Nome: Bruno Tavares

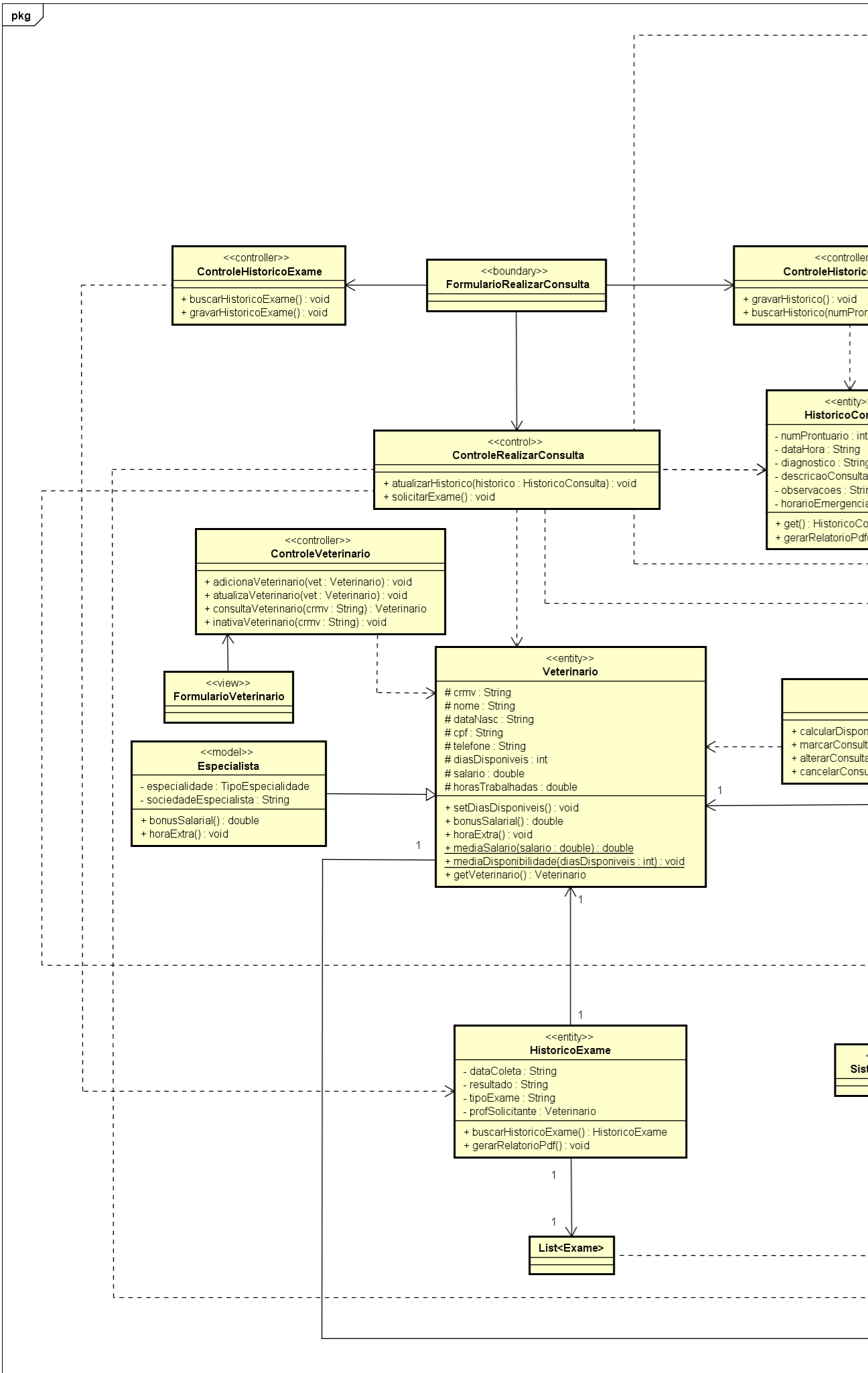
Nome: Felipe Lopes

Nome: Guilherme Dias

Nome: Natalia Georgetti

Nome: Quézia Quirino

1.


pkg
<<controller>>
ControleHistoricoExame
+ buscarHistoricoExame() : void
+ gravarHistoricoExame() : void
<<boundary>>
FormularioRealizarConsulta
<<control>>
ControleRealizarConsulta
+ atualizarHistorico(historico : HistoricoConsulta) : void
+ solicitarExame() : void
<<controller>>
ControleVeterinario
+ adicionaVeterinario(vet : Veterinario) : void
+ atualizaVeterinario(vet : Veterinario) : void
+ consultaVeterinario(crmv : String) : Veterinario
+ inativaVeterinario(crmv : String) : void
<<view>>
FormularioVeterinario
<<entity>>
Veterinario
crmv : String
nome : String
dataNasc : String
cpf : String
telefone : String
diasDisponiveis : int
salario : double
horasTrabalhadas : double
+ setDiasDisponiveis() : void
+ bonusSalarial() : double
+ horaExtra() : void
+ mediaSalario(salario : double) : double
+ mediaDisponibilidade(diasDisponiveis : int) : void
+ getVeterinario() : Veterinario
<<model>>
Especialista
- especialidade : TipoEspecialidade
- sociedadeEspecialista : String
+ bonusSalarial() : double
+ horaExtra() : void
<<entity>>
HistoricoExame
- dataColeta : String
- resultado : String
- tipoExame : String
- profSolicitante : Veterinario
+ buscarHistoricoExame() : HistoricoExame
+ gerarRelatorioPdf() : void
List<Exame>
1
1
1
1
1
1
1

2. Não apresentam, pois, todas as subclasses são exclusivas e não tem possibilidade de se transformar em outra classe, sendo assim não há necessidade de criar várias possibilidades de subclasse.

3. Todas as heranças são completas e disjuntas. Completa, pois não existe outras subclasses. Disjunta, pois são classes exclusivas uma não pode sobrepor a outra.

4. Quando o objeto é mandado por parâmetro ou variável local, há economia de memória, pois assim que o método finaliza aquele objeto instanciado torna-se elegível para o Garbage Collector deslocar da memória, nota-se também que há aumento no encapsulamento, abaixando o acoplamento, porém, o desempenho cai.

5.

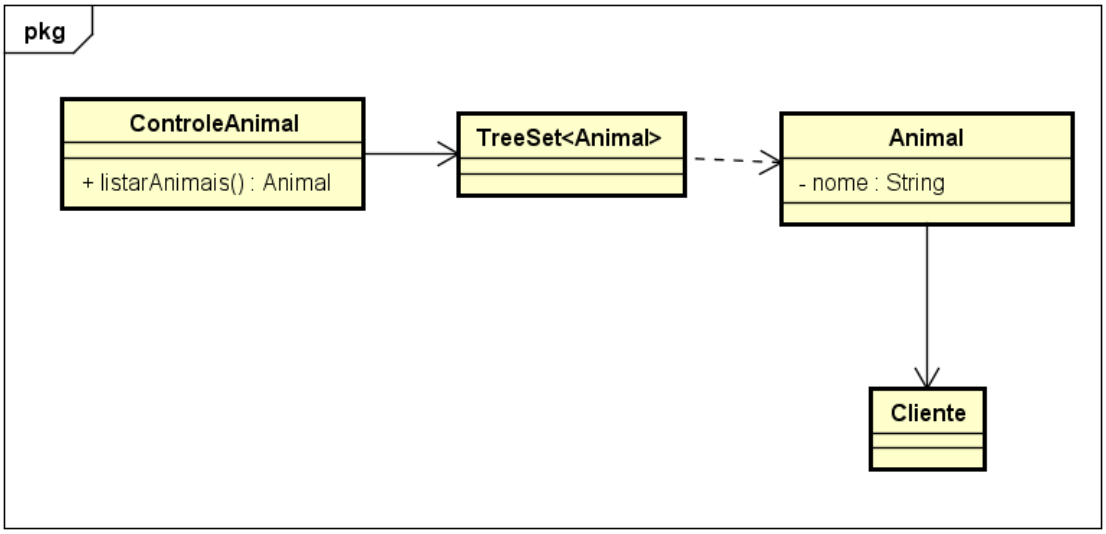
```
public class ControleCliente {
  public void adicionarCliente(Cliente c) {
    ...
  }
}

public class ControleCliente {
  public void adicionarCliente() {
    Cliente c = new Cliente();
     ...
  }
}
```

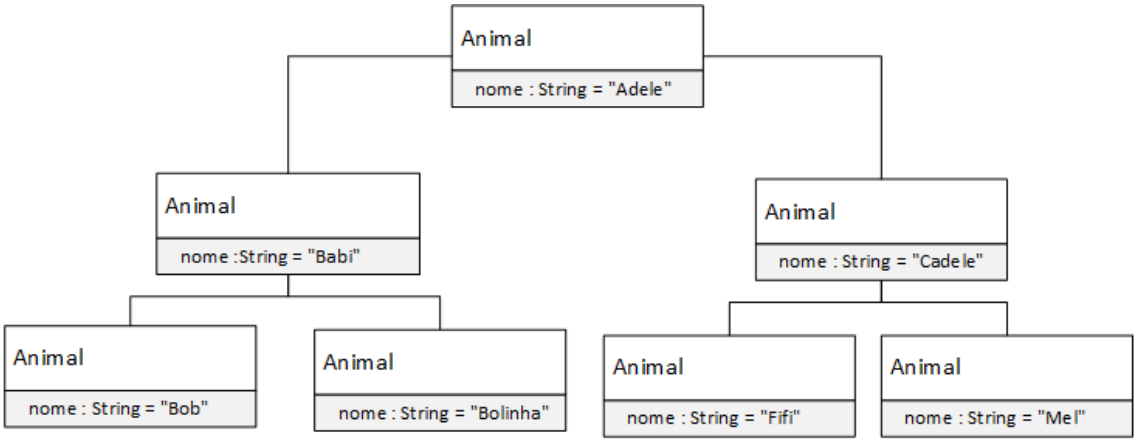
6. Foi escolhida a estrutura de List por ser um domínio que muitos dados podem ser repetidos. Não foi encontrado alguma relação de muitos para muitos que seria resolvida com a estrutura de Set.

7.

```
public class Cliente {
  List<Animal> animal = new LinkedList<> ();
}
public class Animal {
  List<Consulta> consulta = new LinkedList<> ();
}
public class Historico {
  List<Exame> exame = new LinkedList<> ();
  List<Consulta> consulta = new LinkedList<> ();
}
```

8. A estrutura TreeSet compara os elementos e os ordena, os elementos são ordenados à medida que são adicionados na lista.

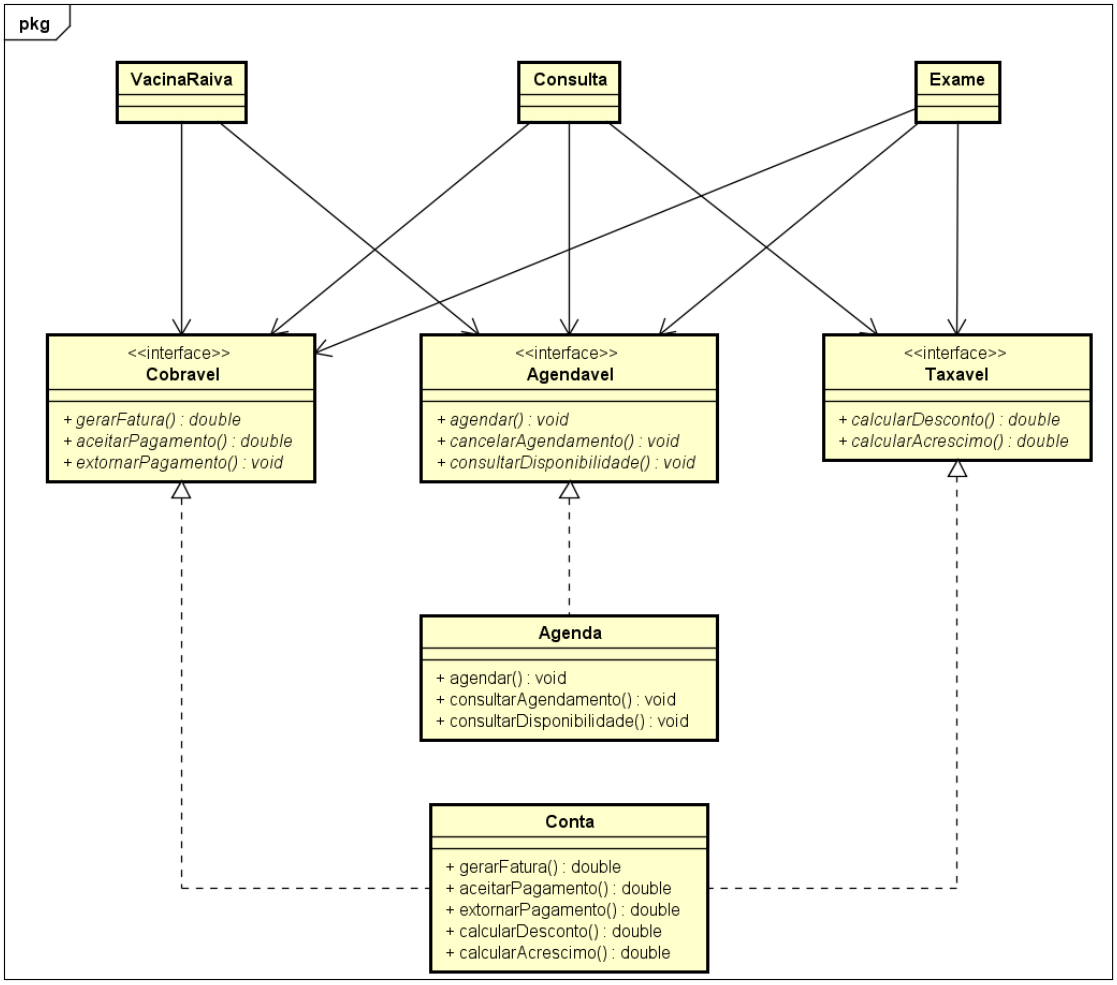




9.

```
public class Cliente {
   TreeSet<Animal> animal = new TreeSet<Animal> (); }
```

10.



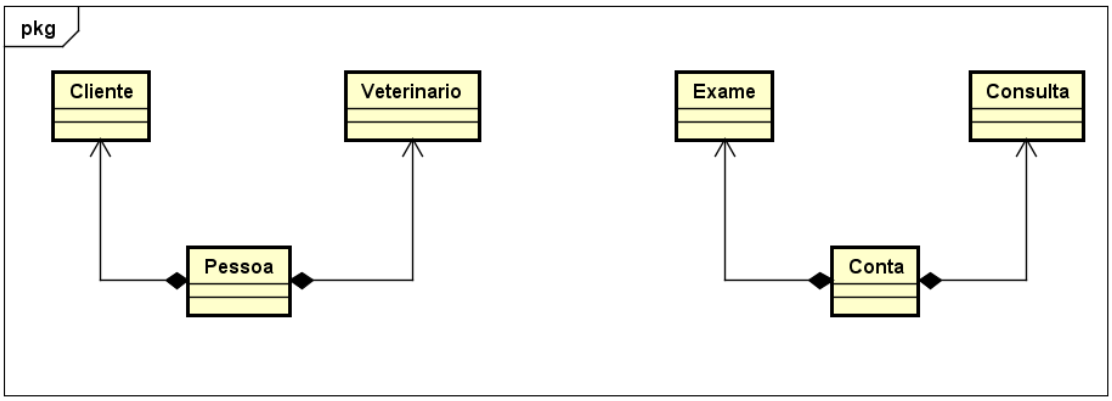
11.

```
public interface Agendavel {
	public void agendar();
	public void cancelarAgendamento();
	public void consultarDisponibilidade();
}
public interface Cobravel {
	public double gerarFatura(double valor);
	public double aceitarPagamento(double valor);
	public void extornarPagamento();
}
public interface Taxavel {
	public double calcularDesconto(double valor);
	public double calcularAcrescimo(double valor);
}
```

```
public class Conta implements Cobravel, Taxavel {
	@Override
	public double calcularDesconto(double valor) {
		return valor * 0.9;
	}
	@Override
	public double calcularAcrescimo(double valor) {
		return valor * 1.10;
	}
	@Override
	public double gerarFatura(double valor) {
		return valor;
	}
	@Override
	public double aceitarPagamento(double valor) {
		return valor;
	}
	@Override
	public void extornarPagamento() {
		...
	}
}

public class Agenda implements Agendavel {
	@Override
	public void agendar() {
		System.out.println("Consulta agendada");
	}
	@Override
	public void cancelarAgendamento() {
		System.out.println("Consulta cancelada");
	}
	@Override
	public void consultarDisponibilidade() {
	}
}
public class Consulta {
	private Agendavel agendavel;
	private Cobravel cobravel;
	private Taxavel taxavel;
}
public class Exame {
	private Agendavel agendavel;
	private Cobravel cobravel;
}
public class VacinaRaiva {
	private Cobravel cobravel;
	private Agendavel agendavel;
}
```

12. Uma pessoa pode ser veterinária e cliente ao mesmo tempo e conta pode ser conta de consulta e conta de exame ao mesmo tempo

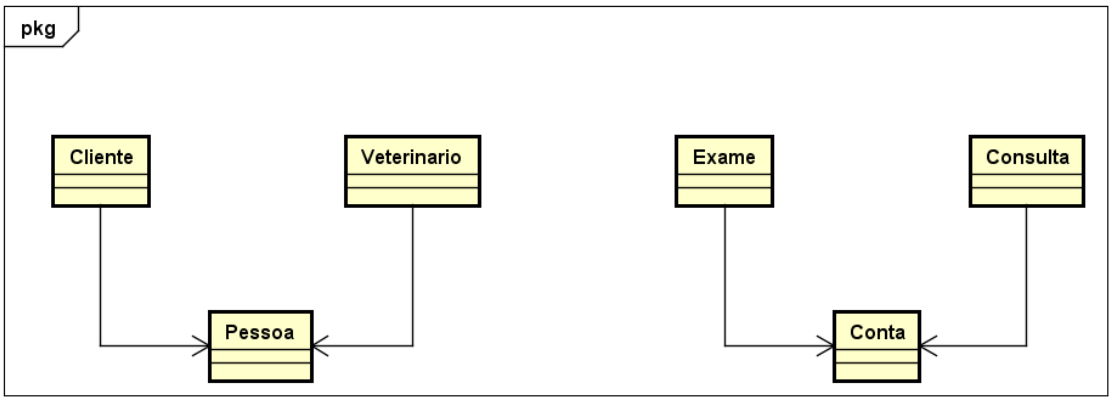
.



13.

```
public class Pessoa {
   public void novoVeterinario(){
      Veterinario v = new Veterinario();
      ...
    }
    public void novoCliente(){
      Cliente = new Cliente();
      ...
    }
}

public class Conta {
   public void gerarContaExame() {
      Exame e = new Exame();
      ...
   }
   public void gerarContaConsulta () {
      Consulta c = new Consulta();
      ...
   }
```

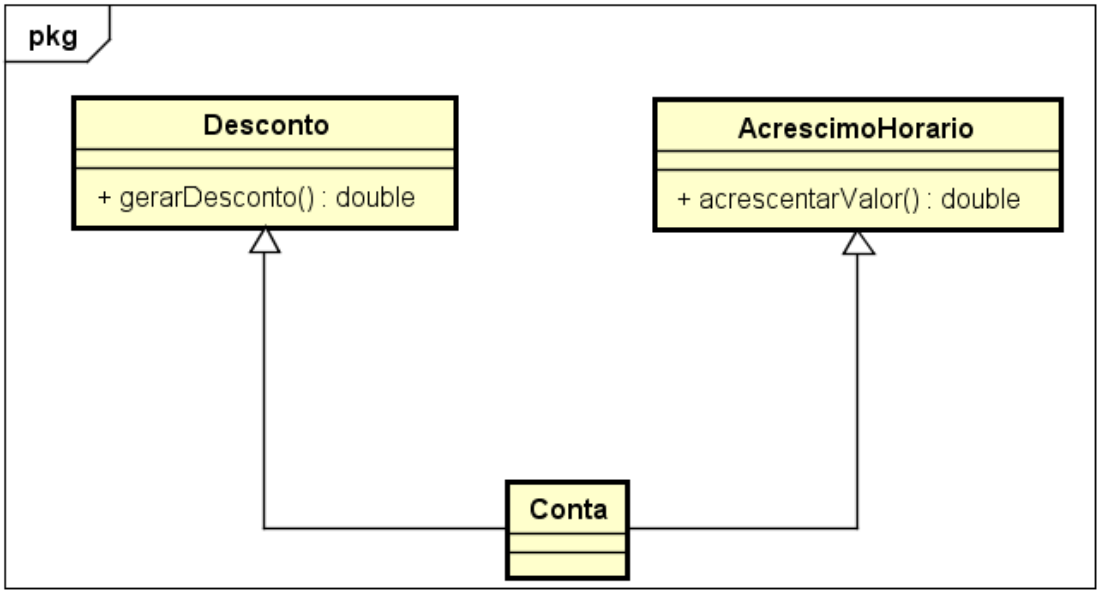
14. A instância do todo ficará na estrutura do código das duas classes partes.


pkg
Cliente
Veterinario
Exame
Consulta
Pessoa
Conta

15. public class Pessoa {

```
   ...
}
public class Veterinario {
   Pessoa pessoa;
}
public class Cliente {
   Pessoa pessoa;
}
public class Conta {
   ...
}
public class Exame {
   Conta conta;
}
public class Consulta {
   Conta conta;
}
```

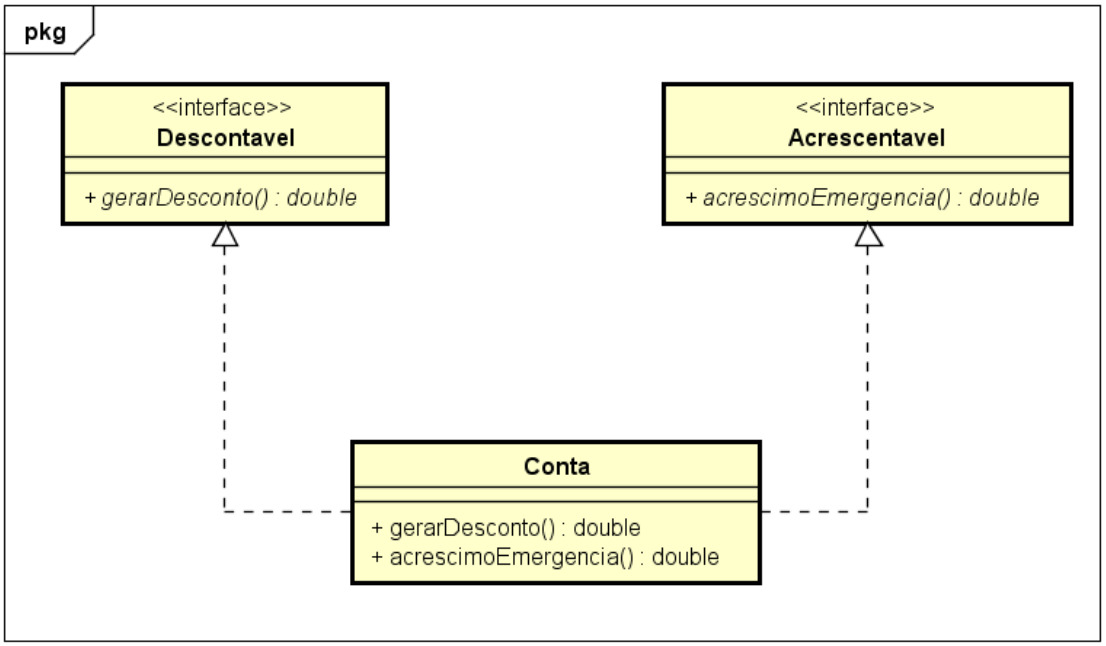
16. A classe conta necessita de dois métodos que estão implementados em classes distintas.



17.

```
class Desconto {

      public:
        double valorDesconto;
        double gerarDesconto(double);
        };
class AcrescimoHorario {
    public:
    double valorAcrescimo;
    double AcrescimoHorario(double);
};
class conta : public Desconto, public AcrescimoHorario{
    double gerarDesconto(double);
    double AcrescimoHorario(double);
};
```

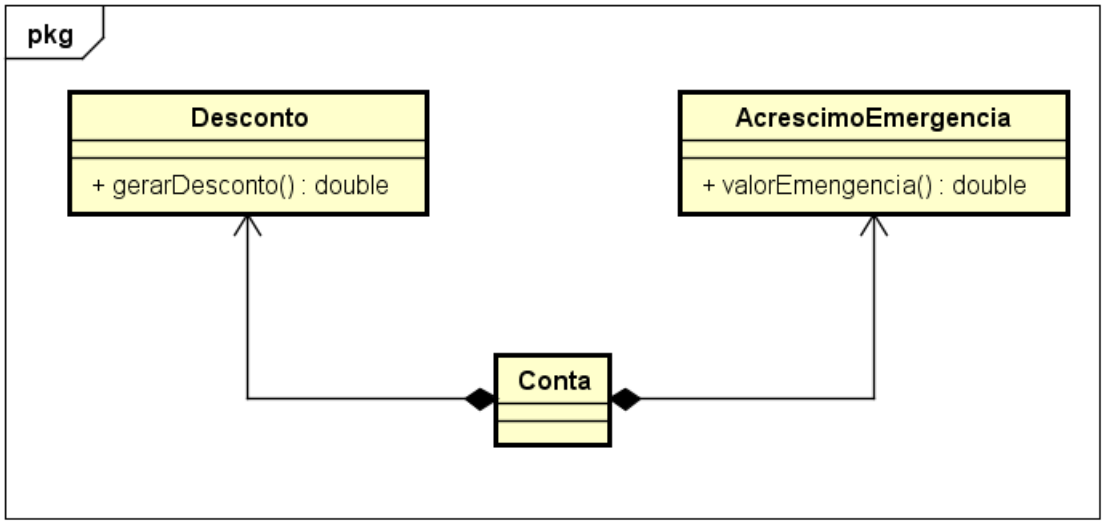
18. As interfaces distintas consistem em métodos que serão utilizados pela a classe conta, sendo assim serão feitas duas realizações para assim a classe conta implementar o método concreto.



19.

```
public interface Acrescentavel {
   public double acrescimoEmergencia();
}
public interface Descontavel {
   public double gerarDesconto();
}
public class Conta implements Descontavel, Acrescentavel {
    private static double valorFinal;
    private double valorDesconto;
    private double valorAcrescimo;

@Override
public double gerarDesconto() {
      …
   }
@Override
public double acrescimoHorario() {
      …
   }
}
```

20. A delegação possibilita contornar a limitação de algumas linguagens de programação no quesito de herança múltipla.



21.

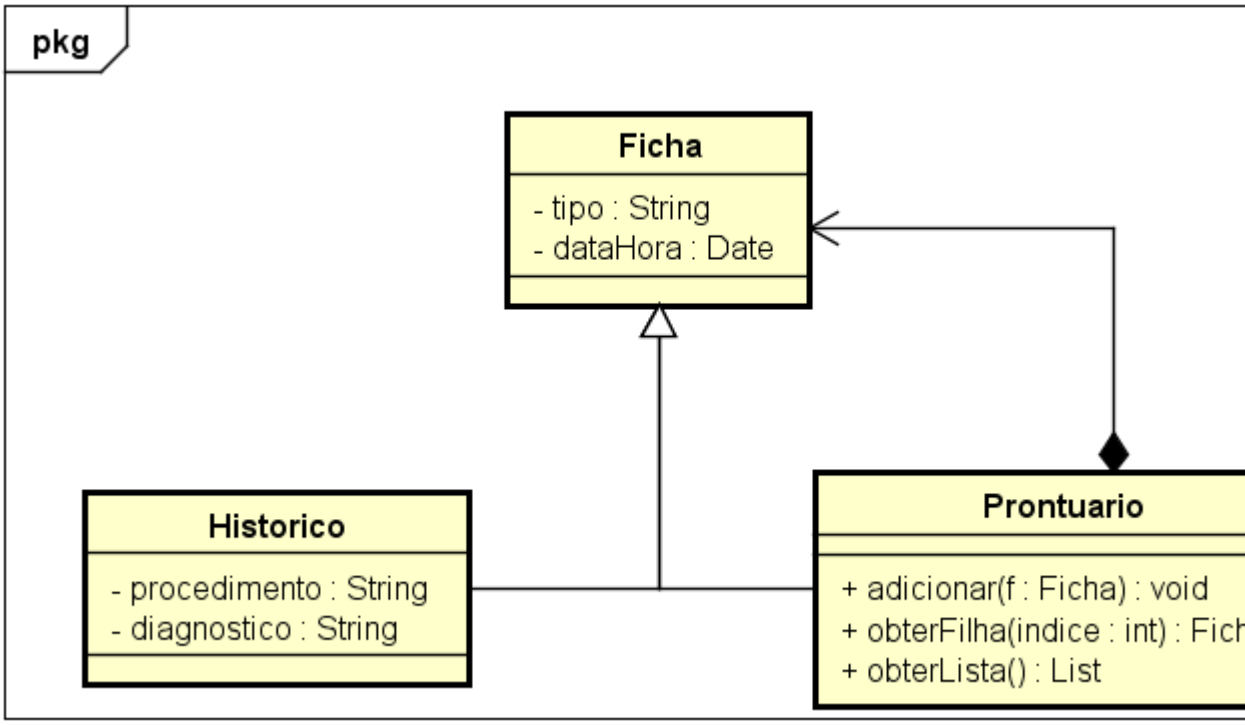
```
public class Conta {
   private static double valorFinal;
   private double valorDesconto;
   private int valorAcrescimo;

   public double gerarDesc () {
      Desconto d = new Desconto();
      valorDesconto = d.gerarDesconto();
   }
   public double acrescimoEmergencia () {
      AcrescimoEmergencia ae = new AcrescimoEmergencia ();
      valorAcrescimo = ae.valorEmergencia();
   }
}
public class AcrescimoEmergencia {
   public double acrescimoEmergencia () {
         …
   }
}
```

```
Public class Desconto {
      public double gerarDesconto () {
      …
     }
}
```

22. Faça um quadro comparativo entre reuso por generalização, realização e delegação, apresentando no mínimo duas vantagens e duas desvantagens para cada um desses conceitos.

	Generalização	Realização	Delegação
Vantagem	Desempenho.	Manutenibilidade.	Alto encapsulamento.
Vantagem	Acoplamento forte.	Revelar as operações de um objeto sem revelar a sua classe.	Dinâmica.
Desvantagem	Diminuição do encapsulamento dependendo de sua profundidade.	Obrigação de ter que implementar todos os métodos.	Fraco acoplamento.
Desvantagem	Heranças múltiplas causam maior complexidade do sistema.	Dificuldade na reutilização do código devido aos métodos serem abstratos.	Baixo desempenho.

23.



24.

```
public class Ficha {
        protected Date dataHora;
```

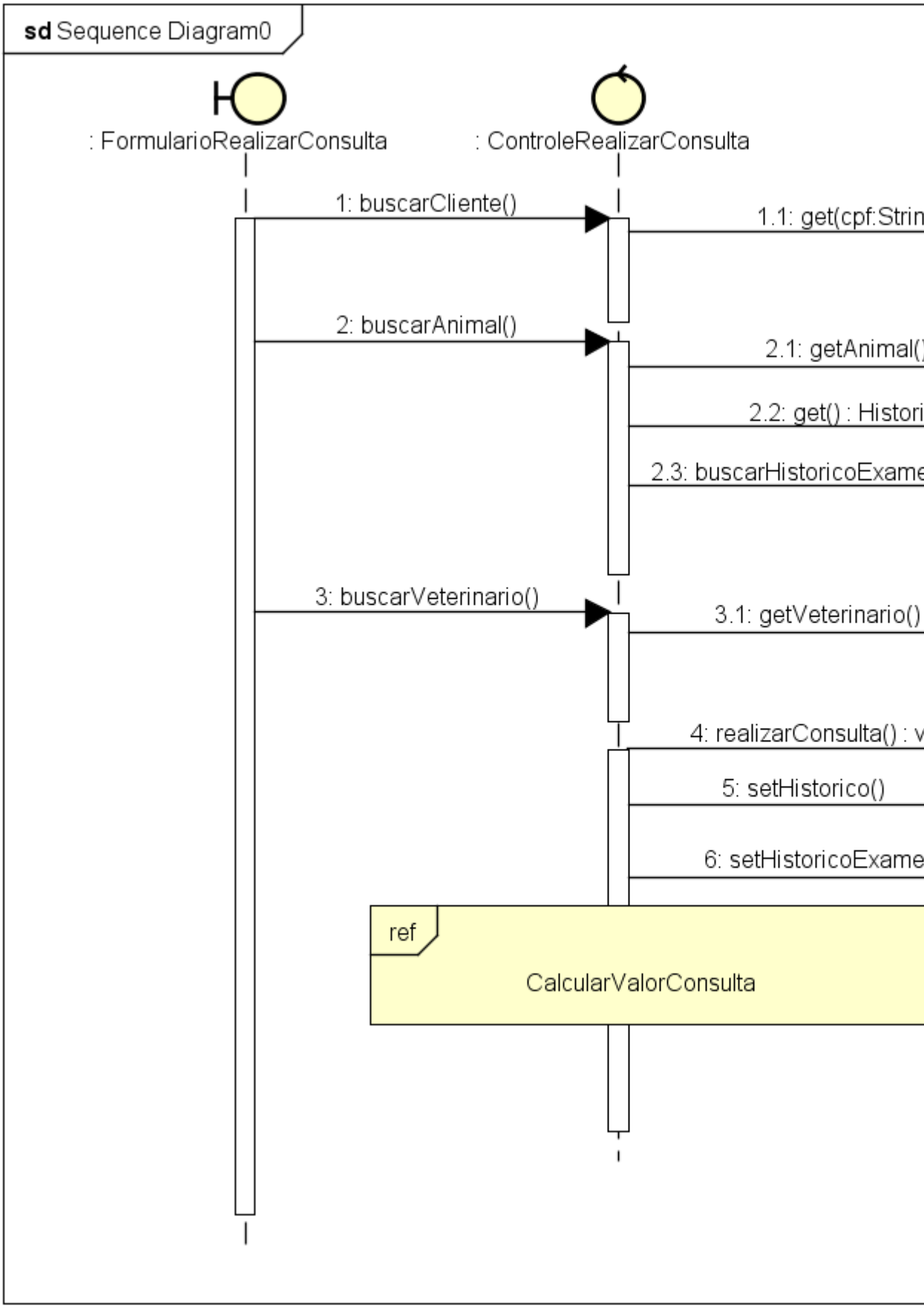
```
        protected String tipo;

        public Date getDataHora() {
            return dataHora;
        }

        public void setDataHora(Date dataHora) {
            this.dataHora = dataHora;
        }

        public String getTipo() {
            return tipo;
        }

        public void setTipo(String tipo) {
            this.tipo = tipo;
        }
    }
    public class Historico extends Ficha {
        private String procedimento;
        private String diagnostico;

        public Historico(Date dataHora, String tipo, String procedimento,
String diagnostico) {
            this.dataHora = dataHora;
            this.tipo = tipo;
            this.procedimento = procedimento;
            this.diagnostico = diagnostico;
        }

        public String getProcedimento() {
            return procedimento;
```

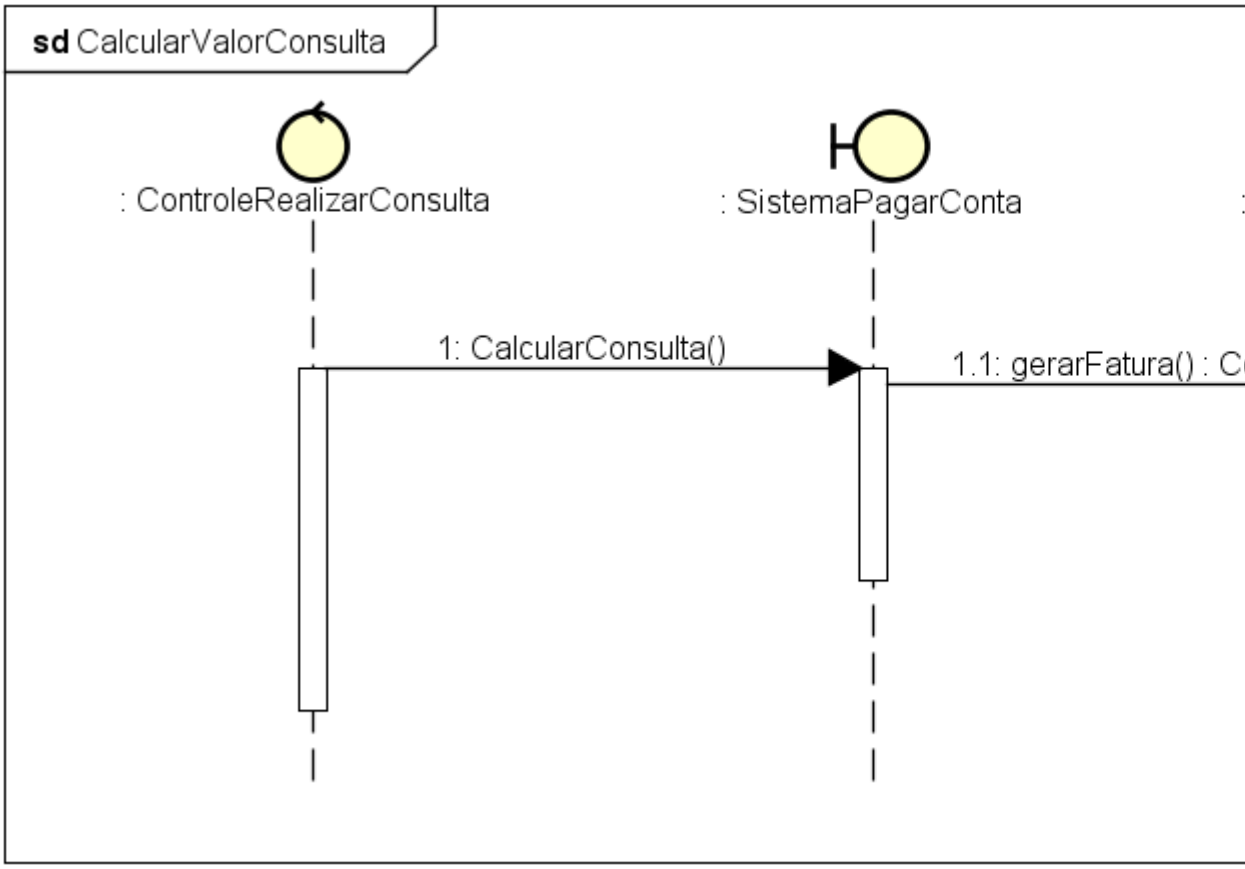
```
}

public void setProcedimento(String procedimento) {
	this.procedimento = procedimento;
}

public String getDiagnostico() {
	return diagnostico;
}

public void setDiagnostico(String diagnostico) {
	this.diagnostico = diagnostico;
}

@Override
public boolean equals(Object o) {
	// Auto comparação
	if (this == o) {
		return true;
	}

	// Comparando null
	if (o == null) {
		return false;
	}

	// Comparando tipo e cast
	if (getClass() != o.getClass()) {
		return false;
	}

	// Comparando campos
```

```
                Ficha f = (Ficha) o;
                return Objects.equals(dataHora, f.dataHora) &&
Objects.equals(tipo, f.tipo);
        }

        @Override
        public String toString() {
                StringBuffer sb = new StringBuffer();

                sb.append(dataHora.toString() + '\n');
                sb.append(tipo + '\n');
                sb.append(procedimento + '\n');
                sb.append(diagnostico + "\n\n");

                return sb.toString();
        }
    }
public class Prontuario {
    private LinkedList<Ficha> prontuario;

    public void adicionar(Ficha f) {
        if (prontuario == null) {
            prontuario = new LinkedList<>();
        }
        prontuario.add(f);
    }
    public Ficha obterFilha(int indice) {
        return prontuario.get(indice - 1);
    }
    public LinkedList<Ficha> obterLista() {
        return prontuario;
    }
```

}

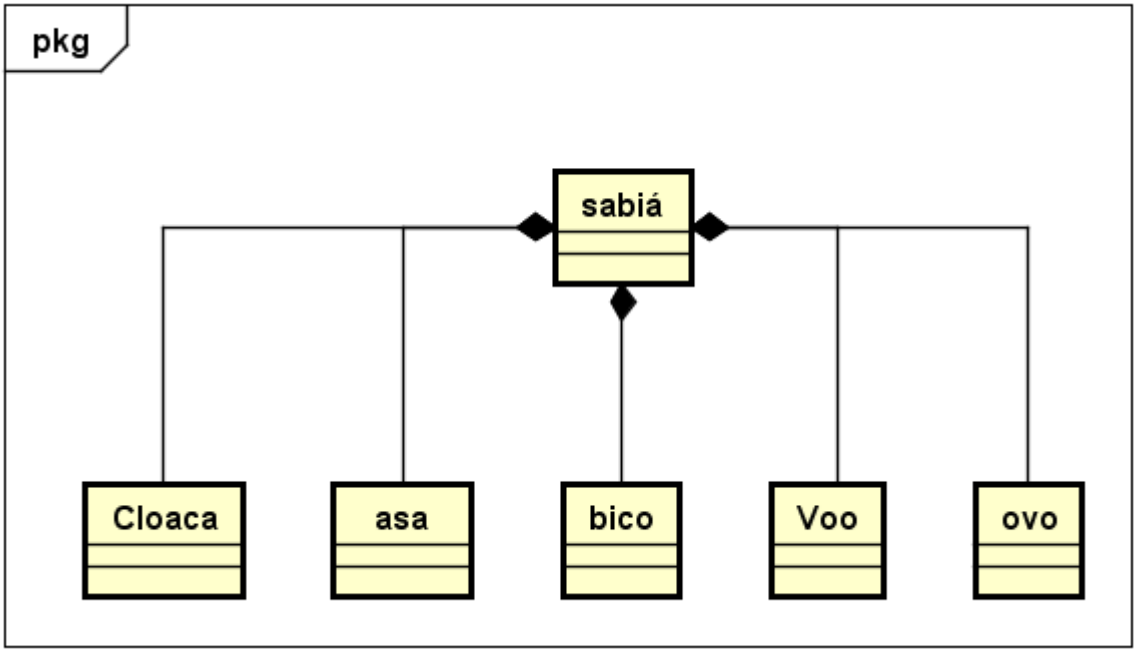
25.


sd Sequence Diagram0
: FormularioRealizarConsulta
: ControleRealizarConsulta
1: buscarCliente()
1.1: get(cpf:Strin
2: buscarAnimal()
2.1: getAnimal(
2.2: get() : Histori
2.3: buscarHistoricoExame
3: buscarVeterinario()
3.1: getVeterinario()
4: realizarConsulta() : v
5: setHistorico()
6: setHistoricoExame
ref
CalcularValorConsulta

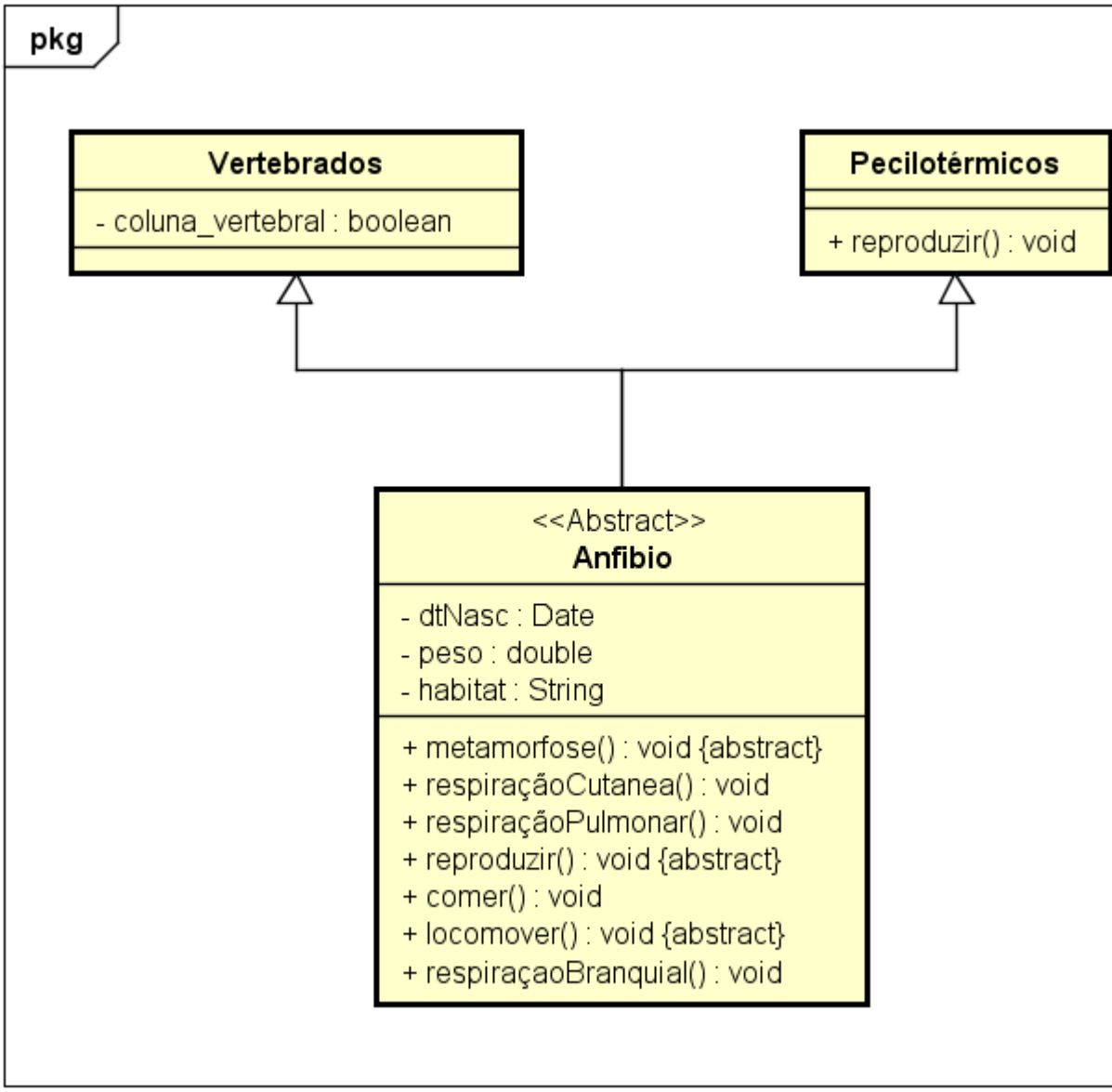

sd CalcularValorConsulta
: ControleRealizarConsulta
: SistemaPagarConta
1: CalcularConsulta()
1.1: gerarFatura() : C

Parte B

1.


pkg
sabiá
Cloaca
asa
bico
Voo
ovo

2.


pkg
Vertebrados
- coluna_vertebral : boolean
Pecilotérmicos
+ reproduzir() : void
<<Abstract>>
Anfibio
- dtNasc : Date
- peso : double
- habitat : String
+ metamorfose() : void {abstract}
+ respiraçãoCutanea() : void
+ respiraçãoPulmonar() : void
+ reproduzir() : void {abstract}
+ comer() : void
+ locomover() : void {abstract}
+ respiraçaoBranquial() : void