        this.apellido1 = "xxxx";
        this.apellido2 = "xxxx";
        this.nroDNI = 0000000;
        this.letraDNI = '?';
        this.edad = 00;
        this.sexo = '?';
    }

    public String to(){
String r = ("Datos Persona: \nNombre: " + nombre + "\nApellidos: " + apellido1 + " " + apellido2 +
        "\nDNI: " + nroDNI + letraDNI + "\nEdad: " + edad + "\nSexo: " + sexo + "\n");
return r;
    }
}

```

Funciones: Existen varias mejoras que no están centradas en los nombres, por ejemplo, las funciones (métodos) solo deben realizar una acción, no es recomendable que creemos una sola función que nos haga todo porque eso puede hacer que nos confundamos y no consigamos lo que queremos.

También podemos añadir que las funciones deberán ser lo más cortas posibles (hablando de líneas en código). Tenemos que asegurarnos que nuestras funciones no colisionan entre ellas y que no hay efectos secundarios “side effects”.

Los programadores tienden a repetirse mucho y esa es una de las características que debemos evitar para nuestro desarrollo, “Don't repeat yourself”, es importante saber que no estamos escribiendo lo mismo con otras palabras cuatro veces, porque así solo perdemos nuestro tiempo y el de los demás.

Comentarios: Por último, mencionando los comentarios. Estos tienen que ser casi inexistentes, los comentarios no se suelen actualizar así que es muy peligroso fiarse

de ellos, para evitar problemas de ese tipo, con un código autoexplicativo minimizamos la necesidad de utilizar comentarios para aclarar todo tipo de detalles que llenan el desarrollo de “mierda” por así decirlo y lo hacen más confuso.

Como vemos, Clean Code es a prueba de vagos y nos ayuda inmensamente para no estar comiéndonos la cabeza para descifrar qué quería decir el compañero cuando puso en el código: **final int “*f relja ñlkjh*” = 10000;**