
Newton
Quem se prepara, não para.

Programação Orienta a Objetos

3º período

Professora: Michelle Hanne

Sumário

- ✓ Reuso
- ✓ Herança

Reuso

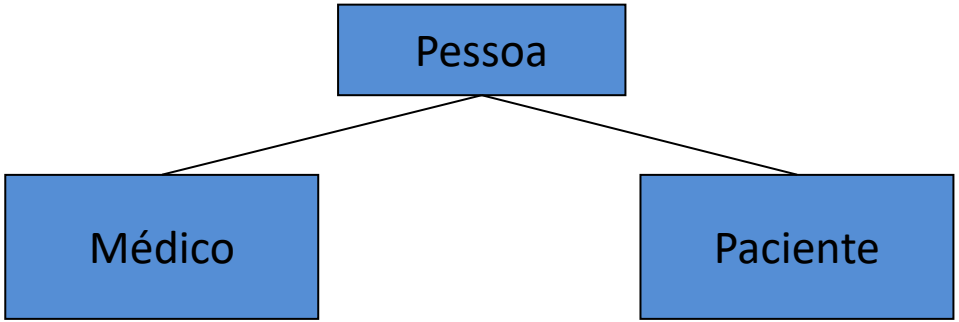
Newton
Quem se prepara, não para.

- O paradigma estruturado é pobre em recursos que propiciem o reuso de software.

- Suponha que você esteja desenvolvendo um software em java para manter o cadastro de **Pacientes** e **Médicos**. Você percebe que há características semelhantes entres estas duas entidades, por exemplo: nome, CPF, RG, endereço, telefone.

Reuso

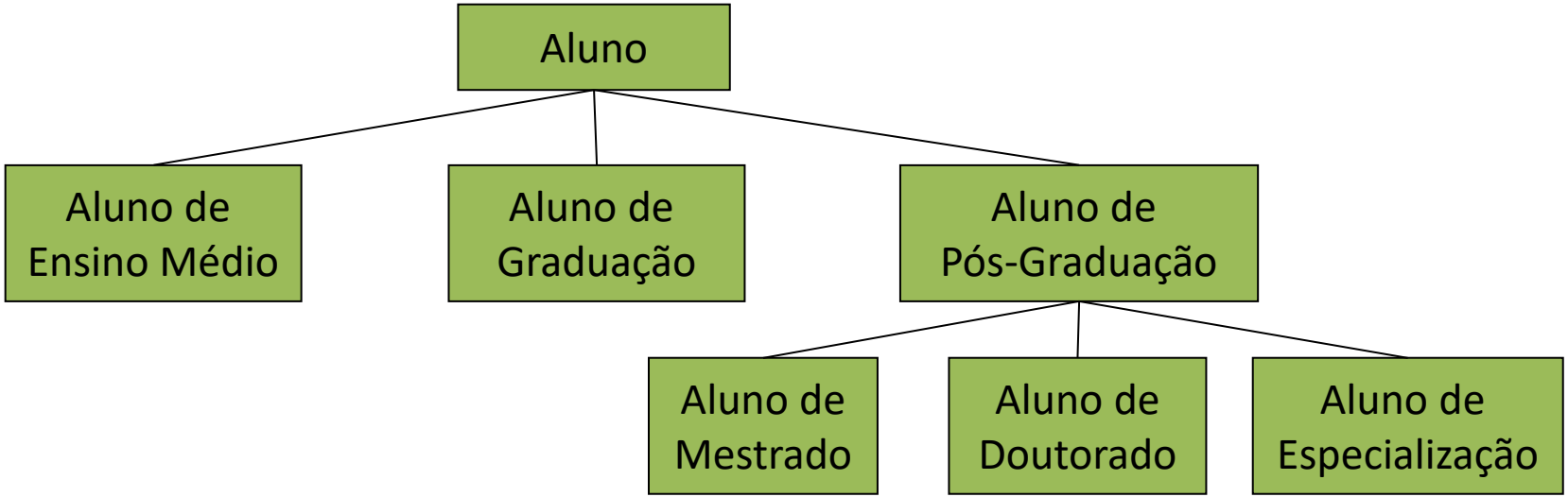
- Situação:



Paciente é uma Pessoa
Médico é uma Pessoa

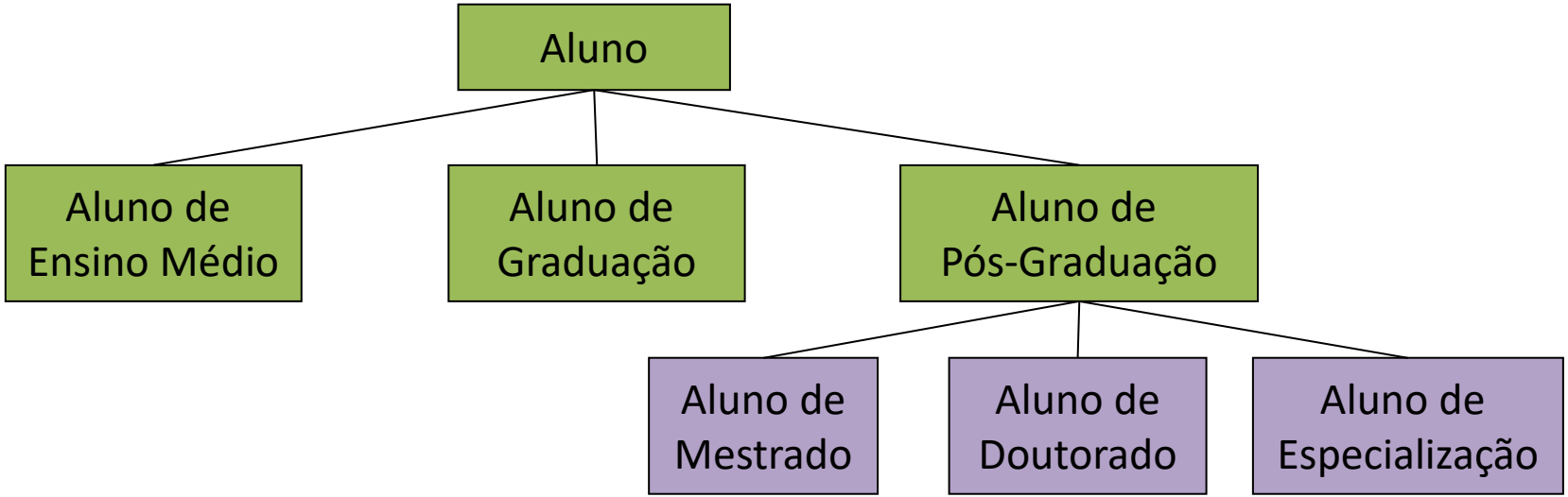
Reuso

- Outra situação:



Reuso

- Outra situação:



Reuso

- Aluno de Ensino Médio é um Aluno
- Aluno de Graduação é um Aluno
- Aluno de Pós Graduação é um Aluno
- Aluno de Mestrado é um Aluno de Pós Graduação
- Aluno de Doutorado é um Aluno de Pós Graduação
- Aluno de Especialização é um Aluno de Pós Graduação

Reuso

Newton
Quem se prepara, não para.

- Estes tipos de relacionamentos são do tipo “é um”.

- **Um relacionamento do tipo “é um” indica que uma entidade possui (herda) características da outra.**

- A programação orientada por objetos possui um recurso poderoso para implementar este tipo de reuso: **Herança.**

Herança

Herança

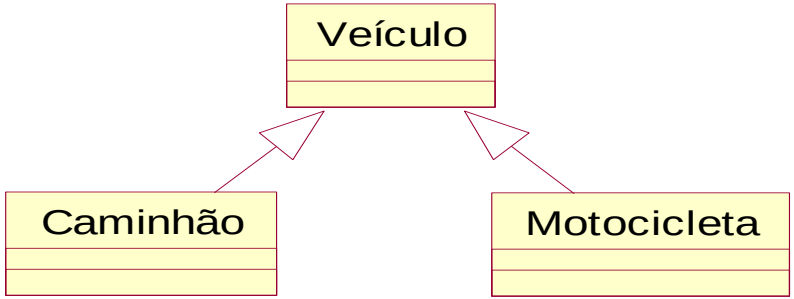
Newton
Quem se prepara, não para.

- Herança é um recurso que permite que novas classes sejam definidas a partir de classes já definidas.

- Herança representa o relacionamento do tipo “é um”.

- Na hierarquia de classes:
  - Super classes (ou ascendente): são as ascendentes de um classe.
  - Sub classes (ou descendente): são as descendentes de um classe
  - Classe mãe: é a ascendente direta de um classe
  - Classe filha: é a descendente direta de uma classe.

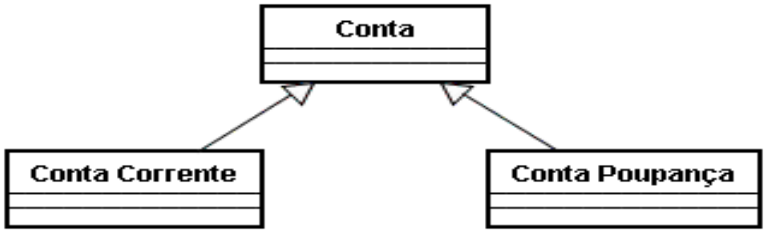
Herança



- Veículo é superclasse de Caminhão e de Motocicleta.
- Caminhão e Motocicleta são subclasses de Veículo.
- Caminhão e Motocicleta herdam as definições da classe Veículo.

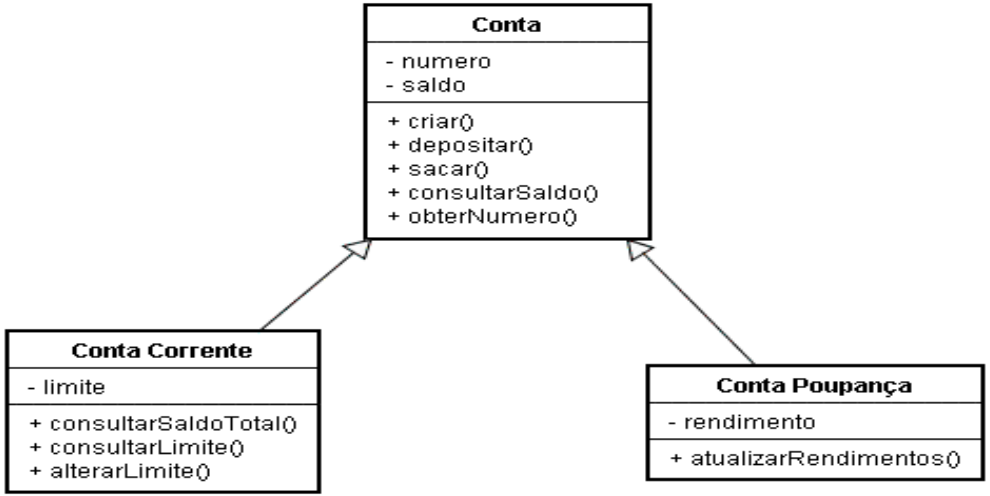
Herança

- No exemplo Conta Bancária:
  - *Conta* é superclasse de *Conta Corrente* e de *Conta Poupança*.
  - *Conta Corrente* e *Conta Poupança* são subclasses de *Conta*.



Herança

- Significa que:
  - *Conta Corrente* herda características e comportamentos de *Conta*.
  - *Conta Poupança* herda características e comportamentos de *Conta*.



Herança

Newton
Quem se prepara, não para.

- No exemplo:
  - *Conta Corrente* possui como atributos: *número*, *saldo* e *limite*, pois herda os dois primeiros da classe *Conta*.
  - *Conta Corrente* possui como métodos: *criar*, *depositar*, *sacar*, *consultarSaldo*, *obterNumero*, e *consultarSaldoTotal*, *consultarLimite* e *alterarLimite*, sendo que os cinco primeiros são herdados de *Conta*.

Herança

Newton
Quem se prepara, não para.

- Exemplo de implementação de herança em Java

  A palavra chave **extends** indica herança em Java.

```
public class A extends B
```

Indica que a classe A herda da classe B

Herança

Newton
Quem se prepara, não para.

- Nos relacionamentos de herança, o modificador de acesso:
  - **public**: indica que o atributo ou método é visível nas subclasses.
  - **private**: indica que o atributo ou método não é visível na subclasse.
  - **protected**: indica que o atributo ou método é visível nas subclasses.

Herança

Newton
Quem se prepara, não para.

- Para invocar o método construtor da superclasse, escreve-se no construtor da subclasse :

```
super(<lista de parâmetros>)
```

- **Redefinição de métodos:** se um método é implementado na subclasse B com a mesma assinatura de um método da superclasse A diz-se que o método foi redefinido. Neste caso, o método que será executado para um objeto da classe B será aquele definido na classe B.

Herança - Exemplo

Newton
Quem se prepara, não para.

```
public class A {
    protected int x, y;
    private int z;

    public A(int a, int b, int c) {
        x = a;
        y = b;
        z = c;
    }

    public int obterX(){
       return (x);
    }
    public int obterY(){
       return (y);
    }
```

Herança - Exemplo

```
public int obterZ(){
    return (z);
}
public void alterarX(int a){
    x = a;
}
public void alterarY(int a){
    y = a;
}
public void alterarZ(int a){
    z = a;
}
```

Herança - Exemplo

Newton
Quem se prepara, não para.

```
 public void ImprimeValores(){
        System.out.println("O valor de X é: " + x);
        System.out.println("O valor de Y é: " + y);
        System.out.println("O valor de Z é: " + z);
    }
}

public class B extends A{
    private int k;

    public B(int a, int b, int c, int d){
       super (a,b,c);
       k = d;
    }
```

Herança - Exemplo

Newton
Quem se prepara, não para.

```
    public void ImprimeValores(){
        System.out.println("O valor de X é: " + x);
        System.out.println("O valor de Y é: " + y);
        System.out.println("O valor de K é: " + k);
        System.out.println("Z não é visível nesta
                            classe");
        System.out.println("O valor de Z da superclasse
                            é: " + obterZ());
    }

}
```

Herança - Exemplo

Newton
Quem se prepara, não para.

```
public class TesteHeranca {

    public static void main (String[] args){
       B obj1 = new B(10,20,30,40);

       obj1.ImprimeValores();

       obj1.alterarY(100);

       obj1.ImprimeValores();

    }
}
```

Herança – Exemplo 2

Newton
Quem se prepara, não para.

O exemplo de código a seguir mostra a implementação das classes `Conta` e `ContaCorrente` em Java. É mostrada também uma classe `MainBanco` para exemplificar o uso das classes criadas.

Herança – Exemplo 2

Newton
Quem se prepara, não para.

```
public class Conta {
    long numero;
    double saldo;

    public Conta(long n) {
        numero = n;
        saldo = 0;
    }

    public void depositar(double v){
       if (v>0)
          saldo = saldo + v;
    }
```

Herança – Exemplo 2

Newton
Quem se prepara, não para.

```
public boolean sacar(double v){
    if ( v>0 && ((saldo-v) >= 0) ){
      saldo = saldo - v;
      return true;
    }
    else
        return false;
 }

 public double consultarSaldo(){
    return(saldo);
 }
```

Herança – Exemplo 2

Newton
Quem se prepara, não para.

```
  public long obterNumero(){
       return(numero);
    }
} // Fim da classe Conta

public class ContaCorrente extends Conta{
    double limite;

    public ContaCorrente(long n, double l) {
        super(n);
        if (l > 0)
           limite = l;
    }
```

Herança – Exemplo 2

Newton
Quem se prepara, não para.

```
public void alterarLimite(double l){
    if (l>0)
        limite = l;
}

public double consultarLimite(){
    return limite;
}
```

Herança – Exemplo 2

Newton
Quem se prepara, não para.

```
 public boolean sacar(double v){
     if (v>0 && ((saldo + limite - v ) >= 0) ){
       saldo = saldo - v;
       return true;
     }
     else
         return false;
    }

    public double consultarSaldoTotal(){
       return(saldo + limite);
    }
} // Fim da classe Conta Corrente.
```

Herança – Exemplo 2 (main)

Newton
Quem se prepara, não para.

```
minhaConta.alterarLimite(200);

System.out.println("Limite: " +
                   minhaConta.consultarLimite());
System.out.println("Saldo Total: " +
                minhaConta.consultarSaldoTotal());


minhaConta.depositar(300);
System.out.println("Saldo após depósito: " +
                  minhaConta.consultarSaldo());
System.out.println("Saldo total após depósito: "
               + minhaConta.consultarSaldoTotal());
```

Herança – Exemplo 2 (main)

Newton
Quem se prepara, não para.

```
if (minhaConta.sacar(200)) {
   System.out.println("Saldo após saque: " +
                  minhaConta.consultarSaldo());
   System.out.println("Saldo total após saque: "
            + minhaConta.consultarSaldoTotal());
}
else
   System.out.println("Não foi possível realizar
       operação. Saldo total disponível é de " +
       minhaConta.consultarSaldoTotal());
```

Herança – Exemplo 2 (main)

Newton
Quem se prepara, não para.

```
        if (minhaConta.sacar(700)) {
           System.out.println("Saldo total após saque: "
                    + minhaConta.consultarSaldoTotal());
        }
        else
           System.out.println("Não foi possível realizar
               operação. Saldo total disponível é de " +
               minhaConta.consultarSaldoTotal());
    }

}//Fim da classe MainBanco
```

Herança

Herança provê **reuso** de classes já construídas.

Alguns benefícios do uso de herança:

- evitar duplicação de código;
- reúso de código;
- manutenção mais fácil (desde que não haja abuso do recurso);
- extensibilidade.

Referências

SILVA, Fabricio Machado da. **Paradigmas de programação**. SAGAH, 2019. ISBN digital: 9788533500426

Barnes, David e Kölling, M. **Programação Orientada a Objetos com Java**. São Paulo: Pearson Prentice Hall, 2004.

Deitel, H. M.; Deitel, P. J. **Java - Como Programar**. 6. ed. Prentice-Hall, 2005. Capítulo 4 e 5.

PRESSMAN, Roger S. **Engenharia de Software**. Rio de Janeiro: MacGraw Hill, 2002.