**6**

|  |  |
| --- | --- |
|  | **Disciplina**: Laboratório de Programação 2  **Professor**: Sidney Nogueira |

Lembrete:

* Formate o seu projeto de acordo com este guia <https://docs.google.com/document/d/1gfR_gt7jW0t1kXysoytQfUI0evximpDy8AubFLDU52E/edit>

Quando concluir a lista, faça um zip (RAR não é aceito) do projeto e envie o zip na sala de aula.

**Lista L6 - Organizando sócios do clube**

**Questão 1 -** Complete este [código](https://drive.google.com/file/d/1sFBFAWz70HrWIkMRmkV4bD0qN0fJ59Db/view?usp=sharing), referente a um sistema de vacinas que está descrito a seguir.

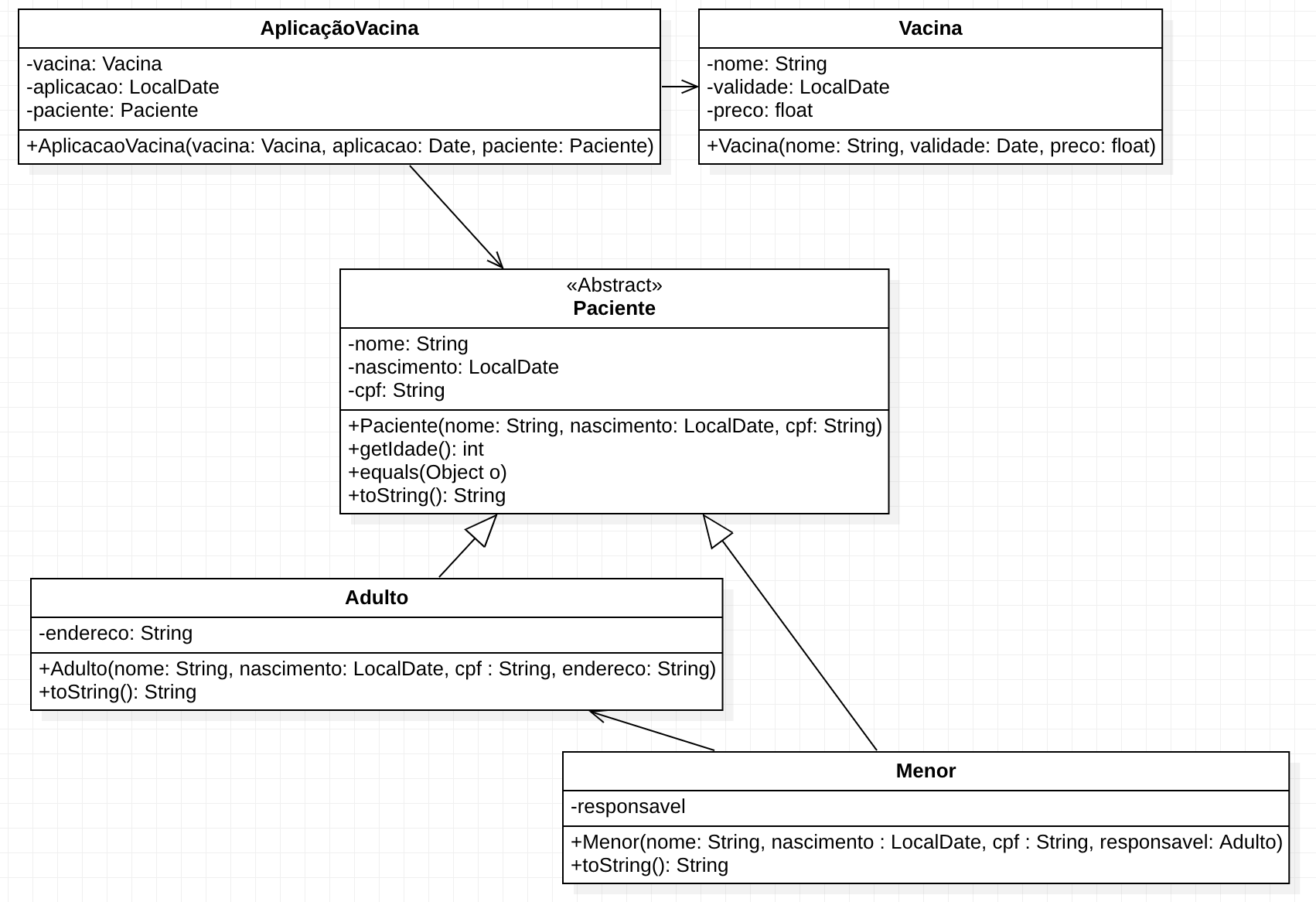
Entenda o código que já está pronto antes de fazer os itens que serão pedidos. A classe **SistemaVacinas** corresponde a fachada do sistema e segue o padrão de arquitetura em camadas. As classes da camada de negócio estão no pacote **negocio** e as classes da camada de dados no pacote **dados**. A camada de interface não foi (será) implementada.

Essa é a lista das classes já prontas no código compartilhado.

1. **negocio.Vacina** : Representa uma vacina.
2. **dados.IRepositorioVacinas** : Representa a interface do repositório de vacinas.
3. **dados.RepositorioVacinasArrayList** : Representa uma implementação da interface que usa ArrayList para armazenamento na memória.
4. **negocio.SistemaVacinas** : Fachada do sistema.
5. **negocio.TesteSistema** : Classe com método main que executar algumas funções do sistema.
6. **negocio.VacinaJaExisteException** : Exceção levantada para indicar que vacina inserida já existe.
7. **util.Util** : Classe que com método utilitário que calcula a diferença em anos entre duas datas.

Neste [link](https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/banco-exemplo/banco-exemplo-14-08-13.zip) encontra um sistema de banco implementado usando arquitetura de camadas que pode usar de referência. Na parte de material também coloquei link para a disciplina que ofereci em semestre anterior com material completo sobre Java e OO.

A figura corresponde ao modelo de entidades do sistema. Na sua implementação use o nome das classes atributos e métodos igual como aparece na figura. Um paciente tem nome, data de nascimento e cpf. Existem dois tipos de paciente, adulto ou menor. Adulto tem um endereço. Menor possui um responsável do tipo adulto. Uma vacina possui nome, data de validade e preço. A aplicação da vacina possui a vacina que foi aplicada, a data de aplicação e o paciente.



Seguem as classes que devem ser implementadas. Você pode criar quantos métodos privativos (private) precisar para auxiliar na implementação dos métodos públicos.

## Pacote **negocio**

Classe **Paciente**:

1. Classe abstrata com atributos privativos **nome**, **nascimento** e **cpf**. O tipo dos atributos está descrito no diagrama acima. A classe **LocalDate** é da biblioteca de Java do pacote **java.time**. Exemplo de criação de objetos desta classe no método main da classe **util.Util e** no método main da classe **negocio.TesteSistema**.
2. Possui métodos públicos:
   1. Construtor, conforme mostrado no modelo, que recebe três parâmetros e inicializa os atributos a partir dos parâmetros.
   2. **getIdade** que não tem parâmetro e retorna um inteiro que é a quantidade de anos do paciente. Use o método Util.diferencaAnos para calcula a diferenca de anos entre a data atual e a data de nascimento do paciente.
   3. **equals** : método herdado da Classe object. Deve ser sobrescrito de forma que retorne true se o objeto recebido como parâmetro for do tipo paciente e tiver cpf igual ao cpf do objeto this.
   4. **toString** : método herdado da Classe object. Deve ser sobrescrito de forma que retorne uma string com o nome, data de nascimento e cpf do paciente.
   5. Um método get e um método set para cada atributo privativo.

Classe **Adulto:**

1. Classe que é subtipo (herda) de **Paciente** e acrescenta o atributo **endereco**.
2. Possui métodos públicos:
   1. Construtor, conforme o modelo, que recebe quatro parâmetros e inicializa os atributos a partir dos parâmetros. Chama o construtor da superclasse (usando super).
   2. **toString** : Sobrescreve o método toString de Paciente de forma que retorne uma string com o nome, data de nascimento, cpf e endereço.
   3. Um método get e um método set para o atributo **endereco**.

Classe **Menor:**

1. Classe que é subtipo (herda) de **Paciente** e acrescenta o atributo **responsavel**.
2. Possui métodos públicos:
   1. Construtor, conforme o modelo, que recebe quatro parâmetros e inicializa os atributos a partir dos parâmetros. Chama o construtor da superclasse (usando super).
   2. **toString** : Sobrescreve o método toString de Paciente de forma que retorne uma string com o nome, data de nascimento, cpf e o nome do responsável.
   3. Um método get e um método set para o atributo **responsavel**.

Classe **AplicacaoVacina**:

1. Possui uma vacina, data de aplicação e paciente.
2. Possui métodos públicos:
   1. Construtor, conforme mostrado no modelo.
   2. Um método get e um método set para os atributos.
   3. **toString** : Sobrescreve o método toString de Object de forma que retorne uma string com as informações da vacina, data de aplicação e do paciente.

Classes de excecao:

1. **PacienteJaExisteException** : levantada quando tenta inserir um paciente com mesmo CPF de outro que já está cadastrado.
2. **ResponsavelnvalidoException** : levantada quando tenta inserir um menor que não tem um responsável maior.
3. **PacienteNaoCadastradoException** : levantada quando tenta inserir uma aplicação de vacina com um Paciente que não está cadastrado no repositório de pacientes.
4. **VacinaNaoCadastradaException** : levantada quando tenta inserir uma aplicação de vacina com uma vacina que não está cadastrado no repositório de vacinas.
5. **VacinaVencidaException** : levantada quando tenta inserir uma aplicação de vacina com uma vacina vencida.

Completar a Classe **SistemaVacinas** :

1. Adicionar atributos **RepositorioPacientesArrayList** e **RepositorioAplicacaoVacinasArrayList** que são explicados a seguir. Siga a mesma idéia que foi utilizada para o atributo private IRepositorioVacinas vacinas da classe **SistemaVacinas**.
2. Atualizar construtor para inicializar os atributos adicionados.
3. Inserir os métodos públicos:
   1. **void cadastrarPaciente(Paciente paciente) throws PacienteJaExisteException, ResponsavelnvalidoException** : este método insere no repositório de pacientes o paciente recebido como parâmetro, se não existe no repositório outro paciente com o mesmo CPF.
      1. Se existe um paciente com o mesmo CPF, o método levanta a exceção **PacienteJaExisteException**.
      2. Se o paciente inserido é um menor cujo um responsável é nulo ou tem idade menor que 18, levanta a exceção **ResponsavelnvalidoException.**
   2. **void cadastrarAplicacao(AplicacaoVacina aplicacao) throws PacienteNaoCadastradoException, VacinaNaoCadastradaException, VacinaVencidaException** : este método insere no repositório de aplicações a aplicação recebida como parâmetro, se existe no repositório um paciente com o CPF indicado, se a vacina aplicada está cadastrada no repositório e se a vacina está dentro da validade.
      1. Se não existe um paciente com o CPF, o método levanta a exceção **PacienteNaoCadastradoException**.
      2. Se não existe uma vacina levanta a exceção **VacinaNaoCadastradaException**.
      3. Se a vacina está fora da validade levanta a exceção **VacinaVencidaException**.
   3. **Paciente[] todosPacientes()** : retorna vetor com todos os pacientes cadastrados.
   4. **AplicacaoVacina[] todasAplicacoes()** : retorna vetor com todas as aplicações cadastradas.

Completar a Classe **TesteSistema**:

1. Para cadastrar pacientes e aplicações de vacinas.
2. Para imprimir os pacientes e as aplicações existentes.

## Pacote **dados**

Antes de fazer estas classes veja como foi implementado **IRepositorioVacinas** e **RepositorioVacinasArrayList.** Vai usar a mesma idéia.

Você pode estranhar que só existe um repositório para **Paciente**, uma vez que existem dois tipos de paciente. Para entener como isto é possível, veja o sistema do banco linkado acima. No sistema do banco existe uma classe do tipo **ContaAbstrata** que possui vários subtipos. Os repositórios naquele sistema trabalham apenas com o parâmetro do tipo **ContaAbstrata**.

Interface **IRepositorioPacientes**:

1. Possui a assinatura dos métodos:
   1. **void Paciente buscaPaciente(String cpf)** : que recebe um CPF e procura no repositório por um objeto paciente com o mesmo CPF. Retorna o objeto do paciente que foi encontrado. Retorna nulo se nenhum paciente possui o CPF informado.
   2. **void inserePaciente(Paciente paciente)** : que insere um paciente no repositorio caso não existe outro paciente com o mesmo CPF.
   3. **Paciente[] todos()** : que retorna um vetor com todos os pacientes do repositório.

Classe **RepositorioPacientesArrayList :**

1. Possui um atributo do tipo **ArrayList<Paciente>** para armazenar os pacientes.
2. Possui construtor público sem parâmetros que inicializa o atributo.
3. Implementa os métodos da interface **IRepositorioPacientes**

Classe **IRepositorioAplicacaoVacinas:**

1. Possui a assinatura dos métodos:
   1. **AplicacaoVacina[] buscaAplicacao(String cpf)** : que recebe o CPF de um paciente e procura no repositório por aplicações de vacina que sejam de um paciente que tem o CPF informado. Retorna um array com objetos do tipo AplicacaoVacina onde o atributo paciente tem o CPF informado.
   2. **void insereAplicacao(AplicacaoVacina aplicacao)** : que insere uma aplicação no repositorio.
   3. **AplicacaoVacina[] todas()** : que retorna um vetor com todas as aplicações do repositório.

Classe **RepositorioAplicacaoVacinaArrayList :**

1. Possui um atributo do tipo **ArrayList<AplicacaoVacina>** para armazenar as aplicações.
2. Possui construtor público sem parâmetros que inicializa o atributo.
3. Implementa os métodos da interface **IRepositorioPacientes**