Actividad 3

Pablo Sanchez Galdamez (21001135)

Contenidos

Clean Code rules

De que se trata?

El código es limpio si es fácil de entender. La idea de escribir código limpio es que este es mas fácil de leer, mantener, extender, etc.

Las reglas para el código limpio buscan crear convenciones estándar para que sea fácil escribirlo.

Ejemplos

1. No seas redundante con tus comentarios

Mal ejemplo:

```
/**
 * Un numero con valor inicial de 1
 *
 * @type {number}
 * @global
 * @tutorial https://www.w3schools.com/js/js_variables.asp
 * @todo Escribir mas ejemplos
 * @example
 *  // Asigna 2 al numero
 *  number = 2;
 */

let number = 1;
```

Buen ejemplo:

```
/**
 * Verifica si dos poligonos estan colisionando.
 *
 * *Nota:* Usamos el teorema del eje de separacion para verificar las
 * colisiones
 *
 * @param {Poligon} a
```

```
 * @param {Poligon} b
 * @return {boolean}
 */
const check_for_collision = (a, b) => {
    // ... Codigo ...
}
```

2. Usa nombres descriptivos para tus variables

Mal ejemplo:

```
var a=1, b="Pedro", c=0, d, e, f=0xBEEF;
```

Buen ejemplo:

```
const id = 1;
const name = "Pedro";

let days_active = 0;
let profile_page_background = 0xBEEF;
let current_selection, last_selection;
```

3. Declara las variables cerca de donde serán usadas

Mal ejemplo:

```
const id = 1;
const name = "Pedro";

let days_active = 0;
let current_selection, last_selection;

// ...
// 300 lineas de codigo usando las variables de arriba
// ...
```

Buen ejemplo:

```
// Datos del usuario
const id = 1;
const name = "Pedro";
let profile_page_background = 0xBEEF;

// ...
// Crear el perfil
// ...

let days_active = 0;
```

```
// ...
// Enviar notificacion
// ...

let current_selection, last_selection;

// ...
// Guardar eleccion
// ...
```

Code Size rules

De que se trata?

Si no se planea bien el código, es fácil estructurarlo incorrectamente. Puede resultar en funciones muy grandes, o muy complejas. Esto puede aumentar su complejidad exponencialmente.

Ejemplos

1. Funciones muy largas

Mal ejemplo:

```
void do_everything(void) {
    // ...
    // 1500 lineas de codigo
    // ...
}
```

Buen ejemplo:

```
WebData get_data() {
    // ...
    // 30 lineas de codigo
    // ...
}

std::vector<Entry> get_entries(const WebData& data) {
    // ...
    // 45 lineas de codigo
    // ...
}

// ...
// Varias funciones asi
// ...
```

2. Excesiva cantidad de parámetros

Mal ejemplo:

```
int get_average(int number1, int number2, int number3, int number4,
                int number5, int number6, int number7, int number8,
                int number9, int number10, int number11, int number12,
                int number13, int number14, int number15, int number16,
                int number17, int number18, int number19, int number20,
                int number21, int number22, int number23, int number24,
                int number25, int number26, int number27, int number28,
                int number29);
```

Buen ejemplo:

```
int get_average(const std::vector<int>& numbers);
```

3. Alta complejidad ciclomática

```
void do_everything(void) {
    while (running) {
        while (true) {
            for (const auto& pending : tasks) {
                if (service_is_up) {
                    // ...
                } else {
                    switch (backup_option) {
                    case USE_CACHE:
                        // ...
                        break;
                    case RETRY:
                        // ...
                        break;
                    default:
                        // ..
                        break;
                    }
                }
            }
        }
    }
}
```

Naming rules

De que se trata?

La convención de nombres o convención de nomenclatura es un conjunto de reglas para la elección de la secuencia de caracteres que se utilice para los identificadores que denoten variables, tipos, funciones y otras entidades en el código fuente y la documentación.

Ejemplos

1. Usa nombres descriptivos

Mal ejemplo:

```
var a=1, b="Pedro", c=0, d, e, f=0xBEEF;
```

Buen ejemplo:

```
const id = 1;
const name = "Pedro";

let days_active = 0;
let profile_page_background = 0xBEEF;
let current_selection, last_selection;
```

2. Usa nombres "buscables"

Mal ejemplo:

```
// Ctrl+F, buscas d, y obtienes cientos de resultados que no tienen nada que
// ver

let d = {
    // ...
}
```

Buen ejemplo:

```
// Ctrl+F, buscas proyect_data, y obtienes pocos resultados que revisar

let proyect_data = {
    // ...
}
```

3. No agregues prefijos ni información de tipos

Mal ejemplo:

```
# Con prefijos (notacion hungara) y postfijos.
```

```
iCantidad = 1
nombreString = "Pedro"
```

Buen ejemplo:

```
# Usando las anotaciones de tipo integradas al lenguaje

cantidad: int = 1
nombre: str = "Pedro"
```

Unused Code rules

De que se trata?

Ya que en el proceso de desarrollo podemos no empezar con una idea completamente clara de como sera la implementación, es normal que paremos con variables, funciones, etc, que no utilicemos en la versión final.

Estas reglas determinan que hacer con ese código que no se uso en la implementación final.

Ejemplos

1. Parámetro sin usar

```
const foo = (param) => {
    // No usamos `param` en ningun lado
}
```

2. Variable local sin usar

```
for (let i = 0; i < 100; i++) {
    const bar = i;

    // ...
    // Codigo sin usar `baz`
    // ...

    sum += parameter * i;
}
```

3. No comentes el código, solo remuévelo

Mal ejemplo:

```
const fib = (n) => {
    // console.log(`Current: ${n}`);
```

```
    if (n <= 1) {
        return n;
    }
    return fib(n-1) + fib(n-2);
}
```

Buen ejemplo:

```
const fib = (n) => {
    if (n <= 1) {
        return n;
    }
    return fib(n-1) + fib(n-2);
}
```

Controversial rules

De que se trata?

Estas son reglas sobre las que no se tiene un consenso general.

Ejemplos

1. No uses variables globales

```
#include <iostream>

int a, b, sum;

void calc_sum(void) {
    sum = a + b;
}

int main(void) {
    std::cout << "ingrese 2 numeros separados por 1 espacio: ";
    std::cin >> a >> b;
    calc_sum();
    std::cout << "La suma de los numeros es: " << sum << '\n';
}
```

2. No dupliques código:

Al tomar:

```
let Rectangle = {
  resizeTopLeft(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
```

```
  resizeTopRight(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
  resizeBottomLeft(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
  resizeBottomRight(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
};

let Oval = {
  resizeLeft(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
  resizeRight(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
  resizeTop(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
  resizeBottom(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
};

let Header = {
  resizeLeft(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
  resizeRight(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
}

let TextBlock = {
  resizeTopLeft(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
  resizeTopRight(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
  resizeBottomLeft(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
  resizeBottomRight(position, size, preserveAspect, dx, dy) {
```

```
    // 10 repetitive lines of math
  },
};
```

Y convertirlo en:

```
let {top, bottom, left, right} = Directions;

function createHandle(directions) {
  // 20 lines of code
}

let fourCorners = [
  createHandle([top, left]),
  createHandle([top, right]),
  createHandle([bottom, left]),
  createHandle([bottom, right]),
];
let fourSides = [
  createHandle([top]),
  createHandle([left]),
  createHandle([right]),
  createHandle([bottom]),
];
let twoSides = [
  createHandle([left]),
  createHandle([right]),
];

function createBox(shape, handles) {
  // 20 lines of code
}

let Rectangle = createBox(Shapes.Rectangle, fourCorners);
let Oval = createBox(Shapes.Oval, fourSides);
let Header = createBox(Shapes.Rectangle, twoSides);
let TextBox = createBox(Shapes.Rectangle, fourCorners);
```

Aunque se elimina en gran parte la duplicación, el código pierde una gran parte de su flexibilidad.

Ejemplo de **este blog**.

3. No uses abreviaciones al nombrar tus variables

No tiene que ser necesariamente malo en todos los casos. Por ejemplo, `tmp` es perfectamente claro que es una variable temporal, no es necesario llamar a la variable `temporary_storage_for_number`. Como mencionan **las convenciones**

de codigo para el kernel de linux, escribir únicamente `tmp` es mucho mas fácil, y también es mas fácil de entender.

Design rules

De que se trata?

Son reglas relacionadas a los problemas al diseñar el software.

Ejemplos

1. No usar `exit` (PHP, Python, etc), `abort` (C/C++), y similares.

Usar una expresión que termine el programa hace que sea difícil de testear.

```
class Foo {
    public function bar($param)  {
        if ($param === 42) {
            exit(23);
        }
    }
}
```

2. No usar `eval` y similares

Usar una expresión que evalué un `string` como código es difícil de probar y también es un problema de seguridad.

```
class Foo {
    public function bar($param)  {
        if ($param === 42) {
            eval('$param = 23;');
        }
    }
}
```

3. No usar `goto` y saltos similares

`goto` hace que el hilo del código sea difícil de seguir.

```
class Foo {
    public function bar($param)  {
        A:
        if ($param === 42) {
            goto X;
        }
        Y:
        if (time() % 42 === 23) {
```

```
            goto Z;
        }
        X:
        if (time() % 23 === 42) {
            goto Y;
        }
        Z:
        return 42;
    }
}
```