Clean Code

BLOQUES 1, 2 Y 3

VICENTE MENA MICAELO

1º DAM | IES Alonso de Avellaneda

Enlace al repositorio de GitHub: https://github.com/VicenteM23/CleanCode1Ev_Vicente/

Vicente Mena Micaelo

Índice

Contenido

Vicente Mena Micaelo

Ejemplo 1

En el primer ejemplo que encontramos de código sucio, vemos que este no es autoexplicativo y además de ello no disponemos de ningún comentario que nos ayude a la hora de saber qué es lo que está realizando el código.

```
    public static void main(String[] args) {
        Scanner scan = new Scanner(source: System.in);
            String i;
            System.out.print(s: "Inserta el número de aciertos en 20 preguntas :" );
            i = scan.nextLine();
            double j = Double.valueOf(s: i);
            double f = 20-j;
            System.out.println("Número de errores: " + f);
            double h = f / 2;
            System.out.println(x: h);
            double k = (f - h)/2;
            if(k < 5)
                System.out.println("Calificación: " + k + " Insuficiente");
            else if(k >= 5 && k <6)
                System.out.println("Calificación: " + k + " Suficiente");
            else if(k >= 6 && k <7)
                System.out.println("Calificación: " + k + " Bien");
            else if(k >= 7 && k <9)
                System.out.println("Calificación: " + k + " Notable");
            else if(k >= 9)
                System.out.println("Calificación: " + k + " Sobresaliente");
    }

}
```

Para poder mejorar y limpiar este código podemos realizar lo siguiente:

1. Modificar los nombres de las variables por nombres con significado, fáciles de pronunciar y que se puedan buscar:

```
String numeroAciertos;
double numeroErrores = 20-numAciertosDouble;
double notaResta = numeroErrores / 2;
double notaResta = numeroErrores / 2;
double notaFinal = (numAciertosDouble - notaResta)/2;
```

2. Añadimos ciertos comentarios considerados necesarios para indicar que realiza el código, a pesar de que el código ya es autoexplicativo:

```
double numeroErrores = 20-numAciertosDouble; //Operación que calcula el número de errores
System.out.println("Número de errores: " + numeroErrores);

double notaResta = numeroErrores / 2; //Operación que calcula la nota que restará a la final
System.out.println(x: notaResta);

double notaFinal = (numAciertosDouble - notaResta)/2; //Operación para calcular la nota final
```

Vicente Mena Micaelo

Ejemplo 2

En el segundo ejemplo que encontramos de código sucio, vemos que este tampoco es autoexplicativo y además de ello no disponemos de ningún comentario que nos ayude a la hora de saber qué es lo que está realizando el código.

```
package cleancodeproject;
class c {
    double x;
    double y;
}
/**
 *
 * @author Vicente Mena
 */
public class EjemploCodigoSucio2 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        c P = new c();
        c Q = new c();
        c R = new c();
        c O = new c();
        P.x=1.1;
        P.y=2.2;
        Q.x=4.7;
        Q.y=7.4;
        R.x=0.5;
        R.y=3.5;
        O.x=0;
        O.y=0;
        double h = Math.sqrt((Math.pow(a: P.x, b: 2))+(Math.pow(a: P.y, b: 2)));
        double i = Math.sqrt((Math.pow(a: Q.x, b: 2))+(Math.pow(a: Q.y, b: 2)));
        double j = Math.sqrt((Math.pow(a: R.x, b: 2))+(Math.pow(a: R.y, b: 2)));
        System.out.printf(format: "P (%1.1f:%1.1f) - O (%1.1f:%1.1f) = %1.3f\n", args: P.x , args: P.y , args: O.x , args: O.y , args: h);
        System.out.printf(format: "Q (%1.1f:%1.1f) - O (%1.1f:%1.1f) = %1.3f\n", args: Q.x , args: Q.y , args: O.x , args: O.y , args: i);
        System.out.printf(format: "R (%1.1f:%1.1f) - O (%1.1f:%1.1f) = %1.3f\n", args: R.x , args: R.y , args: O.x , args: O.y , args: j);
        double PQ = Math.sqrt((Math.pow((P.x-Q.x),b: 2 ) + Math.pow((P.y-Q.y),b: 2 )));
        System.out.printf(format: "P: (%1.1f:%1.1f) - Q: (%1.1f:%1.1f) = %02.5f\n", args: P.x , args: P.y ,args: Q.x , args: Q.y , args: PQ);
        double PR = Math.sqrt((Math.pow((P.x-R.x),b: 2 ) + Math.pow((P.y-R.y),b: 2 )));
        System.out.printf(format: "P: (%1.1f:%1.1f) - R: (%1.1f:%1.1f) = %02.5f\n", args: P.x , args: P.y ,args: R.x , args: R.y , args: PR);
        double RQ = Math.sqrt((Math.pow((R.x-Q.x),b: 2 ) + Math.pow((R.y-Q.y),b: 2 )));
        System.out.printf(format: "P: (%1.1f:%1.1f) - Q: (%1.1f:%1.1f) = %02.5f\n", args: R.x , args: R.y ,args: Q.x , args: Q.y , args: RQ);

    }
```

Para poder mejorar y limpiar este código podemos realizar lo siguiente:

1. Modificar el nombre de la clase, objetos y variables por nombres con sentido, fáciles de pronunciar y buscar:

```
class Coordenadas {
    double x;
    double y;
}
Coordenadas PuntoP = new Coordenadas();
Coordenadas PuntoQ = new Coordenadas();
Coordenadas PuntoR = new Coordenadas();
Coordenadas PuntoO = new Coordenadas();
```

```
PuntoP.x=1.1;
PuntoP.y=2.2;

PuntoQ.x=4.7;
PuntoQ.y=7.4;

PuntoR.x=0.5;
PuntoR.y=3.5;

PuntoO.x=0;
PuntoO.y=0;
double distanciaPOrigen
double distanciaQOrigen
double distanciaROrigen
```

2. Añadir ciertos comentarios aclarativos para indicar que realiza el código, a pesar de que el código es autoexplicativo:

```
//Operaciones para calcular la distancia entre puntos y origen:
double distanciaPOrigen = Math.sqrt((Math.pow(a: PuntoP.x, b: 2))+(Math.pow(a: PuntoP.y, b: 2)));
double distanciaQOrigen = Math.sqrt((Math.pow(a: PuntoQ.x, b: 2))+(Math.pow(a: PuntoQ.y, b: 2)));
double distanciaROrigen = Math.sqrt((Math.pow(a: PuntoR.x, b: 2))+(Math.pow(a: PuntoR.y, b: 2)));
```

```
//Operación para calcular la distancia entre los puntos P y Q:
double resultadoPQ = Math.sqrt((Math.pow((PuntoP.x-PuntoQ.x),b: 2 ) + Math.pow((PuntoP.y-PuntoQ.y),b: 2 )));
System.out.printf(format: "La distancia entre PuntoP: (%1.1f:%1.1f)y PuntoQ: (%1.1f:%1.1f) es : %02.5f\n", args: PuntoP.x , args: PuntoP.y ,args: PuntoQ.x , args: PuntoQ.y , args: resultadoPQ);

//Operación para calcular la distancia entre los puntos P y R:
double resultadoPR = Math.sqrt((Math.pow((PuntoP.x-PuntoR.x),b: 2 ) + Math.pow((PuntoP.y-PuntoR.y),b: 2 )));
System.out.printf(format: "La distancia entre PuntoP: (%1.1f:%1.1f)y PuntoR: (%1.1f:%1.1f) es : %02.5f\n", args: PuntoP.x , args: PuntoP.y ,args: PuntoR.x , args: PuntoR.y , args: resultadoPR);

//Operación para calcular la distancia entre los puntos R y Q:
double resultadoRQ = Math.sqrt((Math.pow((PuntoR.x-PuntoQ.x),b: 2 ) + Math.pow((PuntoR.y-PuntoQ.y),b: 2 )));
System.out.printf(format: "La distancia entre PuntoP: (%1.1f:%1.1f)y PuntoQ: (%1.1f:%1.1f) es : %02.5f\n", args: PuntoR.x , args: PuntoR.y ,args: PuntoQ.x , args: PuntoQ.y , args: resultadoRQ);
```

Vicente Mena Micaelo

Ejemplo 3

En el tercer y último ejemplo que encontramos de código sucio, vemos que este tampoco es autoexplicativo, no disponemos de ningún comentario que nos ayude a la hora de saber qué es lo que está realizando el código y tenemos una función.

```
public static void main(String[] args) {
    Scanner scan = new Scanner(source: System.in);
    final int J = 5;
    String i;
    int k = (int)(Math.random()*101);
    int n, m = 0;
    System.out.println(x: "Esto es un juego en el que tienes que adivinar qué numero del 0 al 100 estoy pensando, para ello tienes 5 oportunidades.");
    do{
    System.out.printf(format: "%s, Intentos: [%02d] : ",args: "Por favor, introduzca un número del 0 al 100", args: m);
    i = scan.nextLine();
    n = Integer.valueOf(s: i);
    System.out.println("El número introducido es : " + n);
    m++;
    }while( n != k && m <= J );
    if( n == k ){
    System.out.println(x: "¡Felicidades! Has adivinado el número");
    }else{
    System.out.println("Lo siento, te has quedado sin intentos, el número era : " + k);
    }
}
```

Para poder mejorar y limpiar este código podemos realizar lo siguiente:

1. Modificar el nombre de la clase, objetos y variables por nombres con sentido, fáciles de pronunciar y buscar al igual que en los ejemplos anteriores:

```
final int Oportunidades = 5;
String introducidoString;
int aleatorio = (int)(Math.random()*101);
int introducidoInt, intentos = 0;
```

2. Añadir ciertos comentarios aclarativos para indicar que realiza el código, a pesar de que el código es autoexplicativo al igual que en ejemplos anteriores:

```
int aleatorio = (int)(Math.random()*101); //Operación para calcular numero aleatorio del 0 al 100
int introducidoInt, intentos = 0;
System.out.println(x: "Esto es un juego en el que tienes que adivinar qué numero del 0 al 100 estoy pensando, para ello tienes 5 oportunidades.");
  //Función para verificar si el número introducido corresponde con el generado
    do{
        System.out.printf(format: "%s, Intentos: [%02d] : ",args: "Por favor, introduzca un número del 0 al 100", args: intentos);
        introducidoString = scan.nextLine();
        introducidoInt = Integer.valueOf(s: introducidoString);
        System.out.println("El número introducido es : " + introducidoInt);
        intentos++;
    }while( introducidoInt != aleatorio && intentos <= Oportunidades );
```

3. Vemos que la función no es excesivamente larga, realiza una única cosa bien hecha, no abusa de switch / when, no tiene muchos argumentos, tampoco tiene los llamados “flag arguments”, no genera side effects y tampoco se repite, por tanto podemos decir que se trata de una función limpia:

```
//Función para verificar si el número introducido corresponde con el generado
  do{
      System.out.printf(format: "%s, Intentos: [%02d] : ",args: "Por favor, introduzca un número del 0 al 100", args: intent
      introducidoString = scan.nextLine();
      introducidoInt = Integer.valueOf(s: introducidoString);
      System.out.println("El número introducido es : " + introducidoInt);
      intentos++;
  }while( introducidoInt != aleatorio && intentos <= Oportunidades );
```