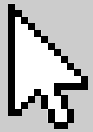
PUC Minas

Programação Modular



PUC Minas
Bacharelado em Engenharia de Software
Prof. Daniel Kansaon

Slides adaptados do professor Danilo Boechat

Interfaces
e
Polimorfismo

Composição x Especialização

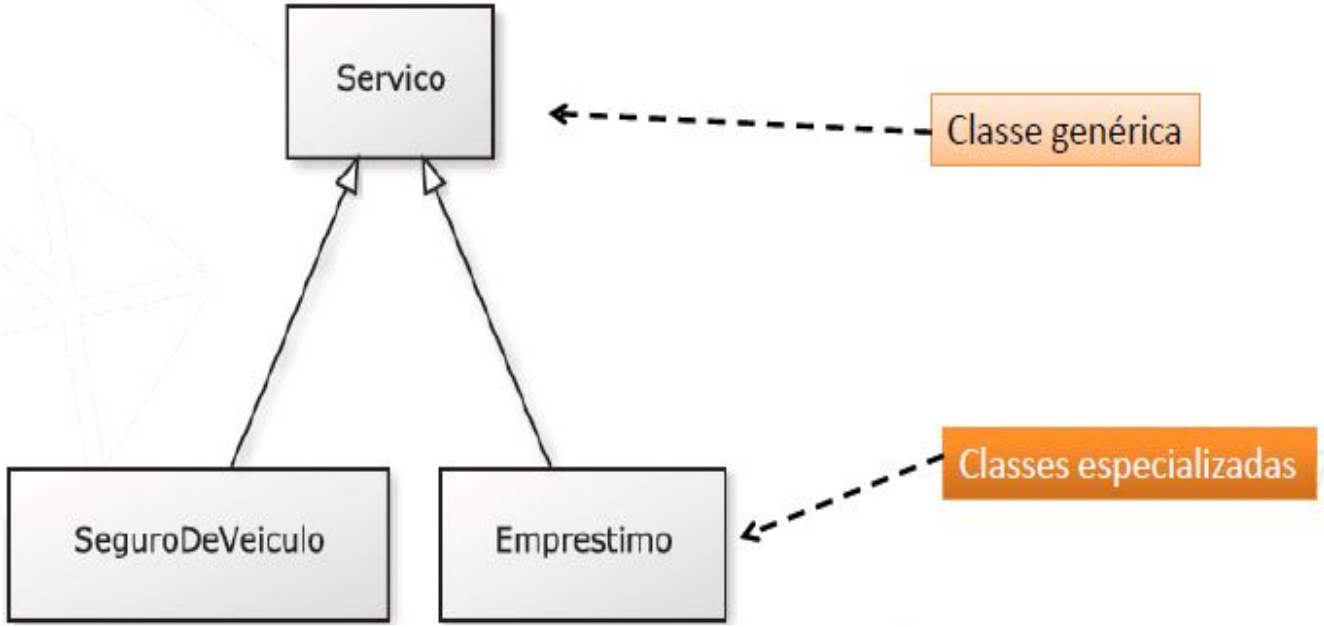

POO Tradicional

Especialização/herança largamente utilizados:

- Recurso poderoso para relacionamento entre classes
- Reutilização de código
- Polimorfismo

São considerados como a maior contribuição do paradigma

Herança


Servico
Classe genérica
SeguroDeVeiculo
Emprestimo
Classes especializadas

POO Tradicional

Mas, pensando bem...

Como fica o **encapsulamento**?

Como fica o **acoplamento**?

E se houver necessidade de **mudança de comportamento** em tempo de execução?

POO Tradicional

Como fica o **encapsulamento**?

- Encapsulamento pode ser comprometido
- Subclasses têm acesso direto aos métodos e atributos da classe pai

Como fica o **acoplamento**?

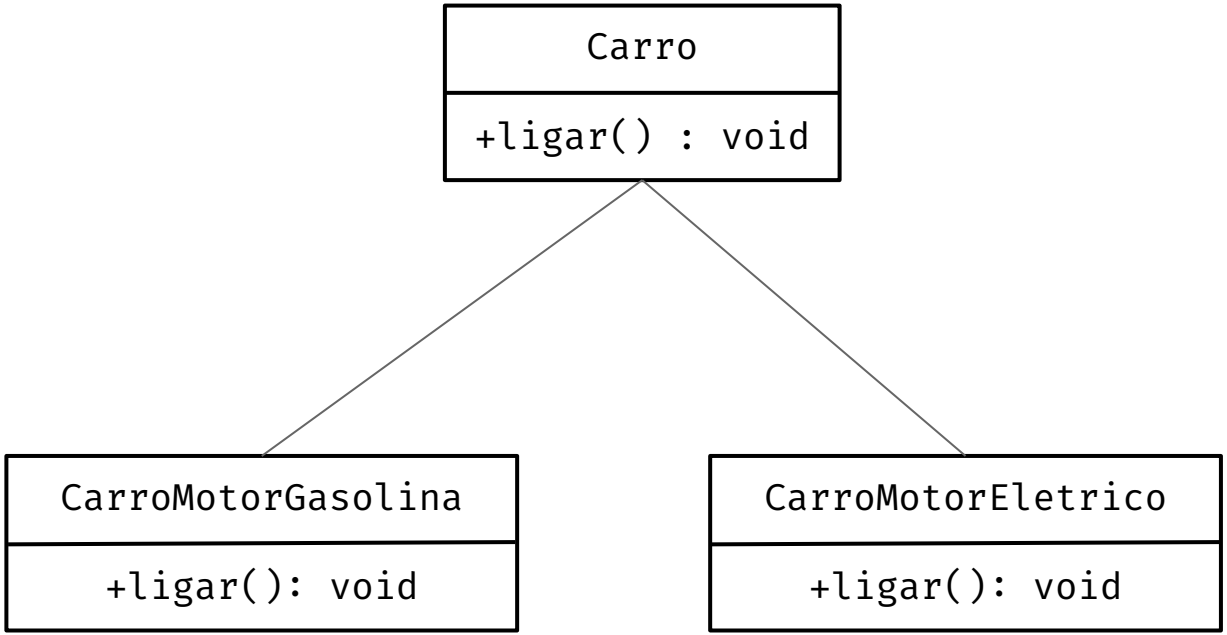
- Acoplamento entre as classes se torna muito forte
- Mudanças na classe pai afetam diretamente todas as subclasses

POO Tradicional

E se houver necessidade de **mudança de comportamento** em tempo de execução?

- Mudar o comportamento em tempo de execução é muito difícil
- O comportamento da subclasse é fixo

EXEMPLO CARRO



EXEMPLO CARRO

```
class Carro {
    public void ligar() {
        System.out.println("Carro ligando.");
    }
}
```

```
class CarroMotorGasolina extends Carro {
    public void ligar() {
        System.out.println("Carro a gasolina ligando");
    }
}

class CarroMotorEletrico extends Carro {
    public void ligar() {
        System.out.println("Carro elétrico ligando (energia).");
    }
}
```

Quais os Problemas?

Limitações da Implementação

- Qualquer alteração na classe base Carro pode **afetar as subclasses**, exigindo revisões e possíveis modificações

- E se quisermos um **carro híbrido**? Motor a gasolina + motor elétrico

- Motor fica atrelado ao conceito de "carro". E se quisermos usar um **motor** a gasolina em **outro veículo** (um barco)?

Qual a solução?

Substituindo Herança por Composição

"Prefira composição à herança."

— Gang of Four, Design Patterns: Elements of Reusable Object-Oriented Software (1994)

Substituindo Herança por Composição

Por meio do uso de interfaces e composição de classes, podemos **injetar comportamento** em classes

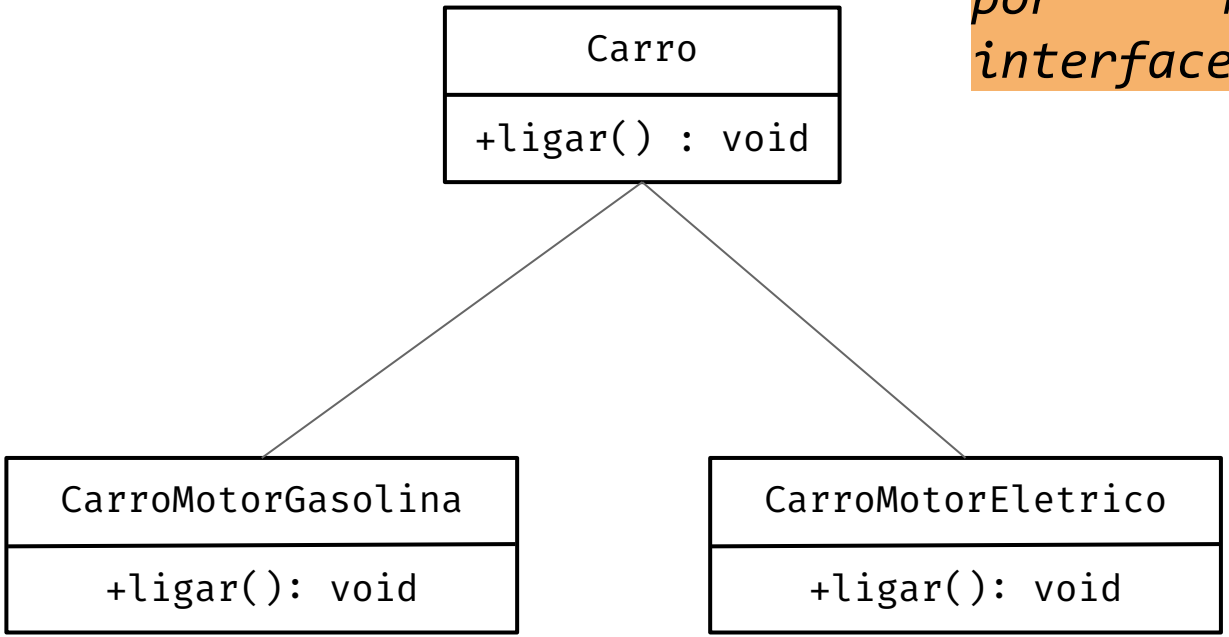
- Delegação de responsabilidades
- Mudança de comportamento em tempo de execução

Exemplo

Qual a solução para o exemplo do carro?

EXEMPLO CARRO

Solução: *composição por meio de interface*



Exemplo Carro

```java
interface Motor {
    void ligar();
}

class MotorGasolina implements Motor {
    @Override
    public void ligar() {
        System.out.println("Motor a gasolina ligado.");
    }
}

class MotorEletrico implements Motor {
    @Override
    public void ligar() {
        System.out.println("Motor elétrico ligado.");
    }
}
```

```java
class Carro {
    private Motor motor;

    public Carro(Motor motor) {
        this.motor = motor;
    }

    public void ligar() {
        motor.ligar();
    }
}
```

Composição :)

Exemplo - Universidade e seus Alunos

Em uma grande universidade, são oferecidos cursos de graduação e pós-graduação

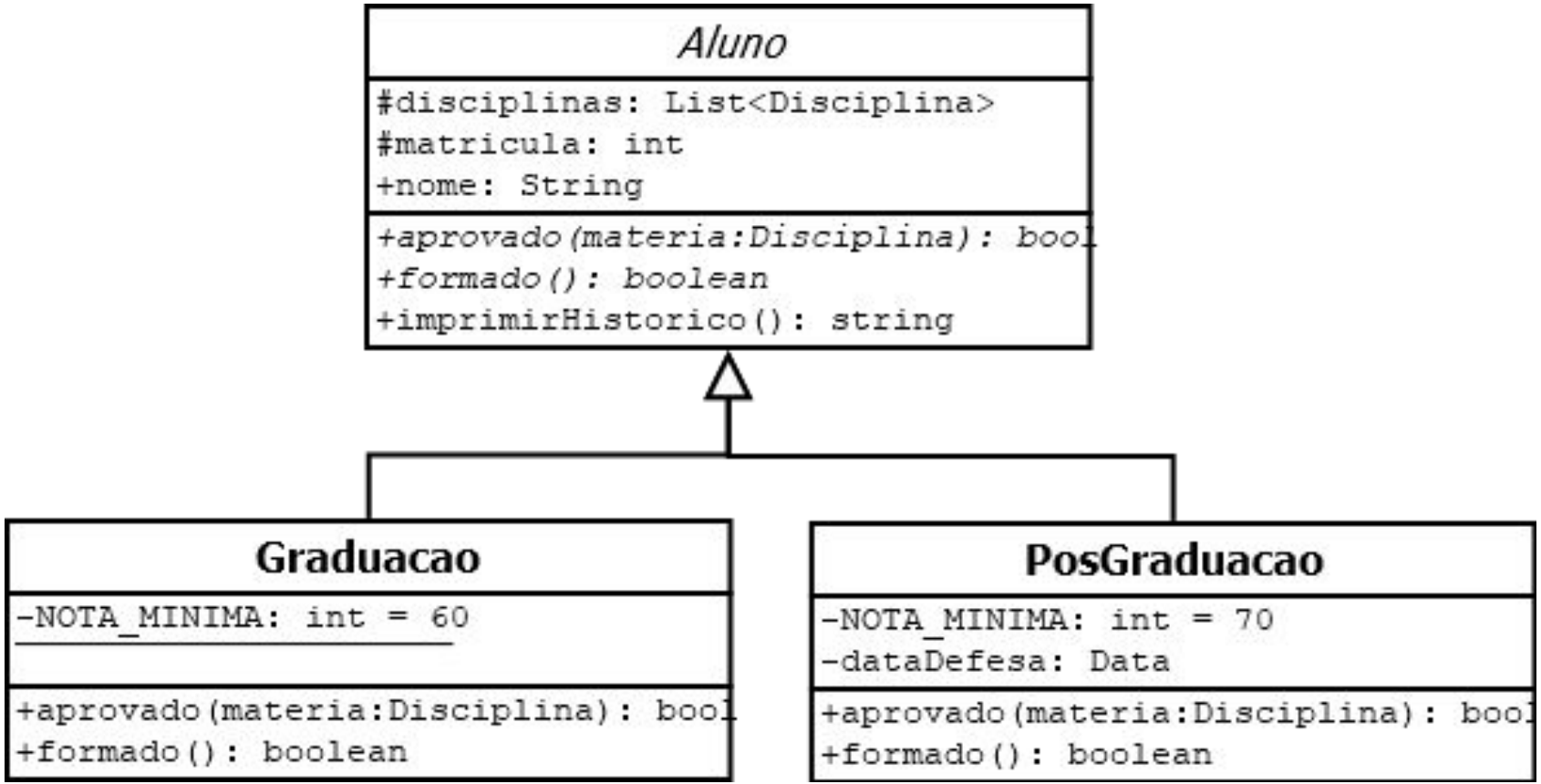
- Aprovação em disciplinas
- Formatura

Universidade e seus Alunos

- Alunos da graduação são aprovados com 60 pontos
- Alunos da pós-graduação precisam de 70 pontos

- Alunos da graduação se formam após a integralização de certo número de créditos
- Alunos da pós-graduação precisam, além dos créditos, defender sua dissertação ou tese

Universidade e seus Alunos



Exemplo simplificado de implementação

```java
abstract class Aluno{
    public String nome;
    protected int matricula;
    protected double[] notas;

    abstract boolean aprovado();

    double calcularMedia(){
        double soma = 0;
        for (double nota : notas) {
            soma += nota;
        }
        return soma / notas.length;
    }
}
```

```java
class Graduacao extends Aluno{
    @Override
    boolean aprovado() {
        return calcularMedia() >= 6.0;
    }
}
```

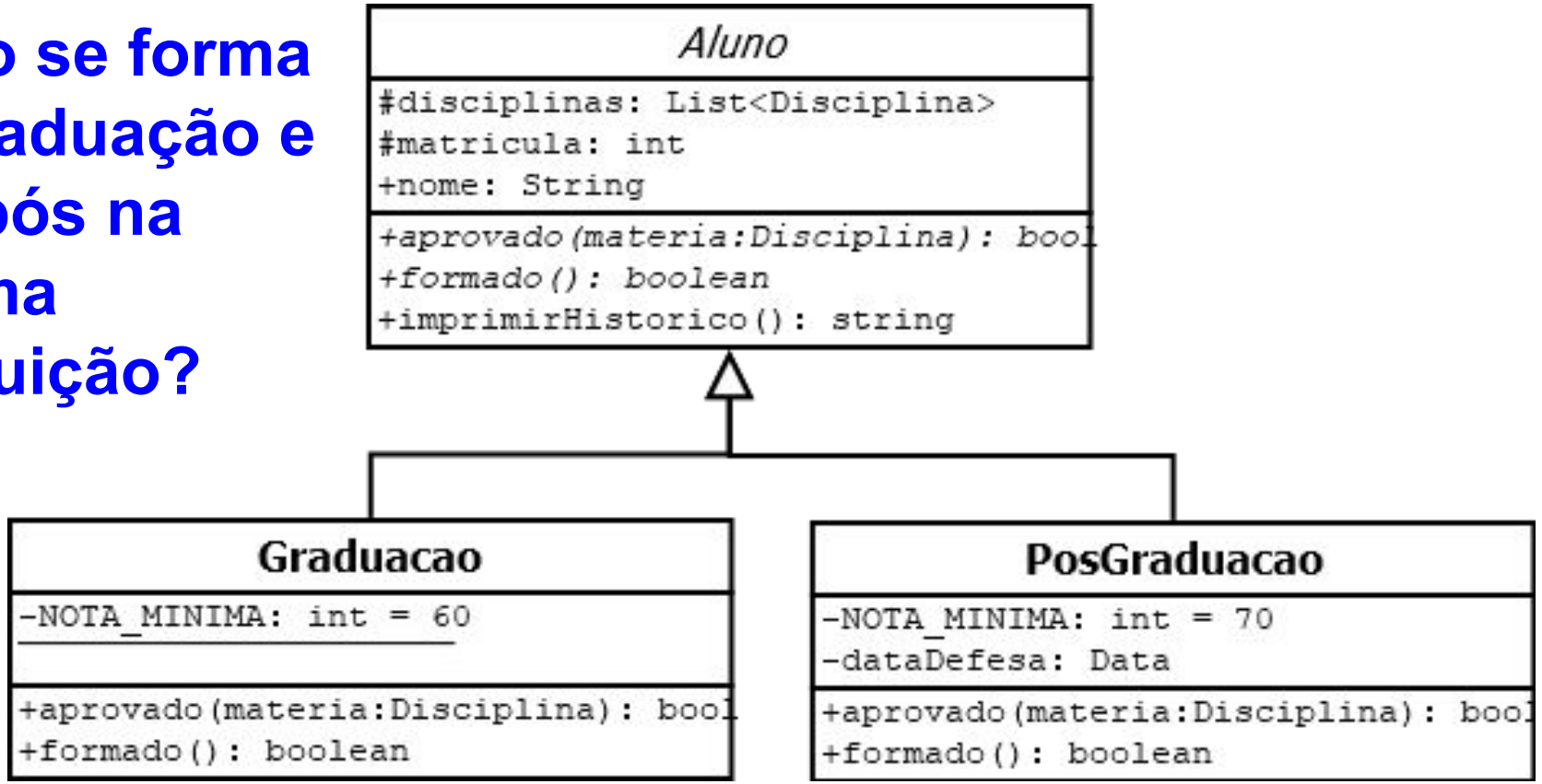
O que acontece se calcularMedia() mudar?

```java
raduacao extends Aluno{
ide
    boolean aprovado() {
        return calcularMedia() >= 7.0;
    }
}
```

Qual outro problema?

Universidade e seus Alunos

Aluno se forma na graduação e fará pós na mesma instituição?



Universidade e seus Alunos

Vamos imaginar que um aluno formou e entrou na pós graduação.

Como transformar um objeto Graduacao em um objeto PosGraduacao? Como representar essa mudança?

Universidade e seus Alunos

```java
abstract class Aluno{
    public String nome;
    protected int matricula;
    protected double[] notas;

    public Aluno(String nome, int matricula, double[] notas){
        this.nome = nome;
        this.matricula = matricula;
        this.notas = notas;
    }

    abstract boolean aprovado();

    double calcularMedia(){
        double soma = 0;
        for (double nota : notas) {
            soma += nota;
        }

        return soma / notas.length;
    }
}
```

```java
class Graduacao extends Aluno{
    public Graduacao(String nome, int matricula, double[] notas) {
        super(nome, matricula, notas);
    }

    @Override
    boolean aprovado() {
        return calcularMedia() >= 6.0;
    }
}
```

```java
class PosGraduacao extends Aluno{

    public PosGraduacao(String nome, int matricula, double[] notas) {
        super(nome, matricula, notas);
    }

    @Override
    boolean aprovado() {
        return calcularMedia() >= 7.0;
    }
}
```

Universidade e seus Alunos

E se eu quiser
alterar aluno
para graduação?

```
public static void main(String[] args) {
    Aluno aluno = new Graduacao("João", 321456, new double[]{8, 9, 7});
    System.out.println(aluno.aprovado());  // Usará a aprovação de graduação
```

Universidade e seus Alunos

O aluno não pode mudar de tipo sem recriar uma nova instância do tipo PosGraduacao.

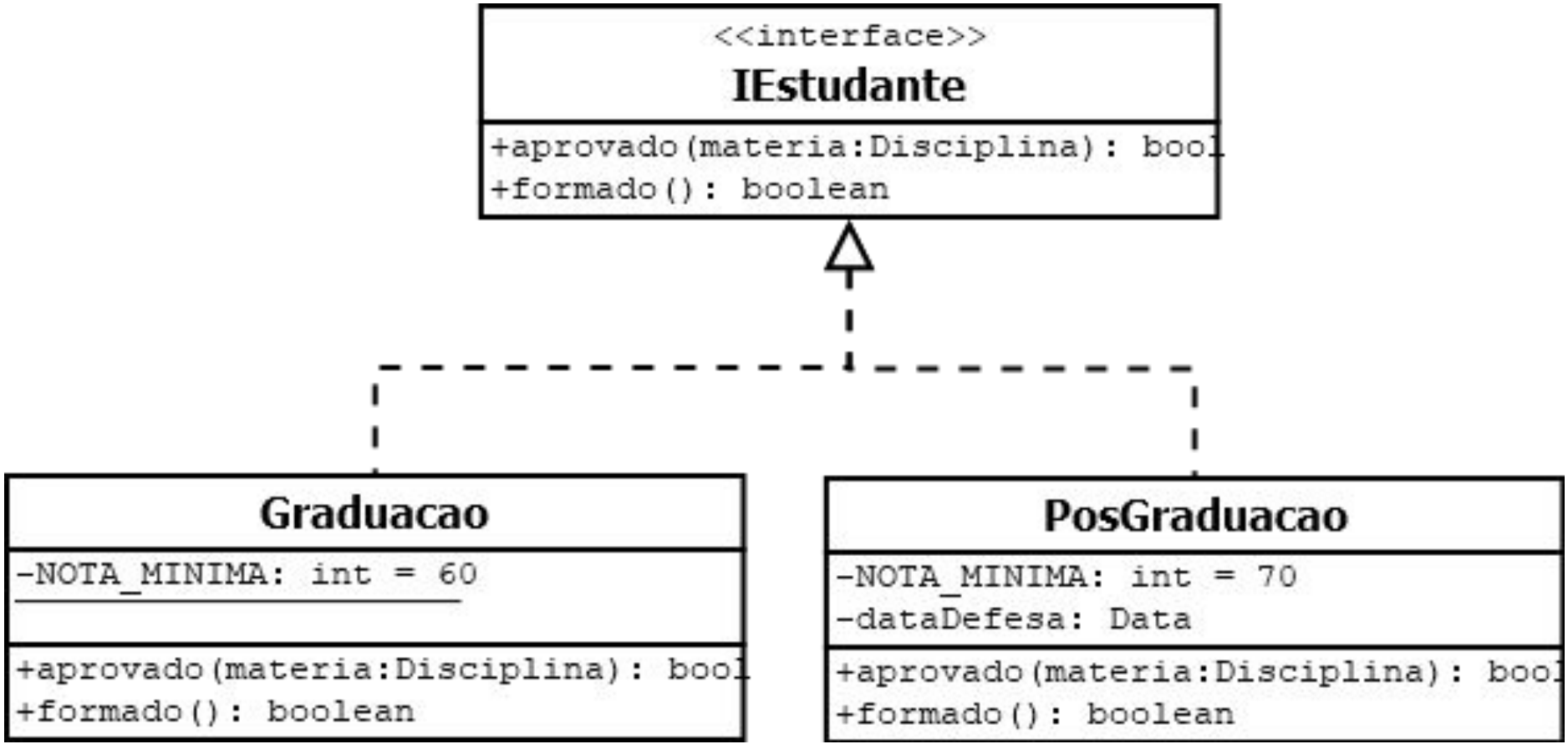
```
public static void main(String[] args) {
    Aluno aluno = new Graduacao("João", 321456, new double[]{8, 9, 7});
    System.out.println(aluno.aprovado());  // Usará a aprovação de graduação

    // Agora o aluno decide mudar para pós-graduação, mas...
    aluno = new PosGraduacao("João", 456789, new double[]{5, 3, 8});
    System.out.println(aluno.aprovado());  // Usará a aprovação de pós-graduação
```

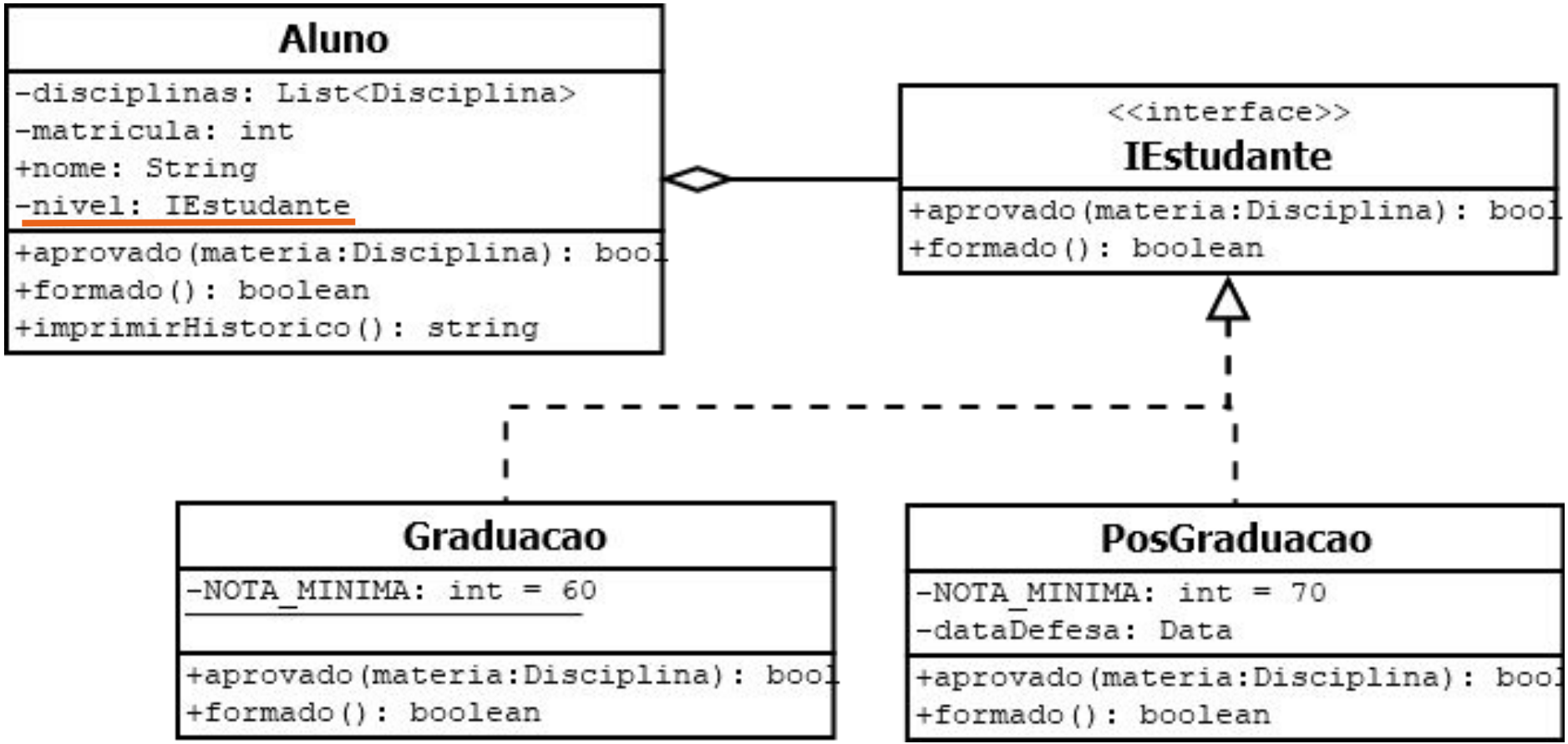
E se usar composição?

Universidade e seus Alunos

Universidade e seus Alunos



Universidade e seus Alunos



Universidade e seus Alunos

```
Aluno estudante = new Aluno ("João", 123, new
double[]{6, 7, 5.5}, new Graduacao());


 //alterando tipo do aluno

 estudante.setTipo(new PosGraduacao());
```

Por quê Interfaces?

Por que Interfaces?

- Capturar similaridades e padronizar métodos de classes não relacionadas
- Composição x Especialização/Herança
- Declarar métodos que representam comportamentos comuns de várias classes
- Polimorfismo paramétrico
- Revelar interfaces sem revelar os objetos que a implementam: útil em pacotes de componentes
- Programação remota / padronização de serviços

Quando usar Interfaces?

- Situações nas quais ambos se equivalem

- Situações que não conseguimos resolver sem utilizar herança múltipla

- Situações nas quais herança trará um acoplamento inadequado (acima do já esperado)

Quando usar herança?

Herança x composição: regras de Coad

As 5 regras de Peter Coad.

1. Subclasse denota “é um tipo especial de” e não “um papel assumido por”, nem “tem um”
2. Uma instância de uma subclasse nunca precisará mudar para outra subclasse
3. Uma subclasse estende e não sobrescreve ou anula as responsabilidades da classe mãe

1- COAD, Peter; YOURDON, Edward. **Analise baseada em objetos.** Rio de Janeiro: Campus, 1992.

Herança x composição: regras de Coad

4. Uma subclasse não estende as capacidades de uma classe utilitária

5. Para uma classe do domínio problema tratado, a subclasse especializa um papel, transação ou dispositivo

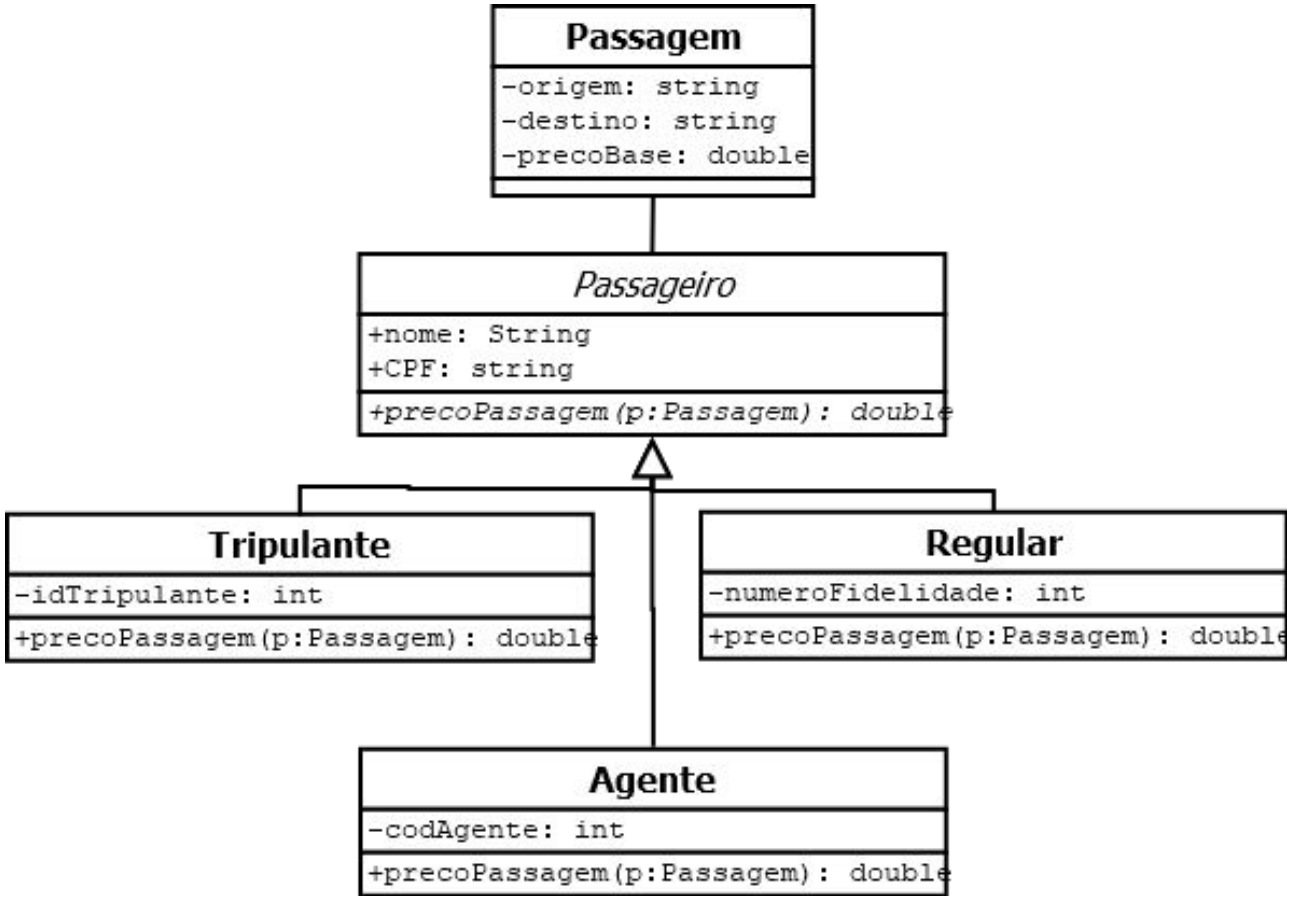
1- COAD, Peter; YOURDON, Edward. **Analise baseada em objetos.** Rio de Janeiro: Campus, 1992.

Herança ou composição?

Uma companhia aérea vende passagens com regras de preços diferentes para:

- Seus tripulantes
- Agentes de viagem
- Passageiros regulares

Herança ou composição?



Uma companhia aérea vende passagens com regras de preços diferentes para:

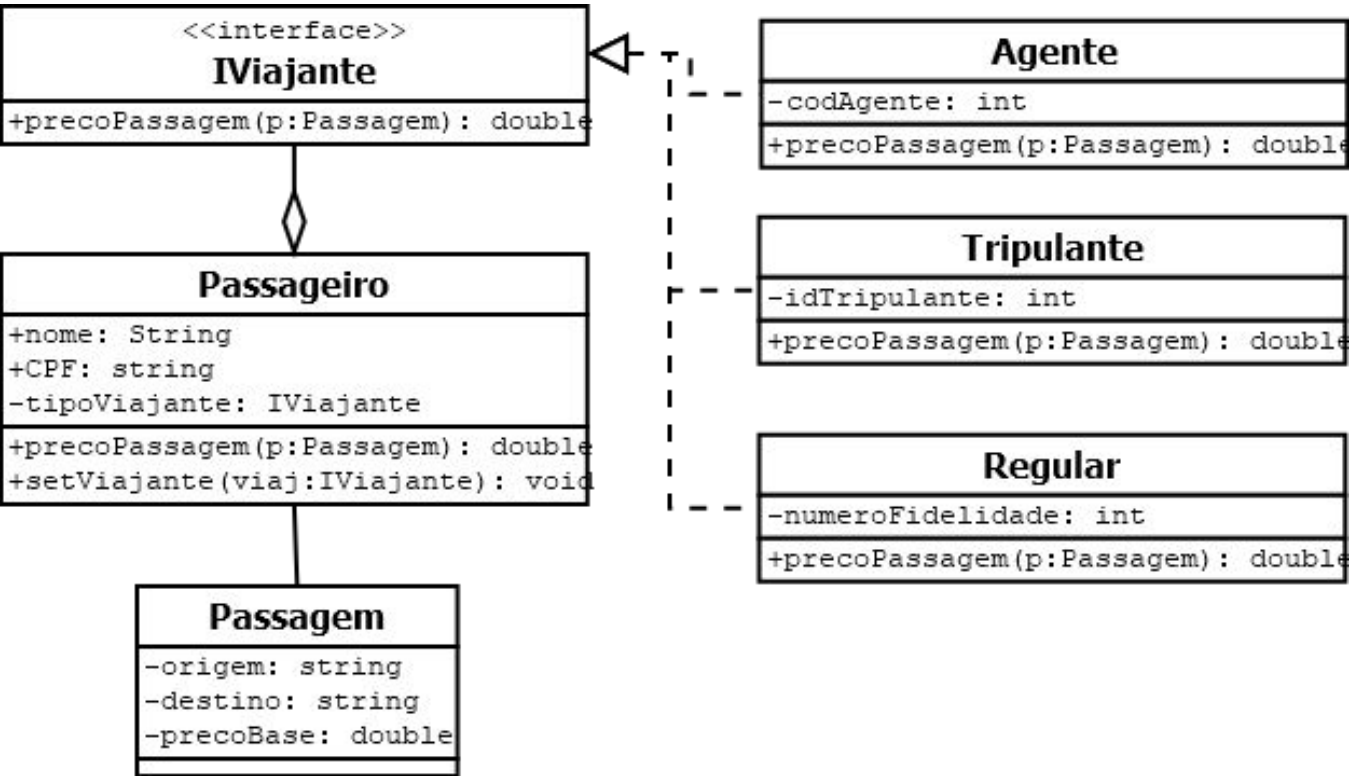
- Seus tripulantes
- Agentes de viagem
- Passageiros regulares

Herança ou composição?

Tripulantes, agentes, regulares:

- "tipo especial" ou "tem papel"?
  ↳ Papel assumido por uma pessoa
- Podem mudar de papel/subclasse ao longo do tempo, em tempo de execução?
  ↳ Sim, podem
- Anula ou sobrescreve responsabilidades da classe mãe?
  ↳ Ok ✔️
- Não estende as capacidades de uma classe utilitária?
  ↳ Ok ✔️
- A subclasse especializa um papel, transação ou dispositivo?
  ↳ Não. Especializa uma pessoa, e não um papel dela.

Herança ou composição?



Uma companhia aérea vende passagens com regras de preços diferentes para:

- Seus tripulantes
- Agentes de viagem
- Passageiros regulares

Para Casa

Considerando o cenário de um sistema de reservas, onde existem salas standard, vip e premium.

O que é mais apropriado. Herança ou Composição?

Responda usando as 5 regras de Peter Coad.

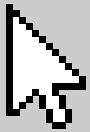
Dúvidas?

Material

https://github.com/danielkansaon/pm_material

PUC Minas

Obrigado!



PUC Minas
Bacharelado em Engenharia de Software
Prof. Danilo Boechat Seufitelli
daniloboechat@pucminas.br