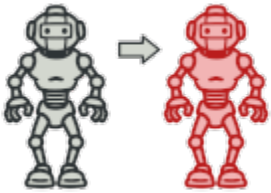
refactoring.guru

Prototype em Go / Padrões de Projeto

4–5 minutes



O **Prototype** é um padrão de projeto criacional que permite a clonagem de objetos, mesmo complexos, sem acoplamento à suas classes específicas.

Todas as classes de prototypes(protótipos) devem ter uma interface comum que permita copiar objetos, mesmo que suas classes concretas sejam desconhecidas. Objetos protótipos podem produzir cópias completas, pois objetos da mesma classe podem acessar os campos privados um do outro.

Exemplo conceitual

Vamos tentar descobrir o padrão Prototype usando um exemplo baseado no sistema de arquivos do sistema operacional. O sistema de arquivos do SO é recursivo: as pastas contêm arquivos e pastas, que também podem incluir arquivos e pastas e assim por diante.

Cada arquivo e pasta pode ser representado por uma interface `inode`. A interface `inode` também possui a função `clone`.

Ambas as structs `file` e `folder` implementam as funções `print` e `clone`, uma vez que são do tipo `inode`. Além disso, observe a função `clone` em `file` e `folder`. A função `clone` em ambos retorna uma cópia do respectivo arquivo ou pasta. Durante a clonagem, acrescentamos a palavra-chave "_clone" ao campo de nome.

inode.go: Interface do prototype

```
package main

type Inode interface {
    print(string)
    clone() Inode
}
```

file.go: Prototype concreto

```
package main

import "fmt"

type File struct {
    name string
}

func (f *File) print(indentation string) {
    fmt.Println(indentation + f.name)
}
```

```
func (f *File) clone() Inode {
    return &File{name: f.name + "_clone"}
}
```

folder.go: Prototype concreto

```
package main

import "fmt"

type Folder struct {
    children []Inode
    name     string
}

func (f *Folder) print(indentation string) {
    fmt.Println(indentation + f.name)
    for _, i := range f.children {
        i.print(indentation + indentation)
    }
}

func (f *Folder) clone() Inode {
    cloneFolder := &Folder{name: f.name + "_clone"}
    var tempChildren []Inode
    for _, i := range f.children {
        copy := i.clone()
        tempChildren = append(tempChildren, copy)
    }
    cloneFolder.children = tempChildren
```

```
    return cloneFolder
}
```

main.go: Código cliente

```
package main

import "fmt"

func main() {
    file1 := &File{name: "File1"}
    file2 := &File{name: "File2"}
    file3 := &File{name: "File3"}

    folder1 := &Folder{
        children: []Inode{file1},
        name:     "Folder1",
    }

    folder2 := &Folder{
        children: []Inode{folder1, file2, file3},
        name:     "Folder2",
    }
    fmt.Println("\nPrinting hierarchy for Folder2")
    folder2.print("  ")

    cloneFolder := folder2.clone()
    fmt.Println("\nPrinting hierarchy for clone Folder")
    cloneFolder.print("  ")
}
```

output.txt: Resultados da execução

```
Printing hierarchy for Folder2
  Folder2
    Folder1
      File1
    File2
    File3

Printing hierarchy for clone Folder
  Folder2_clone
    Folder1_clone
      File1_clone
    File2_clone
    File3_clone
```

Prototype em outras linguagens