PROGRAMAÇÃO ORIENTADA A OBJETOS

AULA 12 – CLASSES ABSTRATAS E INTERFACES

WALTER TRAVASSOS SARINHO

@PROF.WALTERTRAVASSOS

WALTER.TRAVASSOS@SECTRAS.EDU.BR

Dúvidas sobre a aula passada

Mapa de Estudos

AULA 01
Paradigmas de Programação e história do Java.
IDEs e Hello Universe

AULA 02
Manipulação de datas.
Sintaxe da Linguagem Java e variáveis primitivas e de referência.

AULA 03
Visibilidade de variáveis, classe String e casting.

AULA 04
Classes Wrapper.
Operadores e métodos de conversão.

AULA 05
Formatar a saída de dados numérica.
Operador condicional if.
Métodos para comparar Strings

Avaliação 1

Java

Mapa de Estudos

AULA 06
Manipulação de Strings. Operador condicional switch.

AULA 08
Vetores (for each) e matrizes. Classe ArrayList

AULA 10
Construindo métodos e Métodos construtores

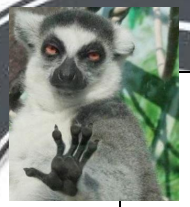
Avaliação 2

AULA 07
Operadores de atribuição compostos. Estruturas de repetição: while, do while e for.

AULA 09
O paradigma Orientado a Objetos

Java

Mapa de Estudos



AULA 12
Classes Abstratas e Interfaces

AULA 11
Pilares da POO: Abstração, Encapsulamento, Herança e Polimorfismo

Avaliação 3

Java

Aula de Hoje

- Apresentar o conceito de classes abstratas.
- Discutir a percepção de herança entre a hierarquia de classes abstratas.
- Implementação de interfaces.


Classes
Abstratas

Definição

- Em Java, uma **classe abstrata** é um tipo especial de classe que serve como base para outras classes, **mas não pode ser instanciada diretamente** (não podemos criar objeto dessa classe).
- A classe abstrata define características e comportamentos comuns que serão herdados por suas subclasses, as chamadas classes concretas.

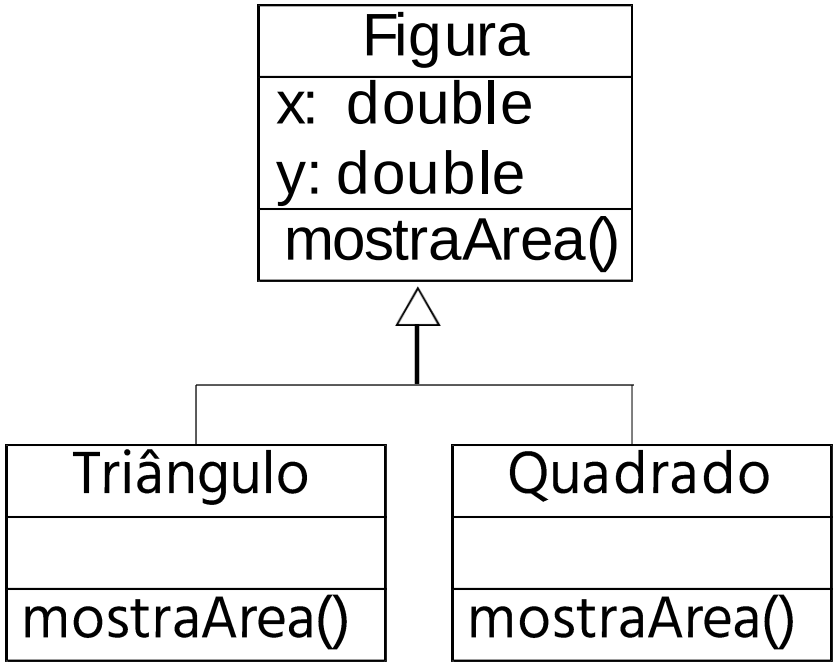
Classe abstrata

- Por **não poder ser instanciada**, não podemos criar objetos de classes abstratas.
- Apenas as **subclasses** (as classes filhas concretas) de uma classe abstrata **poderão ser instanciadas.**

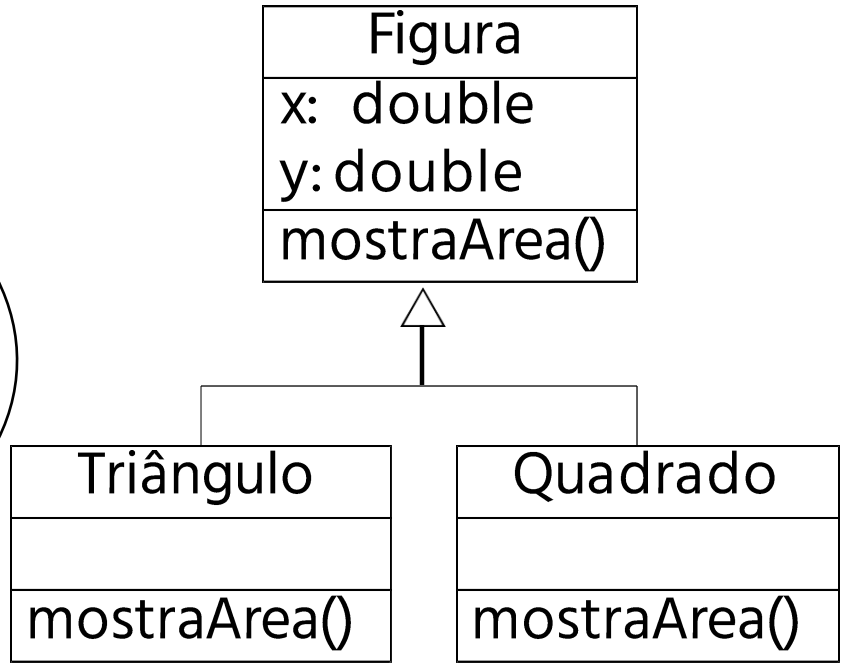
- É utilizada para manter a consistência de implementações.
- Em Java, para definir uma classe abstrata, utilizamos a palavra reservada **abstract** na declaração da classe!

Classes Abstratas

Considerando o exemplo da aula anterior:



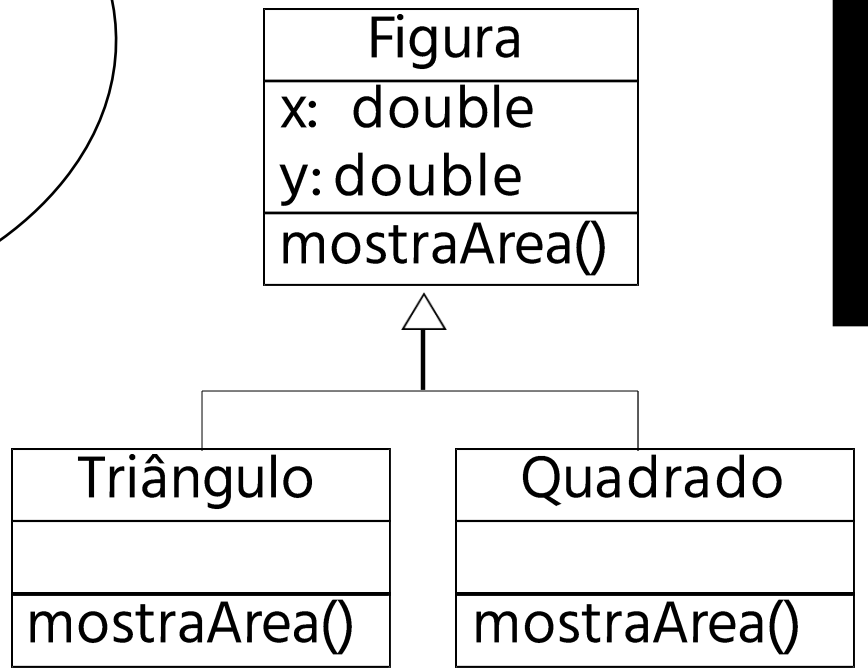
Classes Abstratas



A classe Figura não era instanciada (não criamos objetos desta classe).

Definida apenas para questões de herança e polimorfismo.

Classes Abstratas



Esta classe é uma **forte candidata** a ser uma **classe abstrata**!

Implementação: modificador abstract

```
public abstract class Figura {
        private double x;
        private double y;

        public double getX ( ) {
                return x;
        }
        // ....
}
```

Métodos Abstratos

Métodos Abstratos

- Todas as classes concretas que são filhas de uma classe abstrata deverão **reescrever (sobrescrever)** métodos que forem classificados como abstratos.
- É uma espécie de "**polimorfismo obrigatório**".
- É como se as classes filhas concretas tivessem a responsabilidade de implementar aquele método!

Métodos Abstratos

- Em Java, para definir um **método abstrato**, utilizamos o modificador abstract na sua **assinatura**.
- O método abstrato **não possui corpo**.
- Coloca-se ; aos invés do { e }

- Basicamente, colocamos o modificador **abstract** na **assinatura do método**!
- Uma vez que adicionamos um método abstrato na classe, essa classe precisa se tornar abstrata.

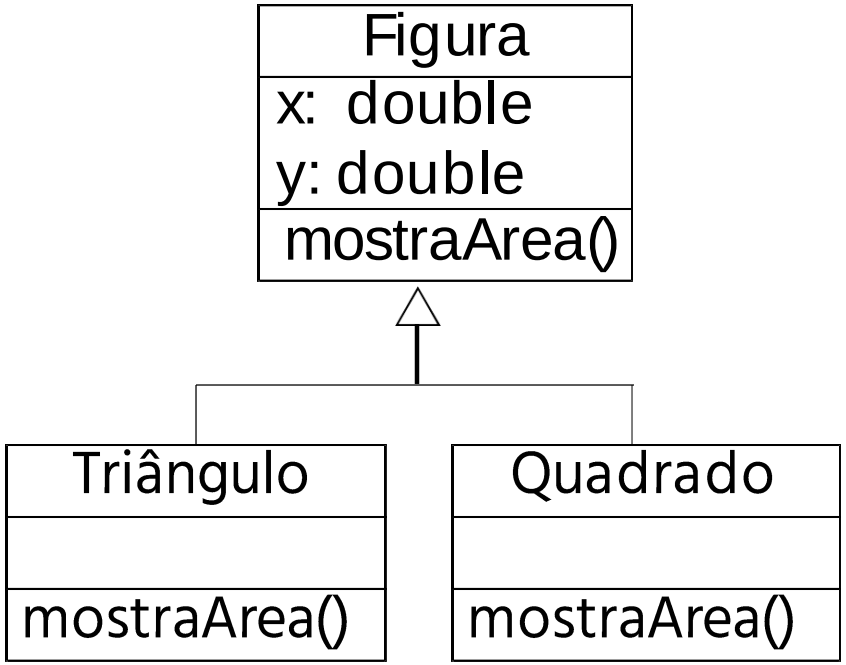
Relembrando

- **Classes abstratas** possuem o modificador abstract em sua declaração, por exemplo, **public abstract class Pessoa { }**

- **Métodos abstratos** também utilizam o modificador abstract
- Estes métodos **não possuem corpo**. Assim, declaramos apenas a **assinatura do método** (contendo o modificador abstract) na classe, por exemplo, **public abstract void escreverNome();**

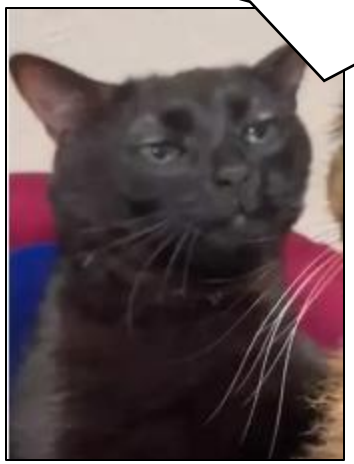
Métodos Abstratos

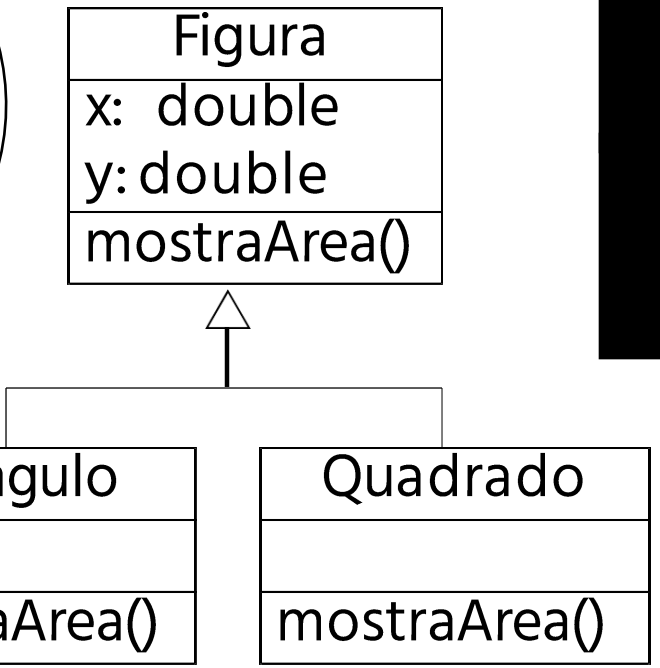
Voltando ao exemplo do diagrama de classes:



Métodos Abstratos

NÃO FAZ SENTIDO IMPLEMENTAR O MÉTODO MOSTRARAREA() DA CLASSE FIGURA.
NÃO CONSIGO IMAGINAR A FIGURA, QUEM DIRÁ UM MÉTODO PRA CALCULAR A ÁREA DESSA FIGURA.



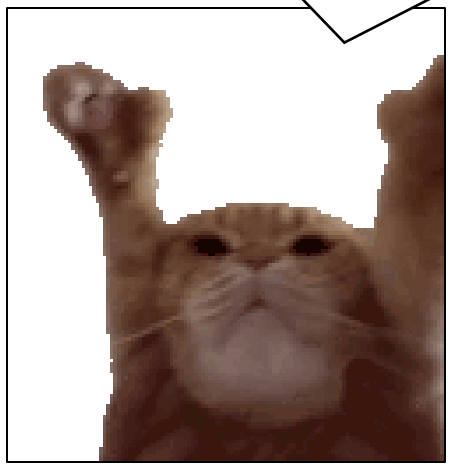


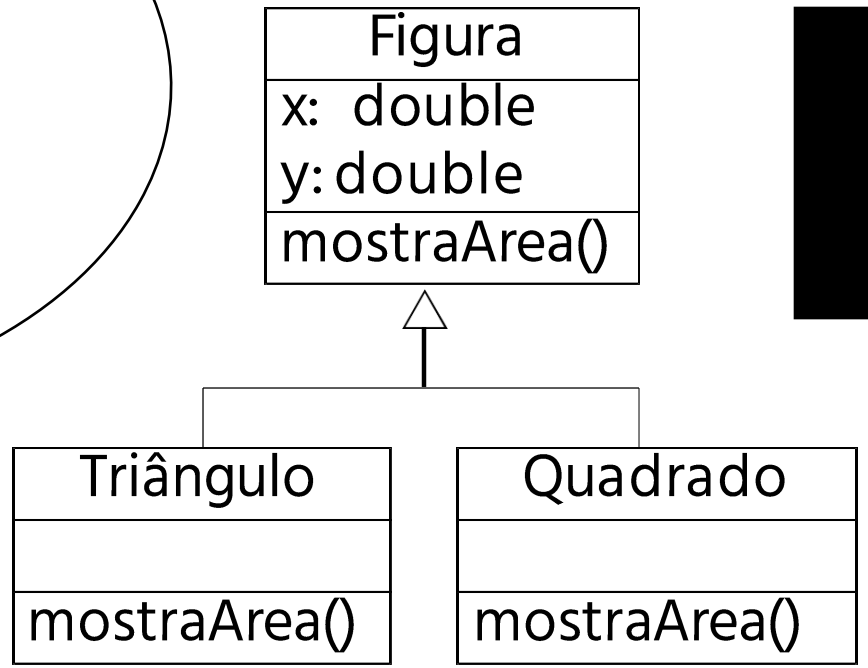
O método **mostraArea() da classe Figura** "não fazia nada" (não possuía instruções a serem executadas).

Ele foi definido apenas para questões de polimorfismo.

Métodos Abstratos

COMO A CLASSE FIGURA POSSUI UM MÉTODO ABSTRATO, AS CLASSES FILHAS GANHAM A RESPONSABILIDADE PARA IMPLEMENTAR ESSES MÉTODOS.





Este método é um **forte candidato** a ser um **método abstrato**!

Implementação: Métodos Abstratos

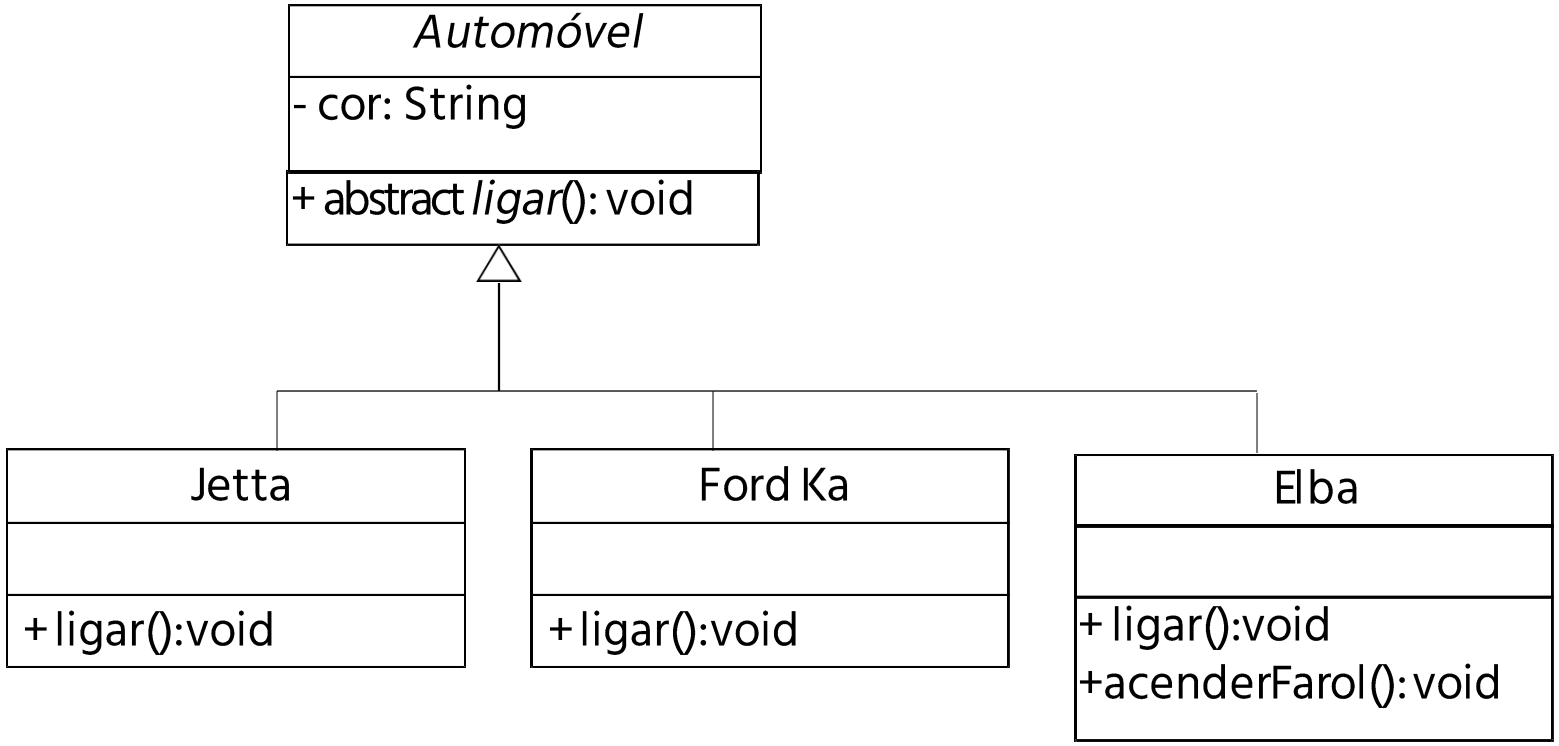
```
public abstract class Figura {
        private double x;
        private double y;

        public abstract void mostraArea ();
        // ...
}
```

Mais um detalhe...

Classes abstratas podem possuir tanto métodos abstratos, quanto métodos concretos (implementados).

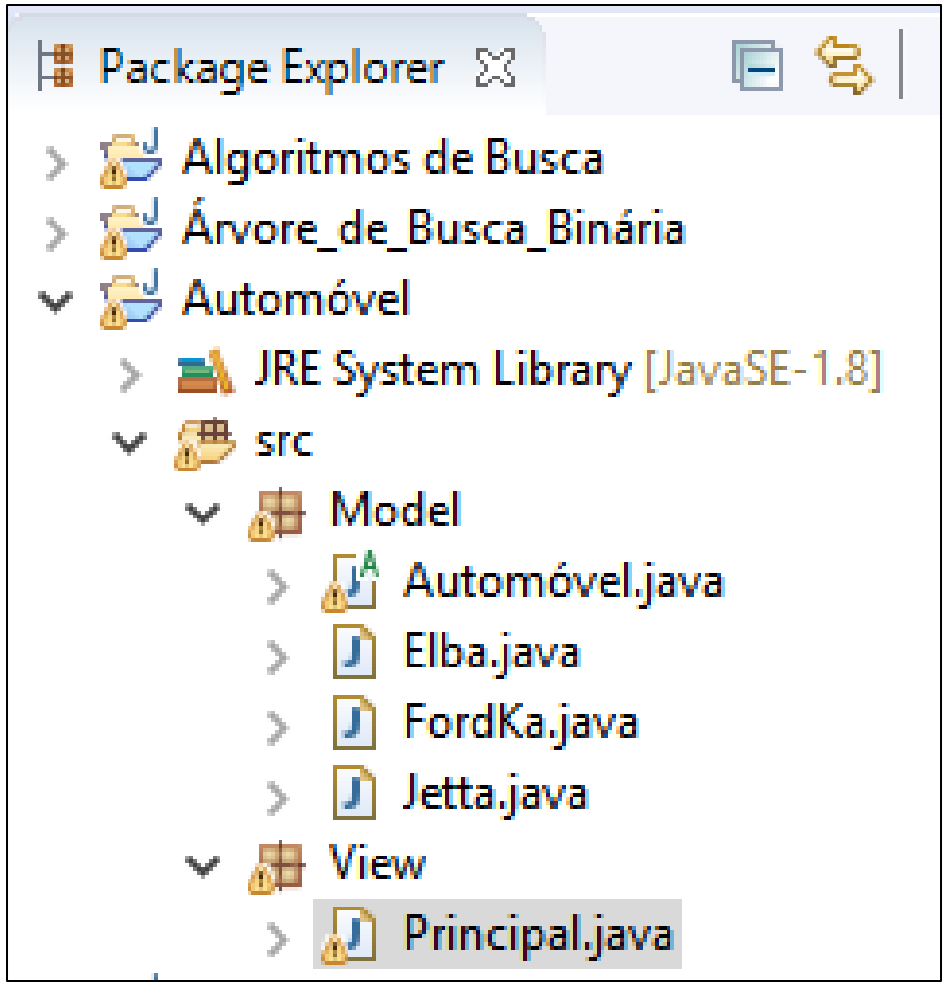
Exercício Resolvido

Diagrama de Classes



Estrutura do Projeto

- 2 packages: **Model** e **View**
- 5 classes
- A classe **Principal** contém o método main
- As classes **Elba**, **Jetta** e **FordKa** são filhas de **Automóvel**.



Pacote Model

Automóvel.java | Jetta.java | FordKa.java | Elba.java

```java
package Model;

public abstract class Automóvel {

    private String cor;

    public abstract boolean isAlarm();

    public abstract void ligar();


}
```

Automóvel.java | Jetta.java | FordKa.java | Elba.java

```java
package Model;
public class Elba extends Automóvel {

    @Override
    public boolean isAlarm() {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public void ligar() {
        System.out.println("Pufffff!");

    }

    public void acenderFarol() {
        System.out.println("farol aceso");
    }

}
```

Automóvel.java | Jetta.java | FordKa.java | Elba.java

```java
package Model;
public class Jetta extends Automóvel{

    @Override
    public boolean isAlarm() {
        // TODO Auto-generated method stub
        return true;
    }

    @Override
    public void ligar() {
        System.out.println("Vrummmmmmmm");

    }

}
```

Automóvel.java | Jetta.java | FordKa.java | Elba.java

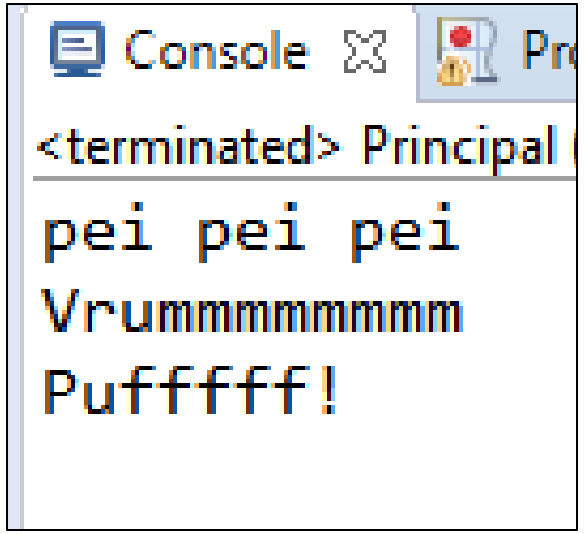
```java
package Model;
public class FordKa extends Automóvel {

    @Override
    public boolean isAlarm() {
        // TODO Auto-generated method stub
        return true;
    }

    @Override
    public void ligar() {
        System.out.println("pei pei pei");
    }


}
```

Pacote view

Ao executar, veremos as chamadas ao método **ligar()** implementadas de forma diferente nas 3 classes filhas.

Automóvel.java | Jetta.java | FordKa.java | Elba.java | *Principal.java

```
1  package View;
2
3  import Model.*;
4
5  public class Principal {
6
7      public static void main(String[] args) {
8          FordKa a1 = new FordKa();
9          a1.ligar();
10         Jetta a2 = new Jetta();
11         a2.ligar();
12         Elba a3 = new Elba();
13         a3.ligar();
14     }
15
16 }
17
```



Relacionamento de Herança

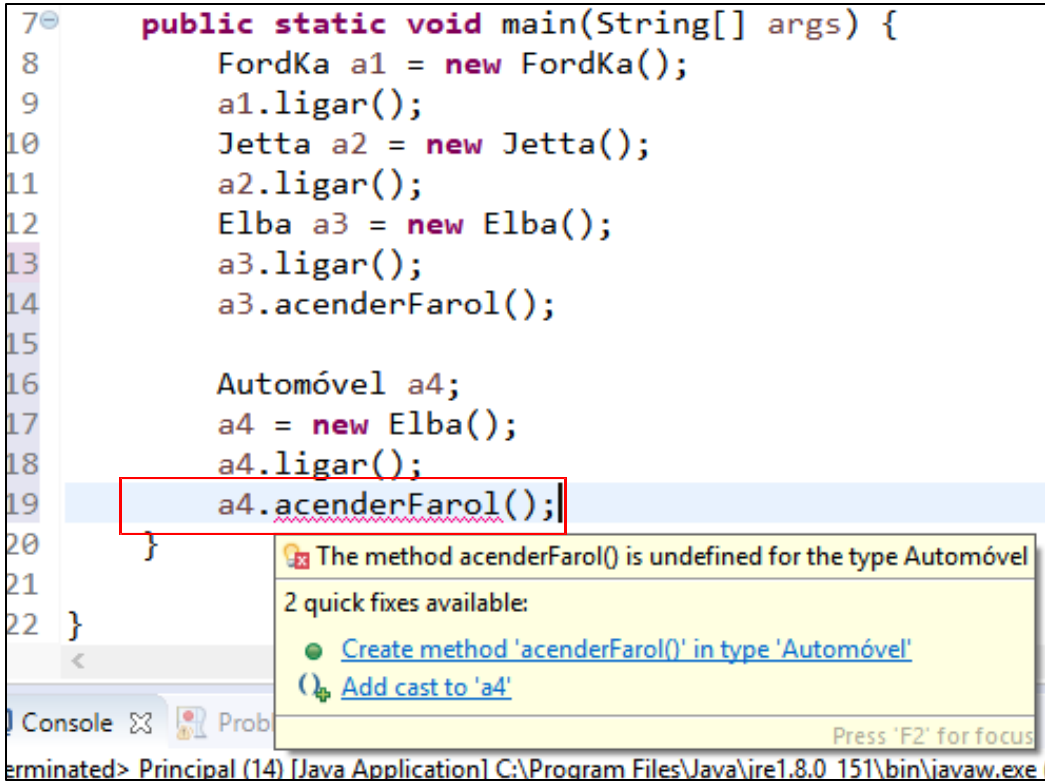
- E se criarmos uma **variável da classe automóvel** e adicionássemos um **objeto de uma das classes filha...**
- Vai funcionar?
- SIM! É um **upcasting**!

```
5  public class Principal {
6
7      public static void main(String[] args) {
8          FordKa a1 = new FordKa();
9          a1.ligar();
10         Jetta a2 = new Jetta();
11         a2.ligar();
12         Elba a3 = new Elba();
13         a3.ligar();
14
15         Automóvel a4;
16         a4 = new Elba();
17         a4.ligar();
```

- Conseguimos até chamar o método **ligar()**, no entanto....

Relacionamento de Herança

- Não conseguimos chamar métodos que existem apenas na classe Elba e que não pertençam a classe pai.
- O método acenderFarol() que só pertence a classe Elba não poderá ser chamado enquanto o objeto **new Elba()** estiver numa variável da classe pai Automóvel.
- O método acenderFarol() só pode ser acionado quando o objeto new Elba() está numa variável da classe Elba!



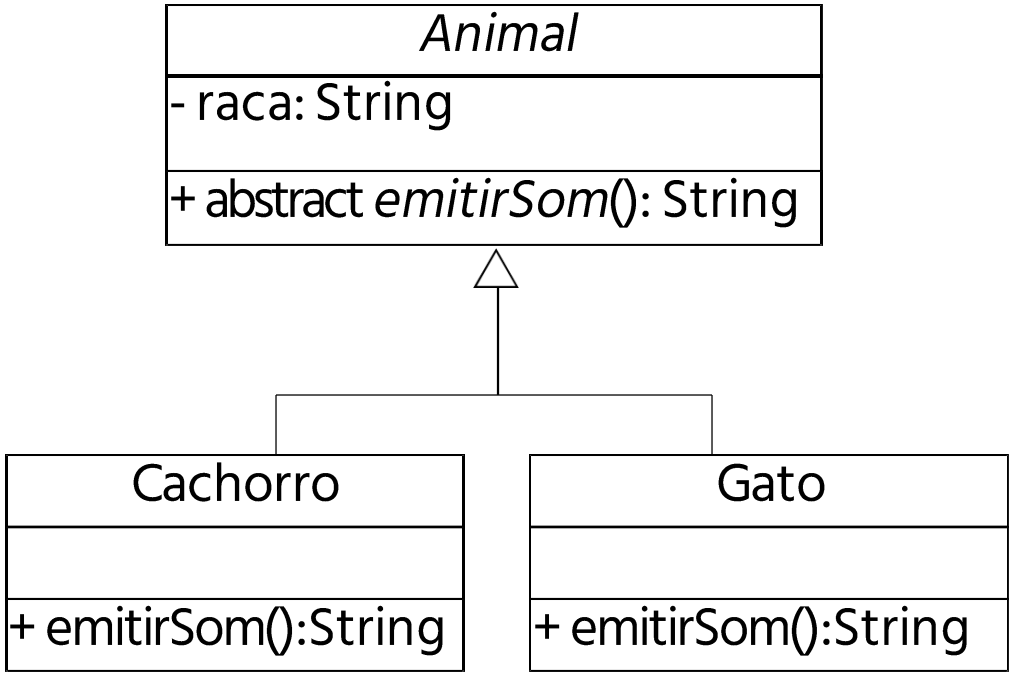
Mais Exercícios

Exercício 1

Implemente as classes:

O método **emitirSom()** exibe no console o barulho de cada objeto do diagrama de classe.

Todos as variáveis de instância são privadas.
Crie os métodos get e set necessários.



Exercício 1

- Crie uma classe Java chamada Principal, que...
- Terá o método main implementado.
- Irá instanciar as classes Scanner (para receber as entradas) e objetos da classe Cachorro e Gato.
- Receberá a raça de cada animal.
- Irá enviar os dados para cada objeto.
- Irá exibir a raça e o barulho que cada animal faz.

Exercício 1 - Resposta

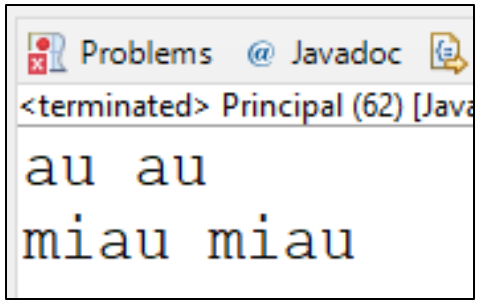
```java
package animais;

public abstract class Animal {
    private String raça;

    public abstract void emitirSom();

}
```

```java
package animais;

public class Cachorro extends Animal {

    @Override
    public void emitirSom() {

        System.out.println("au au");
    }
}
```

```java
package animais;

public class Gato extends Animal {

    @Override
    public void emitirSom() {

        System.out.println("miau miau");
    }
}
```

Exercício 1 - Resposta

```java
package view;

import animais.*;

public class Principal {

    public static void main(String[] args) {

        Animal a1 = new Cachorro();
        a1.emitirSom();

        Animal a2 = new Gato();
        a2.emitirSom();
    }
}
```



Intervalo: 15 minutos

Interfaces

Sugestões de Referências

Use a cabeça Java! Java. Capítulo 8, páginas: 147-172.

Vídeo aula de Loiane Groner:

https://www.youtube.com/watch?v=6uLLfRNgRA4

CAELUM. Java e Orientação a Objetos. Disponível em:
https://www.caelum.com.br/apostila-java-orientacao-objetos/

DEVMEDIA. Entendendo interfaces em Java. Disponível em:
https://www.devmedia.com.br/entendendo-interfaces-em-java/25502

O que é uma interface?

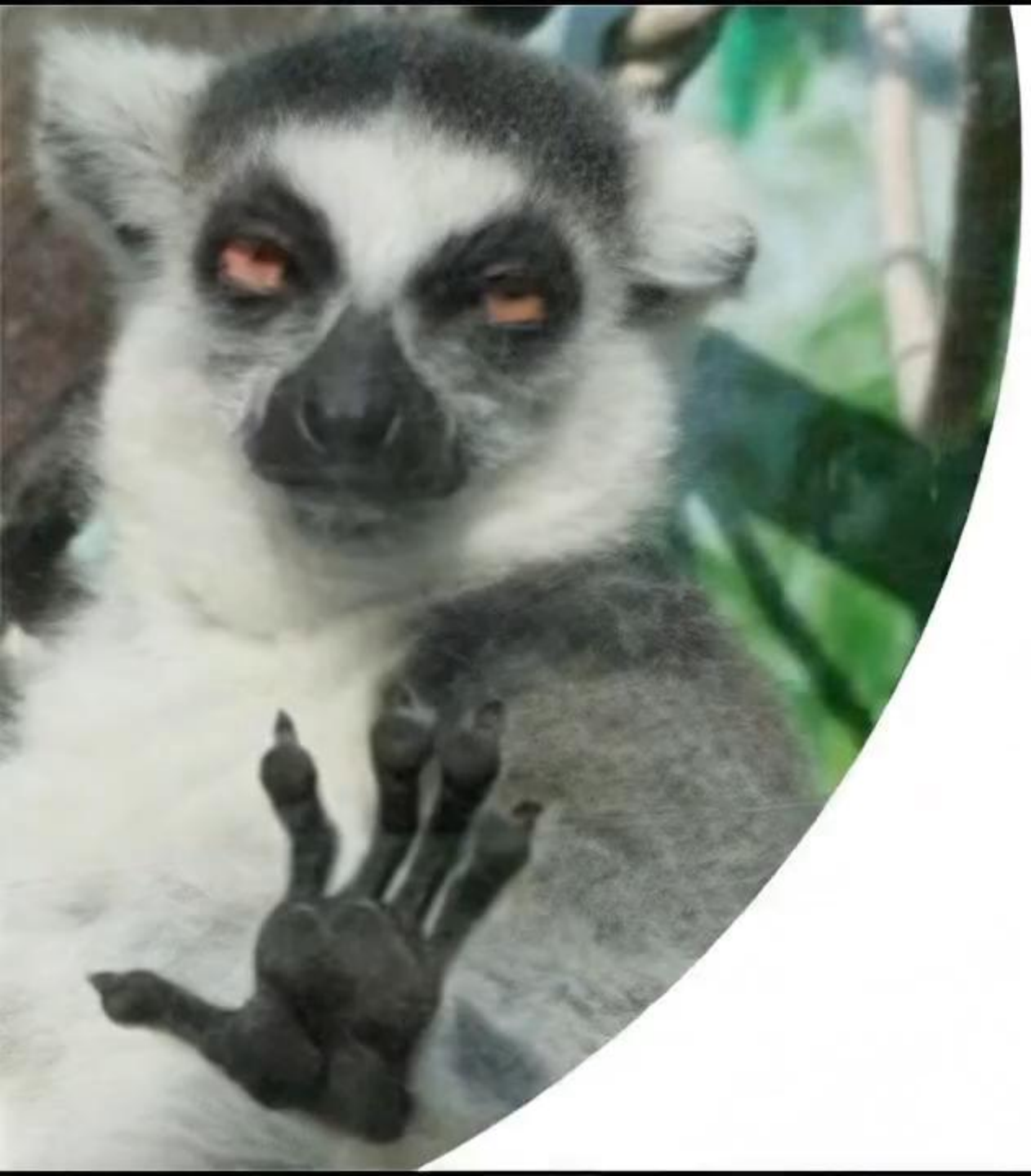
É uma classe 100% abstrata!

O que é uma classe abstrata?

É uma classe que não pode ser instanciada!

E ISSO SERVE PRA QUE?

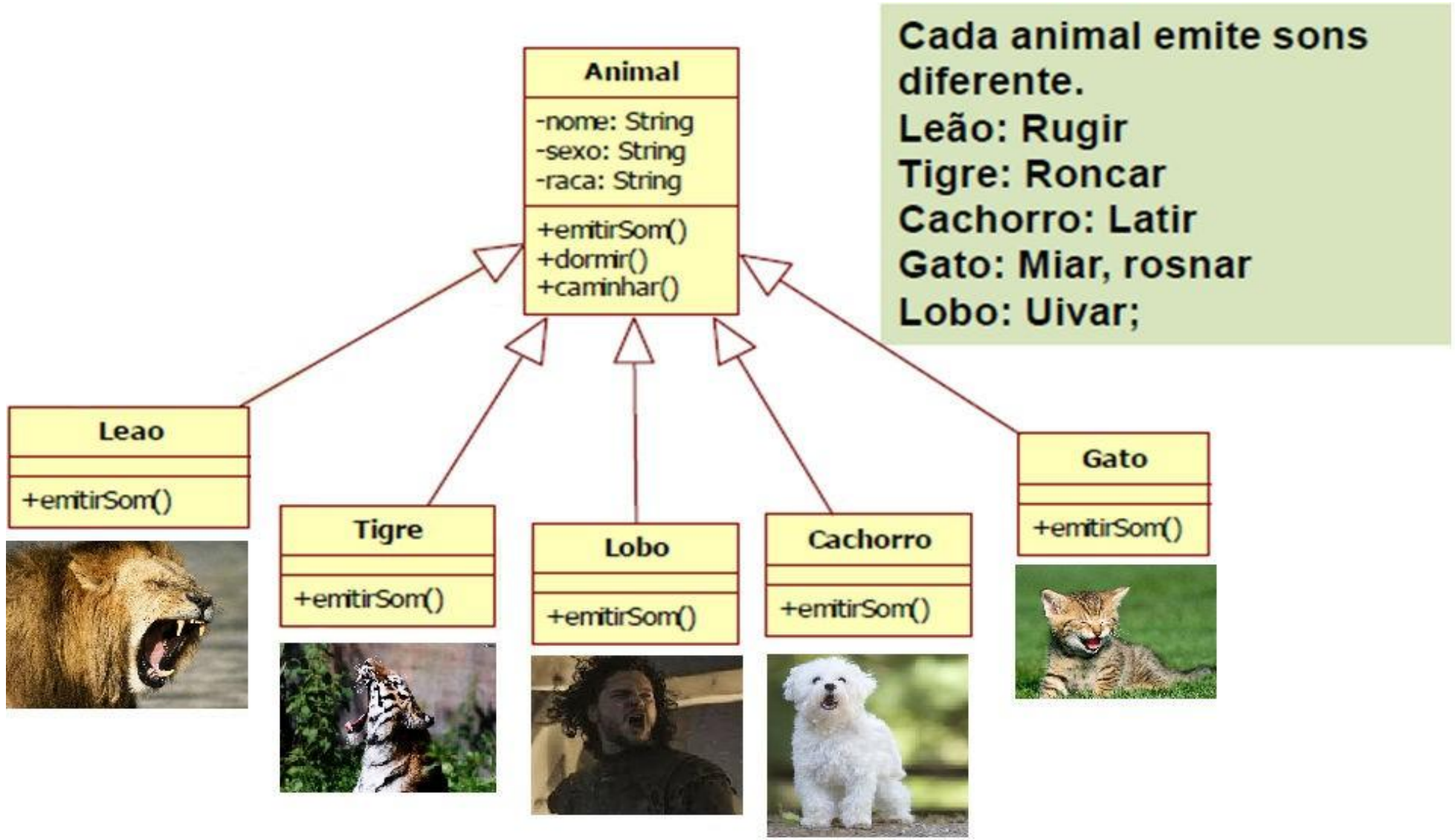
Não entendo porque preciso de uma classe abstrata, um método abstrato ou mesmo uma classe 100% abstrata (interface)!



Definição

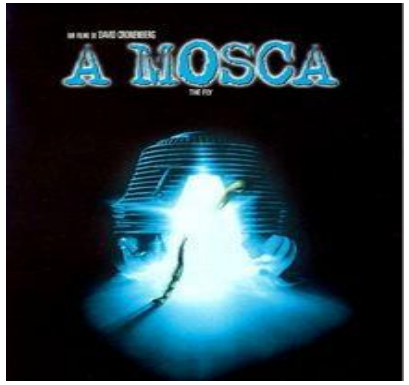
- Uma interface define um conjunto de métodos que um objeto deve implementar.
- Ela funciona como um contrato entre a classe e o mundo exterior, especificando o comportamento que a classe deve fornecer, mas sem detalhar **como** esse comportamento será implementado.
- Em outras palavras, a interface define o "o quê" (os métodos) sem o "como" (a implementação).

Vamos conversar um pouco mais sobre classes abstratas, interfaces e polimorfismo...



Algumas classes não deveriam ser instanciadas!

- **Não faz sentido criar um objeto da classe Animal!**
- Faz sentido criar objetos das classes Leão, Tigre, Lobo...
- Mas tentar criar um objeto da classe Animal, é como ter um pesadelo em que acontece um acidente no teletransporte de Jornada nas Estrelas (**ou no filme A Mosca**)!
- Uma classe abstrata **praticamente*** só tem valor quando for estendida (derivada)!



Qual é a vantagem em ter um método abstrato?

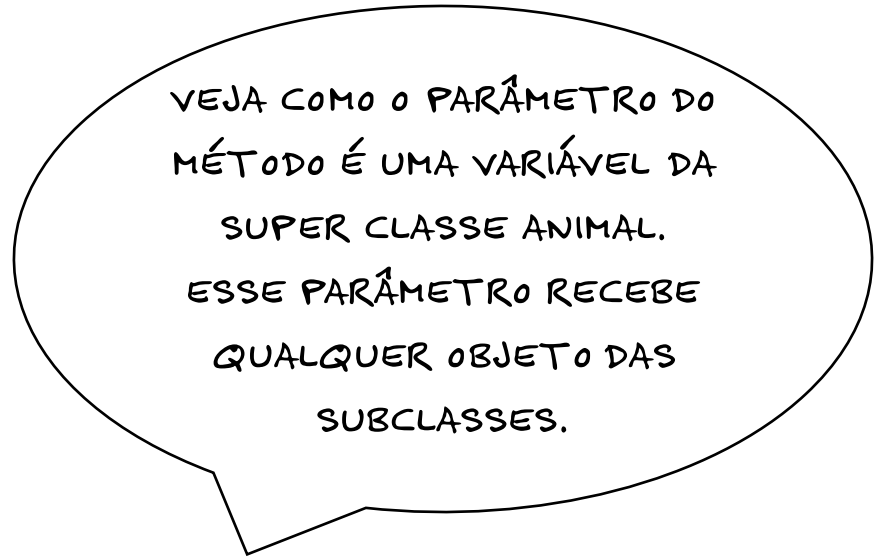
- Uma vantagem é que, embora você **não tenha inserido nenhum código de método real**, você já terá definido **parte do** protocolo para um grupo de subtipos (subclasses).
- Lembre-se do **polimorfismo**!
- O que queremos é a possibilidade de usar **uma variável do tipo da superclasse (Animal** por exemplo) como **parâmetro, tipo de retorno de um método ou como variável de instância – princípios SOLID).**
- Assim, você poderá criar outros subtipos (outros animais) a sua superclasse sem precisar reescrever ou adicionar novos métodos. Ou seja... Criar novos filhos no diagrama de classe vai permitir uma **ampliação** do comportamento.

Por exemplo, se criarmos mais uma classe filha da classe Animal, como a classe Guiné...

```
public abstract class Animal {
        public abstract void emitirSom();
}

public class Guiné extends Animal {
        public abstract void emitirSom() {
                System.out.println("Tofraco! Tofraco!");
        }
}
```

Em outros métodos da aplicação, poderemos continuar usando a variável da superclasse para acomodar os objetos das subclasses.

```
public class Principal {
        public static void main (String[] args) {
                Guiné g1 = new Guiné();
                conversarComAnimal(g1);
        }
        public static void conversarComAnimal(Animal a) {
                System.out.println("Bom dia "+a.getNome() );
                System.out.println( a.emitirSom() );
        }
}
```



Assim...

Quando usamos um **método abstrato**, estamos dizendo: **todos os subtipos (subclasses) desse tipo (superclasse) terão ESSE método!**

Não esqueça: Quando a assinatura de um método abstrato existe na superclasse abstrata, ao criarmos objetos de classes filhas e implementarmos o método nessas classes (filhas), poderemos utilizar a variável da superclasse com objetos das classes filhas para acessar o método que vai conter o comportamento específico de cada classe.

E as interfaces?

Interface

- A interface "obriga" um **determinado grupo de classes** a ter métodos em comum.
- Funciona como uma espécie de **contrato** que, quando assumido por uma classe, deve ser implementado.

Interfaces

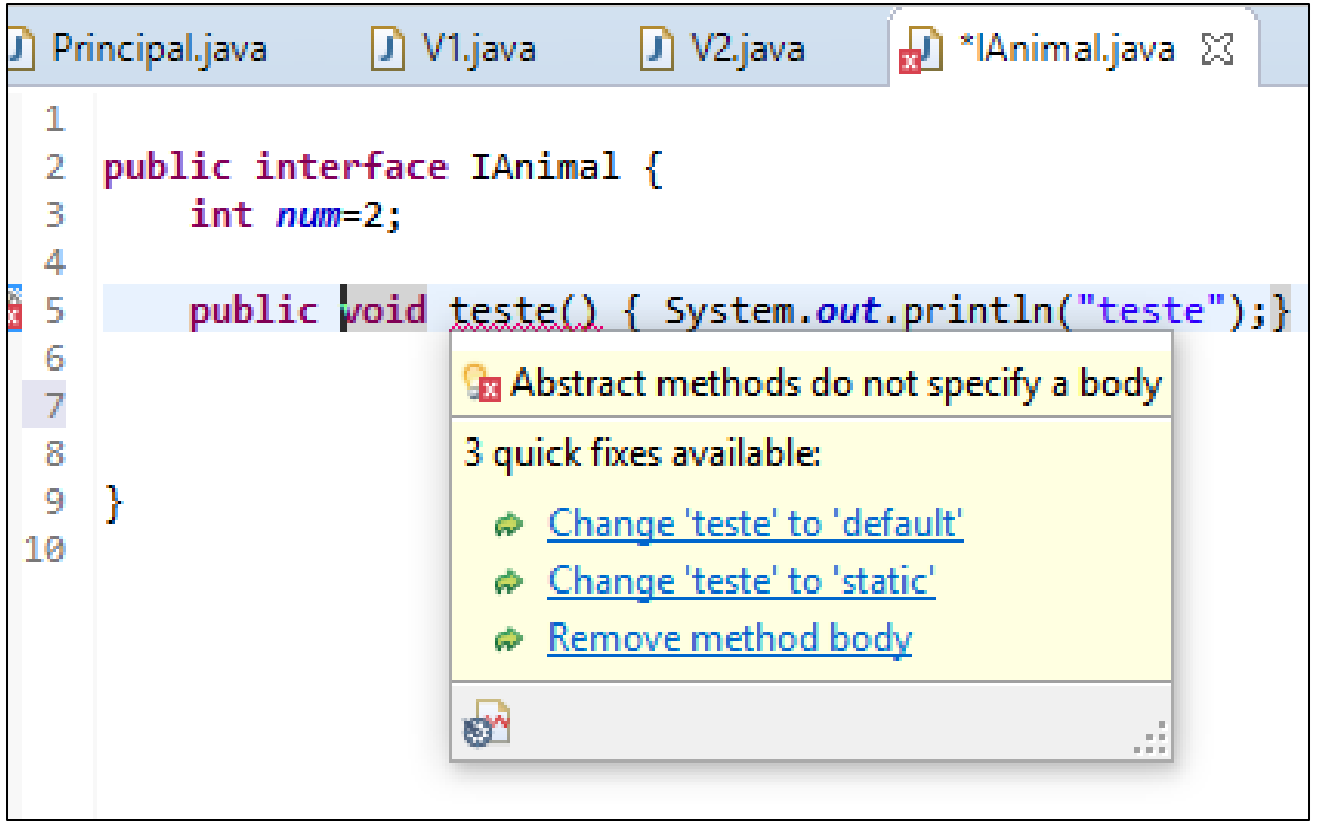
- Até o lançamento do Java 8 (2014), eram apenas as **assinaturas de métodos** que faziam parte do **corpo de interfaces**.
- As **classes** que **seguem este "contrato"** (ou seja, que implementam a interface) deverão implementar o comportamento dos métodos.
- Em Java, para definir uma interface, utilizamos a palavra reservada **interface** em sua especificação (ao invés de utilizar **class**, usamos **interface**).

Default Methods
Métodos Padrão

- No Java 8 é que foram introduzidos os **métodos padrão (default methods)** em interfaces. Antes dessa versão, as interfaces Java **só podiam conter assinaturas de métodos, sem implementações.**
- Com a introdução dos métodos padrão, as interfaces agora podem ter métodos concretos (com implementação) além de métodos abstratos.
- A ideia por trás dos métodos padrão é permitir que interfaces evoluam sem quebrar a compatibilidade com versões anteriores. Ou seja, mesmo que uma nova funcionalidade seja adicionada a uma interface, as implementações existentes dessa interface continuarão funcionando sem a necessidade de serem atualizadas.

Default Methods

- Interfaces não podem ter métodos públicos implementados.
- Os métodos padrão utilizam o modificador **default**.



Assinaturas de métodos nas Interfaces

- Numa interface, a assinatura do método não pode ter o modificador default.
- São permitidas apenas assinaturas de métodos públicos.

```
1
2  public interface ITeste {
3
4      default void teste1();
5
6      default void teste() {
7          System.out.println("teste");
8      }
9  }
10
```

Modificadores de Acesso

Mais detalhes em:

http://www.devmedia.com.br/modificadores-de-acesso-do-java/25404

Tabela dos modificadores de acesso

	private	default	protected	public
mesma classe	sim	sim	sim	sim
mesmo pacote	não	sim	sim	sim
pacotes diferentes (subclasses)	não	não	sim	sim
pacotes diferentes (sem subclasses)	não	não	não	sim

Pergunta

Resposta: B

A partir do Java 8 é possível que interfaces possam ter métodos concretos, ‘herdados’ por todos que implementam essa interface, sem quebrar a compatibilidade das classes mais antigas que implementam essa interface. O nome dado a esse recurso é:

A) Expressão Lambda.

B) Default methods.

C) Stream.

D) Method reference.

E) Concret methods.

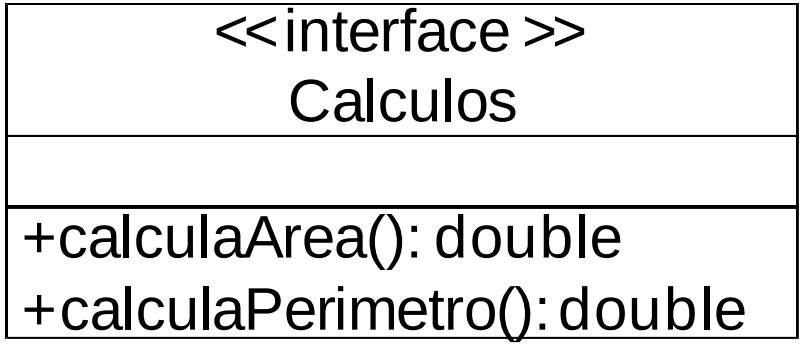
Os métodos padrão são uma adição importante ao Java, pois permitem que as interfaces evoluam de forma mais flexível e sejam mais utilizadas em cenários onde funcionalidades padrão são desejadas sem quebrar o código existente.

Como criar uma interface no eclipse?

File – new - Interface

Interfaces
No diagrama de classes

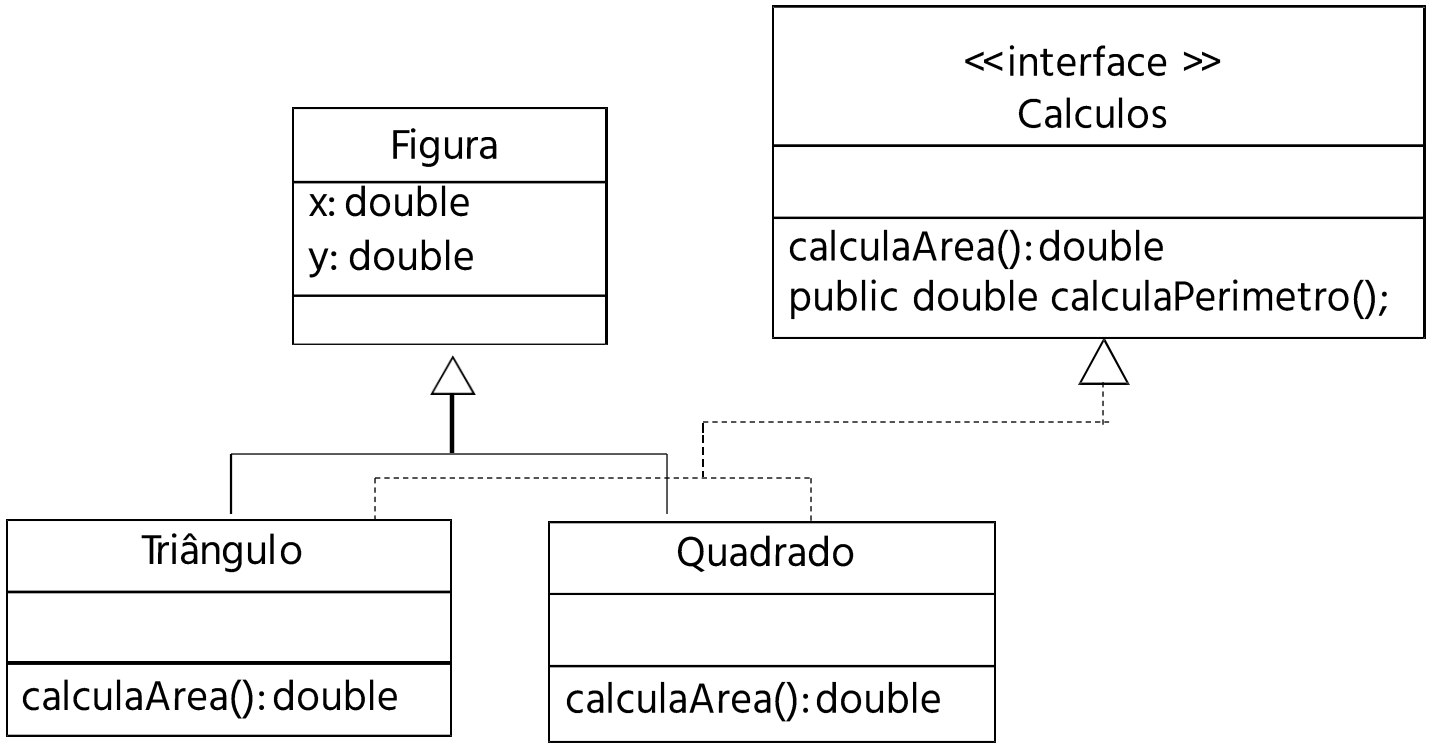
Exemplo:




```
public interface Calculos {
        public double calculaArea();
        public double calculaPerimetro();
}
```

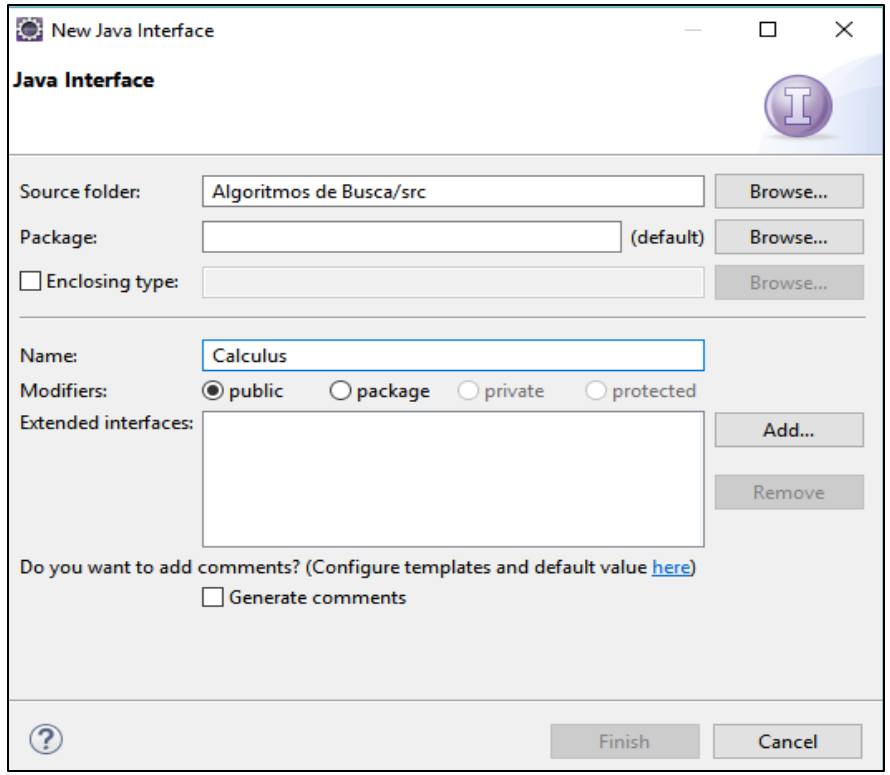
Diagrama de Classes

Exemplo:



Criando a Interface

- Crie uma nova interface:
- File – new – interface. Nome da interface será **Calculos**




```
public interface Calculos {

        public double calculaArea();

        public double calculaPerimetro();
}
```

Interfaces

Para implementar uma interface numa classe, utilizamos a palavra reservada **implements** na declaração da classe, seguida do nome da interface. Por exemplo:

```
public class Quadrado implements Calculos {
        public double calculaArea() {
                // Código para o cálculo da área
        }
        public double calculaPerimetro() {
                // Código para o cálculo do perímetro
        }
}
```

IMPLEMENTS

Interfaces

Caso a classe em questão seja filha da outra classe (esteja utilizando o extends), deveremos utilizar o implements após a definição da extensão. Por exemplo:

```
public class Quadrado extends Figura implements Calculos {
        public double calculaArea() {
                // Código para o cálculo da área
        }
        public double calculaPerimetro() {
                // Código para o cálculo do perímetro
        }
}
```

Interfaces

Ao contrário do uso do extends (uma classe filha só pode ter uma classe pai), **é possível implementar mais de uma interface para a mesma classe**. Por exemplo:

```
public class Quadrado extends Figura implements Calculos, Cores, Textura {

        public double calculaArea() {
                // Código para o cálculo da área
        }
        public double calculaPerimetro() {
                // Código para o cálculo do perímetro
        }
}
```

Resumo sobre Instanciação

Classe Concreta

- Pode ser instanciada.

Classe Abstrata

- **Não** pode ser instanciada.
- Apenas classes filhas de classes abstratas podem ser instanciadas.

Interface

- Não pode ser instanciada.
- Apenas classes que implementam a interface podem ser instanciadas.

Resumo sobre os métodos

Classe Concreta

- Possuem apenas métodos concretos.

Classe Abstrata

- Possuem métodos concretos e métodos abstratos.
- Classes Filhas recebem a responsabilidade de implementação dos métodos abstratos.

Interface

- Possuem assinaturas de métodos públicas ou métodos completos com o modificador default.

Três Observações Finais

Primeira Observação Final

Todas as assinaturas de métodos da interface devem ser públicas (ou públicas e abstratas). As assinaturas de métodos **não podem ser private, default, protected** ou mesmo ter o modificador **final**!

OBS: Para **implementar** um **método padrão (default methods)**, deve-se utilizar o modificador **default**!

```
2  public interface ITeste {
3      default void teste() {
4          System.out.println("teste");
5      }
6  }
7
```

Segunda Observação Final

Todas as **variáveis adicionadas numa interface devem ser inicializadas**!

Além disso, elas funcionarão como constantes (mesmo sem adicionar o modificador final a elas)!

Terceira Observação Final

Você até pode criar uma variável do tipo da interface, no entanto, **não pode criar um objeto do tipo da interface nesta variável.**

Só é possível adicionar nesta variável da interface, **os objetos de classes que implementam esta interface!**

Exemplo

```
public interface Calculos {
        public void teste();
}

public class Principal {
   public static void main (String[] args) {
        Calculus c = new Calculos();
        Calculus c;
   }
}
```

```
//Está errado. Não compila!
/*É possível criar a variável!
Mas não podemos colocar um objeto do tipo da interface dentro da variável.
*/
```

Como fazer para funcionar?
Crie uma classe que implementa a interface!

```
public class Média implements Calculos {
        public void teste() {
                System.out.println("teste");
    }
}
public class Principal {
        public static void main (String[] args) {
                Calculos c = new Média();
        }
}
```

/*Isso é possível! Adicionar na variável do tipo da interface um objeto que implementa a interface Calculus. */

E por fim...

Java não permite herança múltipla entre classes,

...no entanto, as interfaces são um mecanismo que simulam as características de uma herança múltipla.

A herança de várias interfaces é uma herança múltipla verdadeira?

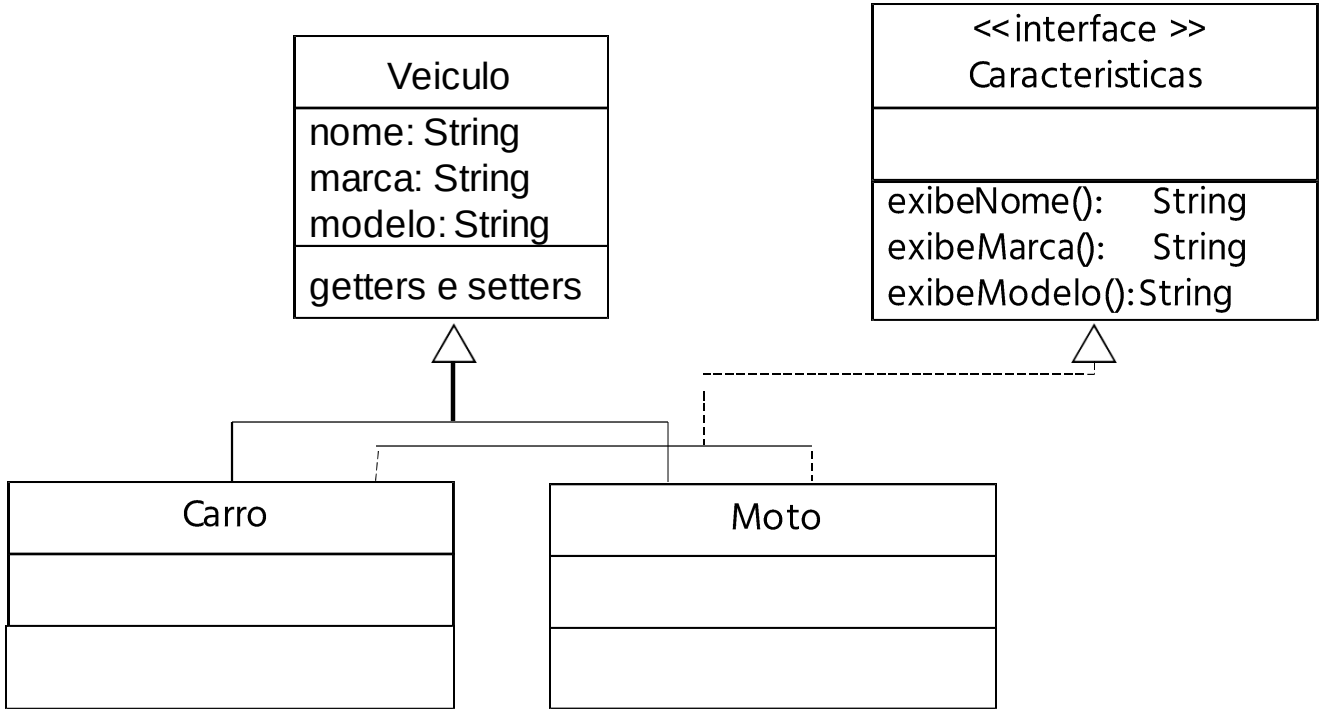
- **A resposta não é tão simples**. Até antes do Java 8, poderíamos afirmar que sim, que não é uma herança múltipla verdadeira.

- No entanto, a herança múltipla de interfaces oferece um tipo de "herança" que simula alguns dos **benefícios** da herança múltipla (ex.: **reuso de métodos e código**), mas sem os problemas associados a ela (ex.: **violação do SRP** (Princípio da Responsabilidade Única) e a ambiguidade de métodos em interfaces diferentes que definem métodos com mesmo nome e assinatura).

Exercícios

Exercício 1

Implemente as classes abaixo:



Exercício 1

- Crie uma classe Java chamada Principal, que...

- Terá o método main implementado.
- Irá instanciar as classes Scanner (para receber as entradas) e objetos das classes Carro e Moto.
- Receberá os dados de cada veículo.
- Irá enviar os dados para cada objeto.
- Irá exibir as características de cada veículo.

Exercício 1 - Resposta

Principal.java | Veículo.java | Carro.java | Moto.java | Características.java

```
1
2  public abstract class Veículo {
3      private String nome;
4      private String marca;
5      private String modelo;
6
7      public Veículo(String n, String m, String mod) {
8          this.nome = n;
9          this.marca = m;
10         this.modelo = mod;
11     }
12     public String getNome() {
13         return nome;
14     }
15     public void setNome(String nome) {
16         this.nome = nome;
17     }
18     public String getMarca() {
19         return marca;
20     }
21     public void setMarca(String marca) {
22         this.marca = marca;
23     }
24     public String getModelo() {
25         return modelo;
26     }
27     public void setModelo(String modelo) {
28         this.modelo = modelo;
29     }
30 }
```

Principal.java | Veículo.java | Carro.java | Moto.java | *Características.java

```
1
2  public interface Características {
3      public String exibeNome();
4      public String exibeMarca();
5      public String exibeModelo();
6  }
7
```

Exercício 1 - Resposta

Principal.java | Veículo.java | Carro.java | Moto.java | *Características.java

```
1
2  public class Carro extends Veículo implements Características {
3
4      public Carro(String n, String m, String mod) {
5          super(n, m, mod);
6      }
7
8      @Override
9      public String exibeNome() {
10         return "O nome do carro é "+ this.getNome();
11     }
12
13     @Override
14     public String exibeMarca() {
15         return "A marca do carro é "+ this.getMarca();
16     }
17
18     @Override
19     public String exibeModelo() {
20         return "O modelo do carro é "+ this.getModelo();
21     }
22 }
```

Exercício 1 - Resposta

Principal.java | Veículo.java | Carro.java | *Moto.java | *Características.java

```
1
2  public class Moto extends Veículo implements Características {
3
4      public Moto(String n, String m, String mod) {
5          super(n, m, mod);
6      }
7
8      @Override
9      public String exibeNome() {
10         return "O nome da moto é "+ this.getNome();
11     }
12
13     @Override
14     public String exibeMarca() {
15         return "A marca da moto é "+ this.getMarca();   }
16
17     @Override
18     public String exibeModelo() {
19         return "O modelo da moto é "+ this.getModelo(); }
20 }
```

Exercício 1 - Resposta

Principal.java | Veículo.java | Carro.java | *Moto.java | *Características.java

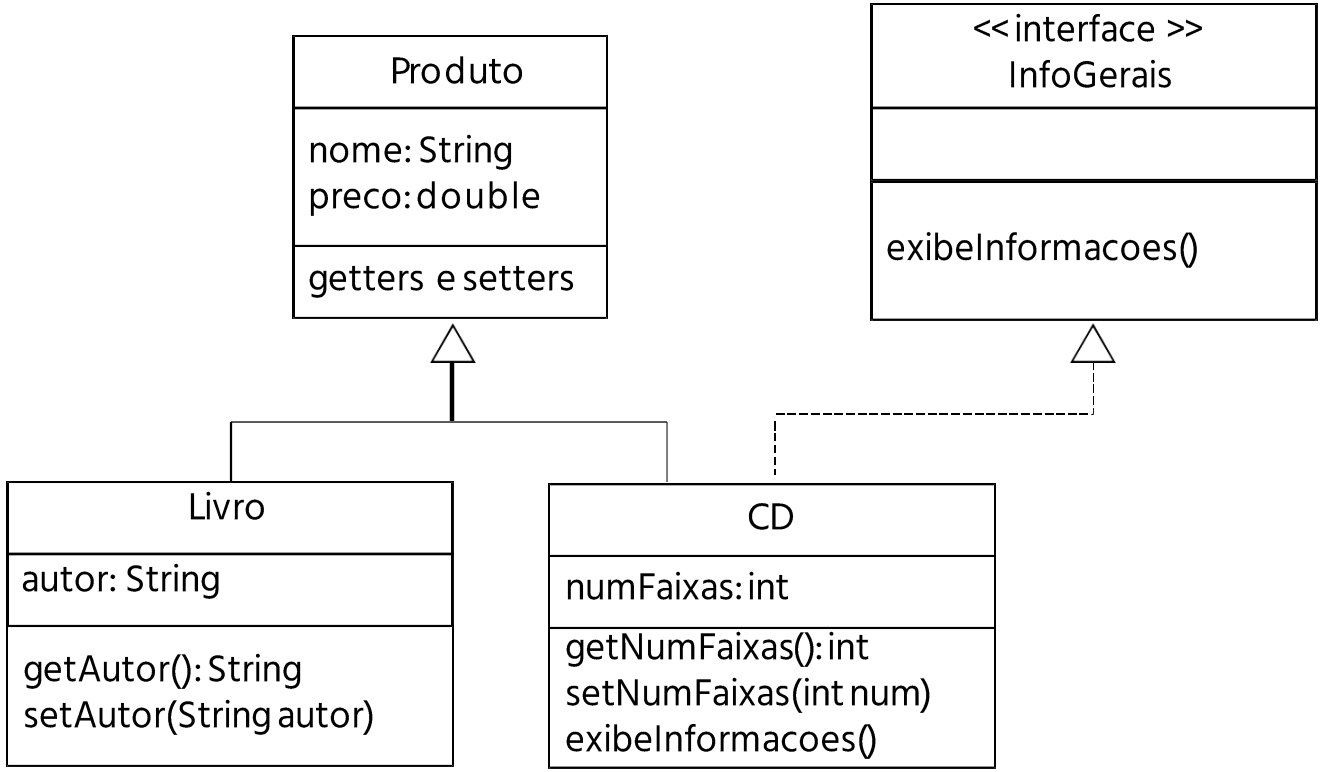
```
1
2  public class Principal {
3
4      public static void main(String[] args) {
5          Carro c = new Carro("Gol","Volkswagen","Mi");
6          System.out.println(c.exibeNome());
7          System.out.println(c.exibeMarca());
8          System.out.println(c.exibeModelo());
9
10         Moto m = new Moto("CG 150","Honda","Titan");
11         System.out.println(m.exibeNome());
12         System.out.println(m.exibeMarca());
13         System.out.println(m.exibeModelo());
14     }
15
16 }
```

Problems | Javadoc | Declaration | Console

```
<terminated> Principal (8) [Java Application] C:\Program Files\Ja
O nome do carro é Gol
A marca do carro é Volkswagen
O modelo do carro é Mi
O nome da moto é CG 150
A marca da moto é Honda
O modelo da moto é Titan
```

Exercício 2

Implemente as classes abaixo:



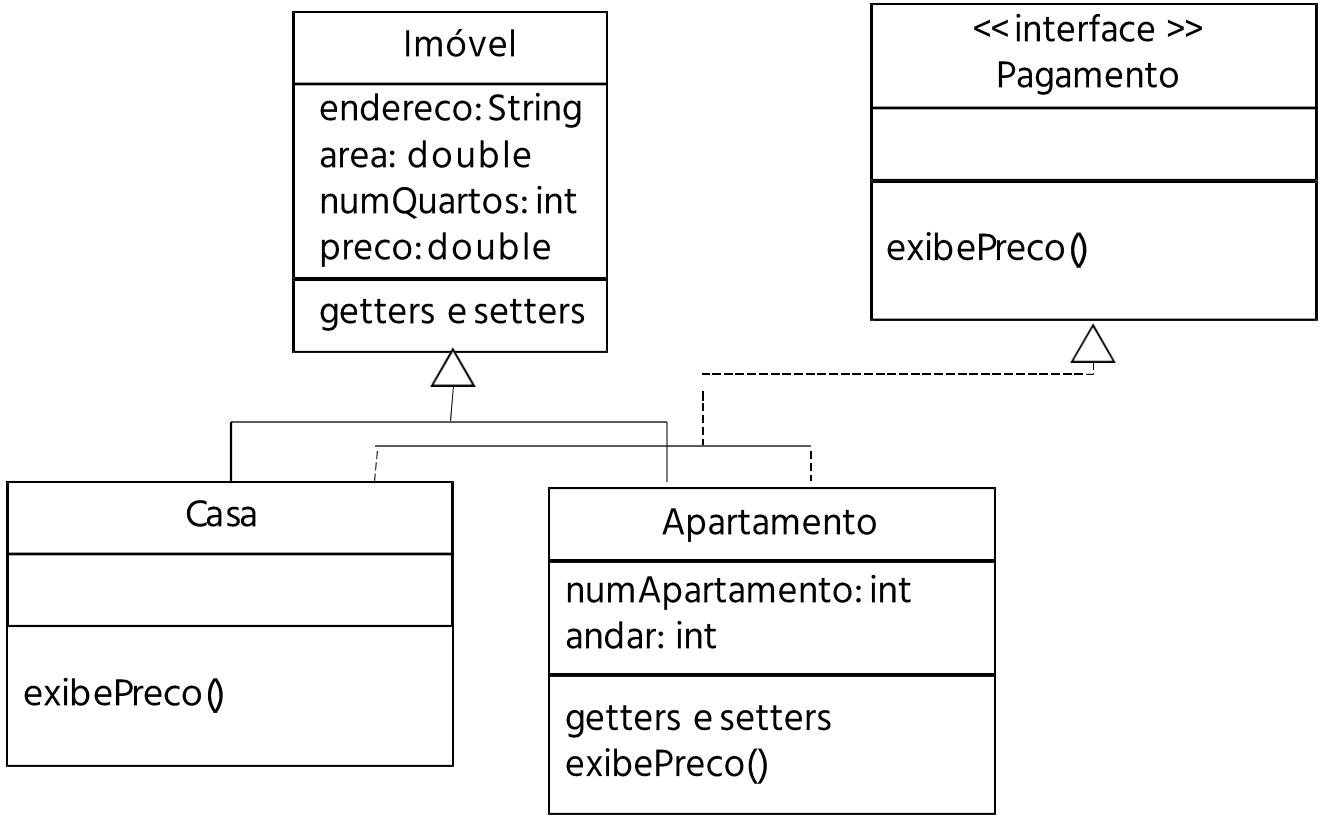
Exercício 2

- Crie uma classe Java chamada Principal, que...

- Terá o método main implementado.
- Irá instanciar as classes Scanner (para receber as entradas) e objetos das classes Livro e CD.
- Receberá os dados de cada produto.
- Irá enviar os dados para cada objeto.
- Irá exibir as informações do CD cadastrado.

Exercício 3

- Implemente as classes abaixo:



Exercício 3

- Crie uma classe Java chamada Principal, que...

- Terá o método main implementado.
- Irá instanciar as classes Scanner (para receber as entradas) e objetos das classes Casa e Apartamento.
- Receberá os dados de cada imóvel.
- Irá enviar os dados para cada objeto.
- Irá exibir as informações de preço de cada imóvel.

Lista de Exercícios

PROGRAMAÇÃO ORIENTADA A OBJETOS

AULA 12 – CLASSES ABSTRATAS E INTERFACES

WALTER TRAVASSOS SARINHO

@PROF.WALTERTRAVASSOS

WALTER.TRAVASSOS@SECTRAS.EDU.BR