# Polimorfismo

Prof. Valdir Pedrinho de Tomin Junior, Eng.

Universidade Federal de Santa Catarina
Centro Tecnológico de Joinville
EMB5631 – Programação III

e-mail: valdir.pedrinho@ufsc.br

2 de outubro de 2018

# Introdução

## O que é polimorfismo?

- O polimorfismo permite 'programar no geral' em vez de 'programar no específico';
- O polimorfismo permite escrever programas que processam objetos de classes que fazem parte da mesma hierarquia de classes como se todos fossem objetos da classe básica da hierarquia;
- Com o polimorfismo podemos projetar e implementar sistemas que são facilmente extensíveis
  - Novas classes podem ser adicionadas com pouca ou nenhuma modificação a partes gerais do programa, contanto que as novas classes façam parte da hierarquia de herança que o programa processa genericamente.

# Introdução

## Um exemplo conceitual

- Suponha que criamos um programa que simula o movimento de vários tipos de animais para um estudo biológico;
- As classes **Peixe**, **Anfíbio** e **Pássaro** representam esses animais;
- Cada uma dessas classes herda da classe básica **Animal**;
- Toda classe derivada implementa o método **mover**, herdado de **Animal**;
- Nosso programa mantém um vetor de ponteiros para objetos das várias classes derivadas de **Animal**;
- Para simular os movimentos dos animais, o programa envia a mesma mensagem a cada objeto;
- Cada tipo específico de **Animal** responde a uma mensagem mover de maneira própria e única;
- O fato de que cada objeto sabe "fazer a coisa certa" em resposta à mesma chamada de método é o conceito-chave do polimorfismo.

# Polimorfismo

## Quando é Polimorfismo e quando é Herança?

- Uma função pode produzir a ocorrência de ações diferentes, dependendo do tipo do objeto em que a função é invocada.
- Se a classe **Retangulo** é derivada da classe **Quadrilatero**, então um objeto **Retangulo** é uma versão mais específica de um objeto **Quadrilatero**;
- Qualquer operação que pode ser realizada em um objeto da classe **Quadrilatero** também pode ser realizada em um objeto da classe **Retangulo**;
- O polimorfismo ocorre quando um programa invoca uma função **virtual** por meio de um ponteiro de classe básica ou referência;
  - O C++ escolhe dinamicamente (isto é, em tempo de execução) a função correta para a classe a partir da qual o objeto foi instanciado.

# Polimorfismo

## Como Proceder?

- Com herança pública, um objeto de uma classe derivada pode ser tratado como um objeto de sua classe base;
- Apesar do fato de que os objetos de classe derivados são tipos diferentes, o compilador permite isso porque cada objeto de classe derivada é um objeto de sua classe base.
- Não podemos tratar um objeto de classe base como um objeto de alguma de suas classes derivadas.

# Polimorfismo

**Exemplo: Chamada de funções da classe básica a partir de objetos da classe derivada**

Programa 13.1-13.5 do livro *DEITEL. C++, Como programar. 5ª ed.*

**Exemplo: Apontando ponteiros de classe derivada para objetos da classe básica**

Programa 13.6 do livro *DEITEL. C++, Como programar. 5ª ed.*

**Exemplo: Chamadas de função de classe derivada via ponteiros de classe básica**

Programa 13.7 do livro *DEITEL. C++, Como programar. 5ª ed.*

# Manipulação de objetos da classe base

## Upcasting

- **Upcasting** é o processo de tratar um ponteiro ou uma referência de objeto da classe derivada como um ponteiro da classe base.
- Não é necessário fazer o **upcast** manualmente. Basta atribuir um ponteiro de classe derivada (ou referência) ao ponteiro da classe base;

## Quais as restrições?

- O **upcast** não muda o objeto;
- Quando **upcast** de um objeto é feito, só é possível acessar os membros de dados que estão definidos na classe base.

## Exemplo

Classes **Employee**, **Manager** e **Clerk**

# Manipulação de objetos da classe derivada

## Downcasting

- **Downcasting** é um processo oposto ao **upcasting**;
- Realiza a conversão de um ponteiro da classe base para um ponteiro de classe derivada;
- Deve ser feito manualmente. Isso significa que é necessário especificar o tipo de conversão explícito.

## Problemas com o **Downcasting**

- **Downcasting** não é seguro como o **upcasting**.
- Um objeto de classe derivada pode ser sempre tratado como objeto de classe base. No entanto, o oposto não está certo.
  - Por exemplo, um gerente é sempre uma pessoa; mas uma pessoa nem sempre é um gerente.

# Manipulação de objetos da classe derivada

## **Downcasting** com Segurança

- **dynamic_cast** é um operador que converte com segurança um tipo para outro tipo.
- Se a operação é possível e segura, devolve o endereço do objeto que é convertido.

## Sintaxe

```
dynamic_cast <new_type> (object)
```

## Exemplo

Classes **Employee**, **Manager** e **Clerk**

## Funções Virtuais

- O tipo do **handler** determina que funcionalidades da classe invocar;
- Com funções virtuais, o tipo de objeto, não o tipo de identificador usado para chamar a função de membro, determina qual versão de uma função virtual para invocar;
- Permite que o programa determine dinamicamente (em tempo de execução) qual método deve ser chamado;
- A escolha da função apropriada para a chamada em tempo de execução (em vez de em tempo de compilação) é conhecido como vinculação dinâmica.

# Funções Virtuais

## Sintaxe

```
virtual função (lista de parâmetros)
```

## Redefinição do Método

O uso da palavra reservada **virtual** não sobrescreve o método, mas implica na **redefinição** do mesmo.

## Exemplo

Programa 13.8-10 do livro *DEITEL. C++, Como programar. 5ª ed.*

# Fixando...

## Exercício

Crie uma hierarquia de **Shape** simples: uma classe base chamada **Shape** e classes derivadas chamadas **Circle**, **Square** e **Triangle**. Na classe base, crie uma função **virtual** chamada **draw()** e a substitua nas classes derivadas. Crie um *array* de ponteiros para os objetos **Shape**, dinamicamente (e, assim, execute o **upcasting** dos ponteiros), e chame **draw()** pelos ponteiros da classe base, para verificar o comportamento da função **virtual**.

# Resumo

## Ponteiros de mesmo tipo do objeto

- Apontar um ponteiro de classe básica para um objeto de classe básica é simples e direto — as chamadas feitas a partir do ponteiro de classe básica simplesmente invocam as funcionalidades da classe básica;
- Apontar um ponteiro de classe derivada para um objeto de classe derivada é simples e direto — as chamadas feitas a partir do ponteiro de classe derivada simplesmente invocam as funcionalidades de classe derivada.

## Ponteiro de uma classe derivada para classe básica

- Apontar um ponteiro de classe derivada para um objeto de classe básica gera um erro de compilação. Um objeto de classe básica não contém membros exclusivos de classe derivada que podem ser invocados a partir de um ponteiro de classe derivada.

# Resumo

## Ponteiros de mesmo tipo do objeto

- É seguro apontar um ponteiro de classe básica para um objeto de classe derivada, porque o objeto de classe derivada é um objeto de sua classe básica. Esse ponteiro pode ser utilizado para invocar apenas os métodos da classe básica.
- Se o programador tentar referenciar um membro exclusivo da classe derivada por meio do ponteiro de classe básica, o compilador informa um erro.
- Para evitar esse erro, o programador deve fazer coerção do ponteiro de classe básica para um ponteiro de classe derivada. O ponteiro de classe derivada então pode ser utilizado para invocar as funcionalidades completas do objeto de classe derivada.

# Estudo de caso

**Exemplo: Sistema de folha de pagamento utilizando polimorfismo e informações de tipo de tempo de execução com downcasting, dynamic_cast, typeid e type_info**

Programa 13.25 do *DEITEL. C++, Como programar. 5ª ed.*

# Classe Abstratas

## Características

- Classes abstratas são classes que não podem ser instanciadas;
- Geralmente são usadas como classes bases em hierarquia de heranças;
- Classes abstratas são incompletas - classes derivadas devem definir as partes faltantes;
- Classes que podem ser usadas para instanciar objetos são chamadas de classes concretas;

# Funções Virtuais Puras

## Características

- Tornamos uma classe abstrata declarando uma ou mais de suas funções virtual como 'puras';
- Uma função virtual pura é especificada colocando-se "= 0" em sua declaração;
- O "= 0" é conhecido como um especificador puro.

# Funções Virtuais Puras

## Diferença entre funções Virtuais e Virtuais Puras

- A diferença entre uma função virtual e uma função virtual pura é que uma função virtual tem uma implementação e fornece à classe derivada a opção de substituir a função;
- Uma função virtual pura não fornece uma implementação e requer que a classe derivada substitua a função para que essa classe derivada seja concreta; caso contrário, a classe derivada permanece abstrata.

# Funções Virtuais Puras

## Diferença entre funções Virtuais e Virtuais Puras

- A diferença entre uma função virtual e uma função virtual pura é que uma função virtual tem uma implementação e fornece à classe derivada a opção de substituir a função;
- Uma função virtual pura não fornece uma implementação e requer que a classe derivada substitua a função para que essa classe derivada seja concreta; caso contrário, a classe derivada permanece abstrata.

# Funções Virtuais Puras

## Quando usar?

- São usados quando não faz sentido para a classe base ter uma implementação de uma função, mas deseja-se que todas as classes derivas concretas implementem a função;
- Não podemos instanciar objetos de uma classe base abstrata, mas podemos usar a classe base abstrata para declarar ponteiros e referências que podem se referir a objetos de quaisquer classes concretas derivadas da classe abstrata.

## Exemplo

Classe Abstrata **Shape**, classes derivadas **Triangle** e **Rectangle**.

# Fixando...

## Exercício

Crie uma hierarquia de herança de **Roedor**: **Rato**, **Gerbil**, **Hamster**, etc. Na classe base, forneça métodos que sejam comuns a todos os roedores e os redefina nas classes derivadas para executar diferentes comportamentos, dependendo do tipo específico de roedor. Crie uma matriz de ponteiros para o **Roedor**, preencha-o com diferentes tipos específicos de roedores e chame seus métodos de classe base para ver o que acontece.

## Exercício

Modifique o Exercício do Slide 12 para que **draw()** seja uma função **virtual pura**. Tente criar um objeto do tipo **Shape**. Tente chamar a função **virtual pura** dentro do construtor e ver o que acontece. Deixando-a como um virtual pura, dê uma definição a **draw()**.

# Destrutores virtuais

## Comportamento Indefinido

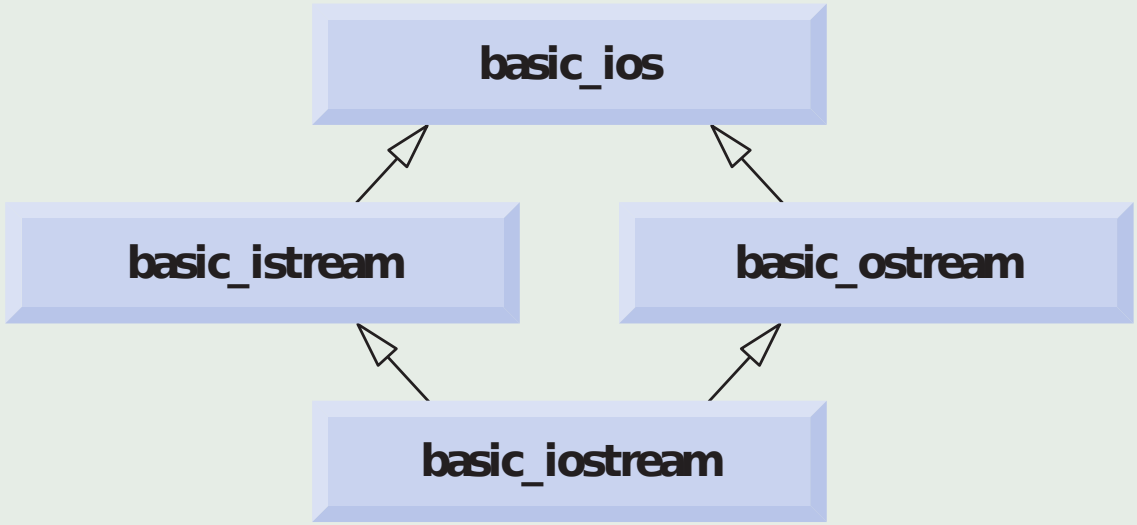
Se um objeto de classe derivada com um destrutor não virtual é destruído explicitamente aplicando o operador delete a um ponteiro de classe básica para o objeto, o padrão C++ especifica que o comportamento é indefinido.

## Solução

- Utilização de um destrutor virtual (destrutor que é declarado com a palavra-chave **virtual**);
- Isso torna virtual todos os destrutores de classe derivada mesmo que eles não tenham o mesmo nome do destrutor de classe básica;
- Se um objeto na hierarquia é destruído explicitamente aplicando o operador **delete** a um ponteiro de classe básica, o destrutor da classe apropriada é chamado com base no objeto para o qual o ponteiro de classe básica aponta.

# Herança múltipla e classes básicas virtual

## Herança em forma de losango




## Problemas Potenciais

- Como as classes basic_istream e basic_ostream herdam cada uma de basic_ios , há um problema potencial para basic_iostream;
- A classe basic_iostream poderia conter duas cópias dos membros de classe basic_ios — uma herdada via classe basic_istream e, outra, via classe basic_ostream, gerando um erro de compilação;

# Herança múltipla e classes básicas virtual

## Como proceder?

- As classes derivadas vão possuir subobjeto de classe básica, isto é, os membros da classe base, os quais podem entrar em conflito quando herdados para outra classe derivada, mais abaixo na hierarquia de herança;
- O problema de subobjetos duplicados é resolvido com a herança virtual;
- Quando uma classe básica é herdada como virtual , somente um subobjeto aparecerá na classe derivada — um processo chamado de herança da classe básica virtual;

# Herança múltipla e classes básicas virtual

## Como proceder?

- Visto que cada uma das classes básicas utiliza a herança virtual para herdar membros da classe base, o compilador assegura que apenas um subobjeto do tipo base é herdado na classe derivada mais abaixo na hierarquia;
- O compilador agora permite a conversão implícita do ponteiro de classe derivada.

## Exemplo

Programa 24.07-11 do *DEITEL. C++, Como programar. 5ª ed.*

# Referências e Exercícios

## Livro Texto

Paul Deitel e Harvey Deitel.**C++: Como programar**. 5ªed. Ed. Prentice Hall Brasil. 2006.

## Conteúdo dessa Aula

- Capítulo 13
- Seção 24.8

## Exercícios do Livro

13.3, 13.6, 13.8, 13.10, 13.12, 13.13 e 13.15