Projeto “Lista Pra Mim”
Relatório de construção

Abel Antunes
Hércules Rodrigues
José Adrião
Ramon Sarmento

03 de Agosto de 2018

Design geral:

O design do projeto foi selecionado para permitir uma melhor coesão entre as diferentes partes do sistema, a partir de camadas de abstração que reduzem o acoplamento, como por exemplo no caso de ItemComprável(Interface), e item e suas variantes ou filhas se preferir(ItemQlo, ItemUnidade, ItemQtd). Quanto aos controladores (controllers), fizemos uso de quatro, pois assim evitaríamos a sobrecarga de uma classe, sendo assim mais fácil de fazer qualquer alteração necessária no projeto, pois uma só classe não seria tão impactante. Também fizemos uso de fachada para dividir de forma mais eficiente as camadas pois, oculta de forma eficiente todos os detalhes de implementação de uma camada para outra.

As classes foram encapsuladas em pacotes filtrados, para melhor entendimento e rápido acesso, com isso foi dividido em “controllers”, testes, classes, interfaces e fachada. Nos próximos tópicos entraremos em detalhes parte a parte.

Caso 1:

O caso 1 define que deve ser possível criar, pesquisar, atualizar e deletar itens, os itens podiam ser do tipo com quantidade fixa (ItemQtd), não industrializado por quilo (ItemQuilo) e por unidade (ItemUnidade), diferentes tipos de itens, porém, com algumas semelhanças, com isso foi decidido montar uma herança a partir da Classe geral (Item), e a partir dela fazer abstrações que nos permitisse chegar em cada uma das especificações para cada item, as entidades foram estanciadas pelo “controller” de item (ControllerItem), e armazenadas em um mapa com cada item tendo seu Id (identificador único) como chave de acesso. Recorremos a métodos “setters” para atualizar atributos dos produtos quando necessário, exceto aqueles imutáveis como o id. Os itens podiam ser registrados em diversos supermercados, e sua

pesquisa era feita através do id, nos próximos tópicos veremos que a pesquisa terá mais possibilidades.

Caso 2:

O caso 2 é focado nas funcionalidades para pesquisa do item, primeiro permitir a listagem de todos os itens cadastrados no sistema em ordem alfabética e por sua representação textual (String), para isto fizemos um método chamado “getItem”, que armazena todo os valores do mapa em uma lista e em seguida é usado o “Collections.sort” que deixa a lista ordenada em ordem alfabética, por último retorna um item de acordo com a posição passada. Em seguida deve ser possível a listagem dos produtos a partir da categoria e ainda em ordem alfabética, para isso criamos o método “getItemPorCategoria”, nesse método verificamos a todos os itens e suas categorias, selecionamos o desejado e em seguida ordenamos. A próxima listagem devia seguir parâmetros de preço, sendo do menor para o maior, para este caso fizemos uso de um método chamado “getItemPorMenorPreco”, para este método foi necessário a criação de uma interface que faz a comparação dos preços através de um “Comparator”. E para o último caso era preciso listar o item a partir de uma determinada “string”, e para isso fizemos o método “getItemPorPesquisa”, que utiliza do nome para pesquisar os produtos.

Caso 3:

No caso 3 é introduzido uma nova utilidade ao programa, a lista de compras. Com isto é criada uma classe para a lista de compras, e as listas são armazenadas em um mapa e têm como chave a descrição da mesma, pois, nos casos a frente será importante recuperá-las. As listas também possuem como atributo a data e hora a qual foram feitas e uma lista de itens a serem comprados especificando a quantidade, e com isso se fez necessário a criação da classe “Produto lista”, que serão os produtos a serem adicionados à lista. Depois de feita a única coisa que pode ser alterada na lista é a quantidade de produtos, caso o contador o item chegue à zero de modificado ele é removido da lista isso é feito a partir de um esquema de condições e caso chegue a zerar, o método “remove” é chamado e o item é removido da lista. Itens duplicados não podem ser adicionados na lista e para isso usamos de exceções não checadas que lançam mensagens de erro caso venha a ocorrer. A lista tem dois ordenamentos, o primeiro é por categoria e é feito a partir da interface “OrdenaItemLista”, e dentro desse ordenamento vem o segundo que é feito por ordem alfabética, este foi previamente ordenado a partir do atributo “saída ordenada” que por sua vez é uma lista com as palavras já ordenadas. Por último a lista é finalizada e seu preço final e supermercado onde a compra foi feita são introduzidos, e essas

funções é feita pelo método “finalizarLista”, nos tópicos a frente será introduzido como deverá ser feita a pesquisa de listas como citado no começo desse método.

Caso 4:

No caso 4 será introduzido e especificado a pesquisa de lista de compras, o caso base é usando a própria chave que guarda a lista no mapa, no caso, o descritor da lista, como chaves não podem ser repetidas, apenas uma lista é retornada. No segundo caso as listas podem ser pesquisadas pela data em que foram criadas, sendo assim podendo retorna mais de uma lista, com isso é necessário informar também a posição da lista, para isso aplicamos o método chamado “getListaPorData”, que percorre todas as listas selecionando as com datas iguais, e em seguida selecionando a lista de acordo com a posição escolhida. O terceiro caso permite selecionar listas a partir de um produto desejado, que será identificado pelo seu Id, a pesquisa retornará todos os descritores suas respectivas datas que será critério de ordenação, isto é feito pelo método “buscaListaPorItem”, o qual gera exceções para inconstâncias do tipo o item não existir em nenhuma lista ou caso a compra não seja encontrada na lista, e percorre as listas verificando se possui o produto desejado, se achar alguma, essa lista de compras é adicionada a uma segunda lista que irá armazenar aquelas que possuem o produto.

Caso 5:

No quinto caso é habilitado o modo de gerar listas automáticas, porém com diversas estratégias, a primeira apenas repete a última lista criada pelo método “geraAutomaticaUltimaLista”, nele a estratégia utilizada é verificar a data da lista criada, e para efeitos de desempate a hora também é verificada já que listas podem ser criadas na mesma data, no descritor da lista gerada automaticamente será especificado que a lista é do tipo “automática1”. No segundo caso igualmente ao primeiro, também será retornada a última lista feita, porém, a última lista feita com um produto específico, a estratégia é implementada pelo método “geraAutomaticaItem”, nele é percorrido as listas e seus produtos até achar uma com o produto desejado, diferente das outras, essa lista não inicia com um preço final e não é finalizada, essa lista é do tipo “automática2” tal informação estará em seu descritor, caso nenhuma lista possua o item, é lançada uma exceção que informa que não existem compras de determinado item. A última estratégia a lista é gerada a partir do produto que mais se repete nas listas anteriores, semelhante a segunda estratégia, porém nesta o item também é gerado automaticamente. Para esta estratégia foi feito o método ”geraAutomaticaItensMaisPresentes”, nesse método é verificado a quantidade de vezes que o item se repete nas listas e também do

quanto ele se repete em uma única lista, para isto foi feito um laço percorrendo as listas e verificando a quantidade de vezes que os itens se repetiam em uma determinada lista, esta lista inicia sem preço final e também não finaliza, em seu descritor será explícito a informação que ela é do tipo “automática3”.

Caso 6:

Neste caso é disponibilizado a função de encontrar em qual supermercado seria mais viável de acordo com uma determinada lista de compras, para isso foi necessário uso de todos os “controllers”, o “controllerGeral” recebe o identificador da lista e em seguida passa a informação para o “controllerLista” que por sua vez gera um mapa com os Ids e com as quantidades de cada item presente, em seguida a informação volta para o “controllerGeral” que só o repassa para preservar a coesão, com isso chega ao “controllerSupermercado” e então é verificado quais supermercados cadastrados possuem os itens. Para ordenar os supermercados pelo valor das compras é feita uma ordenação no próprio controller de supermercados, mas, para ordenar os itens de cada um desses supermercados, o controller geral recebe do controller de supermercados os itens de cada supermercado que estão na lista e o controller de listas faz a devida ordenação. Então foram utilizados polimorfismos de sobrecarga que possuem os nomes de “sugereMelhorEstabelecimento” que armazena todos os mercados que possuem pelo menos um item da lista, e os ordena por ordem de preço, sugerindo assim o melhor supermercado para determinadas compras sejam feitas.

Caso 7:

O último caso é onde se assegura a real utilidade do projeto, caso o programa seja fechado, ele armazena dados que mantém o seu estado, e se voltado a usar permitir utilizar as informações anteriores para gerar listas, pesquisar itens, entre outras funcionalidades. Isso é feito a partir da serialização de objetos que implementam a intenface Serializable. São criados arquivos binários que o java é capaz de interpretar e quando o sistema é inicializado, os dados são lidos e recarregados no programa.

Considerações finais:

Apesar do pouco tempo para se finalizar o projeto, optamos pelas opções que traziam mais harmonia e coesão para o “lista pra mim”, os erros inesperados em alguns casos de teste nos pegaram de surpresa muitas vezes, mas mesmo assim abriram espaço para um pouco mais de aprendizado. Em suma, tudo correu bem no final e o projeto foi finalizado no tempo previsto.

Link para o repositório no GitHub:
https://github.com/ramonssarmento/Projeto-Lp2