Programação Orientada a Objetos para Redes de Computadores

Prof. Miguel Elias Mitre Campista

http://www.gta.ufrj.br/~miguel

PARTE 2

Programação em C++ - Classes e Objetos

Linguagem de Programação C++

- Linguagem Imperativa, estruturada e orientada a objetos
  - Oferece:
    - Reuso
    - Modularidade
    - Rapidez de desenvolvimento
    - Correção de código
    - Facilidade de compreensão e modificação
    - Baixo custo de desenvolvimento

Linguagem de Programação C++

- Estruturada
  - Classes e funções
- *C++ standard library*
  - Coleção de classes e funções existentes
- Abordagem de construção de blocos de programação para criar novos programas
  - Possível com a característica de modularidade e reuso

Linguagem de Programação C++

- Simplificação de projetos
  - Possibilita enfoque estruturado para o desenvolvimento de programas para computadores
- Programas em C++ processam informações e exibem resultados

- C++ permite apenas tradução
  - Compilador: g++ (Programas *.cpp, *.cc, *.cxx e *.C)
    - Compila o código
      - `g++ -Wall <arq-codigo> -o <arq-compilado>`

É possível usar o gcc?

Linguagem de Programação C++

- Primeiros programas em C++
  - Exibição de mensagens
  - Obtenção de informações do usuário
  - Execução de cálculos aritméticos
  - Tomada de decisões

Linguagem de Programação C++

- Primeiros programas em C++
  - Exibição de mensagens
  - Obtenção de informações do usuário
  - Execução de cálculos aritméticos
  - Tomada de decisões

Como ficariam esses programas em C++?

Primeiro Exemplo em C++

- Programa simples:
  - Imprime uma linha do texto
    - Ilustra vários recursos importantes da linguagem C++

Primeiro Exemplo em C++

- Programa: **HelloWorld.cpp**

```
// Primeiro exemplo em C++
// Autor: Miguel Campista

#include <iostream>

int main () {
        std::cout << "Hello, world!";
        return 0;
}
```

Primeiro Exemplo em C++

- Programa: **HelloWorld.cpp**

```
// Primeiro exemplo em C++
// Autor: Miguel Campista

#include <iostream>

int main () {
        std::cout << "Hello, world!";
        return 0;
}
```

Diretiva de pré-processamento para incluir o arquivo de cabeçalho de fluxo de entrada e saída

Primeiro Exemplo em C++

- Programa: **HelloWorld.cpp**

```
// Primeiro exemplo em C++
// Autor: Miguel Campista

#include <iostream>

int main () {
        std::cout << "Hello, world!";
        return 0;
}
```

Diretiva de pré-processamento para incluir o arquivo de cabeçalho de fluxo de entrada e saída

Primeiro Exemplo em C++

- Programa: **HelloWorld.cpp**

```
// Primeiro exemplo em C++
// Autor: Miguel Campista

#include <iostream>

int main () {
        std::cout << "Hello, world!";
        return 0;
}
```

Operador de inserção de fluxo

Primeiro Exemplo em C++

- Programa: **HelloWorld.cpp**

```
// Primeiro exemplo em C++
// Autor: Miguel Campista

#include <iostream>

int main () {
        std::cout << "Hello, world!";
        return 0;
}
```

O "`std::`" é necessário sempre que se usa uma função definida por uma diretiva de pré-processador. No caso, o "`#include<iostream>`"

Primeiro Exemplo em C++

- Programa: **HelloWorld.cpp**

```
// Primeiro exemplo em C++
// Autor: Miguel Campista

#include <iostream>

int main () {
        std::cout << "Hello, world!";
        return 0;
}
```

```
shell>$ g++ -Wall HelloWorld.cpp –o hello
shell>$ ./hello
Hello, world!
shell>$
```

  - Compilação: `g++ -Wall HelloWorld.cpp –o hello`

Namespace std

- O uso do "`std::`"
  - Especifica que se deve usar um nome que pertence ao "namespace" std
  - Pode ser removido por meio de instruções `using`

- Objeto de fluxo de saída padrão (*standard output stream object*) do namespace std
  - `std::cout`
    - Está "conectado" à tela
    - É definido no arquivo de cabeçalho de fluxo de entrada/saída <iostream>

Operador de inserção de fluxo <<

- O valor à direita (operando da direita) é inserido no operando da esquerda.
  - Ex.: `std::cout << "Hello";`
    - Insere a string "Hello" na saída-padrão
      - Exibe na tela

Segundo Exemplo em C++

```
/*
 * Segundo exemplo em C++
 * Autor: Miguel Campista
 */
#include <iostream>

int main()
{
    // Declaração de variáveis
    int numero1;
    int numero2;
    int soma;

    std::cout << "Entre com o primeiro inteiro: ";
    std::cin >> numero1; // Lê o primeiro inteiro inserido pelo teclado

    std::cout << "Entre com o segundo inteiro: ";
    std::cin >> numero2; // Lê o segundo inteiro inserido pelo teclado

    soma = numero1 + numero2;

    std::cout << "A soma eh: " << soma << std::endl; // Exibe o resultado na tela


    return 0;
}
```

Segundo Exemplo em C++

```
/*
 * Segundo exemplo em C++
 * Autor: Miguel Campista
 */
#include <iostream>

int main()
{
    // Declaração de variáveis
    int numero1;
    int numero2;
    int soma;

    std::cout << "Entre com o primeiro inteiro: ";
    std::cin >> numero1; // Lê o primeiro inteiro inserido pelo teclado

    std::cout << "Entre com o segundo inteiro: ";
    std::cin >> numero2; // Lê o segundo inteiro inserido pelo teclado

    soma = numero1 + numero2;

    std::cout << "A soma eh: " << soma << std::endl; // Exibe o resultado na tela


    return 0;
}
```

Declaração de variáveis inteiras

Segundo Exemplo em C++

```
/*
 * Segundo exemplo em C++
 * Autor: Miguel Campista
 */
#include <iostream>

int main()
{
    // Declaração de variáveis
    int numero1;
    int numero2;
    int soma;

    std::cout << "Entre com o primeiro inteiro: ";
    std::cin >> numero1; // Lê o primeiro inteiro inserido pelo teclado

    std::cout << "Entre com o segundo inteiro: ";
    std::cin >> numero2; // Lê o segundo inteiro inserido pelo teclado

    soma = numero1 + numero2;

    std::cout << "A soma eh: " << soma << std::endl; // Exibe o resultado na tela


    return 0;
}
```

Operador de extração de fluxo para obter entrada do teclado

Segundo Exemplo em C++

```
/*
 * Segundo exemplo em C++
 * Autor: Miguel Campista
 */
#include <iostream>

int main()
{
    // Declaração de variáveis
    int numero1;
    int numero2;
    int soma;

    std::cout << "Entre com o primeiro inteiro: ";
    std::cin >> numero1; // Lê o primeiro inteiro inserido pelo teclado

    std::cout << "Entre com o segundo inteiro: ";
    std::cin >> numero2; // Lê o segundo inteiro inserido pelo teclado

    soma = numero1 + numero2;

    std::cout << "A soma eh: " << soma << std::endl; // Exibe o resultado na tela


    return 0;
}
```

O manipulador de fluxo "`std::endl`" gera uma nova linha e, em seguida, esvazia o buffer de saída

Segundo Exemplo em C++

```
/*
 * Segundo exemplo em C++
 * Autor: Miguel Campista
 */
#include <iostream>

int main()
{
    // Declaração de variáveis
    int numero1;
    int numero2;
    int soma;

    std::cout << "Entre com o primeiro inteiro: ";
    std::cin >> numero1; // Lê o primeiro inteiro inserido pelo teclado

    std::cout << "Entre com o segundo inteiro: ";
    std::cin >> numero2; // Lê o segundo inteiro inserido pelo teclado

    soma = numero1 + numero2;

    std::cout << "A soma eh: " << soma << std::endl; // Exibe o resultado na tela


    return 0;
}
```

Operações de inserção de fluxo por concatenação, encadeamento ou em cadeia

Segundo Exemplo em C++

```
/*
 * Segundo exemplo em C++
 * Autor: Miguel Campista
 */
#include <iostream>

int main()
{
    // Declaração de variáveis
    int numero1;
    int numero2;
    int soma;

    std::cout << "Entre com o primeiro inteiro: ";
    std::cin >> numero1; // Lê o primeiro inteiro inserido pelo teclado

    std::cout << "Entre com o segundo inteiro: ";
    std::cin >> numero2; // Lê o segundo inteiro inserido pelo teclado

    soma = numero1 + numero2;

    std::cout << "A soma eh: " << soma << std::endl; // Exibe o resultado na tela


    return 0;
}
```

```
shell>$ g++ -Wall ex2.cpp –o ex2
shell>$ ./ex2
Entre com o primeiro inteiro: 1
Entre com o primeiro inteiro: 2
A soma eh: 3
shell>$
```

Objeto de Fluxo de Entrada

- `std::cin` do namespace std
  - Em geral está conectado ao teclado
  - Operador de extração de fluxo ">>"
    - Espera o usuário inserir um valor e pressionar **`Enter`**
    - Armazena o valor na variável à direita do operador
      - Converte o valor no tipo de dado da variável
    - Ex.: `std::cin >> numero1;`
      - Lê um inteiro digitado no teclado
      - Armazena o inteiro na variável numero1
  - Programas devem validar os valores de entrada
    - Evitam que informações errôneas afetem o programa

Manipulador de Fluxo "std::endl"

- Gera um nova linha
- Esvazia o buffer de saída
  - Alguns sistemas armazenam dados de saída até que um determinado limiar seja atingido
    - O `std::endl` força os dados de saída armazenados a serem exibidos no momento de sua chamada

Terceiro Exemplo em C++

```
#include <iostream>

using namespace std;

int main()
{
    int numero1, numero2, soma;

    do {
        cout << "Entre com dois numeros inteiros positivos: ";
        cin >> numero1 >> numero2;
    } while (numero1 < 0 || numero2 < 0);

    if (numero1 == numero2) {
        cout << numero1 << "==" << numero2 << endl;
        soma = 2*numero1;
    } else {
        cout << numero1 << "!=" << numero2 << endl;
        soma = numero1 + numero2;
    }

    cout << "Soma eh: " << soma << endl;


    return 0;
}
```

Terceiro Exemplo em C++

```
#include <iostream>

using namespace std;

int main()
{
     int numero1, numero2, soma;

     do {
          cout << "Entre com dois numeros inteiros positivos: ";
          cin >> numero1 >> numero2;
     } while (numero1 < 0 || numero2 < 0);

     if (numero1 == numero2) {
          cout << numero1 << "==" << numero2 << endl;
          soma = 2*numero1;
     } else {
          cout << numero1 << "!=" << numero2 << endl;
          soma = numero1 + numero2;
     }

     cout << "Soma eh: " << soma << endl;


     return 0;
}
```

Uso do namespace std dispensa o prefixo std

Terceiro Exemplo em C++

```
#include <iostream>

using namespace std;

int main()
{
     int numero1, numero2, soma;

     do {
          cout << "Entre com dois numeros inteiros positivos: ";
          cin >> numero1 >> numero2;
     } while (numero1 < 0 || numero2 < 0);

     if (numero1 == numero2) {
          cout << numero1 << "==" << numero2 << endl;
          soma = 2*numero1;
     } else {
          cout << numero1 << "!=" << numero2 << endl;
          soma = numero1 + numero2;
     }

     cout << "Soma eh: " << soma << endl;


     return 0;
}
```

Entrada de dois inteiros em apenas uma sentença

Terceiro Exemplo em C++

```
#include <iostream>

using namespace std;

int main()
{
     int numero1, numero2, soma;

     do {
          cout << "Entre com dois numeros inteiros positivos: ";
          cin >> numero1 >> numero2;
     } while (numero1 < 0 || numero2 < 0);

     if (numero1 == numero2) {
          cout << numero1 << "==" << numero2 << endl;
          soma = 2*numero1;
     } else {
          cout << numero1 << "!=" << numero2 << endl;
          soma = numero1 + numero2;
     }

     cout << "Soma eh: " << soma << endl;


     return 0;
}
```

Uso da estrutura `do-while`

Terceiro Exemplo em C++

```
#include <iostream>

using namespace std;

int main()
{
     int numero1, numero2, s

     do {
          cout << "Entre cor
          cin >> numero1 >>
     } while (numero1 < 0 || numero2 < 0);

     if (numero1 == numero2) {
          cout << numero1 << "==" << numero2 << endl;
          soma = 2*numero1;
     } else {
          cout << numero1 << "!=" << numero2 << endl;
          soma = numero1 + numero2;
     }

     cout << "Soma eh: " << soma << endl;


     return 0;
}
```

```
shell>$ g++ -Wall ex3.cpp –o ex3
shell>$ ./ex3
Entre com os dois numeros inteiros
positivos: 1
2
1 != 2
Soma eh: 3
shell>$
```

Exemplo 1: Fatorial

- Escreva um programa em C++ para calcular o número fatorial de um inteiro passado pelo usuário

?

Exemplo 1: Fatorial

```
/*
 * Exemplo 1
 * Autor: Miguel Campista
*/

#include <iostream>

using namespace std;

// Função para cálculo de número fatorial
int fatorial (int n) {
    if (n == 1)
        return 1;
    else
        return n*fatorial(n - 1);
}

int main()
{
    int n;

    cout << "Entre com um numero inteiro positivo: ";
    cin >> n;

    cout << "Fatorial: " << fatorial(n) << endl;

    return 0;
}
```

Exemplo 1: Fatorial com fatorial.h

Arquivo principal

```
/*
 * Exemplo 1
 * Autor: Miguel Campista
*/

#include <iostream>
#include "fatorial.h"

using namespace std;

int main()
{
    int n;

    cout << "Entre com um numero inteiro positivo: ";
    cin >> n;

    cout << "Fatorial: " << fatorial(n) << endl;

    return 0;
}
```

Arquivo: fatorial.h

```
int fatorial (int n);
```

Arquivo: fatorial.cpp

```
#include "fatorial.h"

// Função para cálculo de número fatorial
int fatorial (int n) {
    if (n == 1)
        return 1;
    else
        return n*fatorial(n - 1);
}
```

Estrutura em Classes e Funções

- Programas até aqui...
  - Todas as sentenças estavam localizadas na função `main` ou nas funções utilizadas nela

- Programas de agora em diante...
  - Em geral consistem
    - Na função `main` e
    - Em uma ou mais classes
      - Cada uma conterá **membros de dados** (variáveis) e **funções-membro** (funções ou métodos)

O que é uma Classe?

- Classe é um conceito estendido de estrutura de dados
  - Porém, além de apenas organizar dados, as classes também oferecem funções de manipulação
  - Em outras palavras...
    - Uma classe pode ser comparada a uma `struct` que engloba **atributos (variáveis) e métodos (funções)**
      - **ENCAPSULAMENTO**

O que é uma Classe?

- Classe é um conceito estendido de estrutura de dados
  - Porém, além de apenas organizar dados, as classes também oferecem funções de manipulação
  - Em outras palavras...
    - Uma classe pode ser comparada a uma `struct` que engloba **atributos (variáveis) e métodos (funções)**
      - **ENCAPSULAMENTO**

```
struct nome_struct {
        variáveis;
};
```

```
class nome_classe {
        variáveis;
        funções();
};
```

O que é um Objeto?

- Uma classe não pode ser manipulada diretamente pelo programador
  - Como uma estrutura que não é manipulada diretamente
- Características dos objetos são definidos pela sua classe
  - Em termos de variáveis, uma classe é um tipo e o objeto é a variável

```
struct nome_struct {
        variáveis;
} estrutura;
```

```
class nome_classe {
        variáveis;
        funções();
} objeto;
```

Um Exemplo Prático de Classes e Objetos

- Exemplo do carro
  - Métodos descrevem os mecanismos responsáveis pela execução das tarefas
    - Ex.: Aceleração do carro
  - Tarefas complexas são ocultadas do usuário
    - Ex.: Motorista pode usar o pedal do acelerador, mas não precisa saber como é o processo de aceleração
  - As classes devem ser definidas antes de serem usadas
    - Da mesma forma, os carros também devem ser construídos antes de serem dirigidos

Um Exemplo Prático de Classes e Objetos

- Exemplo do carro
  - Muitos objetos carro podem ser criados da mesma classe
    - Da mesma forma, muitos carros podem ser construídos com o mesmo desenho de engenharia
  - Chamadas a funções enviam mensagens a um objeto para executar determinadas tarefas
    - Da mesma forma, pisar no acelerador envia uma mensagem ao carro para que acelere
  - Objetos e carros possuem atributos
    - Ex.: Cor e quilômetros rodados

Linguagem de Programação C++ com Uso de Classes

- Mais sete exemplos simples
  - Exemplos usados para construir uma classe **`GradeBook`**
- Tópicos cobertos:
  - Métodos (**Funções ou Funções-membro**)
  - Atributos (**Variáveis ou Membros de dados**)
  - Clientes de uma classe
    - Objetos de uma classe podem ter suas funções chamadas em outras classes ou funções
  - Separando a interface da implementação
  - Validação de dados
    - Garante que os dados em um objeto estejam em um determinado formato ou intervalo

Definição de uma Classe

- A definição da classe indica ao compilador que métodos e atributos pertencem àquela classe
- A declaração de uma classe requer o uso da palavra-chave **`class`**
  - A palavra-chave **`class`** é seguida do nome da classe
- O corpo da classe é colocado entre chaves (`{ }`)
  - Especifica variáveis e funções
  - Especificador de acesso **`public:`**
    - Indica que um método ou atributos são acessíveis a outros métodos e a métodos definidos em outras classes

Primeiro Exemplo utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 1
 * Autor: Miguel Campista
*/

#include <iostream>

using namespace std;

// Definição da classe GradeBook
class GradeBook {
      public:
            void displayMessage() {
                  cout << "Bem-vindo ao seu primeiro programa com classes!" << endl;
            }
};

int main()
{
    GradeBook MyGradeBook;
    MyGradeBook.displayMessage();

    return 0;
}
```

Primeiro Exemplo utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 1
 * Autor: Miguel Campista
*/

#include <iostream>

using namespace std;

// Definição da classe GradeBook
class GradeBook {
      public:
            void displayMessage() {
                  cout << "Bem-vindo ao seu primeiro programa com classes!" << endl;
            }
};

int main()
{
    GradeBook MyGradeBook;
    MyGradeBook.displayMessage();

    return 0;
}
```

Início da definição da classe `GradeBook`

Primeiro Exemplo utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 1
 * Autor: Miguel Campista
*/

#include <iostream>

using namespace std;

// Definição da classe GradeBook
class GradeBook {
      public:
            void displayMessage() {
                  cout << "Bem-vindo ao seu primeiro programa com classes!" << endl;
            }
};

int main()
{
    GradeBook MyGradeBook;
    MyGradeBook.displayMessage();

    return 0;
}
```

Início do corpo da classe

Final do corpo da classe

Primeiro Exemplo utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 1
 * Autor: Miguel Campista
*/

#include <iostream>

using namespace std;

// Definição da classe GradeBook
class GradeBook {
      public:
            void displayMessage() {
                  cout << "Bem-vindo ao seu primeiro programa com classes!" << endl;
            }
};

int main()
{
     GradeBook MyGradeBook;
     MyGradeBook.displayMessage();

     return 0;
}
```

Especificador de acesso **public**: disponibiliza membros ao público

Primeiro Exemplo utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 1
 * Autor: Miguel Campista
*/

#include <iostream>

using namespace std;

// Definição da classe GradeBook
class GradeBook {
      public:
            void displayMessage()
                  cout << "Bem-vindo ao seu primeiro programa com classes!" << endl;
            }
};

int main()
{
    GradeBook MyGradeBook;
    MyGradeBook.displayMessage();

    return 0;
}
```

A função **displayMessage** não retorna nada

Primeiro Exemplo utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 1
 * Autor: Miguel Campista
*/

#include <iostream>

using namespace std;

// Definição da classe GradeBook
class GradeBook {
      public:
            void displayMessage() {
                  cout << "Bem-vindo ao seu primeiro programa com classes!" << endl;
            }
};

int main()
{
    GradeBook MyGradeBook;
    MyGradeBook.displayMessage();

    return 0;
}
```

O operador ponto é usado para chamar funções de **GradeBook**

Primeiro Exemplo utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 1
 * Autor: Miguel Campista
*/

#include <iostream>

using namespace std;

// Definição da classe GradeBook
class GradeBook {
      public:
            void displayMessage() {
                  cout << "Bem-vindo ao seu primeiro programa com classes!" << endl;
            }
};

int main()
{
    GradeBook MyGradeBook;
    MyGradeBook.displayMessage();

    return 0;
}
```

```
shell>$ g++ -Wall gradebook.cpp –o ex1
shell>$ ./ex1
Bem-vindo ao seu primeiro programa com classes!
shell>$
```

Pergunta

- Como ficaria o código se quiséssemos introduzir a função-membro **`somaNota(nota1, nota2)`** na classe GradeBook?

?

Segundo Exemplo utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 2
 * Autor: Miguel Campista
*/

#include <iostream>
#include <iomanip>

using namespace std;

// Definição da classe GradeBook
class GradeBook {

    public:
           void displayMessage() {
                cout << "Bem-vindo ao seu primeiro programa com classes!"
                     << endl;
           }
           float somaNota(float nota1, float nota2) {
                 return nota1 + nota2;
           }
};

int main()
{
    float nota1 = 1.23, nota2 = 2.34;
    GradeBook MyGradeBook;
    MyGradeBook.displayMessage();

    cout << "Soma das notas foi: " << setprecision(2) << MyGradeBook.somaNota(nota1, nota2) << endl;

    return 0;
}
```

Segundo Exemplo utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 2
 * Autor: Miguel Campista
*/

#include <iostream>
#include <iomanip>

using namespace std;

// Definição da classe GradeBook
class GradeBook {

    public:
            void displayMessage() {
                 cout << "Bem-vindo ao seu primeiro programa com classes!"
                      << endl;
            }
            float somaNota(float nota1, float nota2) {
                  return nota1 + nota2;
            }
};

int main()
{
    float nota1 = 1.23, nota2 = 2.34;
    GradeBook MyGradeBook;
    MyGradeBook.displayMessage();

    cout << "Soma das notas foi: " << setprecision(2) << MyGradeBook.somaNota(nota1, nota2) << endl;

    return 0;
}
```

Uso de uma nova função. Passagem de argumentos para a função `somaNota`

Segundo Exemplo utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 2
 * Autor: Miguel Campista
*/

#include <iostream>
#include <iomanip>

using namespace std;

// Definição da classe GradeBook
class GradeBook {

    public:
        void displayMessage() {
            cout << "Bem-vindo ao seu primeiro programa com classes!"
                 << endl;
        }
        float somaNota(float nota1, float nota2) {
            return nota1 + nota
        }
};

int main()
{
    float nota1 = 1.23, nota2 = 2.34
    GradeBook MyGradeBook;
    MyGradeBook.displayMessage();

    cout << "Soma das notas foi: " << setprecision(2) << MyGradeBook.somaNota(nota1, nota2) << endl;

    return 0;
}
```

Biblioteca `iomanip` define funções para manipular parâmetros de formatação

Ajusta a precisão dos pontos flutuantes, sem fixar o número de casas decimais

Segundo Exemplo utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 2
 * Autor: Miguel Campista
*/

#include <iostream>
#include <iomanip>

using namespace std;

// Definição da classe GradeBook
class GradeBook {

    public:
            void displayMessage() {
                 cout << "Bem-vindo ao seu primeiro programa com classes!"
                      << endl;
            }
            float somaNota(float nota1, float nota2) {
                  return nota1 + nota2;
            }
};

int main()
{
    float nota1 = 1.23, nota2 = 2.34;
    GradeBook MyGradeBook;
    MyGradeBook.displayMessage();

    cout << "Soma das notas foi: " << setprecision(2) << MyGradeBook.somaNota(nota1, nota2) << endl;

    return 0;
}
```

```
shell>$ g++ -Wall gradebook2.cpp –o ex2
shell>$ ./ex2
Bem-vindo ao seu primeiro programa com classes!
Soma das notas foi: 3.6
shell>$
```

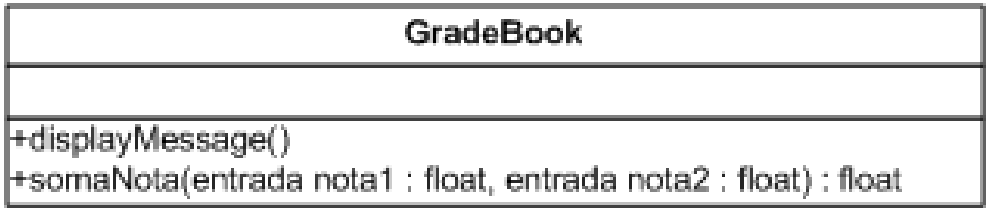
UML (*Unified Modeling Language*)

- Com o aumento da complexidade dos softwares
  - Surgiu a necessidade para que o desenvolvimento se torna-se mais estruturado

- UML surgiu para representar graficamente sistemas
  - Possivelmente sistemas orientados a objetos
    - Padronização permite que o mesmo tipo de figuras sejam compreendidos por desenvolvedores diferentes

UML (*Unified Modeling Language*)

- Diagrama de classe
  - Representada como um retângulo com três compartimentos:
    - No topo, o nome da classe centralizado horizontalmente e em negrito
    - No meio, os atributos da classe
    - Em baixo, as funções membro da classe
      - O sinal de positivo (+) significa que o método é público



Usando Classes

- Classe é um tipo definido por usuário ou programador
  - Pode ser utilizada para criar objetos
    - Variáveis do tipo da classe
  - C++ é uma linguagem extensível
- Operador ponto (.)
  - É usado para acessar atributos e métodos de um objeto
    - Ex.:
      - `myGradeBook.displayMessage()`
        - » **Chama o método** `displayMessage` **do objeto** `myGradeBook` **da classe** `GradeBook`

Usando Classes

- Parâmetro(s) de função
  - Informação necessária para que uma função execute sua tarefa
- Argumento(s) da função
  - Valores fornecidos por uma chamada de função a cada parâmetro da função
    - Os valores de argumento são copiados nos parâmetros de função

```
//Argumento
main () {
        int arg = 1;
        função(arg);
}
```

```
// Parâmetro
int função (int param)
{
        corpo;
}
```

Usando Classes

- Uma **`string`**
  - Representa uma string de caracteres.
  - Objeto da classe **`std::string`** da *C++ Standard Library*
    - É definida no arquivo de cabeçalho `<string>`
- Função de biblioteca **`getline`**
  - Recupera uma entrada até uma nova linha ser encontrada
    - Ex.: `getline( cin, nameOfCourse );`
      - Gera uma linha da entrada-padrão na string `object nameOfCourse`

Terceiro Exemplo utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 3
 * Autor: Miguel Campista
*/

#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

// Definição da classe GradeBook
class GradeBook {

    public:
            void displayMessage(string courseName) {
                 cout << "Bem-vindo ao seu primeiro programa com classes em "
                      << courseName << "!" << endl;
            }
            float somaNota(float nota1, float nota2) {
                  return nota1 + nota2;
            }
};
```

Terceiro Exemplo utilizando Classes em C++

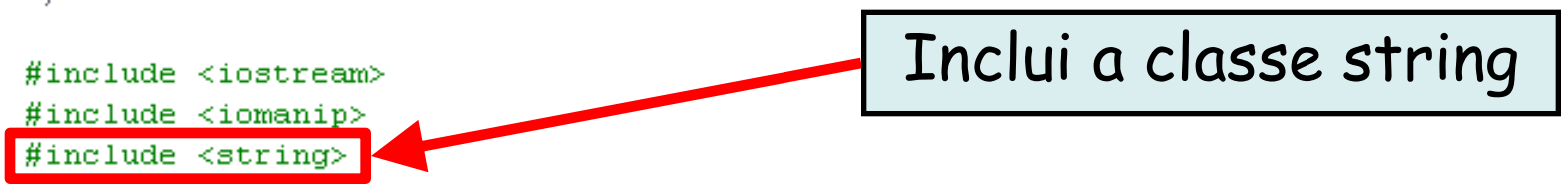
```
/*
 * Aula 4 -- Exemplo 3
 * Autor: Miguel Campista
*/

#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

// Definição da classe GradeBook
class GradeBook {

    public:
            void displayMessage(string courseName) {
                 cout << "Bem-vindo ao seu primeiro programa com classes em "
                      << courseName << "!" << endl;
            }
            float somaNota(float nota1, float nota2) {
                  return nota1 + nota2;
            }
};
```



Terceiro Exemplo utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 3
 * Autor: Miguel Campista
*/

#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

// Definição da classe GradeBook
class GradeBook {

    public:
            void displayMessage(string courseName) {
                 cout << "Bem-vindo ao seu primeiro programa com classes em "
                      << courseName << "!" << endl;
            }
            float somaNota(float nota1, float nota2) {
                  return nota1 + nota2;
            }
};
```

Parâmetro da função

Parâmetro usado como variável

Terceiro Exemplo utilizando Classes em C++

```
int main()
{
    float nota1 = 1.23, nota2 = 2.34;
    string courseName;
    GradeBook MyGradeBook;

    cout << "Entre com o nome do curso: " << endl;
    getline(cin, courseName); // Lê o nome do curso com espaços em branco

    // Chamo a função displayMessage com a passagem do parâmetro
    MyGradeBook.displayMessage(courseName);

    cout << "Soma das notas foi: " << setprecision(2)
         << MyGradeBook.somaNota(nota1, nota2) << endl;

    return 0;
}
```

Terceiro Exemplo utilizando Classes em C++

Uso da função `getline`. O primeiro parâmetro é de onde vem os caracteres e o segundo onde eles são armazenados. Recebe inclusive caracteres em branco

```
int main()
{
    float nota1 = 1.13,
    string courseName;
    GradeBook MyGradeBook;

    cout << "Entre com o nome do curso: " << endl;
    getline(cin, courseName); // Lê o nome do curso com espaços em branco

    // Chamo a função displayMessage com a passagem do parâmetro
    MyGradeBook.displayMessage(courseName);

    cout << "Soma das notas foi: " << setprecision(2)
         << MyGradeBook.somaNota(nota1, nota2) << endl;

    return 0;
}
```

Terceiro Exemplo utilizando Classes em C++

Argumento da função

```
int main()
{
    float nota1 = 1.23, nota2 = 2.34;
    string courseName;
    GradeBook MyGradeBook;

    cout << "Entre com o nome do curso: " << endl;
    getline(cin, courseName); // Lê o nome do curso com espaços em branco

    // Chamo a função displayMessage com a passagem do parâmetro
    MyGradeBook.displayMessage(courseName);

    cout << "Soma das notas foi: " << setprecision(2)
         << MyGradeBook.somaNota(nota1, nota2) << endl;

    return 0;
}
```

Terceiro Exemplo utilizando Classes em C++

```
shell>$ g++ -Wall gradebook.cpp –o ex3
shell>$ ./ex3
Entre com o nome do curso:
Programação
Bem-vindo ao seu primeiro programa com classes em Programação!
Soma das notas foi: 3.6
shell>$
```

```
    cout << "Entre com o nome do curso: " << endl;
    getline(cin, courseName); // Lê o nome do curso com espaços em branco

    // Chamo a função displayMessage com a passagem do parâmetro
    MyGradeBook.displayMessage(courseName);

    cout << "Soma das notas foi: " << setprecision(2)
         << MyGradeBook.somaNota(nota1, nota2) << endl;

    return 0;
}
```

Terceiro Exemplo utilizando Classes em C++

- Como ficaria se eu quisesse armazenar a string passada pelo usuário até a aparição de um caractere específico?
  - Caractere específico é chamado de delimitador

Terceiro Exemplo utilizando Classes em C++

- Como ficaria se eu quisesse armazenar a string passada pelo usuário até a aparição de um caractere específico?
  - Caractere específico é chamado de delimitador

RESPOSTA: Uso da função

```
getline(istream& is, string &str, char delim);
```

Definida em:

http://www.cplusplus.com/reference/string

Terceiro Exemplo utilizando Classes em C++


```
int main()
{
    float nota1 = 1.23, nota2 = 2.34;
    string courseName;
    GradeBook MyGradeBook;

    cout << "Entre com o nome do curso: " << endl;
    getline(cin, courseName, 'a'); // Lê o nome do curso com espaços em branco

    // Chamo a função displayMessage com a passagem do parâmetro
    MyGradeBook.displayMessage(courseName);

    cout << "Soma das notas foi: " << setprecision(2)
         << MyGradeBook.somaNota(nota1, nota2) << endl;

    return 0;
}
```

Terceiro Exemplo utilizando Classes em C++

```
shell>$ g++ -Wall gradebook.cpp –o ex3
shell>$ ./ex3
Entre com o nome do curso:
Programação
Bem-vindo ao seu primeiro programa com classes em Progr!
Soma das notas foi: 3.6
shell>$
```

```
int main
{
    floa
    stri
    Grad

    cout << "Entre com o nome do curso: " << endl;
    getline(cin, courseName, 'a'); // Lê o nome do curso com espaços em branco

    // Chamo a função displayMessage com a passagem do parâmetro
    MyGradeBook.displayMessage(courseName);

    cout << "Soma das notas foi: " << setprecision(2)
         << MyGradeBook.somaNota(nota1, nota2) << endl;

    return 0;
}
```

Uso de Funções `set` e `get`

- Variáveis locais
  - Variáveis declaradas no corpo de uma função
    - Não podem ser utilizadas fora do corpo dessa função
  - Quando uma função termina...
    - Os valores das respectivas variáveis locais são perdidos

- Atributos
  - **Existem por toda a vida do objeto**
  - São representados como membros de dados
    - Variáveis em uma definição de classe
  - Todo objeto de classe mantém sua própria cópia de atributos

Quarto Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 4
 * Autor: Miguel Campista
*/

#include <iostream>
#include <string>

using namespace std;

// Definição da classe GradeBook
class GradeBook {

    public:
        // Função que configura o nome do curso
        void setCourseName(string name) {
            courseName = name;
        }
        // Função que obtém o nome do curso
        string getCourseName() {
            return courseName;
        }
        void displayMessage() {
            cout << "Bem-vindo ao seu primeiro programa com classes em "
                 << getCourseName() << "!" << endl;
        }

    private:
        string courseName;
};
```

Quarto Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 4
 * Autor: Miguel Campista
*/

#include <iostream>
#include <string>

using namespace std;

// Definição da classe GradeBook
class GradeBook {

    public:
        // Função que configura o nome do curso
        void setCourseName(string name) {
            courseName = name;
        }
        // Função que obtém o nome do curso
        string getCourseName() {
            return courseName;
        }
        void displayMessage() {
            cout << "Bem-vindo ao seu primeiro programa com classes em "
                 << getCourseName() << "!" << endl;
        }

    private:
        string courseName;
};
```

As variáveis `private` são acessíveis apenas a funções da classe

Quarto Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 4
 * Autor: Miguel Campista
*/

#include <iostream>
#include <string>

using namespace std;

// Definição da classe GradeBook
class GradeBook {

    public:
        // Função que configura o nome do curso
        void setCourseName(string name) {
            courseName = name;
        }
        // Função que obtém o nome do curso
        string getCourseName() {
            return courseName;
        }
        void displayMessage() {
            cout << "Bem-vindo ao seu primeiro programa com classes em "
                 << getCourseName() << "!" << endl;
        }

    private:
        string courseName;
};
```

A função `set` modifica os dados `private`

Quarto Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 4
 * Autor: Miguel Campista
*/

#include <iostream>
#include <string>

using namespace std;

// Definição da classe GradeBook
class GradeBook {

    public:
        // Função que configura o nome do curso
        void setCourseName(string name) {
            courseName = name;
        }
        // Função que obtém o nome do curso
        string getCourseName() {
            return courseName;
        }
        void displayMessage() {
            cout << "Bem-vindo ao seu primeiro programa com classes em "
                 << getCourseName() << "!" << endl;
        }

    private:
        string courseName;
};
```

A função `get` obtém os dados `private`

Quarto Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 4
 * Autor: Miguel Campista
*/

#include <iostream>
#include <string>

using namespace std;

// Definição da classe GradeBook
class GradeBook {

    public:
        // Função que configura o nome do curso
        void setCourseName(string name) {
            courseName = name;
        }
        // Função que obtém o nome do curso
        string getCourseName() {
            return courseName;
        }
        void displayMessage() {
            cout << "Bem-vindo ao seu primeiro programa com classes em "
                 << getCourseName() << "!" << endl;
        }

    private:
        string courseName;
};
```

As funções `get` e `set` são usadas mesmo dentro da definição da classe

Quarto Exemplo Utilizando Classes em C++

```
int main()
{
    string name;
    GradeBook MyGradeBook;

    // Exibe o valor inicial de courseName
    cout << "Nome inicial do curso eh: " << MyGradeBook.getCourseName() << endl;

    // Pede e configura o nome do curso
    cout << "Entre com o nome do curso: " << endl;
    getline(cin, name); // Lê o nome do curso com espaços em branco
    MyGradeBook.setCourseName(name);

    // Chamo a função displayMessage com a passagem do parâmetro
    MyGradeBook.displayMessage();

    return 0;
}
```

Quarto Exemplo Utilizando Classes em C++

```
int main()
{
    string name;
    GradeBook MyGradeBook;

    // Exibe o valor inicial de courseName
    cout << "Nome inicial do curso eh: " << MyGradeBook.getCourseName() << endl;

    // Pede e configura o nome do curso
    cout << "Entre com o nome do curso: " << endl;
    getline(cin, name); // Lê o nome do curso com espaços em branco
    MyGradeBook.setCourseName(name);

    // Chamo a função displayMessage com a passagem do parâm
    MyGradeBook.displayMessage();

    return 0;
}
```

Acessando dados `private` externamente à definição de classe

Modificando dados `private` externamente à definição de classe

Quarto Exemplo Utilizando Classes em C++

```
shell>$ g++ -Wall gradebook.cpp –o ex4
shell>$ ./ex4
Nome inicial do curso eh:

Entre com o nome do curso:
Programação
Bem-vindo ao seu primeiro programa com classes em Programação!
shell>$
```

```
int
{
    // Pede e configura o nome do curso
    cout << "Entre com o nome do curso: " << endl;
    getline(cin, name); // Lê o nome do curso com espaços em branco
    MyGradeBook.setCourseName(name);

    // Chamo a função displayMessage com a passagem do parâmetro
    MyGradeBook.displayMessage();

    return 0;
}
```

Uso de Funções `set` e `get`

- Especificador de acesso **`private`**
  - Torna uma variável ou uma função acessível apenas a funções da mesma classe
  - Acesso padrão de membros de classe é **`private`**
  - Oculta dados para as classes externas

- Retorno de uma função
  - Uma função que especifica um tipo de retorno diferente de **`void`**...
    - Retorna um valor à função que a chamou

Uso de Funções `set` e `get`

- Como regra geral...
  - Atributos são **`private`** e as funções são **`public`**
- Funções que não estejam definidas em uma classe
  - Não podem acessar um membro **`private`** dessa classe
- Especificadores de acesso **`public`** e **`private`** de uma classe podem ser repetidos e combinados
  - Porém, apresentar todos os membros **`public`** e depois os **`private`** chama a atenção para a interface **`public`**
  - Se os membros **`private`** forem apresentados primeiro, o especificador **`private`** deve ser utilizado

Uso de Funções `set` e `get`

- Não é necessário fornecer sempre funções **`get`** e **`set`** para cada item de dados **`private`**
  - Essas funções devem ser fornecidas somente quando apropriado
    - Quando um serviço for útil ao código-cliente, em geral deve ser fornecido na interface **`public`** da classe

Engenharia de Software com Funções `set` e `get`

- Funções **`set`** e **`get`** são funções **`public`** que...
  - Permitem clientes de uma classe atribuir ou obter valores de membros de dados **`private`**
  - Permitem que o criador da classe controle a forma como os clientes modificam e acessam dados **`private`**
  - Devem também ser utilizadas por outras funções da mesma classe

- Funções **`set`** são também chamadas de **modificadoras** e as funções **`get`** de **funções de acesso**

Construtores

- Funções utilizadas para inicializar dados de um objeto no momento em que esse objeto é criado
  - Realizam chamada implícita quando o objeto é criado
  - Devem ser definidos com o mesmo nome da classe
  - Não podem retornar valores
    - Nem mesmo `void`
- O construtor-padrão não tem nenhum parâmetro
  - O compilador fornecerá um quando uma determinada classe não incluir explicitamente um construtor
    - O construtor-padrão do compilador chama apenas construtores de objetos de classe

Quinto Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 5
 * Autor: Miguel Campista
*/

#include <iostream>
#include <string>

using namespace std;

// Definição da classe GradeBook
class GradeBook {

    public:
        // Construtor inicializa courseName com a string-argumento
        GradeBook(string name) {
            setCourseName(name); // Chama a função set para inicialização
        }
        // Função que configura o nome do curso
        void setCourseName(string name) {
            courseName = name;
        }
        // Função que obtém o nome do curso
        string getCourseName() {
            return courseName;
        }
        void displayMessage() {
            cout << "Bem-vindo ao seu primeiro programa com classes em "
                 << getCourseName() << "!" << endl;
        }

    private:
        string courseName;
};
```

Quinto Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 5
 * Autor: Miguel Campista
*/

#include <iostream>
#include <string>

using namespace std;

// Definição da classe GradeBook
class GradeBook {

    public:
        // Construtor inicializa courseName com a string-argum
        GradeBook(string name) {
            setCourseName(name); // Chama a função set para i
        }
        // Função que configura o nome do curso
        void setCourseName(string name) {
            courseName = name;
        }
        // Função que obtém o nome do curso
        string getCourseName() {
            return courseName;
        }
        void displayMessage() {
            cout << "Bem-vindo ao seu primeiro programa com classes em "
                 << getCourseName() << "!" << endl;
        }

    private:
        string courseName;
};
```

O construtor tem o mesmo nome da classe e não retorna nenhum valor. Além disso, inicializa variáveis do objeto

Quinto Exemplo Utilizando Classes em C++

```
int main()
{
    // Cria dois objetos GradeBook
    GradeBook gradeBook1("Programacao");
    GradeBook gradeBook2("Comp1");

    // Exibe o valor inicial de courseName
    cout << "Nome do curso 1 eh: " << gradeBook1.getCourseName() << endl
         << "Nome do curso 2 eh: " << gradeBook2.getCourseName() << endl;

    return 0;
}
```

Quinto Exemplo Utilizando Classes em C++

O construtor é implícito quando se cria objetos

```
int main()
{
    // Cria dois objetos GradeBook
    GradeBook gradeBook1("Programacao");
    GradeBook gradeBook2("Comp1");

    // Exibe o valor inicial de courseName
    cout << "Nome do curso 1 eh: " << gradeBook1.getCourseName() << endl
         << "Nome do curso 2 eh: " << gradeBook2.getCourseName() << endl;

    return 0;
}
```

Quinto Exemplo Utilizando Classes em C++

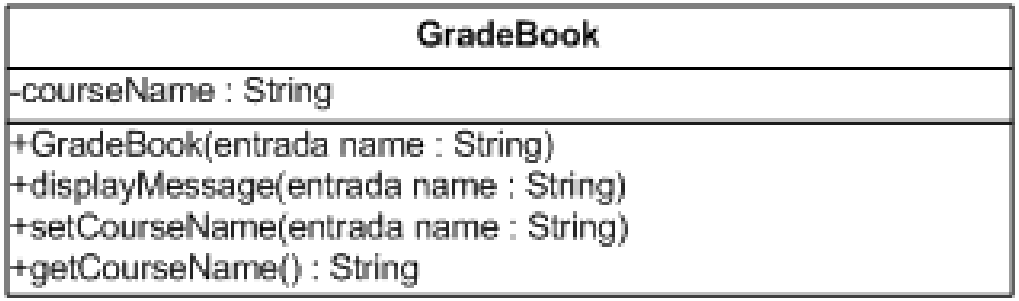
```
int main()
{
    // Cria dois objetos GradeBook
    GradeBook gradeBook1("Programacao");
    GradeBook gradeBook2("Comp1");

    // Exibe o valor inicial de courseName
    cout << "Nome do curso 1 eh: " << gradeBook1.getCourseName() << endl
         << "Nome do curso 2 eh: " << gradeBook2.getCourseName() << endl;

    return 0;
}
```

```
shell>$ g++ -Wall gradebook.cpp –o ex5
shell>$ ./ex5
Nome do curso 1 eh: Programacao
Nome do curso 2 eh: Compl
shell>$
```

Quinto Exemplo Utilizando Classes em C++



Inicialização das Variáveis de uma Classe

- A menos que nenhuma inicialização de atributos da classe seja necessária...
  - Construtores devem ser usados!
    - Asseguram que os atributos da classe sejam inicializados com valores significativos na instanciação de cada objeto

Inicialização das Variáveis de uma Classe

- As variáveis de uma classe podem ser inicializadas em um construtor da classe ou seus valores podem ser configurados depois que o objeto for criado
  - Entretanto, é importante assegurar que o objeto seja completamente inicializado antes do código-cliente invocar as funções do objeto
    - Não é garantido que o código-cliente inicializa os objetos adequadamente

Aumento do Reuso e Modularidade do Código

- Arquivos `*.cpp`
  - Arquivo de código-fonte

- Arquivos de cabeçalho: `*.h`
  - Arquivos separados nos quais são colocadas as definições de classe
    - Permitem que o compilador reconheça as classes quando usadas em outros lugares

Sexto Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 6
 * Arquivo principal
 * Autor: Miguel Campista
*/

#include <iostream>
#include <string>
#include "GradeBook.h" // Inclui a definição da classe

using namespace std;

int main()
{
    // Cria dois objetos GradeBook
    GradeBook gradeBook1("Programacao");
    GradeBook gradeBook2("Comp1");

    // Exibe o valor inicial de courseName
    cout << "Nome do curso 1 eh: " << gradeBook1.getCourseName() << endl
         << "Nome do curso 2 eh: " << gradeBook2.getCourseName() << endl;

    return 0;
}
```

Sexto Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 6
 * Arquivo principal
 * Autor: Miguel Campista
*/

#include <iostream>
#include <string>
#include "GradeBook.h" // Inclui a definição da c

using namespace std;

int main()
{
    // Cria dois objetos GradeBook
    GradeBook gradeBook1("Programacao");
    GradeBook gradeBook2("Comp1");

    // Exibe o valor inicial de courseName
    cout << "Nome do curso 1 eh: " << gradeBook1.getCourseName() << endl
         << "Nome do curso 2 eh: " << gradeBook2.getCourseName() << endl;

    return 0;
}
```

Incluir o arquivo de cabeçalho faz com que a definição de classe seja copiada no arquivo

```
/*
 * Aula 4 -- Exemplo 6
 * Arquivo: GradeBook.h
 * Autor: Miguel Campista
*/

#include <iostream>
#include <string>

using namespace std;

// Definição da classe GradeBook
class GradeBook {

    public:
        // Construtor inicializa courseName com a string-argumento
        GradeBook(string name) {
            setCourseName(name); // Chama a função set para inicialização
        }
        // Função que configura o nome do curso
        void setCourseName(string name) {
            courseName = name;
        }
        // Função que obtém o nome do curso
        string getCourseName() {
            return courseName;
        }
        void displayMessage() {
            cout << "Bem-vindo ao seu primeiro programa com classes em "
                 << getCourseName() << "!" << endl;
        }

    private:
        string courseName;
};
```

```
/*
 * Aula 4 -- Exemplo 6
 * Arquivo: GradeBook.h
```

```
shell>$ g++ -Wall gradebook.cpp –o ex6
shell>$ ./ex6
Nome do curso 1 eh: Programacao
Nome do curso 2 eh: Compl
shell>$
```

```
// Definição da classe GradeBook
class GradeBook {

    public:
        // Construtor inicializa courseName com a string-argumento
        GradeBook(string name) {
            setCourseName(name); // Chama a função set para inicialização
        }
        // Função que configura o nome do curso
        void setCourseName(string name) {
            courseName = name;
        }
        // Função que obtém o nome do curso
        string getCourseName() {
            return courseName;
        }
        void displayMessage() {
            cout << "Bem-vindo ao seu primeiro programa com classes em "
                 << getCourseName() << "!" << endl;
        }

    private:
        string courseName;
};
```

Criação de Objetos

- O compilador deve conhecer o tamanho do objeto
  - Os objetos C++ em geral contêm apenas atributos
  - O compilador cria uma cópia das funções da classe
    - Essa cópia é compartilhada por todos os objetos da classe

Interfaces

- Descrevem os serviços que os clientes de uma classe podem usar e como podem solicitar esses serviços
  - Não revela como a classe executa esses serviços
  - Define classe apenas com o nome das funções, tipos de retorno e tipos de parâmetro
    - Protótipos das funções
- A interface de uma classe consiste nas funções **`public`** da classe (serviços)

Separação das Interfaces das Implementações

- As funções devem ser definidas em um arquivo separado do arquivo de definição de classe
  - Arquivo de código-fonte para uma classe
    - Usa um operador de resolução de escopo binário (::) para unir cada função à definição da classe
  - Os detalhes da implementação são ocultados
    - Não é preciso conhecer a implementação
  - Em um arquivo de cabeçalho para uma classe
    - Os protótipos descrevem a interface **`public`** da classe
- O código-cliente não deve ser quebrado
  - A implementação pode mudar desde que não afete a interface

Sétimo Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 7
 * Arquivo: GradeBookEx7.h
 * Autor: Miguel Campista
*/

#include <string>

using namespace std;

// Definição da classe GradeBook
class GradeBook {

    public:
            // Construtor inicializa courseName com a string-argumento
            GradeBook(string);
            // Função que configura o nome do curso
            void setCourseName(string);
            // Função que obtém o nome do curso
            string getCourseName();
            void displayMessage();

    private:
            string courseName;
};
```

Sétimo Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 7
 * Arquivo: GradeBookEx7.h
 * Autor: Miguel Campista
*/

#include <string>

using namespace std;

// Definição da classe GradeBook
class GradeBook {

    public:
        // Construtor inicializa courseName com a string-argumento
        GradeBook(string);
        // Função que configura o nome do curso
        void setCourseName(string);
        // Função que obtém o nome do curso
        string getCourseName();
        void displayMessage();

    private:
        string courseName;
};
```

A interface contém protótipos das funções

Sétimo Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 7
 * Arquivo: GradeBookEx7.cpp
 * Autor: Miguel Campista
*/

#include <iostream>
#include <string>
#include "GradeBookEx7.h"

// Construtor inicializa courseName com a string-argumento
GradeBook::GradeBook(string name) {
      setCourseName(name); // Chama a função set para inicialização
}
// Função que configura o nome do curso
void GradeBook::setCourseName(string name) {
      courseName = name;
}
// Função que obtém o nome do curso
string GradeBook::getCourseName() {
      return courseName;
}
void GradeBook::displayMessage() {
      cout << "Bem-vindo ao seu primeiro programa com classes em "
           << getCourseName() << "!" << endl;
}
```

Sétimo Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 7
 * Arquivo: GradeBookEx7.cpp
 * Autor: Miguel Campista
*/

#include <iostream>
#include <string>
#include "GradeBookEx7.h"

// Construtor inicializa courseName com a string-argumento
GradeBook::GradeBook(string name) {
      setCourseName(name); // Chama a função set para inicialização
}
// Função que configura o nome do curso
void GradeBook::setCourseName(string name) {
      courseName = name;
}
// Função que obtém o nome do curso
string GradeBook::getCourseName() {
      return courseName;
}
void GradeBook::displayMessage() {
      cout << "Bem-vindo ao seu primeiro programa com classes em "
           << getCourseName() << "!" << endl;
}
```

A implementação de GradeBook é colocada em um arquivo de código-fonte separado

Sétimo Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 7
 * Arquivo: GradeBookEx7.cpp
 * Autor: Miguel Campista
*/

#include <iostream>
#include <string>
#include "GradeBookEx7.h"

// Construtor inicializa courseName com a string-argumento
GradeBook::GradeBook(string name) {
      setCourseName(name); // Chama a função set para inicialização
}
// Função que configura o nome do curso
void GradeBook::setCourseName(string name) {
      courseName = name;
}
// Função que obtém o nome do curso
string GradeBook::getCourseName() {
      return courseName;
}
void GradeBook::displayMessage() {
      cout << "Bem-vindo ao seu primeiro programa com classes em "
           << getCourseName() << "!" << endl;
}
```

Incluir o arquivo de cabeçalho

Sétimo Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 7
 * Arquivo: GradeBookEx7.cpp
 * Autor: Miguel Campista
*/

#include <iostream>
#include <string>
#include "GradeBookEx7.h"

// Construtor inicializa courseName com a string-argumento
GradeBook::GradeBook(string name) {
      setCourseName(name); // Chama a função set para inicialização
}
// Função que configura o nome do curso
void GradeBook::setCourseName(string name) {
      courseName = name;
}
// Função que obtém o nome do curso
string GradeBook::getCourseName() {
      return courseName;
}
void GradeBook::displayMessage() {
      cout << "Bem-vindo ao seu primeiro programa com classes em "
           << getCourseName() << "!" << endl;
}
```

O operador de resolução de escopo binário une uma função à sua classe

Sétimo Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 7
 * Arquivo principal
 * Autor: Miguel Campista
*/

#include <iostream>
#include <string>
#include "GradeBookEx7.h" // Inclui a definição da classe

using namespace std;

int main()
{
    // Cria dois objetos GradeBook
    GradeBook gradeBook1("Programacao");
    GradeBook gradeBook2("Comp1");

    // Exibe o valor inicial de courseName
    cout << "Nome do curso 1 eh: " << gradeBook1.getCourseName() << endl
         << "Nome do curso 2 eh: " << gradeBook2.getCourseName() << endl;

    return 0;
}
```

Sétimo Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 7
 * Arquivo principal
 * Autor: Miguel Campista
*/

#include <iostream>
#include <string>
#include "GradeBookEx7.h" // Inclui a definição da classe

using namespace std;

int main()
{
    // Cria dois objetos GradeBook
    GradeBook gradeBook1("Programacao");
    GradeBook gradeBook2("Comp1");

    // Exibe o valor inicial de courseName
    cout << "Nome do curso 1 eh: " << gradeBook1.getCourseName() << endl
         << "Nome do curso 2 eh: " << gradeBook2.getCourseName() << endl;

    return 0;
}
```

Arquivo de interfaces incluído

Sétimo Exemplo Utilizando Classes em C++

```
/*
```

```
shell>$ g++ -Wall -c gradebook.cpp –o gradebook.o
shell>$ g++ -Wall -c principal.cpp –o principal.o
shell>$ g++ -o ex7 gradebook.o principal.o
shell>$ ./ex7
Nome do curso 1 eh: Programacao
Nome do curso 2 eh: Compl
shell>$
```

```
int main()
{
    // Cria dois objetos GradeBook
    GradeBook gradeBook1("Programacao");
    GradeBook gradeBook2("Comp1");

    // Exibe o valor inicial de courseName
    cout << "Nome do curso 1 eh: " << gradeBook1.getCourseName() << endl
         << "Nome do curso 2 eh: " << gradeBook2.getCourseName() << endl;

    return 0;
}
```

Processo de Compilação e Vinculação

- Compilação do código fonte cria o código objeto da classe
  - Código fonte deve `#incluir` o arquivo de cabeçalho
  - Implementação das classes deve apenas fornecer o arquivo de cabeçalho e o código objeto ao cliente
- O cliente deve `#incluir` o cabeçalho em seu código
  - Assim, o compilador assegura que a função `main` cria e manipula corretamente os objetos da classe
- Para criar um aplicativo executável...
  - Código objeto do código cliente deve ser vinculado ao:
    - Código objeto da classe e das bibliotecas usadas

Testes de Validade

- As funções **`set`** podem validar dados
  - Esse processo é conhecido por teste de validade
  - Isso mantém o objeto em um estado consistente
    - O membro de dados contém um valor válido
  - Podem retornar valores indicativos de que houve a tentativa de atribuir dados inválidos

- Funções da biblioteca **`string`**
  - **`length`** retorna o número de caracteres na **`string`**
  - **`substr`** retorna uma substring específica dentro da **`string`**

Testes de Validade

- Programador deve fornecer testes de validade apropriado e informar os erros
  - Benefícios da integridade dos dados não são automáticos

Oitavo Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 8
 * Arquivo: GradeBookEx8.h
 * Autor: Miguel Campista
*/

#include <string>

using namespace std;

// Definição da classe GradeBook
class GradeBook {

    public:
            // Construtor inicializa courseName com a string-argumento
            GradeBook(string);
            // Função que configura o nome do curso
            void setCourseName(string);
            // Função que obtém o nome do curso
            string getCourseName();
            void displayMessage();

    private:
            string courseName;
};
```

Oitavo Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 8
 * Arquivo: GradeBookEx8.cpp
 * Autor: Miguel Campista
*/

#include <iostream>
#include <string>
#include "GradeBookEx8.h"

// Construtor inicializa courseName com a string-argumento
GradeBook::GradeBook(string name) {
      setCourseName(name); // Chama a função set para inicialização
}
// Função que configura o nome do curso
void GradeBook::setCourseName(string name) {
      if(name.length() <= 25) {
            courseName = name;
      } else {
            courseName = name.substr(0, 25);
            cout << "Warning: Nome \"" << name <<
            "\" excede o limite maximo de 25 caracteres..." << endl <<
            "Nome limitado aos primeiros 25 caracteres: " << courseName <<
            endl;
      }
}
// Função que obtém o nome do curso
string GradeBook::getCourseName() {
      return courseName;
}
void GradeBook::displayMessage() {
      cout << "Bem-vindo ao seu primeiro programa com classes em "
           << getCourseName() << "!" << endl;
}
```

Oitavo Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 8
 * Arquivo: GradeBookEx8.cpp
 * Autor: Miguel Campista
*/

#include <iostream>
#include <string>
#include "GradeBookEx8.h"

// Construtor inicializa courseName com a string-argumento
GradeBook::GradeBook(string name) {
      setCourseName(name); // Chama a função set para inicialização
}
// Função que configura o nome do curso
void GradeBook::setCourseName(string name) {
      if(name.length() <= 25) {
            courseName = name;
      } else {
            courseName = name.substr(0, 25);
            cout << "Warning: Nome \"" << name <<
            "\" excede o limite maximo de 25 caracteres..." << endl <<
            "Nome limitado aos primeiros 25 caracteres: " << courseName <<
            endl;
      }
}
// Função que obtém o nome do curso
string GradeBook::getCourseName() {
      return courseName;
}
void GradeBook::displayMessage() {
      cout << "Bem-vindo ao seu primeiro programa com classes em "
           << getCourseName() << "!" << endl;
}
```

O construtor chama a função `set` para executar o teste de validade

Oitavo Exemplo Utilizando Classes em C++

As funções `set` executam o teste de validade para manter `courseName` em um estado consistente

```
/*
 * Aula 4 -- Exemplo 8
 * Arquivo: GradeBookEx8.cpp
 * Autor: Miguel Campista
*/

#include <iostream>
#include <string>
#include "GradeBookEx8.h"

// Construtor inicializa courseName com a string-argumento
GradeBook::GradeBook(string name) {
      setCourseName(name); // Chama a função set para inici
}
// Função que configura o nome do curso
void GradeBook::setCourseName(string name) {
      if(name.length() <= 25) {
            courseName = name;
      } else {
            courseName = name.substr(0, 25);
            cout << "Warning: Nome \"" << name <<
            "\" excede o limite maximo de 25 caracteres..." << endl <<
            "Nome limitado aos primeiros 25 caracteres: " << courseName <<
            endl;
      }
}
// Função que obtém o nome do curso
string GradeBook::getCourseName() {
      return courseName;
}
void GradeBook::displayMessage() {
      cout << "Bem-vindo ao seu primeiro programa com classes em "
           << getCourseName() << "!" << endl;
}
```

Oitavo Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 8
 * Arquivo principal
 * Autor: Miguel Campista
*/

#include <iostream>
#include <string>
#include "GradeBookEx8.h" // Inclui a definição da classe

using namespace std;

int main()
{
    // Cria dois objetos GradeBook
    GradeBook gradeBook1("Programacao de Computadores e Sistemas Distribuidos");
    GradeBook gradeBook2("Comp1");

    // Exibe o valor inicial de courseName
    cout << "Nome do curso 1 eh: " << gradeBook1.getCourseName() << endl
         << "Nome do curso 2 eh: " << gradeBook2.getCourseName() << endl;

    return 0;
}
```

Oitavo Exemplo Utilizando Classes em C++

String com mais de 25 caracteres

```
/*
 * Aula 4 -- Exemplo 8
 * Arquivo principal
 * Autor: Miguel Campista
*/

#include <iostream>
#include <string>
#include "GradeBookEx8.h" // Inclui a definição da classe

using namespace std;

int main()
{
    // Cria dois objetos GradeBook
    GradeBook gradeBook1("Programacao de Computadores e Sistemas Distribuidos");
    GradeBook gradeBook2("Comp1");

    // Exibe o valor inicial de courseName
    cout << "Nome do curso 1 eh: " << gradeBook1.getCourseName() << endl
         << "Nome do curso 2 eh: " << gradeBook2.getCourseName() << endl;

    return 0;
}
```

Oitavo Exemplo Utilizando Classes em C++

```
/*
 * Aula 4 -- Exemplo 8
 * Arquivo principal
```

```
shell>$ g++ -Wall -c gradebook.cpp –o gradebook.o
shell>$ g++ -Wall -c principal.cpp –o principal.o
shell>$ g++ -o ex8 gradebook.o principal.o
shell>$ ./ex8
Warning: Nome “Programacao de Computadores e Sistemas Distribuidos”
excede o limite maximo de 25 caracteres...
Nome limitado aos primeiros 25 caracteres: Programacao de Computador
Nome do curso 1 eh: Programacao de Computador
Nome do curso 2 eh: Compl
shell>$
```

```
    cout << "Nome do curso 1 eh: " << gradeBook1.getCourseName() << endl
         << "Nome do curso 2 eh: " << gradeBook2.getCourseName() << endl;

    return 0;
}
```

Exemplo 1: Calculadora

- Escreva um programa em C++ para calcular dois números inteiros passados pelo usuário

Exemplo 1: Calculadora

```
/*
 * Arquivo calculadora.h: Classe Calculadora
 * Autor: Miguel Campista
*/

class Calculadora {
      public:
             int calcSoma(int, int);
             int calcDif(int, int);
             int calcProd(int, int);
             float calcDiv(int, int);
};
```

```
/*
 * Arquivo calculadora.cpp: Classe Calculadora
 * Autor: Miguel Campista
*/

#include <iostream>
#include "calculadora.h"

using namespace std;

int Calculadora::calcSoma(int a, int b) { return a + b; }

int Calculadora::calcDif(int a, int b) { return a - b; }

int Calculadora::calcProd(int a, int b) { return a * b; }

float Calculadora::calcDiv(int a, int b) { return (float) a / b; }
```

Exemplo 1: Calculadora

```
int main()
{
    int a, b;
    char op;
    Calculadora calc;

    while(1) {
        cout <<  "Entre com o operador da opcao desejada:";
        cin >> op;
        cout << "Entre com os operandos:";
        cin >> a >> b;

        switch(op) {
               case '+':
                    cout << "A soma eh: " << calc.calcSoma(a, b) << endl;
                    break;
               case '-':
                    cout << "A diferenca eh: " << calc.calcDif(a, b) << endl;
                    break;
               case '*':
                    cout << "O produto eh: " << calc.calcProd(a, b) << endl;
                    break;
               case '/':
                    cout << "A divisao eh: " << calc.calcDiv(a, b) << endl;
                    break;
               default:
                    cout << "Operacao desconhecida" << endl;
        }
    }
    return 0;
}
```

Exemplo 2: Cadastro

- Escreva uma agenda em C++ para armazenar em memória três cadastros contendo nome, telefone e endereço. Cada um dos cadastros deve ser um objeto da classe Cadastro. A classe Cadastro ainda deve oferecer uma função para exibição dos dados de cada cadastro.

?

Exemplo 2: Cadastro

```
/*
 * Aula 5 -- Exemplo 7
 * Arquivo: cadastro.h
 * Autor: Miguel Campista
*/
#include <iostream>
#include <string>
#include <iomanip>

using namespace std;

class Cadastro {
      public:
            // Contrutor da classe Cadastro recebe 3 parâmetros
            Cadastro(string, string, string);
            // Função de exibição de todos os cadastros
            void displayData();

      private:
            string nomeCad, telCad, endCad;
            // Função para exibir o nome do cadastro
            string getNome();
            // Função para exibir o telefone do cadastro
            string getTel();
            // Função para exibir o endereço do cadastro
            string getEnd();
};
```

```
/*
 * Aula 5 -- Exemplo 7
 * Arquivo: cadastro.cpp
 * Autor: Miguel Campista
*/
#include "cadastro.h"

// Contrutor da classe Cadastro recebe 3 parâmetros
Cadastro::Cadastro(string nome, string tel, string endereco) {
       nomeCad = nome;
       telCad = tel;
       endCad = endereco;
}
// Função de exibição de todos os cadastros
void Cadastro::displayData() {
        cout << left << setw(25) << getNome()
             << setw(10) << getTel()
             << setw(25) << getEnd() << endl;
}
// Função para exibir o nome do cadastro
string Cadastro::getNome() {
       return nomeCad;
}
// Função para exibir o telefone do cadastro
string Cadastro::getTel() {
       return telCad;
}
// Função para exibir o endereço do cadastro
string Cadastro::getEnd() {
       return endCad;
}
```

Exemplo 2: Cadastro

```
/*
 * Aula 5 -- Exemplo 7
 * Arquivo Principal
 * Autor: Miguel Campista
*/
#include <iostream>
#include <string>
#include "cadastro.h"

using namespace std;

int main() {

    int ncad = 3;
    string nome[ncad], tel[ncad], endereco[ncad];

    for (int i = 0; i < ncad; i++) {
          cout << "Entre com o novo cadastro:" << endl;
          cout << "Nome: ";
          getline(cin, nome[i]);
          cout << "Telefone: ";
          getline(cin, tel[i]);
          cout << "Endereco: ";
          getline(cin, endereco[i]);
    }

    Cadastro cad0(nome[0], tel[0], endereco[0]);
    Cadastro cad1(nome[1], tel[1], endereco[1]);
    Cadastro cad2(nome[2], tel[2], endereco[2]);
```

Exemplo 2: Cadastro

```
    cout << left << setw(25) << "Nome:"
         << setw(10) << "Tel:"
         << setw(25) << "End:" << endl;
    cad0.displayData();
    cad1.displayData();
    cad2.displayData();

    return 0;
}
```

Leitura Recomendada

- Capítulos 2, 3, 4 e 5 do livro
  - Deitel, "*C++ How to Program*", 5th edition, Editora Prentice Hall, 2005