Padrões de Projeto

Bruno Gava Guerra

PUCRS – Engenharia de Software

Nota do Autor

Documento referente à trabalho da disciplina de Projeto e Arquitetura de Software

Repositório no GitHub: https://github.com/guerra08/ecommerce-patterns/

O que são Padrões de Projeto

Padrões de projeto são, de maneira simplificada, maneiras de organizar e reaproveitar trechos de código para melhor solucionar determinado problema. Assim, tais padrões podem surgir em escalas menores, para resolverem problemas pontuais em cenários de escopo reduzido. No caso do presente documento, trata-se de padrões de projeto formalizados e elaborados pela academia e pelas empresas de software.

Padrões solicitados

Foi solicitada a implementação dos seguintes padrões de projeto para a atividade.

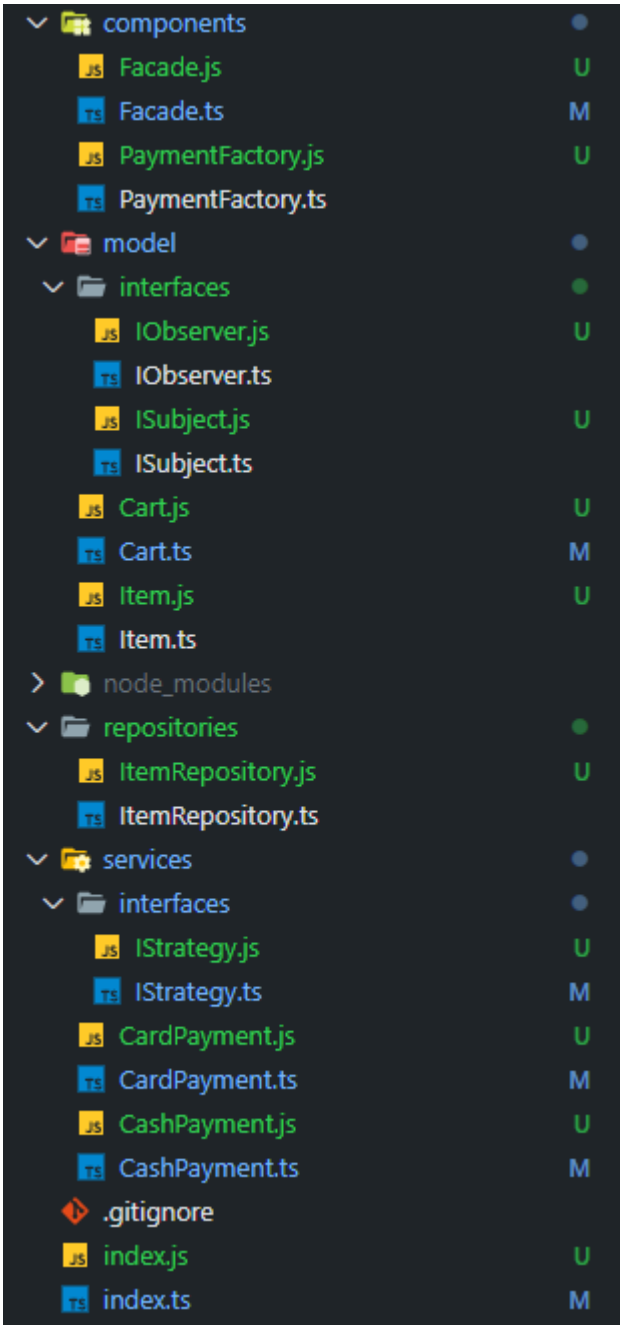
- Singleton
- Observer
- Fachada
- Factory
- Strategy.

O problema

O problema que deve ser resolvido no trabalho envolve o uso de um carrinho de compras para um *e-commerce*. Assim, precisa-se adicionar produtos ao carrinho, visualizar o carrinho, finalizar a compra e escolher um meio de pagamento. Optou-se por uma abordagem de aplicativo de linha de comando.

A implementação

A implementação foi realizada utilizando NodeJS em combinação com TypeScript. Além disso, foi utilizada a biblioteca readline-sync para leitura dos *inputs* do usuário no terminal. A seguir, segue imagem da estrutura de diretórios do projeto.



O *entry point* da aplicação é o arquivo *index.js*. Nota-se a presença dos arquivos .ts e .js, uma vez os primeiros são transpilados para o formato JavaScript. A seguir, segue a implementação de cada padrão de projeto.

Singleton

O padrão Singleton foi utilizado no carrinho, fazendo com que seja possível instanciar apenas um objeto deste tipo por usuário.

```
import Item from './Item';        You, a day ago • Refactoring facade
import IObserver from './interfaces/IObserver';
import ISubject from './interfaces/ISubject';

You, 18 minutes ago | 1 author (You)
export default class Cart implements IObserver {
    private static instance: Cart;

    private items: Map<Item, number> = new Map();

    private constructor() {}

    public static getInstance(): Cart {
        if (!Cart.instance) {
            Cart.instance = new Cart();
        }
        return Cart.instance;
    }
```

Observer

O Observer foi utilizado no carrinho e nos itens (produtos) que podem ser comprados. Na imagem anterior, é possível ver que a classe Cart implementa a interface IObserver. A seguir, serão apresentadas as imagens referentes a esse padrão.

```
import ISubject from "./ISubject";

You, 3 days ago | 1 author (You)
export default interface IObserver {
    update(subject: ISubject, amnt: number): void;
}
```

```
import IObserver from "./IObserver";

You, 3 days ago | 1 author (You)
export default interface ISubject {
    attach(observer: IObserver): void;

    detach(observer: IObserver): void;

    notify(amnt: number): void;
}
```

```
import ISubject from "./interfaces/ISubject";
import IObserver from "./interfaces/IObserver";

You, 2 days ago | 1 author (You)
export default class Item implements ISubject {      You, 3 days ago
    private name: string;
    private manufacturer: string;
    private value: number;
    private observers: IObserver[] = [];

    constructor(n: string, m: string, val: number) {
        this.name = n;
        this.manufacturer = m;
        this.value = val;
    }

    public attach(observer: IObserver): void {
        if (this.observers.includes(observer)) {
            return console.log("Already attatched");
        }
        this.observers.push(observer);
        console.log("Subject attatched");
    }

    public detach(observer: IObserver): void {
        const index = this.observers.indexOf(observer);
        if (index === -1) {
            return console.log("Subject not attached.");
        }
        this.observers.splice(index);
        console.log("Subject detached");
    }

    public notify(amnt: number): void {
        for (const o of this.observers) {
            o.update(this, amnt);
        }
    }
```

Fachada

Utilizou-se a fachada para garantir que apenas ela seja chamada para a execução do código interno da aplicação. A função startShoppingProcess fica encarregada de executar todo o código envolvendo o *e-commerce*, que encontra na função mainLoop.

```
You, a few seconds ago | 1 author (You)
import ItemRepository from '../repositories/ItemRepository';
import Cart from '../model/Cart';
import * as rlSync from 'readline-sync';
import PaymentFactory from './PaymentFactory';

You, a few seconds ago | 1 author (You)
export default class Facade {
    private itemRepo = new ItemRepository();
    private cart = Cart.getInstance();
    private pFactory = new PaymentFactory();

    constructor() {
        this.itemRepo.attachCart(this.cart);
    }

    public startShoppingProcess() {
        console.log('\nSeja bem vindo(a)!');
        this.mainLoop();
    }

    private mainLoop() {
        let flag = false;
        while (!flag) {
            flag = this.opManager();
        }
    }

    private manageCart() {
        console.log('Seu carrinho: \n');
        console.log(this.cart.cartToString());
    }

    private addItemToCart() {
        console.log(this.itemRepo.allItemsToString());
        const index = rlSync.questionInt('Selecione o produto: ');
        const amnt = rlSync.questionInt('Quantas unidades? ');
        this.itemRepo.retrieveItem(index).selectItem(amnt);
    }
```

Factory e Strategy

O padrão Factory foi implementado em conjunto do padrão Strategy, sendo ambos encarregados dos serviços de pagamento. Assim, a classe PaymentFactory fica encarregada de chamar o tipo de pagamento correspondente à escolha feita pelo usuário.

```ts
import IStrategy from '../services/interfaces/IStrategy';
import CardPayment from '../services/CardPayment';
import CashPayment from '../services/CashPayment';

You, 5 hours ago | 1 author (You)
export default class PaymentFactory {
    public getPaymentMethod(type: string): IStrategy | null {
        if (type === 'Cartão') return new CardPayment();
        if (type === 'Dinheiro') return new CashPayment();
        return null;
    }
}
```

```ts
export default interface IStrategy {
    pay(value: number): boolean;
}
```

```ts
import IStrategy from './interfaces/IStrategy';
import * as rlSync from 'readline-sync';

You, a few seconds ago | 1 author (You)
export default class CardPayment implements IStrategy {
    public pay(value: number): boolean {
        rlSync.question('Digite o número do seu cartão: ');
        rlSync.question('Digite o CVV do cartão: ');
        console.log(`Compra de $${value} realizada com sucesso! `);
        return true;
    }
}
```

```ts
import IStrategy from './interfaces/IStrategy';
import * as rlSync from 'readline-sync';

You, a few seconds ago | 1 author (You)
export default class CashPayment implements IStrategy {
    public pay(value: number): boolean {
        const cashVal = rlSync.questionFloat(
            'Digite o valor do pagamento em dinheiro: '
        );
        if (cashVal < value) {
            console.log('Valor inválido, tente novamente!');
            return false;
        }
        if (cashVal > value) {
            console.log(
                `Compra de $${value} realizada com sucesso! Troco de $${
                    cashVal - value
                }`
            );
        }
        if (cashVal == value) {
            console.log(`Compra de $${value} realizada com sucesso!`);
        }
        return true;
    }
}
```