

## Informações da release 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 à primeira entrega.

### Herança

Utilizamos do padrão de herança, pela facilidade de conseguir e o propósito que tínhamos ao estender 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 herdaram da classe pai, sempre implementaram apenas métodos e variáveis específicas para seu contexto criando assim funcionalidades que só dizem respeito àquele escopo trabalhado.

O exemplo no qual utilizamos de herança foi na classe de nome Controller, onde ele define 4 tipos de métodos a serem utilizados, 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.