Linguagem de Programação II

André Rodrigues da Cruz

PhD e MSc em Engenharia Elétrica
BSc em Matemática Computacional
dacruz@cefetmg.br

Herança, Polimorfismo, Interface

Centro Federal de Educação Tecnológica de Minas Gerais

CEFET-MG

Herança

- Construção (derivação) de novas classes à partir de outras.
- Relacionamento "é um" (subtipo de):
  - Pendrive é um Produto: `class Pendrive extends Produto { ... }`
  - Mamífero é um Animal: `class Mamifero extends Animal { ... }`
  - Gerente é um Empregado: `class Gerente extends Empregado { ... }`
- Classe filha (sub, estendida ou derivada) herda (reusa) membros da classe pai (super ou base).
- Adiciona-se novos membros ou altera-se existentes.
- Subclasse pode herdar ou sobrescrever métodos da superclasse.
- Tipo de herança:
  - Simples (herda única classe).
  - Múltipla (herda várias classes de uma vez):
    - ★ Java não suporta: apenas um pai pode ser herdado por vez.
    - ★ Java permite que uma classe implemente várias interfaces de uma vez.

Diagrama UML de Herança Simples

- Exemplo:

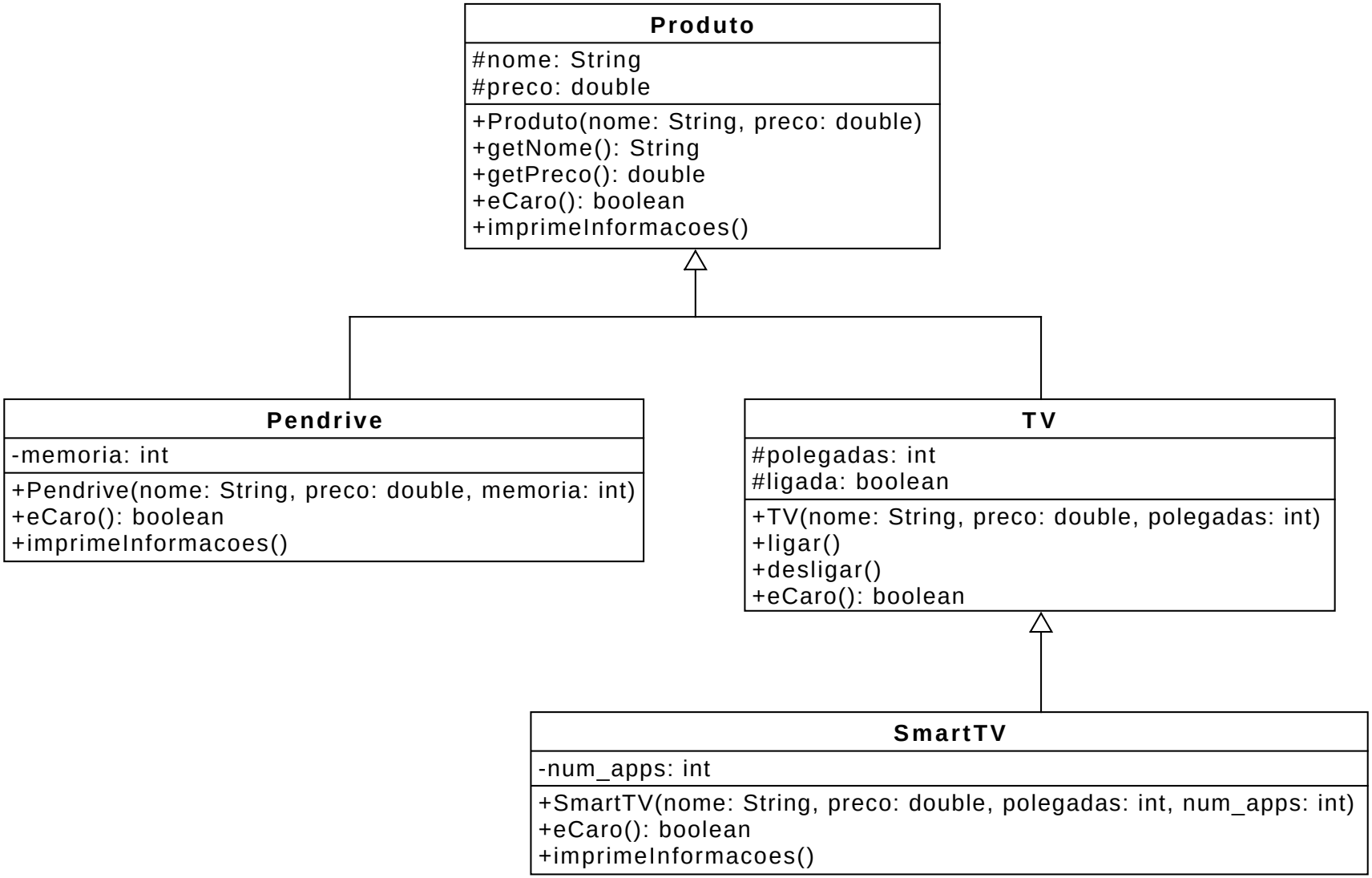
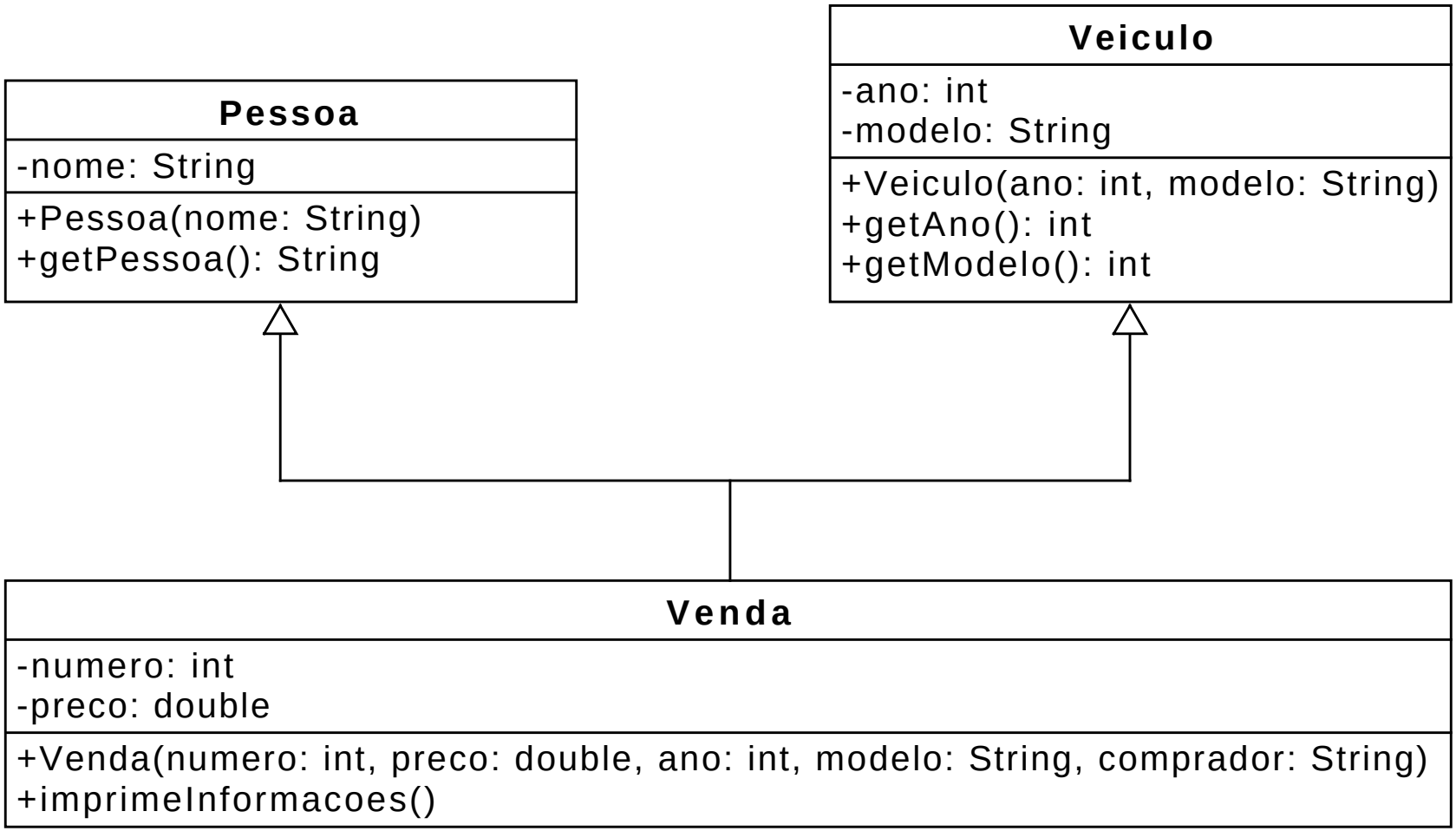


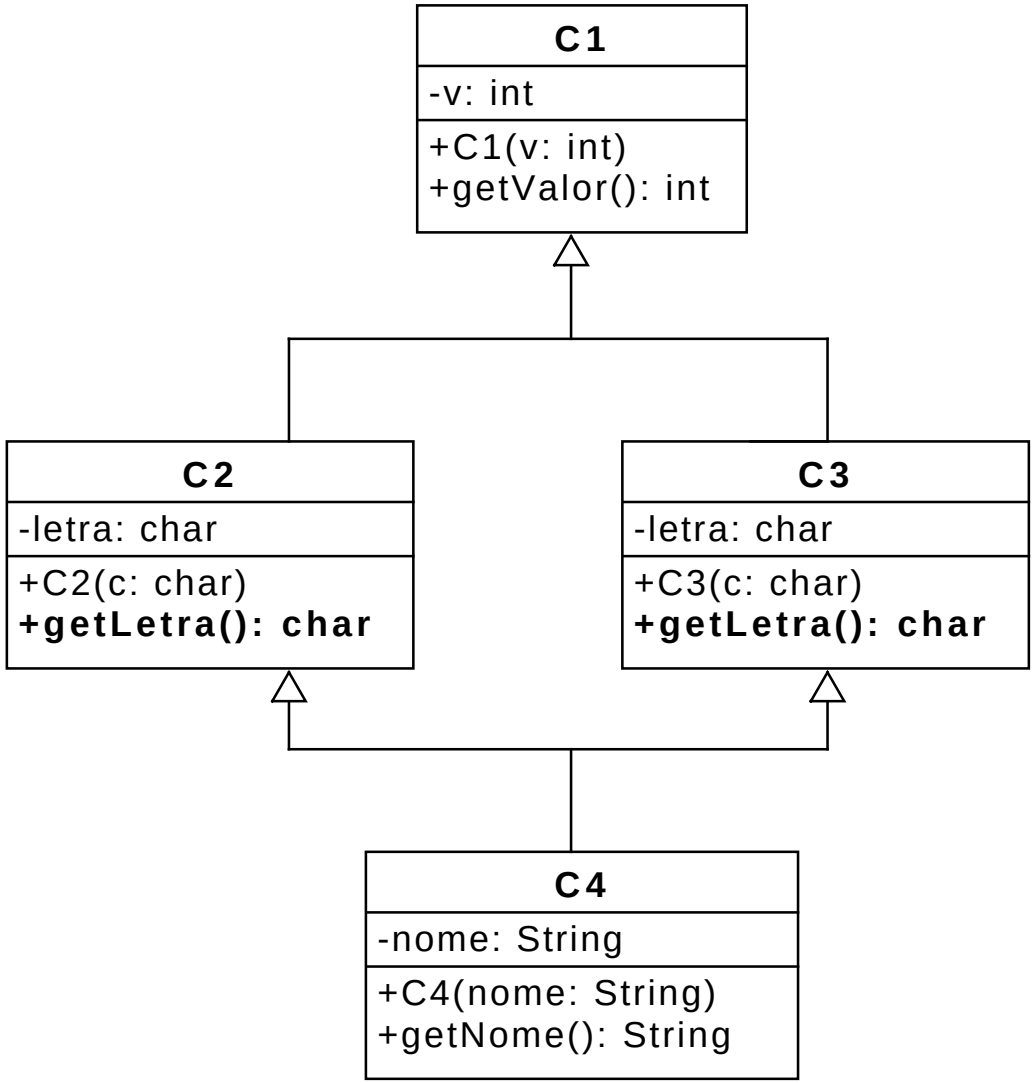
Diagrama UML de Herança Múltipla

- Exemplo:



O Problema do Diamante na Herança Múltipla

- Exemplo:

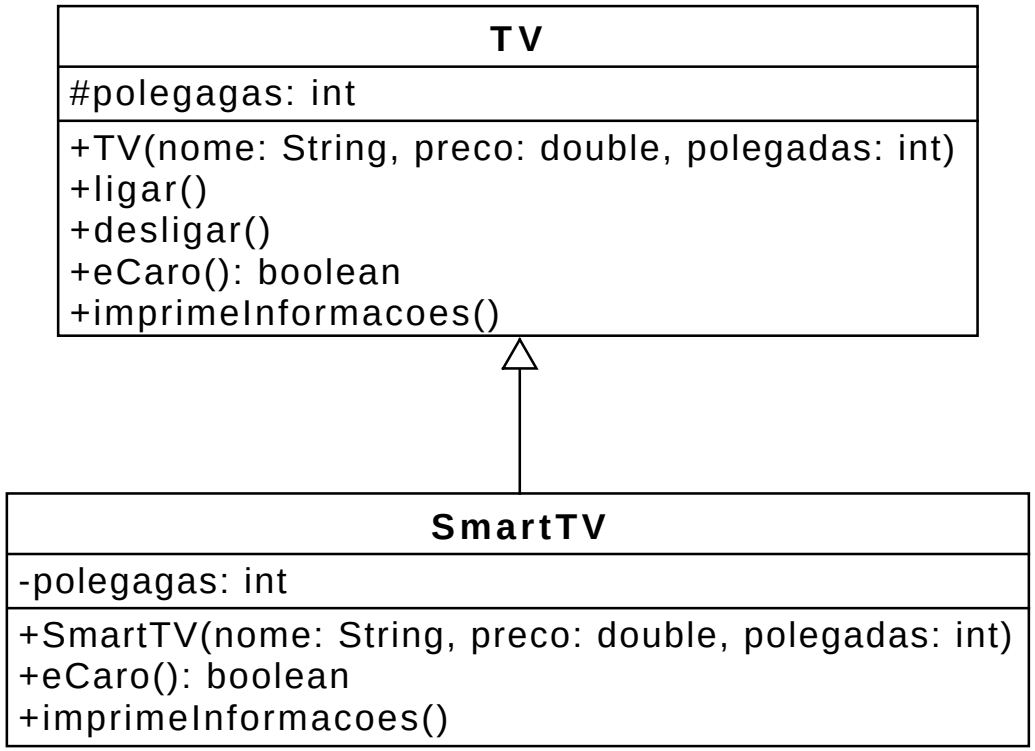


Visibilidade de Membros Herdados

	Modificador			
Acesso na	`private`	`default`	`protected`	`public`
mesma classe	sim	sim	sim	sim
mesmo pacote	não	sim	sim	sim
pacotes diferentes (subclasse)	não	não	sim	sim
pacotes diferentes (subclasse)	não	não	não	sim

Sobrecarga versus Sobrescrita de Métodos

- Sobrecarga:
  - Na mesma classe → Mesmo identificador e assinaturas distintas.
- Sobrescrita:
  - Na classe derivada → Mesmo identificador e assinatura.
  - Toma o lugar do método herdado.
- Exemplo: `eCaro()` e `imprimeInformacoes()` são sobrescritos.



Métodos e Classes `final` e `abstract`

- Não se sobrescreve métodos `final`.
- Não se estende (deriva) classes `final`.
- Classe concreta: Possui todos os métodos implementados.
- Classe abstrata (`abstract`):
  - Possui métodos abstratos: declarados (`abstract`) e não implementados.
  - Força subclasses a implementá-los.
  - Voltado para planejamento de classes para fins gerais.
  - Podem ter métodos não abstratos.
  - Não se instancia objetos de classes abstratas.
- Classe abstrata pura:
  - Possui apenas métodos abstratos (nenhum concreto).
  - Não possui propriedades.

Classe `Object`

- Toda classe no Java herda de alguma classe.
- Se não houver herança explícita, herda-se da classe `Object`.
- Há métodos para serem sobrecarregados de forma a tornarem-se úteis nas particularidades.

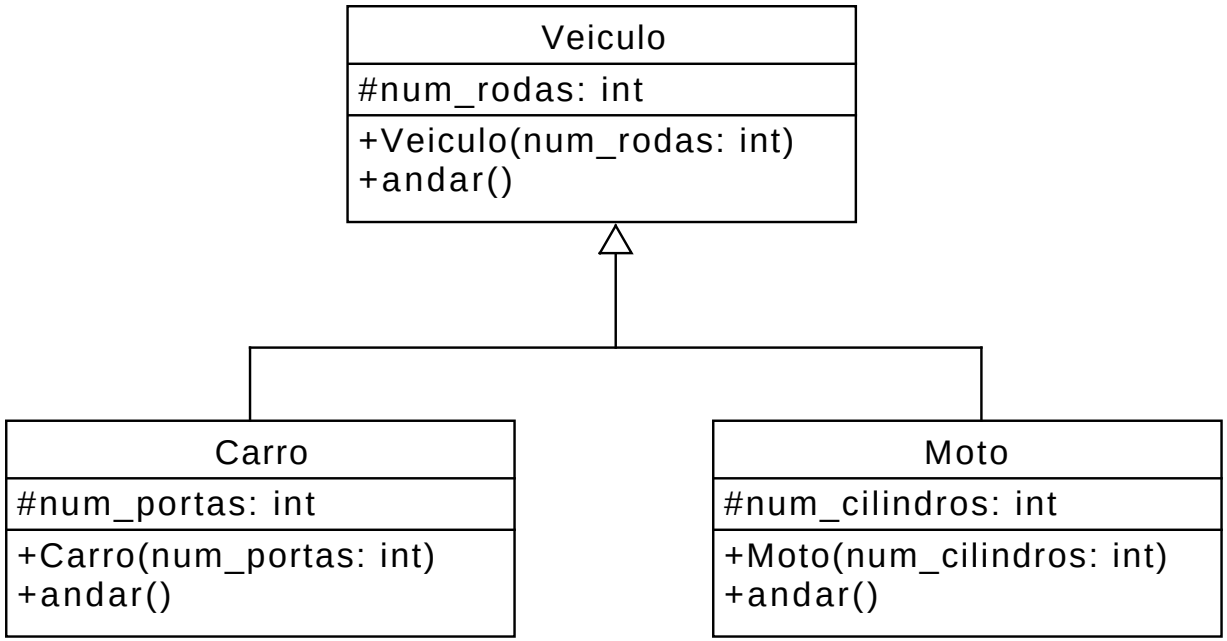
Método	**Descrição**
`String toString()`	Representação em String do objeto.
`boolean equals(Object o)`	Retorna verdadeiro quando considera-se igualdade entre objetos.
`int hashCode()`	Código hash para o conteúdo do objeto.
`Class<?> getClass()`	Descrição da classe do objeto.
`protected Object clone()`	Realiza cópia do objeto.
`protected void finalize()`	Utilizado no garbage collector. Não sobrecarregá-lo.
`wait, notify, notifyAll`	Utilizado em processamento paralelo.

Polimorfismo

- *poli* = muitas, *morphos* = formas.
- *Qualidade de ser capaz de assumir diferentes formas*.
- Polimorfismo na hierarquia de herança.
- Referências de classe pai apontam para instâncias de filhos:
  - Referência da base pode apontar para objeto de classe derivada.
  - Independente do nível de herança (filho, neto, ...).
- Referência de classe abstrata pode apontar para instância de filho.
- *Upcasting*: referência de pai aponta para objeto filho.
- *Downcasting*: referência filho aponta para objeto da classe apontado por uma referência pai.
- Polimorfismo ocorre em tempo de execução.

Polimorfismo

- Exemplo:

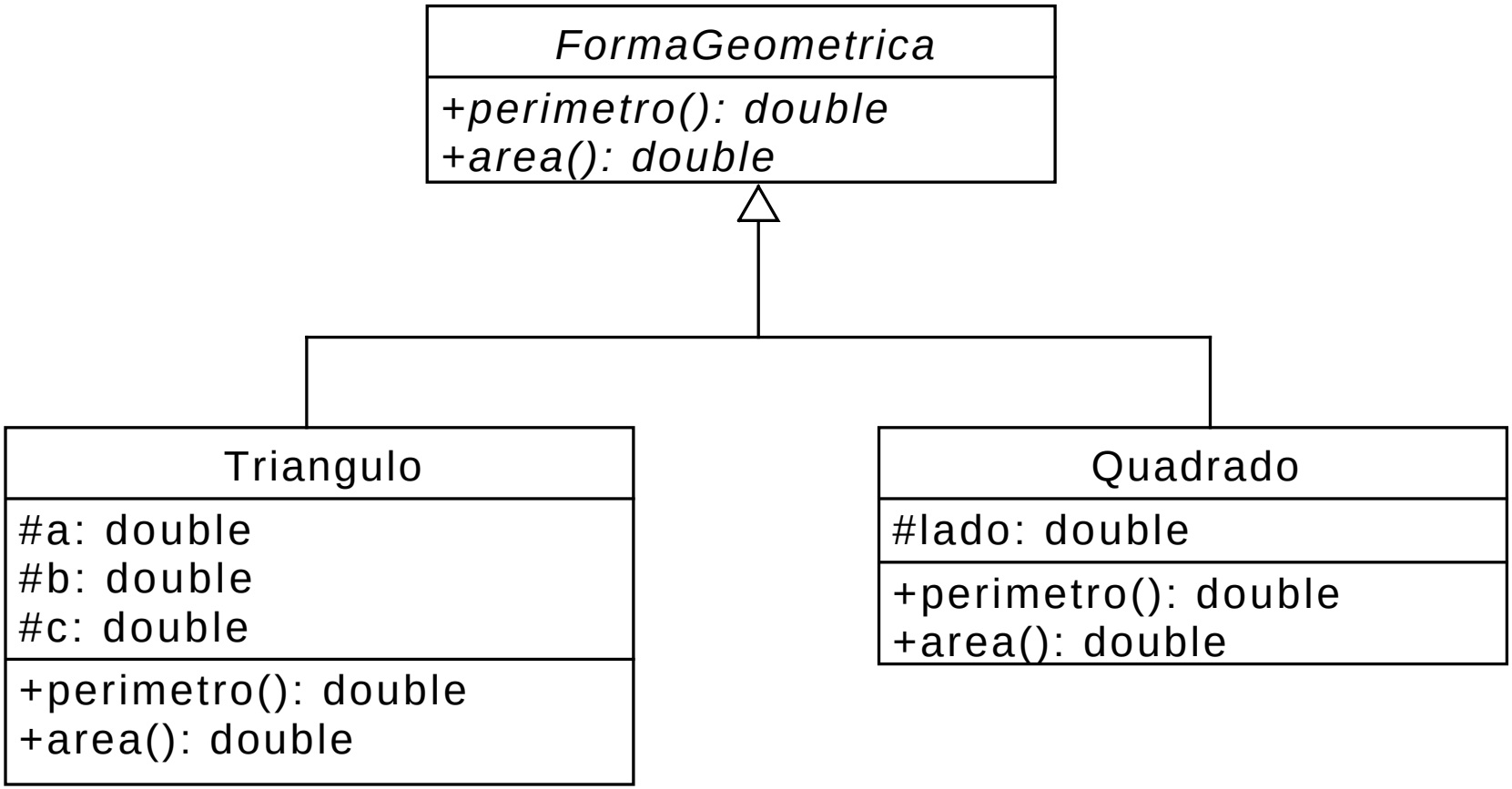


- Upcasting e downcasting:

```
1  Veiculo v = new Carro(5); //upcasting
2  v.andar();
3
4  Carro c = (Carro) v; //downcasting
5  c.andar();
```

Polimorfismo

- Exemplo com classe abstrata base:

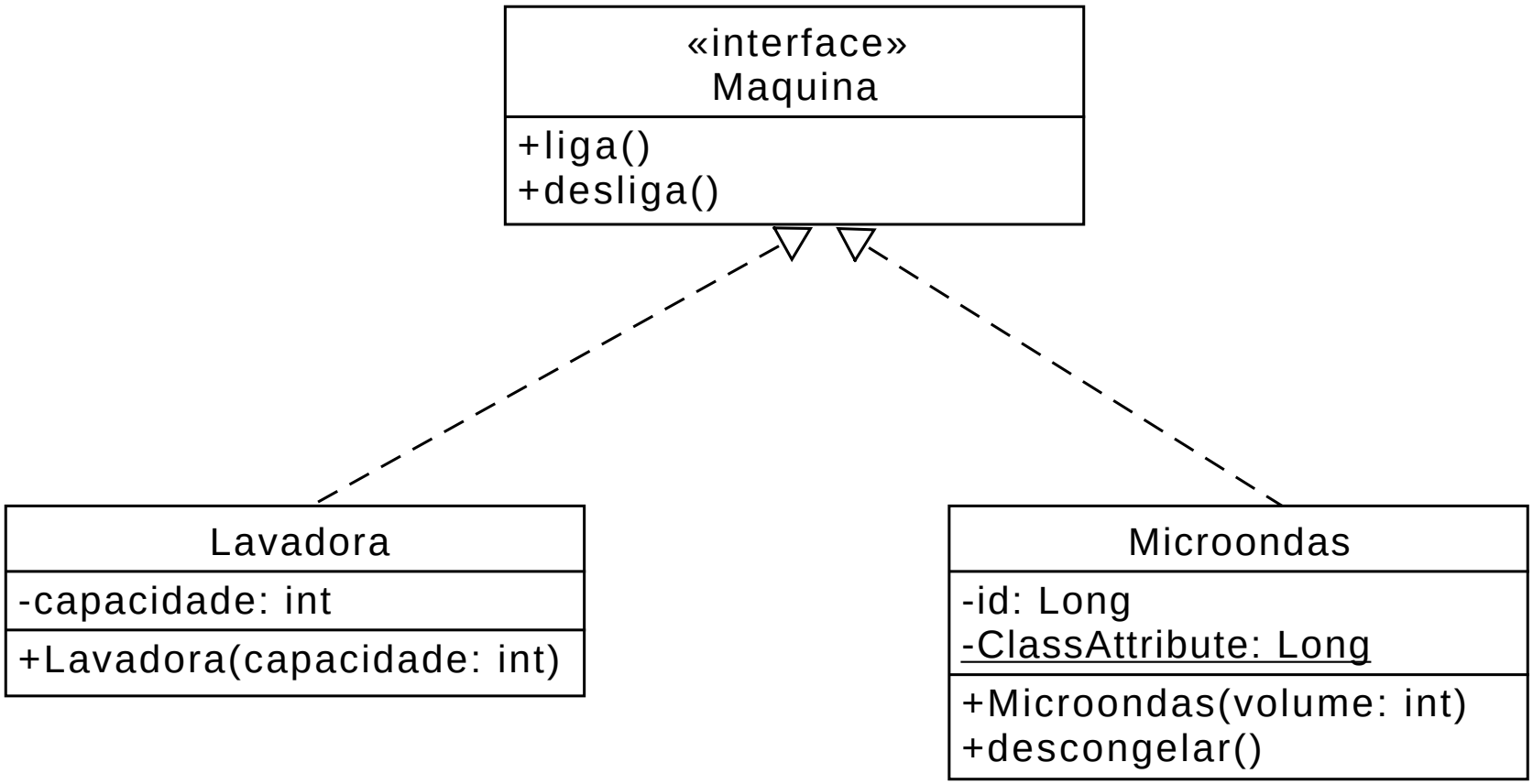


Interface

- Padrões definidos em contratos ou especificações em uma `interface`:
  - Define métodos a serem implementados.
  - Para implementar a usa-se a palavra-chave `implements`.
  - Se não codificar todos os métodos da interface, classe deve ser abstrata.
  - Não instancia objeto, mas é super tipo:
    - ★ Referência pode apontar para objetos de classes que a implementou.
  - **Variáveis** são automaticamente **públicas**, **estáticas** e **finais**.
  - **Métodos** são:
    - ★ Abstratos ou `default` implementados (serão herdados ou sobrescritos).
    - ★ Públicos (automaticamente) ou privados, podendo ser estáticos.
- Implementação de múltiplas interfaces:
  - Uma classe pode ser herdada por vez (herança única).
  - Mas, uma classe pode implementar várias interfaces ao mesmo tempo.
- Interfaces `Comparable` e `Comparator`: usados para comparar objetos.
- Interface funcional: possui um método abstrato.

Interface

- Exemplo:



Interface

- Um interface pode estender outra: adição de novos membros.

```
1 public interface Closeable {
2   void close();
3 }
```

```
1 public interface Channel extends Closeable {
2   boolean isOpen();
3 }
```

- Implementação de múltiplas classes:

```
1 public class FileSequence implements IntSequence, Closeable {
2   ...
3 }
```

Interface

- Solução de conflito em implementação de múltiplas interfaces:

```
1 public interface Person {
2   String getName();
3   default int getId() { return 0; }
4 }
```

```
1 public interface Identified {
2   default int getId() { return Math.abs(hashCode()); }
3 }
```

```
1 public class Employee implements Person, Identified {
2   public int getId() { return Identified.super.getId(); }
3   ...
4 }
```

Verificação de Tipo de Instância

- `obj instanceof C` retorna `true` se instância é da classe.
- Serve para referências de classe ou interface: lembre da herança.

```
1  public interface IntSequenciaInteira {
2    boolean possuiProximo();
3    int proximo();
4  }
```

```
1  class SequenciaQuadrada implements IntSequenciaInteira {
2    private int i = 0;
3
4    public boolean possuiProximo() { return true; }
5
6    public int proximo() {
7      i++;
8      return i*i;
9    }
10 }
```

```
1  public static void main(String[] args) {
2    IntSequenciaInteira s;
3
4    s = new SequenciaQuadrada();
5
6    if(s instanceof SequenciaQuadrada) {
7      SequenciaQuadrada sq = (SequenciaQuadrada) s;
8      System.out.println(sq.proximo());
9    }
10 }
```

Interface `Comparable`

```
1 public interface Comparable<T> {
2   int compareTo(T other);
3 }
```

- Quando se deseja classificar ordem de objetos.
- `x.CompareTo(y)` deve retornar inteiro:
  - 0, para igualdade.
  - positivo, para `x > y`
  - negativo, para `x < y`

```
1 public class Empregado implements Comparable<Empregado> {
2   ...
3   public int compareTo(Employee other) {
4     return Double.compare(salary, other.salary);
5   }
6 }
```

- Array com objetos de `Empregado` ordenável via `Arrays.sort`.
- `String` implementa `Comparable<String>`.

Interface `Comparator`

```
1 public interface Comparator<T> {
2   int compare(T first, T second);
3 }
```

- Segunda versão que pode ser usada em `Arrays.sort`.
- Dado 2 objetos `T`, retorna um inteiro (mesmo significado de `compareTo`).
- Exemplo: Ordenar `String` pela quantidade de caracteres:

```
1 class ComparaTamanho implements Comparator<String> {
2   public int compare(String first, String second) { return first.length()
3 }
```

```
1  public class OrdenaString {
2    public static void main(String[] args) {
3      String[] nomes = {"Joaquim", "Anderson", "Ana", "Joana"};
4
5      Arrays.sort(nomes, new ComparaTamanho());
6
7      for(String n: nomes)
8        System.out.println(n);
9    }
10 }
```