**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.

\* Adicionar Brinquedo - 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: Panthera tigris), uma classe (ex: mamífero, inseto), um codigo(000000432812)

idade(ex: 13 anos, 35 anos), um peso, nomePopular(Pantheras tigris == 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 dos detalhes 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 do criador, 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()"

**Requisitos Gerais**

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

1. Pelo menos 4 atributos x
2. Pelo menos 4 funções membros sem incluir get e set x
3. Diagrama UML completo (obrigatório salvar também o png do diagrama no gitHub) x

**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 x
2. Ao menos três interfaces. A terceira interface deve ser uma interface que liga duas hierarquias como no exemplo da interface **corredor** (Figura 1). x
3. Usar a interface **Comparable** e sobrescrever o método **compareTo** em pelo menos uma hierarquia x
4. Sobrescrever **equals** para de Object x
5. Todas as hierarquias devem ter classes Concretas, e em uma das hierarquias, três classes Concretas relacionadas: Exemplo Servico >> ServicoStream >> (Netflix, HBOStream, AmazonPrime, NowTv). Em uma pesquisa de 10 segundos: <http://www.tomsguide.com/us/pictures-story/620-top-online-streaming-video.html> x
6. Sempre usar o super para o máximo de reaproveitamento de código x
7. Atributos static e const static x
8. Método static x
9. Construtores em todas as classes, e dois na hierarquia principal. Sempre validar os dados em todas as classes x
10. Construtor cópia em uma das hierarquias x
11. ArrayList x
12. ENUM
13. 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;
14. Dividir o projeto em pacotes x
15. Sobrescrever para todas as classes o método toString x
16. 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. x
17. 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.



Figura 1 – Interface Corredor conectando duas hierarquias de classe