Programação Orientada a Objetos

Giselle Lopes Ferrari Ronque

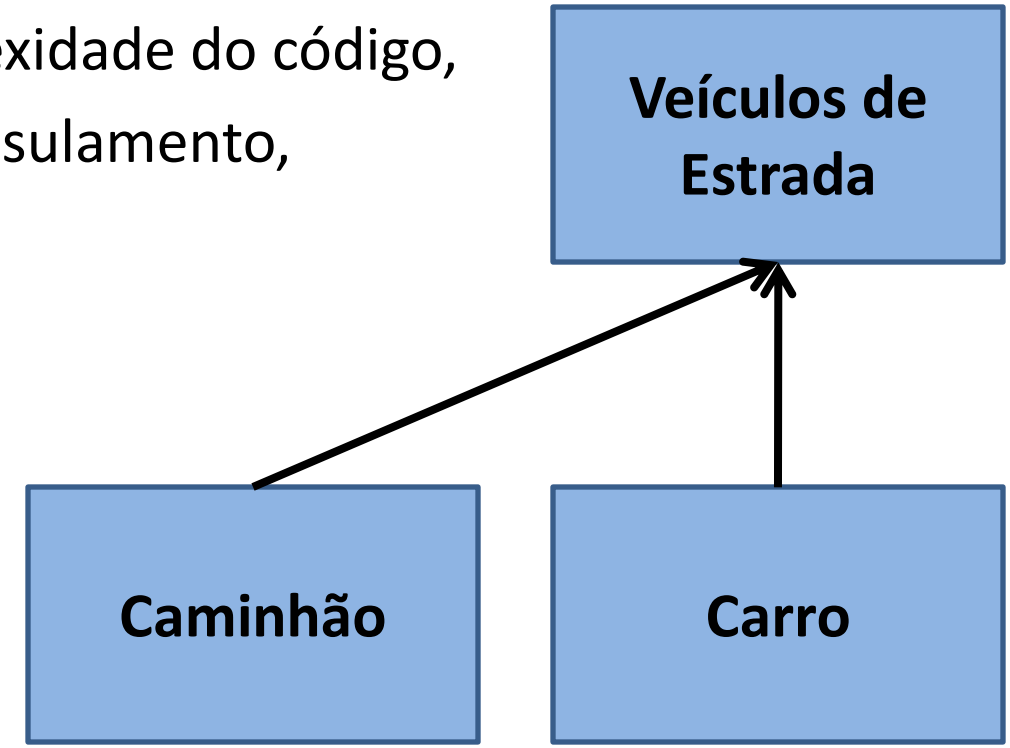
ferrari@eletrica.ufpr.br

Herança

- Uma vez que se defina uma classe mais geral (classe-base, superclasse ou classe-mãe), pode-se gerar desta, classes mais especializadas ➔ **Herança**.
- Estas sub-classes, ou classe-derivadas herdam todas as características da classe-mãe, além das suas particulares adicionais.

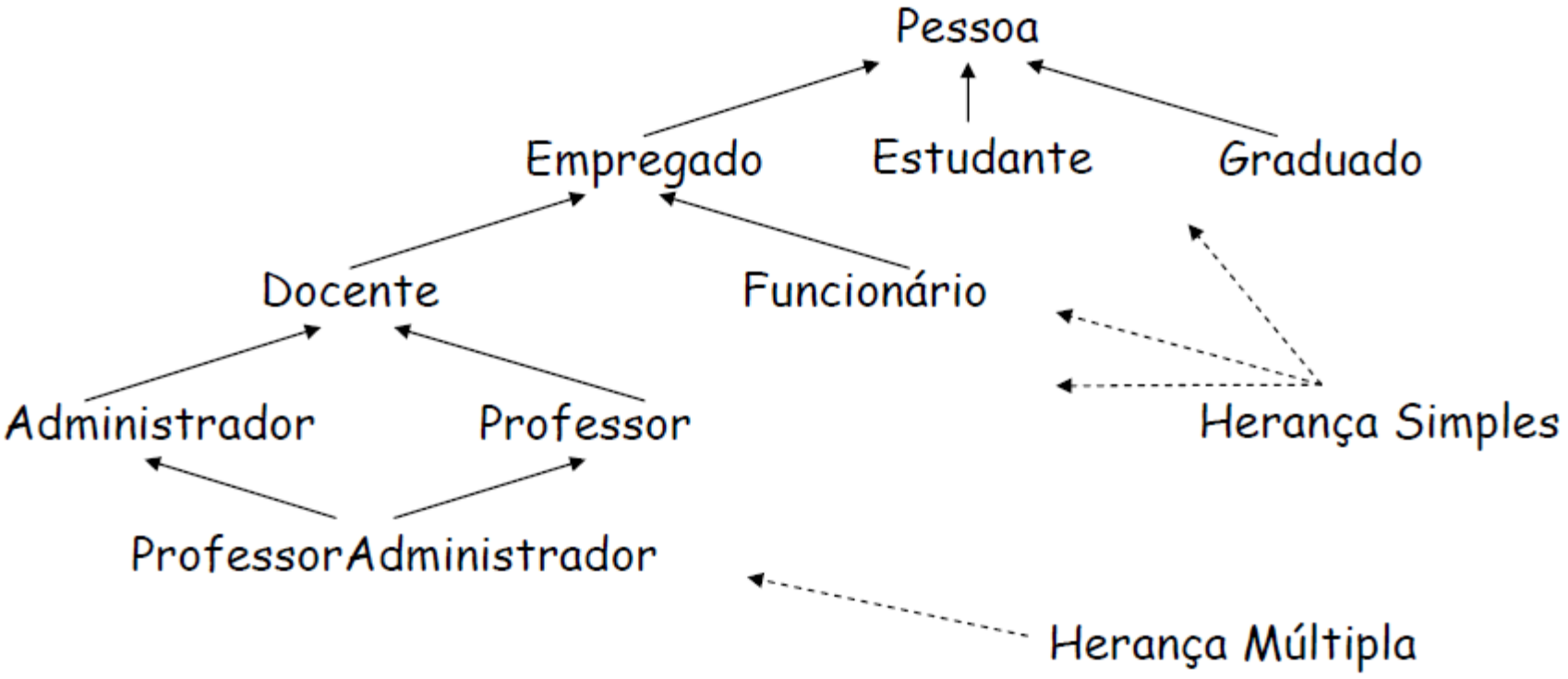
Herança

– Substitui, com vantagens, a reutilização de código ("Ctrl-C/Ctrl-V"):
  - diminui o número de LOCs,
  - diminui a complexidade do código,
  - preserva o encapsulamento,
  - etc.



Herança

- A herança costuma formar estruturas hierárquicas do tipo árvore.



Definição

```
class Derivada: especificadorDeHeranca Base {
...
}
```

- Nessa linha, informa-se que a classe-derivada é herdeira da classe-base. Como só temos uma classe-base temos uma **herança simples**.
- O especificador de herança define a forma como a classe-derivada pode acessar os atributos e métodos da classe-base.

Especificador de Herança

- O especificador de herança altera a forma como se processa a herança, alterando o **tipo** de acesso herdado. Pode-se utilizar os especificadores public, protected e private.
- Protótipos:
  - class Derivada: **public** Base { ... };
  - class Derivada: **protected** Base { ... };
  - class Derivada: **private** Base { ... };

Especificador de Herança

- Herança Pública:
  - Indica que os membros públicos da classe base serão membros públicos da classe derivada e os membros protegidos da classe-base serão membros protegidos da classe derivada;
- Herança Privada:
  - Indica que tanto os membros públicos quanto os protegidos da classe-base serão membros privados da classe derivada;
- Herança Protegida:
  - Indica que tanto os membros públicos quanto os protegidos da classe-base serão protegidos da classe derivada;

Herança

- O acesso aos membros da classe-base irá depender:
  - especificador de herança (public, protect e private);
  - especificadores de acesso na classe-base (public, protect e private).

Especificador de acesso na classe-base	Especificador de herança	Acesso herdado
public	public	public
protected	public	protected
private	public	inacessível
public	protected	protected
protected	protected	protected
private	protected	inacessível
public	private	private
protected	private	private
private	private	inacessível

Exemplo

```
# include <iostream >
using namespace std;

class Base {
      public :
            int publico;
      protected :
            int secreto;
      private :
            int ultrasecreto;
};
```

```
class DerivadaA : public Base {
      public :
        DerivadaA ( ) {
            int a = secreto;            // Ok
            int b = ultrasecreto;       // Erro
            int c =  publico;           // Ok
        }
};

class DerivadaB : private Base {
      public :
        DerivadaB ( ) {
            int a = secreto;            // Ok
            int b = ultrasecreto;       // Erro
            int c =  publico;           // Ok
        }
};
```

Exemplo

```
void main () {
    int x;
    DerivadaA    objA;

    x = objA.a;                 // Erro
    x = objA.b;                 // Erro
    x = objA.c;                 // Ok

    DerivadaB    objB;

    x = objB.a;                 // Erro
    x = objB.b;                 // Erro
    x = objB.c;                 // Erro
}
```

Construtores

- Se nenhum construtor for especificado na classe derivada, o construtor da classe-base será usado.
- Em uma herança e aconselhável que um método construtor chame os métodos construtores ancestrais para a declaração dos atributos ancestrais, em vez de atribuir ele mesmo esses atributos.

Reescrevendo Métodos da Classe-Base

- Quando a classe base e a classe derivada definem funções com o mesmo nome o compilador terá que resolver o escopo das funções:

- A regra é a seguinte:
  - Se duas funções de mesmo nome existem, uma na classe-base e outra na classe derivadas, a função da classe derivada será executada se for chamada por meio de um objeto da classe derivada;
  - Se um objeto da classe-base é criado, usará sempre funções da própria classe-base, pois não conhece nada da classe derivada.

Exemplo

```
#include <cstdlib>
#include <iostream>
#include <string.h>

using namespace std;

class BasAg {
    protected:
        char nome [80];
        char numag[4];
    public:
        BasAg() {
            nome[0]='\0'; numag[0] = '\0';
        }
        BasAg(char n[], char ng[]) {
                if(strlen(n) < 80)
                    strcpy(nome,n);
                if(strlen(ng) < 4)
                    strcpy(numag,ng);
        }
        void print (){
                cout << "\nDados do agente:";
                cout << "\n---------------";
                cout << "\nNome : " << nome;
                cout << "\nNumero : " <<numag;
        }
        };
```

Exemplo

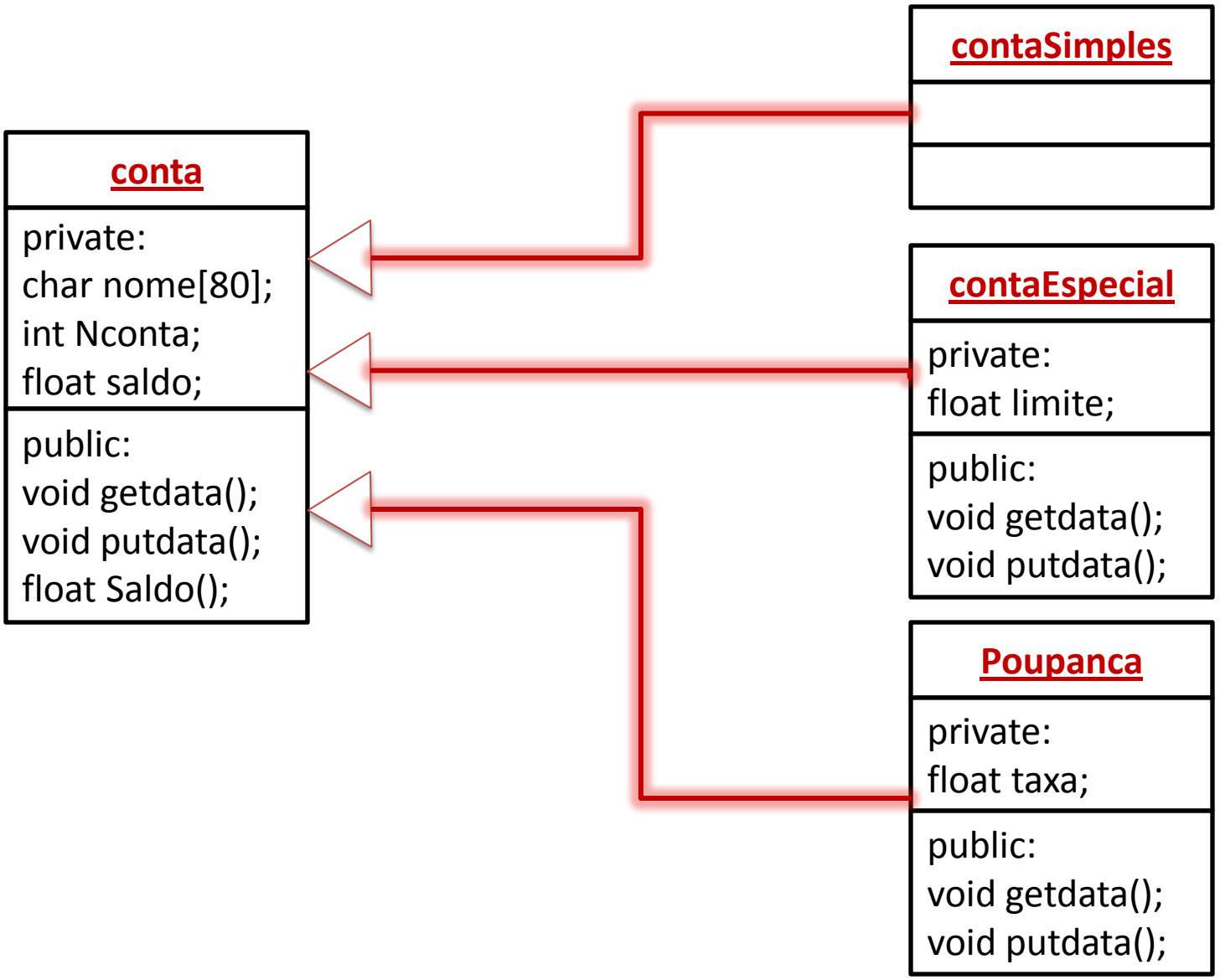
```
class Agente : public BasAg {
    protected:
        float altura;
        int idade;
    public:
        Agente () : BasAg() {
            altura = 0;
            idade = 0;
        }
        Agente (char n[], char ng[], float a,
         int i) : BasAg (n, ng) {
            altura = a;
            idade = i;
        }
```

```
    void print (){
            BasAg::print();
            cout << "\nAltura : " << altura;
            cout << "\nIdade : " << idade;
        }
};

int main(int argc, char *argv[])
{
    Agente agente("James Bond", "007",
      1.90, 35);
    agente.print();
    getchar();
    return EXIT_SUCCESS;
}
```

```
Dados do agente:
----------------------
Nome : James Bond
Numero : 007
Altura : 1.9
Idade: 35
```

Exercício 1


conta
private:
char nome[80];
int Nconta;
float saldo;
public:
void getdata();
void putdata();
float Saldo();
contaSimples
contaEspecial
private:
float limite;
public:
void getdata();
void putdata();
Poupanca
private:
float taxa;
public:
void getdata();
void putdata();

Exercício 1

```
void main(){
    contaSimples c1, c2;
    contaEspecial c3;
    Poupanca c4;

    cout << "\nDigite os dados da conta simples 1. ";
    c1.getdata();
    cout << "\nDigite os dados da conta simples 2. ";
    c2.getdata();
    cout << "\nDigite os dados da conta especial. ";
    c3.getdata();
    cout << "\nDigite os dados da conta poupanca. ";
    c4.getdata();
    cout << "\nConta Simples 1. ";
    c1.putdata();
    cout << "\nConta Simples 2. ";
    c2.putdata();
    cout << "\nConta Especial. ";
    c3.putdata();
    cout << "\nConta Poupanca. ";
    c4.putdata();
}
```