Membros estáticos

NomeClasse.nomeMetodoEstatico(argumentos);

Métodos que podem ser invocados sem que um objeto tenha sido instanciado pela classe (sem invocar a palavra chave **new**)

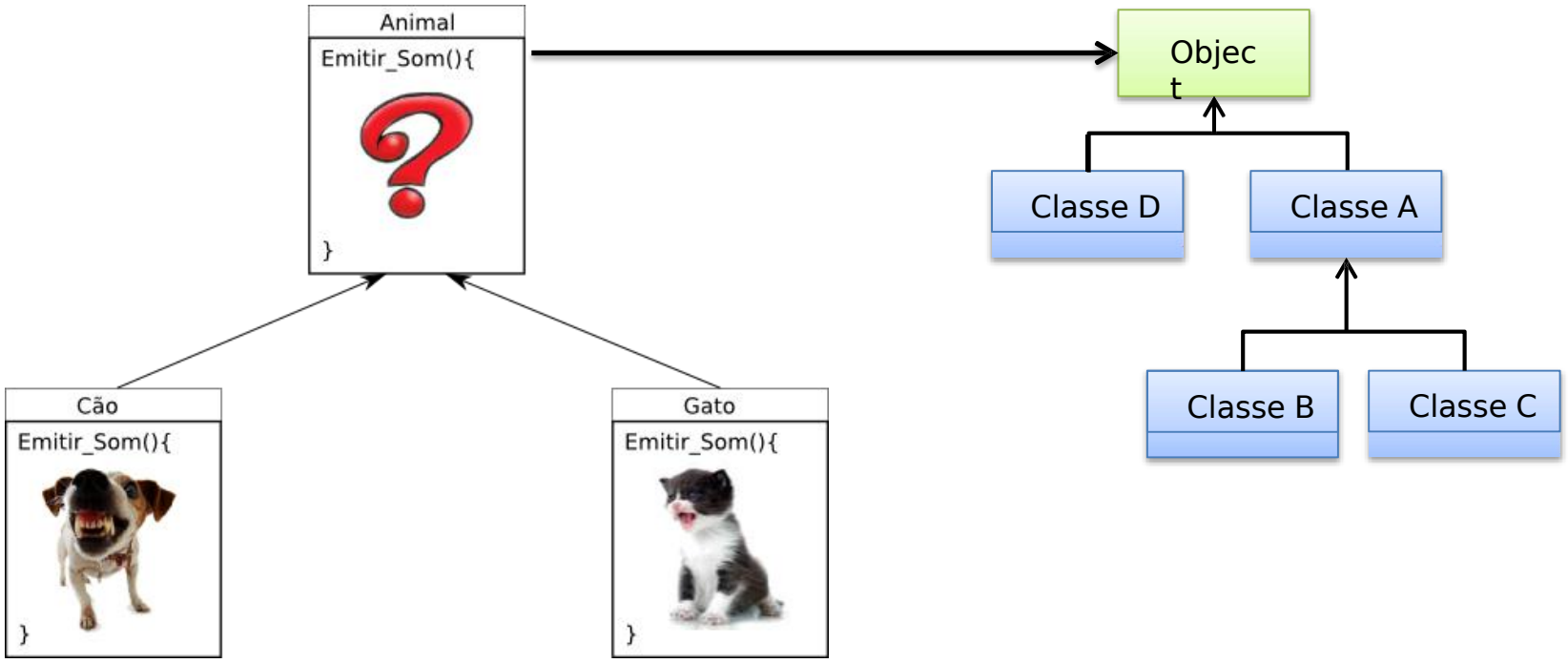
Pertencem à classe como um todo e não a uma instância (ou **objeto**) específico da classe

```
Carro carro1, carro2;

Carro carro3 = new Carro();

carro1 = new Carro();

carro2 = new Carro();

System.out.println(Carro.getContadorInstancia() + " instâncias criadas");
```

Java Orientado a Objetos
Herança e Polimorfismo

Herança

Em Java, todas as classes, incluindo as que formam a API Java, são **subclasses** da classe **Object**



SuperClasse e SubClasse

Diagramas de Classes

Qualquer classe acima de uma classe específica na hierarquia de classes

Generalização

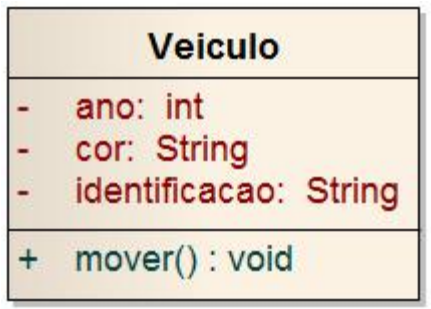
Permite codificar um método apenas uma única vez e este pode ser usado por todas as subclasses

Qualquer classe abaixo de uma classe específica na hierarquia de classes

Especialização

Uma subclasse necessita apenas implementar as diferenças entre ela própria e sua classe pai.

Herança – classe Veiculo

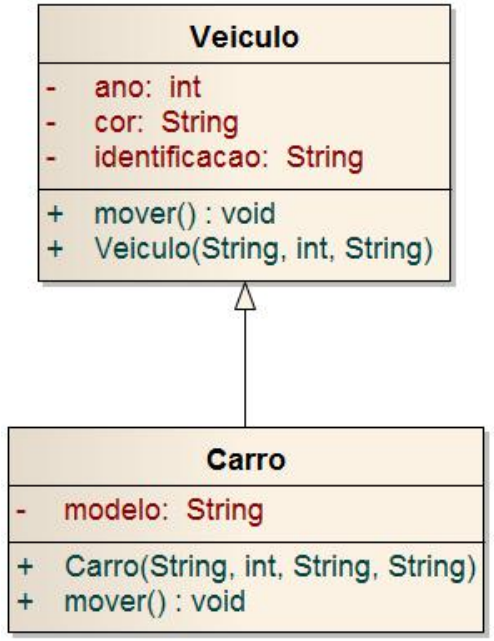



```java
public class Veiculo {// nome da classe

    // (1)Atributos - Variáveis
    private String cor;
    private int ano;
    private String identificacao;

    // (2)Construtor
    public Veiculo( String cor, int ano, String identificacao ) {

        this.cor = cor;
        this.ano = ano;
        this.identificacao = identificacao;
        System.out.println("Criando objeto Veiculo");
    }

    // (3)Métodos
    public void mover() {
        System.out.println("Veiculo se movendo");
    }

}
```

Herança – classe Carro

```
public class Carro extends Veiculo { // nome da classe

    // (1)Atributos - Variáveis
    private String modelo;

    // (2)Construtor
    public Carro( String cor, int ano, String placaIdentificacao, String modelo ) {

        super(cor, ano, placaIdentificacao);
        this.modelo = modelo;

        System.out.println("Criando objeto Carro");
    }

    @Override
    public void mover() {
        System.out.println("Correr");
    }

}
```

Uma chamada a um construtor super() no construtor de uma subclasse resultará na execução do construtor referente da superclasse, baseado nos argumentos passados.



Modificador de Classe *final*

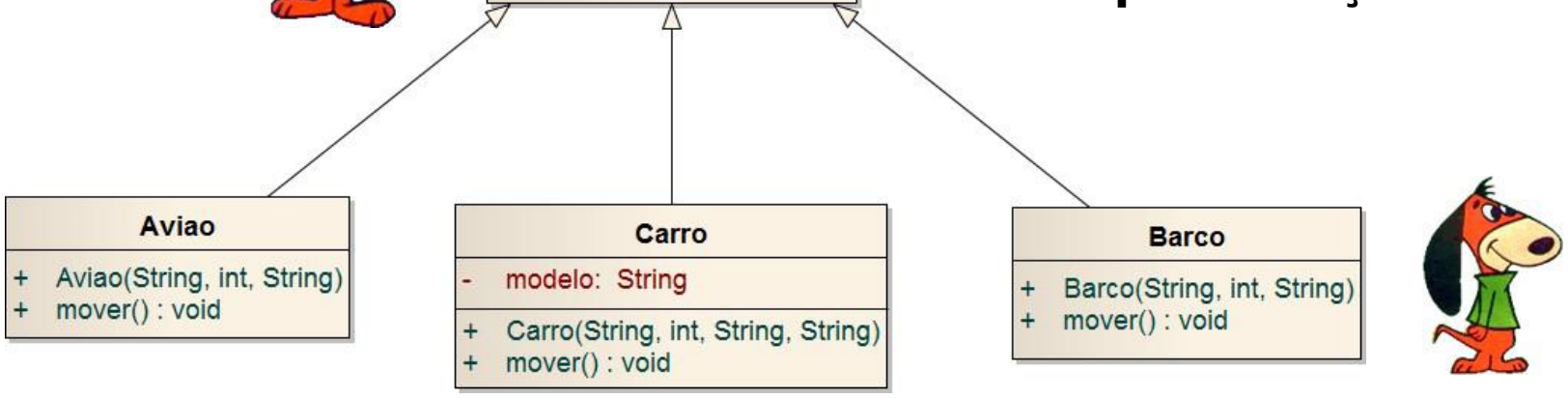
Classes que não podem ter subclasses

<modificador|* final class <nomeClasse| { ... }

Muitas classes na API Java são declaradas **final** para certificar que
seu comportamento não seja herdado e, possivelmente modificado.

Exemplos, são as classes **Integer**, **Double**, **Math** e **String**

Polimorfismo



Generalização

Especialização

Sobreposição de Métodos @Override

```java
public class Veiculo {// nome da classe
    // ... //
    public void mover() {
        System.out.println("Veiculo se movendo");
    }
    // ... //
}
```

```java
public class Barco extends Veiculo {
    // ... //
    @Override
    public void mover() {
        System.out.println("Navegar");
    }
    // ... //
}
```

```java
public class Aviao extends Veiculo {
    // ... //
    @Override
    public void mover() {
        System.out.println("Voar");
    }
    // ... //
}
```

```java
public class Carro extends Veiculo {
    // ... //
    @Override
    public void mover() {
        System.out.println("Correr");
    }
    // ... //
}
```

O tipo de **retorno** do método na **subclasse** deve ser **idêntico** ao do **método sobreposto** na **superclasse**.