

PAC6. Refactoring

Josep V. Monjo

11/01/2021

1 Code smells

1.1 Duplicated code

Es tracta de codi que es repeteix al llarg de dos o més parts del nostre codi. El problema és que contribueix a engreixar el nostre *code base* i el fa més difícil de mantenir.

Exemple:

```
double(num: number): string {  
    const res: number = num * 2;  
    return `El nombre seleccionat és ${res}`  
}  
  
triple(num: number): string {  
    const res: number = num * 3;  
    return `El nombre seleccionat és ${res}`  
}
```

Possible solució: *Extract method*

1.2 Comments

Quan ens veiem obligats a incloure comentaris al nostre codi és possible que el propi codi no estiga prou clar o el nom de la classe o el mètode no inferisca la seua funció i per això necessite el comentari.

Exemple:

```
// Mètode per duplicar cada input

numQueNoTeResAVeure(input: number): number {
    return input * 2
}
```

Possible solució: *Rename method*

1.3 Long method

Es tracta d'un mètode amb massa línies de codi. Això fa el nostre codi més difícil de llegir i el converteix en un *spaghetti code*.

Exemple:

Si el nostre codi té més de 10 línies és possible que el podam simplificar.

Possible solució: *Extract method*

1.4 Switch statements

Es dona quan tenim una seqüència massa complexa de *switch* o de condicionals *if*.

El problema llavors és que si necessitem afegir una condició hem de canviar tot el nostre codi i el fa difícil de mantenir.

Exemple¹:

```
class Bird {  
    // ...  
    getSpeed(): number {  
        switch (type) {  
            case EUROPEAN:  
                return getBaseSpeed();  
            case AFRICAN:  
                return getBaseSpeed() - getLoadFactor() * numberOfCoconuts;  
            case NORWEGIAN_BLUE:  
                return (isNailed) ? 0 : getBaseSpeed(voltage);  
        }  
        throw new Error("Should be unreachable");  
    }  
}
```

Possible solució: *Replace Conditional with Polymorphism*

1.5 Primitive Obsession

Es tracta de l'abús de primitius per a tasques en les que seria més adient l'ús de classes.

Exemple:

```
const isAdmin = 1
```

Possible solució: *Replace Data Value with Object*

¹Exemple extret de sourcemaking.com

1.6 Long Parameter List

Es tracta d'un mètode amb més de tres o quatre paràmetres. Amb molts paràmetres el codi guanya complexitat i es fa difícil d'entendre.

Exemple²:

```
const finalPrice = discountedPrice(  
    basePrice,  
    seasonDiscount,  
    fees);
```

Possible solució: *Replace Parameter with Method Call*

2 Tècniques de refactoring

2.1 Extract method

S'usa quan tenim un mètode o classe molt llargs i podem trencar el codi en diverses parts més especialitzades.

Exemple:

```
//  
// Abans  
//  
productDetails(): void {  
    const product = getProduct();  
  
    console.log("name: " + product.name);  
    console.log("price: " + product.price);
```

²Exemple extret de sourcemaking.com

```
}  
  
//  
// Després  
//  
productDetails(): void {  
    const product = getProduct();  
    printProductDetails(product);  
}  
  
printProductDetails(product: Product): void {  
    console.log("name: " + product.name);  
    console.log("price: " + product.price);  
}
```

2.2 Extract variable

Quan tenim una expressió difícil de comprendre, assignar-la a una variable facilita la seua utilització per exemple en *if statements*.

Exemple:

```
// Abans  
getDetails(user: User): UserDetails {  
    if (user.type === 'Admin' ||  
        user.type === 'Manager' ||  
        user.type === 'Owner'  
    )  
    {  
        // do something  
    }  
}
```

```
    }  
  }  
  
  // Després  
  getDetails(user: User): UserDetails {  
    const isAdmin = user.type === 'Admin';  
    const isManager = user.type === 'Manager';  
    const isOwner = user.type === 'Owner';  
  
    if(isAdmin || isManager || isOwner)  
    {  
      // do something  
    }  
  }  
}
```

3 Refactoring switch/case amb el patró *Strategy*

4 Refactoring Tourist app