Relatório de Projeto LP2 XX

Design geral:

No nosso projeto, optamos por um design que proporcionasse uma melhor visualização, através de uma divisão de classes, de modo que cada classe tivesse funções específicas, seguindo o padrão GRASP. A utilização de um controlador geral (CrudUsuário) é um artifício que foi utilizado para diminuir o acoplamento no sistema. Ademais, tendo em vista que os itens estão relacionados a um Usuário, sendo o CrudUsuário responsável pela manipulação de tal classe, o CrudUsuário também detém indiretamente os comandos referentes aos itens, "terceirizando-os" para um gestor de itens (GestorItem). Dessa forma, empacotando melhor cada grupo lógico de código relacionado à certa funcionalidade, ainda assim, não diminuindo a coesão da classe principal.

Para a criação dos objetos, tentou se seguir o padrão CREATOR, de modo a atender os princípios em que a classe criadora detém e utiliza de forma recorrente o objeto criado, proporcionando dessa forma uma maior coesão à própria classe.

Em relação às exceções, foi criada uma classe única para a validação de todos os processos requeridos, permitindo uma melhor coesão para o tratamento das mesmas. Além, de é claro, uma melhor visualização acerca das limitações e proibições do código, voltando a análise de praticamente todas as exceções para um único local.

Caso 1:

No primeiro caso de uso, o sistema deve permitir a criação, leitura, atualização e remoção de usuários doadores e receptores, e é necessário informar o nome, e-mail, celular, classe e um documento de identificação. Sendo assim, para atender a essa solicitação, optamos por criar o CrudUsuario responsável por manipular os mesmos através da classe abstrata Usuário, que permite a existência de mais de um tipo de usuário.

Caso 2:

No segundo caso de uso, o sistema deve permitir a criação, leitura, atualização e remoção de itens para doação, sendo necessário informar o doador/receptor, descrição do item, quantidade, tags e o id. Nesse caso, resolvemos criar o gestor de itens (GestorItem), que possibilita criar e manipular tanto os itens necessários quanto os itens para doação. Desse modo, centrando a lógica do sistema de doação.

Caso 3:

O terceiro caso de uso pede que sejam implementadas 3 funcionalidades: permitir uma listagem de todos os descritores de itens cadastrados no sistema, ordenada em ordem alfabética pela descrição do item; permitir uma listagem de todos os itens inseridos no sistema, ordenando-os pela sua quantidade; o sistema também deve listar todos os itens relacionados a uma dada string de pesquisa. Posto isso, foi desenvolvido um mapa de descritores, responsável por guardar as descrições dos itens e suas quantidades, podendo ser listado em ordem alfabética através de métodos de formatação de saída. Além disso, foram-se implementados métodos de busca e listagem de itens, os quais, foram devidamente associados a um Comparator específico, criado para cada tipo de ordenação requerida.

Caso 4:

O quarto caso requisita que seja possível cadastrar, atualizar, listar e remover itens necessários ao sistema. Desse modo, assemelhando-se ao processo citado no 2ª caso. Dessa forma, a lógica relacionada a essa US também foi direcionada para o gestor de itens (GestorItem), o qual reaproveita diversas funcionalidades para ambos os tipos de itens, tornando-o, assim, expert na gestão dos tais.

Caso 5:

O quinto caso de uso pede para que seja implementada uma funcionalidade que permita encontrar casamentos (matches) entre itens a serem doados e itens necessários. Para desenvolver tal funcionalidade, foi feita uma classe responsável por identificar esses casamentos (MatchMaker), a qual possui métodos de seleção dos itens que contém a mesma descrição do item solicitado, e métodos responsáveis por calcular a paridade entre os itens. Dessa maneira, encontrando e organizando os itens para doação e disponibilizando-os da melhor forma.

Caso 6:

O sexto caso é referente ao sistema de realização de doações, verificando se a igualdade entre os itens é válida e posteriormente atualizando a quantidade dos itens a serem doados e dos itens necessários envolvidos nesta transação. Para isso, foi implementado um método realizaDoacao, o qual verifica a validade dos itens e da data especificada, atualizando as informações dos itens (removendo-os caso sua quantidade chegue a 0) e colocando um comprovante da doação em uma lista de doações, a qual armazena todo o histórico de concessões ordenadas pela data. Desse modo, podendo listar todas as doações feitas e verificar o histórico de itens existentes durante o processo.

Caso 7:

O sétimo caso diz que o eDoe.com deve armazenar em disco todos os dados necessários para manter o seu estado mesmo que o programa seja fechado. Medida possível por meio da classe Persistência, a qual apresenta métodos de leitura e escrita de arquivos, permitindo assim, junto à implementação da interface Serializable, o armazenamento em uma espécie de banco de dados. Desse modo, permitido a permanência das informações do código em um arquivo .dat, mesmo depois de sua finalização.

Link para o repositório no GitHub:
https://github.com/cilasmarques/eDoe-2018.2-LP2