UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
DSC - DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

Componente Curricular: Laboratório de Programação 2
Professor: Matheus Gaudencio

Cássio Eduardo Gabriel Cordeiro
Geovane do Nascimento Silva
Hemillainy Suellen Sousa Santos

RELATÓRIO DO PROJETO DE LP2
TT - Tracking Things

Design Geral

O design do projeto foi pensado para que obedecesse os padrões do GRASP. Inicialmente tem-se a Facade que faz a chamada de todos os métodos para as que as classes controladoras possam executar suas funcionalidades. Três Controllers foram implementados, cada um controla uma entidade básica (Usuário, Item e Empréstimo) e as classes que auxiliam no seu comportamento (Listagem, Validação, Enum). Esta estratégia de três controladores foi escolhida para que uma classe abaixo da Facade não ficasse extremamente inflada e complexa. Sendo assim nosso projeto segue três especificadas linhas de execução. Quando uma entidade alterava seu comportamento em tempo de execução usamos o padrão Strategy, quando alterava seu estado, Enum. Foi utilizado herança para classes que possuem os mesmos atributos porém comportamentos diferentes. Cada Controller possui uma classe de validação, que é onde ocorre a verificação e o possível lançamento da exceção.

Abaixo detalhamos mais sobre a implementação de cada parte do projeto.

Caso 1

O caso 1 pede que seja feito o CRUD dos usuários do sistema. Para isso, criamos a classe UserController, para atender as devidas funcionalidades, ela possui os metodos: cadastraUsuario, removeUsuario, atualizaUsuario e getInfoUsuario. Para fazer o armazenamento dos usuários cadastrados ela possui um mapa onde a chave desse mapa é um objeto do tipo IdUsuario – esse objeto possui os atributos nome e telefone, que são o nome e o telefone do usuário cadastrado.

Caso 2

O caso 2 pede que seja implementada a estrutura para o CRUD de itens emprestáveis. Os itens podem ser Jogo Eletrônico, Jogo de Tabuleiro ou Bluray, um Bluray por sua vez pode ser de Série, de Filme, ou de Show. Para representar esses diferentes tipos de itens, optamos por criar uma classe abstrata chamada Item que contém a maior parte dos estados e dos comportamentos de um Item. Essa estratégia também foi usada na implementação do tipo Bluray e suas subclasses. Cada tipo de Item tem um enum associado a si que armazena uma característica que diz respeito somente aquele tipo, e a classe Item tem um enum que é a identificação do estado de empréstimo do item.

Para gerenciar os Itens, foi criada uma classe chamada ItemController. Nesta classe, há uma coleção que armazena todos os Itens. Essa coleção é um mapa, onde a chave é a

String de identificação do nome do Item. Essa classe foi composta com um classe itemValidacao que recebe dados referentes a cadastro, busca e alterações de itens e verificada a validade do dado, caso os dados sejam considerados inválidos exceção de IllegalArgumentException sao lançadas com mensagens referentes ao erro.

Caso 3

O caso 3 pede que sejam implementadas três funcionalidades: 1) listagem de todos os itens cadastrados em ordem alfabética pelo nome do item, 2) listagem de todos os itens cadastrados em ordem de valor dos itens e 3) um usuário pode pesquisar informações de um item específico a partir do nome do item e do telefone e nome do dono do item.
Para a funcionalidade 1 implementamos o compareTo em Item (que compara dois itens a partir de seus respectivos nomes) e uma classe ItemListador, essa classe possui o método listaItensOrdenadosPorNome que recebe um ArrayList com os itens cadastrados no sistema, ordena esse ArrayList e retorna uma string com a listagem.
Para a funcionalidade 2 criamos a classe ComparadorValor que implementa o Comparator de Item (que compara dois itens a partir de seus respectivos valores). ItemListador possui o método listaItensOrdenadosPorValor que recebe um ArrayList com os itens cadastrados no sistema, ordena-o utilizando o Comparator de Item e retorna uma string com a listagem.
E para a funcionalidade 3, ItemController tem um método pesquisaDetalhesItens que retorna uma string com as informações do item desejado.

Caso 4

O caso 4 pede que seja criada uma entidade Emprestimo e que possa ser realizados empréstimos entre os usuários cadastrados no sistema. Para isso, além de criamos a entidade Emprestimo criamos também uma classe EmprestimoController.
Para registrar um empréstimo, ela possui o método registraEmprestimo, ele adiciona o emprestimo no mapa de empréstimos presente em EmprestimoController (a chave desse mapa é um objeto do tipo IdEmprestimo – esse objeto possui os atributos dono, requerente – ambos do tipo Usuário –, itemEmprestado – do tipo Item – e dataEmprestimo) e adiciona-o também no ArrayList de empréstimos do dono do item emprestado e do requerente.
Para devolver o item ela possui o método devolveItem, ao ser chamado ele seta o status do item – de "Emprestado" para "Não emprestado" –, adiciona a data de devolução no empréstimo e seta a reputação do requerente.

Caso 5

Para as especificações de listagem criamos três classes responsáveis por isso, uma para as listagens referentes à Item, uma para Usuario e uma para Emprestimo e o Controller responsável por cada entidade foi composto por essa classe. Quanto a necessidade de ordenar os objetos implementamos classes de comparação que implementam a interface Comparator e as classes básicas implementam Comparable, que representa a ordem natural dos objetos da classe. Nas classes de listagem há os métodos que listam cada especificação. Para atender cada especificação de listagem implementamos nas classes Controller e na classe referente métodos que possibilitam a classe Listador exercer sua função.

Caso 6

Neste caso um Usuario necessitou possuir uma reputação, para isso atribuímos a Usuario um double que representaria sua reputação e criamos um método que adicionava

reputação. Quando o Controller de Item chama o método de adicionar Item de Usuario sua reputação vai aumentar. Quando o Controller de Emprestimos registra um empréstimo ele chama o método de adicionar reputação do dono do Item e quando o requerente vai devolver o Controller de Emprestimo verifica se ele está devolvendo com atraso, caso sim ele calcula o desconto de reputação do requerente de acordo com as regras de devolução, e remove passando um valor negativo para o método que adiciona reputação, caso contrário ele adiciona reputação de acordo com as regras.

Caso 7

O caso 7 pede que seja implementada uma estrutura para que usuários passem a ter um cartão fidelidade que identifica uma categoria que o usuário se encaixa no Track Thing segundo a quantidade de itens emprestáveis que ele tem e a quantidade de empréstimo que ele executou. Para esa implementação o projeto não sofreu grandes mudanças. A classe Usuario que antes tinha um double representando a reputação passou a ter um atributo do tipo Cartão Fidelidade que guarda a reputação, o estado do cartão e os comportamentos dele. No CartaoFidelidade temos um Strategy para que o tipo de fidelidade seja alterado em tempo de execução. CartaoFidelidade tem um objeto Fidelidade que é abstrato de fidelidade e tem quatro subclasses(Noob, FreeRyder, BomAmigo e Caloteiro). A medida que um usuário executa ações que alteram sua reputação o objeto CartaoFidelidade executa os cálculos de acréscimo ou decréscimo de reputação e faz a troca de tipo criando um novo objeto Fidelidade que se inicia com a reputação do item anterior.

Caso 8

Para este caso foram feitas alterações quando se pega um empréstimo, foi adicionado uma validação no método pegar emprestado de User Controller, que se o requerente for caloteiro ele quebra. Para listar os caloteiros o listador de usuários recebe um ArrayList com os usuários e verifica os que são caloteiros. Para listar os 10 melhores usuários foi criada uma classe de comparação de usuários que compara pelo valor da reputação em ordem decrescente, o método de listar os top 10 recebe um ArrayList e o ordena baseado na reputação. Para listar os 10 piores usuários o método que faz essa listagem ordena o ArrayList que recebe como parâmetro de acordo com a reputação, usando o mesmo comparador de reputação de listar top 10, mas ele inverte a lista, fazendo com que ela fique em ordem crescente.

Caso 9

O caso 9 pede que seja implementada uma estrutura que possibilite o armazenamento dos dados de execuções passadas. Para isso criamos uma classe de nome Persistência associada a Facade. Na classe tem dois métodos, um referente a salvar os dados e um referente ao testes e outro a carregar os dados. A estratégia que utilizamos foi de salvar as três principais coleções do sistema, que são o mapa de usuários, o mapa de itens e o mapa de empréstimos. Os três são salvos em um mesmo arquivo para que não hora de recuperar eles não percam as ligações de memória. O método de salvar recebe os três mapas e os salva no arquivo “dados.txt”, no caso de o arquivo não ser encontrado o programa pára e mostra a pilha de erros. O método de carregar os dados verifica a existência do arquivo e verifica também se ele tem dados gravados, e os mapas só são atualizados caso existam dados gravados no arquivo de dados.

Considerações

Em dois momentos no nosso sistema nós fazemos um downcast pois enfrentamos um problema: dentre os itens presentes do sistema, adicionarBluRay é algo referente apenas a Serie e adicionaPecaPerdida apenas à Jogos de Tabuleiro. Então tínhamos três alternativas para implementar ambas as funcionalidades: 1) deixar ambos os métodos na superclasse, 2) alocar cada tipo de item em um mapa apenas para seu tipo e 3) alocar todos juntos e fazer um downcast na hora de pegar o item para adicionar a peça perdida ou adicionar o BluRay. Optamos pelo downcast pois nos pareceu ser a alternativa “menos pior”; porque a alternativa 1 permitia que todos os itens do sistema pudessem usar de tais funcionalidade e a 2 aumentava muito a complexidade do sistema, pois toda vez que fosse ser feita alguma operação sobre um item teria de ser feita uma busca por aquele item em todos os mapas dos diversos tipo.

Link para o repositório no GitHub

https://github.com/hemillainysantos/Projeto_LP2