PROGRAMAÇÃO ORIENTADA A OBJETOS

Herança e Polimorfismo

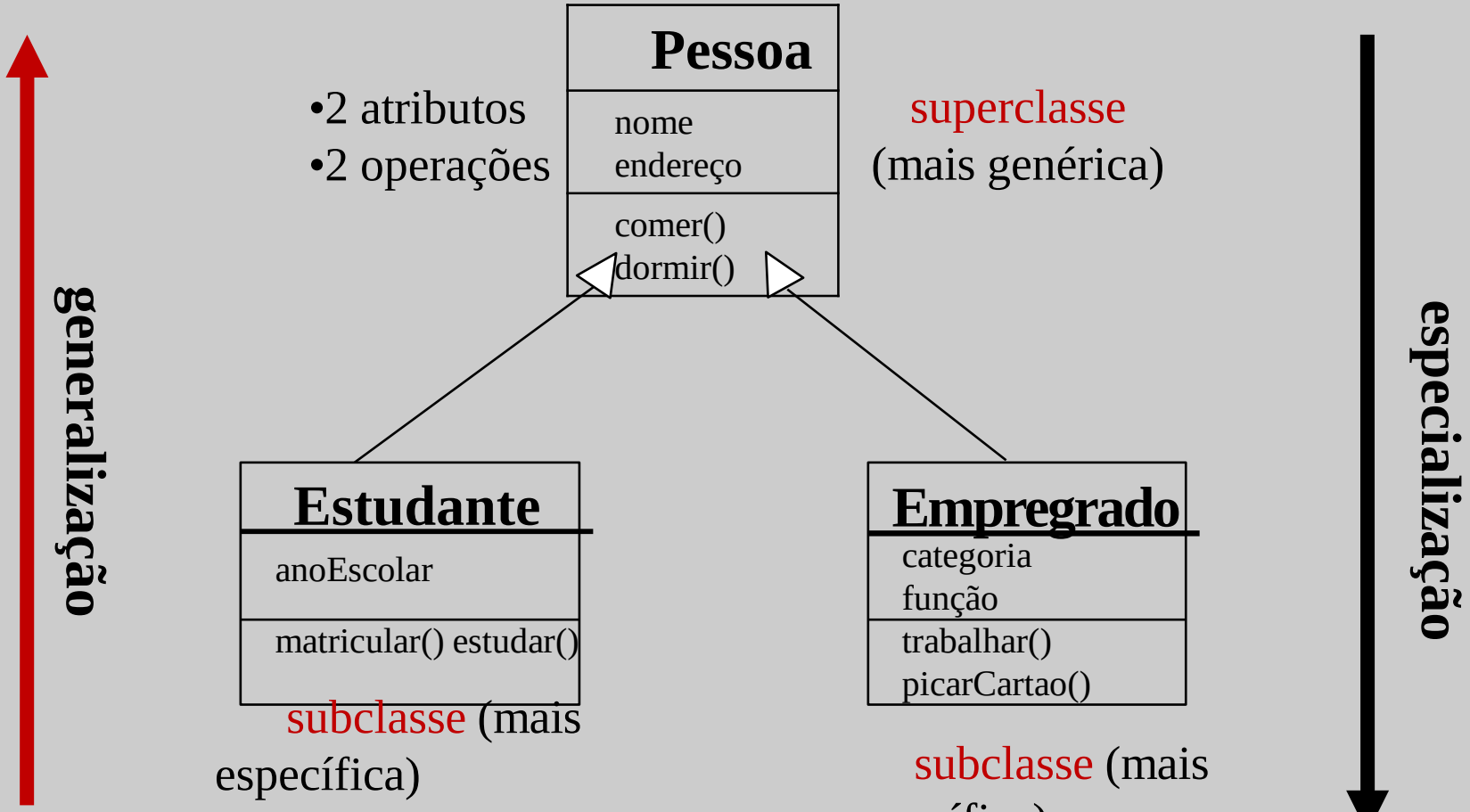
ROTEIRO

- **Herança**
- **Polimorfismo**
- **Classes e Métodos Final**

Herança

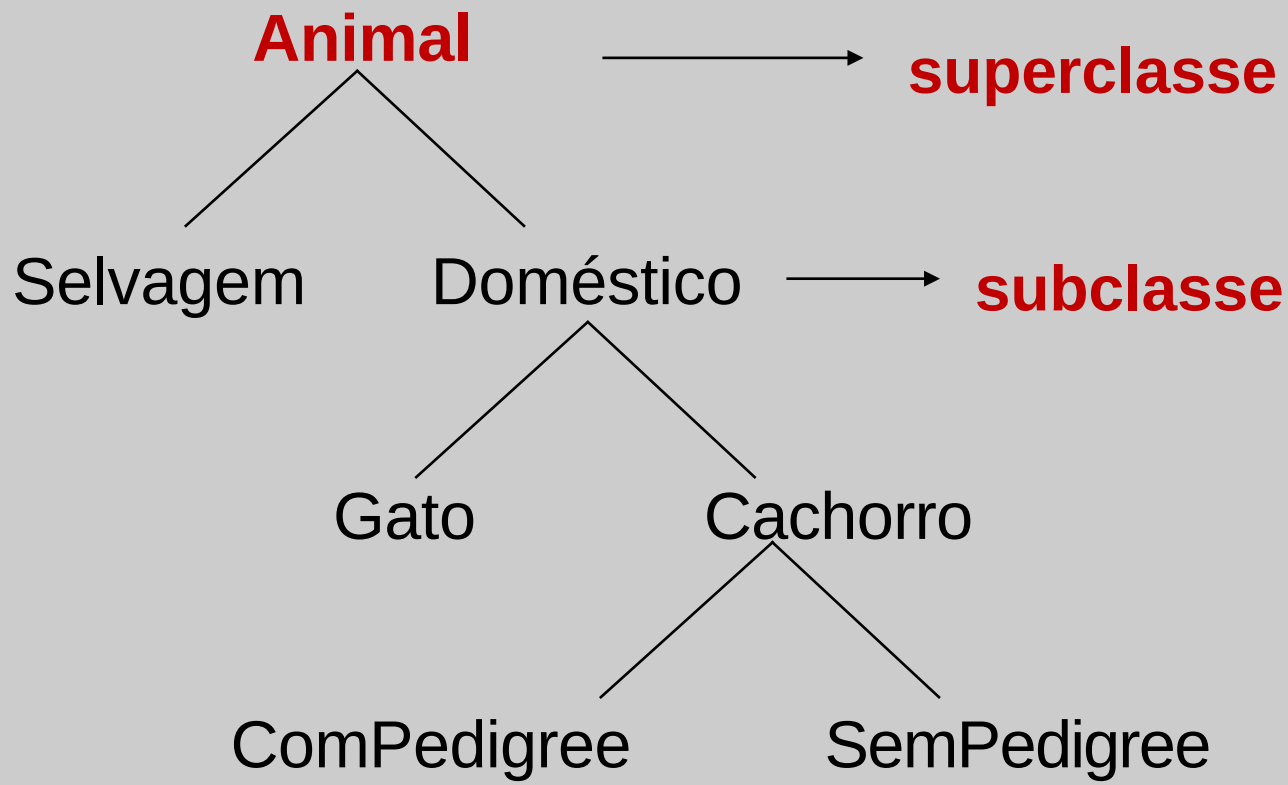
- No mundo real, podemos herdar algumas características de nossos familiares
  - Atributos: cor da pele, cor dos olhos, etc.
  - E até mesmo comportamentos
- Também em POO as classes podem herdar
  - Atributos (propriedades)
  - Métodos (comportamento)
  - Este processo de herdar características denomina-se herança

Herança


Pessoa
nome
endereço
comer()
dormir()
•2 atributos
•2 operações
superclasse
(mais genérica)
generalização
especialização
Estudante
anoEscolar
matricular() estudar()
subclasse (mais específica)
•3 atributos
•4 operações
Empregrado
categoria
função
trabalhar()
picarCartao()
subclasse (mais específica)
•4 atributos
•4 operações

Herança

- **Outro aspecto interessante é que a herança pode ser repetida em cascata, como mostra o exemplo a seguir:**

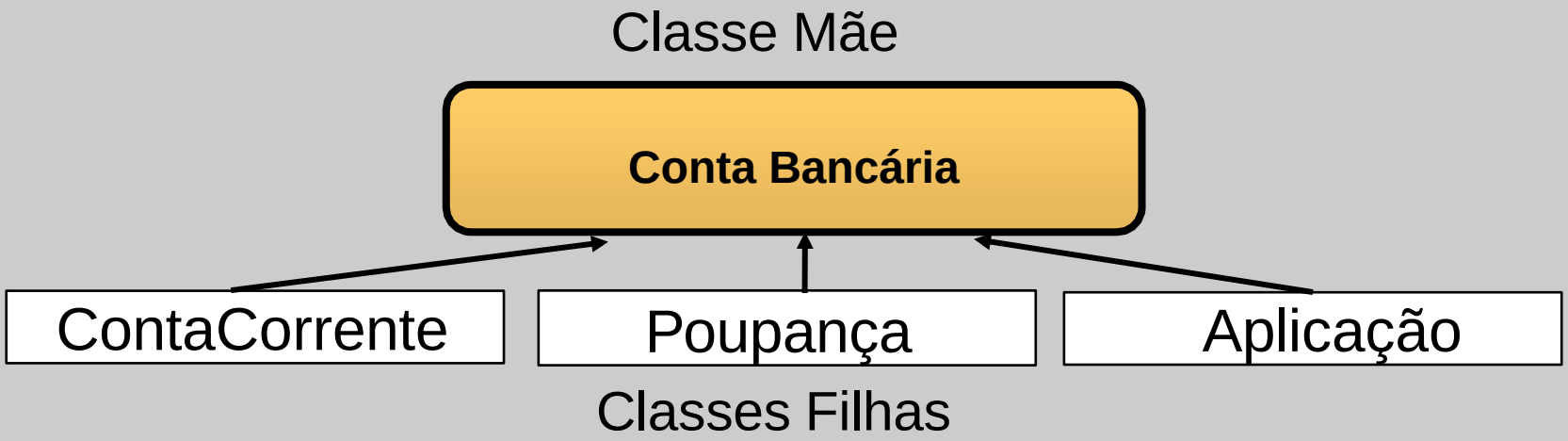


Herança

- Em Java a herança usamos o conceito de herança por meio de classes mãe e classes filha
  - Classe mãe = super classe = classe base
  - Classe filha = subclasse = classe derivada
- Classe Mãe
  - É mais geral e de onde outras classes herdam métodos e atributos (membros)
- Classe Filha
  - É a mais especializada e que herda os membros da classe mãe

Herança

- **Exemplo de Herança em POO**



Herança

- **Em Java, utilizamos a palavra *extends* para lidar com a aplicação do conceito de herança**

```
public class Animal {

}

public class Cachorro extends Animal {

}
```

Herança

- **Vantagens da Herança**
  - **Permite o reuso de software**
  - **Sem a necessidade de escrever o mesmo código novamente**
  - **Pode especializar soluções gerais já existentes**
- **Todo objeto da subclasse é um objeto da superclasse (classe mãe)**
  - ***O contrário não é válido***

Herança

- A classe filha herda todos os membros da classe mãe
  - **Membros privados:** são ocultos na classe filha
  - **Membros públicos:** são acessíveis pela classe filha
  - **Membros protegidos:** são acessíveis na subclasse (e outras do mesmo pacote)
- Construtores não são membros da classe
  - Mas as classes filhas podem chamar um construtor da classe mãe

Herança

- Podemos declarar um campo na classe filha com o mesmo nome da classe mãe, mesmo que os tipos sejam diferentes
- Podemos sobrescrever um método da classe mãe, declarando um método com a mesma assinatura, o que chamamos de **polimorfismo**
- Também podemos declarar novos métodos e campos na classe filha, o que denomina-se de **especialização**

- **Exemplo**
  - **Imagine que temos que modelar uma escola, na figura de professores, alunos e funcionários. Considere que todos são pessoas. Neste caso, Pessoa seria nossa classe mãe e Professores, Funcionários e Alunos seriam nossas classes filhas**

```
public class Pessoa {
    public String nome;
    public String cpf;
    public Date data_nascimento;

    public Pessoa(String _nome, String _cpf, Date _data) {
        this.nome = _nome;
        this.cpf = _cpf;
        this.data_nascimento = _data;
    }
}
```

- **Continuando o exemplo anterior...**

```
public class Aluno extends Pessoa {
        public Aluno(String _nome, String _cpf, Date _data) {
                super(_nome, _cpf, _data);
        }
        public String matricula;
}
public class Professor extends Pessoa {
        public Professor(String _nome, String _cpf, Date _data) {
                super(_nome, _cpf, _data);
        }
        public double salario;
        public String disciplina;
}
public class Funcionario extends Pessoa {
        public Funcionario(String _nome, String _cpf, Date _data) {
                super(_nome, _cpf, _data);
        }
        public double salario;
        public Date data_admissao;
        public String cargo;
}
```

Polimorfismo

- Caracteriza-se por sobrescrever/reescrever um método da classe mãe, declarando um método com a mesma assinatura
  - Capacidade dos objetos responderem a uma mesma mensagem (chamada de métodos) de modos distintos
  - A sobreposição de métodos ocorre quando há polimorfismo
  - Por exemplo, podemos mover objetos (no mundo real)
    - Carro, bicicleta, caminhão
  - Se considerarmos mover um método de uma classe, ele poderá ser especializado para movimentar o carro, a bicicleta e o caminhão de formas diferentes.

Polimorfismo

- **Como que a JVM (Java Virtual Machine) procura a implementação mais especializada do método?**
  - **Seguindo a hierarquia, ou seja, de baixo para cima. Da classe mais específica (filha) para a classe mais genérica (classe mãe)**
- **Se o método não for definido na classe filha, procura-se pela implementação da classe mãe**
- **Quando um método é sobrescrito na classe filha, ele passa a ter o comportamento padrão dessa classe, embora seja possível acessar o método da superclasse**

Polimorfismo

- Quando ocorre a sobrescrita de métodos?
  - Ocorre quando um método da classe mãe é redefinido na classe filha
    - Mesma assinatura
    - Mesmo tipo de retorno
  - Agora, imagine se desejamos aproveitar o comportamento definido pela classe mãe
    - Chamamos a implementação da classe mãe
      - Para isso, utilizamos a palavra *super*
      - *Quando isso ocorre, o método da classe mãe fica sobreposto (overriding)*

Polimorfismo

- **Exemplo – Criar um Carro de Corrida**
  - **Vamos considerar a Classe Carro**

```
public class Carro {
        private int velocidade;
        public Carro(int velocidadeInicial) {
        velocidade = velocidadeInicial;
}
public void acelera() {
        velocidade++;
}
public void freia() {
        velocidade--;
}
}
```

Polimorfismo

- **Exemplo – <u>Criar um Carro de Corrida</u>**

```
public class CarroCorrida extends Carro {
        public CarroCorrida(int velocidadeInicial) {
        super(velocidadeInicial);
}
public void acelera() {
        velocidade+=5;
}
}
```

- **<u>Forma de chamada:</u>**

```
CarroCorrida x = new CarroCorrida();
x.acelera();
```

Polimorfismo

- No exemplo anterior temos:
  - A subclasse CarroCorrida redefiniu o método acelera()
  - Houve a recodificação do método acelera() que foi herdado da classe mãe, Carro
  - Em tempo de execução, o Java saberá qual implementação deve ser usada

Polimorfismo

- A palavra super tem uma função parecida com a palavra *this*
  - this pode ser usada para referenciar o próprio objeto, permitindo distinguir variáveis locais e campos do objeto que contém os mesmos nomes
- Em herança, Super é usado para acessar campos e métodos da superclasse
  - Campos ocultos
  - Métodos sobrescritos (polimorfismo)
  - Construtores da superclasse

Polimorfismo

- **Uso da palavra Super**

Método Sobrescrito

```
public class Superclasse {
    public void imprimeMetodo() {
        System.out.println("Impresso na Superclasse.");
    }
}
```

```
public class Subclasse extends Superclasse {

    public void imprimeMetodo() { super.imprimeMetodo();
        System.out.println("Impresso na Subclasse");
}

    public static void main(String[] args) { Subclasse s = new
        Subclasse(); s.imprimeMetodo();
}
}
```

- **Qual é a saída gerada?**

Classes e Métodos Final

- Métodos final não podem ser sobrescritos
  - A palavra chave final ou o modificador final significa que o método definido com ela não poderá ser sobrescrito em uma subclasse. Este modificador pode ser aplicado em classes, métodos e atributos
- Métodos privados são apenas acessíveis dentro da classe definida, não sendo possível sobrescrever um método private em uma subclasse, quando usamos o modificador final

Classes e Métodos Final

```
public final class ClasseFinal{
    protected final String nome;
    protected final int idade;
    protected final int[] vetor;

    public ClasseFinal(){
        idade = 40;
        nome = "UNIVESP";
        vet = new int[100];
    }
```

Classes e Métodos Final

```
    public ClasseFinal(int id, String n, int tam) {
        i = id;
        nome = n;
        vet = new int[tam];
    }

    public final void Imprime(final int pos) {
        System.out.println(vetor[pos]);
    }

    public final void inicializa() {
        for (int x = 0; x < vetor.length; x++) {
            vet[x] = x * 5;
        }
    }
}
```

Referências

1. Java Como Programar: Paul Deitel & Harvey Deitel - 10ª Edição

2. Java Como Programar: Paul Deitel & Harvey Deitel - 8ª Edição

PROGRAMAÇÃO ORIENTADA A OBJETOS

Herança e Polimorfismo