**Main.Java**

Aqui que a execução do programa ocorre. Recebendo os import das classes auxiliares e subclasses, esta classe possui um menu para poder realizar as ações desejadas(Adicionar animal, alimentá-lo, verificar se ele corre risco de extinção, etc.).

Ela possui dois métodos: "ShowMenu()" que mostra as opções que versão ser tomadas, e o "cls()", que "emula" a função system("cls") do c++. Opções disponíveis pelo switch-case auxiliado pelo método “Menu”:

\* Adicionar Animal Domesticado. Adicional um animal ao vetor de animais. Caso o nome popular deste animal seja "cachorro", um objeto da classe cachorro será adicionado.

\* Adicionar Animal Selvagem. - Cria um animal da classe Animal\_Selvagem e o adiciona ao vetor.

\* Adicionar Dono. - Caso o animal seja domesticável, um dono pode ser adicionado a ele.

\* Verificar se animal esta doente. Diz se o animal escolhido está doente ou não

\* Alimentar Animal - Alimenta o animal, podendo aumentar o peso deste.

\* Verificar quão velho o animal está - Indica em qual faixa etária o animal está.

\* Verificar o risco de extinção do animal- Retorna quão ameaçada o animal está.

\* Listar- Escreve todos os animais do vetor.

\* AdicionarBrinquedo- Adicionar um brinquedo, que o animal pode brincar ou não.

\* Brincar com o animal- Se o animal for um cachorro, ele brinca com o brinquedo.

\* Sair. - Sai do programa.

No mais, a classe possui diversos atributos que auxiliarão na criação dos objetos, e a maior parte do programa é feito nas outras classes.

**Animal.Java**

SuperClasse abstrata responsável por definir todo Animal deste programa. Extende para duas classes: "Animal\_Domesticado" e

"Animal\_Selvagem". Decidi usar estas subclasses, pois poderia criar facilmente subclasses a partir delas (Animal\_Domesticado derivaria em cão, gato, coelho, etc.).

Assumindo que cada animal tenha um nome Científico (ex: Pantheratigris), uma classe (ex: mamífero, inseto), um codigo(000000432812)

idade(ex: 13 anos, 35 anos), um peso, nomePopular(Pantherastigris == Tigre), um contador de doenças( cresce conforme o animal come alimentos estragados) e um verificador se ele está doente ou não, é possível criar um objeto desta classe.

Nos métodos, temos o "Comer(Comida)", onde, dada uma comida X, o animal alimenta-se dela, um método "Verificar\_perigo\_Extincao"

que retorna um indicador de qual o perigo de extinção de um animal, "Velhice(int Expectativa Vida)", que dada uma expectativa de vida, retorna uma mensagem que varia dependendo da (idade atual/expectativa vida), além de um método que indica se ele está doente.

Existem outros métodos, como o "toString(Animal)", responsável por realizar uma função parecida com o operador "<<" do c++, e o "Habitat()", indicando qual lugar será mais provável encontrar estes animais, e este será sobrescrito por suas subclasse, além de sobrescrever os métodos "hashcode()" e "equals()"

**Animal\_Domesticado.Java**

Sendo uma "filha" da classe Animal, “Animal\_Domesticado” herda todos os atributos da classe Animal. Portanto não explicarei dosdetalhes da superclasse.Então, o que a torna especial? Nao muito **infelizmente**. Alem de possuir um static final para ter o controle do que esta criando, cada animal possui um apelido (cachorro = Totó), e o objeto dono e auxDono indica quais são as informçaoes do Dono. A classe tem um método para verificar qual a qualidade docriador, sem contar que ele sobreescreve as funções "Habitat", "equals" e "toString".

**Animal\_Selvagem.Java**

"Irmã" de Animal\_Domesticado, a classe ANumal\_Selvagem é mais uma classe que herda da classe animal. Não possui tantos atributos quanto à superclasse, mas possui uns importantes. Cada animal da classe "Animal\_Selvagem" possui uma variável indicando se ele é o líder do grupo(Caso já tenha um líder, nenhum outro poderá ser, conforme esta a condição no main), e existeo vetor grupo, constituído de várioas objetos Animal\_Selvagens, além do atributo MAXGRUPO, que delimita o tamanho máximo do grupo.

Para os métodos, temos um método para adicionar algum animal novato ao grupo, além dos "ToString()" e dos "Habitat()"

**Cachorro.java**

Subclasse da classe "Animal\_Domesticado". Armazena a raça do cão, além dos atributos de sua super classe.

**Bola\_Brinquedo.java**

A classe "Bola\_Brinquedo" corresponde a todas as bolas de Brinqendo (Aquelas bolinhas pequenas e fofas, feitas de borracha, normalmente usada para ficar jogando na parede)

Toda bola de brinquedo tem uma marca, é feita de um material X, e se encontra em um Y estado de econservação. A bola pode ter de 1 a 3 cores A classe sobrescreve os métodos "toString", "Hashcode" e "equals"

**Comida.java**

Comida é uma interface referente a todos os tipos de comida, seja ração ou uma refeição mais elaborada. Não possui muitos métodos, só o de verificar a sua qualidade, retornando se está consumível ou não.

**Racao.java**

A classe "Racao" contém as informações mínimas para uma ração. Não possui muitos atributos

"nomeComida" recebe o nome do limento, que é de certa forma desnecessário, diferente das "calorias", que definirão quanto o animal vai engordar. MesesProdução indica quanto tempo esta ração/comida foi produzida ou encontra-se podre, sendo este atributo útil para verificar a "qualidade" da comida, que impacta na saúde do animal.

Além dos velhos conhecidos getters e setters, além de sobrescrição dos métodos "toString", "equals" e "Hashcode".

**Pessoa.java**

A interface Pessoa, além de implementar a interface "Movimento" (Uma pessoa se movimenta", ela é implementada por Dono (Todo dono é uma pessoa). Só possui o método "Adicionar\_Profissão".

**Dono.java**

Todo animal deve ter um dono, certo? Teoricamente, não (E na pratica também não, mas vamos que vamos). Esta classe vai ser importante na classe Dono, para indicar que todo animal tem um dono (Um mundo melhor seria este imagino).

O que o dono possui? Primeiramente, um nome, uma ou mais profissões e o numero de animais que este possui, a qualidade do tratamento será definida através de um método, aonde entra uma nota entre 0 e 10 e sai um conceito (Parecido com o sistema de notas de uma Universidade)

**Movimento.java**

A maior parte das classes deste projeto se movimentam, então não seria nada justo implementar uma interface correspodente a esta ação.

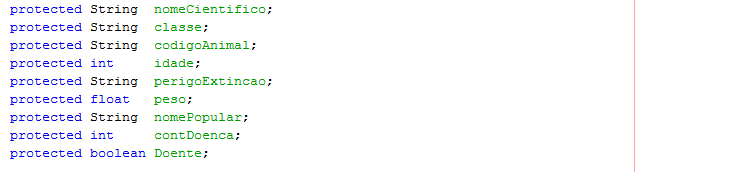
Diversas classes sobrescrevem o método mover (Ex: Pessoa), indicando como eles se movimentam.

**Requisitos Gerais**

***Todos os atributos e funções membros devem estar relacionados a classe***

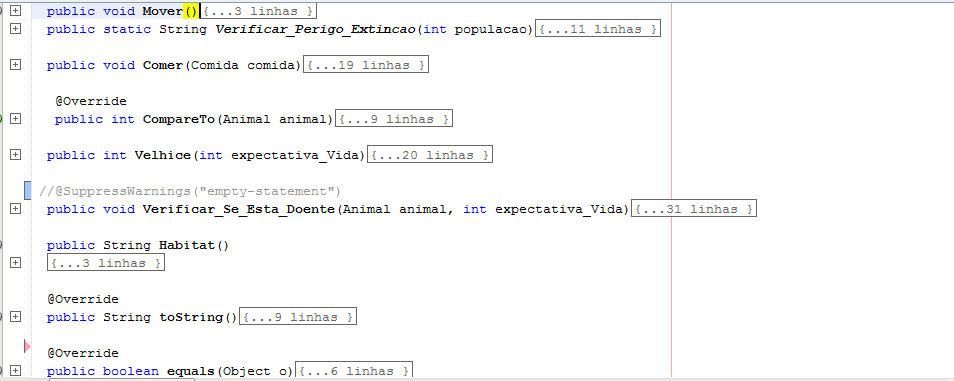
1. Pelo menos 4 atributos

Classe: Animal.java

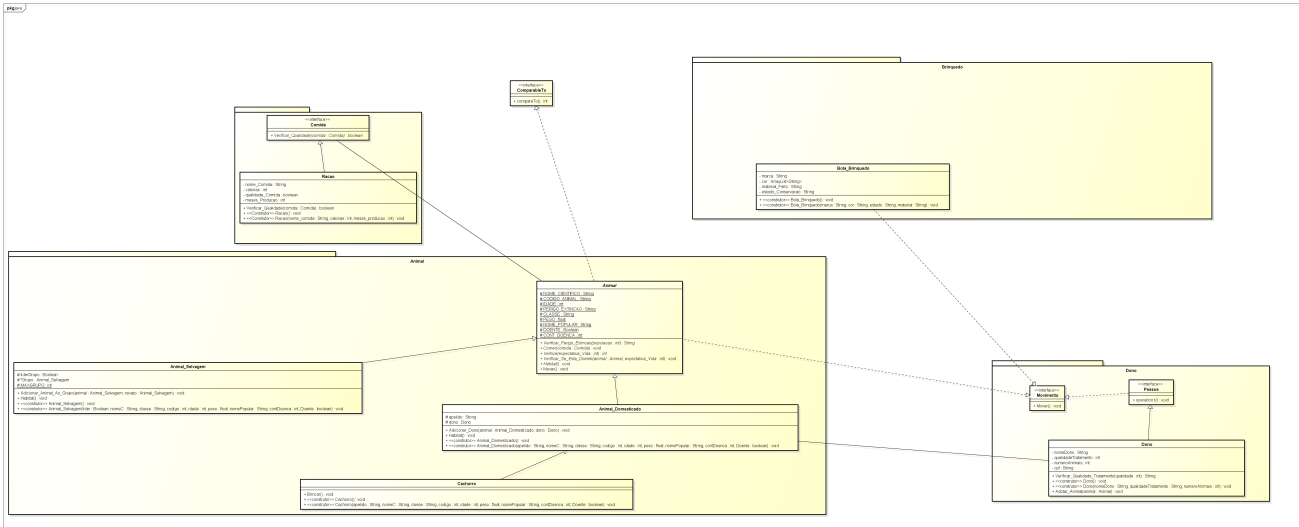


1. Pelo menos 4 funções membros sem incluir get e set

Classe: Animal.java



1. Diagrama UML completo (obrigatório salvar também o png do diagrama no gitHub)



**Requisitos de implementação**

1. Todas as classes concretas devem vir de interfaces ou classes abstratas. Pelo menos três hierarquias de classes. Uma das hierarquias deve ter três níveis. Exemplo: Personagem >>Ciborgue>>Robocop; Class Arma (interface) >> Beretta93R

Classe: Animal\_Selvagem.java

03.PNG

Classe: Cachorro.java

04.PNG

Classe: Dono.java

05.PNG

Classe: Bola\_Brinquedo.java

07.PNG

Classe: Animal\_Domesticado.java

06.PNG

Classe: Racao.java

09.PNG

1. Ao menos três interfaces. A terceira interface deve ser uma interface que liga duas hierarquias como no exemplo da interface **corredor**(Figura 1).

Classe: Comida.java

08.PNG

Classe: Pessoa.java

11.PNG

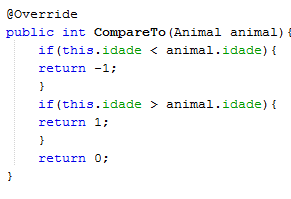
Classe: Movimento.java

12.PNG

1. Usar a interface **Comparable** e sobrescrever o método **compareTo** em pelo menos uma hierarquia x

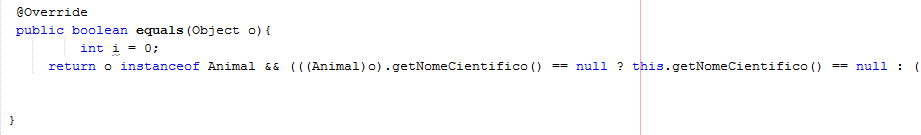
Classe: CompareTo.java

13.PNG

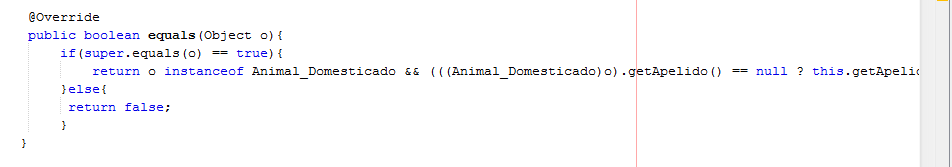
Classe: Animal.java

1. Sobrescrever **equals** para de Objectx

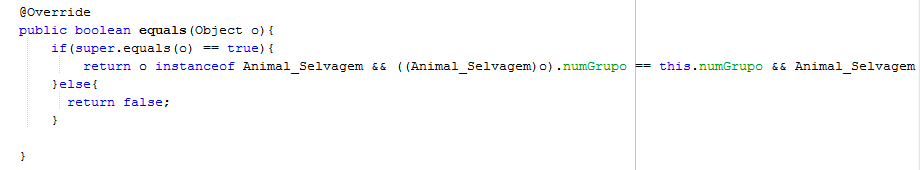
Classe: Animal.java



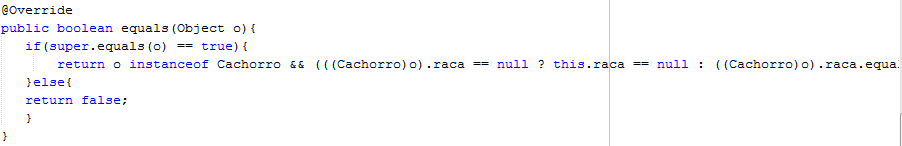
Clase:Animal\_Domesticado.java



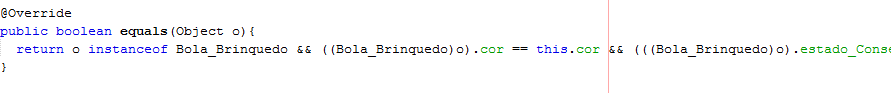
Classe: Animal\_Selvagem.java



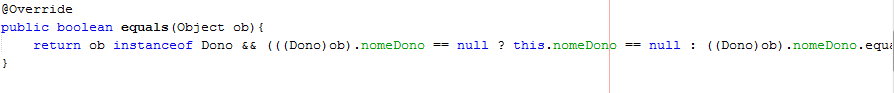
Classe: Cachorro.java



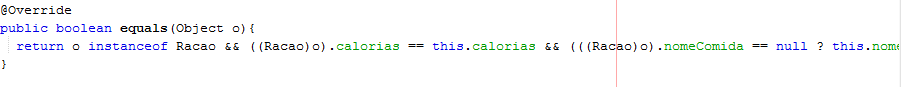
Classe: Bola\_Brinquedo.java



Classe: Dono.java



Classe: Racao.java



1. Todas as hierarquias devem ter classes Concretas, e em uma das hierarquias, três classes Concretas relacionadas

Hierarquia Animal

47.PNG48.PNG49.PNG50.PNG

Hierarquia Dono

51.PNG52.PNG

56.PNG

Hierarquia Brinquedo

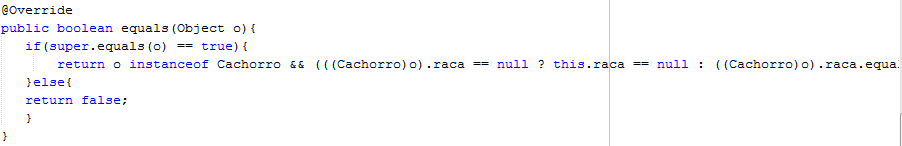
53.PNG

Hierarquia Comida

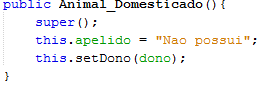
54.PNG 55.PNG

1. Sempre usar o super para o máximo de reaproveitamento de código x

Classe: Cachorro.java



Classe: Animal\_Domesticado.java



1. Atributos static e const static

Classe: Main.java

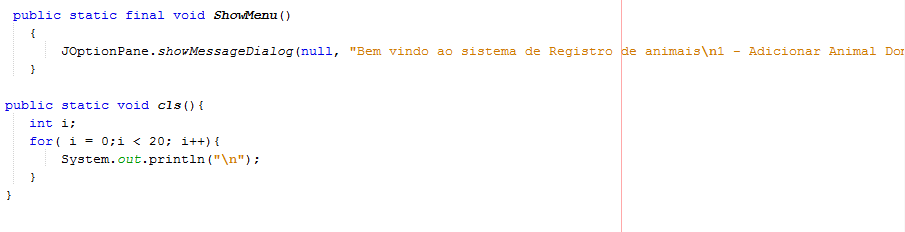
22.PNG

Classe: Animal\_Selvagem.java

23.PNG

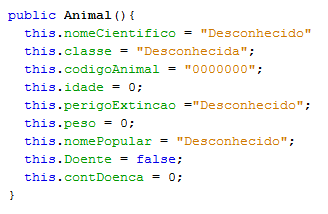
1. Método static

Classe: Main.java

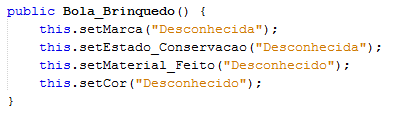


1. Construtores em todas as classes, e dois na hierarquia principal. Sempre validar os dados em todas as classes

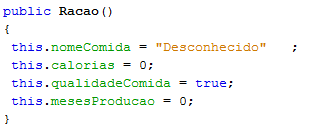
Class:Animal.java



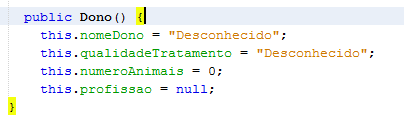
Class: Bola\_Brinquedo.java



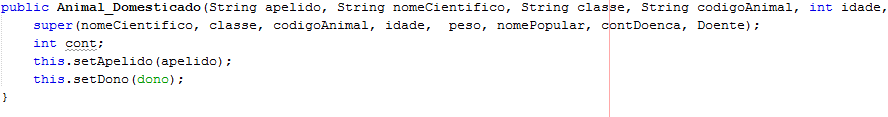
Class: Racao.java



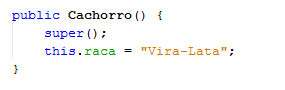
Class: Dono.java



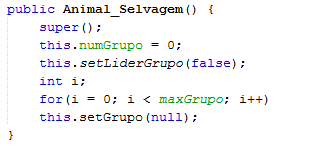
Class: Animal\_Domesticado.java



Class: Cachorro.java

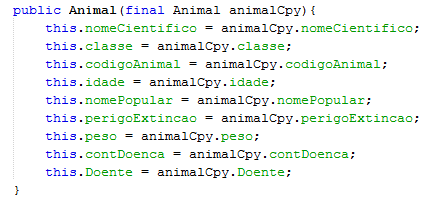


Class: Animal\_Selvagem.java



1. Construtor cópia em uma das hierarquias

Classe: Animal.java



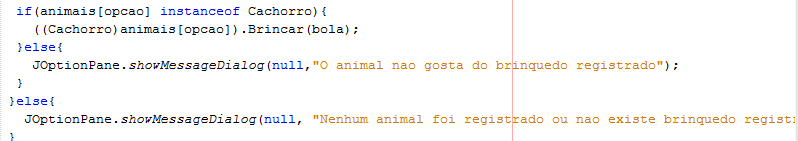
1. ArrayListx

Classe: Dono.java

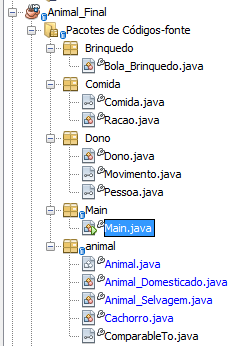
25.PNG

1. ENUM
2. Usar o **instanceof** no main junto com as classes concretas. Para uma da classe concreta identificada, chamar um método dessa classe e fazer uma ação;

Classe: Main.java

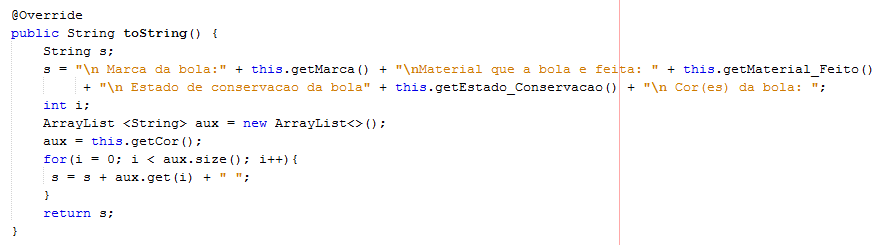


1. Dividir o projeto em pacotes

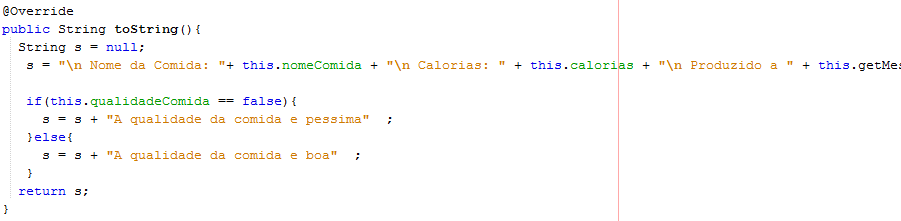


1. Sobrescrever para todas as classes o método toString

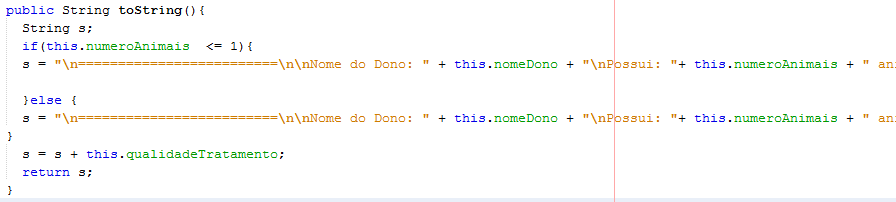
Classe: Bola\_Brinquedo.java



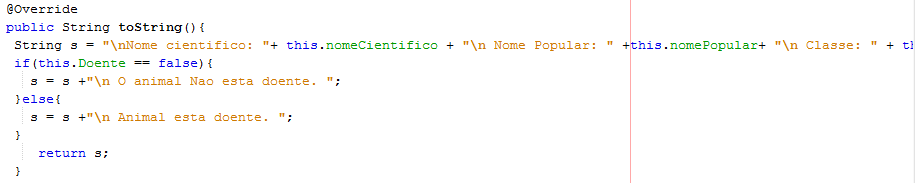
Classe: Racao.java



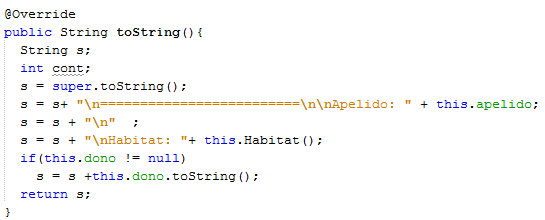
Classe: Dono.java



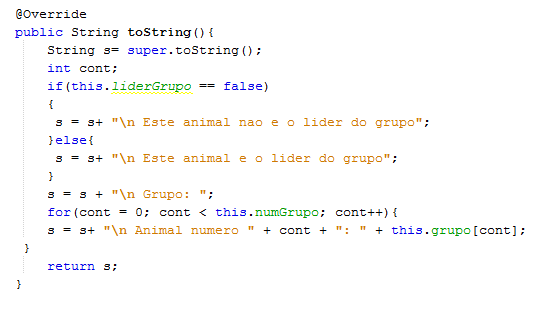
Classe: Animal.java



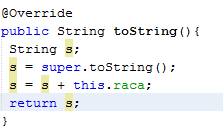
Classe: Animal\_Domesticado.java



Classe: Animal\_Selvagem.java

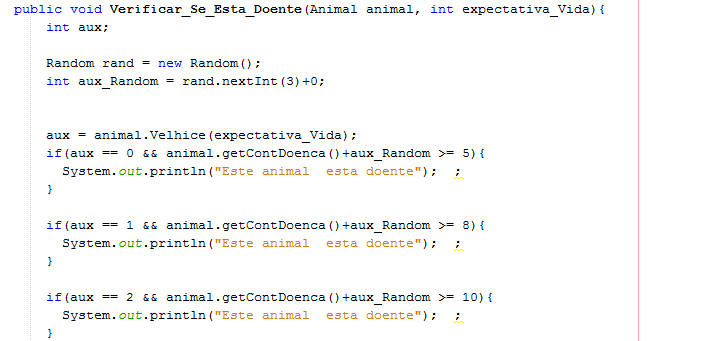


Classe: Cachorro.java



1. Usar a classe Random do pacote java.util (java.util.Random). Nota: deve ser usado conforme o contexto do projeto. Se for usado em um método genérico sem relação com a classe e apenas para cumpri-lo, esse requisito será desconsiderado.

Classe: Animal.java



1. No main o usuário deve fazer entrada via teclado e interagir com a aplicação. Opcional de bônus: pode ser usada a classe JOptionPane do pacote javax.swing. Vejam: showInputDialog e showMessageDialog.

Classe: Main.java

