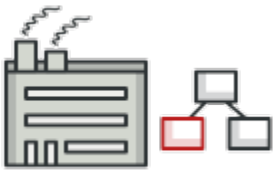
refactoring.guru

Factory Method em Go / Padrões de Projeto

3–4 minutes



O **Factory method** é um padrão de projeto criacional, que resolve o problema de criar objetos de produtos sem especificar suas classes concretas.

O Factory Method define um método, que deve ser usado para criar objetos em vez da chamada direta ao construtor (operador `new`). As subclasses podem substituir esse método para alterar a classe de objetos que serão criados.

> Se você não conseguir descobrir a diferença entre os padrões **Factory**, **Factory Method** e **Abstract Factory**, leia nossa Comparação Factory.

Exemplo conceitual

É impossível implementar o padrão Factory Method clássico no Go devido à falta de recursos OOP, como classes e herança. No entanto, ainda podemos implementar a versão básica do padrão, o Factory Simples.

Neste exemplo, vamos construir vários tipos de armas usando uma struct factory.

Primeiro, criamos a interface `iGun`, que define todos os métodos que uma arma deve ter. Existe um tipo de struct gun que implementa a interface iGun. Duas armas concretas — `ak47` e `musket` — ambas incorporam a struct da arma e indiretamente implementam todos os métodos `iGun`.

A struct gun`Factory` serve como um factory, que cria armas do tipo desejado com base em um argumento de entrada. O *main.go* atua como o cliente. Em vez de interagir diretamente com o `ak47` ou `musket`, ele conta com o gun`Factory` para criar instâncias de várias armas, usando apenas parâmetros de tipo string para controlar a produção.

iGun.go: Interface do produto

```
package main

type IGun interface {
    setName(name string)
    setPower(power int)
    getName() string
    getPower() int
}
```

gun.go: Produto concreto

```
package main

type Gun struct {
```

```
    name  string
    power int
}

func (g *Gun) setName(name string) {
    g.name = name
}

func (g *Gun) getName() string {
    return g.name
}

func (g *Gun) setPower(power int) {
    g.power = power
}

func (g *Gun) getPower() int {
    return g.power
}
```

ak47.go: Produto concreto

```
package main

type Ak47 struct {
    Gun
}

func newAk47() IGun {
    return &Ak47{
```

```
        Gun: Gun{
            name:  "AK47 gun",
            power: 4,
        },
    }
}
```

musket.go: Produto concreto

```
package main

type musket struct {
    Gun
}

func newMusket() IGun {
    return &musket{
        Gun: Gun{
            name:  "Musket gun",
            power: 1,
        },
    }
}
```

gunFactory.go: Factory

```
package main

import "fmt"
```

```
func getGun(gunType string) (IGun, error) {
    if gunType == "ak47" {
        return newAk47(), nil
    }
    if gunType == "musket" {
        return newMusket(), nil
    }
    return nil, fmt.Errorf("Wrong gun type passed")
}
```

main.go: Código cliente

```
package main

import "fmt"

func main() {
    ak47, _ := getGun("ak47")
    musket, _ := getGun("musket")

    printDetails(ak47)
    printDetails(musket)
}

func printDetails(g IGun) {
    fmt.Printf("Gun: %s", g.getName())
    fmt.Println()
    fmt.Printf("Power: %d", g.getPower())
    fmt.Println()
}
```

output.txt: Resultados da execução

```
Gun: AK47 gun
Power: 4
Gun: Musket gun
Power: 1
```