UNIVERSIDADE FEDERAL DE CAMPINA GRANDE - UFCG
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
DEPARAMENTO DE SISTEMAS E COMPUTAÇÃO
CAMPUS DE CAMPINA GRANDE - PB

RELATÓRIO DO PROJETO DE LABORATÓRIO DE PROGRAMAÇÃO 2

Eric Breno Barros do Santos
Estacio Pereira da Silva Neto
Thaynan Andrey Rocha Nunes

Campina Grande - PB
24 de Maio de 2016

Relatório do Projeto

"O SOOS (Sistema Orientado a Objetos para a Saúde) é um sistema que será usado por gestores e colaboradores de clínicas particulares que possuem parceria com o Sistema Único de Saúde (SUS). O objetivo é fornecer meios que possam agilizar e conduzir a gerência dos recursos humanos e físicos do hospital, como também auxiliar nos procedimentos internos inerentes à organização e seus departamentos".

1. Conceitos Aplicados

1.1 Controller e Façade

A Façade do nosso programa é a "fachada", o que abstrai a lógica de negócios do mesmo, permitindo uma relação entre o usuário e o sistema, sem que seja necessário o conhecimento de como o mesmo funciona. Logo, possui os métodos necessários para realizar as operações almejadas pelo usuário.

O Controller do nosso sistema possui gerenciadores que dividem as responsabilidades e, assim, evitam uma God Class, ou seja, uma classe com muitas atribuições, sendo repassada a responsabilidade da lógica de negócios para os seus gerenciadores específicos, nos quais também caracterizamos a farmácia e o banco de órgãos como gerenciadores por definição dada de como os mesmos funcionam.

1.2 Validações e Exceções

As exceções que temos no projeto, responsáveis por parar a execução de certa funcionalidade do sistema devido a casos inesperados de entradas de determinado dado ou permissões para executar uma alguma funcionalidade, são abstraídas e verificadas pelos validadores da classe "ValidadorDeDados" e "ValidadorDeLogica", o que encapsula a responsabilidade de validar dados ou permissões no momento de execução do programa, e, assim, permite uma manutenção mais rápida e simples. Também é notório observar o reuso de código, além da modularização, delegando responsabilidades para classes específicas e que são utilizadas em diversos pontos do código.

As Exceções são subdivididas em exceções de lógica e dados, obtendo uma hierarquia que visa orientar o programador a visualizar, de forma mais rápida, o tipo de erro acometido. Utilizamos exceções não checadas quando desejamos lançá-las além da Façade e Controller (o que acontece com as exceções que se espera que aconteçam nos testes disponibilizados) e para camadas mais "baixas" do sistema

usamos exceções checadas, para o programador poder saber em quais lugares explicitamente pode ser lançada uma exceção que deve ser tratada e não necessariamente parar a execução do programa.

1.3 Constantes e Reuso

Constantes de nosso programa são encapsuladas na classe “Constantes”, a qual carrega consigo constantes que serão utilizadas em diversas partes do sistema. Na classe “MensagensDeErro” encapsulamos as mensagens mais recorrentes de erro que serão retornadas para o usuário, quando um determinada exceção ocorrer em nosso programa. Na classe “Util”, temos métodos de propósito geral que são utilizados em diversas classes do programa. Logo, essa modularização permite o reuso de funcionalidades e constantes por todo nosso programa.

1.4 Packages

Os “packages” foram criados obedecendo uma ordem hierárquica e uma forma de divisão lógica das partes e entidades do sistema, a qual possuímos o package principal, projeto, e o subdividimos em três packages: “exceptions”, “hospital” e “util”. Nos mesmos, as classes possuem caraterísticas comuns entre si, por exemplo, encontramos no package hospital, toda a lógica do hospital: funcionários, pacientes, farmácia, banco de órgãos, entre outros. Enquanto que o package exceptions possui toda hierarquia de exceções utilizada no programa.

1.5 Expressões Lambda

O uso de expressões lambda no projeto da disciplina fez com que pudessemos dispensar a existência de algumas classes do projeto e “convertê-las” em dois métodos. Com isso, proporcionou um código mais “enxuto” e bem mais elegante, visto que utilizamos uma interface funcional, a qual define os tipos criados pelas expressões Lambda.

1.6 Reflexividade (Reflection)

Reflexividade é um recurso altamente poderoso de programação disponível na linguagem Java, a mesma nos proporcionou a abstração de diversas verificações, assim como linhas de validação antes da criação de objetos. Sobre as verificações, temos o exemplo do método “getInfo” e o método “atualizaInfo” que, em outra

abordagem, conferiria entre um conjunto definido separadamente pelo programador qual campo seria retornado ou atualizado, e no com o Reflection isso é feito de uma forma direcionada à classe diretamente, e não um conjunto separado de opções válidas.

A reflexividade nos auxilia também na criação de objetos, onde tendo o "package" que contém as classes disponíveis de serem criadas, o nome da classe e os parâmetros do construtor da classe, pode-se criá-lo sem estruturas condicionais, enquanto que em "factories" padrões, a criação é totalmente baseada em condições, da mesma forma que mostrado para os método "getInfo" e "atualizaInfo".

1.7 Collections

Uso das Java Collections no projeto:

ArrayList em Banco de Órgãos - É usado um ArrayList, pois podem haver órgãos iguais na coleção, e vão haver mais operações de busca do que de remoção, neste contexto há vantagem de usar ArrayList ao invés de LinkedList.

ArrayList em Farmacia - É usado um ArrayList, pois podem haver medicamentos iguais na coleção, pode ser necessária ordenação da coleção de formas distintas, o que não permitiria o uso de um HashSet. Além disso, vão haver mais operações de busca do que de remoção, e aí se tem vantagem de usar ArrayList em vez de LinkedList.

HashMap em Gerenciador de Funcionário - É usado um mapa de Matricula - Funcionário pois não haverão dois funcionários com uma mesma matricula no sistema, e muitas operações envolvem recuperar um funcionário por sua matricula ou usar a matrícula do funcionário.

HashSet em Cargo - É usado um HashSet, pois não haverão duas permissões iguais, e não é necessário ordenação nem posições relativas das permissões contidas para determinado cargo.

TreeMap em Gerenciador de Prontuario e Paciente - É usado um mapa ordenado de Paciente - Prontuário, pois não haverão dois pacientes iguais no sistema, e é necessário ter a ordenação dos pacientes de acordo com seu nome, por ordem lexicográfica crescente, e o TreeMap já oferece esta ordenação na inserção dos elementos.

ArrayList em Prontuario - É usado um ArrayList pois os procedimentos a serem guardados podem ser iguais, além de que a ordem que foram inseridos importa, e não vão ocorrer operações de deleção, apenas de busca para resumir os procedimentos, o que dá a vantagem do ArrayList em relação ao LinkedList.

2. Casos de Usos

2.1 Caso de uso 1

Neste caso de uso 1, o programa cumpre com as metas preestabelecidas pelo cliente, no qual, é possível, realizar a liberação de um sistema, o cadastro de um funcionário, “login” e “logout”, respeitando as regras do mesmo, nas quais determinado funcionário pode ou não ter acesso a função.

Foi criado um “Enumeration” para as permissões, mantendo a integridade do código e facilitando a análise de permissões que dado funcionário possui.

Para o cargo que cada funcionário (composite) possui, foi utilizado herança , na qual existe a superclasse “Cargo”, uma entidade abstrata pois ela carrega apenas a ideia de como é e como funciona um cargo. Suas respectivas filhas, caracterizam cada funcionário com seu cargo, o qual possui um conjunto de permissões para realizar dada função no hospital.

2.2 Caso de uso 2

Neste caso de uso 2, o programa cumpre com as metas preestabelecidas pelo cliente, no qual, é possível, atualizar as informações de um funcionário (por meio da reflexividade), assim como, excluir um determinado funcionário, respeitando a permissão concedida ao mesmo para execução da função.

2.3 Caso de uso 3

Neste caso de uso 3, o programa cumpre com as metas preestabelecidas pelo cliente, no qual, é possível, cadastrar um paciente, salvar suas informações em um prontuário, e consultá-las a partir de um id único que caracteriza o paciente.

O prontuário (composite) compõe procedimento, guardando todos os procedimentos realizados pelo paciente e (composite) compõe um paciente que armazena todas suas informações, tendo uma relação 1 para 1, o prontuario é usado para fins de consulta e histórico do hospital. Os id’s dos pacientes são gerados automaticamente de forma dinâmica e sem permitir repetição pela classe “GeradorIdPaciente”.

Para o prontuario e o paciente temos o “GerenciadorProntuarioPaciente”, que encapsula as responsabilidades e as lógicas de gerenciamento que envolvem paciente e prontuario.

2.4 Caso de uso 4

Neste caso de uso 4, o programa cumpre com as metas preestabelecidas pelo cliente, no qual é possível criar e gerenciar medicamentos, caracterizado pelos seus tipos, os quais tornam os medicamentos únicos e classificáveis.

Utilizou-se uma interface "TipoMedicamento" para caracterizar o tipo de medicamento existente, a qual é implementada por duas classes - "Generico" e "Referencia" -, sendo, neste contexto, possível calcular o preço final do medicamento mediante o tipo do mesmo (Referência ou Genérico). Logo, a classificação de cada medicamento pelo seu tipo, utilizando interfaces (Strategy), proporciona um código modularizado, com baixo acoplamento e suscetível a mudanças, caso sejam necessárias, o que possibilita uma manutenção rápida e com baixo custo.

As categorias que caracterizam os medicamentos foram implementadas pela criação de um Enumeration de categorias ("CategoriasValidasMedicamentos"), o qual acarreta na definição de constantes de categorias que podem configurar um medicamento, sendo suscetível a mudanças, caso sejam adicionada e/ou removidas categorias, assim como, validar categorias passadas pelo usuário de forma simples e elegante. Logo, possuímos um código modularizado e reusável.

A farmácia é responsável por gerenciar os medicamentos, a mesma cria e armazena os medicamentos em uma coleção - lista -, o que possibilita a ordenação dos medicamentos de acordo com a necessidade do usuário. Para isso, utilizamos comparadores em lambda, os quais configuram um código mais simples e flexível a uma ordenação dinâmica no momento de execução do programa.

2.5 Caso de uso 5

Neste caso de uso 5, o programa cumpre com as metas preestabelecidas pelo cliente, no qual, é possível, cadastrar e gerenciar órgãos que serão utilizados futuramente em procedimentos de transplantes.

"Orgao" é uma classe que define o tipo órgão, e é gerenciado no banco de órgãos, o qual cadastra, retira e busca órgãos. Toda a lógica de negócios relacionada a órgãos é realizada pelo banco de órgãos, atuando como os demais gerenciadores pra suas respectivas entidades.

2.6 Caso de uso 6

Neste caso de uso 6, o programa cumpre com as metas preestabelecidas pelo cliente, no qual, é possível, realizar determinados procedimentos solicitados no hospital, e o mesmo ser armazenado no prontuário do paciente, indicando data, valor e médico que realizou o devido procedimento.

Há uma superclasse "Procedimento", a qual é uma entidade abstrata, pois carrega apenas a ideia do que é e como se realiza um procedimento. As classes que

caracterizam um determinado procedimento, herdam da classe “Procedimento” e implementam seus comportamentos específicos. Escolhemos herança, pois todo Procedimento possui atributos em comum, que poderiam ficar na superclasse e comportamentos semelhantes além dos específicos, portanto, evitamos repetição de código apesar do alto acoplamento gerado por uma herança. O procedimento de transplante compõe órgão.

2.7 Caso de uso 7

Neste caso de uso 7, o programa cumpre com as metas preestabelecidas pelo cliente, no qual, todo paciente (strategy) possui um cartão fidelidade, o qual é responsável por calcular pontos ganhos pelo paciente devido determinado procedimento, assim como ganhar descontos devido seu cartão.

Escolhemos usar o padrão de design “strategy”, pois com o acumulo de pontos, o paciente muda dinamicamente o seu cartão fidelidade, o que proporciona um maior desconto no pagamento para realizar determinado procedimento, assim como, ganhar pontos extras de acordo com o cartão que o mesmo possui.

2.8 Caso de uso 8

Neste caso de uso 8, o programa foi modificado para ter uma nova funcionalidade e poder exportar fichas de pacientes, contendo informações do paciente e dos procedimentos já realizados pelo mesmo. As fichas são geradas e guardadas em arquivos de texto de acordo como pedido na especificação, através do método “exportarFichaPaciente”, em que se passa o id do paciente desejado a ter suas informações exportadas do sistema.

2.9 Caso de uso 9

Neste caso de uso 9, o programa foi modificado para que todo o sistema possa ser guardado, no qual podemos, mesmo depois de que o mesmo seja fechado, recuperar as informações previamente cadastradas.