Relatório de Projeto LP2 - Track Things

Integrantes :
Caio Sanches Batista de Lira, Igor Araújo Tavares de Farias
Javan Kauê Tavares Lacerda, Lucas Victor Silva Araújo.

Design Geral:
Nosso design geral é baseado em uma Facade que está ligada a um Sistema que gerencia o funcionamento de três controladores (ControllerUsuario, ControllerItem e ControllerEmprestimo), tendo em vista que os possui como instância da classe, cada controlador possui responsabilidades bem definidas, mantendo alta coesão entre as classes. O ControllerUsuario é responsável por reger a lógica associada ao Usuario, o ControllerItem conduz processos que estão relacionados aos itens e o ControllerEmprestimo comanda atividades relacionadas aos Emprestimos. Cada Controller está instanciado no Sistema e só podem ter acesso às funcionalidades uns dos outros a partir deste, que será responsável por coordenar o funcionamento e ser intermédio entre os próprios Controllers e entre estes e a Facade; tal implementação reduz o acoplamento e aumenta a coesão, haja vista que eventuais alterações em algum dos Controllers ou a implementação de um novo não afetará os demais Controllers e não exigirá alterações nas demais classes, demandando modificações apenas no Sistema e demonstrando a baixa dependência entre as classes que caracteriza o baixo acoplamento. Assim, caso um Controller dependa de alguma funcionalidade de outro, quem realizará a interação entre ambos será o Sistema, recebendo e repassando informações entre eles a partir de seus métodos**.** Além disso, utilizamos Enumeradores para organizar as constantes de Plataforma, Gênero e Classificação, que são atributos de alguns dos Itens relacionados, os Enums permitem acesso prático aos valores das constantes e realizam a verificação de possíveis valores inválidos, lançando exceções quando necessário.

Ainda, no Cartão de reputação do Usuário, em que precisamos de um comportamento dinâmico durante o tempo de execução, adotamos o padrão Strategy, que faz uso de uma interface que será implementada pelos diferentes tipos de cartões, permitindo que cada Cartão implemente os métodos de acordo com o seu comportamento e proporcionando a possibilidade de mudança do tipo de Cartão de maneira prática. Tal escolha é justificada pelos seguintes benefícios: baixo acoplamento, devido à independência entre cada tipo de Cartão; vantagens obtidas pelo comportamento polimórfico, tendo em vista as implementações da interface permite que cada classe modele sua própria lógica de funcionamento interno e a possibilidade de mudança do tipo de Cartão em tempo de execução, que é possível graças à implementação da interface.

Ademais, as exceções são lançadas de acordo com a responsabilidade da classe. Por exemplo: quando um Controller recebe um dado inválido para busca de uma informação, a exceção é lançada no controlador; contudo, se o dado inválido é passado na construção do objeto, a exceção é lançada no construtor do objeto. Além disso, utilizamos herança na classe abstrata item, visto que essa possui métodos e atributos em comum com Bluray e Jogos, mas não pode ser diretamente instanciada.

Caso 1:
No caso 1, é pedido a criação de uma entidade que represente um Usuário, que contém nome, email e telefone. Além disso é pedido a criação de um controlador que permita criar, ler informações, atualizar informações e deletar um usuário. Para isso, criamos a classe Usuario, que encapsula as informações de nome, email e telefone; e ControllerUsuario, que é o controlador, possuindo uma coleção Map que armazena os usuários, e os acessa por um objeto do tipo ChaveUsuario que encapsula o nome e o telefone do usuário e possui métodos getters e setters para as operações pedidas no caso de uso, no ControllerUsuario foram criados os métodos cadastrarUsuario(String nome, String email, String telefone) que cria um objeto Usuário com os parâmetros dados, tornando o ControllerUsuario como Creator de usuário, o método cria também o objeto ChaveUsuario com os parâmetros nome e telefone e armazena no mapa de usuários: o objeto ChaveUsuario como chave, e o objeto Usuário como valor. Ainda no ControllerUsuario, há o método getInfoUsuário(String nome, String telefone, String atributo) e retorna o atributo desejado em String. Já o método atualizarUsuario(String nome, String telefone, String atributo, String valor) e atualiza o atributo dado do Usuário para o novo valor passado como parâmetro, caso a informação a ser atualizada seja nome ou telefone, o método também altera a ChaveUsuario, fazendo uso de uma variável temporária que armazena o Usuário, removendo a chave antiga, e adicionando no mapa de usuários o Usuário armazenado na variável temporária junto à nova chave. O método removerUsuario(String nome, String telefone) identifica um Usuario a partir dos parâmetros e o remove do mapa de usuários. Todos esses métodos são chamados da Facade para o Sistema, que por sua vez, os delega para o ControllerUsuario. Vale salientar que a decisão de escolha pelo mapa se deu pela facilidade de uso através de um identificador, que é a chave do objeto, tornando fácil o acesso aos Usuarios. Também é válido salientar que em todos os métodos de ControllerUsuário que acessam um usuário há um método chamado checaSeUsuarioExiste(), que verifica através do método containsKey(ChaveUsuario) se a chave existe e para acessar um usuário há o instanciamento do Usuario acessado através da ChaveUsuario.

Caso 2:
No caso de uso 2, é pedido a implementação dos itens que serão disponibilizados para empréstimo. Para isso, criamos uma classe abstrata Item, que possui os atributos (nome (String), valor (double), estado de empréstimo(boolean) e numEmprestimos(int)) e métodos (getters, setters e contaEmprestimos() que soma 1 em numEmprestimos a cada vez que um empréstimo é realizado) em comum entre todos os itens e não pode ser instanciada porque não existe um Item genérico. Desse modo, utilizamos do polimorfismo por subtipo para implementar classes concretas filhas de Item, que correspondem aos diferentes tipos de BluRay (Bluray, BlurayEpisodio, BluraySeries, BlurayShow e BlurayFilme), JogoEletronico e JogoTabuleiro.

Para a implementação dos Blurays, foi criada uma classe Bluray, herdeira de Item, que é também uma classe pai e serve de modelo para os demais tipos de Bluray, possuindo os atributos (duração (int) e Classificação (enum)) e métodos comuns entre todos os Blurays (getters e os métodos validaBluray(), toString(), equals() e hashcode() que sofrem modificações nas classes filhas) e é estendida pelas classes filhas, que implementam

apenas os atributos e métodos particulares de sua funcionalidade, representando um polimorfismo de subtipo.

Dentre as classes herdeiras de Bluray estão:
A classe BlurayFilme que possui os atributos próprios: Gênero (que faz uso de um enumerador) e ano de lançamento (int) e os getters desses atributos. A classe BlurayShow que possui os atributos: nomeArtista(String) e numFaixas(int), que representam o nome do artista e o número de faixas presentes neste Bluray e os getters desses atributos. A classe BluraySeries que representa um Bluray de Série de TV e possui como atributos: uma coleção de episódios ArrayList<BlurayEpisodio> que armazena objetos de BlurayEpisodio, temporada(int), descrição (String) e Gênero (enum) e os métodos: adicionarBlurayEpisodio(BlurayEpisodio bluray), que recebe um BlurayEpisodio e adiciona no ArrayList; contemEpisodio() que verifica se a coleção de episódios está ou não vazia e retorna um boolean, getTamanho() que retorna a quantidade de episódios na coleção e getDescricao() que retorna a descrição em String. Por fim, há também a classe BlurayEpisodio, que possui como atributo próprio a duração (int).

Dentre as demais classes herdeiras de Item, estão:
A classe JogoEletronico, que possui como atributo próprio a Plataforma(enum) e o método validaAtributo(Plataforma) que verifica se a Plataforma dada é válida e possui como métodos os getters e o toString(); e a classe JogoTabuleiro, que possui como atributos um ArrayList de peças perdidas(String) e completo(boolean), além dos métodos existePecasPerdidas() que retorna uma String que diz se há ou não peças perdidas, adicionarPecaPerdida(String) que adiciona uma peça perdida na coleção de peças perdidas, o método getPecasPerdidas() que retorna a lista de peças perdidas e o toString(). Para cada um dos subtipos de Item, com exceção de JogoTabuleiro, foram utilizados Enums para armazenar as constantes (Classificação e Gênero para os Blurays e Plataforma para JogoEletronico), que permite fácil acesso aos valores das constantes e lança as exceções caso receba um valor de constante inválido.

Caso 3:

No caso 3 é pedido um método que faça a listagem de todos os itens cadastrados em todos os usuários, estejam emprestados ou não, de modo que esteja ordenada em ordem alfabética pelo nome do item. Para implementar tal funcionalidade trabalhamos com dois controladores, visto que a funcionalidade trabalha com Item e Usuário; primeiramente a entidade usuário contém um mapa de itens que tem como chave o nome do item e como valor o item, em Usuário há o método getItens() que retorna tal coleção, e em ControllerUsuario há o método getItensUsuarios() que retorna uma lista com os itens de todos os usuários. Assim, o Sistema chama o método getItensUsuarios() em ControllerUsuario, armazena a lista em uma variável e em seguida invoca o método listarItensOrdenadosPorNome() do ControllerItem passando-a como parâmetro, no ControllerItem tal método cria uma variável do tipo List com os itens recebidos por parâmetro, e sobre esta lista aplica uma ordenação que é baseada na implementação da interface Comparable na classe abstrata Item, sendo que Item implementa o método

CompareTo de modo que a ordenação se dá em ordem lexicográfica, então o método listarItensOrdenadosPorNome itera pela lista já ordenada e vai concatenando os toString() dos itens em uma só String, no final essa String é retornada para Sistema que também a retorna. Ainda, é pedido neste caso de uso, uma listagem semelhante à anterior; porém, a ordenação deverá ser de acordo com o valor dos itens. Para que isso seja possível, o mesmo esquema do método anterior é aplicado em listarItensOrdenadosPorValor() de Sistema, recebendo a lista de todos os itens através de ControllerUsuario e passando para ControllerItem como parâmetro, a diferença agora consiste apenas na ordenação, tendo vista que Item agora possui um comparador Default por nome, foi necessário criar uma classe que implementa Comparator e compara itens por valor (ComparaItemValor), de modo que a ordenação aplicada na lista passando o objeto da classe que implementou o Comparator ordena a lista por ordem de valor e retorna uma String dos toString() acumulados como no método anterior para Sistema que também a retorna. Além disso, é pedido um método que retorna informações de um Item e seu estado de empréstimo; para implementar tal método, Sistema delega trabalho para dois controladores, o de itens e o de usuários, como parâmetros o Sistema recebe nome e telefone do Usuario, nome do Item e o atributo desejado, para acessar a coleção de itens do Usuario, o Sistema invoca o método getItensUsuario(String nome, String telefone) do ControllerUsuario, armazenando a coleção em uma variável, então Sistema chama o método getInfoItem(String nomeItem, String atributo, List<Item> listaItens) em ControllerItem, o método então retorna para Sistema o atributo desejado, e Sistema o retorna.

Caso 4:
No caso do us4, foi solicitado: registrar um Emprestimo, com registro do dono do item, requerente, nome do item, data de empréstimo e período na qual será emprestado, e também que fosse implementado a função para devolver item, que receberá as mesmas informações, porém não contém o período e sim a data da devolução.

Para registrar um Emprestimo a facade irá invocar um método de Sistema, que por sua vez irá invocar um método do controllerEmprestimo, que será o registrarEmprestimo(dados do Emprestimo), que recebe como parâmetro os dados necessários mencionados no parágrafo acima, porém a String da dataDoEmprestimo é convertida para LocalDate para facilitar a manipulação nos outros procedimentos, e guarda no ArrayList<Emprestimo> que esse controller possui. Após finalizar esse processo a função é encerrada sem retornar nada. Agora, para a função devolverItem(dados da devolução) a invocação seguirá o mesmo fluxo, nesse método, é verificado primeiramente se o emprestimo existe, caso não, é lançada uma IllegalArgumentException, passando por isso é alterado a dataDeDevolucao(Atributo de Emprestimo) e realizado um cálculo dos dias de atraso com base no período dado no inicio do emprestimo e a data real da devolução, o cálculo é facilitado usando a biblioteca java.time.LocalDate, que converte uma String para uma data e usando a função between, que retorna os dias entre duas datas distintas, após usar essa função é subtraído do período dado inicialmente no emprestimo, o resultado deste calculo será o retorno dessa função devolverItem().

Caso 5:

No caso de uso 5, foram pedidos: listar os itens não emprestados no momento, listar o top-10 itens mais emprestados, listar os itens emprestados (incluindo nome do dono), listar o histórico de empréstimos de um item, listar os empréstimos de um usuário em 2 situações: quando este está pegando emprestado e quando está emprestando.

Para listar os itens não emprestados no momento, a Facade invoca o método listarItensNaoEmprestados() do Sistema, que irá invocar o método getItensUsuarios() do ControllerUsuario que retorna a lista de itens de todos os Usuarios e armazenará em uma variável, esta será passada como parâmetro para o método listarItensNaoEmprestados() do ControllerItem, que percorrerá a lista analisando o estado de empréstimo de cada item e irá retornar uma String com as informações de cada um dos itens que não estejam emprestados, isto é, com estado de empréstimo setado como false.

De modo semelhante, para listar o top-10 itens mais emprestados, a Facade invoca o método listarTop10Itens() do Sistema, que irá invocar o método getItensUsuarios() do ControllerUsuario para armazenar a lista de todos os itens dos Usuarios em uma variável, que será passada como parâmetro para o método listarTop10Itens() do ControllerItem, que ordenará a lista de itens utilizando um Comparator (*ComparaItemNumEmprestimos*), que compara os itens pela quantidade de vezes que foram emprestados de forma decrescente e em seguida retorna uma String com as informações de cada um dos itens da lista ordenada, de modo que o item com maior quantidade de empréstimos seja o primeiro e os demais estejam em ordem decrescente de empréstimo. O método listarTop10Itens() só lista itens que tenham sido emprestados, dessa forma, se há menos de 10 itens que tenham sido emprestados apenas uma vez, apenas estes serão listados.

Para os outros metodos de listarItensEmprestados(), listarEmprestimosUsuarioEmprestando(), listarEmprestimosUsuarioPegandoEmprestado(), e listarEmprestimosItem() seguirão o mesmo fluxo, que no caso, a facade irá invocar esses métodos no Sistema, e por sua vez o Sistema invoca no controllerEmprestimo. O método listarItensEmprestados() não recebe parâmetros e concatena em uma String os itens que estão contidos em Emprestimo que ainda estão em andamento, porém antes de concatenar é feito uma cópia da lista atual dos empréstimos apenas para usar a função Collections.sort() para ordena-los e ir concatenando na String que será usada como retorno desse método, que retorna para o Sistema, que retorna para Facade. O método listarEmprestimosItem(String nomeItem) irá listar todos os empréstimos na qual esse item passado por parâmetro esteja contido, de forma similar, é feito a concatenação em uma String dos toString() dos empréstimos vinculados ao Item e a função retorna essa String para o Sistema, que retorna para Facade. Nas funções listarEmprestimosUsuarioEmprestando() e listarEmprestimosUsuarioPegandoEmprestado(), é passado como parametro o nome e telefone do Usuario, nessa mesma função é feito uma invocação de um outro método que retorna a lista ordenada por nome na qual esse Usuario seja o dono ou requerente dependendo da função, voltado para a função principal, é feito a concatenação dos toString() desses empréstimos em uma String que será retornado para o Sistema, seguindo o mesmo fluxo das funções mencionadas neste parágrafo.

Caso 6:

Nesse caso é pedido a adição de um atributo em Usuário, agora usuário passa a contar com uma reputação, então foi adicionado na entidade um atributo do tipo double que representa a reputação, que inicia por default em 0.0, para implementar tal funcionalidade foi necessário além da adição do atributo do tipo double, a adição de métodos em Usuário, addReputacaoItemAdicionado(double valor), que recebe como parâmetro o valor do Item adicionado, e incrementa 5% desse valor na reputação do usuário, emControllerUsuario foi adicionado o método respectivo que addReputacaoItemAdicionadoUsuario(String nome, String telefone, double valor), que a partir da identificação de um usuário e do valor do Item, chama o método em usuário, então nos métodos que adicionam um item ao um usuário, Sistema agora além de chamar ControllerUsuario para obter a coleção de itens do referido usuário, também chama ControllerUsuario no método addReputacaoItemAdicionado() passando o valor do Item e o id do Usuário. Ainda, foi acrescentado em usuário o método addReputacaoItemEmprestado e em controller de Usuários addReputacaoItemEmprestadoUsuario, que tem comportamento semelhante ao anterior. No entanto agora é incrementado 10% do valor do item emprestado ao dono do item, e a chamada em Sistema ocorre no método registrarEmprestimo(String nome, String telefone, double valor), que recebe nome e telefone do Usuario e valor do Item. Além disso, foi adicionado em Usuário o método addReputacaoItemDevolvidoNoPrazo(), e, em ControllerUsuario, foi adicionado o método addReputacaoItemDevolvidoNoPrazoUsuario(); agora ao devolver um item, o Sistema que recebe de ControllerEmprestimo a quantidade de dias atrasados, identifica se essa quantidade é <= 0, caso positivo é chamado em ControllerUsuario o método addReputacaoItemDevolvidoNoPrazoUsuario(String nome, String telefone, double valor) em que nome e telefone são informações do Usuário requerente e valor é o valor do item devolvido passando o id do Usuário requerente do Empréstimo e o valor do item devolvido, então é incrementado na reputação do usuário 10% desse valor, foi também adicionado o método addReputacaoItemDevolvidoAtrasado() em usuário e addReputacaoItemDevolvidoAtrasadoUsuario() em ControllerUsuario. Para calcular a reputação, o método devolverItem() do Sistema identifica a quantidade de dias atrasados retornado da sua chamada de devolverItem() em ControllerEmprestimo, se esta for maior que 0 Sistema chama addReputacaoItemDevolvidoAtrasadoUsuario passando o id do requerente do empréstimo, a quantidade de dias atrasados e o valor do item devolvido, então em usuário é decrementado a reputação em 1% multiplicado pela quantidade de dias atrasados multiplicado pelo valor do item devolvido, foi ainda adicionado o getReputacao em usuário, e no método getInfoUsuario() em Controller de Usuários foi adicionado o atributo reputação que retorna em String o valor da reputação.

Caso 7:

Neste Caso é pedido a adição de um outro atributo ao Usuário, o Cartão Fidelidade, que possui os tipos Noob ,Caloteiro, Bom Amigo e Free Ryder, o Cartão começa por default como Free Ryder, no entanto tem seu tipo alterado em tempo de execução de acordo com a reputação do usuário e sua coleção de Itens, para implementar tal funcionalidade foi adotado o padrão Strategy tendo em vista que os Cartões não possuem atributo em comum, mas possuem comportamentos em comum, e tem seu tipo alterado durante a execução do programa, para tal alteração o padrão Strategy proporciona a vantagem do Polimorfismo que reduz acoplamento no programa e dá mais segurança dispensando uso

de casts, sendo assim foi criada a interface Cartao, que possui os métodos: validaPeriodo() que recebe um valor inteiro representando um período de empréstimo e retorna um valor booleano informando se as diretrizes daquele cartão permite um empréstimo por esse período, e getTipo() que retorna em String o tipo do cartão, foram criadas as classes para os quatro tipos de cartão implementando a interface Cartao, tendo o corpo de seus métodos de acordo com suas diretrizes, usuário passou a delegar o método validaPeriodo() também é delegado por Usuario e ControllerUsuario, sendo chamado em Sistema no método Registrar Empréstimo para verificar se seu período de empréstimo requerido é válido acordado com seu cartão, o método getTipo é delegado em Usuario por getCartao(), em ControllerUsuario o método getInfoUsuario() teve o atributo Cartao adicionado, que retorna em String o tipo do cartão do usuário, ainda em usuário foi adicionado o método atualizaCartao(), tal método verifica os critérios que determinam qual deve ser o cartão do usuário, então instancia o novo tipo e atualiza o tipo do cartão, além disso nos métodos que alteram a reputação do usuário foi adicionado uma chamada a atualizaCartao(), visto que a reputação é um dos critérios de classificação do cartão, ainda em Controller de Usuários há um método que delega atualizaCartao() de Usuario, e tal método é chamado em Sistema em todos os métodos que adicionam um item ao usuário, visto que esse é outro critério de classificação do Cartão.

Caso 8:

Neste caso é pedido que usuários com reputação negativa não possa mais tomar empréstimos até que sua reputação esteja maior ou igual a zero novamente, para implementar tal funcionalidade foi observado que usuários com reputação menor que zero possuem o cartão do tipo caloteiro, logo na interface Cartao foi adicionado o método emprestimoLiberado() que retorna um valor booleano informando se aquele tipo de cartão pode pegar itens emprestados, consequentemente tal métodos foi implementado nas classes que implementam Cartao, e todas retornam true, exceto Caloteiro.

Então usuário delega o método emprestimoLiberado() de Cartao, e ControllerUsuario o delega novamente de usuário, logo ao registrar um empréstimo Sistema chama de Controller de Usuários o método emprestimoLiberadoUsuario(String nome, String telefone), assim verifica se o usuário está apto a pegar itens emprestados, caso não esteja é lançado uma exceção.

O caso pede ainda a listagem de todos os usuários com reputação negativa, sendo tal listagem ordenada pelo nome dos usuários, para implementar tal Usuario passou a implementar Comparable e o método CompareTo, sendo que este retorna o CompareTo entre o nome do usuário e do outro a ser comparado, em ControllerUsuario foi adicionado o método listarCaloteiros(), que cria um lista com os usuários com reputação menor que 0 e posteriormente aplica um ordenação nesta lista baseada na implementação do Compare em Usuário, para finalizar há um iteração nesta lista que acumula os toString() dos usuários em uma única String que é retornado no fim, vale salientar que Sistema delega esse método.

O caso também pede a listagem do top10 de melhores usuários tendo a ordenação baseada na reputação dos usuários, para implementar tal foi necessário a criação de uma

classe que implementa Comparator e compara um usuário por sua reputação, sendo válido salientar que a comparação é inversa pois a classificação é do menor para o maior, então o método de Controller de Usuários listarTop10MelhoresUsuarios cria uma lista com todos os usuários, e aplica nesta uma ordenação baseada na nova classe comparadora implementada, posteriormente há um iteração nesta lista do indíce 0 a 9 que acumula os toString() dos usuários em uma única String, tal método também é delegado por Sistema.

É pedido ainda a listagem do top10 de piores usuários, a implementação deste caso foi bastante semelhante a anterior, o que difere é a criação de uma outra classe que implementa comparator e ordena os usuários da pior reputação para maior, então em Controller de Usuários no método listarTop10PioresUsuario() que também é delegado por Sistema, é criada uma lista com todos os usuários, e aplicada nesta uma ordenação baseada na nova classe comparadora de usuário mencionada anteriormente, posteriormente há uma iteração nesta lista de 0 a 9 que acumula os toString() dos usuários em uma única String e a retorna.

Vale Salientar que nas listagens há um verificação, para o caso de existir menos de 10 usuários, em tal caso é lançada uma exceção informando que há menos que 10 usuários.

Caso 9:

O Caso pede que o estado do Sistema seja salvo, para que quando aberto todos os dados persistam nele, para tal implementação foi adicionado o método fecharSistema() na fachada, tal método tenta criar um objeto do tipo FileOutputStream, em seguida encapsular tal no objeto ObjectOutputStream, posteriormente o método tenta gravar sob este o objeto Sistema, caso haja um erro, há a captura desta através do Catch, nesse a exceção é relançada com a mensagem de Falha ao Salvar Sistema, a decisão de salvar o objeto Sistema foi tomada tendo em vista que Sistema instancia os Controladores e consequentemente as outras classes, logo as coleções são encapsuladas e salvas dentro do objeto Sistema, para realizar tal, foi necessário a implementação da interface em todas as classes desde Sistema, exceto as de teste, tal implementação se deve ao fato de Sistema se relacionar direta ou indiretamente com todas as classes.

Foi ainda implementado na fachada o método iniciarSistema() que caso o arquivo informado exista tenta o encapsular no objeto ObjectInputStream para que este seja lido, e posteriormente o Sistema instanciado na fachada passa a ser o lido do arquivo, caso o arquivo não exista, um novo Sistema é instanciado, caso alguma haja alguma falha na leitura do arquivo a exceção é capturada e relançada com mensagem de Falha na leitura.

Link para o repositório no Git Hub:
https://github.com/igoratf/Projeto-lp2