Construtores e interfaces

Maurício Linhares

Herança e construtores

- Quando uma classe herda de outra, ela herda as ações (os métodos) e os atributos (variáveis de instância), mas os construtores não são herdados;

- Em qualquer construtor, a primeira linha é, obrigatoriamente, uma chamada a um construtor da superclasse, quando a nossa superclasse tem um construtor padrão, o compilador adiciona essa linha automaticamente;

Exemplo

```
public class Animal {
    public Animal() { super(); }
}

public class Leao extends Animal {

    public Leao() {
            super();
    }

}
```

Se a superclasse não tiver um construtor padrão...

- Na primeira linha do construtor da sua subclasse você vai ter que chamar um dos construtores da supeclasse diretamente;
- Você pode escolher entre qualquer um dos construtores que houverem na superclasse;
- A chamada a o construtor da superclasse é feita através de uma chamada a "super()" passando os parâmetros dentro dos parênteses;
- Não pode haver nenhuma linha de código anterior a chamada do construtor da superclasse;

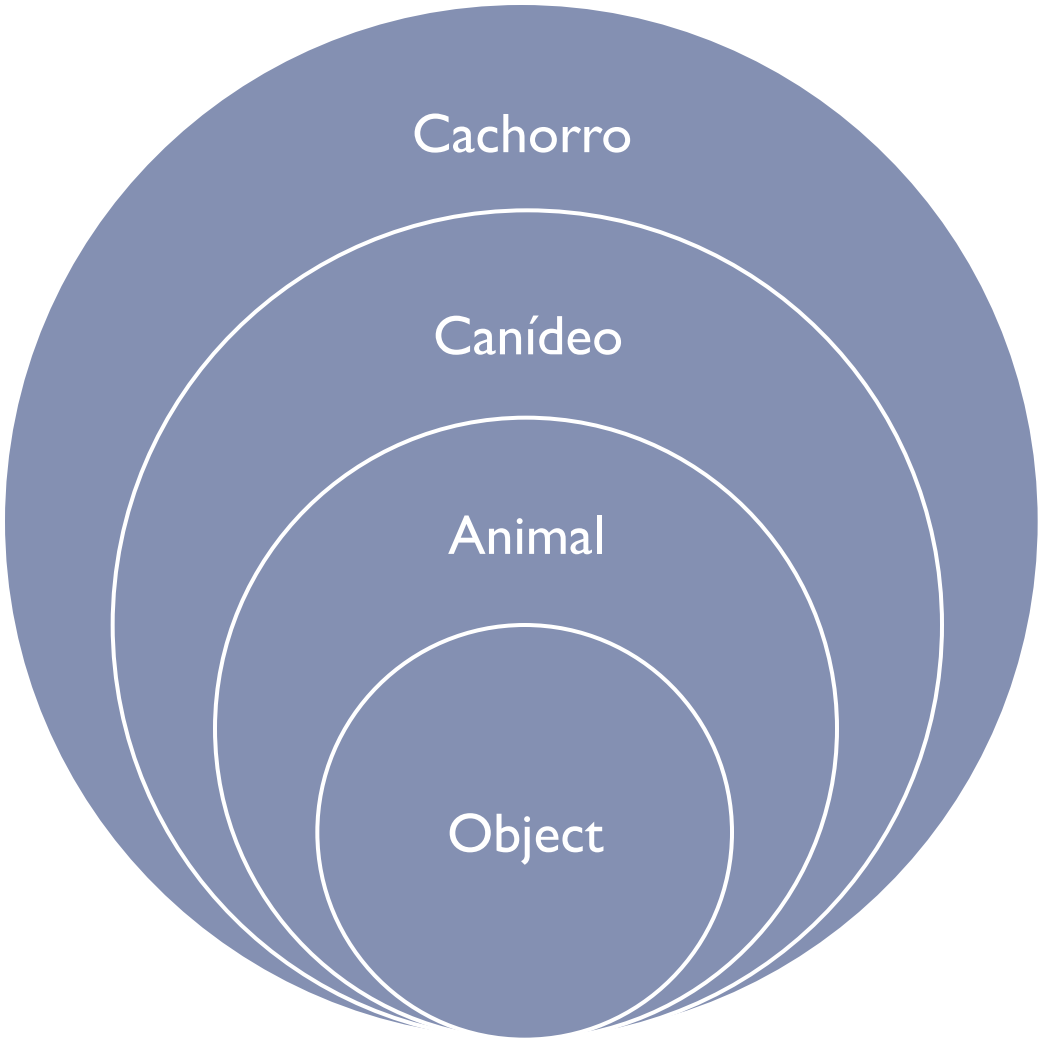
Exemplo de superclasse sem construtor padrão

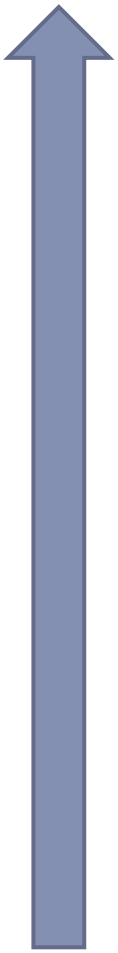
```
public class Canideo extends Animal {

   private String raca;

   public Canideo( String raca ) {
         this.raca = raca;
   }

   public String getRaca() {
         return raca;
   }

}
```

Exemplo da subclasse

```
public class Cachorro extends Canideo {

    public Cachorro() {
        //não compila
    }

    public Cachorro(String raca) {
        super(raca);
    }

}
```

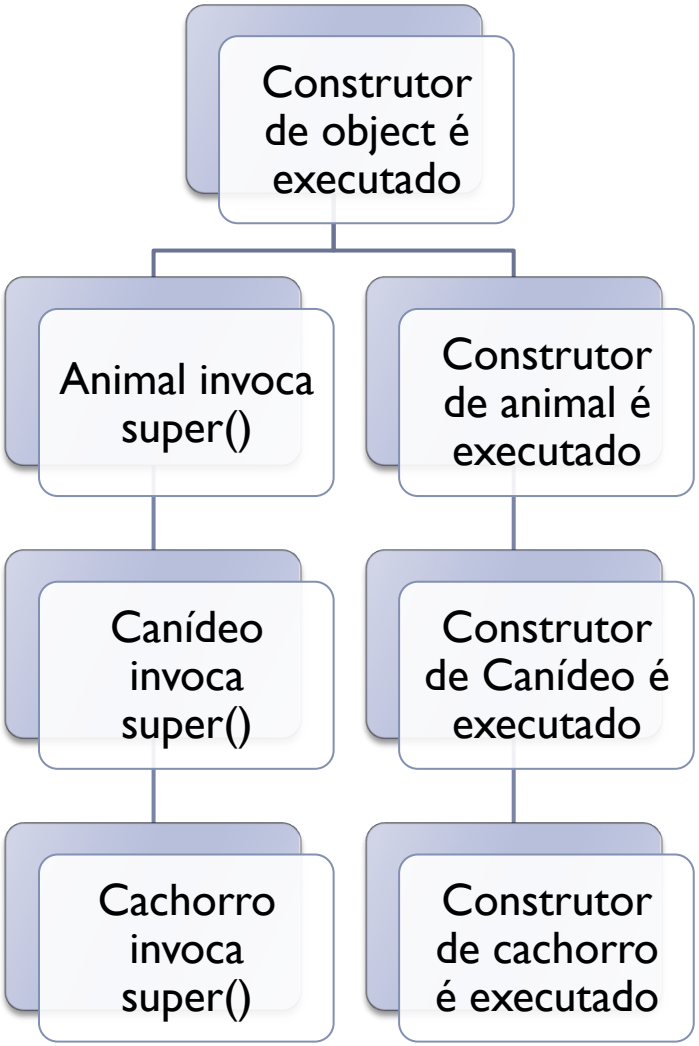
A anatomia de um objeto que herda de outros objetos





Ordem de execução dos construtores

Quando eu invoco o construtor de Cachorro...



Chamando um construtor da mesma classe

- É possível invocar, de dentro de um construtor, outro construtor da mesma classe;
- Você faz isso através de uma chamada a "this()" passando pelos parênteses os parâmetros para o construtor que vocêr quer invocar;
- Essa chamada a "this()" deve ser feita na primeira linha do seu construtor;

Exemplo de chamada a construtor da mesma classe

```
public class Cachorro extends Canideos {

    public Cachorro() {
            this( "Pastor Alemão" );
    }

    public Cachorro(String raca) {
            super(raca);
    }

}
```

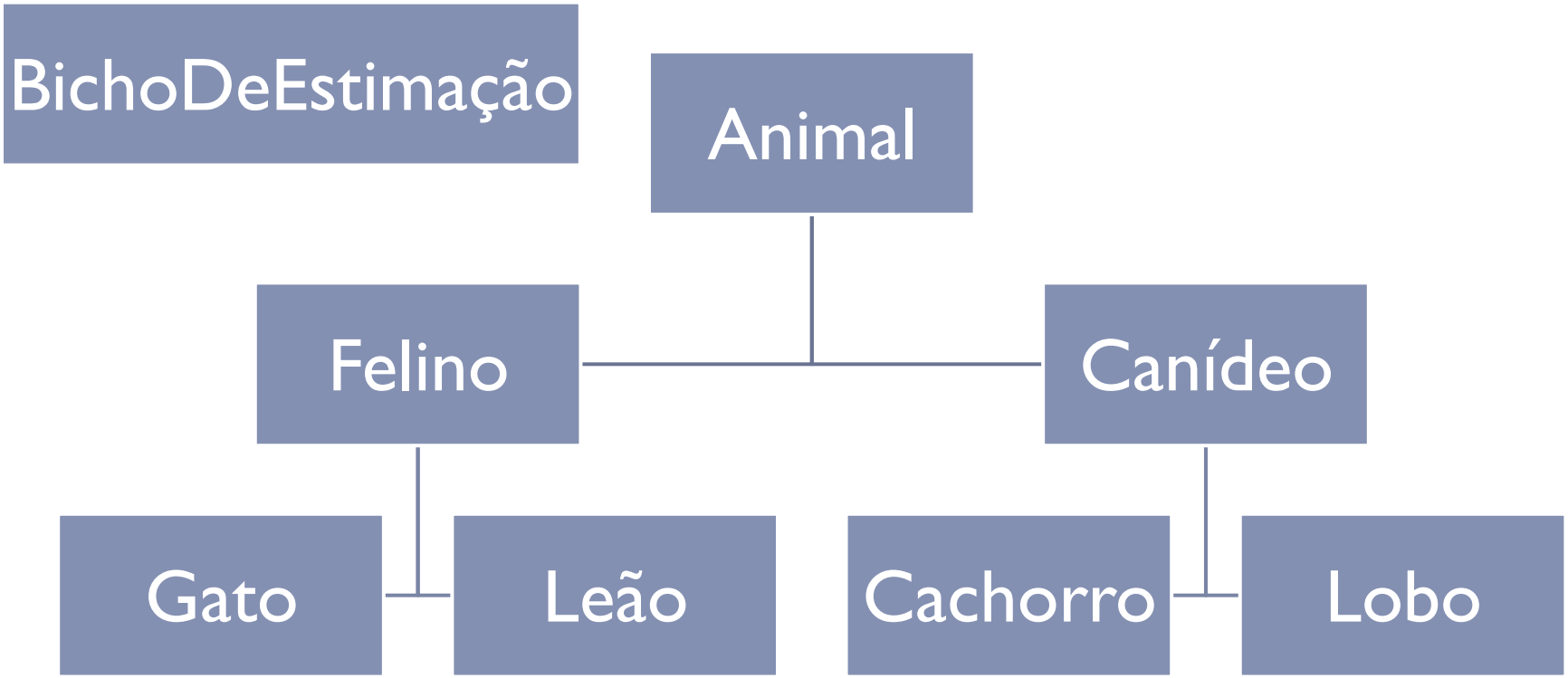
Complicando o nosso problema de herança

- Nossos animais viviam em um zoológico, mas agora o nosso sistema também precisa ser capaz de lidar com animais de estimação;
- Nem todos os animais vão ser animais de estimação (até porque você náo teria espaço para criar um hipopótamo em casa);
- Exemplos de animais que seriam bichos de estimação são Gatos e Cachorros;

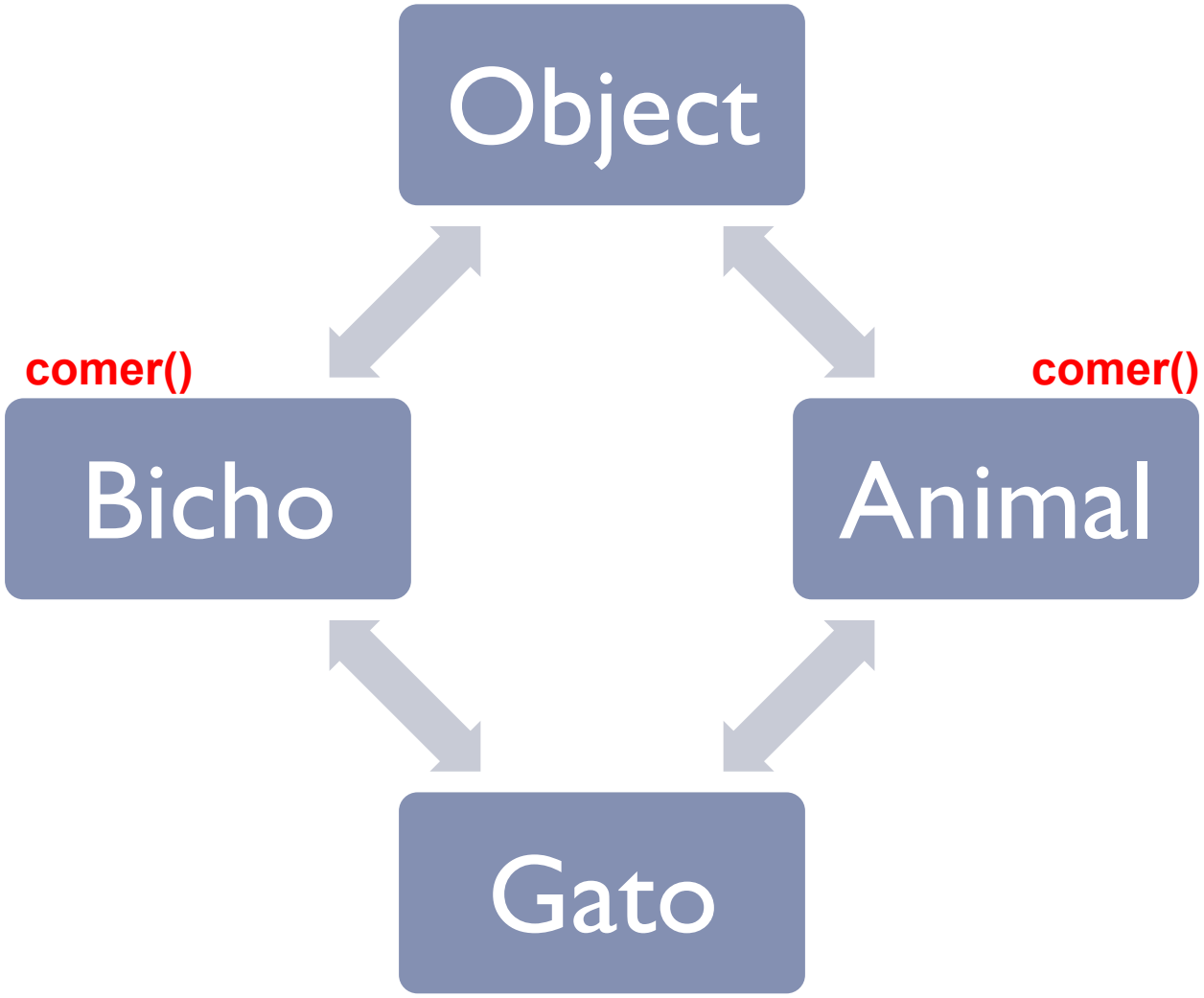
Criando a classe bixo de estimação

```
public class BichoDeEstimacao {
    private String dono;

    public void setDono(String dono) {
            this.dono = dono;
    }

    public String getDono() { return dono;}

    public void brincar() {
            System.out.println("Brincando com " + this.dono);
    }

    public void comer() {
            System.out.println("Comendo ração");
    }

}
```

Agora, aonde colocar ela?



O Diamante da Morte



Herança múltipla

- O famoso diamante da morte é um dos resultados de se usar herança múltipla em projetos;
- Em Java, felizmente, é impossível usar herança múltipla, então não precisamos nos preocupar com esse problema;
- Mas o fato de não haver herança múltipla também não resolve nosso problema, que é fazer gatos e cachorros poderem ser tratados como bichos de estimação;

E as interfaces vem para salvar o dia!

- Interfaces são "contratos" definidos em Java que dizem como um objeto deve se comportar;
- Em uma interface você define métodos que um objeto que implemente o contrato da interface deve ter implementados;
- Uma interface funciona como uma classe abstrata que não tem nenhuma implementação;

A nossa primeira interface

```
public interface BichoDeEstimacao extends Cloneable {

    public abstract void setDono( String dono );

    public abstract String getDono( );

    public abstract void comer();

    public abstract void brincar();

}
```

Em uma interface...

- Todos os métodos são abstratos;
- Todos os métodos são públicos;
- Não existem construtores;
- Não existem variáveis de instância, apenas constantes;

Implementando uma interface

```
public class Cachorro extends Canideo implements BichoDeEstimacao {

    private String dono;

    public String getDono() {
            return this.dono;
    }

    public void setDono(String dono) {
            this.dono = dono;
    }

    public void brincar() {
            System.out.println("Brincando com " + this.dono);
    }

}
```

Implementando interfaces

- Diferentemente de classes abstratas, onde você também pode herdar implementação, em uma interface você herda apenas o contrato, as assinaturas dos métodos;

- Em vez de usar "extends", em interfaces nós usamos "implements", então no caso de interfaces nós estamos mesmo herdando comportamentos;

- Um mesmo objeto pode implementar várias interfaces, ele não precisa implementar apenas uma;

Usando interfaces

```
public class CachorroTest {

   public static void main(String[] args) {
         BichoDeEstimacao bicho = new Cachorro();

         bicho.setDono("José");
         bicho.brincar();

   }

}
```

Onde mais eu posso ver interfaces?

- Coleções
  - Collection
    - List
      - ArrayList
      - LinkedList
    - Set
      - HashSet
      - LinkedHashSet
    - Map
      - HashMap,
      - LinkedHashMap

Quando usar interfaces

- Quando o problema levar você para o diamante da morte :P

- Quando você quiser que classes diferentes respondam a um mesmo contrato, independente da implementação delas (“Se fala, anda, nada e voa como um pato, é um pato!”) ;

- Quando você quer evitar “amarrar” a supeclasse de um conjunto de classes desnecessariamente;

Exercício

- Crie duas implementações para a interface abaixo:

```
public interface BancoDeDados {
        public void adicionar( Pessoa pessoa );
        public void remover( Pessoa pessoa );
        public List<Pessoa> listar();
        public Pessoa encontrarPessoaPeloNome( String nome );
        public Pessoa encontrarPessoaPeloCPF( String cpf );
}
```

Exercício - detalhes

- Uma das implementações deve manter os dados em uma coleção dentro do objeto (uma LinkedList ou ArrayList);
- A outra implementação deve escrever o resultado das operações em um arquivo. Esse arquivo pode ter o formato que você quiser, mas todas as operações devem ser refletidas nesse arquivo;
- Pra escrever em arquivo você tanto pode criar o seu próprio formato, como pode usar ObjectInputStream, ObjectOutputStream e Serializable;

Exercício – mais detalhes

- Quem entregar o exercício usando um arquivo estilo CSV ou XML ganha uma ajudinha na nota;

- Pra lidar com arquivos em Java, as classes são File, InputStream, OutputStream, Reader e Writer;