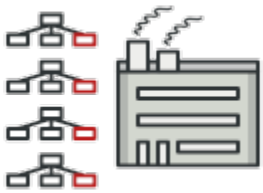
refactoring.guru

Abstract Factory em Go / Padrões de Projeto

5–7 minutes



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

O Abstract Factory define uma interface para criar todos os produtos distintos, mas deixa a criação real do produto para classes fábrica concretas. Cada tipo de fábrica corresponde a uma determinada variedade de produtos.

O código cliente chama os métodos de criação de um objeto fábrica em vez de criar produtos diretamente com uma chamada de construtor (usando operador new). Como uma fábrica corresponde a uma única variante de produto, todos os seus produtos serão compatíveis.

O código cliente trabalha com fábricas e produtos somente através de suas interfaces abstratas. Ele permite que o mesmo código cliente funcione com produtos diferentes. Você apenas cria uma nova classe fábrica concreta e a passa para o código cliente.

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

Exemplo conceitual

Digamos que você precise comprar um kit esportivo, um conjunto de dois produtos diferentes: um par de sapatos e uma camisa. Você gostaria de comprar um kit esportivo completo da mesma marca para combinar com todos os itens.

Se tentarmos transformar isso em código, o abstract factory nos ajudará a criar conjuntos de produtos para que sempre correspondam uns aos outros.

iSportsFactory.go: Interface do abstract factory

```
package main

import "fmt"

type ISportsFactory interface {
    makeShoe() IShoe
    makeShirt() IShirt
}

func GetSportsFactory(brand string) (ISportsFactory, error) {
    if brand == "adidas" {
        return &Adidas{}, nil
    }
```

```
    if brand == "nike" {
        return &Nike{}, nil
    }

    return nil, fmt.Errorf("Wrong brand type passed")
}
```

adidas.go: Factory concreto

```
package main

type Adidas struct {
}

func (a *Adidas) makeShoe() IShoe {
    return &AdidasShoe{
        Shoe: Shoe{
            logo: "adidas",
            size: 14,
        },
    }
}

func (a *Adidas) makeShirt() IShirt {
    return &AdidasShirt{
        Shirt: Shirt{
            logo: "adidas",
            size: 14,
        },
    }
```

```
}
```

nike.go: Factory concreto

```
package main

type Nike struct {
}

func (n *Nike) makeShoe() IShoe {
    return &NikeShoe{
        Shoe: Shoe{
            logo: "nike",
            size: 14,
        },
    }
}

func (n *Nike) makeShirt() IShirt {
    return &NikeShirt{
        Shirt: Shirt{
            logo: "nike",
            size: 14,
        },
    }
}
```

iShoe.go: Produto abstrato

```
package main
```

```
type IShoe interface {
    setLogo(logo string)
    setSize(size int)
    getLogo() string
    getSize() int
}

type Shoe struct {
    logo string
    size int
}

func (s *Shoe) setLogo(logo string) {
    s.logo = logo
}

func (s *Shoe) getLogo() string {
    return s.logo
}

func (s *Shoe) setSize(size int) {
    s.size = size
}

func (s *Shoe) getSize() int {
    return s.size
}
```

adidasShoe.go: Produto concreto

```
package main

type AdidasShoe struct {
    Shoe
}
```

nikeShoe.go: Produto concreto

```
package main

type NikeShoe struct {
    Shoe
}
```

iShirt.go: Produto abstrato

```
package main

type IShirt interface {
    setLogo(logo string)
    setSize(size int)
    getLogo() string
    getSize() int
}

type Shirt struct {
    logo string
    size int
}
```

```
func (s *Shirt) setLogo(logo string) {
    s.logo = logo
}

func (s *Shirt) getLogo() string {
    return s.logo
}

func (s *Shirt) setSize(size int) {
    s.size = size
}

func (s *Shirt) getSize() int {
    return s.size
}
```

adidasShirt.go: Produto concreto

```
package main

type AdidasShirt struct {
    Shirt
}
```

nikeShirt.go: Produto concreto

```
package main

type NikeShirt struct {
    Shirt
```

```
}
```

main.go: Código cliente

```
package main

import "fmt"

func main() {
    adidasFactory, _ := GetSportsFactory("adidas")
    nikeFactory, _ := GetSportsFactory("nike")

    nikeShoe := nikeFactory.makeShoe()
    nikeShirt := nikeFactory.makeShirt()

    adidasShoe := adidasFactory.makeShoe()
    adidasShirt := adidasFactory.makeShirt()

    printShoeDetails(nikeShoe)
    printShirtDetails(nikeShirt)

    printShoeDetails(adidasShoe)
    printShirtDetails(adidasShirt)
}

func printShoeDetails(s IShoe) {
    fmt.Printf("Logo: %s", s.getLogo())
    fmt.Println()
    fmt.Printf("Size: %d", s.getSize())
    fmt.Println()
```

```
}

func printShirtDetails(s IShirt) {
    fmt.Printf("Logo: %s", s.getLogo())
    fmt.Println()
    fmt.Printf("Size: %d", s.getSize())
    fmt.Println()
}
```

output.txt: Resultados da execução

```
Logo: nike
Size: 14
Logo: nike
Size: 14
Logo: adidas
Size: 14
Logo: adidas
Size: 14
```