Relatório de Projeto LP2 2018.2

Grupo: Caio Fernandes Moreira
Klaywert Danillo Ferreira de Souza
Mathias Abreu Trajano

Design geral do projeto:

A escolha do design geral do projeto ocorreu com base nos princípios de organização propostos tanto nas aulas de Programação II, quanto nas aulas de Laboratório de Programação II, que juntas trabalham o conceito e aplicação de programação orientada a objeto. O design geral escolhido traz uma representação simples e eficaz das funcionalidades exigidas pelo sistema. Para fins de organização, foram utilizados diferentes níveis de entidades que controlam, direcionam, criam e armazenam os diferentes objetos do sistema. Com isso, o projeto apresenta um nível alto de coesão e baixo acoplamento (dois importantes princípios GRASP). Cada camada do sistema, por menores que sejam, possuem objetos e entidades que o controlam e realizam todas as funções possíveis que são cabíveis.

Para a organização do projeto, o design geral escolhido prezou pela separação do sistema em níveis diferentes que permitem uma melhor compreensão do sistema desde a primeira vez que é visto por uma pessoa que não conheça o projeto. Essa separação ocorreu por meio da criação de diferentes packages que possuem classes que possuem certa semelhança no sistema.

Além disso, contamos também com a implementação de diferentes exceções em diferentes casos considerados indesejados, porém já esperados. No sistema, existem exceções para: erro na escrita, erro na leitura, argumento inválido, item inválido, usuário inválido e id negativo.

Como o projeto foi dividido em casos de uso (7 para ser mais específico), explicaremos melhor nossas escolhas e implementações separadamente em cada caso de uso a seguir:

CASO DE USO 1:

No caso de uso 1, foi pedido o CRUD (Create, read, update, delete) de usuário doadores e receptores. Para diferenciação dos dois tipos de usuário, nosso design geral prezou por evitar criar um sistema de herança, substituindo por apenas um parâmetro que diferencia os dois tipos de usuário (“receptor” para receptor e “doador” para doador), fazendo com que o sistema se torne mais simples e se livre de maiores complicações desnecessárias.

Para armazenar os diferentes usuários, foi criado um controller, que se comunica diretamente com a facade. O controller em questão (ControllerEDOE), é responsável pelo CRUD de usuário. Lembrando que os usuários receptores são adicionados através da leitura de um arquivo.

CASO DE USO 2:

No caso de uso 2, o CRUD de itens a serem doados foi pedido. Para realização de tal função, todos os métodos necessários foram adicionados no controllerEdoe, já armazenando no hashMap de usuários que foi implementado e melhor explicado no caso de uso 1. A partir desse ponto do sistema, a criação de uma exceção já se fez necessária segundo nossa escolha, por isso InvalidItem foi criado para evitar alguma ação para com um item inexistente no sistema; e InvalidArgument foi criado para evitar que algum parâmetro passado fosse vazio ou nulo.

CASO DE USO 3:

O caso de uso 3 permitiu a pesquisa de itens a partir de diferentes parâmetros: ordem alfabética pelo descritor, ordenação pela quantidade e ordenação por meio de uma string de pesquisa. Essa ordenação por meio de uma string de pesquisa desconsiderava maiúscula e minúscula e achava itens com descritores que possuíam a string de pesquisa. Para isso, a classe dos comparables já foram sendo implementadas, ganhando importante função nesse caso de uso.

CASO DE USO 4:

O CRUD de itens necessários foi o principal foco do caso de uso 4. Itens necessários eram os itens que os receptores estivessem precisando receber, esses itens eram de extrema importância principalmente para a posterior realização do sistema de doações implementado no caso de uso 6.

CASO DE USO 5:

Talvez o caso de uso mais complexo e trabalhoso do sistema, o caso de uso 5 pedia a identificação de possíveis matches entre itens necessários e itens a serem doados. Essas demandas exigiram a criação de uma classe match que relaciona dois itens. Entre esses dois itens é criado um coeficiente de match que diz o quanto os dois itens são iguais e apresentam características e especificações semelhantes.

CASO DE USO 6:

No caso de uso 6, finalmente, as doações puderam ocorrer. Já de conhecimento dos casamentos (matches) do sistema, os métodos de realizar doações analisavam a demanda e a necessidade, logo após realizava a doação. Essas doações ocorridas eram mantidas em um histórico de doações, tornando necessária a implementação da classe doação, que recebia todos os parâmetros e criava uma nova doação na qual o toString representava a representação textual de como a doação era salva no histórico de doações.

CASO DE USO 7:

Por fim, o sistema, no caso de uso 7, exigia a função de armazenamento em disco de todos os dados necessários para manter o seu estado mesmo o programa sendo fechado. Como o nível de abstração era maior, achamos

necessário criar um controllerDAO que regula a leitura e escrita de todos os dados necessários do sistema.

Considerações finais:

Esse projeto foi desenvolvido de forma colaborativa, com todos os integrantes contribuindo para cada caso de uso. Durante o desenvolvimento, utilizamos o JUnit tentando alcançar 70% ou mais de cobertura para testar as classes envolvidas no design explicado anteriormente. Dessa forma, o projeto foi concluído e todos as funções básicas requisitadas foram atendidas conforme a especificação.

Link do repositório no GitHub:

https://github.com/MathiasAbreu/ProjectEdoe.com.git