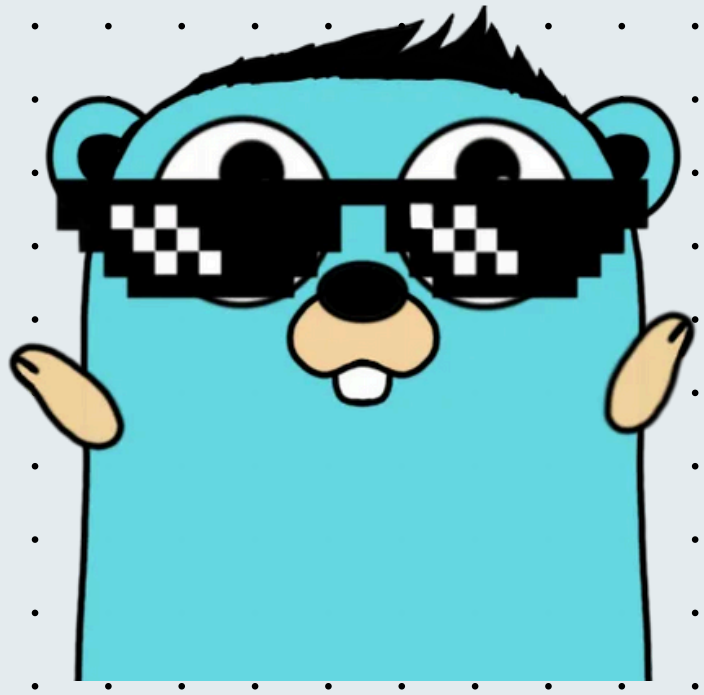
SOLID na prática

OPEN-CLOSED PRINCIPLE



Inspirado nos aprendizados adquiridos no curso Full Cycle e LeetCode.

/evelyncristinioliveira

SOLID na prática

OPEN-CLOSED PRINCIPLE

O Princípio Aberto Fechado (OCP) consiste em entidades (classes módulos, funções) devem estar abertas para extensão e fechadas para modificação. A ideia central é que o código deva funcionar sem alterações ao introduzir novos comportamentos. Manutenabilidade, flexibilidade, reusabilidade, e segurança são vantagens de se implementar o OCP.

No exemplo a seguir, será implementado um algoritmo para encontrar o moda (valor mais frequente em uma sequência) dentro de um *array* com números desordenados sem aplicar boas práticas e um outro algoritmo aplicando o O (*Open-Closed Principle*) do SOLID, respectivamente.

De maneira breve, o que será modificado de um algoritmo sem implementar boas práticas para um algoritmo que aplica o *Open-Closed Principle*, será:

- Adicionar interface *MajorityChecker* para verificação de elemento majoritário com método *FindMajorityElement*;
- Implementação de *MajorityElement*;
- Função *main* refatorada.

MODA SEM BOAS PRÁTICAS

```go
package main

import (
        "errors"
        "fmt"
)

func isAnEmptySlice(nums []int64) (int64, error) {
    if len(nums) == 0 {
        return 0, errors.New("slice is empty")
    }
    return 1, nil
}
```

/evelyncristinioliveira

MODA SEM BOAS PRÁTICAS

```go
func findMajorityElement(nums []int64) int64 {
    candidate := nums[0]
    count := 1

    for i := 1; i < len(nums); i++ {
        if nums[i] == candidate {
            count++
        } else {
            count--
            if count == 0 {
                candidate = nums[i]
                count = 1
```

/evelyncristinioliveira

MODA SEM BOAS PRÁTICAS

```
                }
            }
        }
        return candidate
    }

    func main() {
        nums1 := []int64{1000, 2, 300, 40, 300, 300, 1000}
        nums2 := []int64{2, 1000, 1, 500, 2, 20, 1000}

        _, firstSliceError := isAnEmptySlice(nums1)
        if firstSliceError != nil {
```

MODA SEM BOAS PRÁTICAS

```
    fmt.Println("error checking nums1:", firstSliceError)
    }
    _, secoundSliceError := isAnEmptySlice(nums2)
    if secoundSliceError != nil {
        fmt.Println("error checking nums2:", secoundSliceError)
    }

        majorityElement1 := findMajorityElement(nums1)
        majorityElement2 := findMajorityElement(nums2)
        fmt.Println("majority element in nums1:", majorityElement1)
        fmt.Println("majority element in nums2:",
majorityElement2)
}
```

/evelyncristinioliveira

MODA COM OCP

```go
package main

import (
	“errors”
	"fmt"
)

type MajorityChecker interface {
  FindMajorityElement(num []int64) (int64, error)
}

type MajorityElement struct{}
```

/evelyncristinioliveira

MODA COM OCP

```
func (h *MajorityElement) FindMajorityElement(nums
[]int64) (int64, error) {
        candidate := nums[0]
        count := 1

       for i := 1; i < len(nums); i++ {
            if nums[i] == candidate {
                count++
            } else {
                count--
                if count == 0 {
```

/evelyncristinioliveira

MODA COM OCP

```
                candidate = nums[i]
                count = 1
            }
        }
    }
    return candidate, nil
}

func isAnEmptySlice(checker MajorityChecker, nums
[]int64) (int64, error) {
        if len(nums) == 0 {
            return 0, errors.New("slice is empty")
```

/evelyncristinioliveira

MODA COM OCP

```
        }
        return checker.FindMajorityElement(nums)
    }

    func main() {
        nums1 := []int64{2, 40, 300, 40, 1000, 300}
        nums2 := []int64{1000, 2, 300, 300, 300, 40, 1000,
    1000}

        checker := &MajorityElement{}

        _, firstSliceError := isAnEmptySlice(checker, nums1)
```

MODA COM OCP

```
    if firstSliceError != nil {
        fmt.Println("error checking nums1:", firstSliceError)
    }

    _, secoundSliceError := isAnEmptySlice(checker, nums2)
    if secoundSliceError != nil {
        fmt.Println("error checking nums2:",
secoundSliceError)
    }

    majorityElement1, err :=
checker.FindMajorityElement(nums1)
```

/evelyncristinioliveira

MODA COM OCP

```
    if err != nil {
        fmt.Println("error finding majority element in
nums1:", err)
    } else {
        fmt.Println("majority element in nums1:",
majorityElement1)
    }

    majorityElement2, err :=
checker.FindMajorityElement(nums2)
```

/evelyncristinioliveira

MODA COM OCP

```
        if err != nil {
            fmt.Println("error finding majority element in
    nums2:", err)
        } else {
            fmt.Println("majority element in nums2:",
    majorityElement2)
        }
    }
```

/evelyncristinioliveira