Relatório de Projeto LP2

Design Geral:

O design geral foi escolhido de forma a conseguir baixo acoplamento entre as classes, alta coesão do sistema, responsabilidades bem definidas para cada entidade, etc. Como ditam os padrões GRASP. Isso é feito através da criação de dois controllers distintos que lidam com diferentes tipos de tarefas e entidades que, por sua vez, também possuem formas de gerenciamento próprias, conseguindo assim abstração para o sistema.
Usamos um enumerator para quando o atributo é imutável e tem influência na catalogação dos objetos no sistema.
Temos identidades de um mesmo tipo que podem comportar-se de maneiras distintas, então adotamos o padrão strategy. Quanto a exceções, foi utilizada uma hierarquia de acordo com o tipo de erro, que pode acontecer em casos de atributos inválidos, em situações como criação ou atualização de objetos.
Segue abaixo as implementações de cada caso em mais detalhes.

Caso 1:

Neste caso é pedido que se crie uma entidade que representa um item comprável. Os itens podem ser comprados com base em seu peso (Item Por Quilo), em uma quantidade fixa (Item Por Quantidade) ou por unidade (Item Por Unidade). Para representá-los criamos uma classe abstrata Item que contém todos os comportamentos em comum entre os diferentes itens, além de um Id numérico, gerado automaticamente. Os diferentes tipos de itens herdam desta classe, sendo necessário apenas que cada subclasse mantenha atributos e métodos próprios a ela, conseguindo assim que seja necessária pouca reescrita de código.
O gerenciamento dos itens é feito por uma classe chamada Controller Itens, esta classe mantém um mapa que atribui uma chave única (ID do item) a um Item. O Controller pode criar, pesquisar, atualizar e excluir um item do sistema.

Caso 2:

Aqui são adicionadas algumas funcionalidades de pesquisa, usando critérios distintos. Para isso foi usado um Comparator (ComparatorPreço) que compara os preços dos itens, já a comparação da representação dos itens usando ordem lexicográfica que é case sensitive é feita implementando a interface Comparable e sobrescrevendo o método compareTo() da classe String.
Há a possibilidade de ordenar os itens por ordem alfabética, listando todos os itens cadastrados ou apenas de uma categoria determinada pelo usuário. É possível ordená-los em ordem crescente de preço para listagem, também é possível informar todos os itens que contém uma certa palavra e é case insensitive, para isto é usado um outro Comparator, que desconsidera maiúsculas e minúsculas.
São lançadas exceções quando são passados atributos inválidos, como uma categoria não existente ou nula, por exemplo.

Caso 3:

Para o caso 3 criamos três novas entidades chamadas ControllerListaCompras, ListaDeCompras e Compra.
O ControllerListaCompras lida com todas as tarefas de criação, atualização, pesquisa e finalização de listas de compras. Usa um mapa que associa uma descrição única a uma Lista De Compras
A entidade ListaDeCompras representa uma lista, que contém uma coleção de compras, novamente usamos um mapa, associando o ID de um Item a uma Compra. Também armazena informação sobre sua data de criação.
Por fim, uma Compra mantém informação da quantidade de um Item na lista e é associada a um Item que foi previamente cadastrado no sistema.
Exceções são lançadas quando atributos inválidos são passados. EX.: Na tentativa de criar uma compra com um item que não existe no sistema, ou de adicionar listas/compras que já existem no sistema.
As listas de compras são normalmente ordenadas por categoria (ComparatorCategoria), e em segundo plano por ordem alfabética (ComparatorDescrLista).

Caso 4:

Neste caso não foram criadas entidades adicionais, pois, como no caso 2, trata-se de pesquisa, porém de listas. Há, no entanto, adição do Comparator necessário à ordenação.
Seguindo a ordenação do caso 3, é possível achar uma lista usando seu descritor. Também é possível recuperar usando uma data de criação(ComparatorData), onde é feita uma listagem de todas as listas criadas na mesma data, em ordem lexicográfica. E usando o código (ID) de um item, onde deve ser feita uma listagem de todas as listas que contém uma compra associada ao item, em ordem crescente de data de criação e, em segundo plano, em ordem lexicográfica.

Caso 5:

Como no caso anterior, há apenas criação de Comparators necessários à ordenação.
É pedido que sejam implementados métodos de geração automática de listas de compras.
Primeiro é gerada uma lista automática idêntica à última lista criada pelo usuário.
A segunda estratégia usa um item específico para encontrar e retornar a última lista criada que contém este item.
A terceira estratégia pede que seja verificado se um item aparece em pelo menos metade das listas de compras anteriores para que sejam colocados na lista automática.

Caso 6:
O caso 6 deve implementar condições para sugestão de um melhor estabelecimento de compras, para isso é necessário comparar os preços dos itens de uma lista em todos os estabelecimentos em que ele pode ser encontrado para que seja obtido o menor preço parcial final para uma lista que ajude o usuário a decidir qual o melhor lugar para realizar suas compras.
Lança exceção apenas se não houver nenhum preço cadastrado na lista de compras.

Caso 7:

Cria condições para que o sistema não perca todos os dados ao ser fechado, aqui foi usado persistência para armazenar todos os dados necessários ao funcionamento do sistema em um arquivo de texto chamado “data.txt” que é criado usando as classes do pacote java.io que possibilitam leitura e escrita de informações em arquivos.

Por fim, temos uma classe Facade que recebe as requisições do usuário e delega as operações para seu controller específico. O uso da Facade aumenta o nível de abstração do sistema, pois esconde do usuário/interface gráfica como é feita a lógica da aplicação e permite o encapsulamento das informações internas que não precisam ser expostas ao usuário.

Colaboração dos membros do grupo em cada caso:

Caso 1:
Implementação: Igor Franca, Victor Braga
Testes de Unidade: Rostanth Santana Silva, Igor Franca
Exceções: Cleciana Santana

Caso 2:
Implementação: Igor Franca, Victor Braga, Rostanth Santana Silva
Exceções: Cleciana Santana

Caso 3:
Implementação: Victor Braga, Cleciana Santana
Testes de Unidade: Cleciana Santana, Igor Franca
Exceções: Cleciana Santana

Caso 4:
Implementação: Cleciana Santana, Rostanth Santana Silva
Exceções: Cleciana Santana

Caso 5:
Implementação: Rostanth Santana
Exceções: Rostanth Santana Silva

Caso 6:
Implementação: Igor Franca
Exceções: Igor Franca

Caso 7:
Implementação: Cleciana Santana
Exceções: Cleciana Santana

Documentação de código (javadoc):

Cleciana Santana, Igor Franca

Link para o repositório no GitHub:

https://github.com/FrancaIgor/ProjetoLp2