Herança múltipla

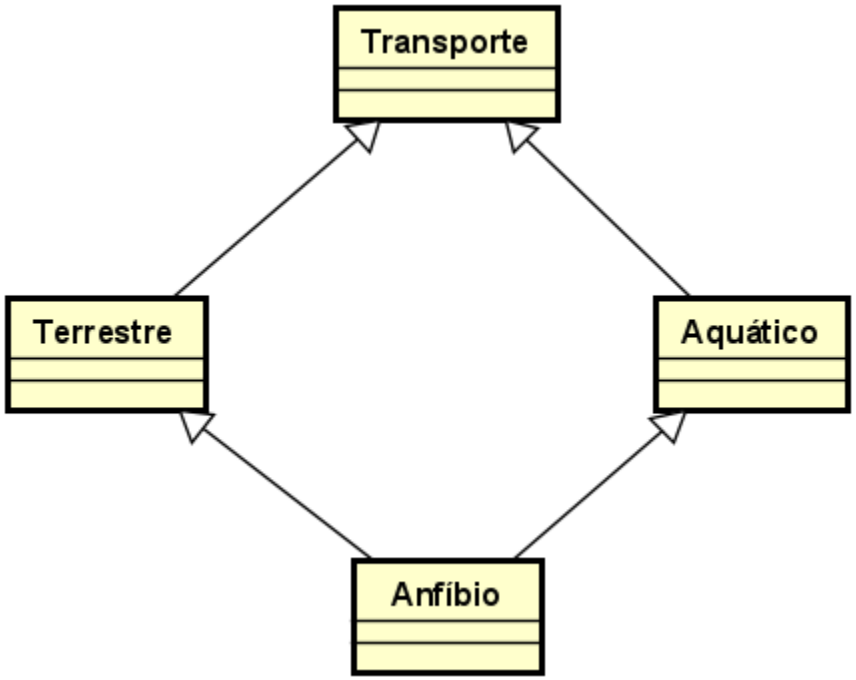
PROFESSOR: EDUARDO HABIB BECHELANE MAIA

HABIB@CEFETMG.BR

Herança múltipla

É possível uma Classe herdar de mais de uma classe:

- Herança múltipla



Herança múltipla

```
class Veiculo {
private:
   int posx;
   int posy;
public:
   void andar(int x);
   void parar();
   void virar(int angulo);
};
```

```
class Carro: public Veiculo {
public:
   void func();
   ...
};

class Navio: public Veiculo {
public:
   void func();
   ...
};

class Anfibio: public Carro,
public Navio {
   ...
};
```

Ambiguidade em herança múltipla

Suponha que na classe anfíbio não fosse implementado func. O que ocorreria?

- E se a classe anfíbio tentasse chamar a função func através de um objeto do tipo anfíbio?
  - Não saberia qual das duas funções func executar.
  - Pode-se solucionar esta ambiguidade usando o operador de resolução de escopo global func() para especificar qual função func utilizar:
- Mesmas ambigüidades podem existir com membros de dados.

Função construtora em Herança múltipla

A declaração da função construtora numa classe derivada de bases múltiplas deve especificar os argumentos para as funções construtoras de todas as classes base.

- A seguinte notação serve para uma função construtora deste tipo:

```
class Arquivo : public Hora, public Data
{
    Arquivo(char *nomeArquivo,int dd,int mm,
    int aaaa, int hh, int mn, int ss) : Hora(hh, mn, ss),
    Data (dd, mm, aaaa)
};
```

Função construtora em Herança múltipla

As funções construtoras para as classes base são chamadas primeiro.

A ordem de execução é a ordem com que as bases são declaradas como base na função derivada.

- `class Anfibio: public Carro, public Navio { ... }`
  - A ordem de execução das funções construtoras é:
    - Carro
    - Navio
    - Anfibio

Função construtora em Herança múltipla

E se a classe possuir um atributo de classe como membro?

- a função construtora desta classe será executada após as funções construtoras das classes base
  - antes da função construtora para a classe que está sendo definida.

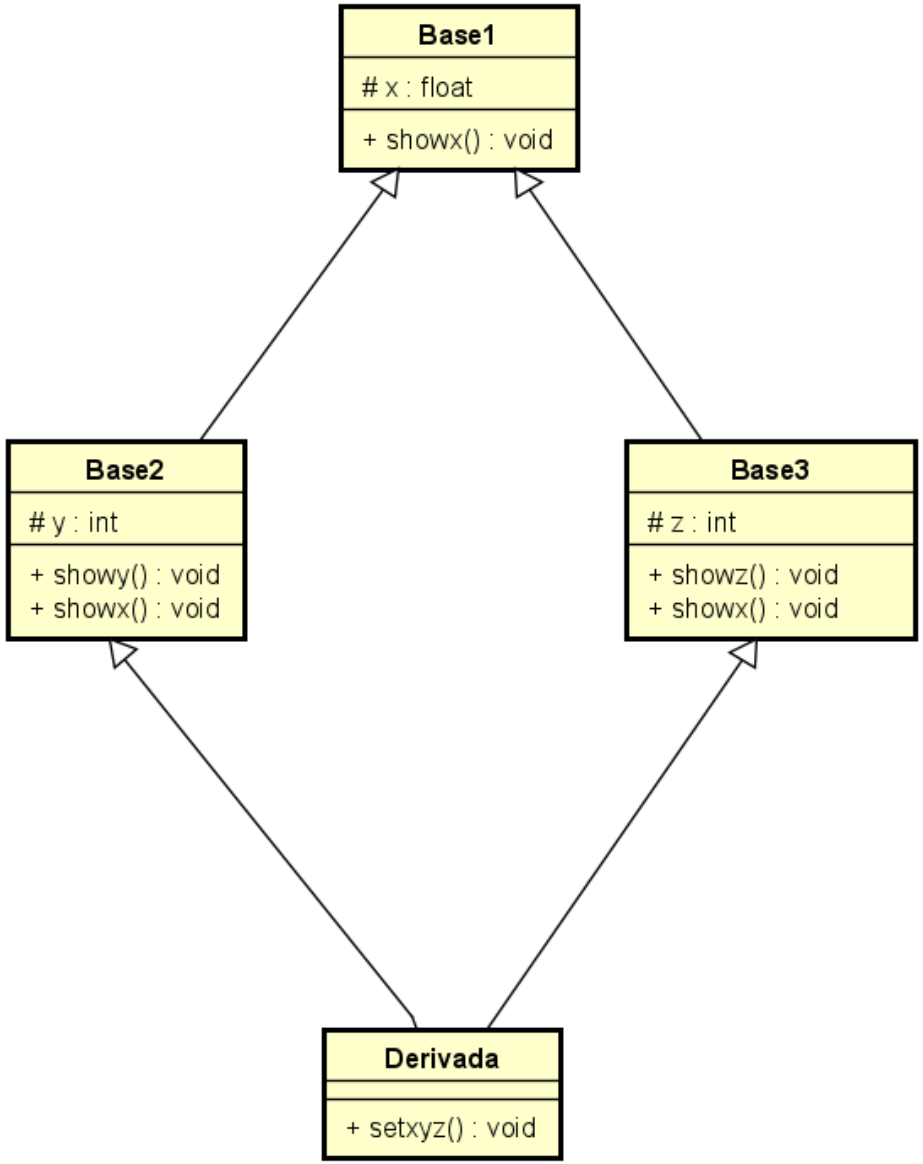
Função construtora em Herança múltipla

E se a classe possuir um atributo de classe como membro?

- Ex:

```
1   class Teste{
2       public:
3           teste(...)
4           {
5           }
6   }
7   class Anfibio: public Carro, public Navio {
8   public:
9       Teste x;
10      ...
11  }
```

A ordem de execução dos construtores das classes seria: Carro, Navio, Teste e Anfibio.



O problema do Diamante

Problema

A classe derivada herda o método showx() através de base2 e base3

- surge a ambigüidade quanto a qual membro de dados / função seria chamado?
- Isso confunde o compilador e exibe erro.

Como resolver????

DESAFIO

Solução

Solução

Para resolver essa ambigüidade quando base1 é herdada na classe **base2** e na classe **base3** , ela é declarada como **classe base virtual** colocando uma palavra-chave **virtual** como

Solução: https://onlinegdb.com/vft-6JEGS

Superclasse Virtual

- A declaração virtual para uma classe soluciona a ambiguidade
- Elimina a criação de dois endereços para os membros na superclasse base1 da hierarquia.
- Em uma analogia com membros estáticos
  - a superclasse virtual provoca o mesmo efeito para as instâncias superiores na hierarquia da herança múltipla

Herança múltipla

Outro exemplo.

- Quando se utiliza herança múltipla
  - Anfibio herda de Terrestre e Aquatico
    - Ambas estas classes herdam de Transporte,
      - C++ não sabe de qual das bases (Terrestre ou Aquatico) a classe Transporte deve ser considerada.

- Em C++, você deve tornar a classe base Transporte uma base virtual.
  - Garante que apenas uma cópia da classe base Transporte existirá

Em C++:

- https://onlinegdb.com/cWGC7JfgF

Problema do diamante

A Herança de Transporte nas classes Terrestre e Aquatico deve ser virtual

O Problema do diamante será resolvido e o compilador saberá como lidar com a herança ambígua da classe Transporte em Anfibio.

Na herança virtual:

- Classe base é apenas inicializada uma vez,
- independentemente de quantas vezes aparece na hierarquia de herança.
- Essa inicialização é feita pela classe mais derivada (no caso, Anfibio).
  - As classes intermediárias (Terrestre e Aquatico) não o farão automaticamente.

Chamar explicitamente o construtor da classe Transporte.

- Adicionar Transporte(marca, capacidade) antes de Terrestre e Aquatico na lista de inicialização

Herança múltipla em JAVA

Com composição

Java gerencia a memória automaticamente

- Não existe Herança múltipla em JAVA
- Não é necessário preocupar com alocação dinâmica e liberação de memória.
- Assim:
  - Não precisamos de desalocação explícita de memória (delete).
  - Usamos a palavra-chave extends para herança.
  - Usamos @Override para indicar que estamos substituindo um método da superclasse.
  - No caso do Anfibio, como o Java não suporta herança múltipla, uma abordagem é combinar composição e herança. Assim, o Anfibio estende Terrestre e possui um objeto Aquatico.
  - Código:
    - https://onlinegdb.com/d0f_ZpaXD

Com Interfaces

Java oferece uma alternativa para herança múltipla

- **Interfaces**: Em vez de herdar de várias classes, em Java você pode implementar várias interfaces.
- Como funciona:
  - Em Java, uma interface é uma referência de tipo, semelhante a uma classe, composta por constantes e métodos abstratos (ou métodos padrão a partir do Java 8).
  - Uma interface é uma forma de especificar o que as classes devem fazer, sem especificar como devem fazê-lo.
- Implementando Interfaces:
  - Uma classe pode implementar qualquer número de interfaces.
  - Você pode alcançar a "herança múltipla" em Java através do uso de interfaces, já que uma classe pode implementar mais de uma interface.

Com Interfaces

Exemplo: https://onlinegdb.com/3nRRAAqRZ

Anfibio implementa ambas as interfaces **Terrestre** e **Aquatico**

- permitindo que ela adote o comportamento de ambos
- sem a necessidade de herança múltipla direta de classes

Ao implementar uma interface:

- obrigatório implementar todos os métodos abstratos definidos na interface.
- Se um método não é implementado, o compilador lançará um erro.

Exemplo 1

1) Imagine que você está desenvolvendo um sistema para uma biblioteca. Você precisa modelar diferentes tipos de materiais que podem ser emprestados. Siga os passos abaixo:

A) Crie uma classe chamada Material. Esta será a classe base. Ela deve ter os seguintes atributos e métodos:

Atributos:

- titulo: título do material.
- autor: autor do material.

Métodos:

- descricao(): retorna uma string que diz "Este é um material da biblioteca".

B) Crie duas subclasses: Livro e DVD.

A classe Livro deve possuir um novo atributo:

- numero_de_paginas: número total de páginas do livro.

A classe DVD deve possuir um novo atributo:

- duracao: duração total em minutos.

C) Crie uma terceira classe chamada Audiolivro que herda tanto de Livro quanto de DVD. Esta classe representa um livro que foi gravado em áudio e pode ser ouvido como um DVD.

A classe Audiolivro deve possuir um novo atributo:

- narrador: pessoa que narra o audiolivro.

Teste as classes criadas. Crie objetos das classes Livro, DVD e Audiolivro, defina seus atributos e imprima as descrições usando o método descricao().

Exemplo 2

Imagine que você está desenvolvendo um sistema para gerenciar recursos de um parque de diversões. Precisamos modelar diferentes tipos de atrações, como montanhas-russas, carrosséis e casas mal-assombradas.

Atração: Esta será a classe base.

- **Atributos:**
  - nome: nome da atração.
  - capacidade: número máximo de pessoas por vez.
- **Métodos:**
  - info(): retorna uma string que diz "Esta é uma atração".

Mecânica:

- **Atributos:**
  - velocidade: velocidade da atração em km/h.

Tematica:

- **Atributos:**
  - tema: tema da atração (por exemplo: "piratas", "fantasmas").

MontanhaRussa e Carrossel: Ambas as classes herdam de Atração, Mecânica e Tematica.

Elas devem sobrescrever o método info(). Para a MontanhaRussa, pode retornar "Esta montanha-russa é temática de [tema] e atinge [velocidade] km/h" e para Carrossel, "Este carrossel tem o tema [tema] e move-se a [velocidade] km/h".