

CLEAN CODE

1. NOMBRES

- En primer lugar, los nombres que pongamos a diferentes variables o funciones **han de tener un significado**.



```
// Constantes para las marcas de participación
final static int M0 = 1;
final static int M1 = 2;
final static int M2 = 3;
final static int MxP = 10;
```

En este ejemplo, los nombres “M0”, “M1”, “M2” y “MxP” no nos dan ninguna información acerca del contenido, de forma que quien lea nuestro código no sabrá de qué datos estamos hablando.

- Estos nombres deben de ser **fáciles de pronunciar**.



```
// Constantes para las marcas de participación
final static int Mrc2000 = 1;
final static int Mrc2001 = 2;
final static int Mrc2002 = 3;
final static int nmMxPrtpes = 10;

// Variable para asignar dorsales
static int nmDrs1 = 0;
```

Si no, cuando queramos poner en común algo sobre el código y tengamos que hacer referencia de nuestra variable será más tedioso.

Nuestro código corregido sería el siguiente.



```
// Constantes para las marcas de participación
final static int Marca2000 = 1;
final static int Marca2001 = 2;
final static int Marca2002 = 3;
final static int numMaxParticipantes = 10;

// Variable para asignar dorsales
static int numDorsal = 0;
```

- Estos nombres deben de ser **fáciles de buscar**.



```
if (numDorsal < 10) {
    numDorsal++;
}
```



```
if (numDorsal < numMaxParticipantes) {
    numDorsal++;
}
```

Es preferible usar nombres lógicos para saber sobre qué estamos iterando.

- Si hablamos de métodos o **funciones**; usaremos verbos, de otro modo usaremos nombres corrientes.



```
// Registro de participantes
Participante(nombre: "Juan", marca: Marca2001);
Participante(nombre: "María", marca: Marca2000);
Participante(nombre: "Pedro", marca: Marca2002);

// Mostrar información de los participantes registrados
mostrarInformacionParticipantes();
}

// Función para registrar participantes
static void Participante(String nombre, int marca) {
```



```
// Registro de participantes
registrarParticipante(nombre: "Juan", marca: Marca2001);
registrarParticipante(nombre: "María", marca: Marca2000);
registrarParticipante(nombre: "Pedro", marca: Marca2002);

// Mostrar información de los participantes registrados
mostrarInformacionParticipantes();
}

// Función para registrar participantes
static void registrarParticipante(String nombre, int marca) {
```

- Usaremos **un mismo término para conceptos similares**.




```
final static int Marca2000 = 1;
final static int Marca2001 = 2;
final static int Marca2002 = 3;
```


En este caso, los términos "Marca2000", "Marca2001" y "Marca2002" son consistentes. Cada constante representa un año de participación en la competición, y la elección de nombres sigue un patrón claro y comprensible.

2. FUNCIONES


- Han de ser **pequeñas**, sin sobrepasar las 50-60 líneas, y cada una de ellas, no sobrepasar los 100-120 caracteres. La línea que se aprecia en las imágenes es una herramienta que nosotros podemos fijar para ayudarnos a ver ese límite.



```
// Función para registrar participantes
static void registrarParticipante(String nombre, int marca) {
    if (numDorsal < numMaxParticipantes) {
        numDorsal++;
        System.out.println("Registrando a " + nombre + " con dorsal " +
            numDorsal + " y marca " + obtenerNombreMarca(codigoMarca:marca));
    } else {
        System.out.println(x: "No hay cupo para más participantes.");
    }
}
```



```
// Función para obtener el nombre de la marca según el código
static String obtenerNombreMarca(int codigoMarca) {
    if (codigoMarca == Marca2000) {
        return "Marca 2000";
    } else if (codigoMarca == Marca2001) {
        return "Marca 2001";
    } else if (codigoMarca == Marca2002) {
        return "Marca 2002";
    } else {
        return "Marca Desconocida";
    }
}
```



```
// Función para mostrar la información de los participantes registrados
static void mostrarInformacionParticipantes() {
    System.out.println(x: "\nInformación de Participantes:");
    for (int i = 1; i <= numDorsal; i++) {
        System.out.println("Dorsal " + i + ": " +
            obtenerNombreMarca(codigoMarca:Marca2000) + " - Nombre: " + "Participante " + i);
    }
}
```

- Deben de **hacer 1 sola cosa**. Teniendo en cuenta el nombre que le damos a la función; esta no puede hacer otras cosas que el nombre no nos indica. Conocido como “**side effects**”.



```
// Función para registrar participantes
static void registrarParticipante(String nombre, int marca) {
    if (numDorsal < numMaxParticipantes) {
        numDorsal++;
        System.out.println("Registrando a " + nombre + " con dorsal " +
            numDorsal + " y marca " + obtenerNombreMarca(codigoMarca:marca));
    } else {
        System.out.println(x: "No hay cupo para más participantes.");
    }
    System.out.println(x: "\nInformación de Participantes:");
    for (int i = 1; i <= numDorsal; i++) {
        System.out.println("Dorsal " + i + ": " +
            obtenerNombreMarca(codigoMarca: Marca2000) + " - Nombre: " + "Participante " + i);
    }
}
```



```
static void registrarParticipante(String nombre, int marca) {
    if (numDorsal < numMaxParticipantes) {
        numDorsal++;
        System.out.println("Registrando a " + nombre + " con dorsal " +
            numDorsal + " y marca " + obtenerNombreMarca(codigoMarca:marca));
    } else {
        System.out.println(x: "No hay cupo para más participantes.");
    }
}
```



```
// Función para mostrar la información de los participantes registrados
static void mostrarInformacionParticipantes() {
    System.out.println(x: "\nInformación de Participantes:");
    for (int i = 1; i <= numDorsal; i++) {
        System.out.println("Dorsal " + i + ": " +
            obtenerNombreMarca(codigoMarca: Marca2000) + " - Nombre: " + "Participante " + i);
    }
}
```

- Los controles **switch** tienen dos problemas: suelen ser largos y realizan varias tareas a la vez. Por esto conviene evitarlos y cuando no es posible; recurrir al **polimorfismo**. Otra opción es hacer uso de bucles if-else:



```
static String obtenerNombreMarca(int codigoMarca) {  
    switch (codigoMarca) {  
        case Marca2000:  
            return "Marca 2000";  
        case Marca2001:  
            return "Marca 2001";  
        case Marca2002:  
            return "Marca 2002";  
        default:  
            return "Marca Desconocida";  
    }  
}
```



```
static String obtenerNombreMarca(int codigoMarca) {  
    if (codigoMarca == Marca2000) {  
        return "Marca 2000";  
    } else if (codigoMarca == Marca2001) {  
        return "Marca 2001";  
    } else if (codigoMarca == Marca2002) {  
        return "Marca 2002";  
    } else {  
        return "Marca Desconocida";  
    }  
}
```

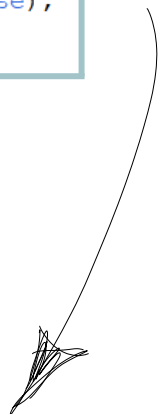
- El número de **argumentos por función** debe de ser menor a 3.

En cualquiera de los ejemplos anteriormente empleados, podemos ver que respetamos este principio, así el código no es fácil de razonar, y será complicado comprender qué va a ocurrir.

- **Evitar flag arguments.** Estos son valores adicionales que podemos asignar a los booleanos *true* y *false*. El problema de estos es que otra vez, supone que se hagan varias cosas a la vez en una función.



```
// Registro de participantes  
registrarParticipante(nombre: "Juan", marca: Marca2001, esElite: false);  
registrarParticipante(nombre: "María", marca: Marca2000, esElite: true);  
registrarParticipante(nombre: "Pedro", marca: Marca2002, esElite: false);  
mostrarInformacionParticipantes();
```





```
// Función para registrar participantes
static void registrarParticipante(String nombre, int marca, boolean esElite) {
    if (numDorsal < numMaxParticipantes) {
        numDorsal++;
        System.out.println("Registrando a " + nombre + " con dorsal " +
            numDorsal + " y marca " + obtenerNombreMarca(codigoMarca:marca));
        if (esElite) {
            System.out.println(nombre + " es un competidor de elite.");
        }
    } else {
        System.out.println(x: "No hay cupo para más participantes.");
    }
}
```

- **Don't repeat yourself.** Esto implica que si la lógica de nuestro código cambia, nos tenemos que acordar de modificarla en todas las partes donde se encuentre la duplicación.

3. COMENTARIOS

- **Los comentarios mienten.** Si modificamos nuestro código y no el comentario, es posible que el comentario contradiga el funcionamiento de nuestro código. Además, como los comentarios no compilan, nada nos avisa de este problema.
- **Usar código autoexplicativo.**



```
// Función para mostrar la información de los participantes registrados
static void mostrarInformacionParticipantes() {
    System.out.println(x: "\nInformación de Participantes:");
    for (int i = 1; i <= numDorsal; i++) {
        System.out.println("Dorsal " + i + ": " +
            obtenerNombreMarca(codigoMarca: Marca2000) + " - Nombre: " + "Participante " + i);
    }
}
```

En esta imagen, podemos ver que el comentario no nos está aportando nada. Si nombramos bien nuestros métodos y nuestro código es legible, será fácil entender qué va a suceder en él. Nuestro comentario está repitiendo lo que indica el nombre “*mostrarInformaciónImportante*”. Sobra.



```
static void mostrarInformacionParticipantes() {
    System.out.println(x: "\nInformación de Participantes:");
    for (int i = 1; i <= numDorsal; i++) {
        System.out.println("Dorsal " + i + ": " +
            obtenerNombreMarca(codigoMarca: Marca2000) + " - Nombre: " + "Participante " + i);
    }
}
```

- **A veces, los comentarios son necesarios.** Cuando el código es rebuscado o no puede explicarse por sí mismo debes usarlos para aclarar posibles confusiones.



```
// Función para obtener el nombre de la marca según el código
static String obtenerNombreMarca(int codigoMarca) {
    if (codigoMarca == Marca2000) {
        return "Marca 2000";
    } else if (codigoMarca == Marca2001) {
        return "Marca 2001";
    } else if (codigoMarca == Marca2002) {
        return "Marca 2002";
    } else {
        return "Marca Desconocida";
    }
}
```

En este caso, el comentario tampoco es necesario porque si leemos el código ya nos damos cuenta de que sacamos el nombre de la marca a partir del código de esta. No estamos ante un caso de código rebuscado o no-autoexplicativo.



```
static String obtenerNombreMarca(int codigoMarca) {
    if (codigoMarca == Marca2000) {
        return "Marca 2000";
    } else if (codigoMarca == Marca2001) {
        return "Marca 2001";
    } else if (codigoMarca == Marca2002) {
        return "Marca 2002";
    } else {
        return "Marca Desconocida";
    }
}
```

- Finalmente, **los comentarios dicen qué hace el código, no cómo lo hace.** El ejemplo anterior también refleja este principio; no es necesario decir de qué forma funcionará nuestro método.



```
// Función para obtener el nombre de la marca según el código
static String obtenerNombreMarca(int codigoMarca) {
```