UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
DSC - DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO
Componente Curricular: Laboratório de Programação 2
Professora: Lívia Sampaio

|Grupo 6|
Caroliny Regina Valença Leandro
Dayvid Daniel da Silva
Matheus Augusto Silva do Nascimento
Thiago Nascimento de Lima

RELATÓRIO DO PROJETO DE LP2 - 2018.2
eDoe.com

Design geral:

O Design geral do nosso projeto foi escolhido com o intuito de garantir uma maior independência entre as entidades do sistema, sendo intrínseco o surgimento de controladores próprios para cada entidade, assim como um controlador geral para promover a junção de tarefas e funcionamento do sistema.

Para a criação dos objetos, utilizamos o padrão Expert. O uso de Enumerators foi utilizado em situações onde o atributo era imutável, a fim de garantir que ele seguisse um certo padrão de valores e facilitar a catalogação dos mesmos no sistema. A criação de comparadores se deu pela necessidade, em alguns métodos, de retornar objetos de forma ordenada. Em relação às exceções, o surgimento de uma classe própria para o lançamento e tratamento das mesmas foi de extrema importância para o aumento da abstração do código.

As classes foram organizadas em pacotes de acordo com sua principal funcionalidade: se era uma entidade, um controlador, comparador ou se apenas realizava uma função distinta das anteriores no sistema.

Abaixo detalhamos mais sobre implementação de cada parte do projeto.

Caso 01:

O caso 1 pode ser criado uma entidade que representa diferentes tipos de Usuários no sistema. O Usuário pode ser Receptor ou Doador, ambas possui as mesmas funcionalidades.

Para representar esses diferentes tipos de Usuários optamos por uma classe abstrata Enum, que permite uma fixação dos status do usuário que não irá mudar evitando repetição de código, O ponto importante a ser citado é que eles não podem ser estendidos de nenhuma outra classe porque Enums estendem implicitamente a partir de java.lang.Enum.

Para armazenar e gerenciar os Usuários, foi criado uma classe chamada de ControladorUsuarios(). Nesta classe, há um coleção que armazena todos os Usuários. Essa coleção é um LinkedHashMap, onde a chave é o Id do Usuário, representado pelo seu

Documento que é único para cada usuário. Foi preferido utilizar o LinkedHashMap, pois ele armazena o Usuário por ordem de chegada, algo necessários para testes de saída.

Caso 02:

No caso de uso 2, o foco foi a inserção de itens a serem doados no sistema. De tal modo, foi necessária a criação de um Controlador de Item, onde os itens passaram a serem criados e armazenados em mapas, dado um certo descritor de identificação. Assim que criados através do método adicionaItem() e armazenados no mapa de itens doados, tendo como atributos um inteiro de identificação, uma string que identifica o doador, uma descrição, tags descritivas, quantidade do item a ser doada disponibilizada pelo doador e a data de inserção do item no sistema, os itens são enviado para o Controlador de Usuário para que os mesmos sejam associados aos seus respectivos doadores.

A classe de controle dos itens também foca na criação e inserção de descritores ainda não presentes no mapa de armazenamento de itens através do método adicionaDescritor(), na exibição de itens dado seu inteiro de identificação através do método exibeItem(), na atualização de atributos mutáveis (tags e quantidade) de um certo item identificado pelo seu id através do método atualizaItem() e na remoção de itens a serem doados através do método removeItem().

Caso 03:

No caso de uso 3, foram criados métodos com o objetivo de listar os itens inseridos no mapa de itens doados, dependendo de um referencial de listagem pedido na especificação do projeto.

O método listaDescritorDeItensParaDoacao() retorna uma string com todos os descritores de itens registrados no mapa e a quantidade de itens que cada um guarda, já ordenado por ordem alfabética pelo TreeMap. O método listaItensParaDoacao() retorna uma string com todos os itens disponíveis para doação inseridos no sistema e a identificação do seu doador (nome/documento), ordenados por ordem decrescente de quantidade de cada item disponível. Por fim, o método pesquisaItemParaDoacaoPorDescricao() retorna uma string contendo os objetos com a mesma descrição passada no parâmetro, ordenada novamente por ordem alfabética de acordo com os descritores de cada objeto.

Para esse caso de uso, foi necessária a criação de uma entidade comparadora, o ComparadorQuantidade, responsável por comparar as quantidades dos itens disponíveis para doação e retornar o com maior quantidade para fins de ordenação.

Caso 04:

No caso de uso 4, começamos a trabalhar com os itens necessários. Os métodos usados para cadastro, atualização e remoção são os mesmos usados para trabalhar com itens doados, o que ajuda na abstração e diminuição do tamanho do código. Apenas os métodos de listagem são diferentes para os dois tipos de itens. Tais itens também são armazenados no Controlador de Item como os itens doados.

Os itens necessários funcionam da mesma maneira que os itens doados, com diferença apenas no tipo de usuário associado, que aqui passam a ser do tipo receptor.

Caso 05:

No caso de uso 5, o que há de novidade é a função de match, que mostra o itens (com seus respectivos doadores/receptores) ordenados de forma decrescente pela sua pontuação. A pontuação é feita a partir dos descritores, da igualdade e ordem das tags. Caso dois itens tenham as mesmas pontuações, eles são reordenados a partir de seu id. Nesse processo, foi necessária a criação de uma entidade comparadora, o ComparadorId, que é responsável por comparar os ids dos itens e retornar o de menor valor.

Toda a parte que envolve os itens, sua pontuação e sua ordenação é feita no controlador de item que passa essas informações prontas para o controlador de usuários para completar os dados que faltam, estes que pertencem aos usuários.

Caso 06:

No caso de uso 6, duas novas funcionalidades são apresentadas. A primeira é a de realizar doação, em que uma doação é criada com todas as informações, tanto dos itens doados quanto dos usuários, sejam eles receptores ou doadores. Devido a sua função de armazenamento atribuída no caso 2, as doações de itens ficam sob a responsabilidade do controlador de Item.

É passado ao controlador de usuário somente as informações necessárias para saber se há necessidade de exclusão de item do sistema, o que ocorre quando os itens doados ou necessários chegam a 0 durante a doação, enquanto no controlador de item é feita a criação da doação, a atualização da quantidade dos itens e sua remoção caso necessária.

A outra funcionalidade se trata de listar todas as doações, tais ordenadas pelas datas. Caso existam datas iguais, estas devem ser reordenadas pela ordem lexicográfica do seu descritor.

Caso 07:

No caso de uso 7, pede que seja implementado uma estrutura que possibilite o armazenamento dos dados de execuções passadas. Para isso foi criada uma classe de nome Persistência associada à Facade. Na classe tem dois métodos, um referente à salvar e fazer os testes e outro à carregar os dados. A estratégia utilizada foi a de salvar a principal coleção do sistema, que é o mapa de usuário. Seus dados são salvos em um arquivo para que na hora de recuperar eles não percam as ligações de memória.

O método de salvar recebe o mapa e o salva no arquivo "dados.txt". No caso de o arquivo não ser encontrado, o programa é interrompido e a pilha de erros é mostrada. O método de carregar os dados verifica a existência do arquivo e verifica também se o mesmo tem dados gravados, para assim poder atualizar o mapa, que só é atualizado caso existam dados gravados no arquivo.

Considerações:

Durante o desenvolvimento do presente trabalho, tivemos que refatorar para aperfeiçoar a estrutura do código com intuito de evitar problemas futuros. Um dos desafios foi o armazenamento de usuários, onde os mesmos deveriam ser inseridos em ordem de adição. Para tal, optamos pelo uso do LinkedHashmap para solucionar o problema.

Optamos também pela união dos controladores de item e de usuário em um único controlador que os invoca e os coordena. Então, toda a ideia do sistema principal foi evitar uma dependência circular entre as classes, pela grande interdependência das mesmas.

Link para o repositório no GitHub:

https://github.com/dayvidds/projetolp2