Programação Orientada a Objetos

Giselle Lopes Ferrari Ronque

ferrari@eletrica.ufpr.br

Polimorfismo

- Polimorfismo é a capacidade de assumir várias formas.

- Em C++ indica a habilidade de uma única instrução chamar diferentes funções e portanto assumir formas diferentes.

Polimorfismo

- Aplicações:
  - Diferentes construtores de uma mesma classe;
  - Sobrecarga de Operadores (tipo especial de Polimorfismo);
  - Exemplo: Funções/Métodos para cadastro, alteração e exclusão de Dados.

Exemplo 1

```
#include <cstdlib>
#include <iostream>

using namespace std;

class Cliente {
   private:
      char Nome[30];
      char Endereco[50];
   public:
      Cliente();
      Cliente(char [], char []);
      void Alterar(char []);
      void Alterar(char [], char []);
      void imprimir();
};
```

```
Cliente::Cliente(){
   strcpy(Nome, "None");
   strcpy(Endereco, "None");
};
Cliente::Cliente(char nome[], char end[]) {
   strcpy(Nome, nome);
   strcpy(Endereco, end);
};
void Cliente::Alterar(char nome[]) {
   strcpy(Nome, nome);
}
void Cliente::Alterar(char nome[], char end[])
{
   strcpy(Nome, nome);
   strcpy(Endereco, end);
}
```

Exemplo 1

```
void Cliente::imprimir()
{
   cout<< "\n Nome: " << Nome << "\n
    Endereco: " << Endereco<< "\n";
}
```

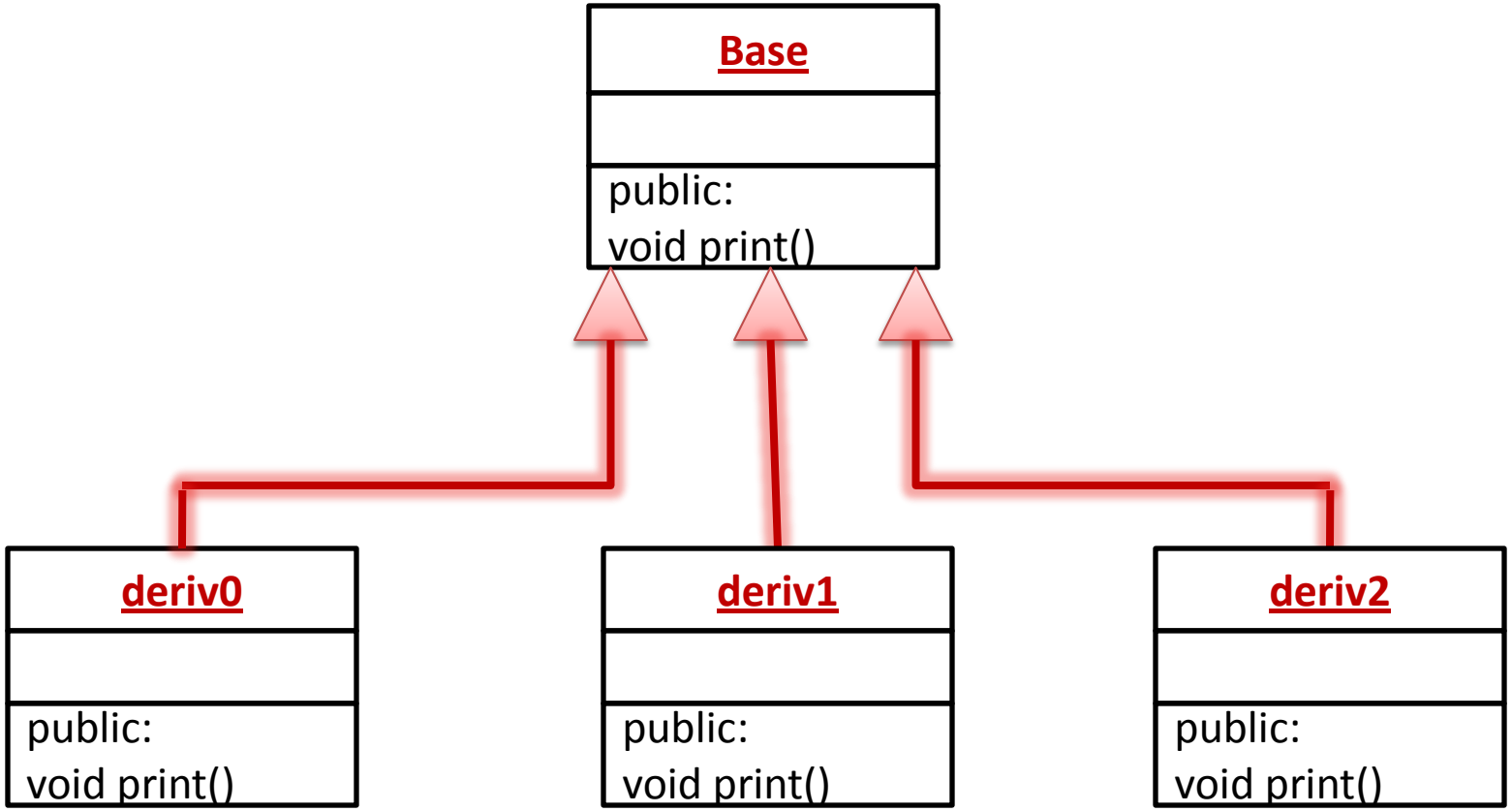
Saída:

Nome: None
Endereco: None
Nome: Joao
Endereco: Av. Brasilia, 10
Nome: Julia
Endereco: Rua Batel, 200
Nome: Maria
Endereco: Av. Brasilia, 10

```
int main(int argc, char *argv[])
{
   Cliente obj, obj2("Joao", "Av. Brasilia,
     10");
   obj.imprimir();
   obj2.imprimir();
   obj.Alterar("Julia","Rua Batel, 200");
   obj.imprimir();
   obj2.Alterar("Maria");
   obj2.imprimir();

   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Exemplo 2


Base
public:
void print()
deriv0
public:
void print()
deriv1
public:
void print()
deriv2
public:
void print()

Exemplo 2

```
#include <iostream.h>
class base {
      public:
            void print( ) {cout << "\nBase";}
};
class deriv0 : public base {
      public:
            void print( ) {cout << "\nDeriv0";}
};
class deriv1 : public base {
      public:
            void print( ) {cout << "\nDeriv1";}
};
class deriv2 : public base {
      public:
            void print( ) {cout << "\nDeriv2";}
};
void main() {
      base *p[3]; // Matriz de ponteiros
      deriv0 dv0; // Objeto da classe derivo
      deriv1 dv1; // Objeto da classe deriv1
      deriv2 dv2; // Objeto da classe deriv2
      p[0]=&dv0; // Preenche a matriz
      p[1]=&dv1; // com endereço dos objetos
      p[2]=&dv2; // das classes derivadas
      for(int i=0; i<3; i++)
            p[i]->print( ); // chama print( )
}
```

Saída:
Base
Base
Base

Funções Virtuais

- Para acessar objetos de diferentes classes usando a mesma instrução, devemos declarar as funções da classe - base que serão reescritas em classes derivadas usando a palavra **virtual.**

Exemplo 3

```
#include <iostream.h>
class base {
      public:
      virtual void print( ) {cout << "\nBase";}
};
class deriv0 : public base {
      public:
            void print( ) {cout << "\nDeriv0";}
};
class deriv1 : public base {
      public:
            void print( ) {cout << "\nDeriv1";}
};
```

```
class deriv2 : public base {
      public:
            void print( ) {cout << "\nDeriv2";}
};
void main() {
      base *p[3]; // Matriz de ponteiros
      deriv0 dv0; // Objeto da classe derivo
      deriv1 dv1; // Objeto da classe deriv1
      deriv2 dv2; // Objeto da classe deriv2
      p[0]=&dv0; // Preenche a matriz
      p[1]=&dv1; // com endereço dos objetos
      p[2]=&dv2; // das classes derivadas
      for(int i=0; i<3; i++)
            p[i]->print( ); // chama print( )
}
```

Saída: **Deriv0**
Deriv1
Deriv2

Resolução dinâmica

- Permite que uma instrução seja associada a uma função no momento de sua execução. O programador especifica que uma determinada ação deve ser tomada em um objeto por meio de uma instrução.
- O programa, na hora da execução, interpreta a ação e vincula a instrução à função apropriada. Ex. ***p[i] -> print( ); do exemplo anterior.***

Funções Virtuais Puras

- Definição: são funções sem bloco de código.

- Utilizado em situações em que a função nunca será executada, contribuindo apenas que seja redefinidas em todas as classes derivadas.

- Para criá-las usamos o operador de atribuição seguido de um zero após o seu protótipo. Ex:

```
class base {
        public:
                virtual void print( ) = 0; // função virtual pura
 };
```

Não é permitido declarar nenhum objeto da classe em que a função virtual pura foi incluída!

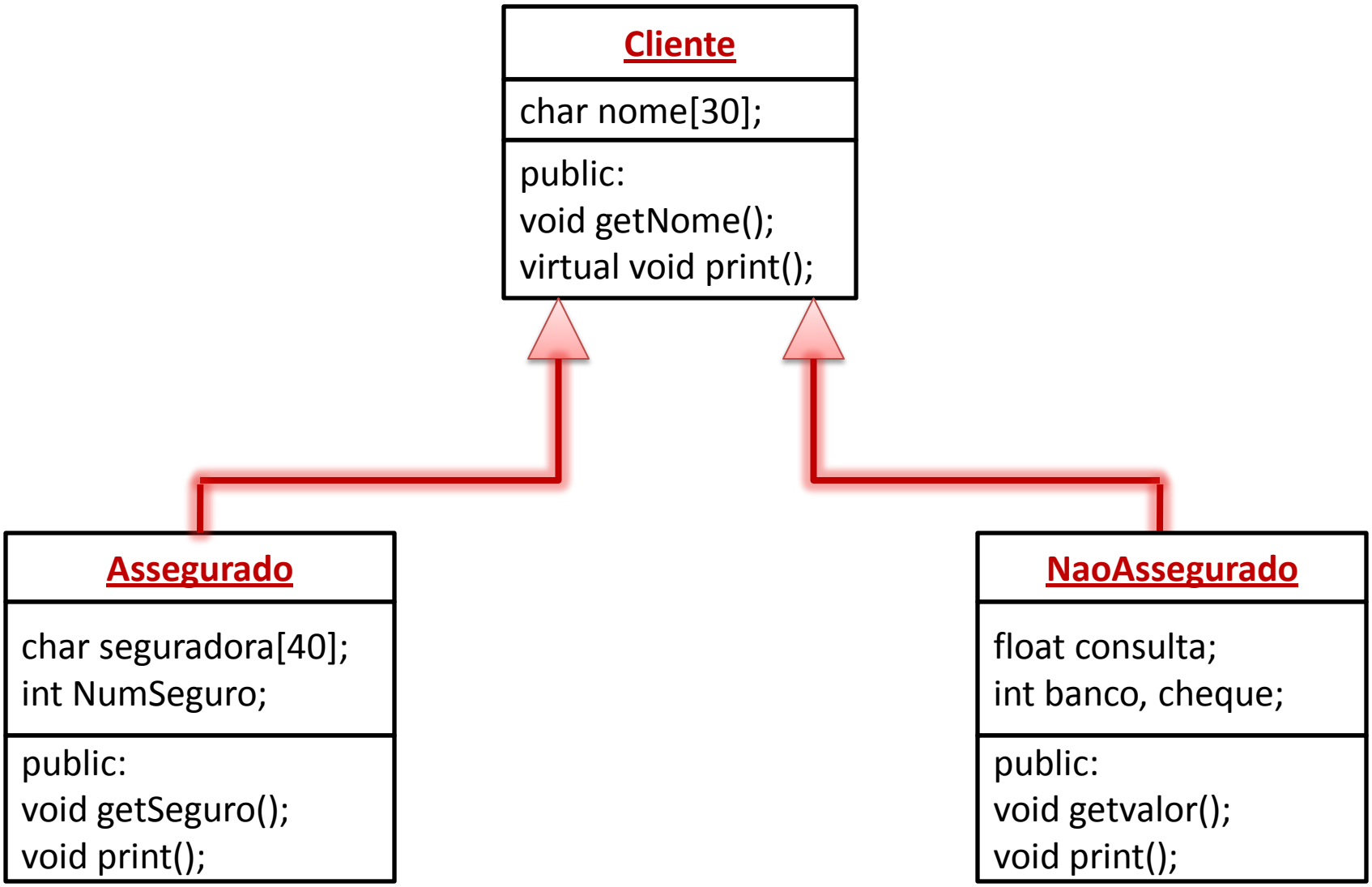
Classe Abstrata

- Não é permitido que sejam criados objetos.
- Existe somente para ser usada como base pelas outras.
- A classe que define uma função virtual pura é uma classe abstrata pois não se pode declarar objetos dela.
- Pode ser manuseada com ponteiros para manipular objetos de classes derivadas.

Exercício 1

- Implemetar:
  - Classes:
    - Cliente, Assegurado, NaoAssegurado;
  - Programa principal para testar as funcionalidades das classes.

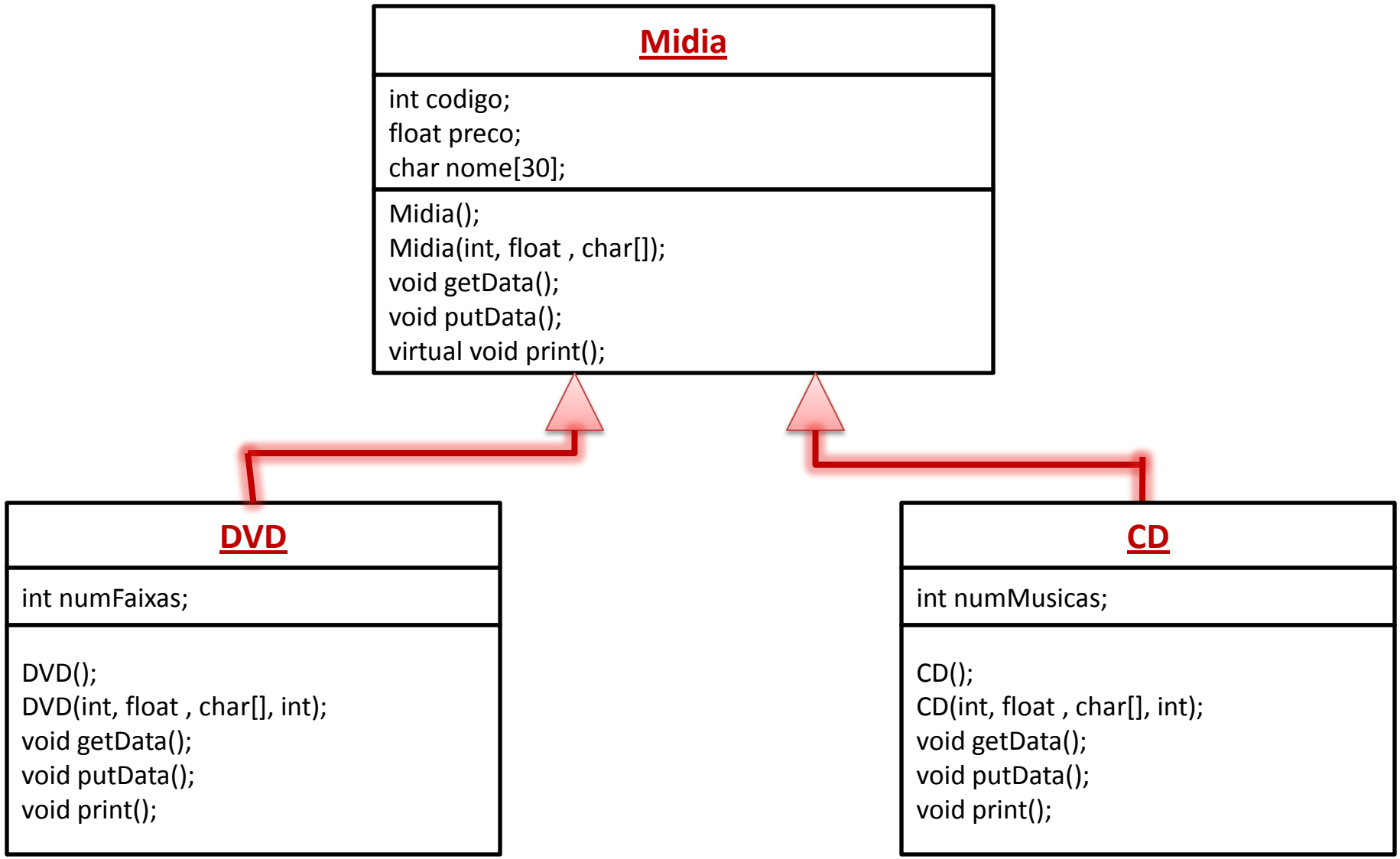
Exercício 1


Cliente
char nome[30];
public:
void getNome();
virtual void print();
Assegurado
char seguradora[40];
int NumSeguro;
public:
void getSeguro();
void print();
NaoAssegurado
float consulta;
int banco, cheque;
public:
void getvalor();
void print();

Exercício 2

- Implemetar:
  - Classes:
    - *Mídia, DVD, CD*;
  - Métodos:
    - *getData* – retorna as informações contida nos campos da classe;
    - *putData* – insere os dados necessários para preencher os campos de um objeto de uma dada classe;
    - *print* – imprime as informações contidas nos campos da classe.
  - Programa principal para testar as funcionalidades das classes.

Exercício 2


Midia
int codigo;
float preco;
char nome[30];
Midia();
Midia(int, float , char[]);
void getData();
void putData();
virtual void print();
DVD
int numFaixas;
DVD();
DVD(int, float , char[], int);
void getData();
void putData();
void print();
CD
int numMusicas;
CD();
CD(int, float , char[], int);
void getData();
void putData();
void print();