Java - Introdução

BCC 221 - Programação Orientada a Objectos(POO)

Guillermo Cámara-Chávez

Departamento de Computação - UFOP

Introdução

- A *Sun Microsystems* financiou uma pesquisa corporativa interna em 1991, com codinome *Green*
  - Microprocessadores;
  - O foco eram dispositivos eletrônicos inteligentes destinados ao consumidor final.
- O projeto resultou em uma linguagem de programação baseada em C e C++
  - Oak (Carvalho): já existia;
  - Java: a cidade, o café.

Introdução (cont.)

Star seven

- Um controle remoto com interface *touchscreen*
- Tinha um mascote (Duke) que ensinava o usuário a utilizar o controle
- Infelizmente a tecnologia da época não estava preparada

Introdução (cont.)

Introdução (cont.)

- Na época, a internet estava ficando cada vez mais popular
- O equipe do *Green Project* começou a pensar em aplicações na internet
- A chave dessas aplicações deveria ser a interação

Introdução (cont.)

Principais Características

- Orientação a objetos;
- Portabilidade;
- Facilidades para criação de programas com recursos de rede;
- Sintaxe similar a C/C++;

Principais Características (cont.)

- Facilidades para criação de programas distribuídos e multitarefa;
- Desalocação automática de memória;
- Vasta coleção de bibliotecas (ou APIs);
- *Frameworks*.

APIs

- As bibliotecas de classes Java são também conhecidas como APIs (*Applications Programming Interface*)
  - Fornecidas por compiladores;
  - Fornecidas por fornecedores independentes de *software*
    - Aplicações gráficas;
    - Estruturas de dados;
    - Acessibilidade;

APIs (cont.)

- Sons;
- Programação distribuída e paralela;
- Bancos de dados
- Jogos
- E-mail
- Etc.

Criação de um Programa Java

- Cria-se um código fonte com a extensão **.java**;
- O programa é **compilado**

```
javac meuPrograma.java
```

- É gerado o **bytecode** (arquivo **.class**), que será interpretado durante a execução;

Criação de um Programa Java (cont.)

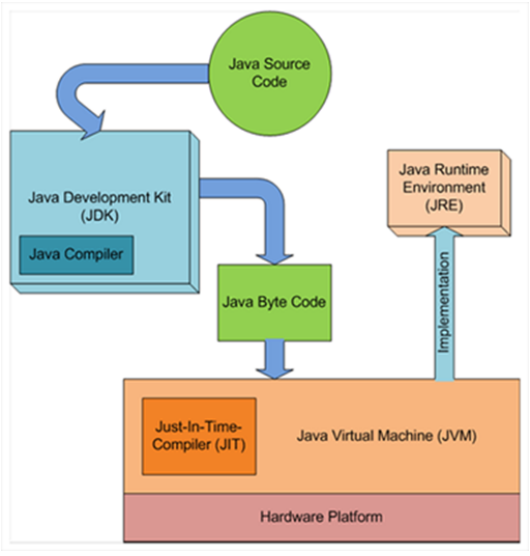
- O(s) arquivo(s) **.class** são carregados para a memória ;
- O interpretador (ou *Java Virtual Machine*) Java executa os programas carregados

```
java meuPrograma
```

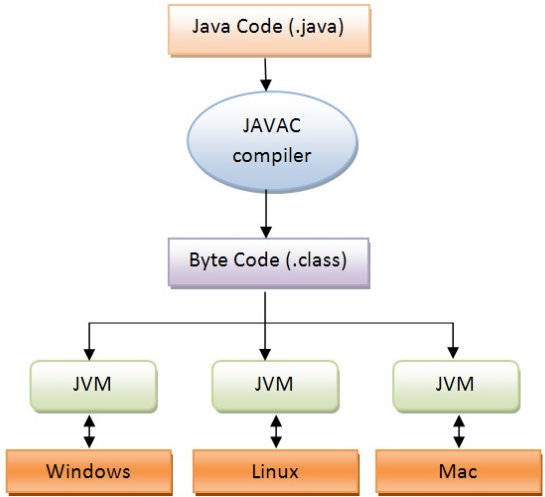
Bytecode

- Os programas em Java são compilados para *bytecode*
  - Uma forma intermediária de código, a ser interpretada pela máquina virtual Java (*Java Virtual Machine* - JVM).
- Isto permite maior portabilidade dos códigos Java
  - Qualquer sistema que inclua uma JVM executa qualquer código Java.
- A JVM é **responsável pelo gerenciamento dos aplicativos**, à medida em que estes são executados

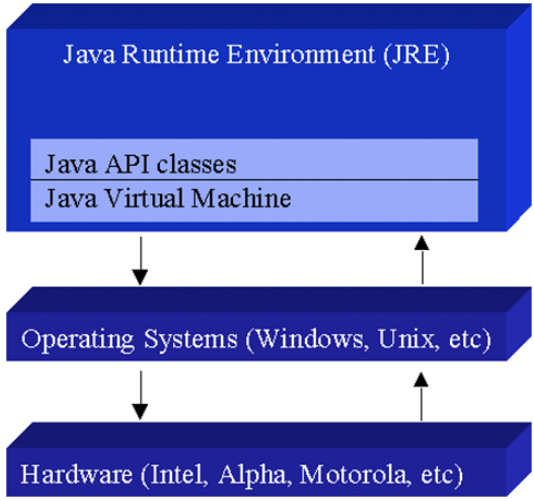
Bytecode (cont.)



Bytecode (cont.)



Bytecode (cont.)



Instruções de Saída

```
public class Welcome1
{
    //o método main inicia a execução da aplicação Java
    public static void main( String args[] )
    {
        //Java é case sensitive, cuidado com as letras maiúsculas
        System.out.println( "Welcome to Java
                             Programming!" );
    }
}
```

Instruções de Saída (cont.)

- Todo programa em Java consiste de pelo menos uma classe definida pelo programador
  - Padrão de nomenclatura igual ao C++;
- As **definições de classe** que começam com o especificador public **devem ser armazenadas em arquivos que possuam o mesmo nome da classe**
  - Definir mais de uma classe *public* no mesmo arquivo é um erro de sintaxe.

Instruções de Saída (cont.)

- A assinatura do método *main* é invariável

```
public static void main(String args[]);
```

- O *static* indica que o método será executado automaticamente pela JVM, sem necessidade de instanciar.

Instruções de Saída (cont.)

Especificador	Descrição
public	Acessível a todos os membros do programa
private	Acessível apenas internamente à classe
protected	Acessível internamente à classe, às subclasses e por classes do mesmo pacote
Acesso de pacote	Atribuído quando nenhum especificador é determinado. Acessível a todas as classes do mesmo pacote, através de uma referência a um objeto da classe.

Instrução de Saída

- **System.out** é conhecido como objeto de saída padrão

```
System.out.println( "Welcome to Java Programming!" );
```

- O método *println* imprime a *string* e quebra a linha ao final
  - Para não quebrar a linha, utiliza-se o método *print*.
  - Ambos também aceitam `'\n'` como caractere de nova linha.

Instrução de Saída (cont.)

Caractere de Escape	Descrição
\n	Nova linha
\t	Tabulação horizontal
\r	Retorno de carro
\\	Barra invertida
\"	Aspas duplas

printf em Java

- Também há o método **System.out.printf** (a partir do Java SE 5.0) para exibição de dados formatados
  - Similar ao printf de C/C++.

printf em Java (cont.)

```
public class Welcome4
{
    public static void main(String args[])
    {
        System.out.printf("%s\n%s\n", "Welcome to", "Java Programming")
            ;
    }
}
```

Importanto Classes

- O compilador utiliza instruções *import* para identificar e carregar classes usadas em um programa Java
  - As instruções de importação são divididas em dois grupos:
    - Núcleo do Java (nomes que começam com java);
    - Extensões do Java (nomes que começam com javax).

Importanto Classes (cont.)

- Java possui um rico conjunto de classes predefinidas
  - Agrupadas em **pacotes**;
  - Por padrão, o pacote **java.lang** é importado automaticamente
    - Classe *System*

Classe *Scanner*

```
import java.util.Scanner;
public class Adicao {
    public static void main(String args[]) {
        Scanner entrada = new Scanner(System.in);

        int numero1, numero2, soma;

        System.out.println("Informe o primeiro inteiro");
        numero1 = entrada.nextInt();//lê o primeiro inteiro
        System.out.println("Informe o segundo inteiro");
        numero2 = entrada.nextInt();//lê o segundo inteiro

        soma = numero1+numero2;
        System.out.printf("A soma é %d\n", soma);
    }
}
```

Classe *Scanner* (cont.)

- Um **Scanner** permite que o programa leia dados
  - Deve ser criado um objeto desta classe;
  - Os dados podem vir de diferentes fontes
    - Disco
    - Teclado
    - Etc.

Classe *Scanner* (cont.)

- Antes de utilizar um *Scanner*, o programa deve especificar qual é a origem dos dados
  - No nosso exemplo, *System.in* indica a entrada padrão.
- O método *nextInt()* lê o próximo número inteiro da entrada;
- É possível evitar a importação da classe Scanner, se utilizarmos o nome completo da classe

```
java.util.Scanner entrada = new java.util.Scanner(System.in);
```

Classe *Scanner* (cont.)

- Outros métodos úteis da classe Scanner são:
  - *next()*;
  - *nextByte()*;
  - *nextDouble()*;
  - *nextFloat()*;
  - *nextLine()*;

Classe *Scanner* (cont.)

- Estes métodos ainda possuem métodos similares *hasNext*, que determinam se ainda há possíveis tokens a serem lidos
  - Por exemplo, `hasNextInt()`.

Classe *Scanner* (cont.)

```
import java.util.*;
public class HasNext {
    public static void main(String[] args) {
        String s = "Hello World! \n 23";

        // create a new scanner with the specified String Object
        Scanner scanner = new Scanner(s);

        // check if the scanner has a token
        System.out.println("" + scanner.hasNext());

        // print the rest of the string
        System.out.println("" + scanner.nextLine());

        // check if the scanner has a token after printing the line
        System.out.println("" + scanner.hasNextInt());
    }
}
```

Classe *Scanner* (cont.)

```
true
Hello World!
true
```

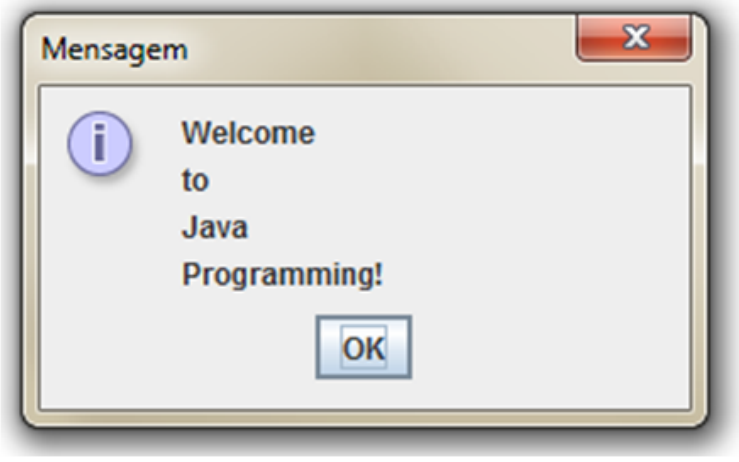
Caixas de Diálogo

- Caixas de diálogo são janelas utilizadas para informar ou obter dados ao usuário
  - Fornecem uma interface mais amigável que o terminal;
  - Janelas simples.
- No exemplo a seguir, será importada a classe **JOptionPane**, que oferece caixas de diálogo
  - A classe está contida no pacote **javax.swing**.

Exemplo 3

```
// Pacotes de extensão Java
import javax.swing.JOptionPane;
//importa a classe JOptionPane

public class Welcome4 {
    public static void main( String args[] ){
        //o parâmetro null posiciona a janela no meio da tela
        JOptionPane.showMessageDialog(null,
            "Welcome\nto\nJava\nProgramming!");
        //necessário em aplicações gráficas
        System.exit( 0 );   //termina o programa
    }
}
```

Exemplo 3 (cont.)



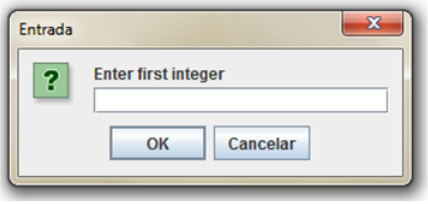
Exemplo 4

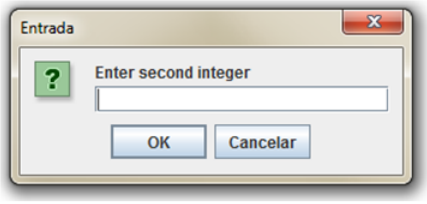
```
import javax.swing.JOptionPane;

public class Addition {
   public static void main( String args[] )
   {
      String firstNumber;    //primeira string digitada pelo usuário
      String secondNumber;   //segunda string digitada pelo usuário
      int number1;           //primeiro número
      int number2;           //segundo número
      int sum;               //soma

      //lê o primeiro número como uma string
      firstNumber = JOptionPane.showInputDialog("Enter first integer");

      //lê o segundo número como uma string
      secondNumber = JOptionPane.showInputDialog("Enter second integer"
          );
```

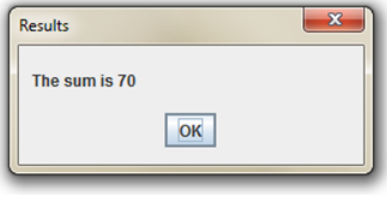
Exemplo 4 (cont.)

```
        // converte os números de String para int
        number1 = Integer.parseInt( firstNumber );
        number2 = Integer.parseInt( secondNumber );

        // adiciona os numeros
        sum = number1 + number2;

        // mostra o resultado
        JOptionPane.showMessageDialog( null, "The sum is "
          + sum, "Results", JOptionPane.PLAIN_MESSAGE );

        System.exit( 0 );
    }
}
```

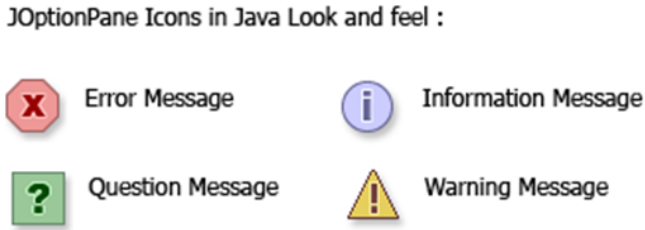
Exemplo 4 (cont.)







Constantes *JOptionPane*



ERROR_MESSAGE	INFORMATION_MESSAGE
QUESTION_MESSAGE	WARNING_MESSAGE
PLAIN_MESSAGE	

Erro Comum

- Separamos texto e o conteúdo de variáveis pelo operador +
  - Converte o valor da variável e a concatena no texto.
- Suponha $y = 5$:
  - "y+2 = "+y+2 imprime "y+2 = 52";
  - "y+2 = "+(y+2) imprime "y+2 = 7".

Operadores e Palavras Reservadas

Operadores (precedência)	Associatividade	Tipo
[], ., ()	Esquerda para direita	Posição, Invocação
++, --, +, -, !, ~	Direita para esquerda	Unário (pré fixo)
***, /, %**	Esquerda para direita	Multiplicativo
+, -	Esquerda para direita	Aditivo
<<, >>, >>>	Esquerda para direita	Shift
<, <=, >, >=, instanceof	Esquerda para direita	Relacional, objeto ou tipo
==, !=	Esquerda para direita	Igualdade
&&	Esquerda para direita	E lógico
||	Esquerda para direita	OU lógico
?:	Direita para esquerda	Condicional
=, +=, -=, *=, /=, %=	Direita para esquerda	Atribuição

Palavras Reservadas

abstract	continue	for	native	strictfp	volatile
boolean	default	goto	new	super	while
break	do	if	package	switch	synchronized
byte	double	implements	private	this	
case	else	import	protected	throw	
catch	extends	instanceof	public	throws	
char	final	int	return	transient	
class	finally	interface	short	try	
const	float	long	static	void	

Similaridades com C/C++

- Comentários;
- Operadores relacionais;
- Atribuições simplificadas;
- Incremento e decremento;
- Operadores lógicos;

Similaridades com C/C++ (cont.)

- Desvio Condicional
  - if , if −else, operador ternário (?:) e aninhamentos.
- Estrutura de seleção
  - switch−case
- Estruturas de repetição
  - while , do−while, for e aninhamentos

Vetores

- Em Java, os vetores são muito semelhantes aos de C++
  - O primeiro índice é zero.

```
int c[]; //declara um vetor
c = new int[ 12 ]; //cria o vetor e o associa a uma variável
```

Vetores (cont.)

- Os vetores possuem o atributo público *length*, que armazena o tamanho do vetor

```
int TAMANHO = 12;
int c[] = new int[TAMANHO];

for(int i = 0; i<c.length; i++)
      System.out.printf("%d ", c[i]);
```

Vetores (cont.)

- Caso seja acessada uma posição fora dos limites de um vetor, a exceção **IndexOutOfBoundsException** ocorre;
- Uma das formas de evitar este tipo de exceção é utilizar o *for* aprimorado
  - O contador é baseado no conteúdo do vetor

Vetores (cont.)

```
public class ForAprimorado
{
    public static void main(String Args[])
    {
        //cria e inicializa o vetor
        int  vet[] = {1, 2, 5, 10, 15, 20, 32};

        //o contador do for é associado aos elementos do vetor
        for(int i : vet)
            //imprime cada elemento do vetor
            System.out.printf("%d\n", i);
       }
}
```

Matrizes

- Matrizes, ou vetores multidimensionais em Java são semelhantes às matrizes em C++;
- As declarações abaixo são válidas tanto em Java quanto em C++:

```
int a[][] = { { 1, 2 }, { 3, 4, 5 } };

int b[][] = new int[ 3 ][ 4 ];

int c[][] = new int[ 2 ][ ]; //cria duas linhas
c[ 0 ] = new int[ 5 ]; //cria 5 colunas para a linha 0
c[ 1 ] = new int[ 3 ]; //cria 3 colunas para a linha 1
```

Matrizes (cont.)

```
public class VetorBidimensional
{
    public static void main (String args[])
    {
        //matriz estática
        int[][] tabuleiro = new int[8][8];
        int[][] dinamico;

        //aloca a primeira dimensão
        dinamico = new int[10][];

        //aloca a segunda dimensão
        for (int i=0; i < dinamico.length; i++)
            dinamico[i] = new int[i+1];
    }
}
```

API Java

- A especificação da API Java pode ser encontrada em: `https://www.oracle.com/java/technologies/javase-jdk19-doc-downloads.html`
- Descrição de todos os pacotes e suas respectivas classes, interfaces, exceções e erros.

API Java (cont.)

- **java.awt**: interfaces gráficas;
- **java.io**: entrada e saída;
- **java.lang**: classes básicas para programas Java;
- **java.math**: operações matemáticas com precisão arbitrária;
- **java.net**: aplicações que utilizam rede;
- **java.rmi**: programação distribuída;
- **java.sql**: banco de dados;
- **java.util**: coleções, utilidades de data e hora, internacionalização e miscelânea (*tokenizer*, números aleatórios, etc.).

Classes e Métodos

- Vejamos um exemplo conhecido sobre classes e métodos
  - *GradeBook* (diário de classe).
- Notem que há algumas semelhanças com a sintaxe de C++
  - E algumas diferenças.

Classes e Métodos (cont.)

```
public class GradeBook
{
    public void displayMessage()
    {
        System.out.println("Bem vindo ao Diario de Classe");
    }
}


public class DriverGradeBook
{
    public static void main(String Args[])
    {
        GradeBook meuDiario = new GradeBook();

        meuDiario.displayMessage();
    }
}
```

Classes e Métodos (cont.)

- Classes públicas são armazenadas em arquivos diferentes;
- O *main* fica dentro de uma classe, obrigatoriamente
  - Logo, é um método;
  - O *static* em seu cabeçalho indica que será executado automaticamente.
- Para instanciar um objeto, é necessário utilizar o operador new()
- Não é necessário importar a classe *GradeBook.java* no *driver*
  - Automático, pois estão no mesmo **pacote** – o pacote padrão.

Classes e Métodos (cont.)

- Vamos alterar o exemplo anterior para agora considerar um parâmetro para o método *displayMessage()*

Classes e Métodos (cont.)

```
public class GradeBook
{
    public void displayMessage(String nomeCurso)
    {
        //o println foi substituído, porque gera erro de compilação
        System.out.printf("Bem vindo ao Diario de Classe de %s",
            nomeCurso);
    }
}
```

Classe *String*

- Java trata cadeias de caracteres utilizando a classe **String**
  - Com 'S';
  - Incluída no pacote *java.lang*
  - Não é necessário importar.

Classe *String* (cont.)

- Possui 83 métodos (Java SE 19.0) para manipular *Strings*
  - Sintaxe diferente de C++;
  - + 15 construtores diferentes;
  - + operadores sobrecarregados.

Lendo *String*

```
public class GradeBook
{
    public void displayMessage(String nomeCurso)
    {
        //o println foi substituído, porque gera erro de compilação
        System.out.printf("Bem vindo ao Diario de Classe de %s",
            nomeCurso);
    }
}
```

Lendo *String* (cont.)

```
import java.util.Scanner;

public class DriverGradeBook
{
    public static void main(String Args[])
    {
        GradeBook meuDiario = new GradeBook();
        Scanner entrada = new Scanner (System.in);
        String nome;

        System.out.println("Digite o nome do curso");
        //Lê a linha inteira, inclusive espaços
        nome = entrada.nextLine();

        meuDiario.displayMessage(nome);
    }
}
```

Lista de Argumentos de Tamanho Variável

- Com **Listas de Argumentos de Tamanho Variável** ou **varargs**, podemos criar métodos que **recebem um número não especificado** de argumentos;
- Um tipo seguido de ... na lista de parâmetros de um método indica que este recebe um número de variáveis daquele tipo
  - Só pode ser feito uma vez por método;
  - Sempre no final da lista de parâmetros.

Lista de Argumentos de Tamanho Variável (cont.)

```
public class VarArgs{
    public static double media(double... numeros)  {
        double total = 0;

        for(double d: numeros)
            total+=d;

        return total/numeros.length;
    }

    public static void main(String args[]) {
        double d1 = 10.0;
        double d2 = 1.0;
        double d3 = 15.0;
        double d4 = 99.0;

        System.out.printf("%lf", media(d1, d2, d3, d4));
    }
}
```

Passagem de Parâmetros

- Duas formas de passar parâmetros para métodos ou funções são:
  - Por valor ou cópia (alterações não afetam a variável ou objeto original);
  - Por referência (alterações afetam a variável ou objeto original).

Passagem de Parâmetros (cont.)

- Java não permite que o usuário escolha qual será a forma de passagem dos parâmetros:
  - Tipos primitivos são sempre passados por valor;
  - Objetos e vetores são passados por referência
  - Vetores são passados por referência por uma questão de desempenho.

Escopo de Variáveis e Atributos

- As regras para o escopo de variáveis e atributos são as mesmas que em C++:
  - Variáveis locais só existem dentro do bloco de código ao qual pertencem, delimitados por { e };
  - Atributos são declarados dentro das classes e fora dos métodos;

Escopo de Variáveis e Atributos (cont.)

- Os especificadores de acesso determinam a visibilidade dos atributos
  - public , private e protected
- Consequentemente, o uso de *getters* e *setters* também é mantido;
- Em Java, os especificadores não delimitam regiões de uma classe (como em C++)
  - São definidos membro a membro.

Exemplo *GradeBook*

```
public class GradeBook{
   private String courseName;
   // método para definir o nome da disciplina
   public void setCourseName( String name ){
      courseName = name; // store the course name
   }

   // método para recuperar o nome da disciplina
   public String getCourseName() {
      return courseName;
   }

   // mostra a mensagem de bem vindas
   public void displayMessage() {
      // mostra a mensagem
      System.out.printf("Welcome to the grade book
        for\n%s!\n", getCourseName() );
   }
} // fim classe GradeBook
```

Exemplo *GradeBook* (cont.)

```
// biblioteca de entrada de dados
import java.util.Scanner;

public class GradeBookTest{
   // método principal que inicia a execução
   public static void main( String[] args )
   {
      // permite a captura de dados de entrada
      Scanner input = new Scanner( System.in );

      // cria um objeto da classe GradeBook
      GradeBook myGradeBook = new GradeBook();

      System.out.printf( "Initial course name is:
        %s\n\n", myGradeBook.getCourseName() );
```

Exemplo *GradeBook* (cont.)

```
        // prompt for and read course name
        System.out.println( "Please enter the
              course name:" );

        // lê uma linha de texto
        String theName = input.nextLine();

        // set courseName
        myGradeBook.setCourseName( theName );

        System.out.println(); // imprime uma linha em branco

        // mostra a mensagem de boas vindas
        myGradeBook.displayMessage();
    }
} // fim classe GradeBookTest
```

Exemplo *GradeBook* (cont.)

```
Initial course name is: null
Please enter the course name:
BCC Programacao Orientada a Objetos
Welcome to the GradeBook for
BCC Programacao Orientada a Objetos
```

FIM