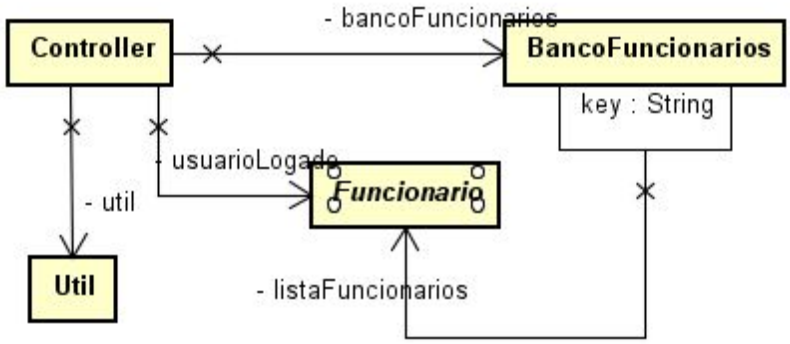
UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CURSO: CIÊNCIA DA COMPUTAÇÃO
DISCIPLINA: LABORATÓRIO DE PROGRAMAÇÃO II
PROFESSORA: LÍVIA SAMPAIO
GRUPO DO PROJETO: RAQUEL PAZ, PEDRO MAIA, IONÉSIO JÚNIOR

RELATÓRIO - PROJETO DE LABORATÓRIO 2015.2

❖ INTRODUÇÃO

Tendo em vista que o ***SOOS*** seria composto por vários setores responsáveis por diferentes operações, modularizamos o projeto de acordo com a necessidade de cada caso. Dessa forma, haveriam os Bancos (Funcionários, Pacientes, Órgãos) e a Farmácia. Cada operação enviada da Façade seria delegada ao Controller, que designaria ao setor responsável. Tal funcionamento será mostrado mais detalhadamente a seguir, com focos em cada caso da especificação.

➔ CASO DE USO 1



Em relação ao caso 1, foram usados os conceitos de ***Facade***, técnica que busca encapsular o código da forma mais simplória possível geralmente possuindo apenas métodos de delegação; ***Controller***, técnica usada para garantir o controle de toda a conexão entre as partes do código; ***Composições*** utilizando ***forwarding*** (delegação de métodos) de *Façade* para *Controller* e deste para *BancoFuncionarios*, visando modularizar, aumentar a organização, o reuso de código, facilitar a compreensão e consequentemente a manutenção.

Também se utilizou de ***Herança*** nas hierarquias de Exceptions (*FuncionarioException* herda de *ControllerException*) e classes que herdam de Funcionário que é uma abstração da ideia de todos os tipos de funcionários, contendo características comuns a todos eles. Nesse mesmo relacionamento de herança também foram usados os conceitos de ***sobrescrita de métodos*** fazendo com que determinado estado tivesse comportamentos distintos de acordo com o tipo de *Funcionário*. Este, por sua vez , por se tratar apenas de uma abstração foi construído de modo que não pudesse ser instanciado (***abstract***) mas tornando obrigatória a implementação de um método de sobrescrita em suas classes dependentes.

Outro conceito utilizado foi o ***factory*** (classe responsável unicamente pela criação de objetos específicos tornando o código mais robusto e modularizado), na criação dos tipos de funcionários instanciados com variáveis do tipo Funcionário (**polimorfismo**).

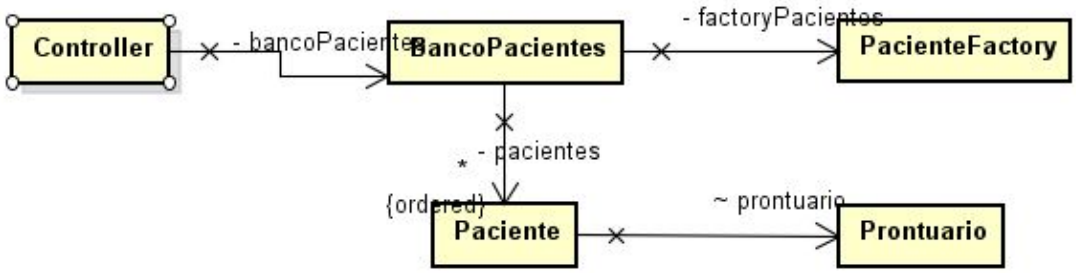
Outra técnica usada foi as **chamadas polimórficas** feitas por métodos sobrescritos para definir as permissões de cada tipo de funcionário e ***singleton*** pela classe Util, responsável apenas pelo auxilio no encapsulamento, sendo instanciada em um único local.

➔ CASO DE USO 2

No segundo caso foi utilizado novamente o conceito de ***composição*** na delegação de métodos entre as classes *Façade*, *Controller* e *BancoFuncionários*, visando manter um código limpo. Usa-se também o conceito de ***polimorfismo*** ao associar uma variável do tipo Funcionário a um objeto (pois tentar instanciar o objeto em sua classe primária seria arriscado visto que existem outros tipos de funcionários armazenados na mesma estrutura de dados) e ***Herança*** pela hierarquia de Exceptions lançadas.

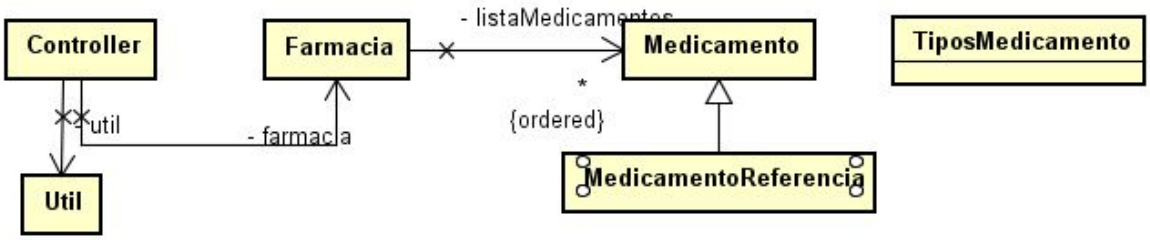
Usou-se também ***singleton*** pela classe Util, responsável apenas pelo auxilio no encapsulamento, sendo instanciada em um único local (*Controller*).

➔ CASO DE USO 3



No caso de uso 3, novamente foram usados conceitos de ***Composição*** visando obedecer as regras de ***alta coesão***, ***baixo acoplamento***, ***construção*** e ***encapsulamento*** definidas pelo **GRASP.** Quebrando o programa em partes definidas responsáveis por funções específicas como **factory**, usado para criação de objetos do tipo paciente. Também foi utilizada ***Herança*** pela hierarquia de Exceptions lançadas (*PacienteException* e *DataInvalidaException* herdam de ControllerException).

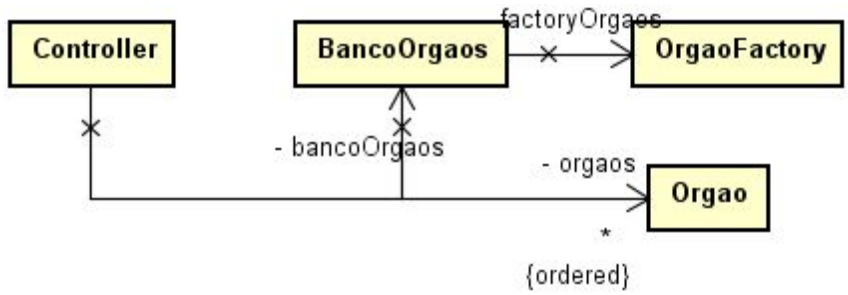
➔ CASO DE USO 4



No caso 4, relacionado à implementação da *Farmácia* e dos Medicamentos, utilizamos o conceito de ***Herança***, para garantir o reuso de código e possibilitar o polimorfismo. A classe *Medicamento* contém os métodos referentes ao seu comportamento e a classe *MedicamentoReferencia* herda da mesma, sobrecarregando métodos de preço e tipo.

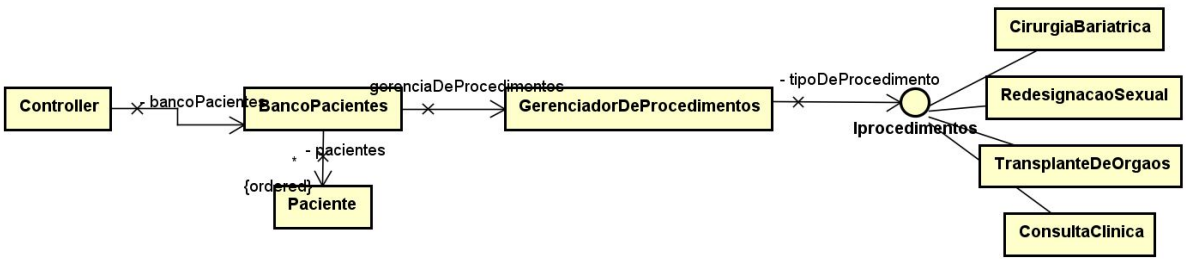
Dessa forma, também ocorre o uso do *Generics* na criação, em *Farmácia*, de uma lista de medicamentos, garantindo uma maneira mais abrangente de iteração e acesso aos métodos das classes de *Medicamento*. Além disso, ocorre o uso de composição de *Farmácia* com *MedicamentoFactory*.Usou-se também o conceito de ***singleton*** pela classe Util , responsável apenas pelo auxilio no encapsulamento, sendo instanciada em um único local(Controller).

➔ CASO DE USO 5



O caso de uso 5, também utilizamos o conceito de ***Factory***, deixando para a classe *OrgaoFactory* a responsabilidade de criação de objetos do tipo Órgão. Da mesma maneira que foi implementada o Banco de Funcionários (conceito de ***Composição***), também criou-se a classe *BancoOrgaos* (que realiza composição com *OrgaoFactory*), onde estaria encapsulada os métodos e a coleção de órgãos, garantindo a modularização do código visando obedecer os conceitos do ***GRASP***. Quanto às exceptions, foi criada também o *OrgaoException*, herdada de *ControllerException (**Hierarquia de Exceptions**)*, que seria lançada se houvesse alguma irregularidade, como busca por um órgão não cadastrado.

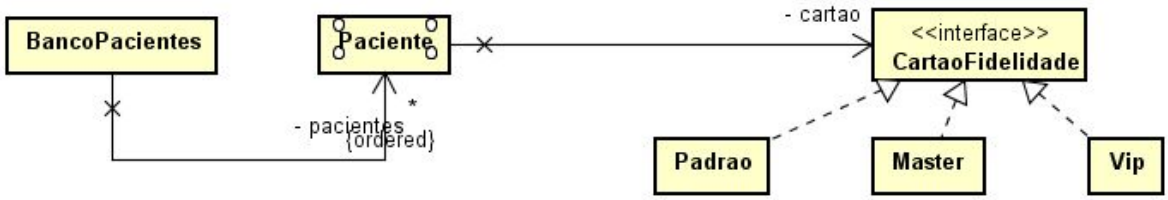
➔ CASO DE USO 6



Para a realização dos procedimentos, criamos uma classe *GerenciadorDeProcedimentos*, que faria o registro e o cálculo do procedimento. Para suprir a necessidade do projeto, utilizamos o conceito de ***Interface*** a partir de *Iprocedimentos*, implementada pelos tipos *CirurgiaBariatrica*, *RedesignacaoSexual*, *TransplanteDeOrgaos* e ConsultaClinica.

Dessa forma, antes de realizar o procedimento, é definido o tipo de operação (fazendo a instância à variável tipoProcedimentos) e então, resta apenas o BancoPacientes realizar o ***fowarding*** para o gerenciador, que chama o método correto (***chamada polimórfica***) e possibilita o andamento do procedimento.

➔ CASO DE USO 7



No caso 7, relacionado ao cartão Fidelidade do paciente, utilizamos o conceito de ***interfaces***, pois a troca do tipo de cartão fidelidade precisa ser dinâmica (característica do **strategy**) a medida da aquisição de pontos. A interface *CartaoFidelidade* contém os métodos referentes ao seu comportamento e as classes *Padrao*, *Master* e *Vip* implementam a mesma. Além disso, ocorre o uso de ***composição*** de *Paciente* com *CartaoFidelidade*.

➔ CASO DE USO 8

Uso de ***arquivos*** visando obter a persistência de dados importantes em uma memória secundária. No caso em específico, se faz necessário salvar as informações do prontuário de determinados pacientes.

➔ CASO DE USO 9

Uso de ***arquivos*** visando obter a persistência de dados importantes em uma memória secundária. No caso em específico se faz necessário para salvar o estado do ***Controller*** e recuperá-lo em uma outra ocasião caso necessário (se o programa for iniciado novamente).