

# Herança

Fátima L. S. Nunes

Luciano A. Digiampietri

Norton T. Roman



# Motivação

- Você está desenvolvendo um sistema de gerenciamento de pessoal para a USP
- Precisa fazer:
  - Cadastro de alunos e professores
  - Há professores doutores e não doutores



```
class Aluno
{
    private long _nrUsp;
    private String _rg;
    private long _cpf;
    private String _nome;
    private char _sexo;
    private int _anoDeIngresso;
    private String _curso;
    // métodos de acesso (public)
    (...)
    public void imprimeDados(){
        System.out.println("Nome: " + _nome);
        System.out.println("Nr USP: " + _nrUsp);
        System.out.println("CPF: " + _cpf);
        System.out.println("RG: " + _rg);
        System.out.println("Sexo: " + _sexo);
        System.out.println("Ano de ingresso: " + _anoDeIngresso);
        System.out.println("Curso: " + _curso);
    }
}
```

```
class Professor
{
    private long _nrUsp;
    private String _rg;
    private long _cpf;
    private String _nome;
    private char _sexo;
    private int _anoDeAdmissao;
    private String _departamento;
    // métodos de acesso (public)
    (...)
    public void imprimeDados(){
        System.out.println("Nome: " + _nome);
        System.out.println("Nr USP: " + _nrUsp);
        System.out.println("CPF: " + _cpf);
        System.out.println("RG: " + _rg);
        System.out.println("Sexo: " + _sexo);
        System.out.println("Ano de admissão: " + _anoDeAdmissao);
        System.out.println("Departamento: " + _departamento);
    }
}
```

```

class ProfessorDoutor
{
    private long _nrUsp;
    private String _rg;
    private long _cpf;
    private String _nome;
    private char _sexo;
    private int _anoDeAdmissao;
    private String _departamento;
    private int _anoObtencaoDoutorado;
    private String _instituicaoDoutorado;
    // métodos de acesso (public)
    (...)
    public void imprimeDados(){
        System.out.println("Nome: " + _nome);
        System.out.println("Nr USP: " + _nrUsp);
        System.out.println("CPF: " + _cpf);
        System.out.println("RG: " + _rg);
        System.out.println("Sexo: " + _sexo);
        System.out.println("Ano de admissão: " + _anoDeAdmissao);
        System.out.println("Departamento: " + _departamento);
        System.out.println("Ano de obtenção do título de Doutor: " + _anoObtencaoDoutorado);
        System.out.println("Instituição do Doutorado: " + _instituicaoDoutorado);
    }
}

```



- O que você vê de estranho aqui? Algo o incomoda?



- O que você vê de estranho aqui? Algo o incomoda?
- São três classes com quase todo o código repetido!!!

```
class Aluno
{
    private long _nrUsp;
    private String _rg;
    private long _cpf;
    private String _nome;
    private char _sexo;
    private int _anoDeIngresso;
    private String _curso;
    // métodos de acesso (public)
    (...)
    public void imprimeDados(){
        System.out.println("Nome: " + _nome);
        System.out.println("Nr USP: " + _nrUsp);
        System.out.println("CPF: " + _cpf);
        System.out.println("RG: " + _rg);
        System.out.println("Sexo: " + _sexo);
        System.out.println("Ano de ingresso: " + _anoDeIngresso);
        System.out.println("Curso: " + _curso);
    }
}
```



```
class Professor
{
    private long _nrUsp;
    private String _rg;
    private long _cpf;
    private String _nome;
    private char _sexo;
    private int _anoDeAdmissao;
    private String _departamento;
    // métodos de acesso (public)
    (...)
    public void imprimeDados(){
        System.out.println("Nome: " + _nome);
        System.out.println("Nr USP: " + _nrUsp);
        System.out.println("CPF: " + _cpf);
        System.out.println("RG: " + _rg);
        System.out.println("Sexo: " + _sexo);
        System.out.println("Ano de admissão: " + _anoDeAdmissao);
        System.out.println("Departamento: " + _departamento);
    }
}
```

```

class ProfessorDoutor
{
    private long _nrUsp;
    private String _rg;
    private long _cpf;
    private String _nome;
    private char _sexo;
    private int _anoDeAdmissao;
    private String _departamento;
    private int _anoObtencaoDoutorado;
    private String _instituicaoDoutorado;
    // métodos de acesso (public)
    (...)
    public void imprimeDados(){
        System.out.println("Nome: " + _nome);
        System.out.println("Nr USP: " + _nrUsp);
        System.out.println("CPF: " + _cpf);
        System.out.println("RG: " + _rg);
        System.out.println("Sexo: " + _sexo);
        System.out.println("Ano de admissão: " + _anoDeAdmissao);
        System.out.println("Departamento: " + _departamento);
        System.out.println("Ano de obtenção do título de Doutor: " + _anoObtencaoDoutorado);
        System.out.println("Instituição do Doutorado: " + _instituicaoDoutorado);
    }
}

```



```

class Professor
{
    private long _nrUsp;
    private String _rg;
    private long _cpf;
    private String _nome;
    private char _sexo;
    private int _anoDeAdmissao;
    private String _departamento;
    // métodos de acesso (public)
    (...)
    public void imprimeDados(){
        System.out.println("Nome: " + _nome);
        System.out.println("Nr USP: " + _nrUsp);
        System.out.println("CPF: " + _cpf);
        System.out.println("RG: " + _rg);
        System.out.println("Sexo: " + _sexo);
        System.out.println("Ano de admissão: " + _anoDeAdmissao);
        System.out.println("Departamento: " + _departamento);
    }
}

```

```

class ProfessorDoutor
{
    private long _nrUsp;
    private String _rg;
    private long _cpf;
    private String _nome;
    private char _sexo;
    private int _anoDeAdmissao;
    private String _departamento;
    private int _anoObtencaoDoutorado;
    private String _instituicaoDoutorado;
    // métodos de acesso (public)
    (...)
    public void imprimeDados(){
        System.out.println("Nome: " + _nome);
        System.out.println("Nr USP: " + _nrUsp);
        System.out.println("CPF: " + _cpf);
        System.out.println("RG: " + _rg);
        System.out.println("Sexo: " + _sexo);
        System.out.println("Ano de admissão: " + _anoDeAdmissao);
        System.out.println("Departamento: " + _departamento);
        System.out.println("Ano de obtenção do título de Doutor: " + _anoObtencaoDoutorado);
        System.out.println("Instituição do Doutorado: " + _instituicaoDoutorado);
    }
}

```



- O que você vê de estranho aqui? Algo o incomoda?
- São três classes com quase todo o código repetido!!!
  - Repetição do trabalho:
    - desperdício de tempo de desenvolvimento
    - desperdício de tempo de validação do código (teste das mesmas coisas....)
  - Mudanças em uma parte comum a elas devem ser feitas (e testadas novamente) nas três classes!
- Há uma solução?

- O que você vê de estranho aqui? Algo o incomoda?
- São três classes com quase todo o código repetido!!!
  - Repetição do trabalho
    - desperdício de tempo de desenvolvimento
    - desperdício de tempo de validação do código (teste das mesmas coisas....)
  - Mudanças em uma parte comum a elas devem ser feitas (e testadas novamente) nas três classes!
- Há uma solução?

## REUSO DE SOFTWARE

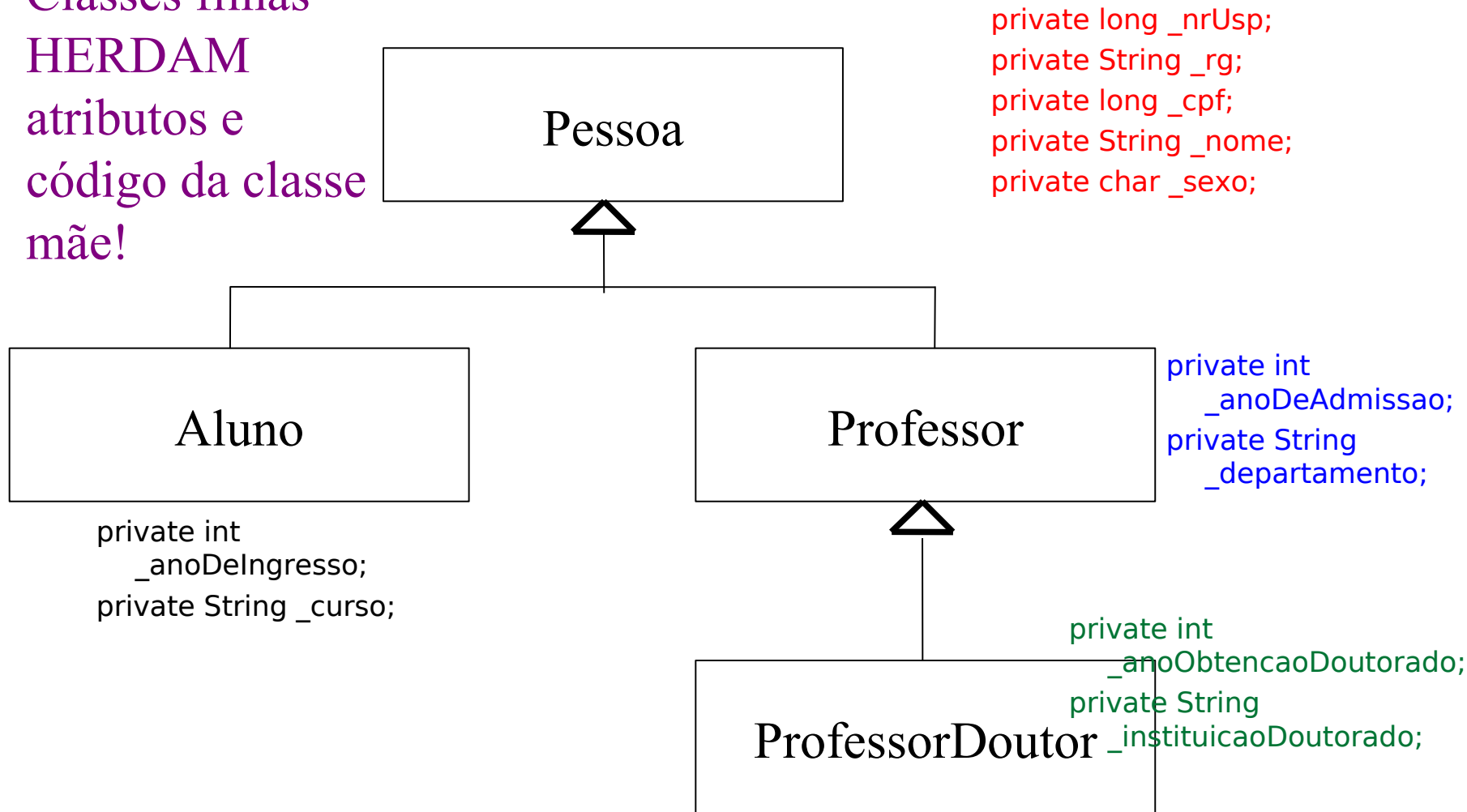


# Reuso e Herança

- **Reuso** é outro conceito importante, não só em Programação Orientado a Objetos como em bom desenvolvimento de software em geral
- Uma das formas de fazer REUSO em POO é através do mecanismo de **herança**
- Fatorar o que é comum, e implementar o que for diferente

# Herança

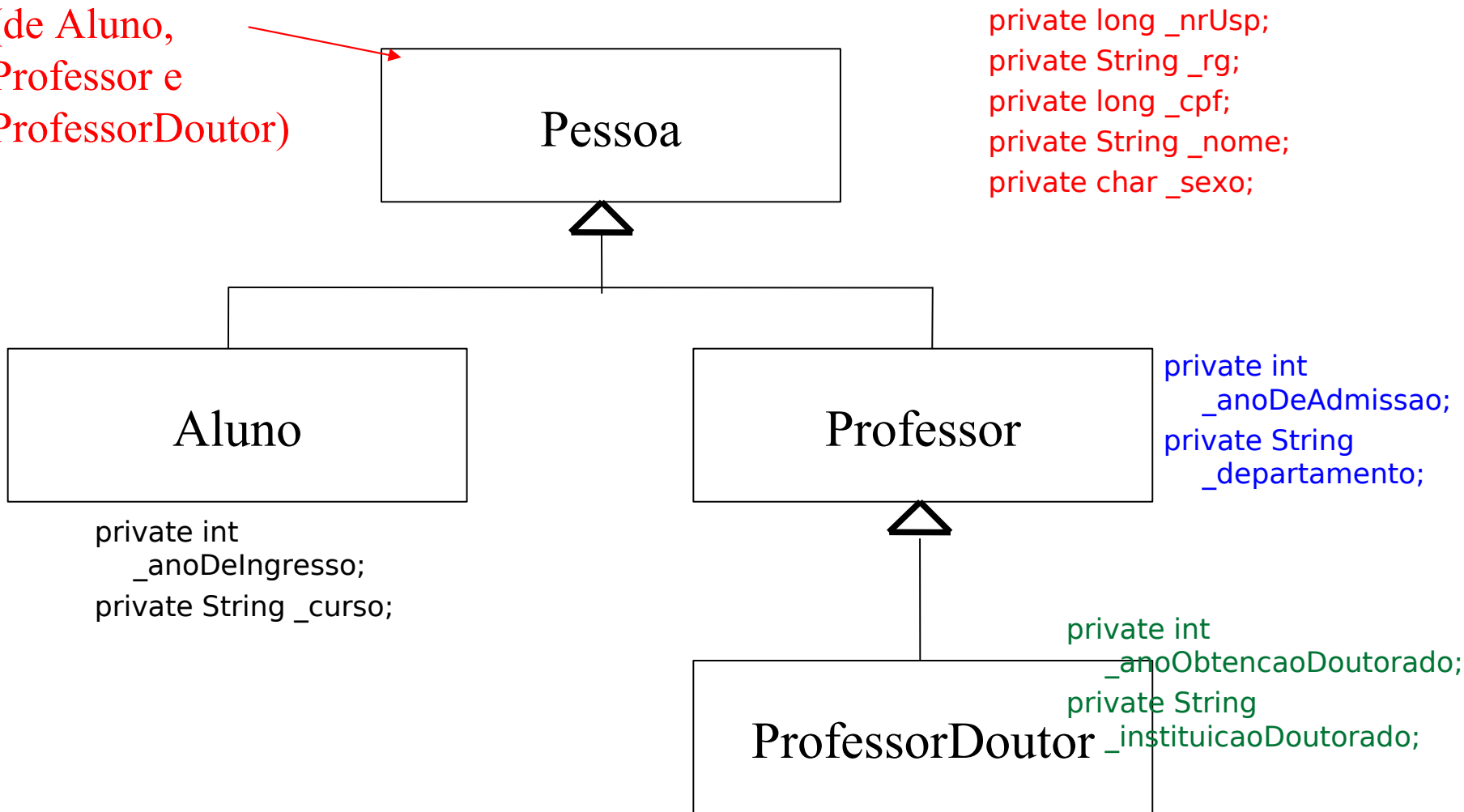
Classes filhas  
HERDAM  
atributos e  
código da classe  
mãe!





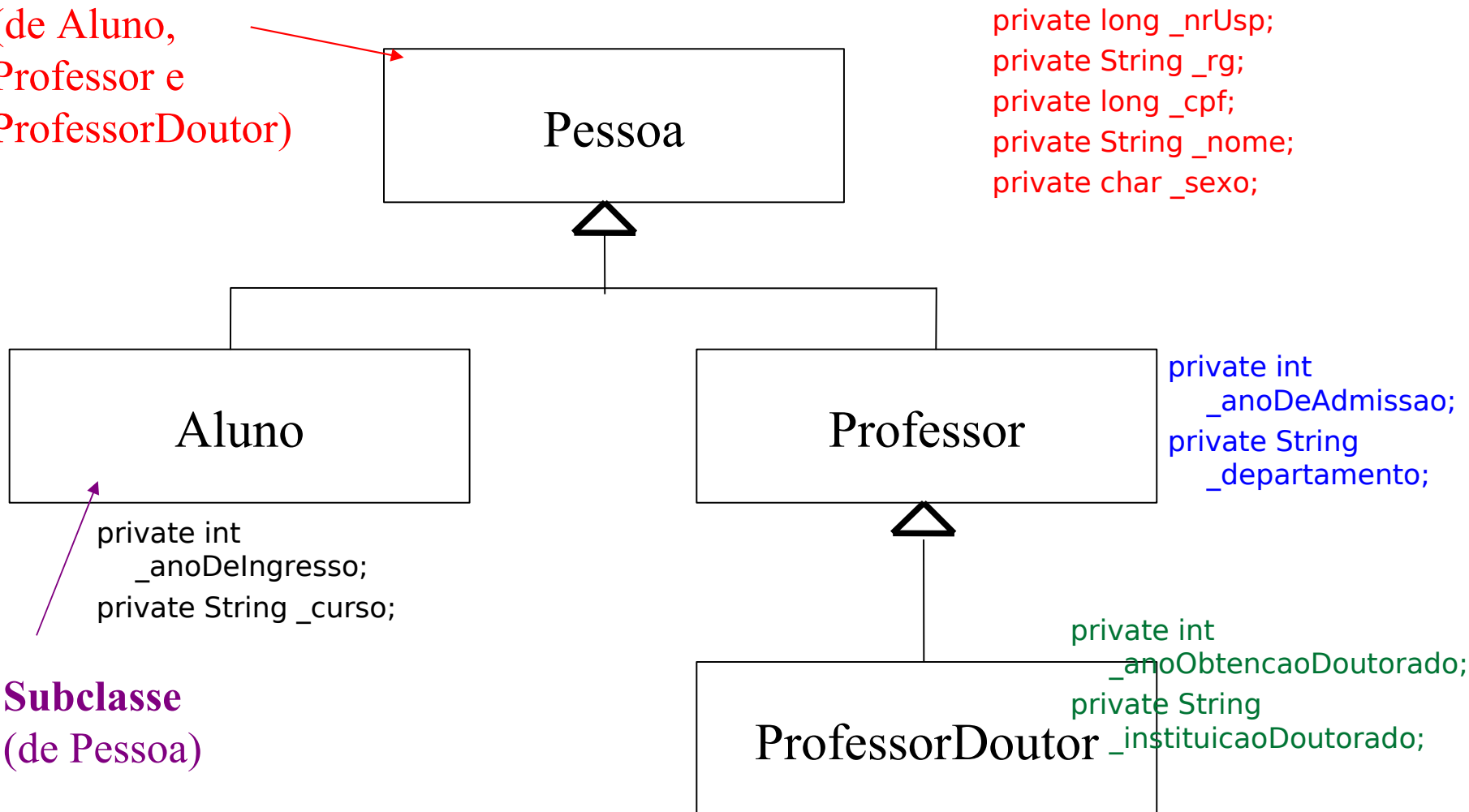
# Herança

**Superclasse**  
(de Aluno,  
Professor e  
ProfessorDoutor)



# Herança

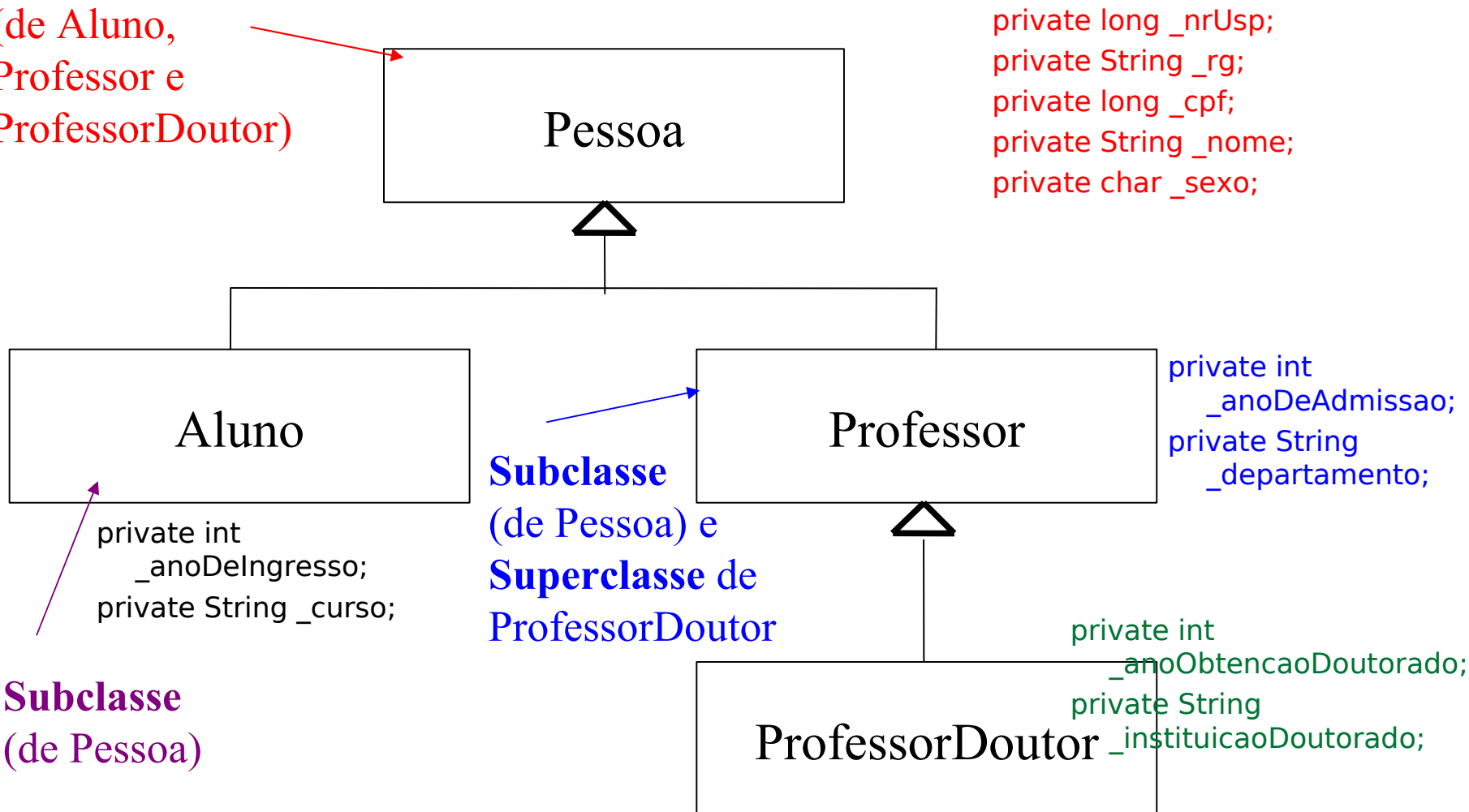
**Superclasse**  
(de Aluno,  
Professor e  
ProfessorDoutor)



**Subclasse**  
(de Pessoa)

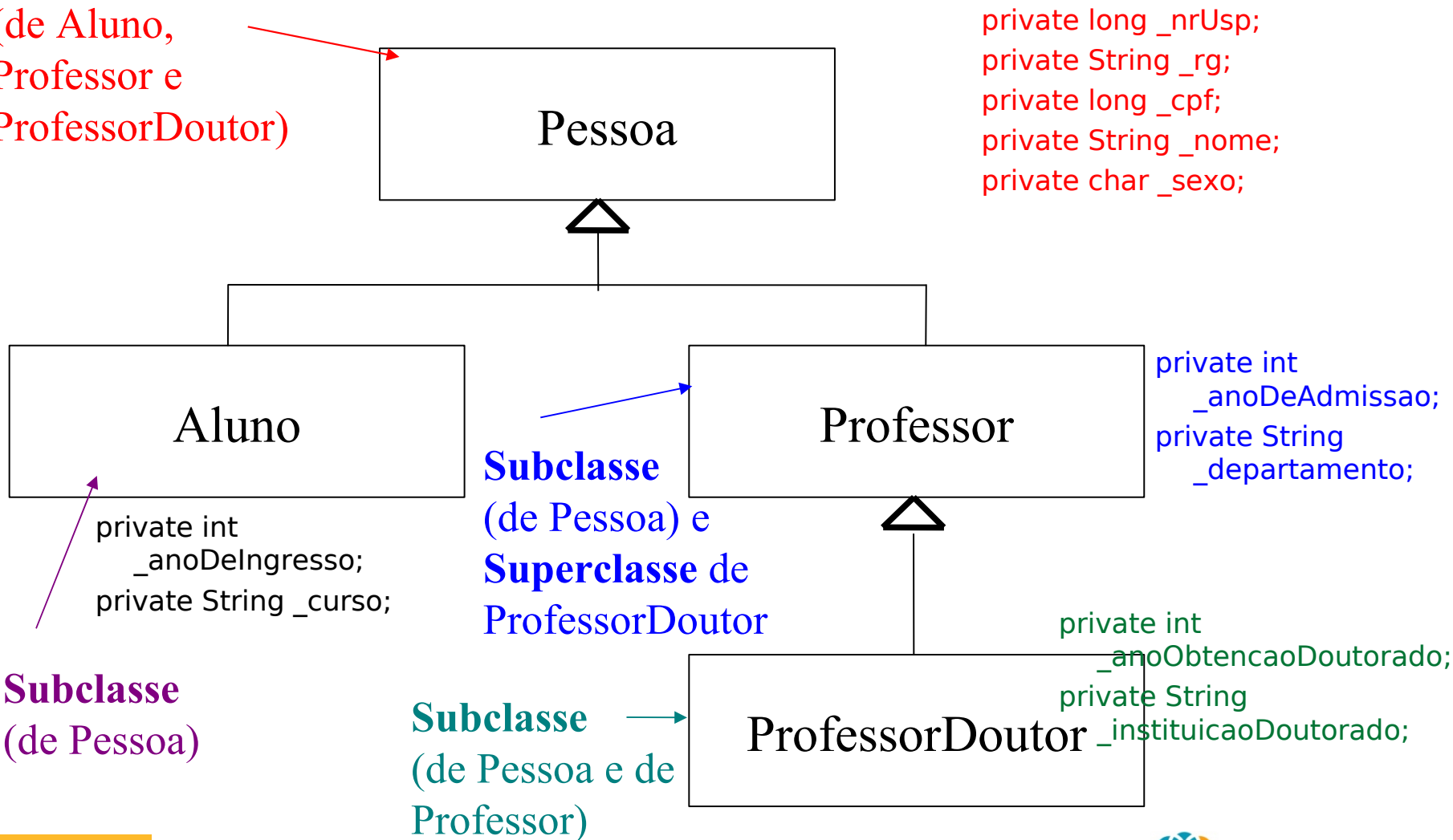
# Herança

**Superclasse**  
(de Aluno,  
Professor e  
ProfessorDoutor)



# Herança

**Superclasse**  
(de Aluno,  
Professor e  
ProfessorDoutor)



# Implementação de herança em Java



```
class Pessoa
{
    private long _nrUsp;
    private String _rg;
    private long _cpf;
    private String _nome;
    private char _sexo;
    // métodos de acesso (public) a esses 5 atributos
    (...)
    public void imprimeDados(){
        System.out.println("Nome: " + _nome);
        System.out.println("Nr USP: " + _nrUsp);
        System.out.println("CPF: " + _cpf);
        System.out.println("RG: " + _rg);
        System.out.println("Sexo: " + _sexo);
    }
}
```

## VERSÃO ANTIGA

```
class Aluno
{
    private long _nrUsp;
    private String _rg;
    private long _cpf;
    private String _nome;
    private char _sexo;
    private int _anoDeIngresso;
    private String _curso;
    // métodos de acesso (public)
    (...)
    public void imprimeDados(){
        System.out.println("Nome: " + _nome);
        System.out.println("Nr USP: " + _nrUsp);
        System.out.println("CPF: " + _cpf);
        System.out.println("RG: " + _rg);
        System.out.println("Sexo: " + _sexo);
        System.out.println("Ano de ingresso: " +
            _anoDeIngresso);
        System.out.println("Curso: " + _curso);
    }
}
```

## VERSÃO COM HERANÇA

```
class Aluno extends Pessoa
{
    private int _anoDeIngresso;
    private String _curso;
    // 2 pares de métodos de acesso
    (public)
    (...)
    public void imprimeDados(){
        ?
        System.out.println("Ano de
            ingresso: " +
                _anoDeIngresso);
        System.out.println("Curso: " +
            _curso);
    }
}
```



## VERSÃO ANTIGA

```
class Aluno
{
    private long _nrUsp;
    private String _rg;
    private long _cpf;
    private String _nome;
    private char _sexo;
    private int _anoDeIngresso;
    private String _curso;
    // 7 pares de métodos de acesso (public)
    (...)
    public void imprimeDados(){
        System.out.println("Nome: " + _nome);
        System.out.println("Nr USP: " + _nrUsp);
        System.out.println("CPF: " + _cpf);
        System.out.println("RG: " + _rg);
        System.out.println("Sexo: " + _sexo);
        System.out.println("Ano de ingresso: " +
            _anoDeIngresso);
        System.out.println("Curso: " + _curso);
    }
}
```

## VERSÃO COM HERANÇA

```
class Aluno extends Pessoa
{
    private int _anoDeIngresso;
    private String _curso;
    // 2 pares de métodos de acesso (public)
    (...)
    public void imprimeDados(){
        super.imprimeDados();
        System.out.println("Ano de ingresso:
            " +
                _anoDeIngresso);
        System.out.println("Curso: " +
            _curso);
    }
}
```

## REDEFINIÇÃO DE MÉTODOS

Modifica ou estende um método da superclasse



## VERSÃO ANTIGA

```
class Professor
{
    private long _nrUsp;
    private String _rg;
    private long _cpf;
    private String _nome;
    private char _sexo;
    private int _anoDeAdmissao;
    private String _departamento;
    // 7 pares de métodos de acesso (public)
    (...)
    public void imprimeDados(){
        System.out.println("Nome: " + _nome);
        System.out.println("Nr USP: " + _nrUsp);
        System.out.println("CPF: " + _cpf);
        System.out.println("RG: " + _rg);
        System.out.println("Sexo: " + _sexo);
        System.out.println("Ano de admissão: " +
            _anoDeAdmissao);
        System.out.println("Departamento: " +
            _departamento);
    }
}
```



## VERSÃO COM HERANÇA



## VERSÃO ANTIGA

```
class Professor
{
    private long _nrUsp;
    private String _rg;
    private long _cpf;
    private String _nome;
    private char _sexo;
    private int _anoDeAdmissao;
    private String _departamento;
    // 7 pares de métodos de acesso (public)
    (...)
    public void imprimeDados(){
        System.out.println("Nome: " + _nome);
        System.out.println("Nr USP: " + _nrUsp);
        System.out.println("CPF: " + _cpf);
        System.out.println("RG: " + _rg);
        System.out.println("Sexo: " + _sexo);
        System.out.println("Ano de admissão: " +
            _anoDeAdmissao);
        System.out.println("Departamento: " +
            _departamento);
    }
}
```



## VERSÃO COM HERANÇA

```
class Professor extends Pessoa
{
    private int _anoDeAdmissao;
    private String _departamento;
    // 2 pares de métodos de acesso (public)
    (...)
    public void imprimeDados(){
        super.imprimeDados();
        System.out.println("Ano de
            admissão: " + _anoDeAdmissao);
        System.out.println("Departamento: "
            + _departamento);
    }
}
```

## VERSÃO ANTIGA

```
class ProfessorDoutor {  
    private long _nrUsp;  
    private String _rg;  
    private long _cpf;  
    private String _nome;  
    private char _sexo;  
    private int _anoDeAdmissao;  
    private String _departamento;  
    private int _anoObtencaoDoutorado;  
    private String _instituicaoDoutorado;  
    // 9 pares de métodos de acesso (public)  
    (...)  
    public void imprimeDados(){  
        System.out.println("Nome: " + _nome);  
        System.out.println("Nr USP: " + _nrUsp);  
        System.out.println("CPF: " + _cpf);  
        System.out.println("RG: " + _rg);  
        System.out.println("Sexo: " + _sexo);  
        System.out.println("Ano de admissão: " + _anoDeAdmissao);  
        System.out.println("Departamento: " + _departamento);  
        System.out.println("Ano de obtenção do título de Doutor: " +  
            _anoObtencaoDoutorado);  
        System.out.println("Instituição do Doutorado: " +  
            _instituicaoDoutorado);  
    }  
}
```



## VERSÃO COM HERANÇA

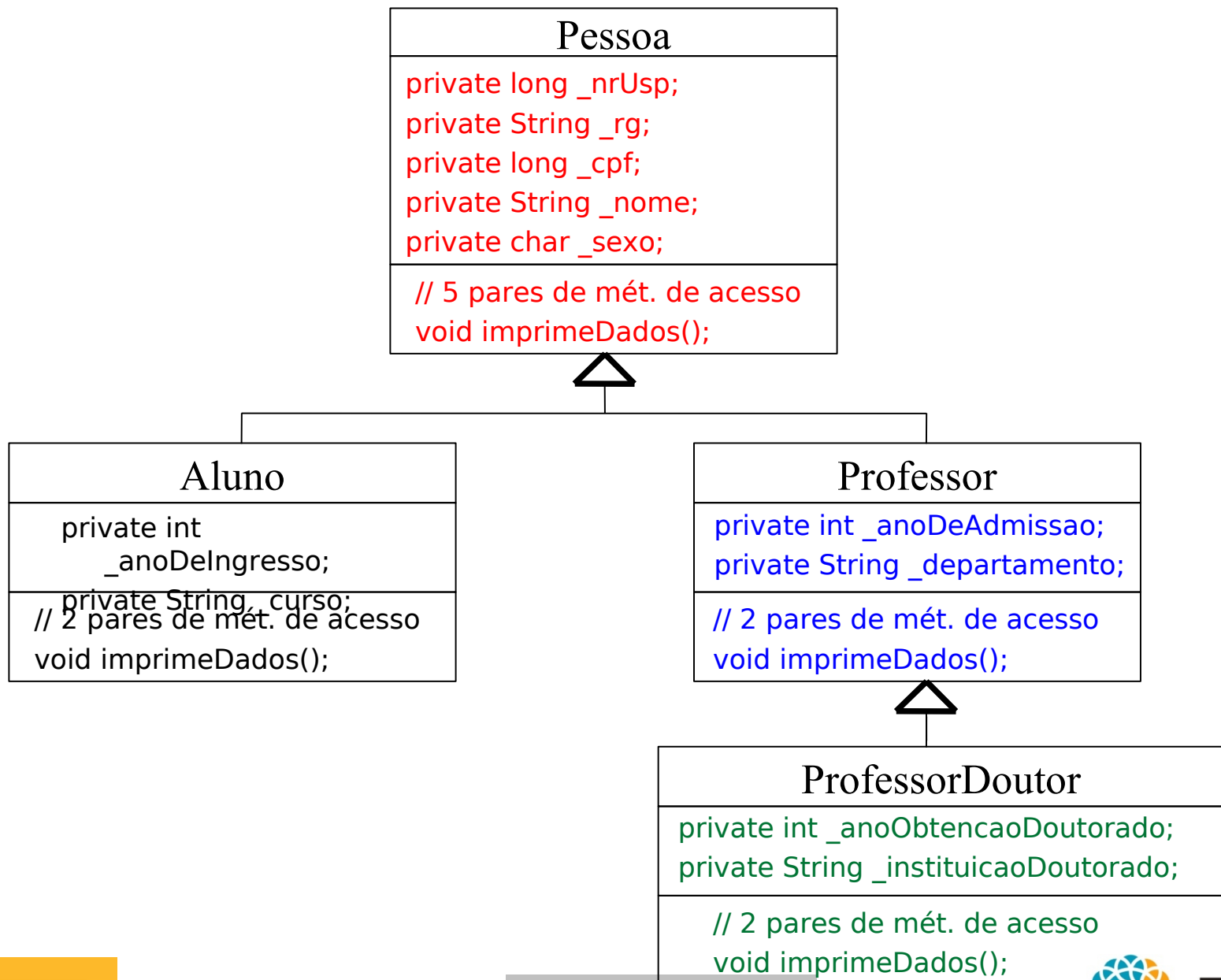
## VERSÃO ANTIGA

```
class ProfessorDoutor {  
    private long _nrUsp;  
    private String _rg;  
    private long _cpf;  
    private String _nome;  
    private char _sexo;  
    private int _anoDeAdmissao;  
    private String _departamento;  
    private int _anoObtencaoDoutorado;  
    private String _instituicaoDoutorado;  
    // 9 pares de métodos de acesso (public)  
    (...)   
    public void imprimeDados(){  
        System.out.println("Nome: " + _nome);  
        System.out.println("Nr USP: " + _nrUsp);  
        System.out.println("CPF: " + _cpf);  
        System.out.println("RG: " + _rg);  
        System.out.println("Sexo: " + _sexo);  
        System.out.println("Ano de admissão: " +  
            _anoDeAdmissao);  
        System.out.println("Departamento: " +  
            _departamento);  
        System.out.println("Ano de obtenção do  
            título de Doutor: " +  
            _anoObtencaoDoutorado);  
        System.out.println("Instituição do  
            Doutorado: " + _instituicaoDoutorado);  
    }  
}
```

## VERSÃO COM HERANÇA

```
class ProfessorDoutor extends Professor  
{  
    private int _anoObtencaoDoutorado;  
    private String _instituicaoDoutorado;  
    // 2 pares de métodos de acesso (public)  
    (...)   
    public void imprimeDados(){  
        super.imprimeDados();  
        System.out.println("Ano de obtenção d  
            título de Doutor: " +  
            _anoObtencaoDoutorado);  
        System.out.println("Instituição do  
            Doutorado: " +  
            _instituicaoDoutorado);  
    }  
}
```

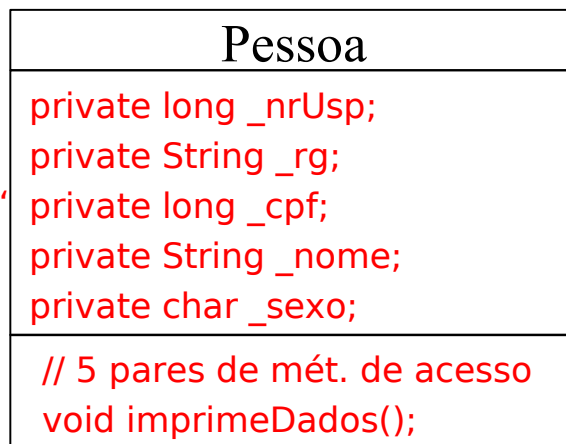




```

public void imprimeDados(){
    System.out.println("Nome: "
        + _nome);
    System.out.println("Nr USP: "
        + _nrUsp);
    System.out.println("CPF: " +
        _cpf);
    System.out.println("RG: " +
        _rg);
    System.out.println("Sexo: " +
        _sexo);
}

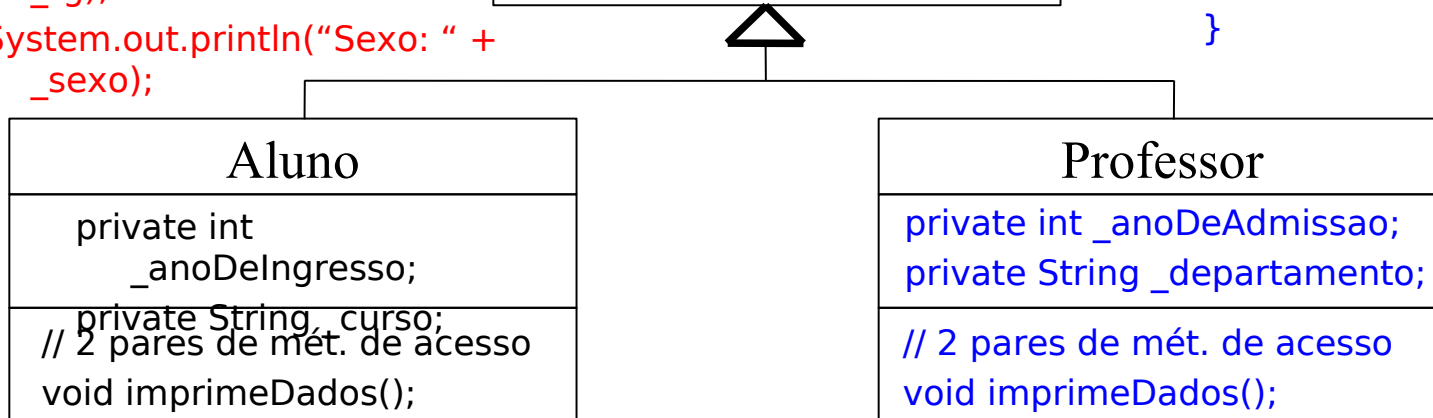
```



```

public void imprimeDados(){
    super.imprimeDados();
    System.out.println("Ano
    de admissão: " +
        _anoDeAdmissao);
    System.out.println("Depa
    rtamento: " +
        _departamento);
}

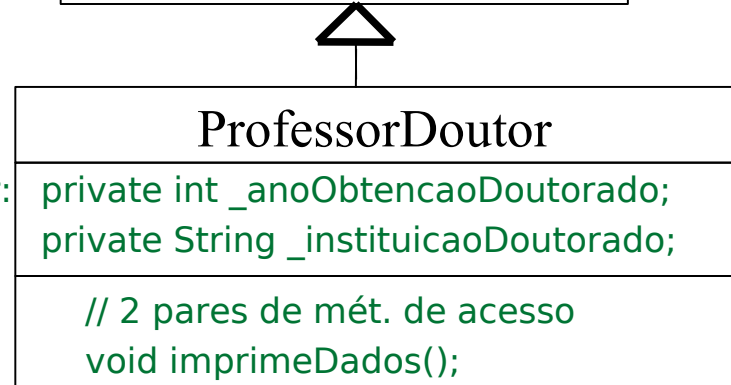
```



```

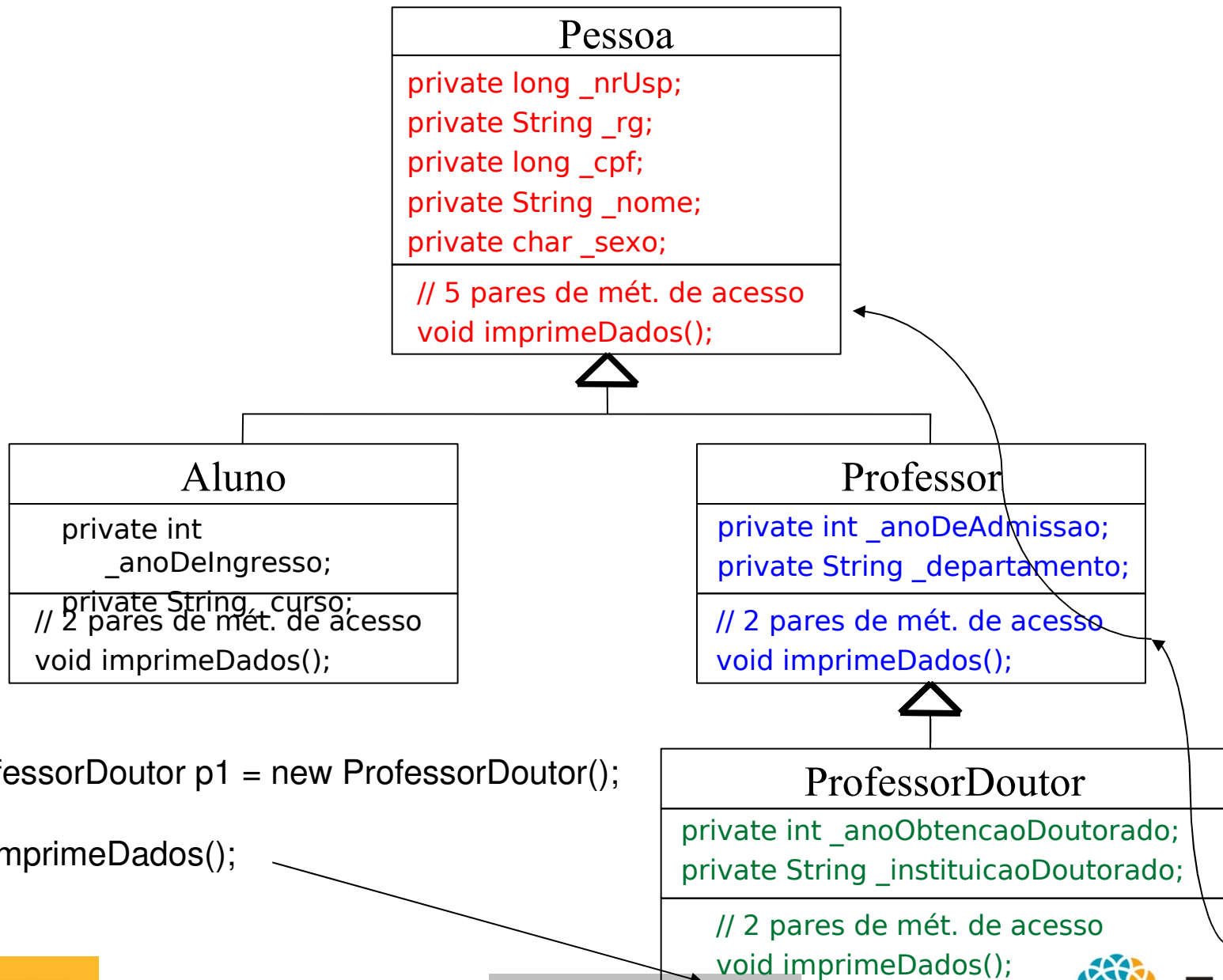
public void imprimeDados(){
    super.imprimeDados();
    System.out.println("Ano de
    obtenção do título de Doutor:
    " +
        _anoObtencaoDoutorado);
    System.out.println("Instituição
    do Doutorado: " +
        _instituicaoDoutorado);
}

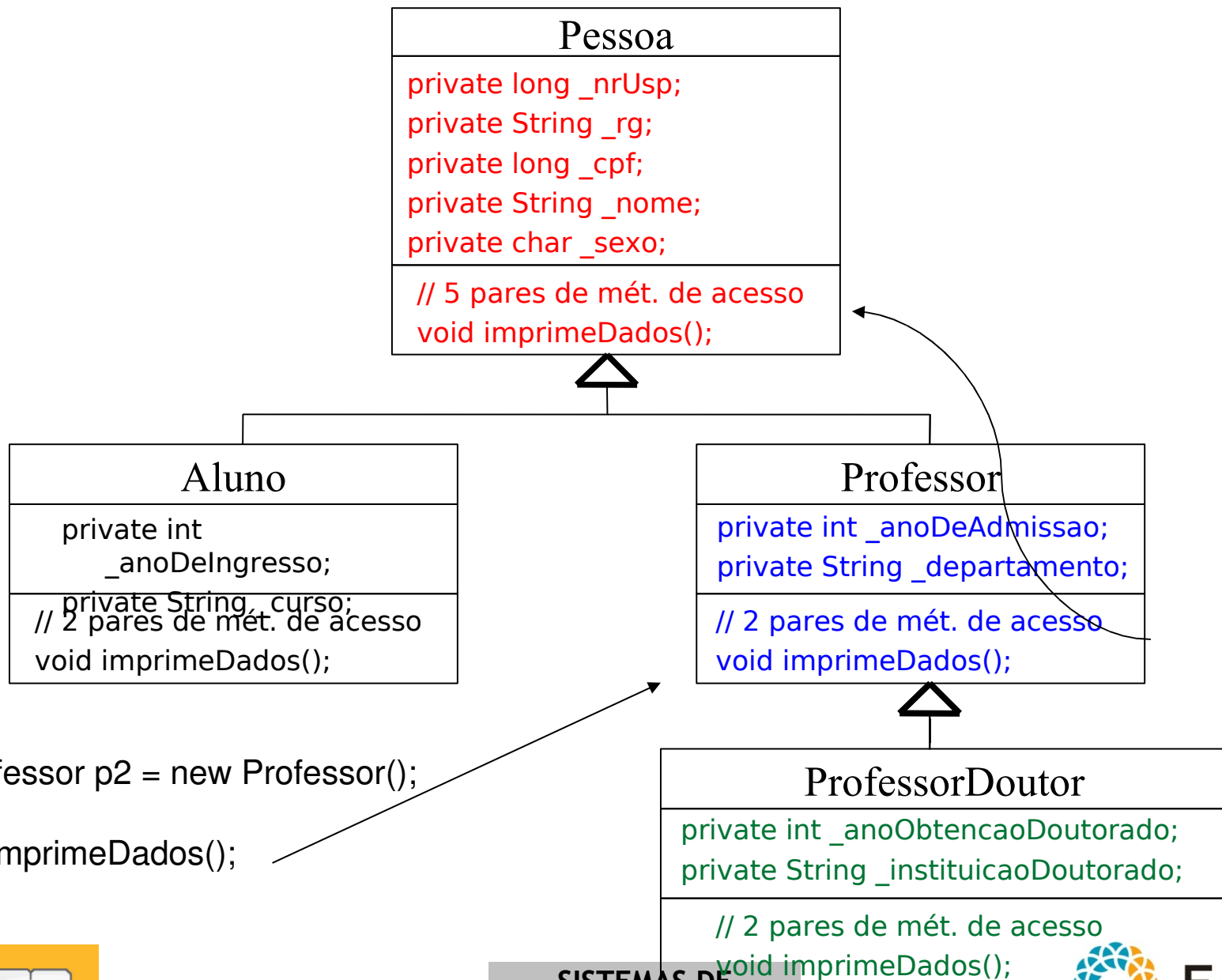
```



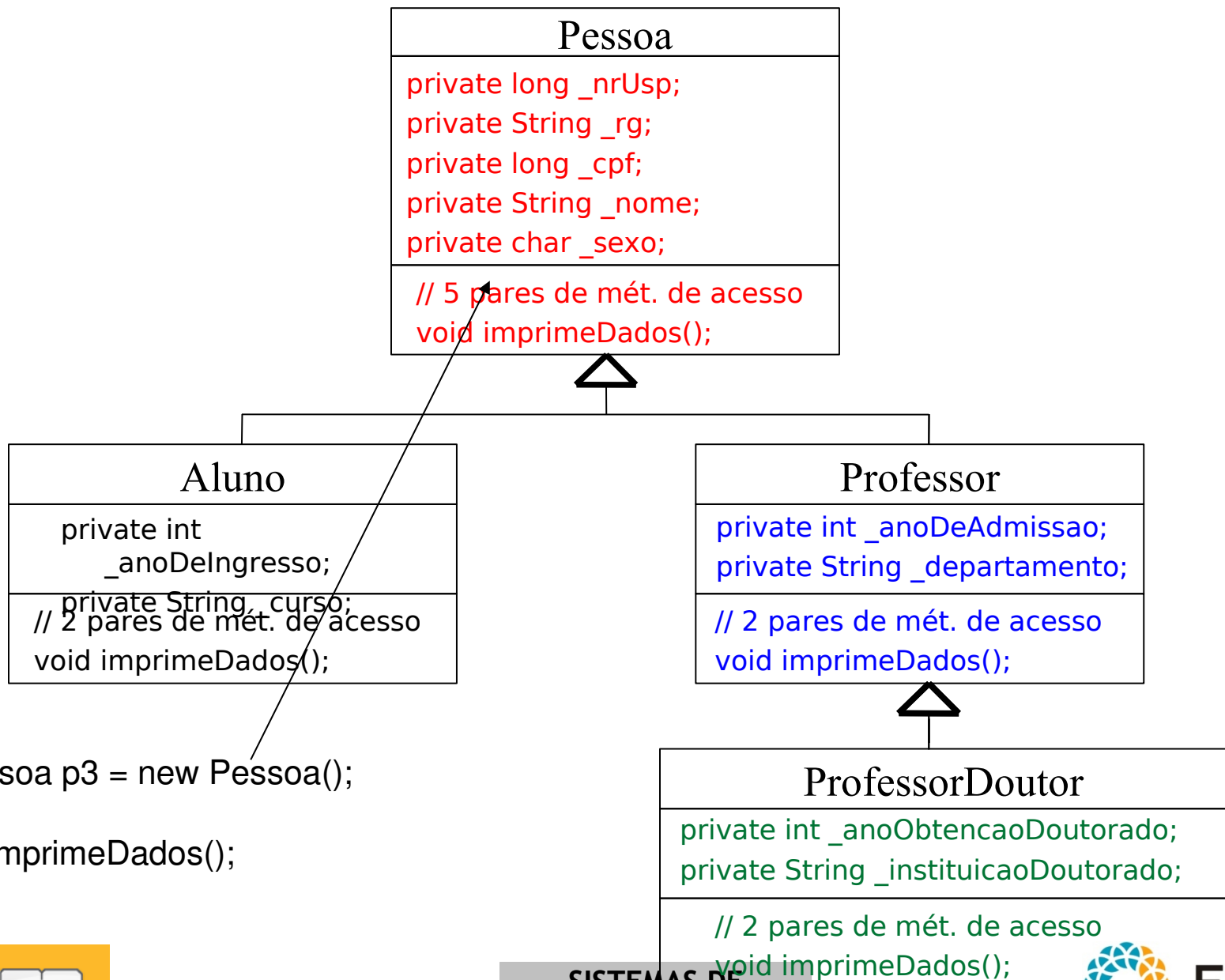
}

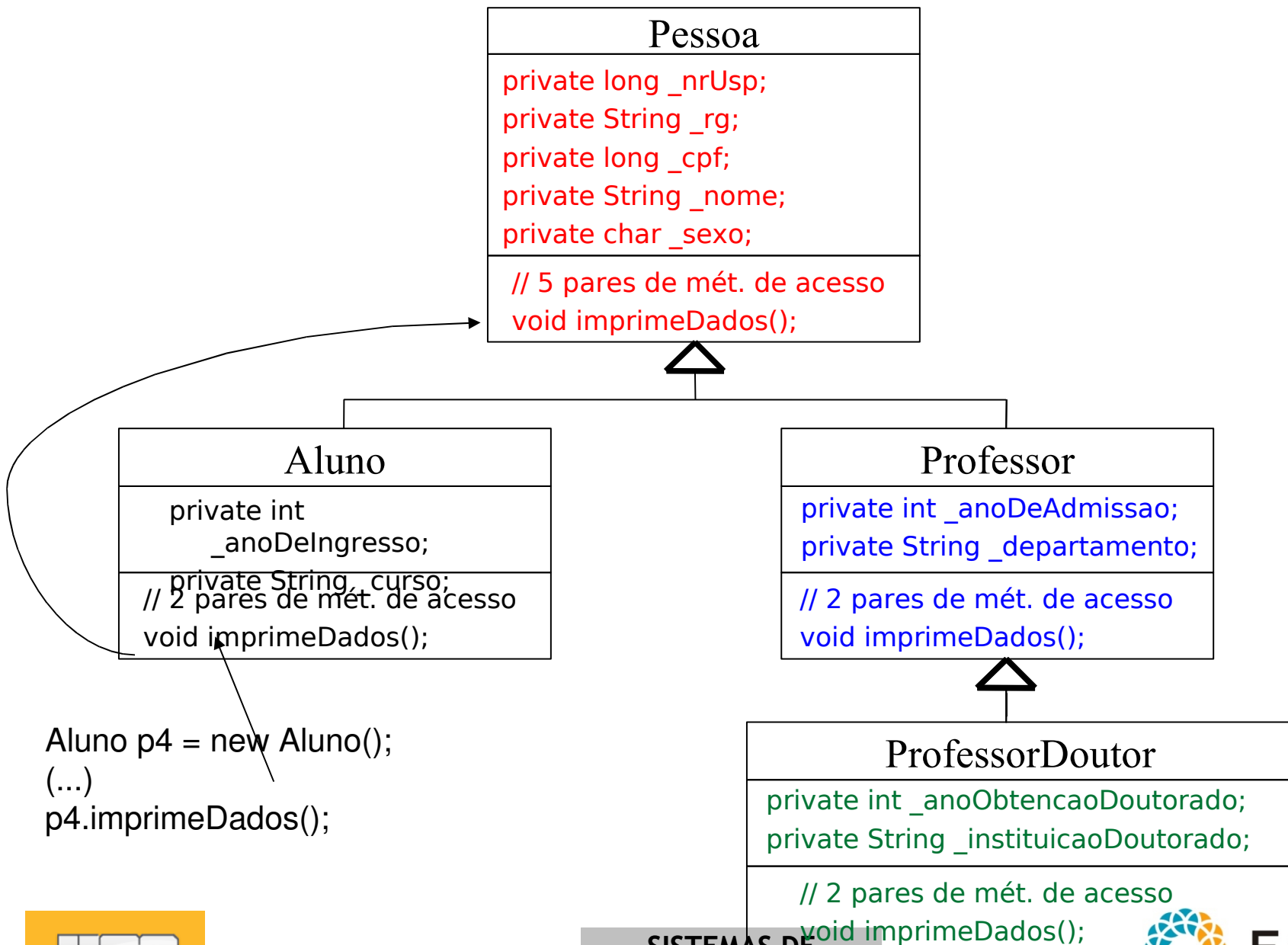






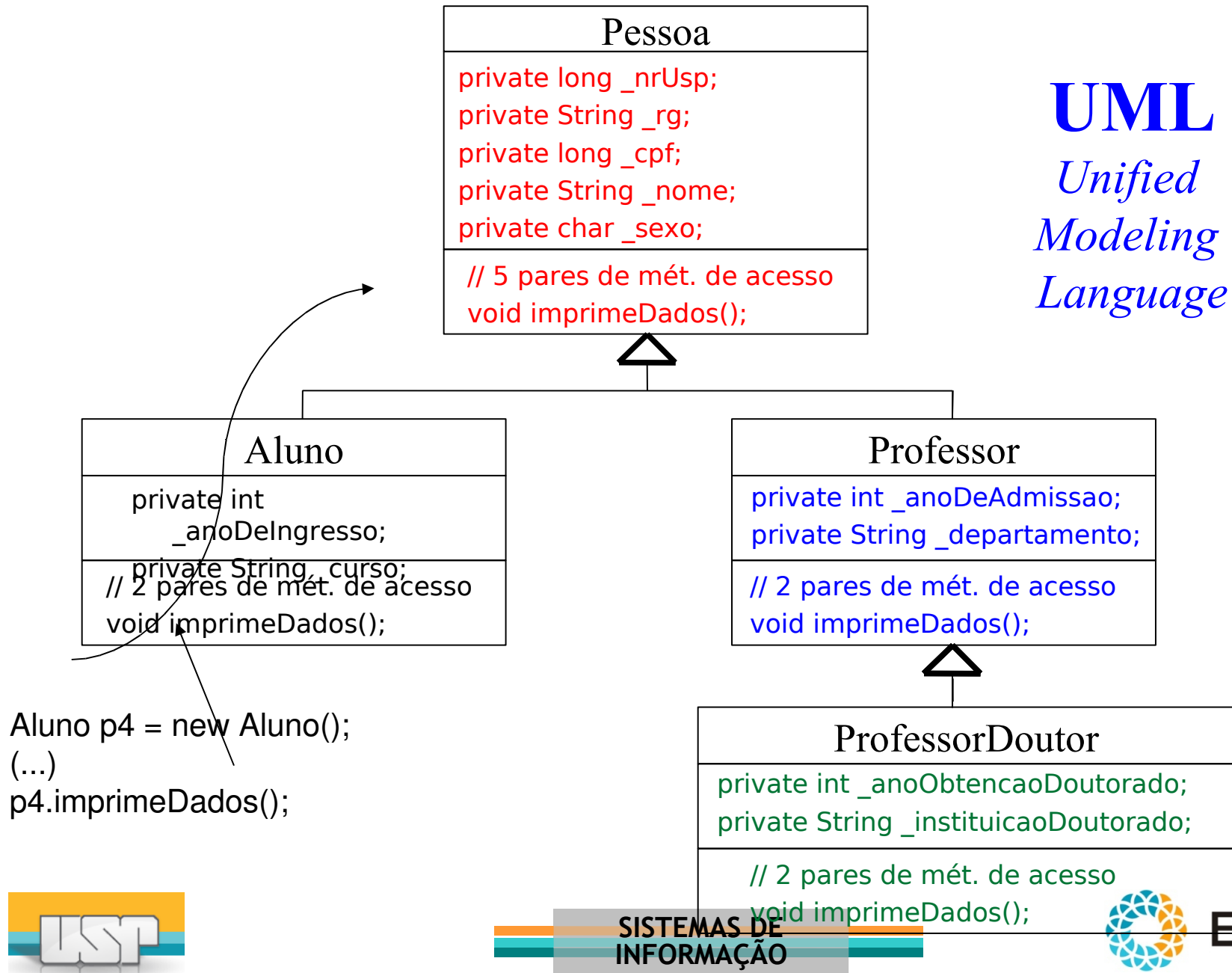






# UML

*Unified  
Modeling  
Language*



# Observações

- As subclasses podem acrescentar novos métodos:
  - Ex: métodos de acesso aos atributos específicos da subclasse
  - Ex: novas funcionalidades típicas daquela subclasse
  - Ex: sobrecarregar métodos da superclasse (mesmo nome, nova assinatura)
- As subclasses podem redefinir métodos da superclasse com a mesma assinatura
  - Ex: `imprimeDados`
  - Não precisa necessariamente chamar o método da superclasse

# Exemplo

```
class Pessoa{
    (...)
    void quemEVoce() { System.out.println("Sou uma pessoa"); }
}
class Aluno extends Pessoa{
    (...)
    void quemEVoce() { System.out.println("Sou um aluno"); }
}
class Professor extends Pessoa{
    (...)
    void quemEVoce() { System.out.println("Sou um professor"); }
}
class ProfessorDoutor extends Professor{
    (...)
    void quemEVoce() { System.out.println("Sou um professor doutor"); }
}
```

# Exemplo

```
class Pessoa{
    (...)
    void quemEVoce() { System.out.println("Sou uma pessoa"); }
}
class Aluno extends Pessoa{
    (...)
    void quemEVoce() { System.out.println("Sou um aluno"); }
}
class Professor extends Pessoa{
    (...)
    void quemEVoce() { System.out.println("Sou um professor"); }
}
class ProfessorDoutor extends Professor{
    (...)
    void quemEVoce() { System.out.println("Sou um professor doutor"); }
}
```

```
Pessoa p1 = new Pessoa(); p1.quemEVoce();
Aluno p2 = new Aluno(); p2.quemEVoce();
Professor p3 = new Professor(); p3.quemEVoce();
ProfessorDoutor p4 = new ProfessorDoutor(); p4.quemEVoce();
```

# Exemplo

```
class Pessoa{
    (...)
    void quemEVoce() { System.out.println("Sou uma pessoa"); }
}
class Aluno extends Pessoa{
    (...)
    void quemEVoce() { System.out.println("Sou um aluno"); }
}
class Professor extends Pessoa{
    (...)
    void quemEVoce() { System.out.println("Sou um professor"); }
}
class ProfessorDoutor extends Professor{
    (...)
    void quemEVoce() { System.out.println("Sou um professor doutor"); }
}
```

```
Pessoa p1 = new Pessoa(); p1.quemEVoce();           // Sou uma pessoa
Aluno p2 = new Aluno(); p2.quemEVoce();             // Sou um aluno
Professor p3 = new Professor(); p3.quemEVoce();      // Sou um professor
ProfessorDoutor p4 = new ProfessorDoutor(); p4.quemEVoce(); // Sou um professor doutor
```

# Observações

- A subclasse pode sobrecarregar/sobrepôr métodos de mesmo nome com diferentes parâmetros
- Ex:
  - Classe Pessoa tem o método
    - void alteraDadosUspianos(long nrUsp);
  - Classe Aluno tem o método
    - void alteraDadosUspianos(long nrUsp, String curso);
  - Os dois métodos podem ser usados no filho!



```
import java.lang.String;
```

```
public class Pessoa {  
    long nusp;  
    String x;  
  
    void altera(long nusp) {  
        this.nusp = nusp;  
    }  
}
```

```
import java.lang.String;
```

```
public class Aluno extends Pessoa {  
    void altera(long nusp, String x) {  
        this.nusp = nusp;  
        this.x = x;  
    }  
}
```

```
public static void main(String[] args) {  
    Aluno a = new Aluno();  
    System.out.println(a.nusp);  
    System.out.println(a.x);  
    a.altera(12);  
    System.out.println(a.nusp);  
    System.out.println(a.x);  
    a.altera(10,"ei");  
    System.out.println(a.nusp);  
    System.out.println(a.x);  
}
```

```
import java.lang.String;
```

```
public class Aluno extends Pessoa{  
    void altera(long nusp, String x) {  
        this.nusp = nusp;  
        this.x = x;  
    }  
}
```

```
public static void main(String[] args) {  
    Aluno a = new Aluno();  
    System.out.println(a.nusp);  
    System.out.println(a.x);  
    a.altera(12);  
    System.out.println(a.nusp);  
    System.out.println(a.x);  
    a.altera(10,"ei");  
    System.out.println(a.nusp);  
    System.out.println(a.x);  
}
```

```
import java.lang.String;
```

```
public class Pessoa {  
    long nusp;  
    String x;
```

```
    void altera(long nusp) {  
        this.nusp = nusp;  
    }  
}
```

Saída

0  
null  
12  
null  
10  
ei



# Construtores

```
class Pessoa
{
    Pessoa(long nrUsp, String rg, long cpf, String nome, char sexo)
    {
        _nrUsp = nrUsp;
        _rg = rg;
        _cpf = cpf;
        _nome = nome;
        _sexo = sexo;
    }
    (...)
}

class Aluno extends Pessoa{
    Aluno(long nrUsp, String rg, long cpf, String nome, char sexo, int anoIngresso,
        String curso)
    {
        ?

    }
}
```



# Construtores

```
class Pessoa
{
    Pessoa(long nrUsp, String rg, long cpf, String nome, char sexo)
    {
        _nrUsp = nrUsp;
        _rg = rg;
        _cpf = cpf;
        _nome = nome;
        _sexo = sexo;
    }
    (...)
}

class Aluno extends Pessoa{
    Aluno(long nrUsp, String rg, long cpf, String nome, char sexo, int anoIngresso,
        String curso)
    {
        super(nrUsp, rg, cpf, nome, sexo);
        _anoDeIngresso = anoIngresso;
        _curso = curso;
    }
}
```

Deve necessariamente ser o primeiro comando.



# Construtores

```
class Professor extends Pessoa{  
    Professor(long nrUsp, String rg, long cpf, String nome, char sexo, int  
        anoAdmissao, String dpto)  
    {  
        super(nrUsp, rg, cpf, nome, sexo);  
        _anoDeAdmissao = anoAdmissao;  
        _departamento = dpto;  
    }  
}
```

```
class ProfessorDoutor extends Professor{  
    ProfessorDoutor(long nrUsp, String rg, long cpf, String nome, char sexo, int  
        anoAdmissao, String dpto, int anoDoutorado, String instituicaoDoutorado)  
    {  
        ?  
    }  
}
```

# Construtores

```
class Professor extends Pessoa{
    Professor(long nrUsp, String rg, long cpf, String nome, char sexo, int
        anoAdmissao, String dpto)
    {
        super(nrUsp, rg, cpf, nome, sexo);
        _anoDeAdmissao = anoAdmissao;
        _departamento = dpto;
    }
}
```

```
class ProfessorDoutor extends Professor{
    ProfessorDoutor(long nrUsp, String rg, long cpf, String nome, char sexo, int
        anoAdmissao, String dpto, int anoDoutorado, String instituicaoDoutorado)
    {
        super(nrUsp, rg, cpf, nome, sexo, anoAdmissao, dpto);
        _anoObtencaoDoutorado = anoDoutorado;
        _instituicaoDoutorado = instituicaoDoutorado;
    }
}
```

# Vantagens de Herança

- Modularização (programa separado em módulos estáveis)
- Reuso
  - Como no caso de Pessoa
  - APIs Java (você pode expandir!!!)
- Tipagem (relação transitiva)
  - Aluno É uma Pessoa
  - Professor É uma Pessoa
  - ProfessorDoutor É um Professor, e também É uma Pessoa
  - → permite agrupamentos

```

class ComunidadeAcademica
{
    Pessoa [] comunidade = new Pessoa [5000];
    int nrPessoas = 0;
    void inserePessoa(Pessoa p)
    {
        comunidade[nrPessoas] = p;
        nrPessoas++;
    }
    Pessoa buscaPessoa (int nrUsp)
    {
        int i = 0;
        while (i < nrPessoas){
            if (comunidade[i].obtemNrUsp() == nrUsp)
                return comunidade[i];
            i++;
        }
        System.out.println("Pessoa não encontrada");
        return null;
    }
}

```

```

ComunidadeAcademica usp = new
    ComunidadeAcademica();

Aluno p1 = new Aluno(55, ... , "SI");
usp.inserePessoa(p1);
Professor p2 = new Professor(51, ...,
    "EACH");
usp.inserePessoa(p2);
ProfessorDoutor p3 = new
    ProfessorDoutor(67, ....., "USP");
usp.inserePessoa(p3);

usp.buscaPessoa(51).quemEVoce();

```





```

class ComunidadeAcademica
{
    Pessoa [] comunidade = new Pessoa [5000];
    int nrPessoas = 0;
    void inserePessoa(Pessoa p)
    {
        comunidade[nrPessoas] = p;
        nrPessoas++;
    }
    Pessoa buscaPessoa (int nrUsp)
    {
        int i = 0;
        while (i < nrPessoas){
            if (comunidade[i].obtemNrUsp() == nrUsp)
                return comunidade[i];
            i++;
        }
        System.out.println("Pessoa não encontrada");
        return null;
    }
    void imprimeDados(){
        for (int i=0; i<nrPessoas; i++)
            comunidade[i].imprimeDados();
    }
}

```

```

ComunidadeAcademica usp = new
    ComunidadeAcademica();

Aluno p1 = new Aluno(55, ... , "SI");
usp.inserePessoa(p1);
Professor p2 = new Professor(51, ...,
    "EACH");
usp.inserePessoa(p2);
ProfessorDoutor p3 = new
    ProfessorDoutor(67, ....., "USP");
usp.inserePessoa(p3);

usp.buscaPessoa(51).quemEVoce();
usp.imprimeDados();

```



# Hierarquia de Classes

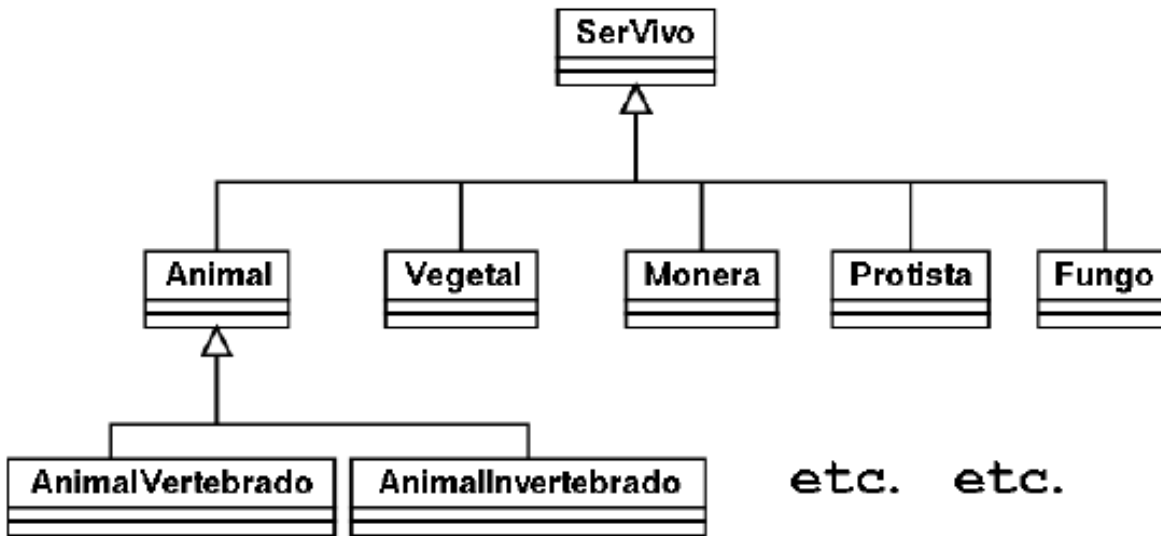
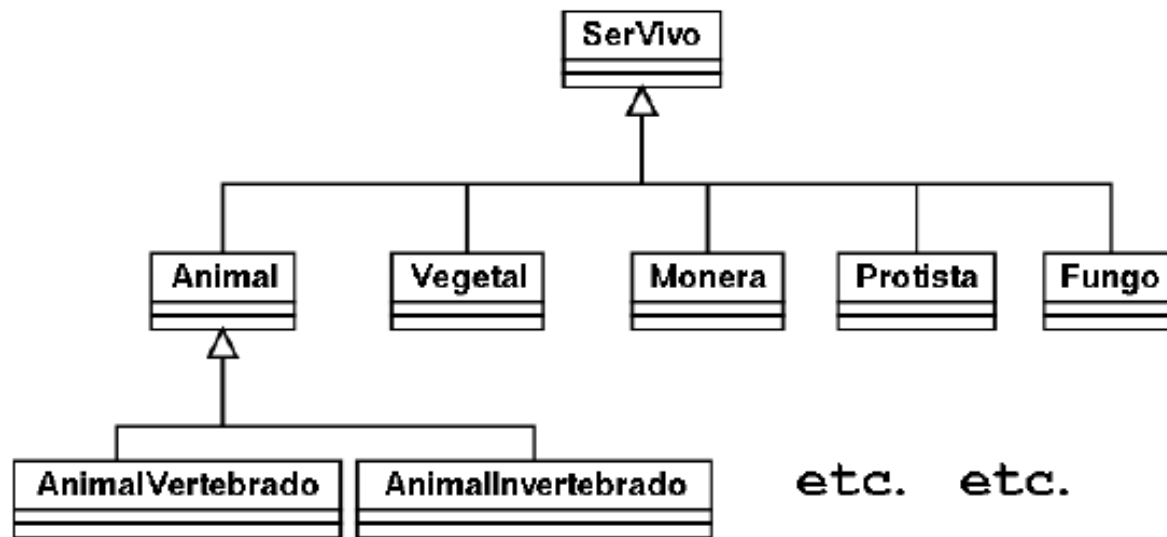


Figura 21.2: Hierarquia de classes representando os seres vivos

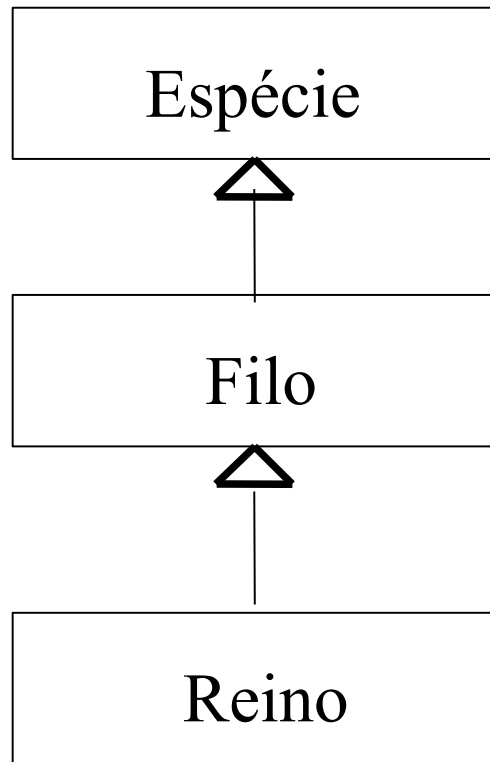
# Hierarquia de Classes



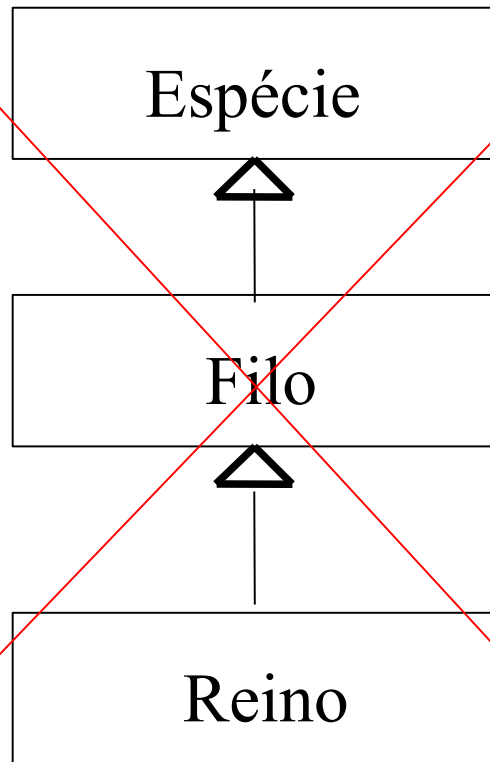
É  
um tipo  
de

Figura 21.2: Hierarquia de classes representando os seres vivos

# Hierarquia



# Hierarquia **ERRADA**



Não há a relação  
“É um tipo de”

# Especificadores de acesso

- **final** class X:

# Especificadores de acesso

- **final** class X: X não pode ser estendida

# Especificadores de acesso

- **final** class X: X não pode ser estendida
- **final** em um método:



# Especificadores de acesso

- **final** class X: X não pode ser estendida
- **final** em um método: o método não pode ser redefinido nas subclasses

# Especificadores de acesso

- **final** class X: X não pode ser estendida
- **final** em um método: o método não pode ser redefinidos nas subclasses
- class X: X pode ser vista só dentro do pacote
- **public** class X: X pode ser vista por todos

```
public class ExMatematica2 {  
    public final static double pi = 3.141592;  
  
    public final static double quadrado(double x){  
        return x*x;  
    }  
  
    public static double cubo(double x){  
        return x*x*x;  
    }  
  
    public final static double perimetro(double raio){  
        return 2*pi*raio;  
    }  
}
```

```
public class ExMatematica2Sub extends ExMatematica2{  
    public final static double pi = 3.1415926; ← ?????
```

```
    /* O metodo abaixo nao pode ser redefinido pois na classe original  
    * (ExMatematica2) ele era final. */
```

```
    /*  
    public final static double quadrado(double x){  
        return x*x;  
    }  
    */
```

```
    /* O metodo cubo nao era final na super-classe (ExMatematica2) e, por isso,  
    * pode ser redefinida.  
    */
```

```
    public static double cubo(double x){  
        double temp = x*x;  
        return temp*x;  
    }  
}
```

```
public class ExMatematica2Sub extends ExMatematica2{  
    public final static double pi = 3.1415926; ←
```

É constante  
apenas dessa  
classe

```
    /* O metodo abaixo nao pode ser redefinido pois na classe  
     * original (ExMatematica2) ele era final. */
```

```
    /*  
    public final static double quadrado(double x){  
        return x*x;  
    }  
    */
```

```
    /* O metodo cubo nao era final na super-classe (ExMatematica2) e, por isso,  
     * pode ser redefinida.  
     */
```

```
    public static double cubo(double x){  
        double temp = x*x;  
        return temp*x;  
    }  
}
```

# Especificadores de acesso

Acesso	public	protected	sem nada	private
Classe	Sim	Sim	Sim	Sim
Pacote	Sim	Sim	Sim	Não
Subclasse	Sim	Sim	Não	Não
Mundo	Sim	Não	Não	Não

# Especificadores de acesso

Acesso	public	protected	sem nada	private
Classe	Sim	Sim		
Pacote	Sim	Sim		
Subclasse	Sim	Sim		
Mundo	Sim	Não	Não	Não

**protected**

indica que o código pode ser acessado somente dentro do pacote ou por uma subclasse de sua classe em outro pacote

# Abstract

- **abstract** class X:
  - X não pode ser instanciada
  - Não podemos fazer new X();
- **abstract** int metodo():
  - O método é deixado sem código na classe
  - Cabe aos filhos implementar
- Todo método abstract torna a classe abstract
- Nem toda classe abstract possui métodos abstract
  - Basta que não desejemos que seja instanciada



```
class Pessoa {  
    String nome;  
    int RG;  
    Pessoa(String nome, int RG){  
        this.nome = nome;  
        this.RG = RG;  
    }  
  
    void respirar(){  
        System.out.println("Pessoa respirando.");  
    }  
  
    void dormir(){  
        System.out.println("Pessoa dormindo.");  
    }  
}
```

```

abstract class Aluno extends Pessoa {
    String nome;
    int RG;
    int numeroUSP;

    Aluno(String n, int rg, int nUSP){
        super(n,rg);
        numeroUSP = nUSP;
    }

    void respirar(){
        System.out.println("Aluno respirando.");
    }

    final void fazerTrabalhos(){
        System.out.println("Trabalhando.");
    }

    abstract void estudar();
}

```

Método abstratos precisam ser implementado nas sub-classes não abstratas.

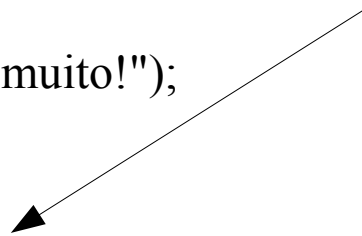
Um método não pode ser abstrato e estático ao mesmo tempo

```
public class AlunoRegular extends Aluno{
    AlunoRegular(String nome, int RG, int NUSP){
        super(nome, RG, NUSP);
    }

    void estudarMuito(){
        System.out.println("Estudando muito!");
    }

    void estudar(){
        System.out.println("Aluno Regular Estudando");
    }
}
```

PRECISA implementar todos  
os métodos abstratos da  
super-classe



```

public final class AlunoEspecial extends Aluno{
    AlunoEspecial(String nome, int RG, int NUSP){
        super(nome,RG,NUSP);
    }

    void respirar(){
        super.respirar();
        System.out.println("Aluno especial respirando.");
    }

    /* void fazerTrabalhos(){
        ...
    }
    */

    void estudar(){
        System.out.println("Aluno Especial Estudando");
    }
}

```

NAO é possível usar  
`super.super.respirar`  
 (super.super não é permitido)

Este metodo não pode ser  
 implementado aqui por ser  
 final na super-classe (Aluno)

PRECISA implementar  
 todos os métodos abstratos  
 da super-classe

# Observações

- Aluno foi escolhida como abstrata pelo fato de, neste sistema, existirem apenas alunos especiais e regulares.
  - Não DEVERIA ser possível criar um aluno genérico
  - Por isso usamos abstract

# Finalmente

- Classes podem implementar mais de uma interface
- Classes só podem estender uma única classe
  - Não existe herança múltipla em java
- Construtores não podem ser final

# Construtores e Herança

```
public class teste {}
```

```
public class teste2 extends teste {  
    public static void main(String[] args) {  
        teste t = new teste2();  
    }  
}
```

Quando criamos um objeto de uma classe que não tem construtor, o Java usará um construtor padrão (sem parâmetros).

Esse construtor, por sua vez, chamará, automaticamente, o construtor padrão da classe-pai.

E assim por diante

# Construtores e Herança

```
public class teste {  
    teste(int x){}  
}
```

```
public class teste2 extends teste {  
    public static void main(String[] args) {  
        teste t = new teste2();  
    }  
}
```

Quando criamos um construtor não padrão em uma classe, o construtor padrão deixa de existir.

E qualquer subclasse que precise dele vai gerar erro de compilação



# Construtores e Herança

```
public class teste {  
    teste(int x){}  
}
```

Infelizmente, toda subclasse que não chama nenhum construtor do pai, implicitamente chama o padrão.

```
public class teste2 extends teste {  
    teste2 (int a) {}  
  
    teste2 () {}  
  
    public static void main(String[] args) {  
        teste t = new teste2(2);  
    }  
}
```

Nada disso irá funcionar...

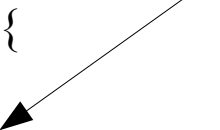


# Construtores e Herança

```
public class teste {  
    teste(int x){}  
}
```

A menos que, ou implementemos um teste() (nem que seja vazio) no pai, ou definamos o construtor do pai que deve ser chamado na classe filho

```
public class teste2 extends teste {  
    teste2 (int a) { super(a); }  
  
    teste2 () { super(0); }  
  
    public static void main(String[] args) {  
        teste t = new teste2(2);  
    }  
}
```



# Herança

Fátima L. S. Nunes

Luciano A. Digiampietri

Norton T. Roman

