Relatório de Projeto LP2

Design geral:

O design do projeto foi, basicamente, dividido entre as classes básicas como usuário e item, e seus controladores que irão delegar todos o métodos inerentes ao desenvolvimento do projeto. Falando de padrões GRASP, a alta coesão foi usada no projeto através da modularização em suas classes, garantindo assim, que cada classe tenha a responsabilidade única de operar sobre seus dados.

O padrão Controller foi usado para gerenciar as funcionalidades do CRUD tanto em usuário, quanto em Item; e para facilitar a manipulação com os descritores, foi criado o controlador de descritor que auxiliará na cadastro e na busca de descritores, criando assim, níveis que diminuem o acoplamento entre as classes.

Atraveś do padrão Creator, os objetos foram criados em seus controladores e alocados em alguma coleção para sua manipulação se tornar possível.

Com relação a exceções, temos-as para atributos inválidos, para objetos que o usuário gostaria de adicionar mas que já existem no sistema, para objetos não encontrados, entre outras situações..

Todas as classes estão encapsuladas em packages menores, que pertencem, eventualmente, a entidades, controllers e fachada.

Caso 1:

O caso 1 pede que seja criado uma entidade que representa usuários. Esses usuários podem se dividir doador e receptor. Para isso, optamos por não usar herança, pois o usuários, independente do tipo, doador ou receptor, compartilham exatamente dos mesmos atributos, o que difere um do outro é a String do status, “doador” para os usuários doadores e “receptor” para os usuários receptores.. O usuário, também, mantém um ID único. A classe Usuário implementa a interface Comparator<T> para auxiliar posteriormente na representação ordenada dos itens, além disso, a classe foi desenvolvida respeitando os padrões de encapsulamento.

Nesta classe, há uma coleção que armazena todos os itens. Essa coleção é um mapa, onde a chave é o código de identificação do item. Também há, nessa classe, uma coleção que armazena todos os itens, tendo como chave o id do item.. Débora foi a responsável pela construção do caso de uso 1.

Caso 2:

No caso de uso 2 pede que seja criado uma entidade que representa os itens a serem doados e que possa ser possível gerenciar esses itens. A entidade item implementa a interface Comparable<Item> que compara o descritor por ordem alfabética; essa, também, mantém uma lista de tags. Os métodos de gerenciamento dos itens é feito pelo controller de usuários. Gabriel foi o responsável pela construção do caso de uso 2.

Caso 3:

No caso de uso 3, pedia-se que o sistema pudesse listar os itens e os descritores cadastrados no sistema de formas diferentes.

Na primeira funcionalidade, todos os descritores de itens cadastrados no sistema devem ser ordenados em ordem alfabética pela descrição do item. Neste caso, implementamos o método listaDescritorDeItensParaDoacao() que faz uma instância de um map cuja chave é uma string que representa a descrição, e o valor é um Integer que representa a quantidade de itens. O método, mantém os descritores dos usuários organizados por ordem alfabética. A escolha do map foi feita levando em conta que não pode haver descritores repetidos.

A entidade Item implementa um Comparable<Item> para que seja possível a ordenação dos itens através de seu descritor.

Na segunda, o sistema deveria poder listar todos os itens inseridos no sistema ordenada pela quantidade do item no sistema. Caso os Itens possuam a mesma quantidade, estes devem ser ordenados pela ordem alfabética de descrição. Portanto, para fazer a ordenação foi criado uma classe chamada ItemComparavel que implementa a interface Comparator<Item>. A classe possui o método do contrato da interface que cria um variável para guardar o valor da subtração entre a quantidade de itens do objeto item2 e do objeto item1. Caso a variável possua um valor igual a zero, então esta passa a armazenar o valor do método compareTo, que compara o descritor do item1 com o descritor do item2. Por fim, é retornado o valor da variável.

Na última funcionalidade, o sistema deve ser capaz de listar todos os itens relacionados a uma dada string de pesquisa, que deve ser insensível a maiúscula/minúscula. Essa listagem deve ocorrer em ordem alfabética considerando os

descritores dos itens. Para tal, o método pesquisaItemParaDoacaoPorDescricao(String pesquisa) varre todos os itens do sistema e adiciona em um array que será ordenado e apresentado ao usuário através de manipulação com strings, conforme pede a especificação do sistema . Erick foi o responsável pela construção do caso de uso 3.

Caso 4:

Para o caso de uso 4, pede-se que seja possível gerenciar os itens necessários, que são os itens que os receptores podem indicar que estão precisando receber, então todo item necessário deve estar sempre associado a um usuário receptor. Para isso, optamos por não usar herança para diferenciar itens a serem doados e itens necessários, pois os dois tipos de item compartilham exatamente os mesmos atributos e quase todos os métodos, o que vai diferenciá-los é em que coleção estão alocados, nas dos usuários receptores ou doadores. Também foi criada outra classe Comparator, que ordena os itens necessarios pelo ID, que é um identificador único, ou seja, pela ordem em que foram cadastrados no sistema. Maico foi o responsável pela construção do caso de uso 4.

Caso 5:

Neste caso de uso, o sistema deve encontrar matches entre itens a serem doados e itens necessários. Apenas os itens para doação que tem o mesmo descritor que o item necessário especificado são selecionados e em seguida as tags são avaliadas. Além disso, há uma pontuação que define quão parecido o item do doador é com o item necessário. Para que não haja a possibilidade de ter itens iguais na coleção, foi decidido criar um map, que a chave é uma string que mantém o toString() do item, e como o valor, a pontuação marcada pelo item. A coleção de itens é ordenada da maior para a menor pontuação, com a construção de uma nova entidade ItemComparavelPorPontuacao que irá dar a base para que a coleção seja ordenada. Caso as pontuações sejam idênticas, então o sistema ordena pelo identificador dos itens. Diante disso, foi criada uma função para fazer o match que recebe o id do receptor e o id do item necessário, e então é realizada uma busca de todos os itens que possuem o mesmo descritor. Após isso, inicia-se uma avaliação das tags para definir a pontuação de cada item, para isso, foi criada uma função para realizar a tarefa que retorna os pontos associados a cada item.

No fim do método, uma string contendo o id de cada item, sua descrição, a tag, a quantidade, o nome do doador e seu documento é retornada. No caso de uso 5, todos começaram a codificar no laboratório. A parte que restou foi finalizada por Débora.

Caso 6:

No penúltimo caso de uso, foi criada uma função para realizar a doação, que recebe o id do item necessário, id do item doado e a data da doação. Então, é realizada a busca de ambos os itens por meio da função privada getItemPeloId(); caso o item não exista, é lançada uma exceção. O sistema verifica se os itens possuem o mesmo descritor. A partir disso, a quantidade de item para doação e item necessária é atualizada. Caso o item chegue a quantidade 0, ele é removido; entretanto não se deve remover o descritor do item.

É retornado para o controlador Edoe, uma string contendo os dados da doação realizada e ela é repassada para a entidade controllerDoacao. Em controllerDoacao existe a função adicionarDoacao(), q adiciona a doação no sistema, e possui a função listadoacao() que lista todas as doações no sistema. Débora codificou o caso de uso 6.

Caso 7:

No último caso de uso, foram utilizados dois métodos (iniciaSistema e finalizaSistema) que possuem a função de armazenar as informações inseridas no sistema em um arquivo .ser. Primeiramente, o método iniciaSistema invoca a classe FileInputStream e recebe o arquivo "objetos.ser" que está dentro da pasta "sistema_serializado". O objeto da classe ObjectInputStream recebe o arquivo em seu construtor e, posteriormente, usando a função readObject(), lê o objeto da classe ControllerEdoe serializado do arquivo "objetos.ser" e passa o objeto lido para o atributo da classe Fachada como mostra na seguinte linha de código:

controllerEdoe = (ControllerEdoe) ois.readObject();

Após isso, o arquivo é fechado.

Por outro lado, o método finalizaSistema faz uso da classe FileOutputStream para criar o arquivo "objetos.ser" e passa o arquivo no construtor do objeto da classe ObjectOutputStream. A partir disso, o objeto controllerEdoe é escrito dentro do arquivo como mostra na seguinte linha de código:

oos.writeObject(controllerEdoe);

Após isso, o arquivo é fechado.