refactoring.guru

Facade em Go / Padrões de Projeto

6–8 minutes

O **Facade** é um padrão de projeto estrutural que fornece uma interface simplificada (mas limitada) para um sistema complexo de classes, biblioteca, ou framework.

Embora o Facade diminua a complexidade geral do aplicativo, também ajuda a mover dependências indesejadas para um só local.

Exemplo conceitual

É fácil subestimar as complexidades que acontecem nos bastidores quando você pede uma pizza com cartão de crédito. Existem dezenas de subsistemas que atuam nesse processo. Aqui está apenas uma lista deles:

- Checar conta
- Verificar o PIN de segurança
- Saldo do crédito/débito
- Fazer uma entrada no livro-caixa

- Enviar notificação

  Em um sistema complexo como este, é fácil se perder e quebrar coisas se você estiver fazendo algo errado. É por isso que existe o conceito do padrão Facade: uma coisa que permite o cliente trabalhar com dezenas de componentes usando uma interface simples. O cliente só precisa inserir os dados do cartão, o PIN de segurança, o valor a pagar, e o tipo de operação. O Facade direciona comunicações adicionais com vários componentes sem expor o cliente a complexidades internas.

  walletFacade.go: Fachada

```
package main

import "fmt"

type WalletFacade struct {
    account      *Account
    wallet       *Wallet
    securityCode *SecurityCode
    notification *Notification
    ledger       *Ledger
}

func newWalletFacade(accountID string, code int) *WalletFacade {
    fmt.Println("Starting create account")
    walletFacacde := &WalletFacade{
        account:      newAccount(accountID),
        securityCode: newSecurityCode(code),
```

```
        wallet:       newWallet(),
        notification: &Notification{},
        ledger:       &Ledger{},
    }
    fmt.Println("Account created")
    return walletFacacde
}

func (w *WalletFacade) addMoneyToWallet(accountID string,
securityCode int, amount int) error {
    fmt.Println("Starting add money to wallet")
    err := w.account.checkAccount(accountID)
    if err != nil {
        return err
    }
    err = w.securityCode.checkCode(securityCode)
    if err != nil {
        return err
    }
    w.wallet.creditBalance(amount)
    w.notification.sendWalletCreditNotification()
    w.ledger.makeEntry(accountID, "credit", amount)
    return nil
}

func (w *WalletFacade) deductMoneyFromWallet(accountID string,
securityCode int, amount int) error {
    fmt.Println("Starting debit money from wallet")
    err := w.account.checkAccount(accountID)
    if err != nil {
```

```
        return err
    }

    err = w.securityCode.checkCode(securityCode)
    if err != nil {
        return err
    }
    err = w.wallet.debitBalance(amount)
    if err != nil {
        return err
    }
    w.notification.sendWalletDebitNotification()
    w.ledger.makeEntry(accountID, "debit", amount)
    return nil
}
```

account.go: Partes do subsistema complexo

```
package main

import "fmt"

type Account struct {
    name string
}

func newAccount(accountName string) *Account {
    return &Account{
        name: accountName,
    }
```

```
}

func (a *Account) checkAccount(accountName string) error {
    if a.name != accountName {
        return fmt.Errorf("Account Name is incorrect")
    }
    fmt.Println("Account Verified")
    return nil
}
```

securityCode.go: Partes do subsistema complexo

```
package main

import "fmt"

type SecurityCode struct {
    code int
}

func newSecurityCode(code int) *SecurityCode {
    return &SecurityCode{
        code: code,
    }
}

func (s *SecurityCode) checkCode(incomingCode int) error {
    if s.code != incomingCode {
        return fmt.Errorf("Security Code is incorrect")
    }
```

```
    fmt.Println("SecurityCode Verified")
    return nil
}
```

wallet.go: Partes do subsistema complexo

```
package main

import "fmt"

type Wallet struct {
    balance int
}

func newWallet() *Wallet {
    return &Wallet{
        balance: 0,
    }
}

func (w *Wallet) creditBalance(amount int) {
    w.balance += amount
    fmt.Println("Wallet balance added successfully")
    return
}

func (w *Wallet) debitBalance(amount int) error {
    if w.balance < amount {
        return fmt.Errorf("Balance is not sufficient")
    }
```

```
    fmt.Println("Wallet balance is Sufficient")
    w.balance = w.balance - amount
    return nil
}
```

ledger.go: Partes do subsistema complexo

```
package main

import "fmt"

type Ledger struct {
}

func (s *Ledger) makeEntry(accountID, txnType string, amount int)
{
    fmt.Printf("Make ledger entry for accountId %s with txnType %s
for amount %d\n", accountID, txnType, amount)
    return
}
```

notification.go: Partes do subsistema complexo

```
package main

import "fmt"

type Notification struct {
}
```

```
func (n *Notification) sendWalletCreditNotification() {
    fmt.Println("Sending wallet credit notification")
}

func (n *Notification) sendWalletDebitNotification() {
    fmt.Println("Sending wallet debit notification")
}
```

main.go: Código cliente

```
package main

import (
    "fmt"
    "log"
)

func main() {
    fmt.Println()
    walletFacade := newWalletFacade("abc", 1234)
    fmt.Println()

    err := walletFacade.addMoneyToWallet("abc", 1234, 10)
    if err != nil {
        log.Fatalf("Error: %s\n", err.Error())
    }

    fmt.Println()
    err = walletFacade.deductMoneyFromWallet("abc", 1234, 5)
    if err != nil {
```

```
        log.Fatalf("Error: %s\n", err.Error())
    }
}
```

output.txt: Resultados da execução

```
Starting create account
Account created

Starting add money to wallet
Account Verified
SecurityCode Verified
Wallet balance added successfully
Sending wallet credit notification
Make ledger entry for accountId abc with txnType credit for
amount 10

Starting debit money from wallet
Account Verified
SecurityCode Verified
Wallet balance is Sufficient
Sending wallet debit notification
Make ledger entry for accountId abc with txnType debit for amount
5
```