**Nomes:**

Bruno Tavares

Felipe Farias

Guilherme Dias

Natália Georgetti

Quézia Quirino

**Exercício 1:**



**Exercício 2:**



**Exercício 3:**

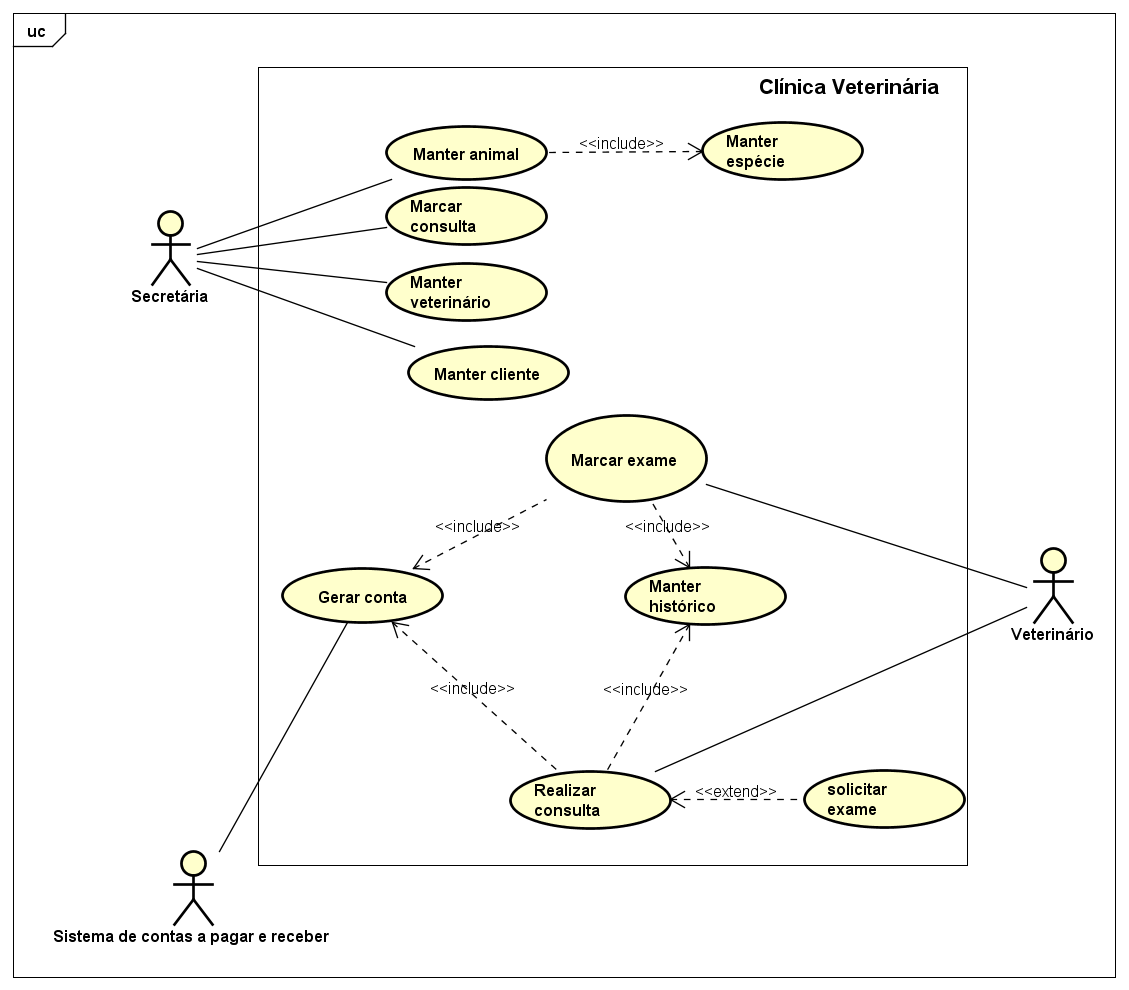


**Exercício 4:**

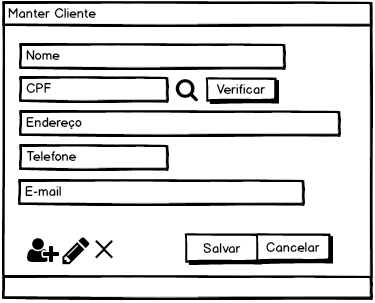


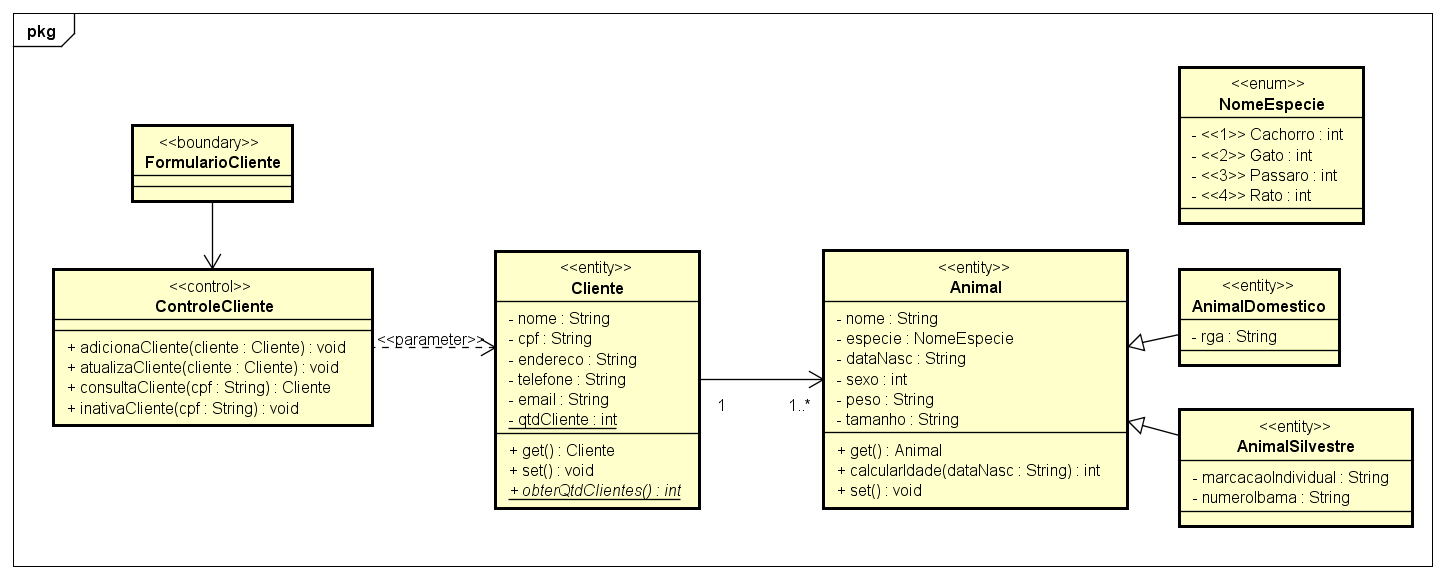
**Exercício 5:**

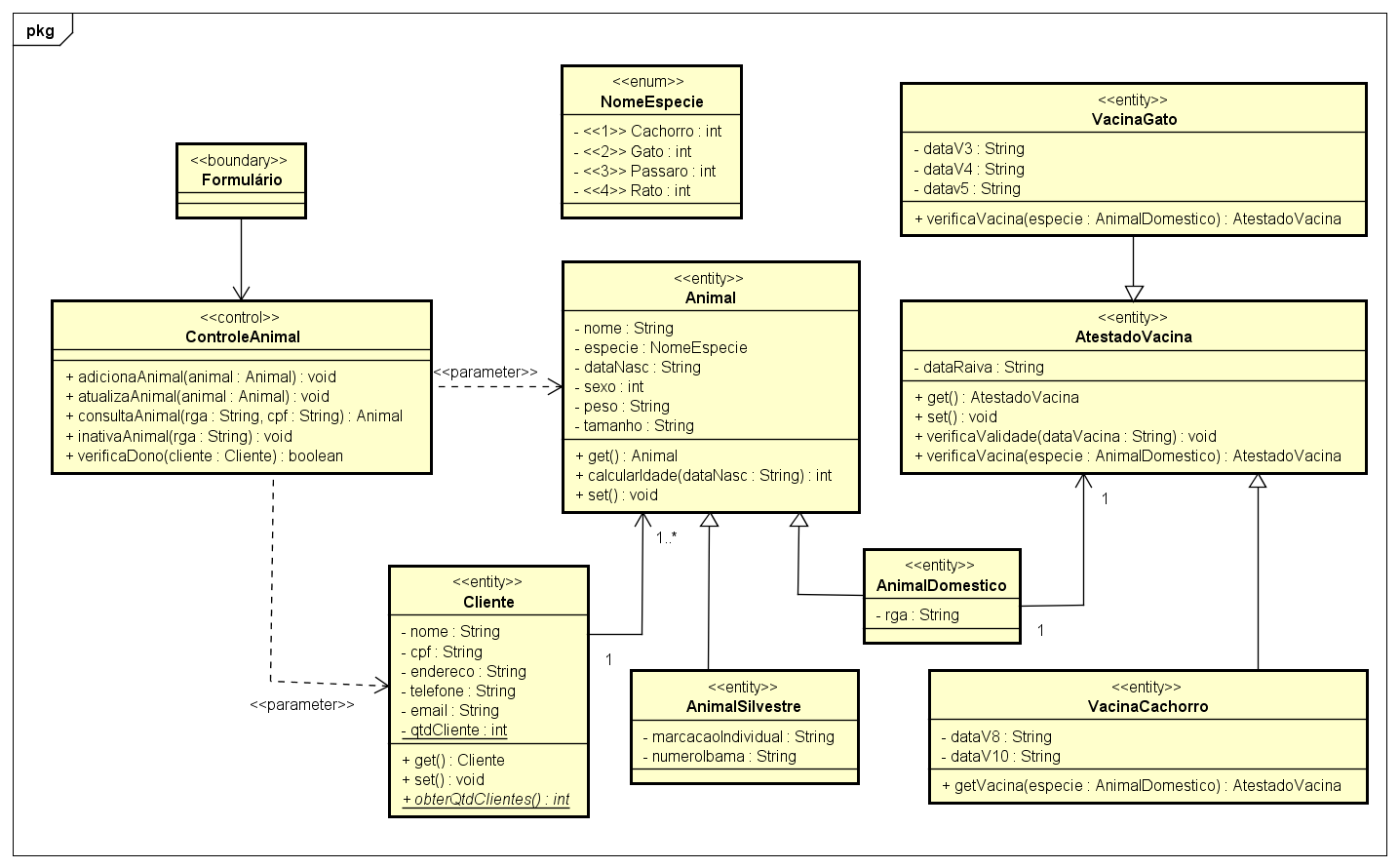
**Exercício 6:** 

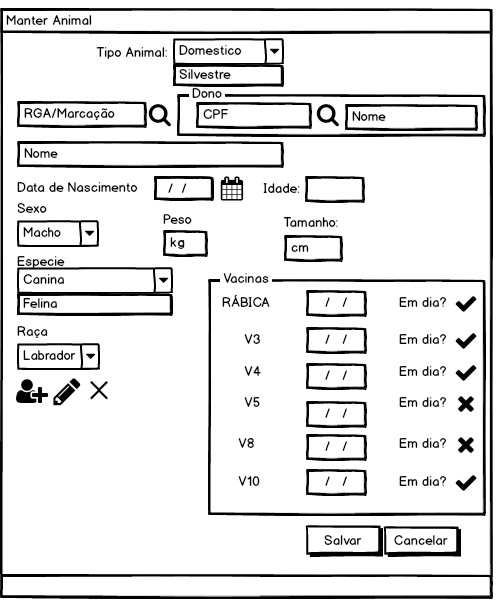
**Exercício 7:** 

**Exercício 8:**

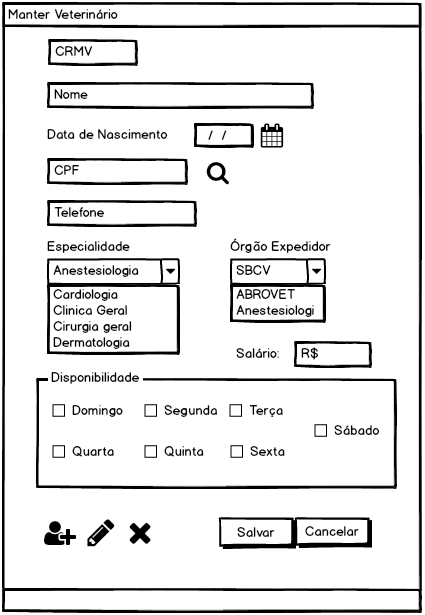


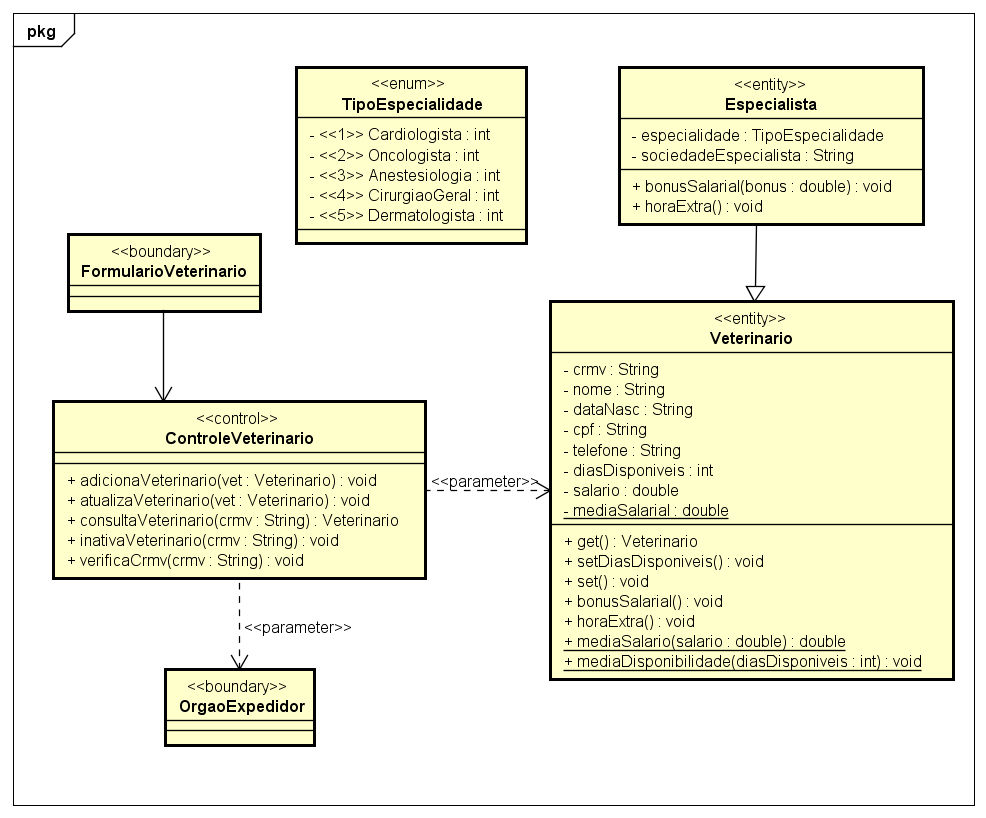


**Exercício 9:** 

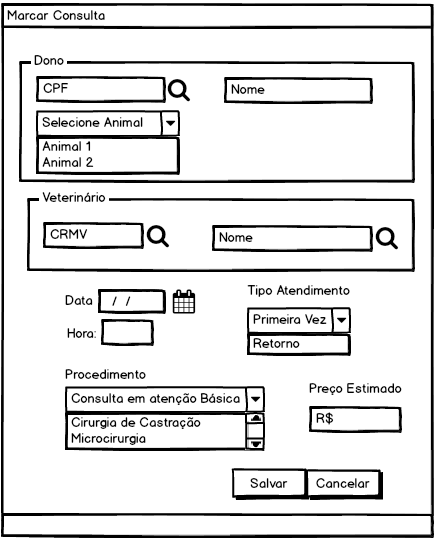


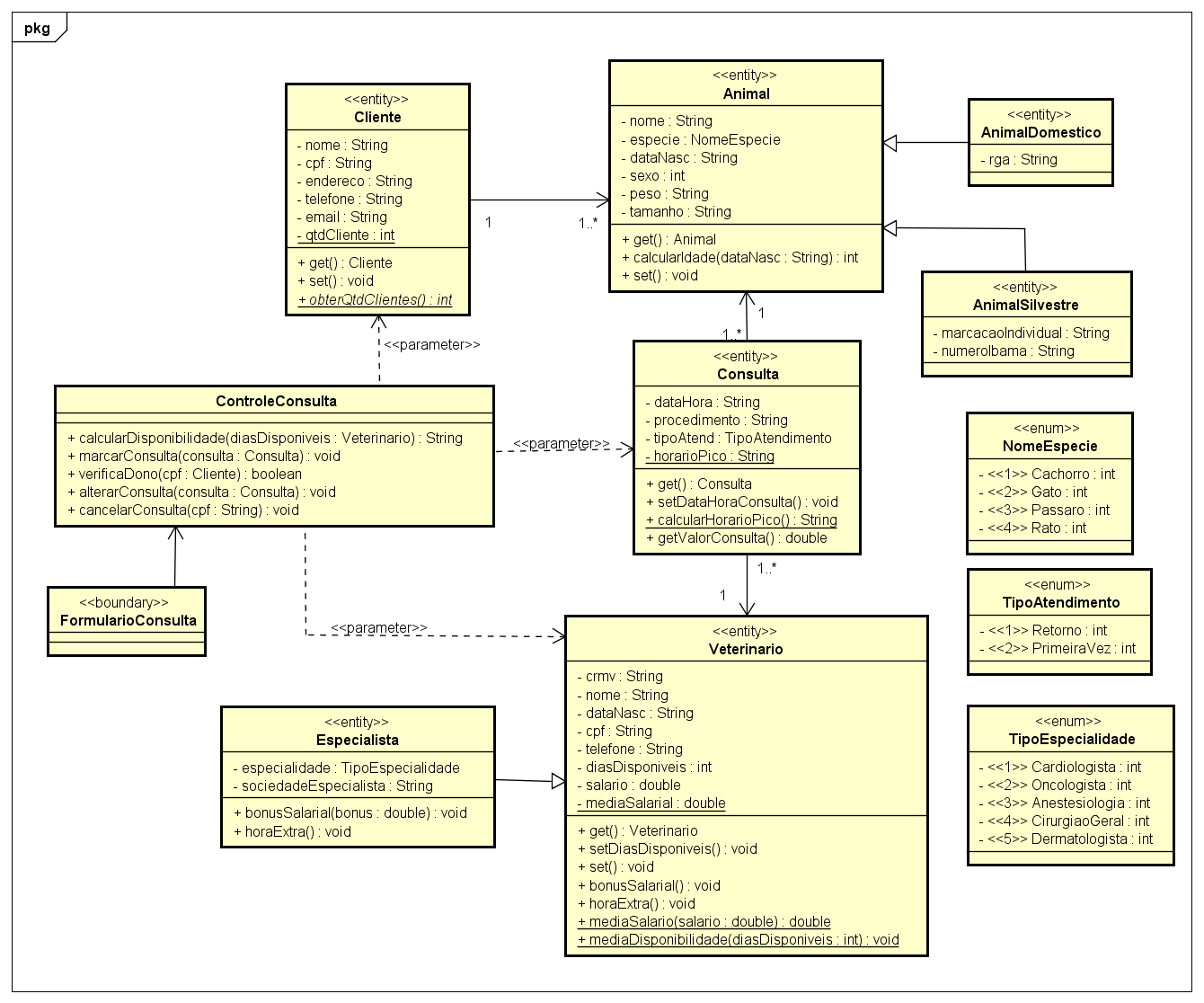
**Exercício 10:**



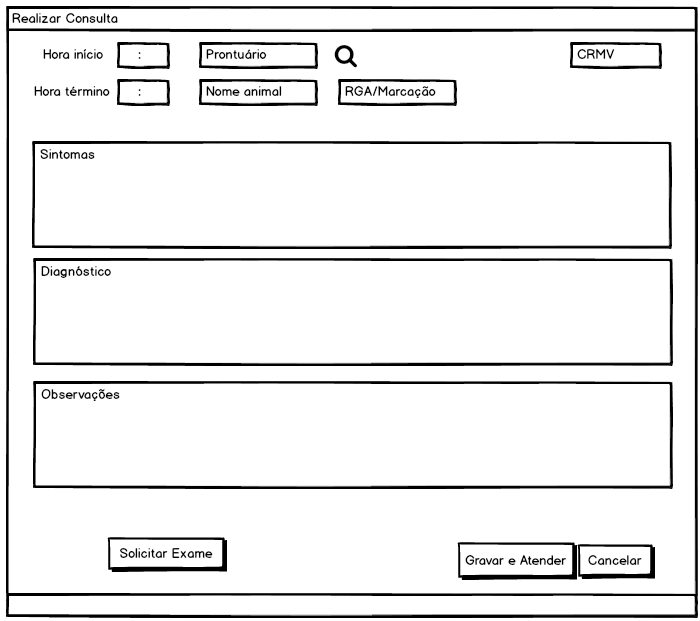


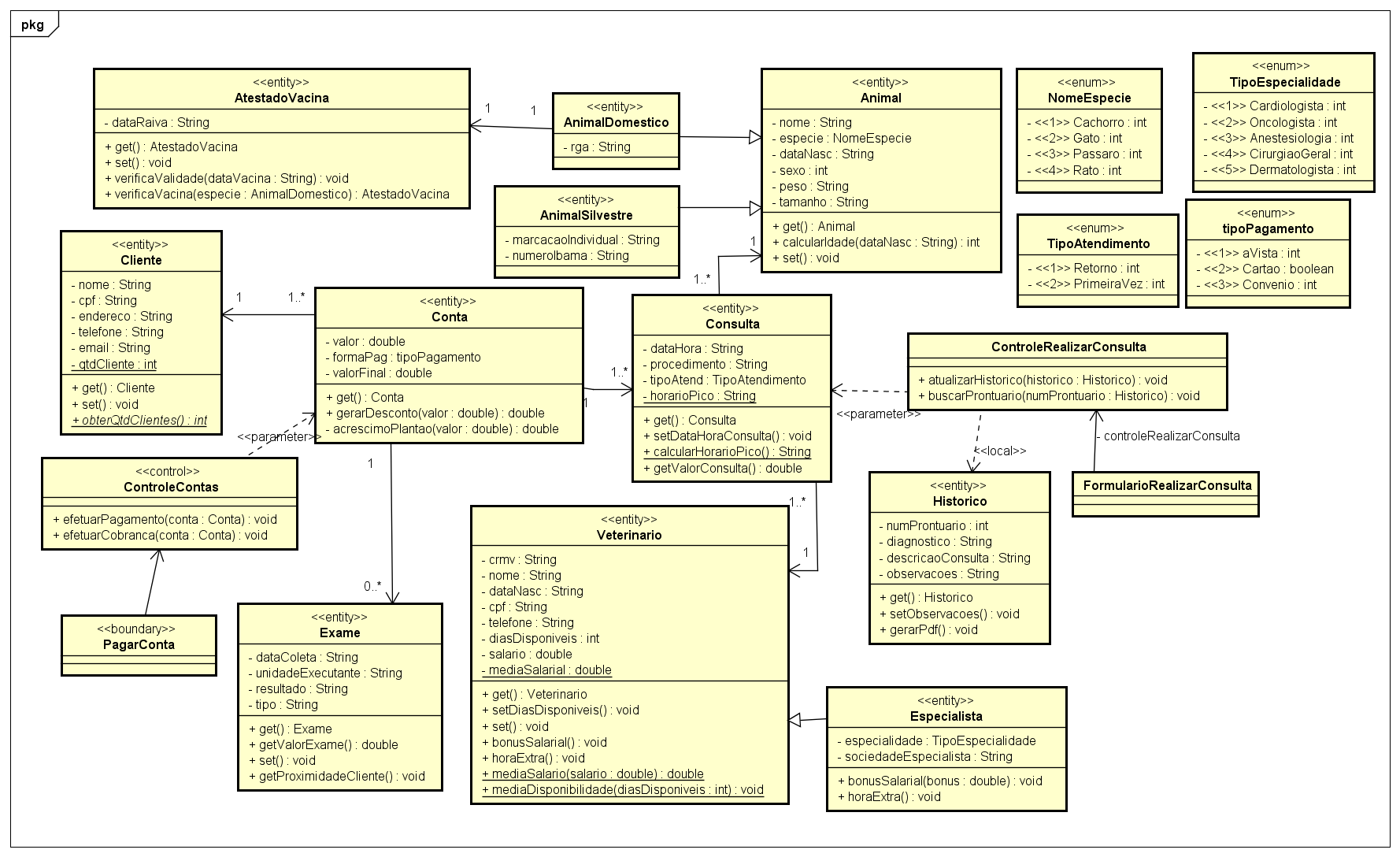
**Exercício 11:**



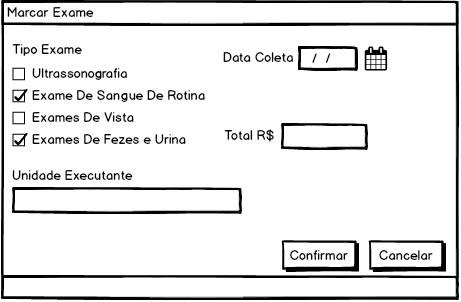


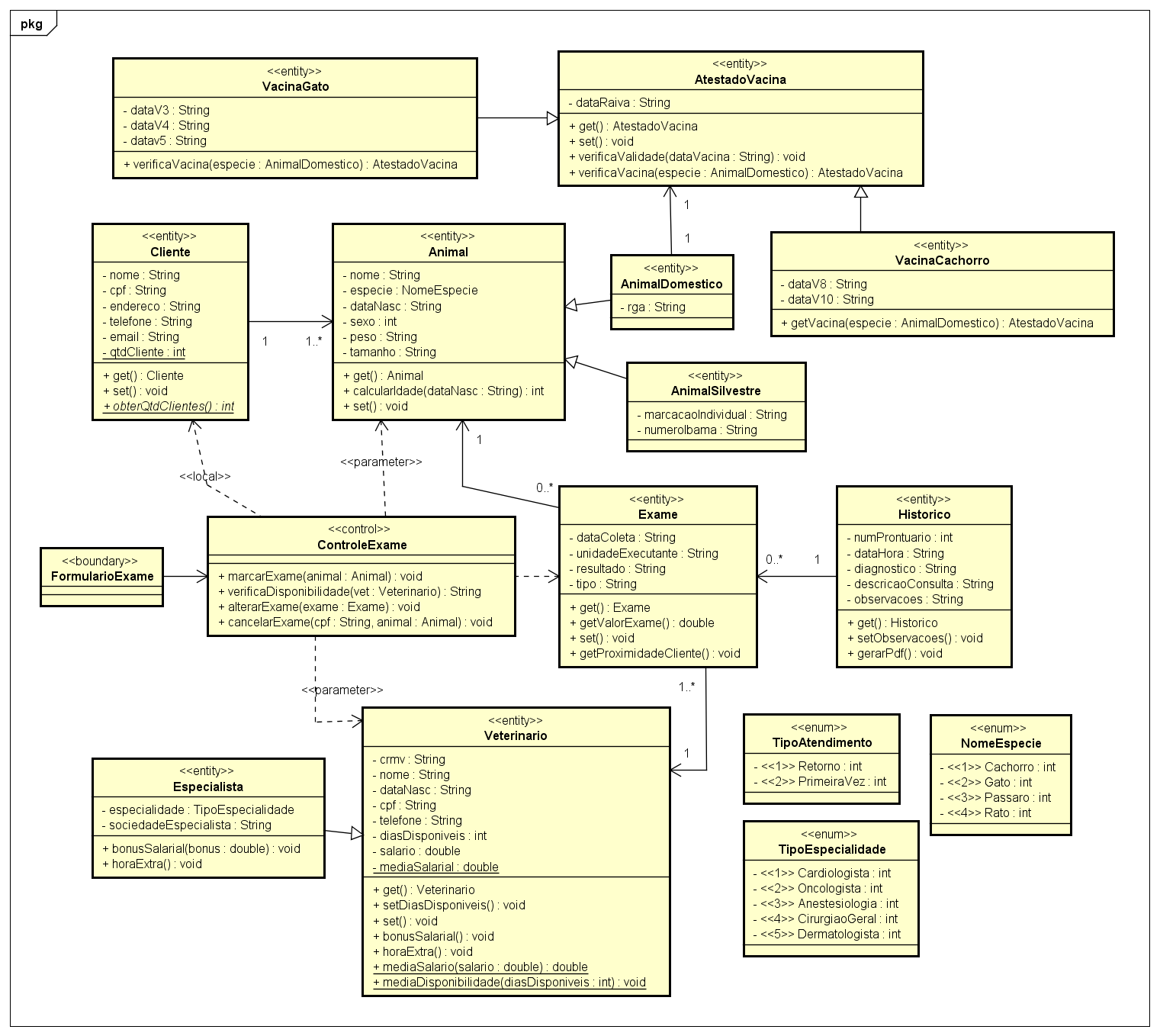
**Exercício 12:**



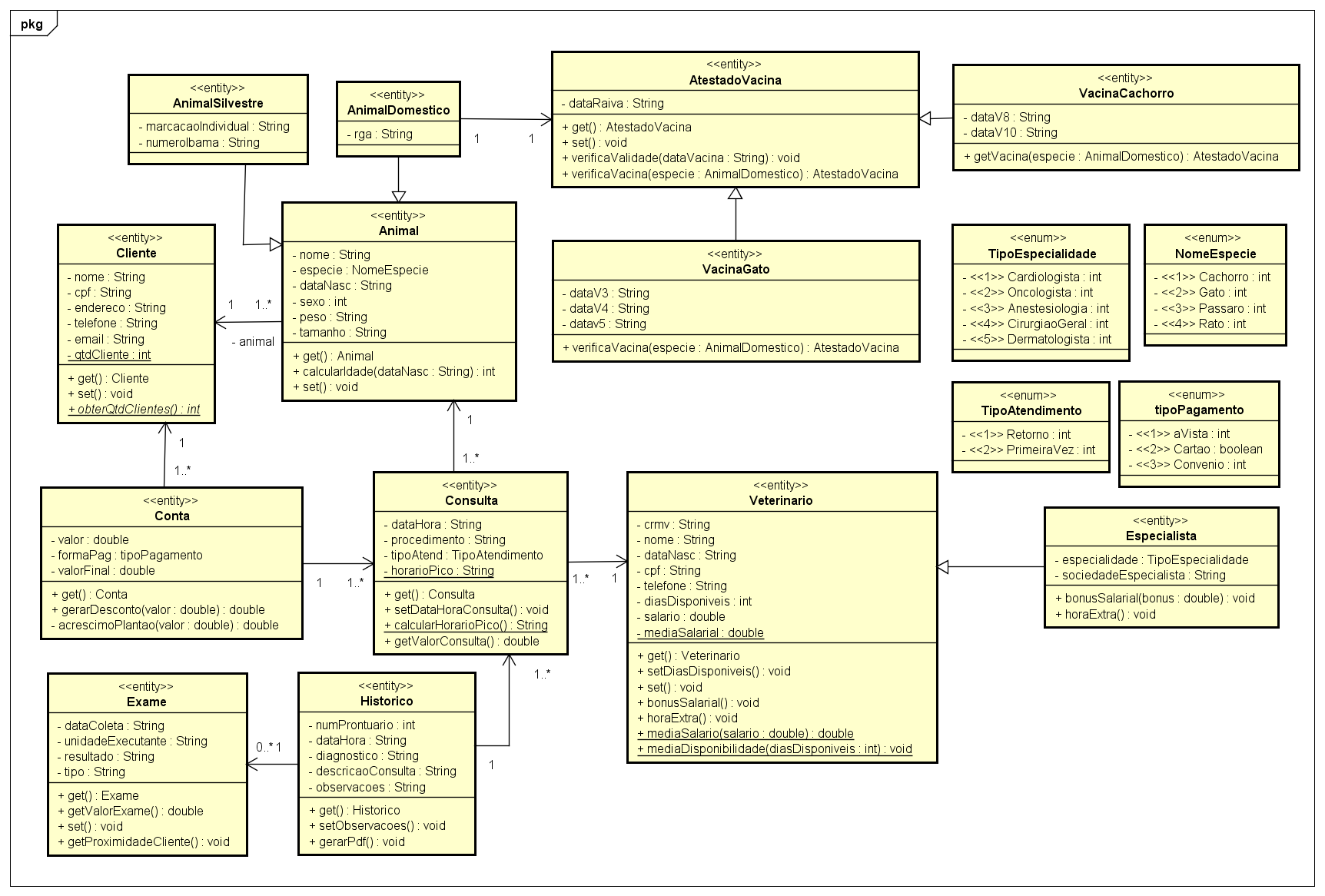


**Exercício 13:**





**Exercício 14:**



**Exercício 15** – Mais coesa é consulta, pois está amarrada em Animal e Veterinario, agrupamento ocorre só porque elas realmente precisam estar juntas para contribuir com algo definido, realizando assim em conjunto as tarefas necessárias. Menos coesa é Historico, pois ela é um histórico de consulta e exame, então ela tem conhecimento de atributos das classes Consulta e Exame.

**Exercício 16** – A classe mais acoplada é Consulta, pois depende de Animal e Veterinario para a realização de suas responsabilidades. A classe menos acoplada é Cliente, pois não depende outras classes para funcionar.

**Exercício 17** – O método polimórfico bonusSalarial existe, pois há um aumento no bônus quando o veterinário tem uma especialidade.

O método verificaVacina para saber qual atestado buscar através do atributo espécie da classe AnimalDomestico.

Não violam, pois, a filha pode se passar por mãe.

**Exercício 18** – public class Especialista extends Veterinario {

public double bonusSalarial(double salario){

...

}

}

Public class VacinaGato extends AtestadoVacina {

public boolean verificaVacina (AnimalDomestico especie) {

...

}

}

**Exercício 19** – A classe TipoEspecialidade foi feita para evitar erros na hora de inserção no banco e para limitar à apenas as especialidades que o domínio atende;

A classe TipoAtendimento limita a somente dois tipos de atendimento, que são Primeira Vez e Retorno;

A classe TipoEspecie limita a seleção somente aos tipos de animais que são atendidos no domínio;

A classe TipoPagamento aceita apenas as formas de pagamento aceitas pelo domínio;

**Exercicio 20** – public enum *TipoEspecialidade* {

*Cardiologista*, *Oftalmologista*, *Anestesiologia*, Ci*rurgiaoGeral*, *Dermatologia*;

}

public enum *TipoAtendimento* {

*Retorno*, *PrimeiraVez*;

}

public enum *TipoEspecie* {

*Cachorro*, *Gato*, *Passaro*, *Roedor*; }

public enum *TipoPagamento* {

*aVista*, *Cartao*, *Convenio*;

}

**Exercício 21** – O atributo mediaSalarial na classe Veterinario, guarda uma

média de salário entre todos os objetos do tipo Veterinario;

O atributo horarioPico na classe Consulta foi feito, pensando em qual horário a clínica atende mais pacientes, podendo até mesmo remanejar o pessoal de forma a atender a demanda.

O atributo QtdClientes na classe Cliente, guarda o valor do total de cliente que a clínica tem em seu banco de dados.

O Método mediaSalarial calcula a média salarial de todos os veterinários.

O método mediaDisponibilidade calcula qual a média de disponibilidade de todos os veterinários, para saber qual dia tem mais veterinário disponível.

O método obterQtdClientes na classe Cliente calcula a quantidade de clientes que possui no banco de dados.

**Exercício 22** – private static double mediaSalarial;

private static int quantidadeClientes;

private static String horarioPico;

public static double mediaSalarial() {

...

}

public static String calcularHorarioPico() {

...

}

public static int obterQtdClientes() {

...

}

**Exercício 23** – A vantagem dessa dependência é aumento no desempenho, a desvantagem é queda no encapsulamento e aumento no acoplamento.

**Exercício 24** – public class Animal {

private Cliente cliente;

...

}

public class Consulta {

private Animal animal;

private Veterinario veterinario;

}

public class Historico {

private Consulta consulta;

private Exame exame;

}

public class Conta {

private Consulta consulta;

private Exame exame;

private Cliente cliente;

}

public class AnimalDomestico extends Animal {

private AtestadoVacina atestadoV;

}

**Exercício 25** – A vantagem é aumento no encapsulamento e queda no acoplamento, a desvantagem é o baixo desempenho.

**Exercício 26** – public class ControleCliente {

public void adicionaCliente(Cliente c) {

...

}

public void consultaCliente(Cliente c) {

...

}

public void atualizaCliente(Cliente c) {

...

}

public void inativaCliente(Cliente c) {

...

}

}

public class ControleConta {

public void efetuarPagamento(Conta conta) {

...

}

public void efetuarCobranca(Conta conta) {

...

}

}

public class ControleAnimal {

public void adicionaAnimal(Animal a) {

...

}

public void consultaAnimal(Animal a) {

...

}

public void atualizaAnimal(Animal a) {

...

}

public void inativaAnimal(Animal a) {

...

}

public boolean verificaDono(Cliente c) {

...

}

public class ControleVeterinario {

public void adicionaVeterinario(Veterinario v) {

...

}

public void consultaVeterinario(Veterinario v) {

...

}

public void atualizaVeterinario(Veterinario v) {

...

}

public void inativaVeterinario(Veterinario v) {

...

}

public void verificaCrmv(String crmv) { ... } }

public class ControleConsulta {

public String calculaDisponibilidade(Veterinario v) {

...

}

public void marcarConsulta (Consulta c) {

...

}

public boolean verificaDono(Cliente c) {

...

}

public void alterarConsulta(Consulta c) {

...

}

public void cancelarConsulta(String cpf) {

...

}

}

public class ControleExame {

public voic marcarExame(Animal a) {

...

}

public String verificaDisponibilidade(Veterinario v) {

...

}

public void alterarExame(Exame e) {

...

}

public void cancelarExame(String cpf, Animal a) {

...

} }

public class ControleRealizarConsulta {

public void buscarProntuario(Historico h) {

...

} }

**Exercício 27** – A vantagem de guardar a instância do objeto em variável local vem quando se precisa acessar o mesmo objeto muitas vezes, assim evitando de ficar criando o objeto toda vez que for manipulá-lo.

**Exercício 28** – public class ControleCliente {

public void adicionaCliente() {

Cliente c = new Cliente();

...

}

public void consultaCliente() {

Cliente c = new Cliente();

...

}

public void atualizaCliente() {

Cliente c = new Cliente();

...

}

public void inativaCliente() {

Cliente c = new Cliente();

...

}

}

public class ControleConta {

public void efetuarPagamento() {

Conta c = new Conta();

...

}

public void efetuarCobranca() {

Conta c = new Conta();

...

}

}

public class ControleAnimal {

public void adicionaAnimal() {

Animal a = new Animal();

...

}

public void consultaAnimal() {

Animal a = new Animal();

...

}

public void atualizaAnimal() {

Animal a = new Animal();

...

}

public void inativaAnimal() {

Animal a = new Animal();

...

}

public boolean verificaDono() {

Cliente c = new Cliente();

...

}

public class ControleVeterinario {

public void adicionaVeterinario() {

Veterinario v = new Veterinario();

...

}

public void consultaVeterinario() {

Veterinario v = new Veterinario();

...

}

public void atualizaVeterinario() {

Veterinario v = new Veterinario();

...

}

public void inativaVeterinario() {

Veterinario v = new Veterinario();

...

}

}

public class ControleConsulta {

public String calculaDisponibilidade() {

Veterinario v = new Veterinario();

...

}

public void marcarConsulta () {

Consulta c = new Consulta();

...

}

public boolean verificaDono() {

Cliente c = new Cliente();

...

}

public void alterarConsulta() {

Consulta c = new Consulta();

...

}

}

public class ControleExame {

public voic marcarExame() {

Animal a = new Animal();

...

}

public String verificaDisponibilidade() {

Veterinario v = new Veterinario();

...

}

public void alterarExame() {

Exame e = new Exame();

...

}

public void cancelarExame() {

Animal a = new Animal();

...

}

}

public class ControleRealizarConsulta {

public void buscarProntuario(Historico h) {

Historico h = new Historico();

...

}

}

public class ControleRealizarConsulta {

public void atualizarHistorico() {

Historico h = new Historico();

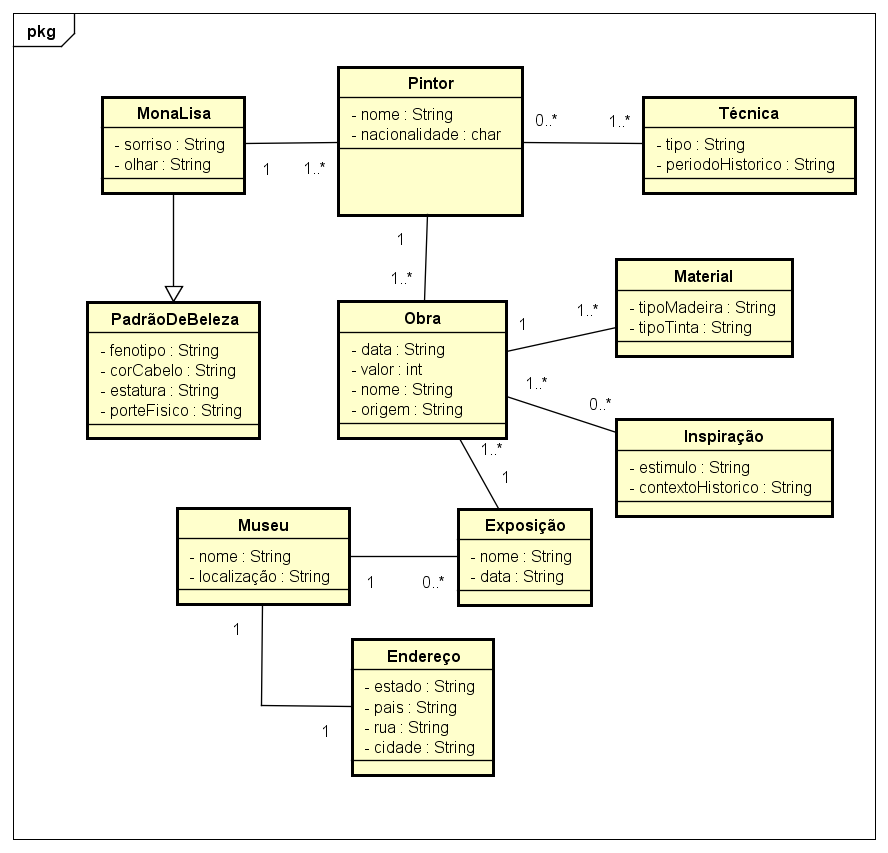
...

}

}

**Parte B:**

**Exercício 1:**



**Exercício 2:**

