Polimorfismo

Roland Teodorowitsch

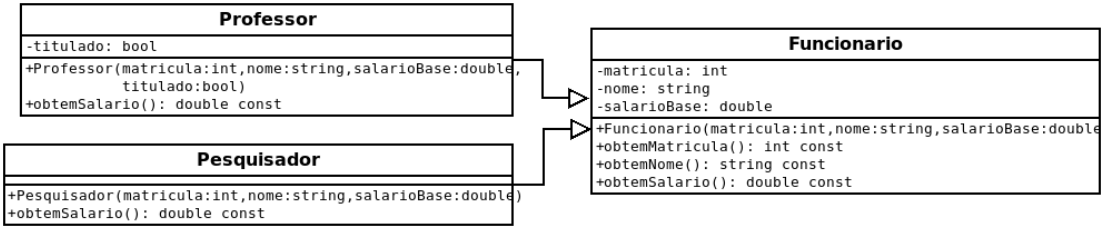
Programação Orientada a Objetos - ECo - Curso de Engenharia de Computação - PUCRS

3 de junho de 2024

Polimorfismo

Introdução

- Observe a seguinte modelagem, em especial o método `double obtemSalario()`



Arquivo Funcionario.hpp

```
#ifndef _FUNCIONARIO_HPP
#define _FUNCIONARIO_HPP

#include <string>

using namespace std;

class Funcionario {
private:
  int matricula;
  string nome;
  double salarioBase;
public:
  Funcionario(int matricula, string nome, double salarioBase);
  int obtemMatricula() const;
  string obtemNome() const;
  double obtemSalario() const;
};
#endif
```

Arquivo Funcionario.cpp

```cpp
#include "Funcionario.hpp"

using namespace std;

Funcionario::Funcionario(int matricula, string nome, double salarioBase) {
  this->matricula = matricula;
  this->nome = nome;
  this->salarioBase = salarioBase;
}

int Funcionario::obtemMatricula() const {
  return matricula;
}

string Funcionario::obtemNome() const {
  return nome;
}

double Funcionario::obtemSalario() const {
  double inss = salarioBase * 0.1;
  double irpf = salarioBase * 0.27;
  return salarioBase - inss - irpf;
}
```

Arquivo Professor.hpp

```
#ifndef _PROFESSOR_HPP
#define _PROFESSOR_HPP

#include <string>
#include "Funcionario.hpp"

using namespace std;

class Professor : public Funcionario {
private:
  bool titulado;
public:
  Professor(int matricula, string nome, double salarioBase, bool titulado);
  double obtemSalario() const;
};
#endif
```

Arquivo Professor.cpp

```
#include "Professor.hpp"

Professor::Professor(int matricula, string nome, double salarioBase, bool titulado) :
     Funcionario (matricula, nome, salarioBase) {
  this->titulado = titulado;
}

double Professor::obtemSalario() const {
  double sal = Funcionario::obtemSalario();
  double adic;
  if (titulado)
     adic = sal * 0.25;
  else
     adic = sal * 0.1;
  return sal + adic;
}
```

Arquivo Pesquisador.hpp

```
#ifndef _PESQUISADOR_HPP
#define _PESQUISADOR_HPP

#include <string>
#include "Funcionario.hpp"

using namespace std;

class Pesquisador : public Funcionario {
public:
  Pesquisador(int matricula, string nome, double salarioBase);
  double obtemSalario() const;
};
#endif
```

Arquivo Pesquisador.cpp

```
#include "Pesquisador.hpp"

Pesquisador::Pesquisador(int matricula, string nome, double salarioBase) :
    Funcionario (matricula, nome, salarioBase) {}

double Pesquisador::obtemSalario() const {
  double sal = Funcionario::obtemSalario();
  double adic = sal * 0.15;
  return sal + adic;
}
```

Arquivo main.cpp

```cpp
#include <iostream>
#include "Funcionario.hpp"
#include "Professor.hpp"
#include "Pesquisador.hpp"

using namespace std;

int main() {
  Professor prof(12341,"Jose da Silva",1000,true);
  Pesquisador pesq(12342,"Lisandro Barbosa",1000);
  Funcionario *funPr, *funPe;
  Professor *pr;
  Pesquisador *pe;
  pr    = &prof;
  funPr = &prof;
  pe    = &pesq;
  funPe = &pesq;
  // Conforme o tipo de ponteiro, uma versao diferente do metodo obtemSalario() sera chamada
  cout << "pr->obtemSalario():    " << pr->obtemSalario() << endl;    // 787.5
  cout << "pe->obtemSalario():    " << pe->obtemSalario() << endl;    // 724.5
  cout << "funPr->obtemSalario(): " << funPr->obtemSalario() << endl; // 630
  cout << "funPe->obtemSalario(): " << funPe->obtemSalario() << endl; // 630
  return 0;
}
```

Funções Virtuais

- Normalmente, o tipo do ponteiro define o método que será acionado
- **Polimorfismo** é a característica única de linguagens orientadas a objetos que permite que diferentes objetos comportem-se de diversas formas, dependendo do contexto
- Em C++ polimorfismo é obtido através de **funções virtuais**
- Funções virtuais são declaradas através da palavra reservada `virtual`
- É possível fazer redefinição de métodos em uma hierarquia

Relembrando o Conceito de Herança...

- Utiliza-se objetos em hierarquia de classes:

```
class Base {
public:
  void func1();
};

class Derivada : public Base {
public:
  void func2();
};
```

- Como a classe `Derivada` é derivada da classe `Base`, todos os membros de `Base` também estarão em `Derivada`
- `Derivada` é um super-conjunto de `Base`: todas operações que podem ser feitas com objetos de `Base` também o podem com objetos de `Derivada`
- Um objeto da classe `Derivada` também é um objeto da classe `Base`

O Que Pode e o Que NÃO Pode Ser Feito?

```
class Base {
public:
  void func1();
};

class Derivada : public Base {
public:
  void func2();
};

int main(){
  Base base, *pBase;
  Derivada derivada, *pDerivada;

  base = derivada;          // copia a parte Base de derivada para base
  derivada = base;          // ERRO!
  pBase = &base;            // ok
  pBase = &derivada;        // ok, pBase aponta para objeto da classe Derivada
  pBase = pDerivada;        // ok
  pDerivada = &base;        // ERRO!
  pDerivada = &derivada;    // ok
  pDerivada = pBase;        // ERRO!

  return 0;
}
```

Explorando Polimorfismo na Classe Funcionario

- Alterando a classe Funcionario:

```
#ifndef _FUNCIONARIO_HPP
#define _FUNCIONARIO_HPP

#include <string>

using namespace std;

class Funcionario {
private:
  int matricula;
  string nome;
  double salarioBase;
public:
  Funcionario(int matricula, string nome, double salarioBase);
  int obtemMatricula() const;
  string obtemNome() const;
  virtual double obtemSalario() const;
};
#endif
```

Explorando Polimorfismo com uma Nova `main()` (2)

- Agora o que define a versão de `obtemSalario()` é o tipo de objeto e não mais o ponteiro:

```
#include <iostream>
#include "Funcionario.hpp"
#include "Professor.hpp"
#include "Pesquisador.hpp"

using namespace std;

int main() {
  Funcionario *vet[5];                                          // RESULTADO:
  vet[0] = new Funcionario(12340,"Carlos Saldanha",1000.0);     // Nome: Carlos Saldanha 630
  vet[1] = new Professor(12341,"Jose da Silva",1000.0,false);   // Nome: Jose da Silva 693
  vet[2] = new Pesquisador(12342,"Lisandro Barbosa",1000.0);    // Nome: Lisandro Barbosa 724.5
  vet[3] = new Professor(12343,"Carmem Borges",1000.0,true);    // Nome: Carmem Borges 787.5
  vet[4] = new Pesquisador(12344,"Luiza Prates",1000.0);        // Nome: Luiza Prates 724.5
  int numEmpregados = sizeof(vet)/sizeof(Funcionario *);
  for (int i=0; i<numEmpregados; ++i)
      cout << "Nome: " << vet[i]->obtemNome() << " " << vet[i]->obtemSalario() << endl;
  for (int i=0; i<numEmpregados; ++i)
      delete vet[i];
  return 0;
}
```

Classes Virtuais Puras

- Em C++, é possível definir classes que nunca serão instanciadas, definindo somente a sua interface, que deverá ser implementada nas suas classes derivadas
- Isso é feito declarando métodos virtuais puros (o médodo é declarado com `virtual` e acrescenta-se `= 0;` na sua declaração):

```
#include <iostream>
class Base { // CLASSE VIRTUAL PURA (ABSTRATA)
public:
  Base() {}                    // Construtor
  virtual void funcao() = 0; // Metodo virtual puro
};
class Derivada : public Base { // CLASSE DERIVADA
public:
  Derivada() : Base() {}                                    // Construtor
  void funcao() { std::cout << "funcao()" << std::endl; } // Metodo implementado na derivada
};
int main() {
  // NAO eh possivel criar um objeto de Base [Base *objetoB = new Base();], pois ela eh virtual pura,
  // mas ainda eh possivel ter um ponteiro de Base para apontar para objetos derivados (polimorfismo)
  Base *objetoD = new Derivada();
  objetoD -> funcao();   // Chama o metodo da derivada
  delete objetoD;
  return 0;
}
```

Lista de Exercícios

Exercício 1

1. Execute os seguintes passos:
   - Crie uma classe abstrata chamada `Animal` que armazene o nome e a idade de um animal (informados via método construtor)
   - Defina métodos de acesso para os atributos nome e idade
   - Defina um método (virtual) chamado `string emiteSom()` – o objetivo deste método, nas classes derivadas, é retornar um string com o som emitido pelo animal (por exemplo: "au au", "piu piu", etc.)
   - Escreva pelo menos 3 classes derivadas de `Animal`, definindo pelo menos 3 tipos de animais diferentes
   - Escreva um programa (`main()`) que instancia um vetor de 6 animais, instanciando 2 animais de cada tipo criado (imagine uma jaula em um zoológico com esses animais)
   - Após, usando um comando `for`, imprima nome e a idade do animal armazenado na jaula, bem como o som emitido por cada animal

Exercício 2

2. Um banco trabalha com três tipos de contas:
   - conta corrente comum (classe `ContaComum`);
   - conta corrente com limite (classe `ContaLimite`);
   - conta poupança (classe `ContaPoupanca`).

   Em todos os casos é necessário guardar o número da conta, o nome do correntista e o saldo. Para a conta poupança é necessário guardar o dia do aniversário da conta (quando são creditados os juros). Já para a conta com limite é necessário guardar o valor do limite. As contas também armazenam uma lista de transações (até 10 transações). Uma transação é definida por uma data, valor da transação e descrição. Se o valor for negativo, a transação é considerada um débito (crédito caso contrário).
   Para obter a data em uma `string`, você pode utilizar as linhas abaixo:

```
#include <ctime>
time_t now = time(0);
string date = ctime(&now);
```

Exercício 2

2. (Continuação)
   As operações possíveis são:
   - depósito;
   - saque;
   - impressão de extrato.

   Essas operações devem ser definidas numa classe abstrata pura (interface) denominada `Conta`. A operação de depósito é igual nos três tipos de conta. O saque só é diferente na conta com limite, pois esta admite que o saldo fique negativo até o limite estabelecido. Finalmente o extrato é diferente para as três:
   - na conta comum exibe o número da conta, nome do cliente, transações e o saldo;
   - na conta limite imprime também o valor do limite;
   - na conta poupança imprime também o dia do aniversário.

Exercício 2

2. (Continuação)
   Implemente a hierarquia de classes das contas explorando **polimorfismo**.
   Ao final, escreva um programa em C++ que permita ao usuário fazer depósitos, saques e verificação de extrato nas suas contas a partir do número da conta. Utilize um único vetor para armazenar todos os tipos de contas.

Créditos

Créditos

- Estas lâminas contêm trechos de materiais disponibilizados pelos professores Rafael Garibotti, Matheus Trevisan, Daniel Callegari, Sandro Fiorini e Bernardo Copstein.

Solução dos Exercícios

Exercício 1: exercicio1.cpp

```cpp
#include <iostream>

using namespace std;

class Animal {
private:
  string nome;  int idade;
public:
  Animal(string n, int i):nome(n),idade(i) {}
  string obtemNome() { return nome; }
  int obtemIdade() { return idade; }
  virtual string emiteSom() = 0;
};

class Porco : public Animal {
public:
  Porco(string n, int i):Animal(n,i) {}            string emiteSom() { return "oinc-oinc"; }
};

class Cachorro : public Animal {
public:
  Cachorro(string n, int i):Animal(n,i) {}         string emiteSom() { return "au-au"; }
};

class Gato : public Animal {
public:
  Gato(string n, int i):Animal(n,i) {}             string emiteSom() { return "miau-miau"; }
};

int main() {
  Animal *zoo[] = { new Porco("Pepa Pig",7),   new Porco("Porquinho",5), new Cachorro("Beethoven",8),
                    new Cachorro("Marley",12), new Gato("Mingau",4),     new Gato("Garfield",10)      };
  int nAnimais = sizeof(zoo) / sizeof(Animal *);
  for (int i=0; i<nAnimais; ++i) cout << zoo[i]->obtemNome() << " " << zoo[i]->obtemIdade() << " " << zoo[i]->emiteSom() << endl;
  for (int i=0; i<nAnimais; ++i) delete zoo[i];
  return 0;
}
```