Universidade Federal de Campina Grande - UFCG
Centro de Engenharia Eletrica e Informática
Unidade Acadêmica de Sistemas de Informação
Curso de Bacharelado em Ciência da Computação
Laboratório de Programação 2

Alunos: Alessandro Lia Fook Santos
Lucas Diniz dos Santos
Filipe Teotônio Ramalho Mendonça

Relatório do Projeto intitulado "Laborário Sistema Orientado a Objeto de Saúde"

Inicialmente cabe falar que o projeto será relatado conforme especificação onde deve ser mencionado o uso de Polimorfismo (Herança, Inferface), Padrões de Projeto(Composição, Strategy), Coleções(Lista, Mapa, Conjunto), e as distribuilções de responsabilidades conforme o GRASP (Construtor, Acoplamento, Expert, Coesão).

Caso 0 - Design do Sistema

Antes de adentrar os casos de uso solitados, é importanto relatar que de forma mais geral o sistema foi implementado seguindo o padrão MVC (Model View Controller), que tem por objetivo modularizar as atribuições do sistema, fato que separa os especialistas na informação, garante uma alta coesão, e assegurando um baixo acoplamento.

No sistema desenvolvido, a parte titulada de Model, que conterá a lógica do sistema relacionando-se com as partes mais baixas da arquitetura, é implementado pela classe "ComiteGestor", contendo instância das classes mais baixas do sistema para poder administrar suas funcionalidades. O construtor da classe mencionada instancia todas as classes da parte mais baixa do sistema necessárias a sua lógica, conforme será explicado mais adiante, nos casos de uso.

Enquanto o View, que tem a responsabilidade de exibir os resultados e coletar as informações fornecidas pelo usuário externo ao sistema, é implementado na classe HospitalFacade, que utiliza o

padrão de fachada para torná-la o equivalente a uma saída e entrada do sistema, haja vista que não é executada nenhuma interface gráfica no projeto.

O Controller, que tem por objetivo intermediar os outros dois, é implementado na classe HospitalController e além de intermediar a comunicação é responsável por garantir a permanência dos dados de execução.

Ainda, cabe mencionar que as exceções foram lançadas sempre de forma checked, conforme mencionado na especificação, e os erros que deveriam lançar os outros tipos de exceção foram aproveitados do próprio java, que já faz as verificações que seriam cabíveis.

Portanto, as exceções são lançadas a partir de cada funcionalidade foram encadeadas conforme o caso para permitir um rastreamento completo da origem do erro.

Caso 1 - Primeiro cadastro, login do diretor geral e cadastro de novos colaboradores:

Para viabilizar a funcionalidade solicitada, foi criada uma classe de nome Comite Gestor,com o objetivo de gerenciar o cadastro de funcionarios e o acesso dos usuários ao sistema. Esta classe armazena como membro o diretor geral, uma vez que só pode haver um. Os funcionários são armazenados em um conjunto(HashSet), pois um funcionário não pode ter dois cadastros no sistema.

Tanto o diretor foi tratado como um funcionário para fim de cadastro, pois ambos são diferenciados apenas pelo número de sua matrícula, portanto utilizou-se uma classe única para registrar ambos.

Importante ressaltar que a classe funcionário é subclasse, pois existe uma superclasse nominada Pessoa, que guarda as informações mais gerais como nome, data de nascimento e identificador no sistema, pois existem dois tipos de pessoa no sistema: os funcionários e pacientes, garantindo um maior reuso de código.

Cabe mencionar, ainda, a implementação do padrão de projeto da fábrica, visando reduzir o acoplamento entre as classes e modularizar a criação dos funcionários, pois a fábrica é que tem a lógica de criar a matrícula do funcionário de acordo com seu cargo.

Por fim, foi escolhido armazenar todas as matrículas e suas respectivas senhas em um mapa, relacionando cada matrícula a sua senha, para assim assegurar uma maior velocidade no processamento de busca da senha compatível com a chave além de proteger as informações individuais de cada funcionário, pela inexistência de método que recupere a senha armazenada em um funcionário.

Caso 2 - Atualizar informações de usuários e excluir usuários do sistema:

Como especificado nos casos de uso, apenas o diretor geral pode atualizar outros funcionários e excluí-los ao passo em que qualquer outro pode atualizar a si mesmo. Para isso, foram criados métodos que validassem a autorização de um funcionário de acordo com seu cargo.

Visando modularizar tais métodos foram criadas classes que comportassem esses métodos, dentre elas a do caso de uso em tela, que observa o primeiro dígito do número relativo à matrícula do funcionário, tendo em vista que este é o dígito que categoriza o funcionário.

Caso 3 - Cadastrar e atualizar prontuários dos pacientes:

O caso de uso foi definido como “Descrição principal: Como técnico administrativo desejo cadastrar novos pacientes para gerenciar suas ações na clínica. ”

Os dados do paciente são armazenados em uma classe própria chamada Paciente que é subclasse de Pessoa, esse *design* permitiu um alto reuso de código através do uso da herança.

Foi criada também uma classe chamada Prontuario e nela foram inseridos dois atributos, um objeto do tipo Paciente e uma lista que futuramente iria armazenar todos os procedimentos realizados no paciente ligado ao prontuário. A composicação entre paciente e prontuário facilitou que cada paciente estivesse ligado a apenas um prontuario como solicitado na especificação. O alto acoplamento entre essas duas classes foi inevitável, entretanto, por se tratar de um hospital todos os procedimentos clínicos realizados no paciente devem constar em seu prontuário.

Caso 4 - Farmácia e Medicamentos:

A especificação foi atendida a partir da criação de de uma farmácia que foi inserida como objeto interno no comite gestor que passou a fazer fowarding dos métodos a serem utilizados pelo usuário.

A farmácia ficou com a atribuição de armazenar os medicamentos, em um conjunto(Set) assegurando a impossibilidade de existem dois cadastros iguais do mesmo medicamento. Essespor sua vez são representados em sua própria classe, assim mantendo um baixo acoplamento no sistema, e deixando cada classe com seu expert.

Por úlitmo, foi implementada uma composição com interface dentro do medicamento, haja vista que conforme especificação haveriam dois tipos de medicamento, um genérico e outro de referência, assim foi gerada uma interface que representaria os dois tipos, sendo cada um implemetado na sua propria classe, através do uso da interface do tipo de medicamento, permitindo uma chamada polimórfica no momento de calcular o preço final do medicamento.

Caso 5 - Banco de Orgãos:

O banco de órgãos possui uma classe própria que se responsabiliza por dar acesso à adição e remoção de órgãos (que são objetos próprios) no sistema. Um *ArrayList* foi escolhido para armazenar os órgãos de maneira a facilitar seu acesso e de atender a todos os requisitos especificados. O banco de órgãos possui uma relação de composição com a clínica para manter o *expert*, pois cabe à clínica gerenciá-lo.

Caso 6 - Realização de Procedimentos:

A realização de procedimentos é realizada em várias etapas, e a primeira é a sobrecarga do método onde o nome do órgão pode ser inserido como parâmetro ou não, pois o procedimento de transplante necessita de informação enquanto os demais não.

Assim, visando aproveitar as composições existentes sem criar ciclos na arquitetura do sistema, mantendo a coesão e o baixo acoplamento, o método segue os seguintes passos:

No comite gestor é delegado a farmácia a função de verificar se os medicamentos informados estão disponíveis, retornando o valor acumulado dos custos.

Posteriormente, o comite repassa o valor dos custos com meditamento como parametro para a clinica, que por sua vez delega ao banco de órgãos o fornecimento de orgão compatível, se possível, no caso do transplante, após isto, o procedimento é o mesmo para todos, a clinica delega ao prontuário o registro do procedimento.

O prontuário por sua vem utiliza o padrão strategy para poder realizar uma chamada polimórfica do procedimento solicitado que é criado utilizado o padrão factory permitindo um baixo acoplamento e uma maior modularização.

Por fim, o procedimento escolhido faz as alterações no paciente, e o protuário encaminha a este o valor da conta, somando os valores retornados pelo procedimento aos custos com medicamento.

Caso 7 - Cartão de Fidelidade:

O cartão fidelidade for criado utilizando o padrão strategy. A interface Fidelidade foi criada e nela foram inseridos dois métodos, um para gerar um desconto sobre o preço de um serviço e outro para adicionar bônus de pontos fidelidade. Foram criadas também as classes Padrao, Master e Vip, que implementam os metodos da interface de acordo com a especificação do projeto.

Além disso, foi criada a classe CartaoFidelidade e nela foi feita uma composição com um objeto do tipo Fidelidade (strategy), isso garantiu menor acoplamento e uma fácil mudança dinâmica entre os tipos de fidelidade do cartão, uma vez que cada classe que implementa a interface não possui atributos e difere apenas no comportamento de seus métodos. E por fim, foi feita uma composição entre CartaoFidelidade e Paciente.

Caso 8 - Exportar ficha de pacientes:

Para a exportação da ficha de um paciente para um arquivo “.txt”, obtém-se uma *String* gerada pelo seu prontuário tendo em vista que é neste em que as informações de todos os procedimentos feitos no paciente se encontram. A geração dessa *String* é feita a partir da concatenação das informações do paciente como nome, peso etc, solicitadas na especificação, com as dos procedimentos realizados armazenados em seu prontuário, como nome do médico, nome do procedimento etc.

A *String* é então retornada até o controller que tem a responsabilidade de gravá-la em um arquivo com extensão “txt”. O arquivo receberá o título de acordo com o nome do paciente e a data de sua criação e substituirá aquele que possuir mesmo nome conforme solicitado na indicação do caso. Finalmente, através de uma stream de escrita grava-se o conteúdo da *String* no arquivo recém-criado,

Caso 9 - Salvar sistema:

A especificação foi implementada em duas partes, no momento de iniciar o sistema, e quando de seu fechamento. Uma vez que o padrão MVC implemente uma composição entre do Model e o Controller, com aquele sendo objeto interno deste, então a opção foi utilizar a instância do Model para garantir a permanência da informação.

Portanto, o médoto que inicia o sistema verifica se existe um arquivo com as informações da instância do Model, se positivo utiliza-se de uma stream de leitura do objeto contido na fonte “.dat” especificada, e atribui o objeto recuperado a instância do Model após realizar a devida conversão de tipo; caso não exista o arquivo, o método inicia uma nova instância, e realiza seu armazenamento na memória secundária através de uma stream de escrita do novo objeto na fonte “.dat” especificada.

Por outro lado, no momento de fechar o sistema, o método simplesmente repete a segunda parte do método que inicia o sistema.