Residência em Software

Herança e Hierarquia de Classes

Professores:
Álvaro Coelho, Edgar Alexander, Esbel Valero e Hélder Almeida

INSTITUIÇÃO EXECUTORA
cepedi UESC

COORDENADORA
MCTI FUTURO Softex

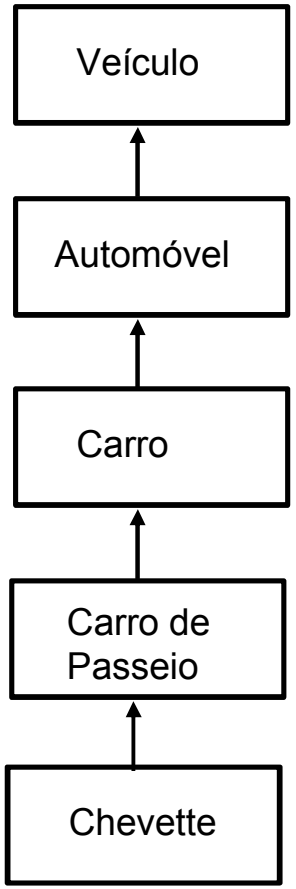
APOIO
MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO
GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

Herança

- Definição: generalização de um conceito
  - Ontologia do Domínio (Platão) – Pesquise!

Herança

- Definição: generalização de um conceito
  - Todo Chevette é um Carro de Passeio
  - Todo carro de passeio é um Carro
  - Todo Carro é um Automóvel
  - Todo Automóvel é um Veículo



Herança

- Claro que nem toda a hierarquia é necessária
- Domínio do Problema e Responsabilidades do Sistema
  - Domínio do Problema: quanto do que acontece na realidade efetivamente vai ser tratada pelo programa (sistema)?
    - Ex: Num sistema de venda de atacado, a verificação dos itens comprados por cada cliente não faz parte do domínio do problema
  - Responsabilidades do Sistema: quanto das atividades e dos processos serão efetivamente produzidos e/ou controlados pelo programa (sistema)?
    - Ex: No mesmo sistema de venda de atacado, a substituição de itens defeituosos ou vencidos (possivelmente) não faz parte das responsabilidades do sistema

Mais conceitos

- Superclasse
  - A classe que é mais geral
- Subclasse
  - A classe que é mais específica
- Ex. Motocicleta é subclasse de Automóvel. Caminhão também.
- Herança
  - Tudo o que existe na superclasse também existe (por herança) na subclasse.
- Ex. Automóvel possui um atributo RENAVAN. Logo, Motocicleta também possui, Caminhão também, Carro também.

Visibilidade

- Como ocorre com qualquer outro atributo ou método, os atributos e métodos de classes e superclasses também tem níveis de visibilidade
- Public: visível para todas as demais classes
- Private: não é visível para as demais classes
- Protected: é tratado como públic para as classes derivadas e private para as demais

Residência em Software

Exercício

- O que vai dar erro de compilação no exemplo ao lado?
- Porque?

```cpp
#include <iostream>


using namespace std;

class Gen {
    private:
        int x;
    protected:
        int y;
    public:
        int z;

};

class Esp: Gen {

    void func() {
        z = 1;
        y = 2;
        x = 3;
    }

};
```

Heranças public, protected e private

- Por default uma herança é public
  - Não altera a visibilidade dos membros
- Herança protected
  - Herda public e protected como protected
- Herança private
  - Herda public e protected como private

Residência em Software

Exercício

- O que vai dar erro de compilação nos exemplos ao lado?
- Porque?

```cpp
#include <iostream>


using namespace std;

class Gen {
    private:
        int x;
    protected:
        int y;
    public:
        int z;

};

class Esp: private Gen {

};

class Esp2: Esp {

    void func() {
        z = 1;
        y = 2;
    }

};
```

```cpp
#include <iostream>


using namespace std;

class Gen {
    private:
        int x;
    protected:
        int y;
    public:
        int z;

};

class Esp: protected Gen {

};

class Esp2: Esp {

    void func() {
        z = 1;
        y = 2;
    }

};
```

Constutores (e destrutores) de classes com herança

- Se a superclasse possui um construtor, é necessário que este seja executado!
- Se não houver parâmetros, é desnecessário explicitar
  - A invocação é automática
- Se houver parâmetros é necessário definir o construtor da superclasse

Exemplos

```
class Empregado {
    private:
        string nome;
        float salario;
    public:
        Empregado(string n, float s) {
            nome = n;
            salario = s;
        }


};

class Gerente: Empregado {
    private:
        string departamento;
    public:

        Gerente(string n, float s, string d):Empregado(n,s) {
            departamento = d;
        }

};
```

```
class Empregado {
    private:
        string nome;
        float salario;
    public:
        Empregado() {
            nome = "Default";
            salario = 0;
        }


};

class Gerente: Empregado {
    private:
        string departamento;
    public:

        Gerente(string d) {
            departamento = d;
        }

};
```

Funções Virtuais

- Podemos definir uma função apenas com sua declaração na superclasse
- Definir sua implementação na subclasse
- Polimorfismo (assunto a se ver)

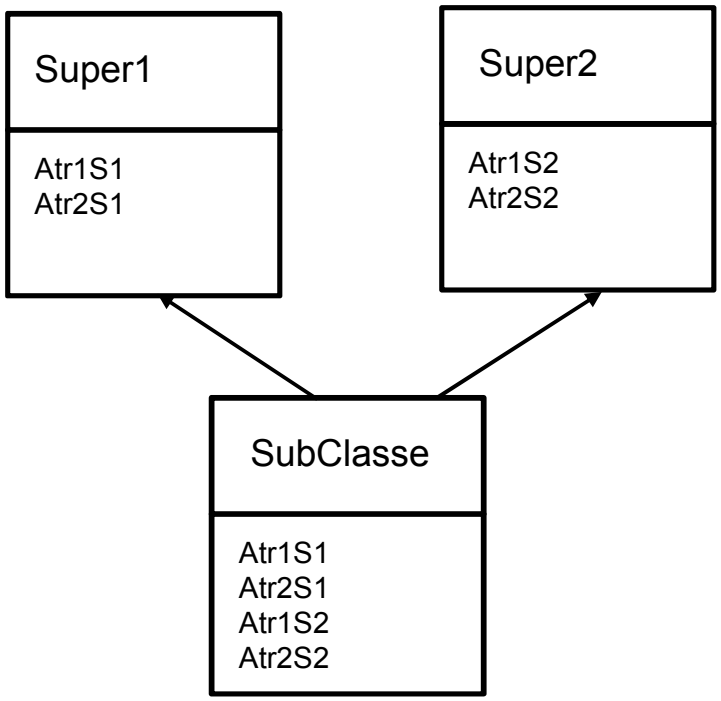
Exemplo

```
class Figura {
    private:

    public:
        float area();
        float perimetro();
};

class Quadrado: Figura {
    private:
        float lado;
    public:
        float area() {
            return pow(lado,2);
        }

        float perimetro() {
            return lado*4;
        }

};
```

```
class Triangulo: Figura {
    private:
        float base, altura, l1, l2, l3;
    public:
        float area() {
            return base*altura/2;
        }

        float perimetro() {
            return l1 + l2 + l3;
        }
};
```

Herança Múltipla

- ATENÇÃO: PRÁTICA ALTAMENTE QUESTIONÁVEL!
  - Use com sabedoria!
- É possível uma classe herdar de mais de uma superclasse.



Residência em Software

Exemplo

```
class Super1 {
    protected:
        int at1s1, at2s1;
};

class Super2 {
    protected:
        int at1s2, at2s2;
};

class sub: Super1, Super2 {
    public:
        void func() {
            at1s1 = 1;
            at2s1 = 2;
            at1s2 = 3;
            at2s2 = 4;
        }

};
```

Residência em Software

Exemplo

- Se houver ambigüidade, gera um erro de compilação!

```
class Super1 {
    protected:
        int at1s1, at2s1;
};

class Super2 {
    protected:
        int at1s1, at2s2;
};

class sub: Super1, Super2 {
    public:
        void func() {
            at1s1 = 1;
            at2s1 = 2;

            at2s2 = 4;
        }

};
```

Exercício 1

- Crie uma superclasse chamada Animal com atributos nome, idade e um método fazerSom().
- Em seguida, crie uma subclasse chamada Cachorro que herda de Animal e substitua o método fazerSom() para que ele imprima "Au Au!" quando chamado.
- Crie objetos de ambas as classes e teste seus métodos (função main()).

Residência em Software

Exercício 2

- Crie uma superclasse chamada Forma com um método calcularArea().
- Em seguida, crie duas subclasses: Retangulo e Circulo (com seus atributos e construtores específicos) que herdam de Forma
  - Implemente o método calcularArea() para calcular a área de um retângulo e de um círculo, respectivamente.
- Crie um programa que permita ao usuário criar objetos de ambas as subclasses e chame o método calcularArea() em cada um deles para calcular suas áreas.

Residência em Software

Exercício 3

- Crie uma superclasse chamada ItemBiblioteca com atributos título, autor e número de cópias disponíveis.
- Em seguida, crie duas subclasses, Livro e DVD, que herdam de ItemBiblioteca.
    - Adicione propriedades específicas de cada tipo de item, como número de páginas para livros e duração para DVDs.
- Faça um programa que crie objetos de ambas as subclasses e teste o seu modelo.