Instituto de computação

Herança e Polimorfismo

Leonardo Gresta Paulino Murta

leomurta@ic.uff.br

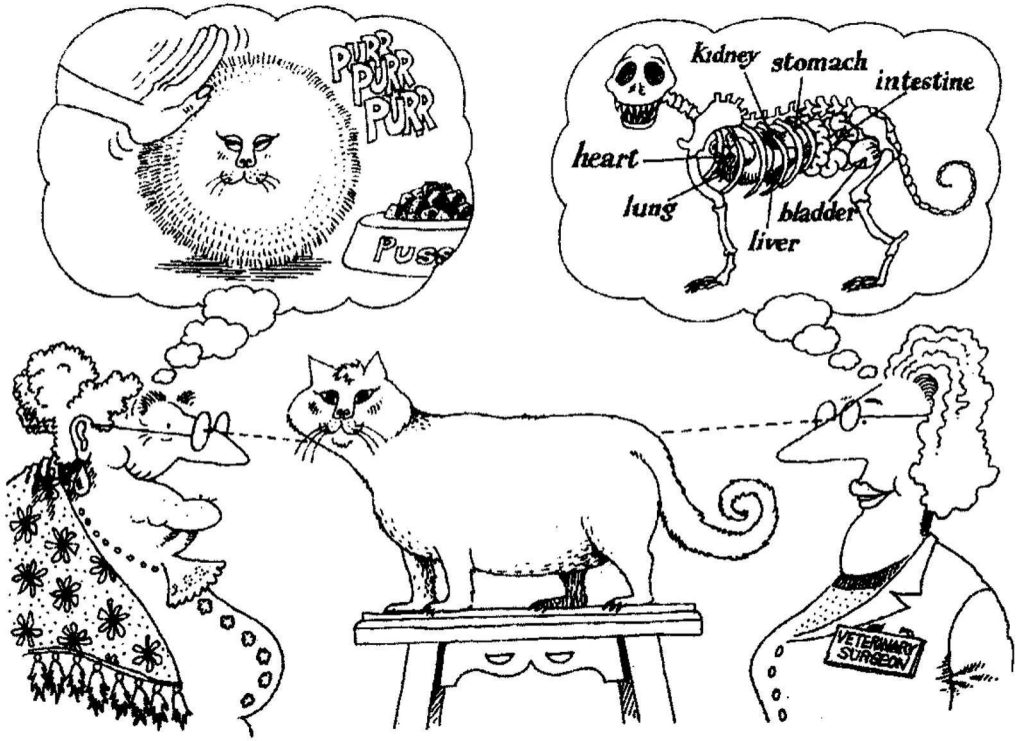
Aula de hoje

- Princípios do paradigma OO
- Herança
- Polimorfismo
- Interfaces
- Classes Abstratas

Princípios do Paradigma OO

Abstração

- A representação computacional do objeto real deve se concentrar nas características que são relevantes para o problema



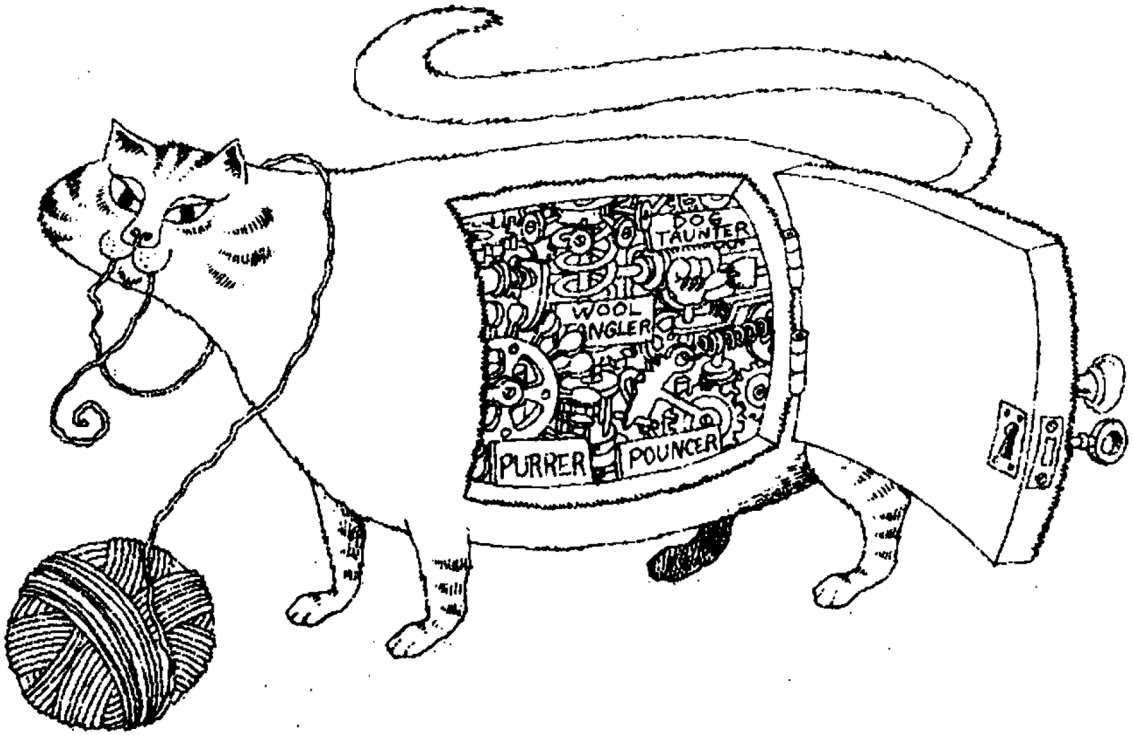
Fonte: livro “Object-Oriented Analysis and Design with Applications”

Abstração

- São criados somente os atributos e métodos necessários para o problema em mãos
- Quais seriam os atributos e métodos para o objeto Carro em cada uma das situações seguintes?
  - Sistema de uma locadora de carros
  - Sistema de uma revendedora de carros
  - Sistema de uma oficina mecânica
  - Sistema do DETRAN

Encapsulamento

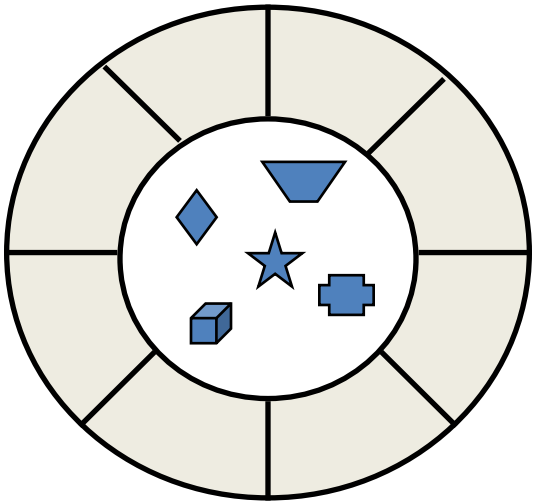
- O objeto deve esconder seus dados e os detalhes de sua implementação



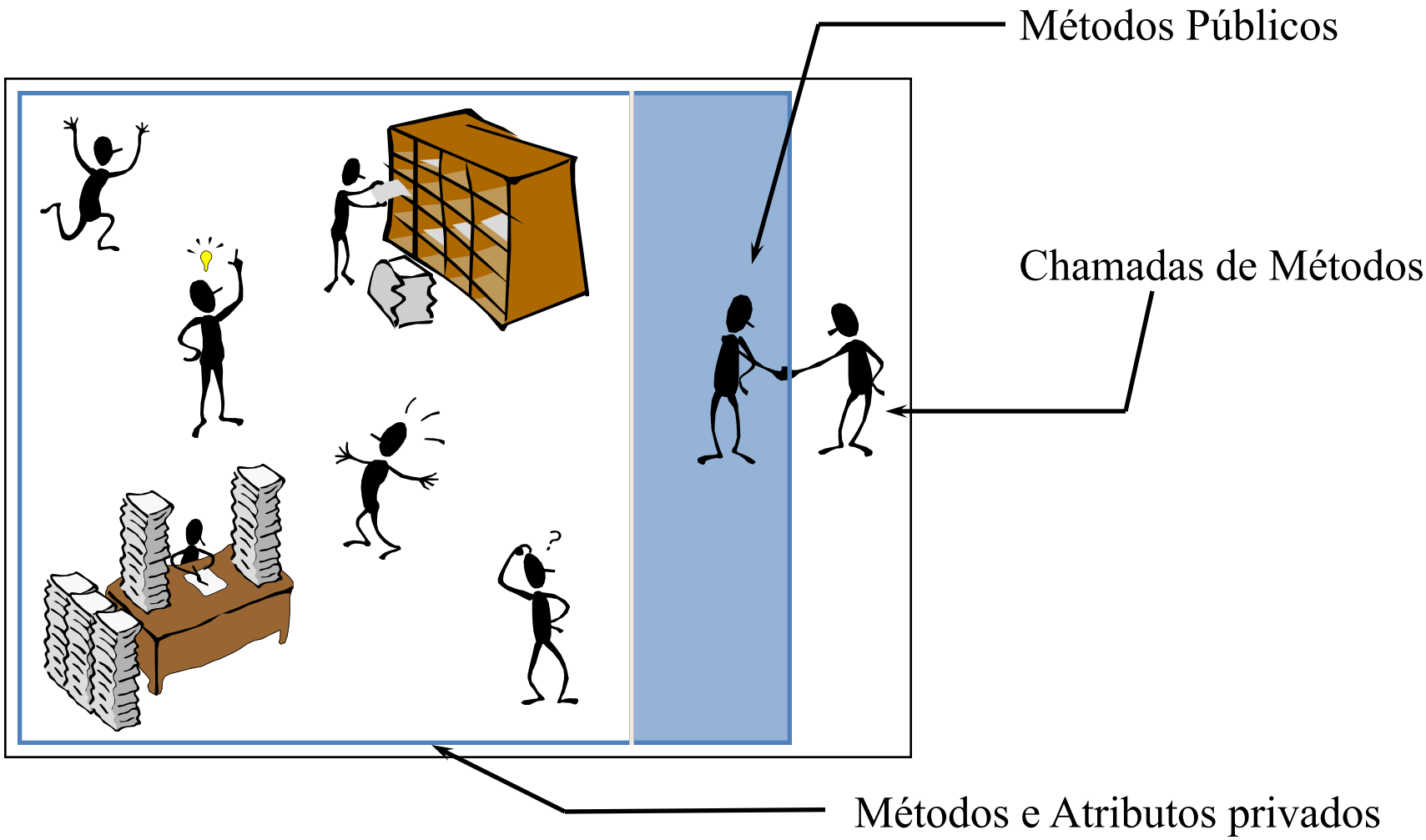
Fonte: livro “Object-Oriented Analysis and Design with Applications”

Encapsulamento

- Atributos e Métodos
  - Os métodos formam uma “cerca” em torno dos atributos
  - Os atributos não devem ser manipulados diretamente
  - Os atributos somente devem ser alterados ou consultados através dos métodos do objeto



Encapsulamento



Encapsulamento

- Em uma classe Quadrado, quais métodos devem ser públicos e quais devem ser privados?
  - Transladar
  - Ampliar
  - Mover ponto
  - Girar
  - Adicionar ponto
  - Calcular área

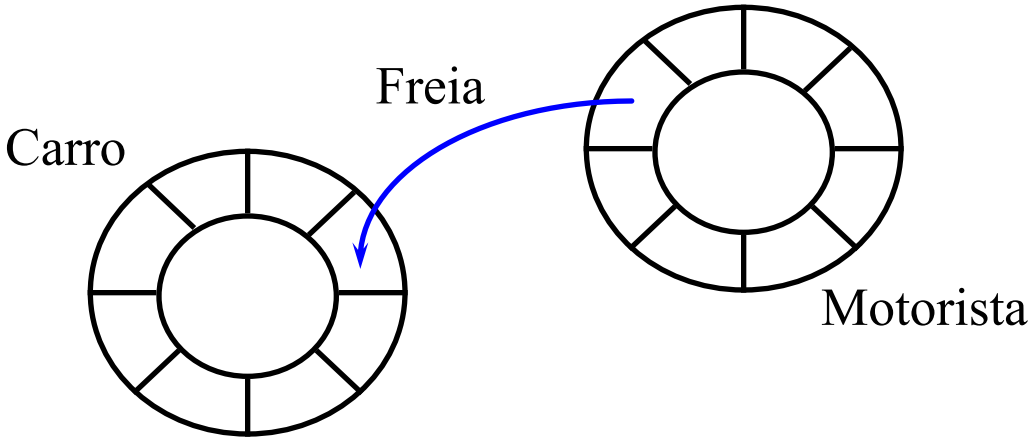
Modularidade

- O sistema deve ser composto de objetos **altamente coesos** e **fracamente acoplados**

Fonte: livro “Object-Oriented Analysis and Design with Applications”

Modularidade

- Um programa OO é um conjunto de objetos que colaboram entre si para a solução de um problema
- Objetos colaboram através de chamadas de métodos uns dos outros

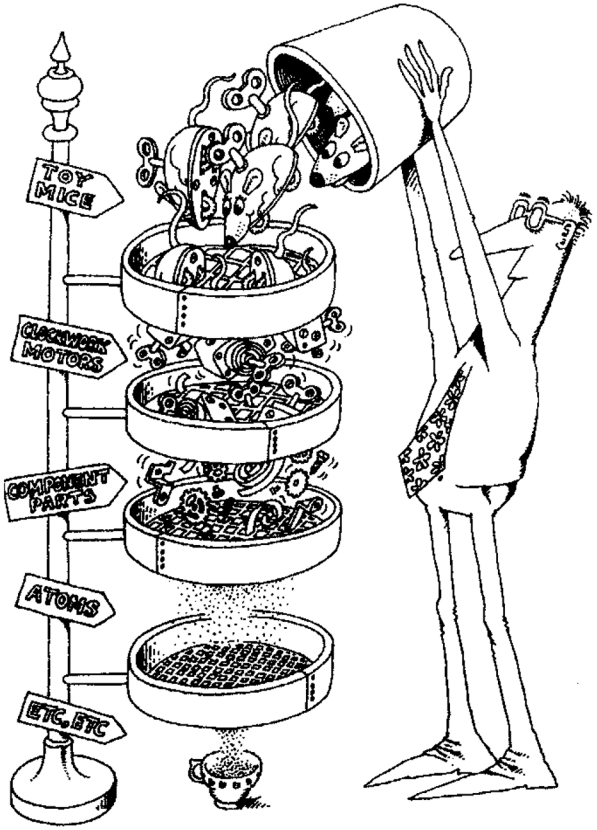


Modularidade

- Em um sistema acadêmico, há conceitos aluno, professor, disciplina, turma e inscrição. Onde colocar cada um dos métodos a seguir:
  - Exibição do histórico do aluno
  - Cálculo da média do aluno em uma turma
  - Obtenção do horário de uma aula
  - Descrição da ementa de uma disciplina
  - Cálculo do CR de um aluno

Hierarquia

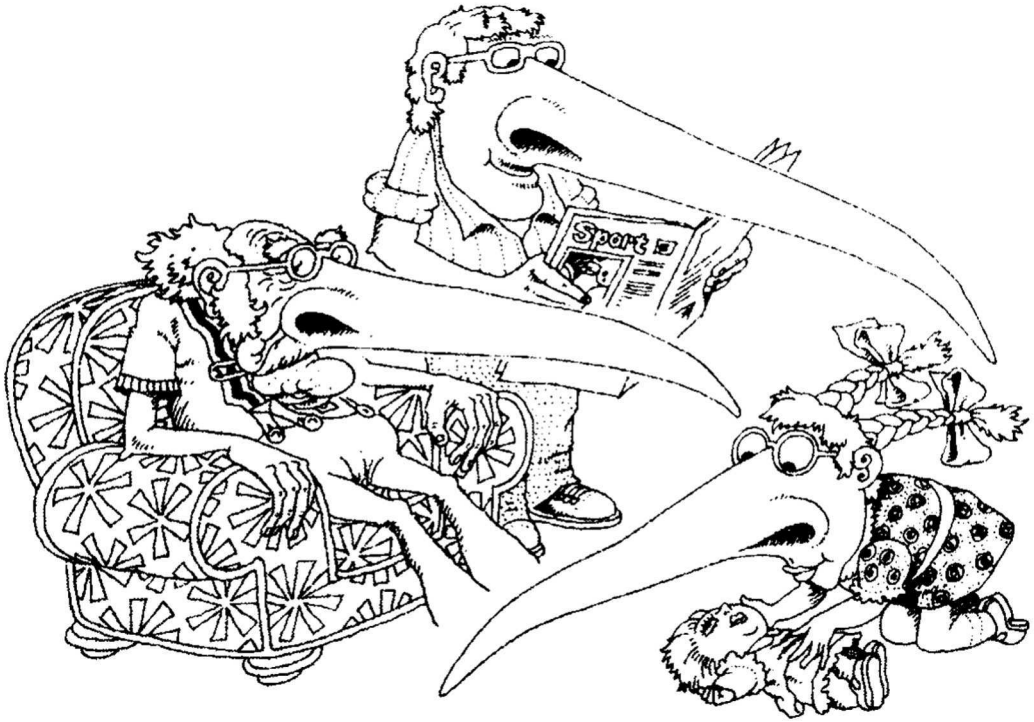
- Os objetos devem ser organizados no sistema de forma hierárquica



Fonte: livro “Object-Oriented Analysis and Design with Applications”

Hierarquia

- Objetos herdam atributos e métodos dos seus ancestrais na hierarquia

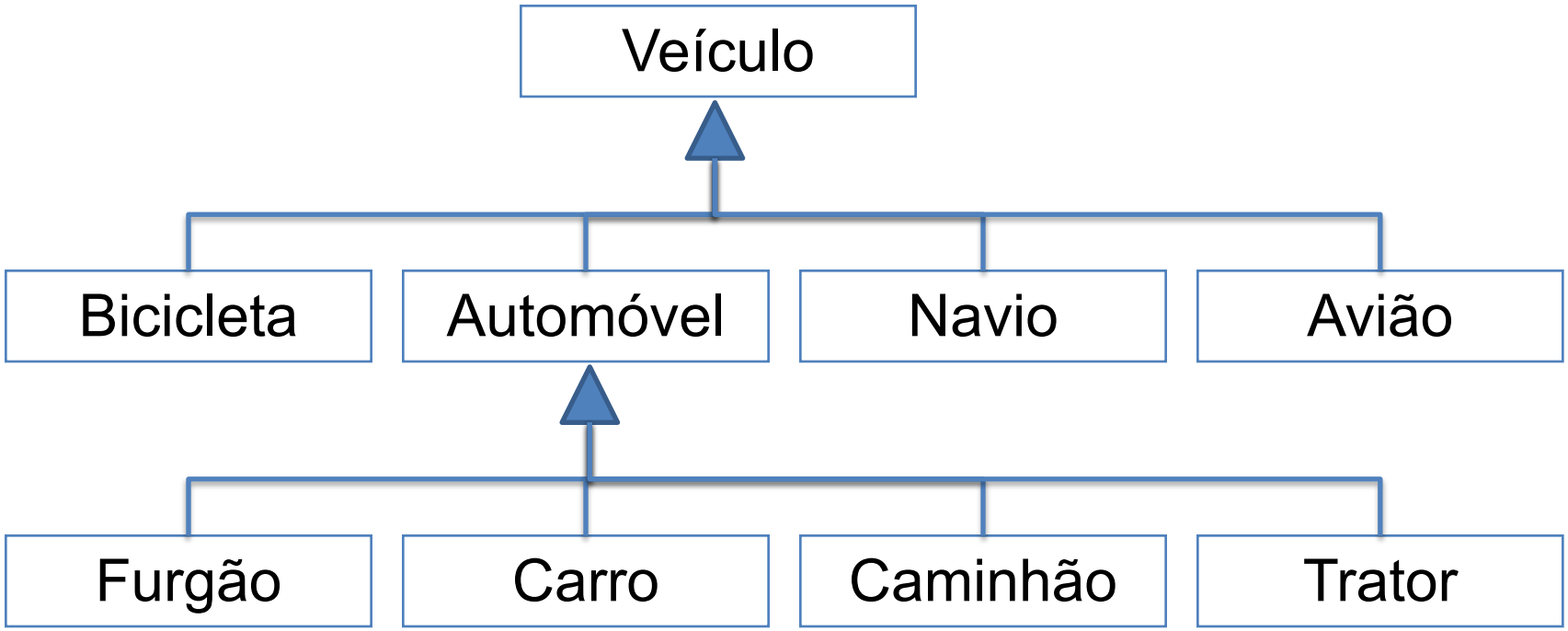


Fonte: livro “Object-Oriented Analysis and Design with Applications”

Herança

- Para viabilizar a hierarquia entre objetos, as classes são organizadas em estruturas hierárquicas
  - A classe que forneceu os elementos herdados é chamada de **superclasse**
  - A classe herdeira é chamada de **subclasse**
  - A subclasse pode **herdar os métodos e atributos** de suas superclasses
  - A subclasse pode **definir novos atributos e métodos** específicos

Exemplo de herança



Teste da Leitura: "subclasse é um superclasse"
Ex.: Carro é um Automóvel; Trator é um Veículo; ...

Exemplo de herança (relembrando a classe Carro)

```
public class Carro {
  private int velocidade;

  public Carro(int velocidadeInicial) {
    velocidade = velocidadeInicial;
  }

  public void acelera() {
    velocidade++;
  }

  public void freia() {
    velocidade--;
  }
}
```

Exemplo de herança (criando um carro inteligente)

- Declaração:

```
public class CarroInteligente extends Carro {
  public CarroInteligente(int velocidadeInicial) {
    super(velocidadeInicial);
  }

  public void estaciona() {
    // código mágico para estacionar sozinho
  }
}
```

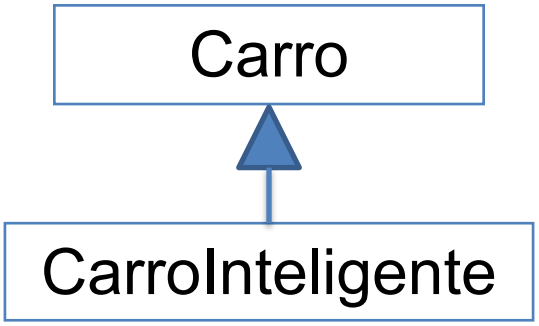
- Uso:

```
CarroInteligente tiguan = new CarroInteligente(10);
for (int i = 10; i > 0; i--) {
  tiguan.freia();
}
tiguan.estaciona();
```

De onde veio isso?

Compatibilidade de tipos

- Qualquer **subclasse** é compatível com a sua **superclasse**
  - Contudo, a reciproca não é verdadeira



✓
```
Carro c = new CarroInteligente(20);
c.acelera();
c.freia();
```

✗
```
CarroInteligente c = new Carro(20);
c.acelera();
c.freia();
c.estaciona();
```

Polimorfismo

- Uma subclasse pode redefinir (sobrescrever) um método herdado
  - Este mecanismo é chamado de **polimorfismo**
  - O polimorfismo se realiza através da recodificação de um ou mais métodos herdados por uma subclasse
  - Em tempo de execução, o Java saberá qual implementação deve ser usada

Exemplo de polimorfismo (criando um carro de corrida)

- Declaração:

```
public class CarroCorrida extends Carro {
  public CarroCorrida(int velocidadeInicial) {
    super(velocidadeInicial);
  }

  public void acelera() {
    velocidade+=5;
  }
}
```

- Uso:

```
CarroCorrida f1 = new CarroCorrida(10);
f1.acelera();
```

Qual a velocidade agora?

Herança em Java

- Uma classe só pode herdar de uma outra classe (herança simples)
- Caso não seja declarada herança, a classe herda da classe *Object*
  - Ela define o método *toString()*, que retorna a representação em *String* do objeto
  - Qualquer subclasse pode sobrescrever o método *toString()* para retornar o que ela deseja.
- Veja os demais métodos da classe *Object* em https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html

Exercício de herança

- Em um sistema de loja, há 3 tipos de usuário: gerente, funcionário e cliente. Todo usuário tem nome e senha. O cliente possui, além do nome e senha, outros dados cadastrais. O funcionário possui métodos relacionados a venda de produtos. O gerente pode fazer tudo que o funcionário pode e também fechamento do caixa. Como é a hierarquia de herança desse sistema no que se refere a controle de usuários?

Interfaces

- Tipo especial de classe, que não tem implementação
  - Uma interface define um protocolo
  - Classes podem implementar uma ou mais interfaces

- Uma **interface é um contrato** assinado por uma classe
  - A interface define as responsabilidades da classe
  - As responsabilidades são mapeadas em métodos
  - A classe que implementa a interface implementa os métodos
  - A interface contém somente assinatura de métodos e constantes

Interfaces

- A definição de uma interface é similar a de uma classe
  - Utilizamos a palavra reservada *interface*
  - A palavra reservada deve ser seguida do nome da interface
  - Uma interface pode herdar de outras interfaces (*extends*)
  - A interface possui apenas métodos sem implementação (abstratos) e constantes

```
public interface Taxavel
{
        int ANO_INICIO = 1996;
        double calculaTaxa ();
}
```

Fica implícito que é public, static e final

Exemplo

```
public class CarroImportado extends Carro implements Taxavel
{
        <atributos da CarroImportado>
        <métodos da CarroImportado>
        <métodos implementados da interface Taxavel>
}
```

Programação orientada a interfaces


```
public interface Stack {
   public Object pop();
   public void push(Object o);
   public int size();
}
```

Programação orientada a interfaces

```
adicionaPedido(Stack s) {
  Pedido p = new Pedido(...);
  s.push(p);
}
```

Desconhece a implementação

```
public interface Stack {
  public Object pop();
  public void push(Object o);
  public int size();
}
```

Programação orientada a interfaces


```
public interface Stack {
   public Object pop();
   public void push(Object o);
   public int size();
}
```

Desconhece o uso

```
public class MyStack implements Stack {
   public void push(Object o) {
   ...
```

Classes Abstratas

- Se uma classe possui algum método sem implementação (abstrato), o modificador ***abstract*** deve preceder sua declaração

```
abstract class Carro
{
    <atributos da classe Carro>
    <métodos comuns da classe Carro>
    <métodos abstratos da classe Carro>

}
```

Classes Abstratas

- Não podem ser usadas para instanciar objetos
- Devem ser vistas como *templates* para subclasses que irão dar uma implementação aos métodos abstratos
- Exemplo:
  - Classe abstrata Pagamento tem um método abstrato paga(...)
  - Classes concretas PagamentoVisa, PagamentoCheque, PagamentoDinheiro dão comportamento ao método paga(...)

Classes Abstratas

```
public abstract class Pagamento {
   ...
   public abstract void paga(...);
}
```

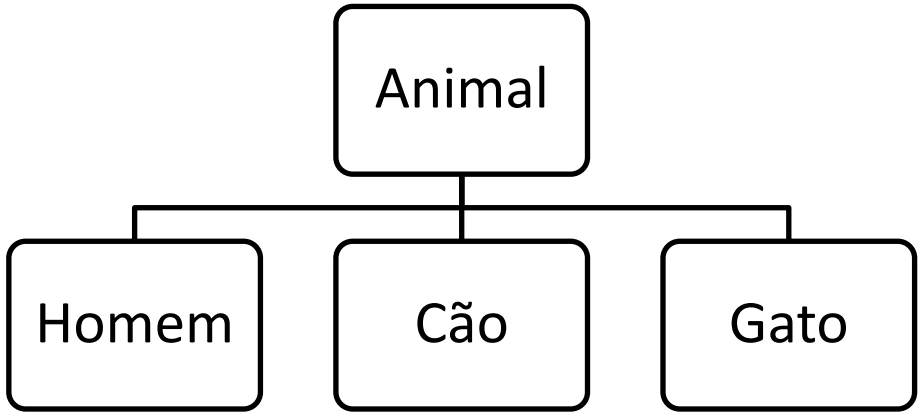
```
public class PagamentoVisa extends Pagamento {
   ...
   public void paga(...) {
      // código de pagamento com cartão Visa
   }
}
```

Exercício

- Refazer o exercício da Agenda, que lista os contatos ordenados por nome e por idade
  - Fazer uma classe abstrata Ordenador com a lógica de ordenação, porém chamando um método abstrato compara(Pessoa, Pessoa)
  - Criar duas classes extras, OrdenadorNome e OrdenadorIdade, que herdam de Ordenador e implementam o método abstrato compara(Pessoa, Pessoa)
  - Instanciar o ordenador apropriado para listar os contatos por nome e por idade

Exercício

- Faça uma classe Animal com um método abstrato “fala”
- Faça as classes Homem, Cão e Gato, herdando de animal, redefinindo o método “fala” para retornar “Oi”, “Au au” e “Miau”, respectivamente
- Crie um vetor de 10 Animais e instancie Homens, Cães e Gatos nesse vetor
- Faça um loop por todos os animais do vetor, pedindo para eles falarem



Instituto de Computação

Herança e Polimorfismo

Leonardo Gresta Paulino Murta

leomurta@ic.uff.br