

Programação Orientada a Objetos

Herança e Composição – Parte 2

**parte deste material didático foi extraído das notas de aula do Profs.
Renato Mesquita e Ana Liddy – DEE/UFMG**

Para onde vamos ...

- Nesta unidade veremos ...
 - Como tratar os relacionamentos de herança e composição utilizando classes e objetos em C++
- Referência básica
 - *Thinking in C++, Vol. 1*
 - Capítulos 14 e 15
 - *Thinking in C++, Vol. 2*
 - Capítulo 6



Agenda

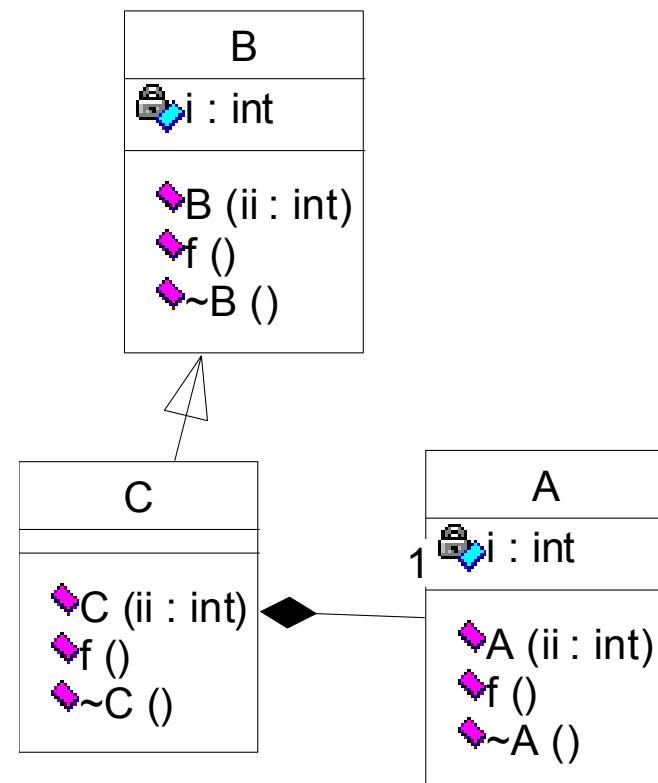
Herança e Composição

IV.3. Combinando composição e herança



IV.3. Combinando composição e herança

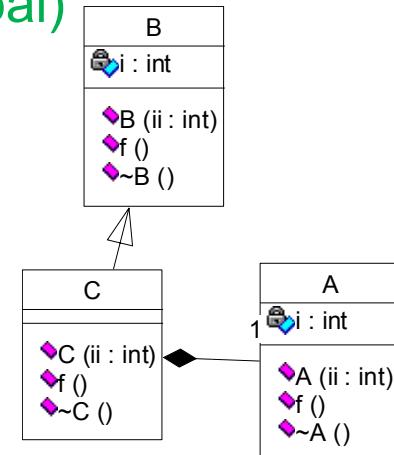
- Obviamente, podemos usar a composição e a herança simultaneamente, quando criamos novas classes
 - C “é um tipo de” B (herança)
 - C “possui” A ou A “é parte de” C (composição)



IV.3. Combinando composição e herança

```
class A {  
    int i;  
public:  
    A(int ii) : i(ii) {}  
    ~A() {}  
    void f() const {}  
};  
  
class B {  
    int i;  
public:  
    B(int ii) : i(ii) {}  
    ~B() {}  
    void f() const {}  
};
```

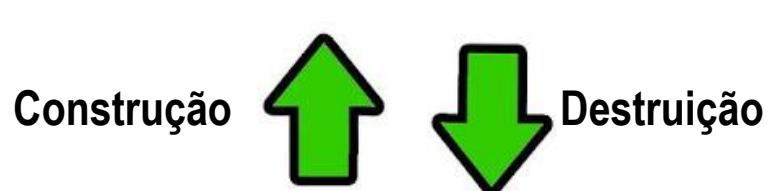
```
class C : public B { // C “é um tipo de” B  
    A a; // C “possui” A  
public:  
    C(int ii) : B(ii), a(ii) {} // constrói C de B e A  
    ~C() {} // chama ~B e ~A  
    void f() const {  
        a.f(); // chama f() de A (componente)  
        B::f(); // chama f() de B (pai)  
    }  
};  
  
int main() {  
    C c(47);  
}
```



OBS: Lembre-se da seqüência de chamada dos construtores: primeiro o da base (aqui = B), depois os dos atributos (aqui = a) e finalmente o código do construtor da classe derivada

IV.3. Combinando composição e herança

- Ordem de chamada de destrutores
 - Quando os objetos de classes derivadas deixam seu escopo, ocorre a chamada de destrutores existentes ao longo da hierarquia
 - A ordem de chamada dos destrutores é **inversa** à ordem de chamada dos construtores
 - 1) chamado o destrutor da classe derivada
 - 2) os atributos da classe derivada são destruídos
 - 3) o destrutor da(s) classe(s) base é(são) chamado(s)
 - 4) os atributos da(s) classe(s) base são destruídos



IV.3. Combinando composição e herança

- Herança de operadores
 - Com exceção do operador de **atribuição**, todos os operadores são herdados pelas classes derivadas (incluindo os operadores de conversão de tipo)
 - É necessário sobrepor o operador de atribuição para poder utilizá-lo na classe resultante!

IV.3. Combinando composição e herança

- Composição ou Herança?
 - Tanto a composição quanto a herança nos permitem a reutilização de código contido em outras classes
 - Ambas colocam sub-objetos dentro da nova classe
 - Ambas utilizam a seção de inicialização dos construtores para construir estes sub-objetos
 - Quais as diferenças entre elas?
Quando escolher uma ou outra?
 - **Herança:** as características da classe base estão na interface da classe filha
 - A nova classe é *um tipo* da classe base, podendo substituí-la
==> **subtipagem**.
 - **Composição:** a nova classe contém as características das classes usadas na composição, mas estas características geralmente não aparecem na interface
 - A nova classe não é um tipo da classe antiga,
mas **contém** objetos da classe

IV.3. Combinando composição e herança: herança privada

- Na **herança privada**, o que era público na classe base passa a ser privado na classe derivada
 - A classe derivada **não é um tipo** da classe base
 - A classe derivada é *implementada em termos da classe base!*
- Podemos expor alguns nomes na interface pública com a palavra chave **using**

```
class Pet {  
public:  
    char eat() const { return 'a'; }  
    int speak() const { return 2; }  
    float sleep() const { return 3.0; }  
    float sleep(int) const { return 4.0; }  
};  
class Goldfish : Pet { //Herança privada  
public:  
    using Pet::eat; //using torna público eat()  
    using Pet::sleep; //using torna público as  
};
```

```
int main() {  
    Goldfish bob;  
    bob.eat(); //Ok!  
    bob.sleep(); //Ok!  
    bob.sleep(1); //Ok!  
    //! bob.speak(); //Erro: private  
}
```

IV.3. Combinando composição e herança: atributos protegidos

- Até o momento, os únicos especificadores de acesso aos membros das classes que estudamos foram **public** e **private**
 - Porém, com a utilização de herança, a palavra chave **protected** passa a fazer sentido
- Tudo o que é privado é acessível apenas pelas funções da classe e por ninguém mais
 - E se quiséssemos que algo fosse privado para os usuários da classe, mas acessível para os membros da própria classe e de suas classes derivadas? → para isso usaríamos **protected**
- **Estratégia geral:** faça os atributos sempre **private**
 - Desta forma você preserva o direito de modificar a implementação das estruturas internas da classe, sem afetar outras partes do código (mesmo o código de classes derivadas)
 - Você pode fornecer acesso controlado a estes dados para as classes derivadas por meio de funções membro **protected**

IV.3. Combinando composição e herança: atributos protegidos

Exemplo:

```
include <fstream>
using namespace std;

class Base {
    int i;
protected:
    int read() const { return i; }
    void set(int ii) { i = ii; }
public:
    Base(int ii = 0) : i(ii) {}
    int value(int m) const { return m*i; }
};

class Derived : public Base {
    int j;
public:
    Derived(int jj = 0) : j(jj) {}
    void change(int x) { set(x); }
};
```

```
int main() {
    Derived d;
    d.change(10);
}
```

IV.3. Combinando composição e herança: herança protegida

- Como já vimos, o tipo mais comum de herança é a **pública**
 - O que era **público** na classe base continua **público** na derivada
 - O que era **protegido** na classe base continua **protegido** na derivada
=> A classe derivada é *um tipo* da classe base
- Na herança **privada**, todos os membros da classe base (públicos ou protegidos) se tornam privados na classe derivada
 - A classe derivada **não é um tipo** da classe base: a classe derivada é *implementada em termos da* classe base
- Uma herança **protegida**, significa *implementada em termos da* classe base para os usuários da classe e é *um tipo de* para as classes derivadas desta nova classe

IV.3. Combinando composição e herança: Resumo

- *Public* na classe derivada
 - Pode ser acessada diretamente por funções membro, funções *friend* e funções não membro
- *Protected* na classe derivada
 - Pode ser acessada diretamente por funções membro e funções *friend*
- *Private* na classe derivada
 - Pode ser acessada diretamente por funções membro e funções *friend*
- Oculta na classe derivada
 - Pode ser acessada por funções membro e funções *friend* por meio das funções membro *public* ou *protected* da classe básica

Tipos de Herança - Classe Derivada			
Classe Básica	<i>Public</i>	<i>Protected</i>	<i>Private</i>
<i>Public</i>	<i>Public</i>	<i>Protected</i>	<i>Private</i>
<i>Protected</i>	<i>Protected</i>	<i>Protected</i>	<i>Private</i>
<i>Private</i>	Oculta	Oculta	Oculta