Code Principles (Princípios de código)

Princípio do ETC (Easier to Change)

> **ETC (Easier to Change)**. A facilidade de modificar o sistema deve ser a principal preocupação na hora que a funcionalidade está sendo desenvolvida.

O programador deve a cada commit, cada teste implementado, cada adição de ferramenta, **"O meu código deixa o sistema mais simples de ser modificado?"**

Se a resposta for **não**, então o sistema deve ser refatorado.

Clean Code

Código limpo é um das principais formas de garantir que um sistema seja mais fácil de modificiar.

- Nomenclatura
- Formatação
- Semântica

Modularidade

> O grau cujo componentes de um sistema podem ser separados e recombinados, com o objetivo de flexibilizar e variar seu uso.

Modularidade

O que esse código faz?

```
// Bad Modularity
function consoleLogCart(){
    var cart = Json.parse(localStorage.getItem("cart"));

    if(cart)
        throw "No items on cart"

    let str = ""
    for(const item of cart){
        str += `Product: ${item.name}\n`
        str += `Price: ${item.price}\n`

        const releaseDateYear = new Date(item.releaseDate).fullYear;
        str += `Release year: ${releaseDateYear}\n`
        str += "\n"
    }

    console.log(str)
}
```

Modularidade

```
// Good Modularity
function consoleLogCart(cart){
    let str = ""
    for(const item of cart){
        str += `Product: ${item.product}\n`
        str += `Price: ${item.price}\n`
        str += `Release year: ${item.releaseYear}\n`
        str += "\n"
    }
    console.log(str)
}

function getCart(mapper){
    var cart = Json.parse(localStorage.getItem("cart"));
    if(cart) throw "No items on cart"
    return mapper(cart)
}

function mapCart(cart) {
    var mappedCart = {items: []}
    for(const item of cart){
        mappedCart.items.push({ product: item.name, price: item.price, releaseYear: getFullYear(dateStr) })
    }
    return mappedCart
}

function getFullYear(dateStr){ return `${new Date(dateStr).fullYear}` }

// Chamada das funções
const cart = getCart(mapCart)
consoleLogCart(cart)
```

Coesão

> Cada componente de um sistema deve ter um **significado** e deve ser implementado de acordo com esse significado. **O principal objetivo de um código é comunicar ideias para humanos.**

- Coesão é mais dos que todas as outras ferramentas para gerenciar complexidade, **contextual**.

Coesão

O que esse código faz?

```
// Bad cohesion
function addToCart(item){
    var user = Json.parse(localStorage.getItem("user"))
    if(!user)
        throw "User must be logged in to add item to cart"

    var cart = Json.parse(localStorage.getItem("cart"));

    cart.push(item)

    localStorage.setItem("cart", Json.stringfy(cart))

    let totalCost = 0
    for(const cartItem in cart){
        totalCost += cartItem.price
    }

    return totalCost
}
```

Coesão

```
// Improved from bad cohesion
function addToCart(item){
    var user = getUser()
    if(!user)
        throw "User must be logged in to add item to cart"

    var cart = getCart();

    cart.push(item)

    setCart(cart)

    let totalCost = 0
    for(const cartItem in cart){
        totalCost += cartItem.price
    }

    return totalCost
}

function getUser(){ return Json.parse(localStorage.getItem("user")) }

function getCart(){ return Json.parse(localStorage.getItem("cart")) }

function setCart(cart) { localStorage.setItem("cart", Json.stringfy(cart)) }
```

Coesão

```
// Good cohesion
function addToCart(cart, item, next){
    // Validação de usuário deve ser feito em outro local

    cart.push(item)
    next(item) // Chama o próximo comando a ser executado
}
```

DRY

> DRY significa *Don't repeat yourself*
> **Intensão de garantir que um conceito dentro de um sistema esteja restrito apenas a um único ponto**

DRY

O que esse código faz?

```
function printBalance(account){

    console.log(`Debits: ${account.debits.toFixed(2)}\n`)
    console.log(`Credits: ${account.debits.toFixed(2)}\n`)

    console.log("        -----------\n")

    if(account.fees < 0){
        console.log(`Fees: ${-account.fees.toFixed(2)}`)
    }
    else {
        console.log(`Fees: ${account.fees.toFixed(2)}`)
    }

    if(account.balance < 0){
        console.log(`Balance: ${-account.balance.toFixed(2)}`)
    }
    else {
        console.log(`Balance: ${account.balance.toFixed(2)}`)
    }

}
```

DRY

Formatação de valores isolado

```
function printBalance(account){

    console.log(`Debits: ${formatAmount(account.debits)}\n`)
    console.log(`Credits: ${formatAmount(account.credits)}\n`)

    console.log("        -----------\n")

    if(account.balance < 0){
        console.log(`Fees: -${formatAmount(account.fees)}`)
    }
    else {
        console.log(`Fees: ${formatAmount(account.fees)}`)
    }

    if(account.balance < 0){
        console.log(`Balance: -${formatAmount(account.debits)}`)
    }
    else {
        console.log(`Balance: ${formatAmount(account.debits)}`)
    }
}

function formatAmount(amount) { return amount.toFixed(2) }
```

DRY

Formatação do valor negativo isolada

```
function printBalance(account){
    console.log(`Debits: ${formatAmount(account.debits)}\n`)
    console.log(`Credits: ${formatAmount(account.credits)}\n`)
    console.log("         -----------\n")
    console.log(`Fees: ${formatAmount(account.fees)}\n`)
    console.log(`Balance: ${formatAmount(account.debits)}\n`)
}

function formatAmount(amount) {
    result = Math.abs(amount).toFixed(2)

    if(amount < 0){
        return "-" + result
    }

    return " " + result
}
```

DRY

Formatação dos espaços e labels isolada

```
function printBalance(account){
    console.log(reportLine("Debits", account.debits))
    console.log(reportLine("Credits", account.credits))
    console.log(printLine("", "-----------"))
    console.log(reportLine("Fees", account.fees))
    console.log(reportLine("Balance", account.balance))
    console.log(reportLine("Debits", account.debits))
}

function reportLine(label, amount){ return printLine(label + ":", formatAmount(amount))}

function printLine(label, value){ return `${label} ${value}\n` }

function formatAmount(amount) {
    result = Math.abs(amount).toFixed(2)

    if(amount < 0){ return "-" + result }

    return " " + result
}
```