PCS3111

Laboratório de Programação Orientada a Objetos para Engenharia Elétrica

Aula 3: Conceitos Básicos de OO

Escola Politécnica da Universidade de São Paulo

Agenda

1. Tipo Abstrato de Dados
2. Conceitos Básicos de OO
3. Programação OO em C++

Tipo Abstrato de Dados

Programação

- Programas pequenos podem ser uma sequência de condições e laços
  - Mas como organizar um software **maior**?
- Uma forma simples: organizá-lo em *funções*
- Mas como lidar com *muitas* funções?
  - Conceito de **Módulo**
    - *Pedaços de código* que podem ser implementados e compilados independentemente
    - Permite isolar os erros e eliminar redundâncias
    - Facilita o gerenciamento

Programação

- Mas quais funções definir? Como organizá-las em módulos?
  - É preciso pensar de uma outra forma
  - Uma ideia: **tipo abstrato de dados (TAD)**

Tipo de Dados

- Define um intervalo de **valores** e as **operações** sobre esses valores
  - *Exemplo*
    - Inteiro
      - *Operações*: soma, subtração, divisão, multiplicação
    - String
      - *Operações*: concatenar, obter uma substring, obter um caractere

- Cada linguagem define um conjunto de tipos *básicos*

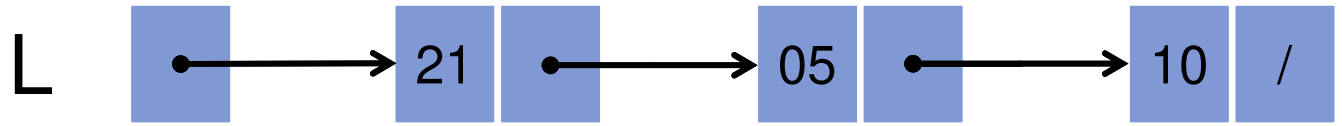
Tipo Abstrato de Dados (TAD)

- Técnica de projeto
  - Permite definir novos tipos
  - Encapsula uma coleção de **dados** e um conjunto de **operações sobre esses dados**
  - Abstrato: independe da representação dos dados
    - Não importa como as operações são implementadas e como os dados são armazenados
  - Independente de linguagem de programação

- Incorpora princípios de modularização
- Pode ter várias implementações diferentes

Exemplo

- Lista ligada



  - Dados
    - Conjunto de informações
  - Operações
    - Procurar, inserir e remover

- Quem usa a estrutura não precisa saber os *detalhes*
  - (Quais os dados e como as operações funcionam)
  - Somente as operações são relevantes: **abstração**

Tipo Abstrato de Dados (TAD)

- Pode-se especificar *textualmente* um TAD
  - Especificar os dados e as operações

- A linguagem de programação pode prover mecanismos para especificar um TAD
  - **Orientação a Objetos**

Conceitos Básicos de OO

Orientação a Objetos

- Organiza o software pelos conceitos do domínio do problema
  - *Abstração* do mundo real

- Combina os dados e as funções que os manipulam em uma *unidade*: **objeto**
  - Cada objeto tem um conjunto de responsabilidades bem definida
  - Métodos do objeto são, normalmente, a **única forma** de acessar os seus dados

Exemplo

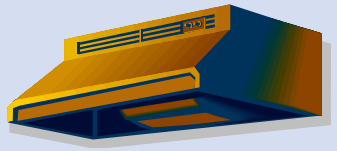
- Automação residencial

Sensor de presença

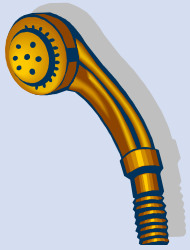
Sensor de temperatura

Sensores

Ventilador



Ar Condicionado



Chuveiro

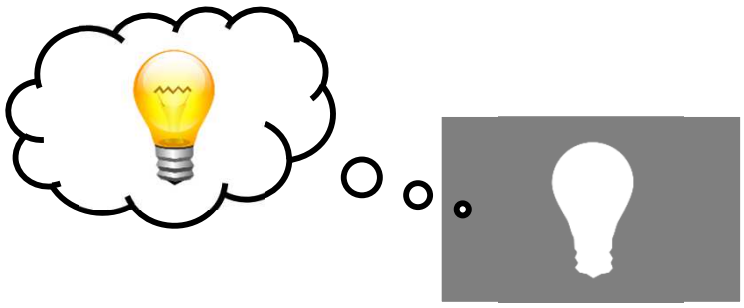
Máquina de lavar roupa

Lâmpada

Equipamentos

Conceitos Básicos

- **Classe**
  - Implementa um **tipo abstrato de dado**
  - Representa as características comuns a um conjunto de objetos
    - Abstração
  - A classe é um *molde* de objetos daquele tipo



Classe lâmpada

→ Quais são os comportamentos comuns aos objetos?

→ Quais são os possíveis estados dos objetos?

Conceitos Básicos

- **Objeto**
  - Objeto é uma **instância** de uma classe
  - Elemento do sistema computacional
    - Provê serviços
  - Características
    - Comportamento, estado e identidade (unicidade)
  - *Exemplo*: automação residencial

Lâmpada

Comportamento
Ligar, desligar, programar, ver estado

Estado
Se está ligada, horário agendado

Identidade*
Lâmpada da sala

Conceitos Básicos

- **Atributo**
  - Propriedades de uma classe
  - Representa os estados
    - Cada objeto tem um **valor** próprio para cada atributo
- **Operação**
  - Serviços disponibilizados por objetos de uma determinada classe
    - “O que o objeto pode fazer?”
  - Podem mudar o estado do objeto
    - Todos objetos de uma classe provêm as **mesmas** operações
- **Método**
  - Implementação de uma operação

Programação OO em C++

Programação OO em C++

- Exemplo de OO em C++

EX01

```
 1  #include <iostream>
 2  using namespace std;
 3
 4  class Lampada {
 5  public:
 6    bool acesa = false;
 7    void ligar();
 8    void desligar();
 9    void imprimir();
10  };
…
24  int main() {
25    Lampada *l1 = new Lampada;
26    Lampada *l2 = new Lampada
…
36  }
```

Definição da classe

Classe

Atributo

Métodos

Criação de um objeto Lampada

Programação OO em C++

- `Lampada` é uma classe (tipo de variável)
  - `l1` e `l2` referenciam objetos (ou instâncias) da classe `Lampada`
  - Cada objeto da classe `Lampada` possui um valor para o atributo `acesa`
    - No início, a lâmpada está apagada (`acesa = false`)
  - Cada `Lampada` possui métodos `ligar`, `desligar` e `imprimir`
    - Em C++, método == função-membro
  - (Veremos o significado de `public` na Aula 4)

- **Mas como os métodos funcionam?**

Programação OO em C++

EX01

```
...
 4  class Lampada {
 5  public:
 6    bool acesa = false;
 7    void ligar();
 8    void desligar();
 9    void imprimir();
10  };
11
12  void Lampada::ligar() {
13    acesa = true;
14  }
15
16  void Lampada::desligar() {
17    acesa = false;
18  }
19
20  void Lampada::imprimir() {
21    cout << acesa << endl;
22  }
```

Definição da classe

Operador de resolução de escopo

Implementação da classe

Programação OO em C++

- C++ separa a definição da classe de sua implementação
  - (Em outras linguagens OO isso pode ser diferente)
  - **Vantagem**: diminuição do tempo de compilação
    - (Mais detalhes na próxima aula)

- **Mas como usar os objetos?**

Programação OO em C++

- Como executar métodos para um objeto?
  - *Para qual objeto queremos executar o método?*
  - "->": operador de acesso a um membro da classe

Chamando os métodos para l1

Chamando o atributo de l1

Chamando os métodos para l2

EX01

```
24  int main() {
25    Lampada *l1 = new Lampada;
26    Lampada *l2 = new Lampada;
27    l1->imprimir();
28    l1->ligar();
29    l1->imprimir();
30    cout << l1->acesa << endl;
31
32    l2->ligar();
33    l2->imprimir();
34    l2->desligar();
35    l2->imprimir();
36    return 0;
37  }
```

Saída

```
0
1
1
1
0
```

Convenções

- **Nome da classe**: substantivo e singular
  - Representa um conceito
  - Convenção de nomes: *upper* CamelCase
    - *Exemplo*: `ContaCorrente`, `Pedido`, `Lampada`
- **Nome do atributo**: substantivo ou verbos representando estado
  - Convenção de nomes: *lower* CamelCase
    - *Exemplo*: `nome`, `id`, `telefone`, `preco`, `isAtivo`, `cancelado`, `processandoPedido`
- **Nome do método**: verbo (em geral infinitivo)
  - Convenção de nomes: *lower* CamelCase
    - *Exemplo*: `retirar`, `adicionarProduto`, `alugar`, `embaralhar`

Exemplo

- Outra classe: Relogio

EX02

```
 1  #include <iostream>
 2  using namespace std;
 3
 4  class Relogio {
 5  public:
 6    int hora = 0;
 7    int minuto = 0;
 8    void imprimir();
 9  };
10
11  void Relogio::imprimir() {
12    cout << hora << ":" << minuto << endl;
13  }
```

Exemplo

- Uma classe pode ser usada como um novo tipo

EX02

```
15  class Lampada {
16  public:
17    Relogio *horaDeLigar;
18    bool acesa = false;
19    void ligar();
20    void desligar();
21    void imprimir();
22    void atualizar(int horaAtual);
23  };
…
33  void Lampada::atualizar(int horaAtual) {
34    if (horaAtual >= horaDeLigar->hora) {
35      ligar();
36      cout << "Ligado as ";
37      horaDeLigar->imprimir();
38    }
39  }
```

Atributo do tipo Relógio

Atualiza a lâmpada e depende da hora atual (tem um parâmetro)

Exemplo

```
45  int main() {
46    Lampada *sala = new Lampada;
47    Relogio *r = new Relogio;
48    r->hora = 10;
49    r->minuto = 05;
50    sala->horaDeLigar = r;
51
52    sala->atualizar(9);
53    sala->imprimir();
54
55    sala->atualizar(10);
56    sala->imprimir();
57    return 0;
58  }
```

Inicializando o relógio (Acessando os atributos)

Definido a hora de ligar

Passando o parâmetro

EX02

- Um objeto da classe Relogio é criado na linha 47 e guardado no atributo horaDeLigar do objeto sala na linha 50
  - Assim, não precisa ser passado como parâmetro para o método atualizar (linha 33)

Exemplo

- Outro método na classe Relógio: `inicializar`
  - Evita que hora e/ou minuto seja inválido

EX03

```
 4  class Relogio {
 5  public:
 6    int hora = 0;
 7    int minuto = 0;
 8    void inicializar(int hora, int minuto);
 9    void imprimir();
10  };
11
12  void Relogio::inicializar(int hora, int minuto) {
13    if (hora < 0 || hora > 23) hora = 0;
14    else hora = hora;
15
16    if (minuto < 0 || minuto > 59) minuto = 0;
17    else minuto = minuto;
18  }
...
```

?

Como diferenciar o *atributo* do *parâmetro*?

Referência para o Objeto Atual

- Palavra reservada `this`
  - Referência para o objeto atual
    - Aponta para o objeto que foi chamado para executar o método
    - Pode ser usado para acessar membros do objeto
    - *Exemplo*:

```
12  void Relogio::inicializar(int hora, int minuto) {
13    if (hora < 0 || hora > 23) this->hora = 0;
14    else this->hora = hora;
15
16    if (minuto < 0 || minuto > 59) this->minuto = 0;
17    else this->minuto = minuto;
18  }
...
```

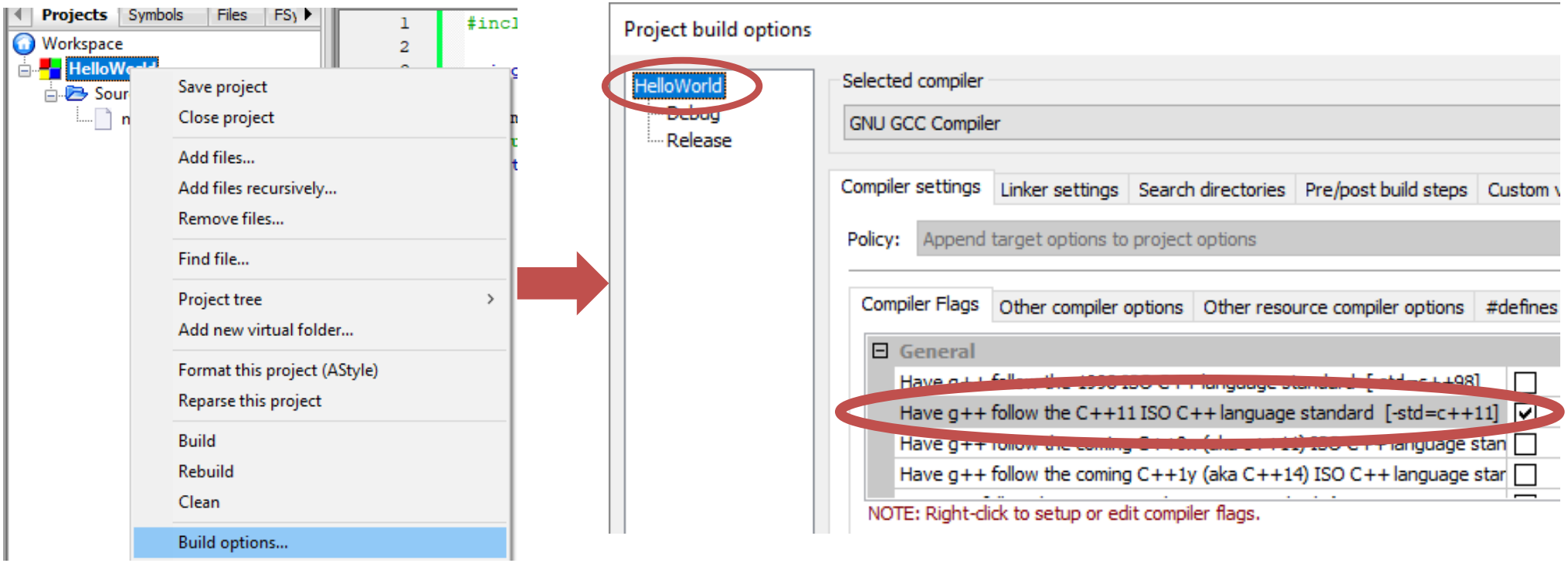
EX03

Acesso ao atributo `minuto`

Parâmetro `minuto`

Cuidado

- No Code::Blocks use sempre o C++11
  - Clique no nome do projeto com o botão direito do mouse: Build Options
    - Escolha o projeto e a opção “Have g++ follow the C++11...”



Bibliografia

- BUDD, T. **An Introduction to Object-Oriented Programming**. 3rd Edition. Addison-Wesley. 2001. Cap. 1 e 4.
- LAFORE, R. **Object-Oriented Programming in C++**. 4th Edition. SAMS. 2002. Cap. 6.