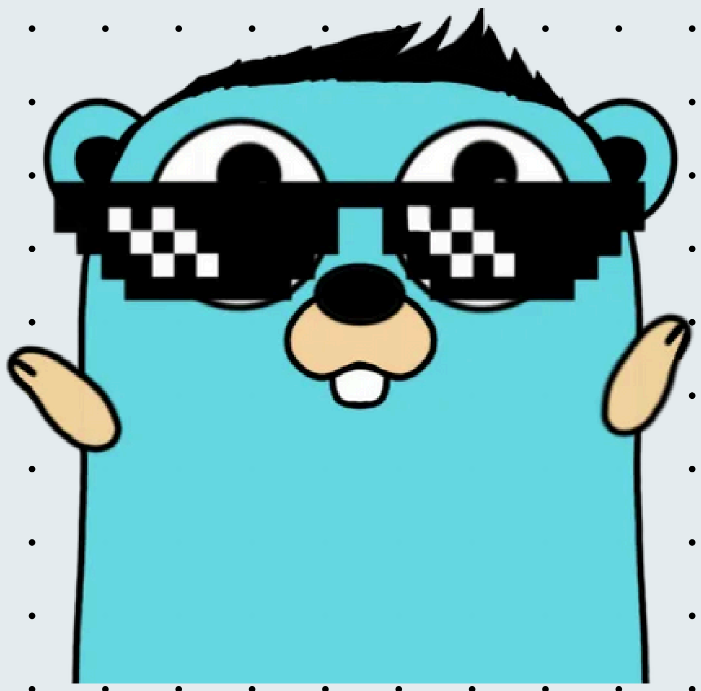
SOLID na prática

SINGLE RESPONSIBILITY PRINCIPLE

Inspirado nos aprendizados adquiridos no curso Full Cycle e LeetCode.



/evelyncristinioliveira

SOLID na prática

SINGLE RESPONSIBILITY PRINCIPLE

O Princípio da Responsabilidade Única (SRP) prevê que uma classe, função ou outro bloco de código deve absorver somente uma responsabilidade e nada além do seu propósito de criação. Os benefícios de implementarmos envolve o código se tornar mais modular, mais flexível, de menor propensão a erros e maior facilidade de testes.

No exemplo a seguir, será implementado um anagrama, palavra ou frase formada pela reorganização das letras de outra palavra ou frase, mantendo as letras originais e sem adicionar outra letra sem aplicar boas práticas seguido por um outro algoritmo aplicado ao S do SOLID.

ANAGRAMA SEM BOAS PRÁTICAS

```go
package main

import "fmt"

func IsAnagram(firstString, secoundString string) bool {
    if firstString == "" || secoundString == "" {
        return false
    }
    firstString = strings.ToLower(firstString)
    secoundString = strings.ToLower(secoundString)

    charactersMap := make(map[rune]int)
    for _, character := range firstString {
        charactersMap[character]++
    }
```

/evelyncristinioliveira

ANAGRAMA SEM BOAS PRÁTICAS

```
    for _, character := range secoundString {
        if count, ok := charactersMap[character]; ok {
            count--
            if count == 0 {
                delete(charactersMap, character)
            } else {
                charactersMap[character] = count
            }
        } else {
            return false
        }
    }
    return len(charactersMap) == 0
}
```

/evelyncristinioliveira

ANAGRAMA SEM BOAS PRÁTICAS

```go
 func main() {
      firstWord := "anagram"
      secoundWord := "nagaram"
      firstAndSecoundComparison := IsAnagram(firstWord,
secoundWord)
      fmt.Printf("First word is %s and secound word is %s. They are
anagrams? %v\n", firstWord, secoundWord,
firstAndSecoundComparison)

      thirdWord := "rat"
      fourthWord := "car"
      thirdAndFourthComparison := IsAnagram(thirdWord, fourthWord)
      fmt.Printf("Third word is %s and fourth word is %s. They are
anagrams? %v\n", thirdWord, fourthWord, thirdAndFourthComparison)
}
```

/evelyncristinioliveira

ANAGRAMA SEM BOAS PRÁTICAS

```
	emptyString := ""
	fifthWord := "test"
	emptyStringAndFifthWordComparison := IsAnagram(emptyString,
fifthWord)
	fmt.Printf("Empty string is %s and fifth w woris %s. They are
anagrams? %v\n", emptyString, fifthWord,
emptyStringAndFifthWordComparison)

	sixthWord := "iRaceMa"
	seventhWord := "aMerIca"
	sixthWordAndseventhWordComparison := IsAnagram(sixthWord,
seventhWord)
	fmt.Printf("Sixth word is %s and seventh word is %s. They are
anagrams? %v\n", sixthWord, seventhWord,
sixthWordAndseventhWordComparison)
}
```

/evelyncristinioliveira

ANAGRAMA COM SRP

```go
package main

import (
	"fmt"
	“strings“
)

type Anagram interface {
	IsAnagram(string, string) bool
}

type AnagramChecker struct{}
```

/evelyncristinioliveira

ANAGRAMA COM SRP

```go
func (a *AnagramChecker) IsAnagram(firstString, secoundString
string) bool {
     preparedFirst, preparedSecound, err :=
prepareStrings(firstString, secoundString)
    if err != nil {
        return false
    }

    charactersMap := make(map[rune]int)
    for _, character := range preparedFirst {
          charactersMap[character]++
    }
```

/evelyncristinioliveira

ANAGRAMA COM SRP

```go
    for _, character := range preparedSecound {
        if count, ok := charactersMap[character]; ok {
            count--
            if count == 0 {
                delete(charactersMap, character)
            } else {
                charactersMap[character] = count
            }
        } else {
            return false
        }
    }
    return len(charactersMap) == 0
}
```

/evelyncristinioliveira

ANAGRAMA COM SRP

```
func prepareStrings(firstString, secoundString string) (string, string,
error) {
        if firstString == "" || secoundString == "" {
            return "", "", fmt.Errorf("empty strings are not allowed")
        }
        return strings.ToLower(firstString),
strings.ToLower(secoundString), nil
}

 func main() {
         anagramChecker := AnagramChecker{}
         firstWord := "anagram"
         secoundWord := "nagaram"
```

/evelyncristinioliveira

ANAGRAMA COM SRP

```
	firstAndSecoundComparison :=
anagramChecker.IsAnagram(firstWord, secoundWord)
	fmt.Printf("First word is %s and secound word is %s. They are
anagrams? %v\n", firstWord, secoundWord,
firstAndSecoundComparison)

	thirdWord := "rat"
	fourthWord := "car"
	thirdAndFourthComparison :=
anagramChecker.IsAnagram(thirdWord, fourthWord)
	fmt.Printf("Third word is %s and fourth word is %s. They are
anagrams? %v\n", thirdWord, fourthWord,
thirdAndFourthComparison)
}
```

/evelyncristinioliveira

ANAGRAMA COM SRP

```
	sixthWord := "iRaceMa"
	seventhWord := "aMerIca"
	sixthWordAndseventhWordComparison :=
anagramChecker.IsAnagram(sixthWord, seventhWord)
	fmt.Printf("Sixth word is %s and seventh word is %s. They are
anagrams? %v\n", sixthWord, seventhWord,
sixthWordAndseventhWordComparison)
}
```

/evelyncristinioliveira