Aula 8 – POO 1
Herança

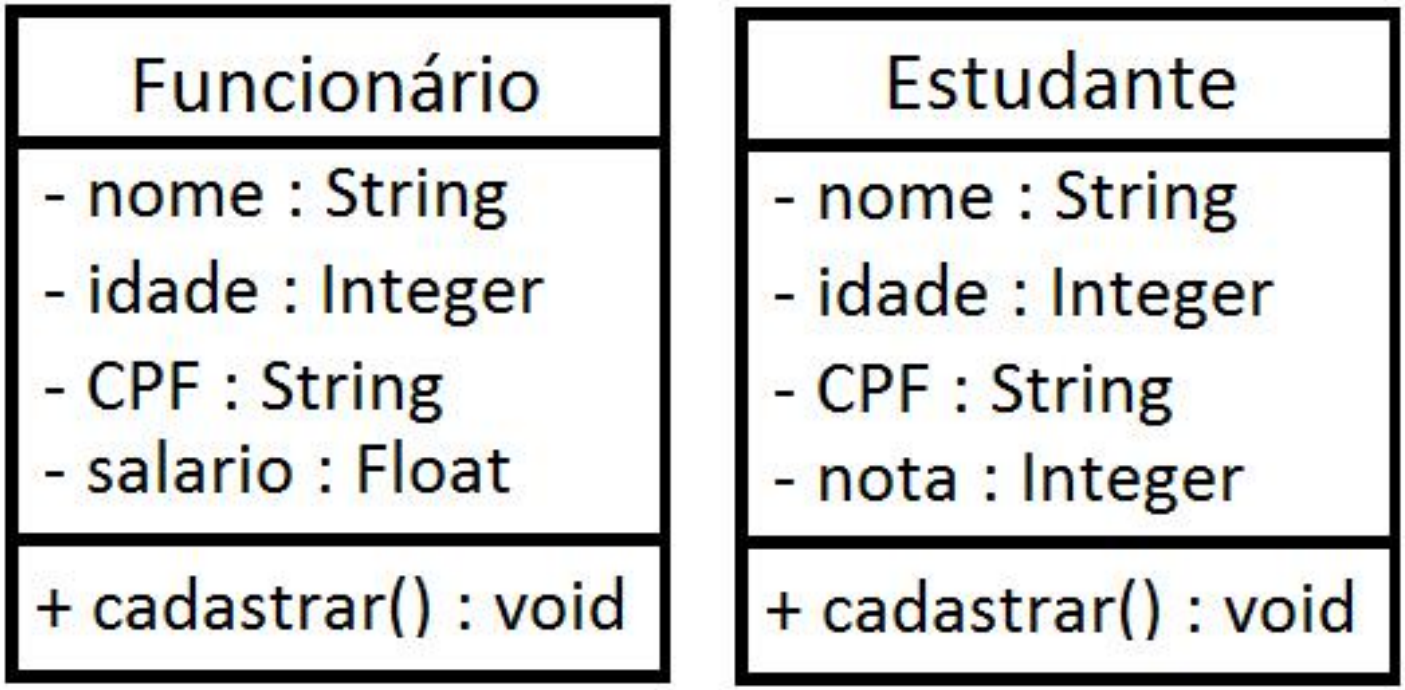
Profa. Elaine Faria
UFU - 2021

Sobre o Material

- Agradecimentos
  - Aos professores José Gustavo e Fabiano, por gentilmente terem cedido seus materiais.

- Os slides consistem de adaptações e modificações dos slides dos professores José Gustavo e Fabiano

Problema


Funcionário
- nome : String
- idade : Integer
- CPF : String
- salario : Float
+ cadastrar() : void
Estudante
- nome : String
- idade : Integer
- CPF : String
- nota : Integer
+ cadastrar() : void

Problema

- As classes Funcionário e Estudante compartilham alguns atributos, mas não todos
  - Não é possível utilizar uma classe para representar instâncias da outra
    - Isso também seria conceitualmente errado, já que Funcionário não é um estudante, e vice-versa
  - Do jeito que está, teremos informações replicadas, o que é ruim

Como resolver o problema?

Herança

- Princípio da POO que permite a criação de novas classes a partir de outras previamente criadas
  - Conceito extremamente útil na POO
- Reutilização de atributos e métodos de uma classe por outra classe
  - Classes podem herdar características comuns de outras classes
- Várias classes com atributos e/ou métodos comuns
  - Classe mais genérica (superclasse)
  - Classe mais específica (subclasse)

Herança

- Classes que herdam atributos/métodos possuem apenas os atributos ou métodos que apenas ela deve possuir (atributos específicos)
- Se uma classe A herda atributos e métodos da classe B, dizemos que
  - A é subclasse (ou classe-filha) de B
  - B é superclasse (ou classe-mãe) de A

Herança

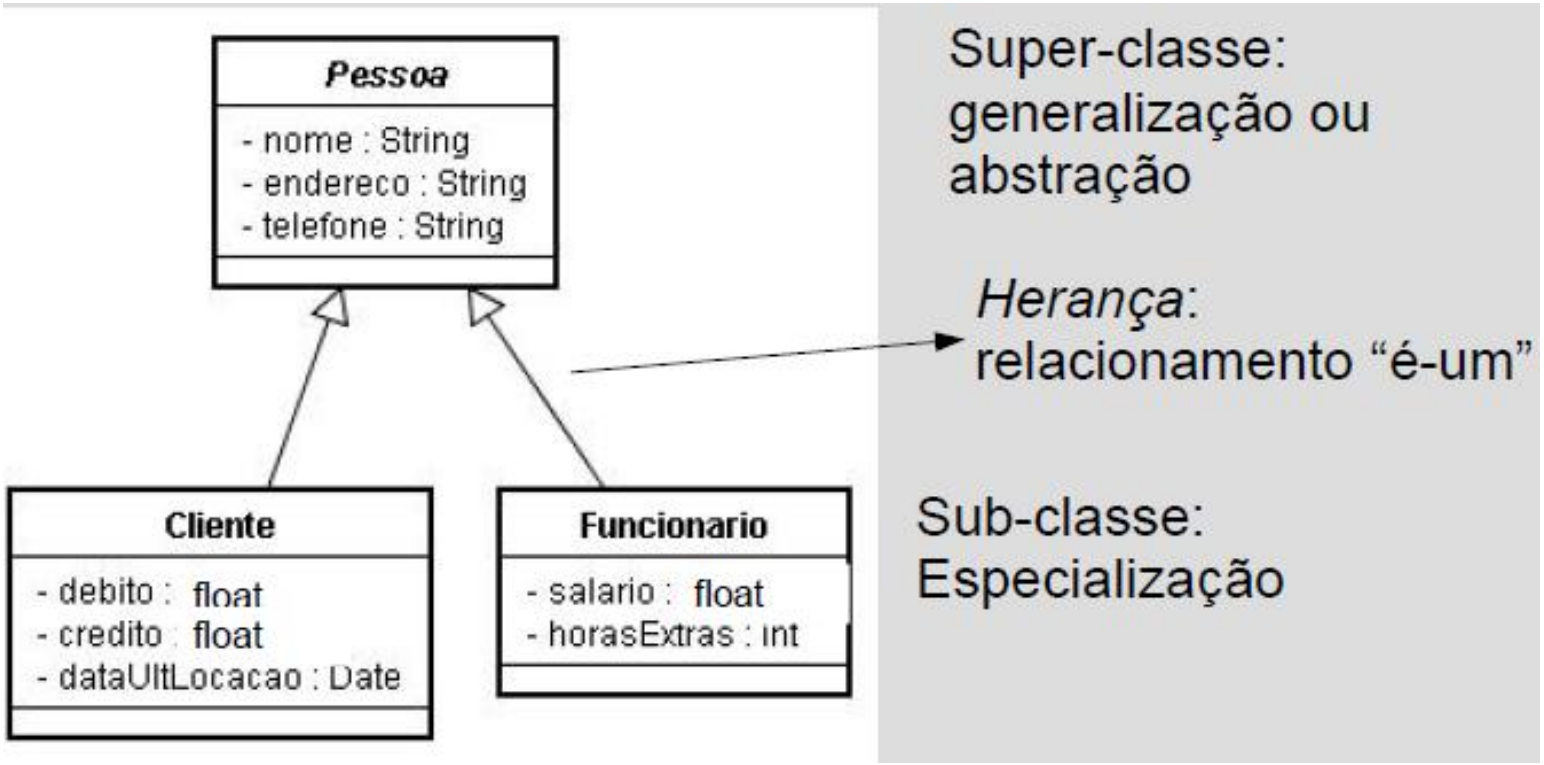
- A classe A utiliza código da classe B, o que proporciona que o código de B se torne disponível automaticamente em A

- A classe A adiciona apenas o código que a torna diferente de B, ou seja, somente incrementa (estende) B, proporcionando um código menor, menos erros e mais simples (especialização)

Herança

- Uma subclasse é mais específica que uma superclasse
- A subclasse exibe comportamentos de sua superclasse e comportamentos adicionais que são específicos à subclasse
- Superclasse direta: superclasse a partir da qual a subclasse herda explicitamente
- Superclasse indireta: qualquer superclasse acima da classe direta na hierarquia de classes

Herança


Pessoa
- nome : String
- endereco : String
- telefone : String
Cliente
- debito : float
- credito : float
- dataUltLocacao : Date
Funcionario
- salario : float
- horasExtras : int
Super-classe:
generalização ou
abstração
Herança:
relacionamento "é-um"
Sub-classe:
Especialização

Herança

- Em java:

```
class Pessoa{
  String nome;
  String endereco;
  String telefone;
  …métodos
}
```

Arquivo Pessoa.java

Herança

Em java:

```
class Cliente extends Pessoa{
  float debito;
  float credito;
  Date dtUltLocacao;
…} Arquivo Cliente.java
```

Herança

- Em java:

```
class Funcionario extends Pessoa{
  float salario;
  int horasExtras;
}
```

Arquivo Funcionario.java

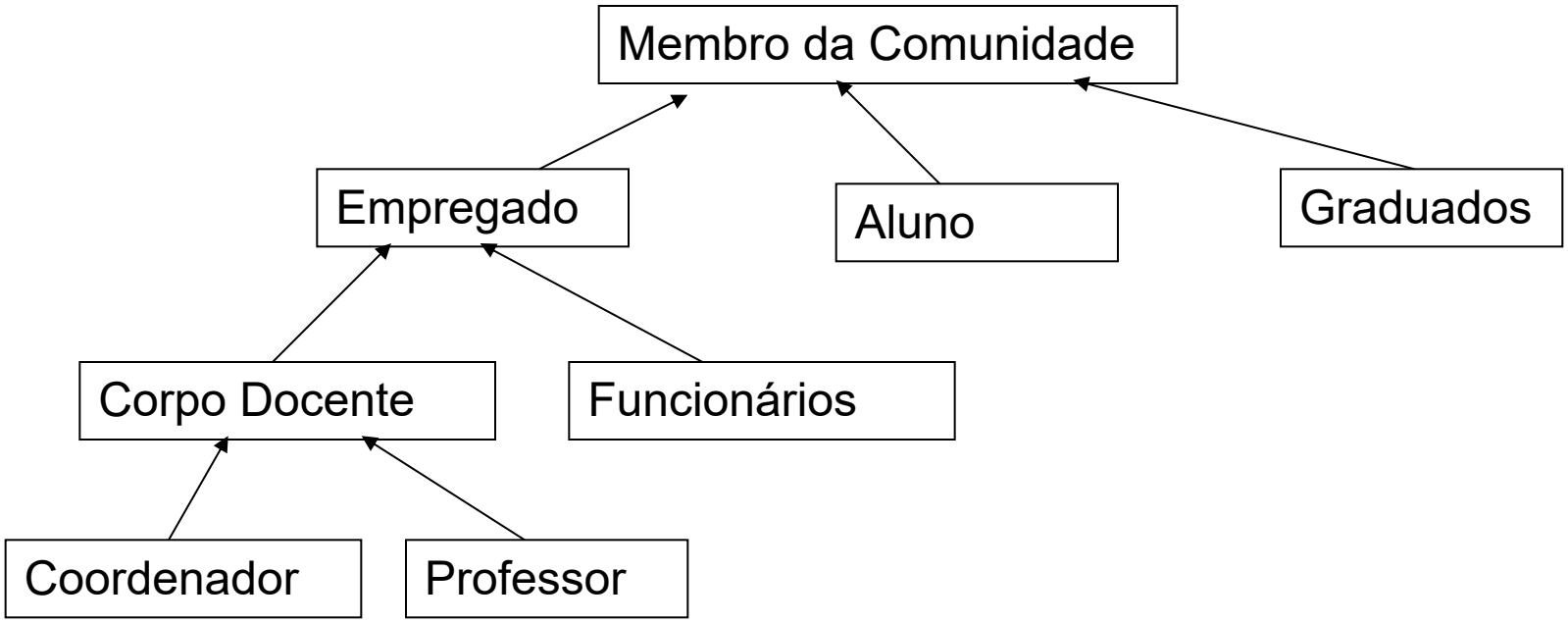
Herança

- Vantagens
  - Diminui a quantidade de código por meio da reutilização de elementos
  - Traz maior integridade e facilidade de manutenção
    - Permite que alterações no código de uma classe mãe sejam compartilhadas com todos os seus herdeiros, sem a necessidade de reprogramação

Herança

- Herança Simples: uma classe é derivada de uma superclasse direta
- Herança múltipla: uma classe é derivada de mais de uma superclasse direta
  - O Java não permite herança múltipla

Herança – Exemplo 1



Herança – Exemplo 2

- Classe *Animal*
  - Todos os objetos da classe Animal possuem características (atributos) comuns, como peso, altura, idade, estados como ter fome, etc.
  - Também fazem determinadas tarefas (serviços ou métodos) como comer, procriar, nascer, morrer, se movimentar, etc.

Herança – Exemplo 2

```
public class Animal {

    private int peso,altura,idade;

    public Animal(int i) {
        idade = i;
    }
    public int retornaPeso() {
        return peso;
    }

    public void alteraAltura(int a) {
        altura = a;
    }
}
```

Herança – Exemplo 2

- A classe Animal foi criada e utilizada em um determinado aplicativo
- Um novo aplicativo deve agora ser feito, utilizando a classe mamífero
- A classe mamífero é muito parecida com a classe Animal, mas possui atributos e métodos que não são comuns a todos os animais, por exemplo, ***mamar( )***
  - Essa classe também realiza algumas atividades de forma diferente dos demais Animais (Nem todos os animais comem ou procriam como os mamíferos, por exemplo)

Herança – Exemplo 2

- Numa linguagem de programação sem **herança**
  - A implementação da classe Mamífero implicaria na replicação do código de Animal, com as modificações necessárias para caracterizar a classe Mamífero
  - Se fosse necessário implementar uma classe Macaco, seria necessário replicar o código de Mamífero na nova classe
  - Uma classe Chimpanzé teria o código de Macaco replicado em si, e assim sucessivamente

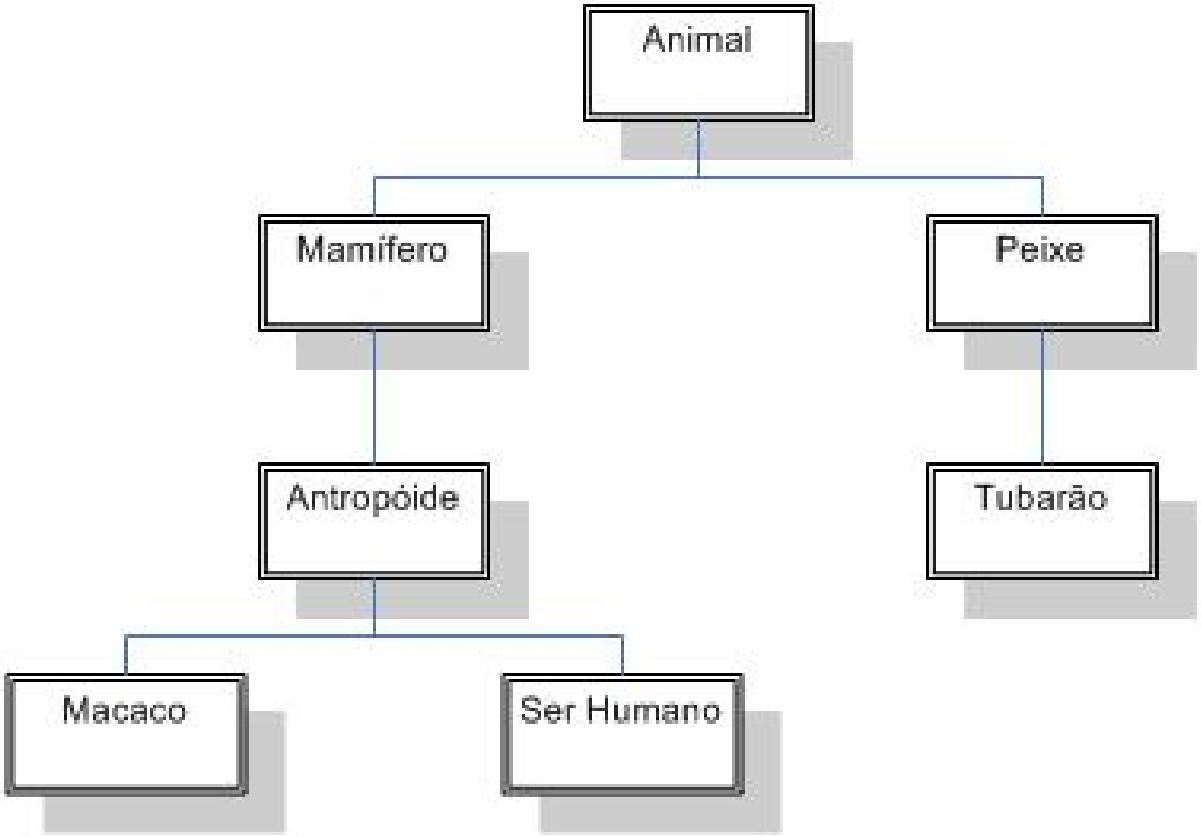
Herança – Exemplo 2

- Numa linguagem de programação sem **herança**
  - Se for necessário implementar todas as classes existentes entre o Ser Vivo e o Mosquito, centenas de classes teriam código replicado das classes anteriores
  - Se houvesse a necessidade de modificar algum método de Animal que fosse comum a todas as centenas de classes, essa modificação teria que ser feita em todas elas, uma a uma

Herança – Exemplo 2

- Mamífero é uma **especialização** de Animal
- A herança funciona como um mecanismo que permite declarar Mamífero como tal, e herdar todos os métodos e atributos da classe ancestral
- Não é preciso replicar código, apenas
  - Incluir os métodos e atributos específicos na classe Mamífero, o que ele tem a mais
  - Redefinir os métodos que achar necessário
- Herança produz uma ordem de hierarquia entre diversas classes relacionadas
- Um objeto herdeiro é uma especialização do seu ancestral, que por consequência será uma generalização de seu sucessor

Herança – Exemplo 2



Herança – Exemplo 2

- É importante ressaltar que normalmente não é preciso redefinir os métodos e atributos herdados, a não ser que haja necessidade de refiná-los
- Antropóide e Ser Humano respiram do mesmo jeito,logo o método ***respira( )*** de Antropóide serve para fazer Homem respirar
- Entretanto, os mamíferos andam de formas diferentes, logo um método ***anda( )*** em Mamífero provavelmente terá que ser redefinido para Antropóide e Elefante, por exemplo

Herança – Exemplo 2

```
public class Animal {
    public Animal() {}
}
class Mamifero extends Animal {
    private boolean querMamar;
    public Mamifero() {}
    void Mamar() {}
    void Respirar() {}
}
class Antropoide extends Mamifero {
    public Antropoide() {}
    void Andar() {}
    void Procriar() {}
}
```

```
class Macaco extends Antropoide {
    public Macaco() {}
    void SobeNaArvore() {}
}
class Homem extends Antropoide {
    public Homem() {}
    void Fala() {}
}
class Peixe extends Animal {
    int número_de_escamas;
    public Peixe() {}
    void Nadar() {}
    void Comer() {}
}
class Tubarao extends Peixe {
    public Tubarao() {}
    void Comer() {}
}
```

Herança – Exemplo 3

```
public class EmpregadoComissao {
   private String primNome;
   private String ultNome;
   private String cartTrab;
   private double vendasBrutas;
   private double porcComissao;

   public EmpregadoComissao(String pN, String uN,
   String cTrab, double vendas, double comis){
       primNome = pN;
       ultNome = um;
       cartTrab = cTrab;
       setVendasBrutas(vendas);
       setPorcComissao(comis);
   }
```

Herança – Exemplo 3

```
public void setPrimNome(String PN){....}
public String getPrimNome(){....}
public void setUltNome(String UM){....}
public String getUltNome(){...}
public void setCartTrab(String cTrab){...}
public String getCartTrab(){...}
public void setVendasBrutas (double vendas){
   vendasBrutas = (vendas <0.0) ? 0.0 : vendas;
}
public double getVendasBrutas() { ...}
public void setPorcComissao(double com){
   porcComissao = (com > 0.0  && com < 1.0) ? com :
   0.0;
}
public double getPorcComissao(){....}
```

Herança – Exemplo 3

- Instâncias da classe ***EmpregadoComissao*** podem aparecer em um aplicativo de banco de dados de folha de pagamento
- Suponha que seja necessário modelar um funcionário comissionado, que tem um salário-base, mais uma pequena comissão por venda
- A classe *EmpregadoComSal* é muito parecida com a classe *EmpregadoComissao*
  - Um objeto *EmpregadoComSal* **é um** *EmpregadoComissao*

Herança – Exemplo 3

```
public class EmpregadoComSal extends
   EmpregadoComissao{
   private double salario;

   public EmpregadoComSal(String pN, String uN,
   String cTrab, double vendas, double porcCom,
   double sal){
       super(pN, uN, cTrab,vendas, porcCom);
       setSalario(sal);
   }
   public void setSalario(double sal){
       salario = (sal < 0.0) ? 0.0 : sal;
   }
   public double getSalario () { return salario;}
   public double calculaSalario () {
       return getSalario() + super.calculaSalario();
   } criar uma nova classe para testes!!
```

Herança – Exemplo 3

- Observação
  - Como os atributos *primNome, ultNome* e *salario* são privados na classe *Empregado*, NÃO PODERÃO SER VISTAS na classe *EmpregadoComissionado*

Referência *this* (Auto Referenciamento)

- Cada objeto pode acessar uma referência a si próprio com a palavra-chave *this*
- Quando um método não-*static* é chamado por um objeto particular, o corpo do método utiliza implicitamente a palavra *this* para referenciar os atributos e outros métodos

Palavra-chave *this*

- Usos comuns
  - Passagem de referência do objeto corrente como parâmetro para outro método
  - Resolução de conflitos de nome: variáveis locais e atributos com o mesmo nome
- Utilizando *this*, objetos de uma classe podem executar métodos de outros objetos passando uma referência de si próprio
- Objetos podem também indicar quais atributos seus serão modificados

Palavra-chave *this*

- Além disso, objetos de uma classe podem invocar, dentro de seu construtor, outro construtor, da mesma classe
  - Economia de código e minimização de duplicação de código (reutilização de código de inicialização)
- Observação: A palavra-chave *this* deve ser a primeira instrução do corpo do construtor

```java
public class Tempo {

    private int hora, minuto, segundo;

    public Tempo() { this(0,0,0); }
    public Tempo(int h) { this(h,0,0); }
    public Tempo(int h, int m) { this(h,m,0); }
    public Tempo(int h, int m, int s) {
        setHora(h);
        setMinuto(m);
        setSegundo(s);
    }
    public Tempo(Tempo t) {
        hora = t.hora;
        minuto = t.minuto;
        segundo = t.segundo;
    }
    public int getHora() { return hora; }
    public void setHora(int hora) {
        this.hora = ((hora >= 0 && hora < 24) ? hora : 0);
    }
    public int getMinuto() { return minuto; }
    public void setMinuto(int minuto) {
        this.minuto = ((minuto >= 0 && minuto < 24) ? minuto : 0);
    }
    public int getSegundo() { return segundo; }
    public void setSegundo(int segundo) {
         this.segundo = ((segundo >= 0 &&segundo < 24) ? segundo : 0);
    }
    public void MostraHoraInteira() {
        System.out.println("A hora é --> "+hora+":"+minuto+":"+segundo+".");
    }
}
```

Super

- Uso do *super*
  - Referência a elementos da super-classe.

```
class Pessoa {
    String nome;
    String endereco;
    String telefone;
    void mostrar() {
        System.out.println("Nome:" + this.nome);
        System.out.println("Endereço: " + this.endereco);
        System.out.println("Telefone: " + this.telefone);
    }
}
```

Super

```
class Funcionario extends Pessoa{
   float salario;
   int horasExtras;
   void mostrar(){
      super.mostrar();
      System.out.println("Salario: " +
         this.salario);
      System.out.println("H.extras: " +
        this.horasExtras);
   }
}
```

Super

- Quando um método da subclasse sobrescrever um método da superclasse, o método da superclasse pode ser acessado a partir da subclasse precedendo o nome do método da superclasse com a palavra-chave *super* e o separador de ponto (.)

Construtores em subclasses

- Instanciar um objeto da subclasse inicia uma cadeia de chamadas de construtor
  - O construtor da subclasse, antes de realizar suas tarefas, invoca o construtor de sua superclasse direta
    - Explícita (via super)
    - Implicitamente (chamando construtor-padrão)

Construtores em subclasses

- Se a superclasse deriva de outra classe o construtor da superclasse invoca o construtor da próxima classe no topo até chegar no construtor da classe *Object* (da qual todas as classes derivam)

Membros *protected*

- Os membros *private* de uma superclasse não são herdados pelas suas subclasses
- Os membros *protected* de uma superclasse podem ser acessados por membros dessa superclasse, por membros da sublcasse e por membros da classe no mesmo pacote
- Os métodos da subclasse podem referir-se a membros *public* e *protected* herdados da superclasse simplesmente utilizando os nomes de membro

Exercício

- Faça um programa que considere três classes, da seguinte maneira:

Pessoa	Cliente	Fornecedor
	nome	nome
nome	sobrenome	sobrenome
sobrenome	idade	idade
idade	RG	RG
RG	lugarNascimento	lugarNascimento
lugarNascimento	CPF	CPNJ
	Endereco	EnderecoEmpresa

Exercício

- Tanto para a classe **Cliente** quanto para a classe **Fornecedor**, implemente os seguintes métodos
  - **`public void infoPessoal()`**: reportar na tela os atributos: "nome", "sobrenome", "idade", "rg" e "lugarNascimento"
  - **`public void info()`:** reportar na tela todos os campos correspondentes à classe da seguinte forma:

No caso da classe Cliente	**No caso da classe Fornecedor**
Nome:<valor do campo nome> Sobrenome:<valor do campo sobrenome> Idade:<valor do campo idade> RG:<valor do campo RG> Lugar:<valor do campo lugarNascimento> CPF:<valor do campo cpf> Endereco:<valor do campo endereco>	Nome:<valor do campo nome> Sobrenome:<valor do campo sobrenome> Idade:<valor do campo idade> RG:<valor do campo RG> Lugar:<valor do campo lugarNascimento> CNPJ:<valor do campo cnpj> Empresa:<valor do campo nomeEmpresa> Endereco:<valor do campo enderecoEmpresa>

Exercícios

- Faça uma implementação Java para a hierarquia de classes professores.
  - Sabe-se que existem dois tipos de professores: horistas e dedicação exclusiva.
  - Ambos possuem nome, matrícula e idade.
  - O salário do professor dedicação exclusiva é um valor fixo
  - O salário do professor horista é obtido multiplicando-se o salário-hora deste professor pelo número de horas trabalhadas

Referências

- DEITEL, H. M., DEITEL, P. J., **Java: Como Programar,** Bookman, São Paulo, 2002
- BARNES, D. J., KOLLING, M., **Programação Orientada a Objetos com Java**, 2004, ISBN: 8576050129
- MEDINA, R., D., **Apostila de ActionScript**, disponível em http://www-usr.inf.ufsm.br/~rose/curso3/cafe/cap3_Heranca.pdf