Programação Orientada a Objetos em C++

Herança

Agostinho Brito

2020

Tal pai, tal filho...

- Na natureza, o conceito de herança é algo bem comum.
- O Pai transmite sua característica genética ao filho, tornando-se semelhante em muitas funcionalidades.
- Os filhos, entretanto, apresentam aparência e comportamento diferentes dos pais, sinal de que sofreram algum tipo de especialização.
- Dá-se a esse processo de um indivíduo herdar as características do outro o nome de herança.
- E, assim como outros conceitos do mundo real, esse também é implementado C++.

Analisando um par de classes

```
1  class Equipamento{
2    char nome[100];
3    char fabricante[100];
4    float preco;
5  public:
6    void setNome(const char *_nome);
7    void setFabricante(const char *
         _fabricante);
8    void setPreco(float _preco);
9    char* getNome(void);
10   char* getFabricante(void);
11   float getPreco(void);
12 };
```

```
1  class Motor{
2    char nome[100];
3    char fabricante[100];
4    float preco;
5    float potencia;
6    float velocidade;
7  public:
8    void setNome(const char *_nome);
9    void setFabricante(const char *
         _fabricante);
10   void setPreco(float _preco);
11   void setPotencia(float _potencia);
12   void setVelocidade(float _velocidade);
13   char* getNome(void);
14   char* getFabricante(void);
15   float getPreco(void);
16   float getPotencia(void);
17   float getVelocidade(void);
18 };
```

Usando herança

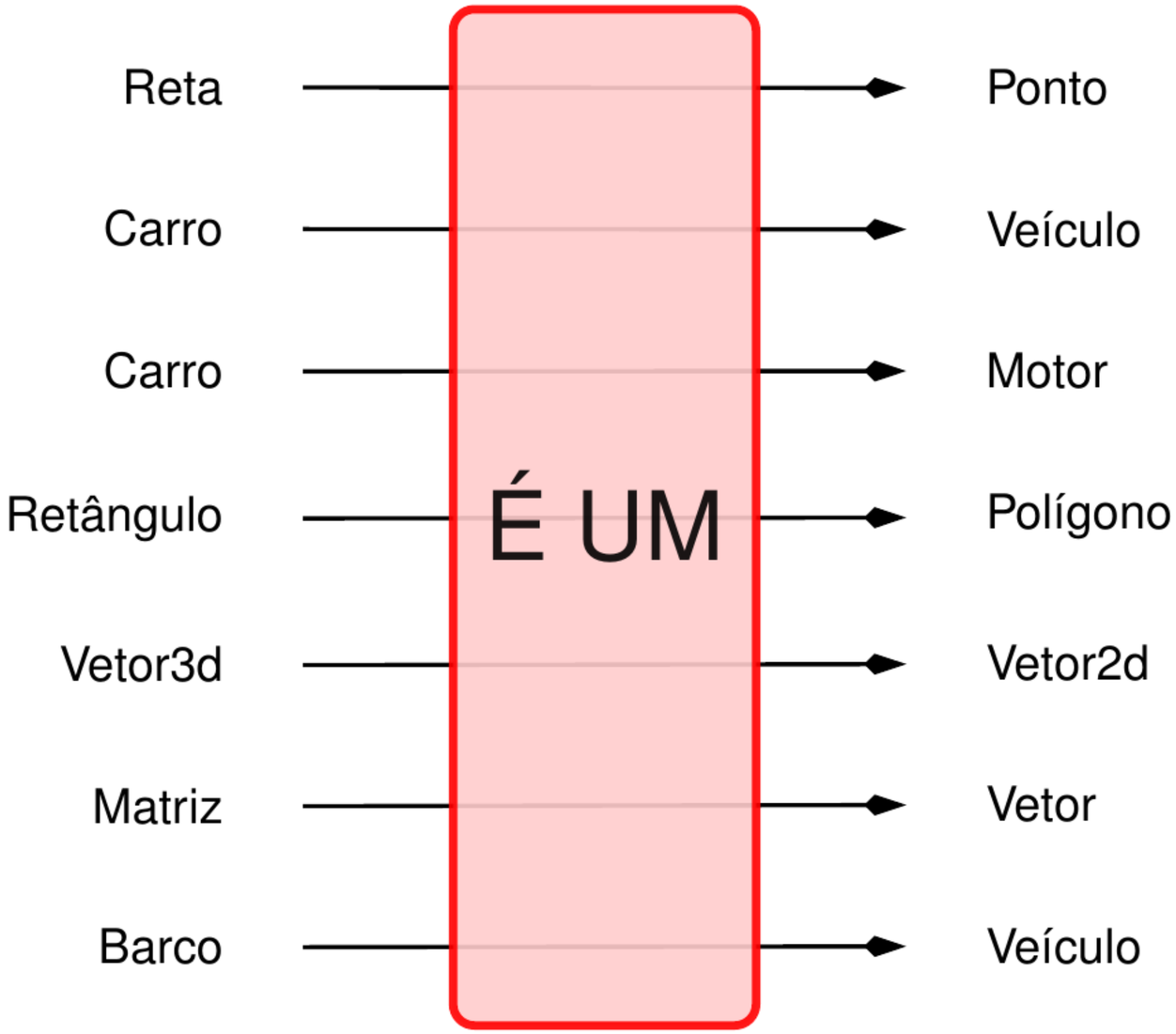
- Não é difícil perceber que as classes `Equipamento` e `Motor` compartilham estruturas em comum.
- Ainda, ambas as classes são intimamente relacionadas. Em verdade, a classe `Motor` é uma especialização da classe `Equipamento`

Quando usar herança?

Quando se diz que um objeto da classe A possui relação do tipo É UM com um objeto da classe B, evidencia-se uma relação de herança entre estas classes.

Motor É UM Equipamento

Usando herança



?

Usando herança em C++

```
1  class Equipamento{
2    char nome[100];
3    char fabricante[100];
4    float preco;
5  public:
6    void setNome(const char *_nome);
7    void setFabricante(const char *
         _fabricante);
8    void setPreco(float _preco);
9    char* getNome(void);
10   char* getFabricante(void);
11   float getPreco(void);
12 };
```

```
1  class Motor : public Equipamento{
2    float potencia;
3    float velocidade;
4  public:
5    void setPotencia(float _potencia);
6    void setVelocidade(float _velocidade);
7    float getPotencia(void);
8    float getVelocidade(void);
9  };
```

```
class Derivada : public Base {};
```

Nome da nova classe

Tipo de herança

Usando herança em C++

Diz-se que quando uma classe B herda de uma classe A...

B é chamada de subclasse ou classe derivada.

A é chamada de superclasse ou classe base.

 Praticando herança em C++...

Obrigado

Efeitos da herança

Efeitos da herança

- Todas as propriedades da classe base são herdadas e podem até ser usadas pela classe herdeira, observadas algumas restrições.
- No processo de herança o especificador de acesso `protected` implica que as herdeiras podem acessar as propriedades sob essa tutela.

```
1 class Equipamento{
2 private:
3   char nome[100]; // SOMENTE ACESSIVEL POR ESTA CLASSE E AMIGAS
4 protected:
5   float preco;  // ACESSIVEL TAMBEM PELAS HERDEIRAS E AMIGAS
6 }
```

Efeitos da herança

- O especificador de herança, que fica logo após os `:` determina a visibilidade das propriedades ou métodos:

Especificador de herança	Tipo de Interface		
	`public`	`private`	`protected`
`public`	`public`	`private`	`protected`
`private`	`private`	`private`	`private`
`protected`	`protected`	`private`	`protected`

- Apesar das possibilidades, o especificador `public` é o mais comum, pois permite tratar as propriedades mantendo o comportamento conhecido da classe base.

Efeitos da herança

- Todos os métodos definidos na superclasse são herdados, MENOS os construtores e o destrutor, que precisam ser desenvolvidos, se necessário.

!

Quando um programa prepara uma classe derivada, PRIMEIRO ele constrói o corpo do objeto da classe base. Portanto, algum construtor da classe base precisa ser chamado a priori.

!

Quando um programa finaliza uma classe derivada, PRIMEIRO ele destrói o corpo do objeto da classe derivada para só então destruir o da classe base.

- O construtor normalmente chamado na classe base é o construtor `default`.

Verificando efeitos da herança...

Usando listas de especificadores

- Caso o construtor `default` não exista na classe base, ou seja necessário chamar outro construtor, utiliza-se uma lista de inicializadores seguindo o construtor da classe derivada.

```
1  #include <iostream>
2  class Base{
3  public:
4    Base(int a){
5      std::cout << "Construtor Base: " << a << "\n";
6    }
7  };
8  class Derivada : public Base{
9  public:
10   Derivada(int a) : Base(a-1){
11     std::cout << "Construtor Derivada: " << a << "\n";
12   }
13 };
```

Atribuição de objetos entre classes

```
1  class Base{
2  protected:
3    int a, b;
4  };
5  class Derivada : public Base{
6    int c;
7  };
8  int main(void){
9    Base b;
10   Derivada d;
11   b = d;
12   d = b; // Erro! "d" possui parte indefinida.
13 }
```

- Dessa forma, a atribuição não é permitida, pois o compilador não vê como preencher as propriedades faltantes.
- Porém, a operação de atribuição pode ser necessária...

Atribuição de objetos entre classes

- Implemente a sobrecarga do método `operator=()`

```
1  class Base{
2  protected:
3    int a, b;
4  };
5  class Derivada : public Base{
6    int c;
7  public:
8    void operator=(Base &x){
9      a = x.getA();  b = x.getB();
10     c = 0;
11   }
12 };
13 int main(void){
14   Base b;
15   Derivada d;
16   b = d;
17   d = b; // Ok! Sobrecarga ativada
18 }
```

Obrigado

Herança múltipla

- Embora não seja comum, C++ provê o uso do recurso de herança múltipla.
- Assemelha-se ao uso da herança simples, porém a subclasse herda de várias superclasses diferentes

```
class A{...};
class B{...};
class C: public A, public B{...};
```

Praticando herança múltipla em C++...

Obrigado