**Exercícios**de Avaliação 4

**Disciplina de Projeto e Arquitetura de Software**

**Alunos**

Alberto Rocha Pinalli

Luiz Guerra

Vitor Demenighi

Neste exercício de avaliação os alunos deverão explorar os padrões de projeto trabalhados nos últimos dias:

**- *Singleton***

**- *Observer***

**- *Facade***

**- *Factory***

**- *Strategy*.**

O problema a ser resolvido continua o mesmo do exercício proposto, o e-commerce.

Sugestões de aplicação destes padrões para este problema:

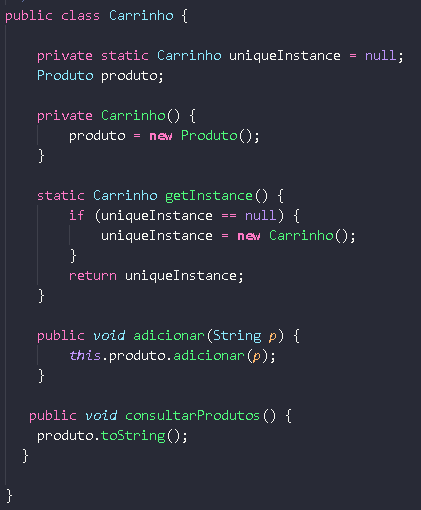
1. ***singleton***, para garantir um único carrinho de compras para um cliente;
2. ***observer***, atualização do carrinho a cada produto selecionado pelo cliente;
3. ***strategy* e *factory*** para implementar as diferentes formas de pagamento e o acesso ao cliente a estas informações (cartão de débito, cartão de crédito, boleto bancário e *paypal*).
4. ***fachada***; classe a ser definida na camada de negócios como um único ponto de acesso da camada de apresentação a ela.

**Cada grupo deve entregar um arquivo texto com uma descrição e screen shots das classes/métodos implementados para cada padrão de projeto**. **Este documento deve conter o link do git para acesso ao projeto.**

**Padrão Singleton**

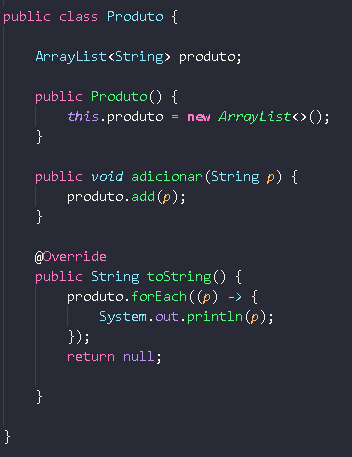
**Classe Carrinho**

Tem por objetivo garantir a criação de uma única instancia do objeto carrinho através do método *getInstance*(). Possui os métodos adicionar e consultar produtos no carrinho e o método construtor.



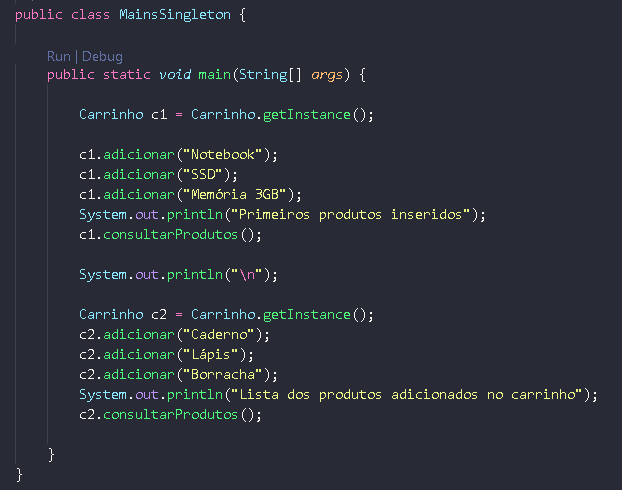
**Classe Produto**

Tem por finalidade criar um *ArrayList* de produtos para inserir no Carrinho. Possui o método construtor, adicionar produtos e o *ToString*.

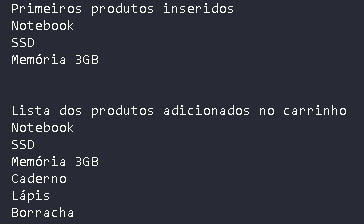


**Classe *MainSingleton***

Tem como objetivo instanciar o Carrinho, adicionar produtos no carrinho chamando o método adicionar. E a apresentação do resultado ao usuário.



**Console**

****

**Padrão *Observer***

**Classe *Subject***

O objetivo da classe é permitir ao usuário escolher um produto para adicionar ao carrinho.

* Possui a declaração de um *ArrayList* para inserção de produtos através do método adicionar;
* Possui o método construtor e o método consultar para a visualização de produtos no carrinho;
* Possui o método inseriu implementado da interface *Observer*, método que atualiza as informações quando há alteração no carrinho o método inseriu é chamado pelo método *notificarObserver* da classe *ConcreteObserver*.



**Classe ConcreteObserver**

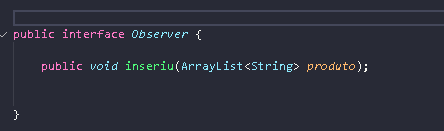
É a classe observada, possui um *ArrayList* de produtos e um *ArrayList* de *observers*.

Além do método construtor, possui os métodos adicionar, responsável por adicionar os produtos no carrinho, o método *addObserver*, responsável por adicionar um *observer*, o método *removeObserver* responsável pela remoção de um *observer* e o método *notificarObserver* que é chamado pelo método inseriu da classe *Subject* e o método *toString*.



***Interface Observer***

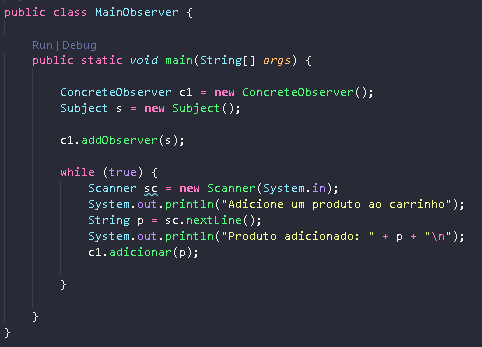
Responsável por garantir que a classe *Subject* implemente o método inseriu, responsável por notificar a classe *ConcreteObserver* sobre alterações no carrinho.



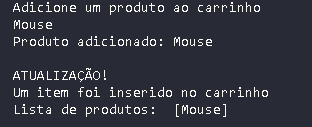
**Classe *MainObserver***

Responsável por instanciar as classes *ConcreteObserver* e *Subject* para poderem criar o objeto c1 e inserir produtos no carrinho.

Possui o método *addObserver* para um *observer* poder informar das alterações realizadas no carrinho.



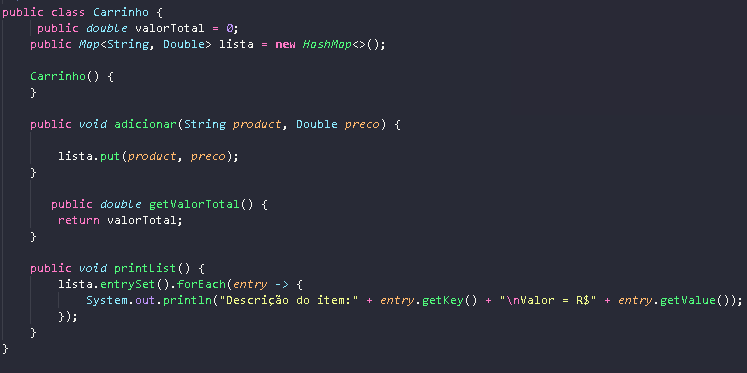
**Console**

****

**Padrão Strategy**

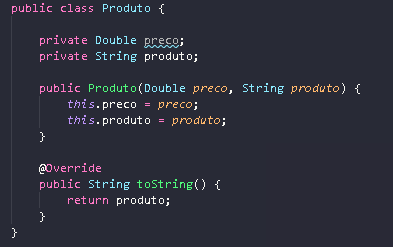
**Classe Carrinho**

Possui um *Map* de produtos e valor. Além do método construtor, possui os métodos adicionar, responsável por adicionar os produtos no carrinho e valor, o método *getValorTotal*, responsável por retornar o valor total do carrinho, o método *printList* responsável pela descrição dos itens e seus valores.

****

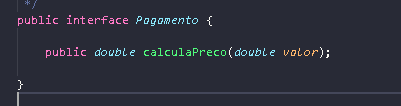
**Classe Produto**

Além do método construtor, possui o *toString* e as variáveis preço e produto.



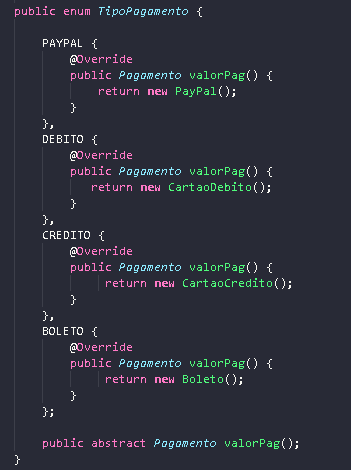
**Interface Pagamento**

Responsável por assegura que as classes CartaoDebito, CartaoCredito, *Paypal* e Boleto implementem o método *calculaPreco*.



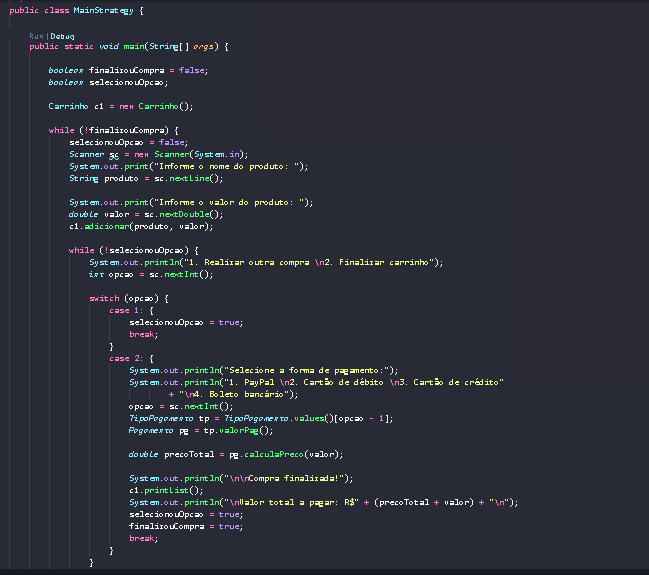
**Enum tipoPagamento**

Responsável por implementar os métodos próprios implementados pelo método abstrato Pagamento *valorPag()*.

****

**Classe *MainStrategy***

Responsável por instanciar as classes Carrinho para poder criar o objeto c1 e inserir produtos no carrinho chamando os métodos responsáveis para adicionar e apresentar os itens do carrinho ao usuário.

****

**Padrão *Facade***

O comportamento desse *facade* se resume nas regras de negócio do usuário poder realizar uma compra a partir do carrinho de compras do *app*, adicionar e remover compras do carrinho, visualizar saldo, carrinho de compras e ver total do carrinho de compras.

**Main**

Essa classe tem por função receber o input do usuário e passar ao *facade* o que ele precisa executar para cada input:

Text

Description automatically generated

**Model**

A *model* possui duas classes: o Item e a loja (*Store*). São formas bem simplificadas da ideia que representam.

Text

Description automatically generated

Text

Description automatically generated

**Facade**

O facade é a classe responsável então por intermediar o input do usuário com as regras de negócio e requisitos funcionais da aplicação. Aqui se controla o input e a aplicação responde de acordo com o esperado. A classe será exibido em texto aqui dado o seu tamanho.

public class Facade {

private InAppPurchaseFacade inAppPurchaseFacade;

private CommerceManagerFacade commerceManagerFacade;

public Facade() {

inAppPurchaseFacade = new InAppPurchaseFacade();

commerceManagerFacade = new CommerceManagerFacade();

}

public void purchase() {

if (commerceManagerFacade.getShoppingCart().isEmpty()) {

System.out.println("Shopping cart is empty!");

System.out.println("Please, add something to cart before buying!\n");

return;

}

if (inAppPurchaseFacade.purchase(commerceManagerFacade.shoppingCartTotal())) {

System.out.println("Purchased all items with success!");

System.out.println("Printing purchased items: ");

commerceManagerFacade.getShoppingCart().forEach(a -> System.out.println(a.name));

commerceManagerFacade.clearCart();

} else {

System.out.println("Insufficient money!");

}

}

public void addItem() {

commerceManagerFacade.addItemToCart();

}

public void removeItem() {

commerceManagerFacade.removeItemFromCart();

}

public void lookShoppingCart() {

if (commerceManagerFacade.getShoppingCart().isEmpty()) {

System.out.println("Shopping cart is empty, please add a item first!");

return;

}

System.out.println("Your shopping cart:");

commerceManagerFacade.getShoppingCart().forEach(System.out::println);

System.out.println();

}

public void lookTotal() {

if (commerceManagerFacade.getShoppingCart().isEmpty()) {

System.out.println("Shopping cart is empty, please add an item first!\n");

return;

}

System.out.println("Current total: U$ " + commerceManagerFacade.shoppingCartTotal() + "\n");

}

public void checkBalance() {

System.out.println("Your current balance is of U$ " + inAppPurchaseFacade.getBalance());

}

}

**CommerceManageFacade**

Essa classe é responsável por manejar os dados das lojas e o carrinho de compra do usuário. Ações como adicionar, remover, esvaziar o carrinho de compras, ver total do carrinho, bem como funções auxiliares da classe (input do usuário para selecionar loja ou item) estão descritas nessa classe.

Text

Description automatically generated

**InAppPurchaseFacade**

Essa classe é a responsável por manejar o dinheiro que o cliente depositou dentro da aplicação. Aqui se controla compras e recargas do usuário quanto ao seu dinheiro.

Text

Description automatically generated