Informações relase 3

Para a release 3, os mesmos padrões seguiram, precisamos adicionar mais um relacionamento de herança entre as classes Produto e Compra, no qual Compra herda os atributos de Produto, e foi necessário adicionar mais dois atributos para essa entrega: "comentario" e "avaliacao".

Com isso, necessitamos fazer uma mudança no tipo de "historicoDeCompras", presente na classe comprador, para que todo o histórico fosse realmente do tipo Compra, e assim fosse possível editar cada compra, passando um comentário e uma avaliação para a compra destinada.

É interessante notar que adicionamos também a pontuação do usuário e baseado nisso ele conseguia três tipos diferentes de benefícios, entre eles:

- Pontuação maior ou igual a 2 = Frete grátis;
- Pontuação maior ou igual a 4 = Frete grátis + Embalagem de presente grátis;
- Pontuação maior ou igual a 6 = Frete grátis + Embalagem de presente grátis + Sedex grátis;

Também precisamos citar a fórmula que escolhemos para avaliar as lojas, basicamente seguimos o cálculo:

- *avaliação atual + nova avaliação recebida / número total de avaliações recebidas*

Com isso conseguimos finalizar nossa entrega de tudo que foi pedido.

Informações relase 2

Para a release 2, decidimos manter os mesmos padrões utilizados anteriormente, pois percebemos que as informações relacionadas ao carrinho de compras, poderiam muito bem ficar acopladas ao escopo do comprador.

Dessa forma, o comprador poderia acessar seu carrinho de compras além do seu histórico de compras.

Dentro das curiosidades dessa entrega, preferimos definir que o carrinho de compras fosse algo que o usuário utilizasse durante tempo de execução, e caso o sistema fosse reiniciado, um carrinho zerado fosse aberto, porém, se o usuário efetuasse a finalização das compras do carrinho, ele teria um histórico de compras persistido.

Sobre essa entrega, vale ressaltar que criamos o mecanismo para que o usuário sempre compre tudo que tem no carrinho de compras. Pois o mesmo tem a opção de retirar produtos que já tinha colocado no carrinho de compras previamente. Dessa forma, sempre que ele for finalizar as compras, é checado se aquele produto ainda existe em estoque.

Patterns utilizados

Relacionado aos padrões utilizados, todas as informações abaixo continuam iguais as entregas.

Herança

Utilizamos do padrão de herança, pela facilidade de conseguir e o propósito que tínhamos ao extender uma classe no nosso projeto, analisamos que nesse exemplo em que aplicamos esse padrão, o objeto era um “tipo especial de”, e que ele não precisaria fazer override ou até anular variáveis ou métodos da classe que seria tida como pai.

Outro ponto que levamos em conta é o fato de que novas implementações nas classes que herdam da classe pai, sempre implementarão apenas métodos e variáveis específicas para seu contexto criando assim funcionalidades que só dizem respeito aquele escopo trabalhado.

O exemplo no qual utilizamos de herança foi na classe de nome Controller, além da classe Compra que extende de Produto, abaixamos damos o exemplo da classe controller e os métodos que ela utiliza, são eles:

Controller:

- insertData(Object obj):Insere um objeto no arquivo JSON correspondente com base no tipo do objeto (Loja, Comprador ou Produto).

- readData(String type):Lê os dados do arquivo JSON correspondente ao tipo especificado (Loja, Comprador ou Produto) e retorna uma lista de objetos.

- deleteData(int idToRemove, String type): Remove um objeto do arquivo JSON correspondente ao tipo especificado.

- updateData(String type, int idToUpdate, String newName):Atualiza o nome de um objeto no arquivo JSON correspondente ao tipo especificado.

Todas as classes que tem o nome Controller como prefixo, irão estender essa classe pai.

Fachade

Entendemos que utilizar de Fachade ajudaria muito o processo de desenvolvimento do nosso código, em vista que a nossa classe MarkteplaceFachade, teria uma visão completa de todas as classes, e apenas exportaria isso para nossa função main, que em tese seria o acesso para as classes e métodos existentes no nosso sistema.

Com isso simplificamos a interface do sistema, dando ao usuário uma interface única e simplificada que acesso todos os serviços do sistema.

Em tese, com a utilização desse padrão, temos a fachada recebendo requisições e solicitações, nas quais ela própria retorna novas interfaces para o usuário utilizar, e a medida que o usuário vai interagindo com essas interfaces, os controllers são chamados através da mesma fachada, e executando comandos e persistências de dados para que o usuário consiga utilizar do sistema.