Universidade
Estadual de Londrina

Herança

Laboratório de Programação (5COP011)

Prof. Bruno Bogaz Zarpelão

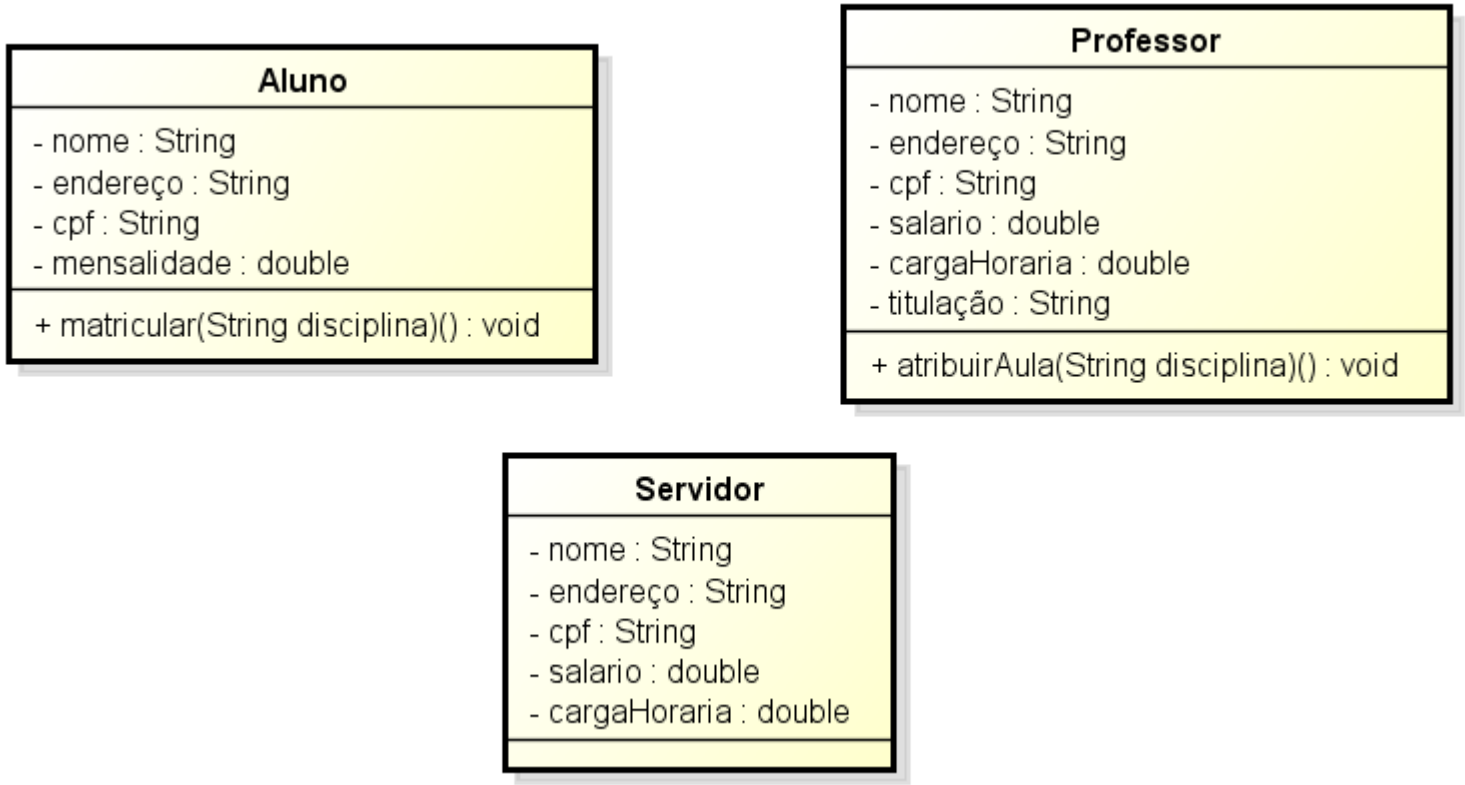
Reutilização de código

- Imagine que temos a seguinte situação:
  - Vamos modelar as pessoas dentro do sistema de cadastro da universidade;
  - Para todas as pessoas, independente se são professores, alunos ou demais servidores, devemos cadastrar nome, endereço, CPF...
  - Contudo, professores e servidores tem salário e carga horária e os alunos não;
  - Aluno pode se matricular. A mesma operação não faz sentido para professores e servidores;

UEL

Reutilização de código

- Solução:

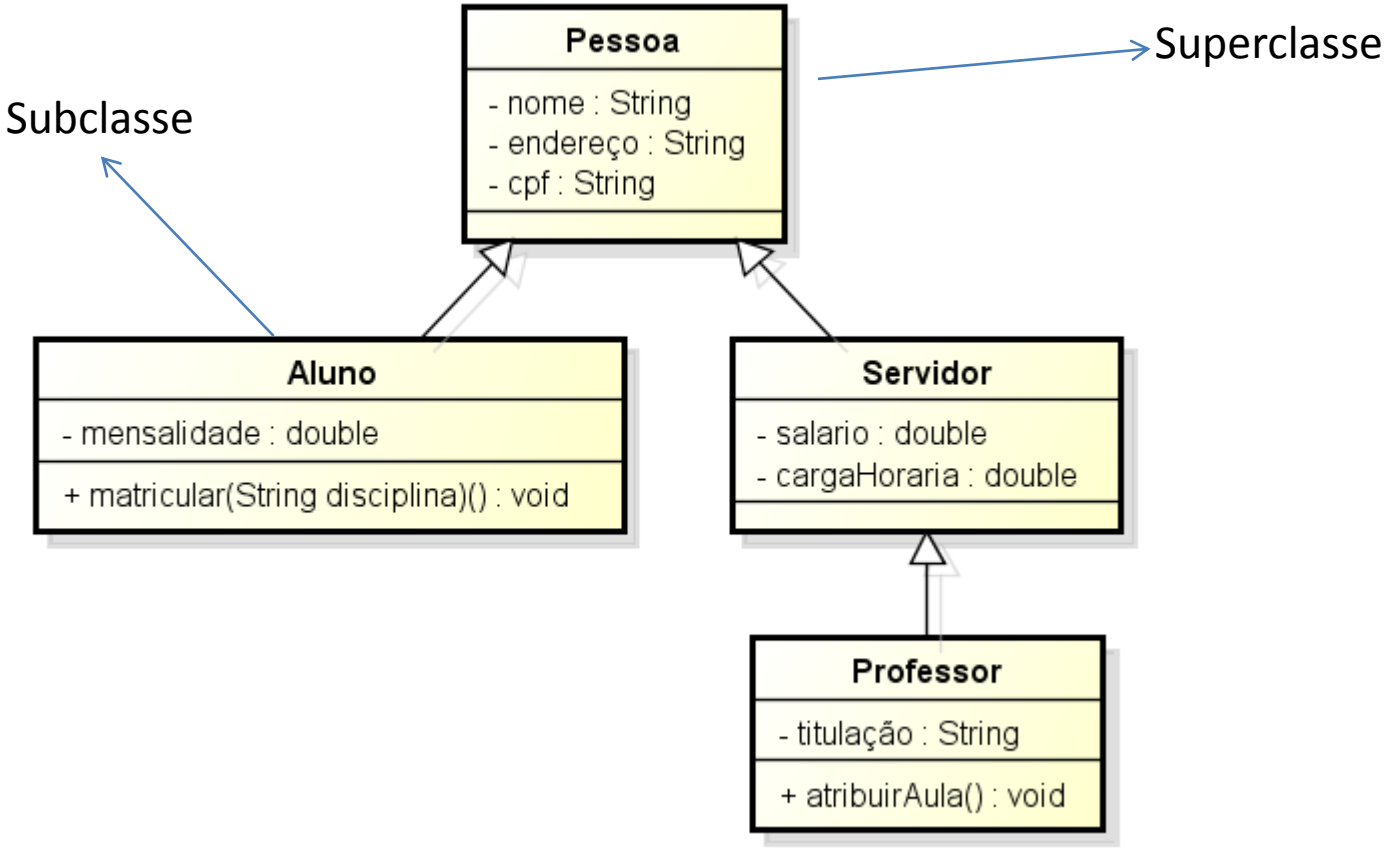


UEL

Reutilização de código

- Se olharmos a figura anterior, veremos que temos atributos repetidos nas três classes;
- A Orientação a Objetos oferece uma ótima ferramenta pra resolver este problema: **a herança!**

Herança


Pessoa
- nome : String
- endereço : String
- cpf : String
Superclasse
Subclasse
Aluno
- mensalidade : double
+ matricular(String disciplina)() : void
Servidor
- salario : double
- cargaHoraria : double
Professor
- titulação : String
+ atribuirAula() : void

Herança

- Uma nova classe (**sub-classe**) é criada absorvendo os membros de uma outra classe (**super classe**).
- Podemos ter uma hierarquia de classes, com vários níveis.
- No Java, toda classe é uma sub-classe de `Object`. Em outras palavras, `Object` é a raiz da hierarquia de classes do Java.

Modificadores de acesso

- As subclasses só acessam atributos e métodos da superclasse que tenham modificadores de acesso *protected* ou *public*;
- Atributos e métodos com modificador de acesso *private* não podem ser acessados diretamente pelas subclasses;
- O modificador *protected* possibilita o acesso do membro pela própria classe, subclasses ou por classes no mesmo pacote. Pode ser visto como um nível de acesso intermediário.

UEL

Herança no Java

```
public class Pessoa{
   ...
}

public class Aluno extends Pessoa{
   ...
}

public class Servidor extends Pessoa{
   ...
}

public class Professor extends Servidor{
   ...
}
```

UEL

Herança no Java

- Suponham que temos os seguintes métodos para as classes Pessoa e Aluno:

```
public class Pessoa{
  ...
  public void cadastrarPessoa(String nome,
           String endereço, String cpf){
    ...
  }
  ...
}
```

Herança no Java

```
public class Aluno extends Pessoa{
   ...
   public void matricular(String disciplina){
      ...
   }
   ...
}
```

UEL

Herança no Java

- Podemos usar estes métodos da seguinte forma:

```
public class Principal{
  ...
  public static void main(String args[]){
    Aluno aluno = new Aluno();
    aluno.cadastrarPessoa(“João”, “Rua da UEL”, “222.222.222-22”);
    aluno.matricular(“Programação OO”);
    ...
  }
  ...
}
```

UEL

Herança no Java

- Invocamos um método da superclasse no objeto que é instância da subclasse:

```
Aluno aluno = new Aluno();
aluno.cadastrarPessoa(“João”, “Rua da UEL”, “222.222.222-22”);
```

UEL

Construtores e herança no Java

- Os construtores não são herdados.
- O construtor de uma subclasse automaticamente invoca o **construtor padrão** da superclasse;
- Se quisermos chamar outro construtor disponível na superclasse, devemos utilizar a instrução *super()*;

UEL

Construtores e herança no Java

```
public class Ponto{

  protected double x,y;

  public Ponto(){
    this.x = 0;
    this.y = 0;
  }

  public Ponto (double pontoX, double pontoY){
    this.x = pontoX;
    this.y = pontoY;
  }

}
```

UEL

Construtores e herança no Java

```
public class Circulo extends Ponto{

  protected double raio;

  public Circulo(){
    //implicitamente é chamado o construtor Ponto();
    raio = 0;
  }

  public Circulo (double pontoX, double pontoY, double pRaio){
    super(pontoX, pontoY);
    raio = pRaio;
  }

}
```

UEL

Sobrescrever métodos

- Na subclasse, podemos sobrescrever métodos da superclasse para incluir comportamentos específicos dos objetos da subclasse;

Sobrescrever métodos

```
public class Animal{
  public void comer (){
    System.out.println(“animal comendo...”);
  }
}

public class Cachorro extends Animal{
  public void comer (){
    System.out.println(“cachorro comendo ração...”);
  }
}
```

UEL

Sobrescrever métodos

```
public class Principal{
  public static void main (String args[]){
    Animal a = new Animal();
    Cachorro c = new Cachorro();
    a.comer();
    c.comer();
  }
}
```

UEL

Sobrescrever métodos

```
Resultado na saída:

animal comendo...
cachorro comendo ração...
```

UEL

Sobrescrever métodos

```
public class Funcionario {

	public String chapa;
	public String nome;
	public double salario;

}
```

```
public class App {

	public static void main(String[] args) {
		Funcionario f = new Funcionario();
		f.chapa = "10556";
		f.nome = "João";
		f.salario = 10000;
		System.out.println(f);
	}

}
```

Sobrescrever métodos

- A impressão realizada pela instrução `System.out.println(f)` não ficou boa...
- Para melhorar o código de maneira elegante, podemos sobrescrever o método `toString()` da classe `Object` na nossa classe `Funcionario`.

Sobrescrever métodos

```
public class Funcionario {

	public String chapa;
	public String nome;
	public double salario;

	public String toString() {

		String funcToString = ("Chapa: "+this.chapa+"\n");
		funcToString = funcToString.concat("Nome: "+this.nome+"\n");
		funcToString = funcToString.concat(
				"Salário:"+NumberFormat.getCurrencyInstance().format(
				this.salario));
		return funcToString;

	}

}
```


UEL

Sobrescrever métodos

- Sempre lembrar:
  - Sobrescrita (overriding) de métodos é diferente de sobrecarga (overload) de métodos;