



# Semana 7

## Hierarquia de Classes

Profª Marina de Lara



# Exemplo: Animais



## Atributos:

Nome  
Raça  
Cor  
Idade  
Classe



## Atributos:

Nome  
Raça  
Cor  
Idade  
Classe



# Dúvida

De que forma você implementaria apenas uma classe para esses dois animais, visto que eles são iguais?



# Exemplo: Animais



## Atributos:

Nome

Raça

Cor

Idade

Classe

**Tipo do Pelo**



## Atributos:

Nome

Raça

Cor

Idade

Classe


**Tamanho da Asa**



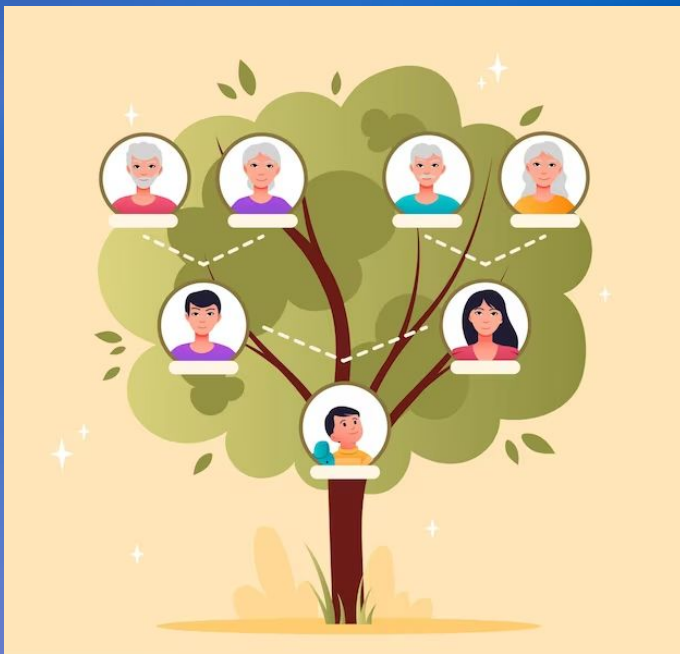
# Dúvida

De que forma você implementaria apenas uma classe para esses dois modelos, visto que eles são iguais?

**Mas o que acontece quando temos um, ou mais, atributo(s) diferente(s)? Isso muda alguma coisa na resposta anterior?**



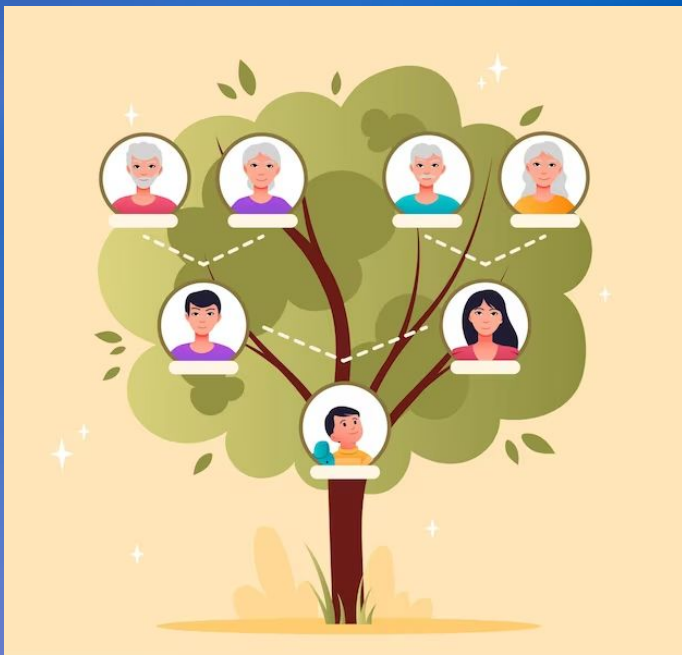
# Conceito da Herança



Toda família possui uma **árvore genealógica** certo? O que significa essa relação hereditária que possuímos com os nossos antepassados? Vocês acreditam que possuem características físicas semelhantes à outros familiares?

Fonte: Freepick

# Conceito da Herança



A **herança** na programação pode ser entendida como a nossa relação com nossos antepassados. Herdamos características da nossa família: cor do olho, cabelo, altura, formato do rosto, etc. Sempre temos algo que nos torna semelhantes à nossa família. Mas mesmo assim, também temos nossas características próprias, que são só nossas.

Fonte: Freepick

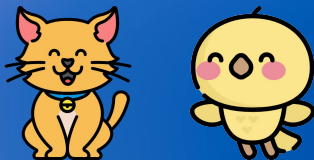
# **Voltando para os Animais**

De que forma tudo isso que vimos seria utilizado em código?



# Como implementar?

De que forma tudo isso que vimos seria utilizado em código?



## Atributos:

Nome  
Raça  
Cor  
Idade  
Classe

```
public class Animal {  
    private String nome;  
    private String raca;  
    private String cor;  
    private int idade;  
    private String classe;  
}
```

Podemos criar uma classe que represente o objeto **Animal** no nosso sistema.

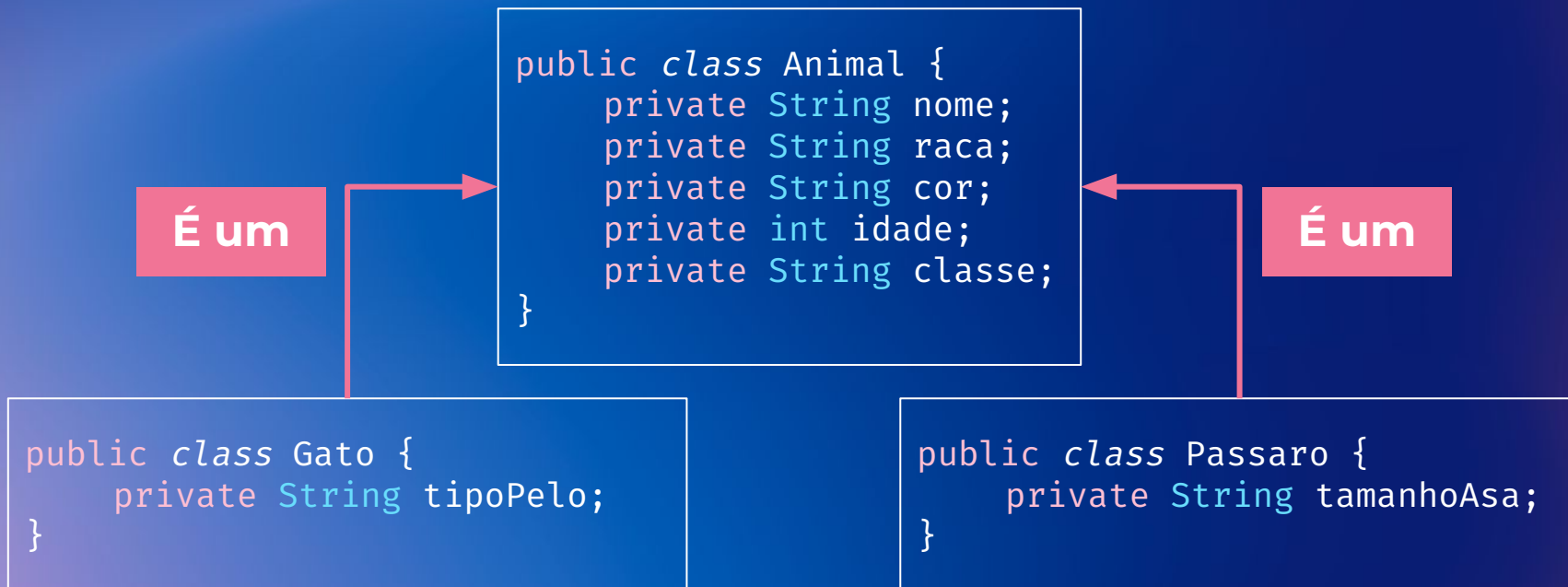


# Atributos diferentes

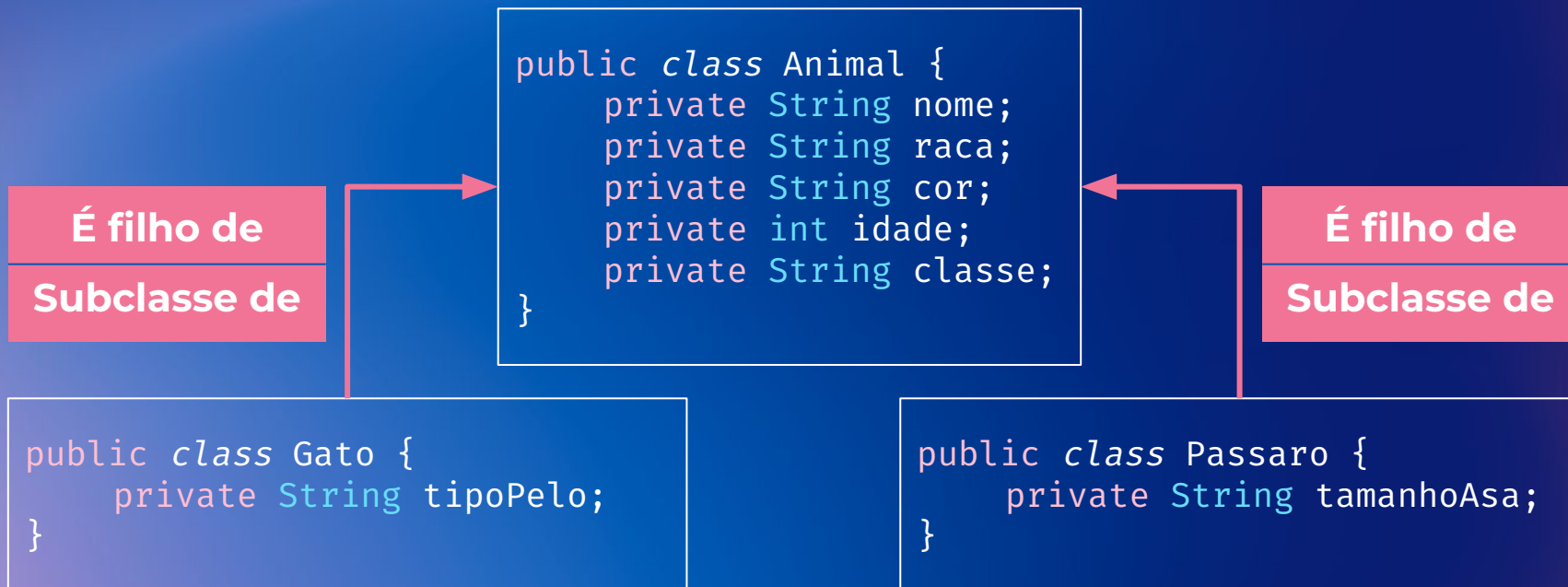
Mas e no caso dos objetos possuírem atributos diferentes?

# Criando relação de herança

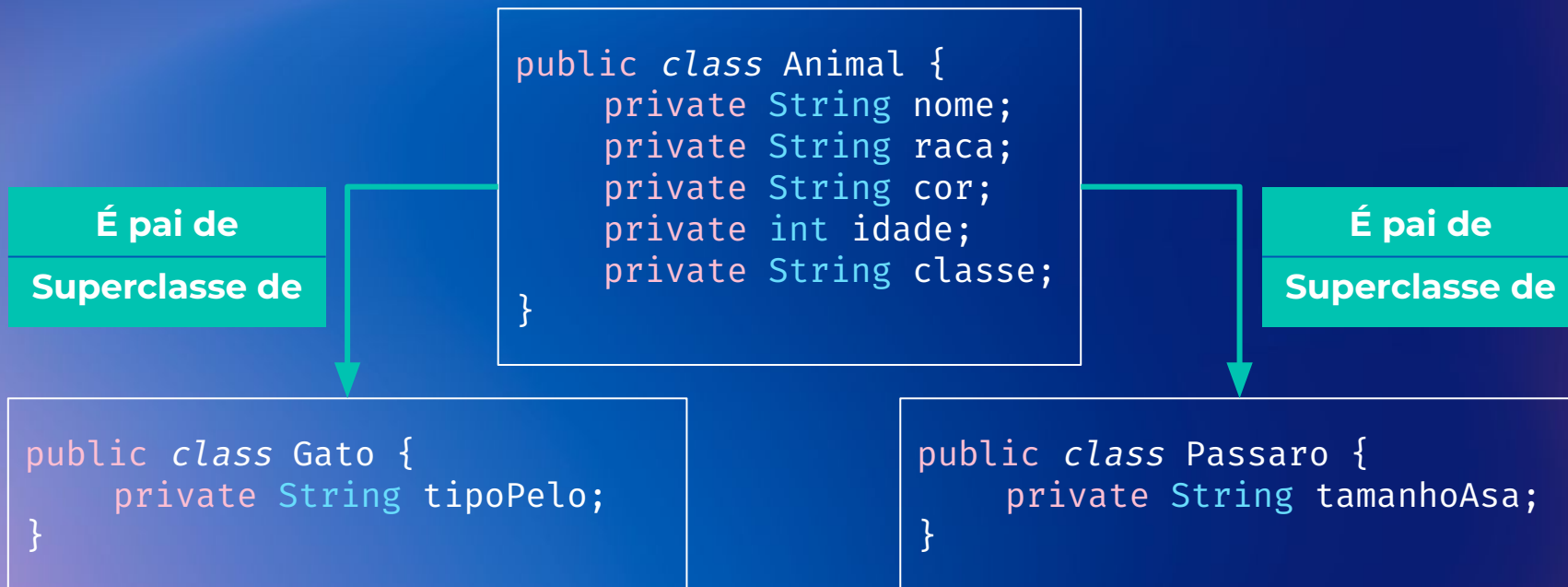
Mas e no caso dos objetos possuírem atributos diferentes?



# Criando relação de herança



# Criando relação de herança



# Criando relação de herança

A palavra **extends**

Para que a relação de herança seja criada entre as classes **Animal** e **Gato**, e **Animal** e **Passaro**, é preciso utilizar a palavra chave **extends** que indica que todos os atributos da classe pai, serão herdados pelas classes filhas.

```
public class Gato extends Animal {  
    private String tipoPelo;  
}
```

```
public class Passaro extends Animal {  
    private String tamanhoAsa;  
}
```

# Modificador de Acesso *Protected*

Para que os atributos possam ser acessados pela **classe filha**, podemos mantê-los **privados** e acessar utilizando os **getters** e **setters**, ou podemos alterar o modificador de acesso para **protected**.

Dessa forma, todos os atributos do **pai**, podem ser acessados diretamente por qualquer **filho**.

```
public class Animal {  
    protected String nome;  
    protected String raca;  
    protected String cor;  
    protected int idade;  
    protected String classe;  
}
```

```
public class Gato extends Animal {  
    private String tipoPelo;  
  
    public Gato(String nome) {  
        this.nome = nome;  
    }  
}
```




# Métodos

Até agora vimos que os **objetos filhos** podem herdar **características (atributos)** da classe pai.

**Mas e os comportamentos?**

**Podemos herdar comportamentos também!**

Todos os **métodos** que forem implementados na **superclasse** também são estendidos para as **subclasses**.





# Exemplo no código

```
public class Animal {  
    protected String nome;  
    protected String raca;  
    protected String cor;  
    protected int idade;  
    protected String classe;  
  
    public void comer() {  
        System.out.println("Nhami");  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Gato gato = new Gato();  
        gato.comer();  
  
        Passaro passaro = new Passaro();  
        passaro.comer();  
    }  
}
```

## Console output:

Nhami  
Nhami

# Dúvida

Podemos criar (instanciar) um objeto do tipo do pai?

```
public class Main {  
    public static void main(String[] args) {  
        Animal animal = new Animal();  
        animal.comer();  
    }  
}
```

# Dúvida

Podemos criar (instanciar) um objeto do tipo do pai?

```
public class Main {  
    public static void main(String[] args) {  
        Animal animal = new Animal();  
        animal.comer();  
    }  
}
```

**SIM**

# Dúvida

Podemos criar (instanciar) um objeto do tipo do pai?

```
public class Main {  
    public static void main(String[] args) {  
        Animal animal = new Animal();  
        animal.comer();  
    }  
}
```

Pode ser que no seu programa você crie animais “genéricos” e queira especificar apenas alguns tipos.

Mais pra frente vamos ver algumas outras técnicas que utilizam a herança como base. Mas vamos devagar para não perder nada!

# Construtores na classe pai

Podemos  
implementar  
construtores  
normalmente em  
uma classe pai, não  
existe diferença na  
implementação.

```
public class Animal {  
    protected String nome;  
    protected String raca;  
    protected String cor;  
    protected int idade;  
    protected String classe;  
  
    public Animal(String nome, String raca, String cor,  
                  int idade, String classe) {  
        this.nome = nome;  
        this.raca = raca;  
        this.cor = cor;  
        this.idade = idade;  
        this.classe = classe;  
    }  
}
```

# Construtores na classe filha



Na classe filha o negócio é um pouco diferente, pois essa implementação não irá funcionar. A IDE vai apontar que existe um erro pois, uma vez que um construtor é implementado na superclasse, é preciso chamá-lo na subclasse

```
public class Gato extends Animal {  
    private String tipoPelo;  
  
    public Gato(String nome, String raca, String cor,  
                int idade, String classe, String tipoPelo) {  
        this.nome = nome;  
        this.raca = raca;  
        this.cor = cor;  
        this.idade = idade;  
        this.classe = classe;  
        this.tipoPelo = tipoPelo;  
    }  
}
```



# Construtores na classe filha

Podemos chamar o construtor da classe pai utilizando a palavra **super**. Da mesma forma que fazemos quando chamamos um construtor da própria classe com o **this**, mas agora utilizando a palavra **super** no lugar de **this**.

```
public class Gato extends Animal {  
    private String tipoPelo;  
  
    public Gato(String nome, String raca, String cor,  
                int idade, String classe, String tipoPelo) {  
        super(nome, raca, cor, idade, classe);  
        this.tipoPelo = tipoPelo;  
    }  
}
```



# Construtores na classe filha

Notem que, na linha sublinhada, o atributo que pertence apenas à classe filha é atribuído igual antes. Nós só iremos passar para o **construtor pai** os atributos que pertencem à ele, pois **o pai não conhece os atributos do filho**.

```
public class Gato extends Animal {  
    private String tipoPelo;  
  
    public Gato(String nome, String raca, String cor,  
                int idade, String classe, String tipoPelo) {  
        super(nome, raca, cor, idade, classe);  
        this.tipoPelo = tipoPelo;  
    }  
}
```





# Importante lembrar sempre:

Uma classe pai é como outra classe qualquer, a **única diferença por enquanto** é que esse tipo de classe **pode** “compartilhar” seus atributos e métodos com seus filhos.

As classes filhas **podem** herdar atributos e comportamentos da classe pai, mas uma classe pai **nunca** herda nada dos filhos, ou seja, as implementações do pai são herdadas pelas classes filhas, mas **nunca o contrário**.

Modificadores de acesso continuam seguindo a mesma regra: **public** pode ser acessado por todos, **private só pode ser acessado dentro da classe** (seja ela superclasse ou não) e **protected** pode ser acessado por **toda a família** (hierarquia de classes).

Quando um construtor é implementado na classe pai, todos os filhos precisam **obrigatoriamente** chamar o **construtor pai** no **seu próprio construtor**. Para isso utilizamos a palavra **super** e passamos os parâmetros que o **construtor do pai** exige.

# Atividade

Utilize o modelo criado nas aulas passadas e modifique o código de forma que a nova implementação utilize herança nos objetos que similares em seus atributos e métodos.

1. As equipes devem entregar uma primeira versão até o final da aula de hoje **(Terça-feira 03/04 - 11:10)**
2. As equipes que não conseguirem finalizar todas as modificações até o final da aula, podem realizar uma nova entrega até o início da próxima aula **(Quinta-feira 05/04 - 07:50)**
3. **Para equipes que não encontraram necessidade de implementar herança no seu código:** Aumente o escopo da implementação até encontrar um caso para aplicação de herança **ou** encontre um novo site/aplicativo (ex: loja de produtos)
4. **Para equipes que já realizaram a implementação com herança anteriormente:** Encontre outro site/aplicativo para praticar mais.