



BCC221

# Programação Orientada a Objetos

Prof. Marco Antonio M. Carvalho

2014/2



UFOP

# Endereços Importantes

- Site da disciplina:  
<http://www.decom.ufop.br/marco/>
- Moodle:  
[www.decom.ufop.br/moodle](http://www.decom.ufop.br/moodle)
- Lista de e-mails:  
[bcc221-decom@googlegroups.com](mailto:bcc221-decom@googlegroups.com)
- Para solicitar acesso:  
<http://groups.google.com/group/bcc221-decom>



UFOP



# Avisos



# Avisos

UFOP



# Na aula passada

UFOP

- Processamento de Arquivos



# Na aula de hoje

UFOP

- Breve história do Java
- Criação de Um Programa Java
- Instruções de Saída
- Importando Classes
  - Classe Scanner
  - Caixas de Diálogo
- Operadores e Palavras Reservadas
- Vetores
- API Java
- Classes e Métodos
  - Passagem de Parâmetros
  - Escopo de Variáveis e Atributos
  - Construtores
  - Finalizadores e Coleta de Lixo Automática
  - Membros *static*
- Outros



# Breve História do Java

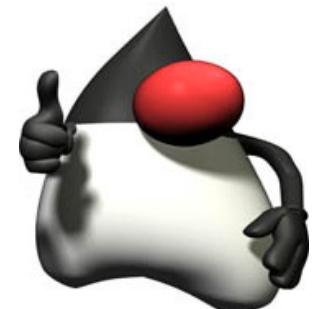
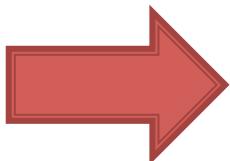
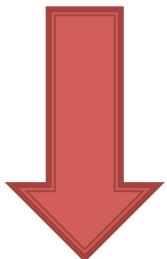
UFOP

- 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.



# Breve História do Java

UFOP





# Principais Características

UFOP

- Orientação a objetos;
- **Portabilidade**;
- Facilidades para criação de programas com recursos de rede;
- Sintaxe similar a C/C++;
- Facilidades para criação de programas distribuídos e multitarefa;
- Desalocação automática de memória;
- Vasta coleção de bibliotecas (ou APIs);
- *Frameworks*.



# APIs

UFOP

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

# Criação de um Programa Java



# Criação de um programa Java

UFOP

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

*java meuPrograma*



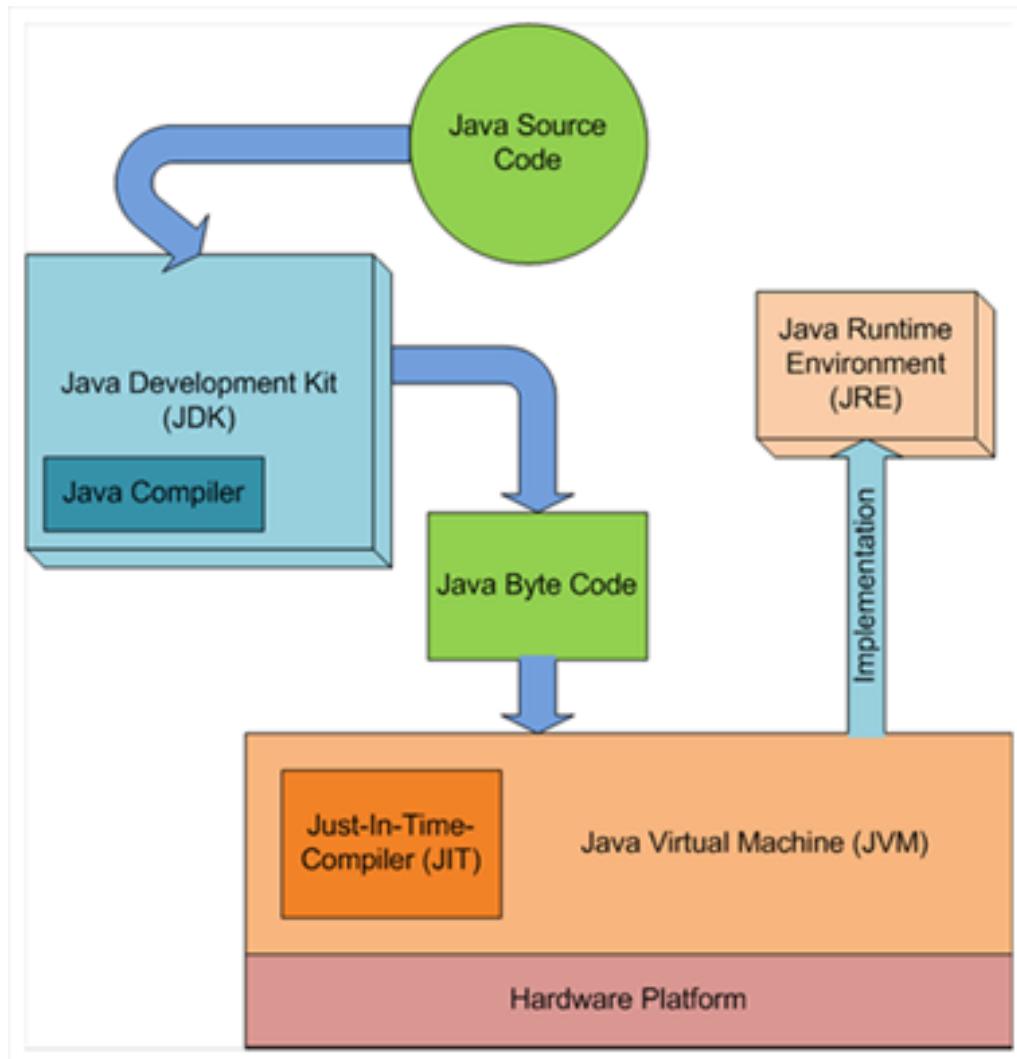
# Bytecode

UFOP

- 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
  - Possui grande controle sobre quais ações um determinado código pode realizar;
  - *Sandbox*.



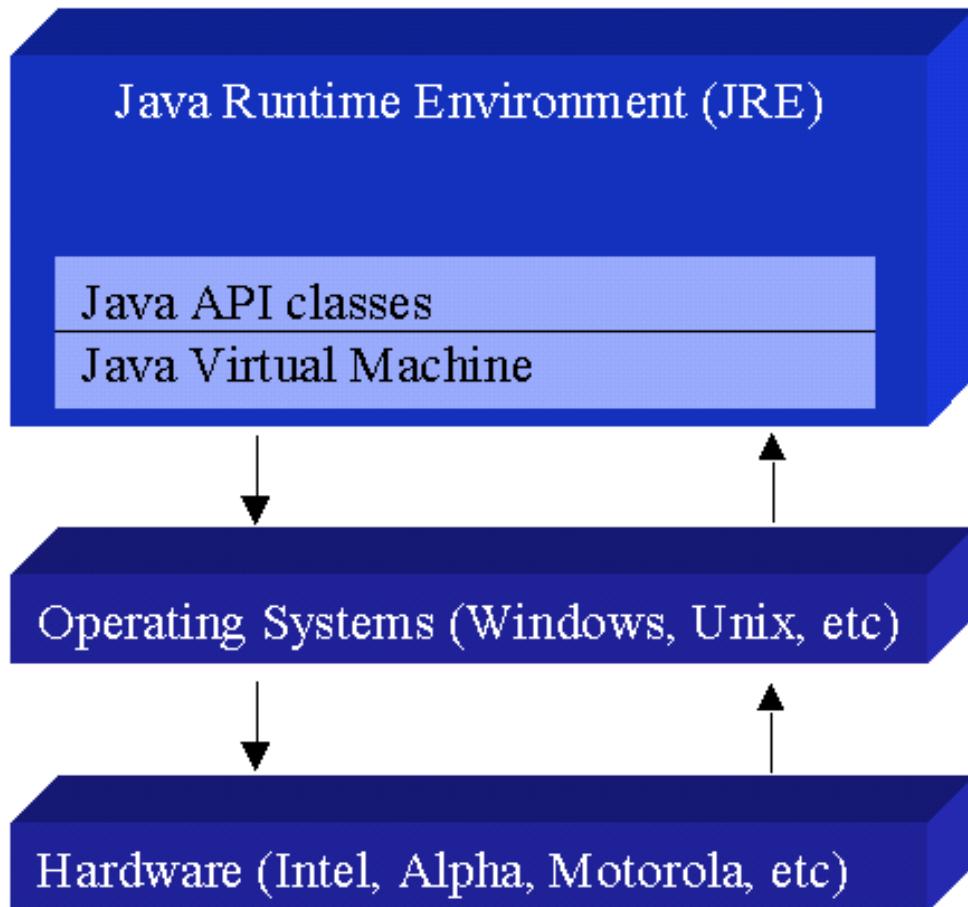
# Ciclo de um programa Java





# *Java Runtime Environment*

UFOP



# Instruções de Saída



# Primeiro Exemplo

UFOP

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



# Primeiro Exemplo

- 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.
- 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.



# Especificadores de Acesso

UFOP

Especificador	Descrição
<i>public</i>	Acessível a todos os membros do programa.
<i>private</i>	Acessível apenas internamente à classe.
<i>protected</i>	Acessível internamente à classe, às subclasses e por classes do mesmo pacote.
<i>Acesso de pacote</i>	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

UFOP

- *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.



# Caracteres de Escape

UFOP

Caractere de Escape	Descrição
\n	Nova linha.
\t	Tabulação horizontal.
\r	Retorno de carro (volta ao início da mesma linha).
\\"	Barra invertida
\\"	Aspas duplas.



# *printf* em Java

UFOP

- 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*

UFOP

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

# Importando Classes



# Importando Classes

UFOP

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

# *Classe Scanner*



# Classe Scanner

UFOP

```
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.printf("Informe o primeiro inteiro\n");
        numero1 = entrada.nextInt(); //lê o primeiro inteiro
        System.out.printf("Informe o segundo inteiro\n");
        numero2 = entrada.nextInt(); //lê o segundo inteiro

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



# Classe Scanner

UFOP

- 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.
- 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

UFOP

- Outros métodos úteis da classe Scanner são:
  - *next();*
  - *nextByte();*
  - *nextDouble();*
  - *nextFloat();*
  - *nextLine();*
- Estes métodos ainda possuem métodos similares *hasNext*, que determinam se ainda há possíveis *tokens* a serem lidos
  - Por exemplo, *hasNextInt()*.

# Caixas de Diálogo



# Caixas de Diálogo

UFOP

- 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

UFOP

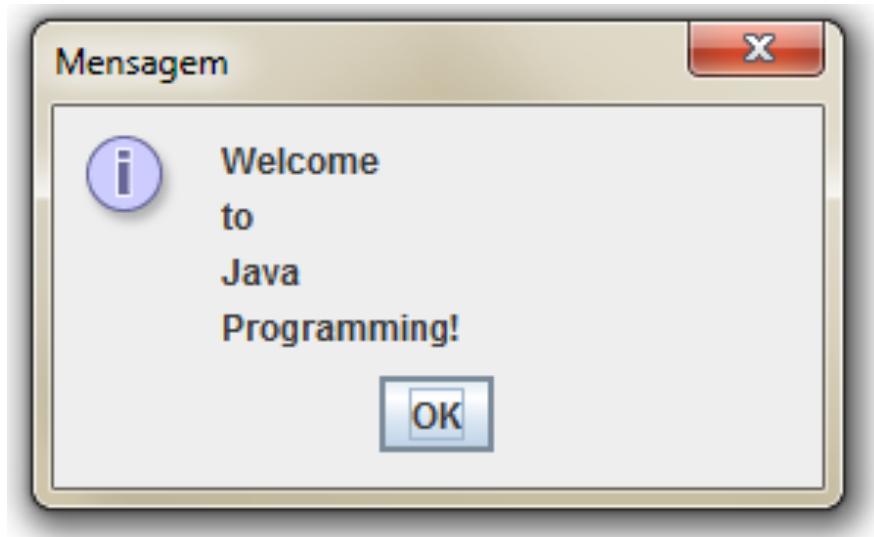
```
// Pacotes de extensão Java
import javax.swing.JOptionPane;
//importa a classe JOptionPane

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



# Exemplo 3

UFOP





# Exemplo 4

UFOP

```
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

UFOP

```
// 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 );
}
```



# Saída

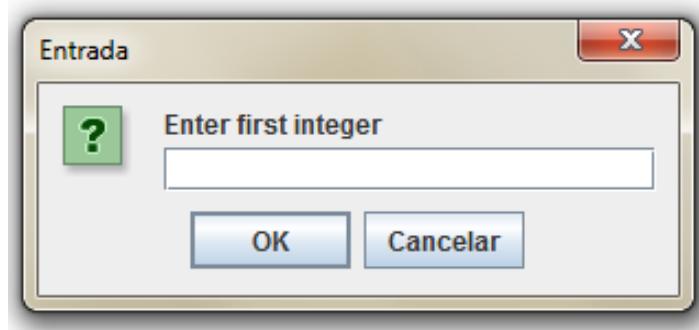
UFOP

Entrada X

?

Enter first integer

OK Cancelar

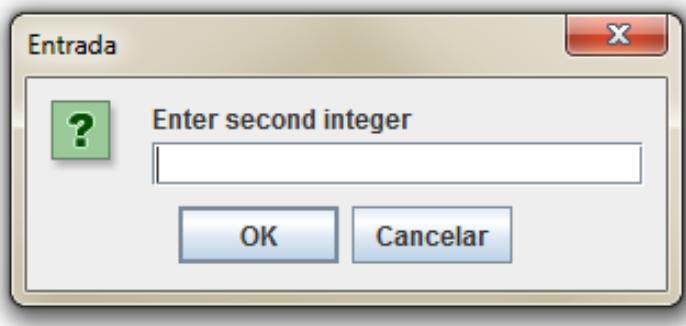


Entrada X

?

Enter second integer

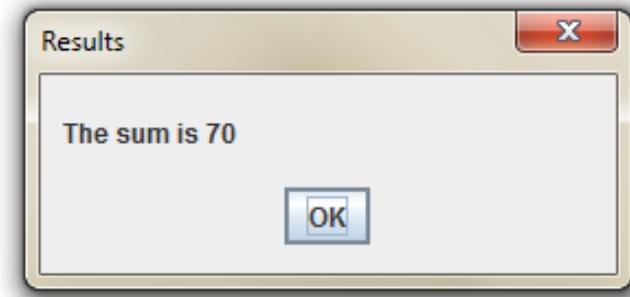
OK Cancelar



Results X

The sum is 70

OK





# Constantes JOptionPane

UFOP

JOptionPane Icons in Java Look and feel :



Error Message



Information Message



Question Message



Warning Message

<b>ERROR_MESSAGE</b>	<b>INFORMATION_MESSAGE</b>
<b>QUESTION_MESSAGE</b>	<b>WARNING_MESSAGE</b>
<b>PLAIN_MESSAGE</b>	



# Exemplo 4

- A *InputDialog* é outra caixa predefinida na *JOptionPane*
  - Caixa de diálogo de entrada;
  - O método *showInputDialog()* a mostra na tela.
- Os dados são lidos como *strings*, e depois convertidos para inteiros
  - As classes *String* e *Integer* são do pacote *java.lang*.
- O mesmo pode ser feito com as classes *Float* (*parseFloat*) e *Double* (*parseDouble*).



# Erro Comum

UFOP

- 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 em Java

UFOP

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



# Palavras Reservadas

UFOP

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++

UFOP

- Comentários;
- Operadores relacionais;
- Atribuições simplificadas;
- Incremento e decremento;
- Operadores lógicos;
- Desvio Condicional
  - *if, if-else, operador ternário (?:) e aninhamentos.*
- Estrutura de seleção
  - *switch-case.*
- Estruturas de repetição
  - *while, do-while, for e aninhamentos.*

# Vetores



# Vetores

UFOP

- 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
```

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

```
final int TAMANHO = 12;  
int c[] = new int[TAMANHO];  
  
for(int i = 0; i<c.length; i++)  
    System.out.printf("%d ", c[i]);
```



# Vetores

UFOP

- 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.



# for Aprimorado

```
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

UFOP

- 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

UFOP

```
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



# API Java

UFOP

- A especificação da API Java pode ser encontrada em:

<http://download.oracle.com/javase/8/docs/api/>

- Descrição de todos os pacotes e suas respectivas classes, interfaces, exceções e erros.



# Alguns Pacotes da API

UFOP

- ***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



# Classes e Métodos

UFOP

- 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

UFOP

```
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

UFOP

- 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()***
  - Uma exceção são *strings* constantes.
- Não é necessário importar a classe *GradeBook.java* no *driver*
  - Automático, pois estão no mesmo **pacote** – o pacote padrão.



# Classes e Métodos

UFOP

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



# Classes e Métodos

UFOP

```
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 %s", nomeCurso);
    }
}
```



# Classe *String*

UFOP

- Java trata cadeias de caracteres utilizando a classe *String*
  - Com 'S';
  - Incluída no pacote *java.lang*
    - Não é necessário importar.
- Possui 65 métodos (Java SE 7.0) para manipular *Strings*
  - Sintaxe diferente de C++;
  - + 15 construtores diferentes;
  - + operadores sobrecarregados.



# Lendo *strings*

UFOP

```
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 *strings*

UFOP

```
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);
    }
}
```

# Listas de Argumentos de Tamanho Variável

# Listas de Argumentos de Tamanho Variável



UFOP

- 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.

# Listas de Argumentos de Tamanho Variável



UFOP

```
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



# Passagem de Parâmetros

UFOP

- 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).
- 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



# Escopo de Variáveis e Atributos

UFOP

- As regras para o escopo de variáveis e atributos são as mesmas em Java:
  - 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;
  - 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
    - São definidos membro a membro.



# Escopo de Variáveis e Atributos

UFOP

```
public class GradeBook
{
    private String courseName;

    public void setCourseName(String nome)
    {
        //atribuição direta
        courseName = nome;
    }

    public String getCourseName()
    {
        return courseName;
    }

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



UFOP



**Continua na  
próxima aula...**

# Construtores



# Construtores

UFOP

- Java também utiliza construtores para inicializar objetos assim que são criados
  - Por padrão, o compilador fornece um construtor *default*, que não possui argumentos e inicializa os membros de um objeto
    - Tipos primitivos são zerados;
    - O construtor de objetos internos são chamados automaticamente.
  - O operador ***new*** invoca o construtor;
  - A chamada do construtor é indicada pelo nome da classe seguida de parênteses
    - O construtor deve ter o mesmo nome da classe;
    - Não retorna nada e não possui tipo.



# Construtores

UFOP

```
public class GradeBook
{
    private String courseName;

    public GradeBook(String nome)
    {
        setCourseName(nome);
    }

    public void setCourseName(String nome)
    {
        //atribuição direta
        courseName = nome;
    }

    public String getCourseName()
    {
        return courseName;
    }
}
```



# Construtores

UFOP

```
public class DriverGradeBook
{
    public static void main(String Args[])
    {
        GradeBook meuDiario = new GradeBook("BCC221");
    }
}
```

# **Finalizadores E Coleta de Lixo Automática**



# Finalizadores e Coleta de Lixo Automática

UFOP

- Toda classe em Java deriva da classe *Object*;
- Um dos métodos herdados é o *finalize*
  - Raramente utilizado;
  - Dos 6500 códigos da API Java, somente 50 o utilizam.
- Cada objeto criado consome recursos de sistema, como memória
  - A JVM realiza a coleta de lixo automática (*garbage collection*);
  - Quando não há mais referências a um objeto na memória, tal objeto é marcado para a coleta de lixo;
  - Quando o coletor de lixo (*garbage collector*) for executado, os recursos utilizados por aquele objeto serão liberados.

# Finalizadores e Coleta de Lixo Automática



UFOP

- Desta forma, estouros de memórias, comuns em C e C++, são menos propensas a ocorrer
  - O que não quer dizer que não ocorram.
- O método ***finalize*** é chamado pelo coletor de lixo para realizar a terminação do objeto antes que seus recursos sejam liberados;
- O método ***finalize***:
  - É especificado como *protected*;
  - Possui tipo *void*;
  - Não possui parâmetros.



# Finalizadores e Coleta de Lixo Automática

UFOP

- Um problema com o método *finalize* é que não há garantia de que o coletor de lixo será executado antes de o programa terminar
  - Logo, a execução de sua aplicação não deve depender dele.
- De fato, desenvolvedores profissionais indicam que o método *finalize* não é útil em aplicações Java para empresas
  - Outras técnicas de liberação de recursos devem ser utilizadas;
  - Geralmente, as classes relacionadas à manipulação de recursos providenciam outras maneiras de liberá-los.

# Membros *static*



# Membros *static*

UFOP

- Assim como em C++, o modificador ***static*** define membros que serão instanciados uma única vez
  - Ou seja, não haverá uma cópia para cada objeto;
  - Todos os objetos compartilham uma única cópia.
- O exemplo a seguir mostra a utilização de membros *static* e também do método *finalize*.



# Membros *static*

```
public class Rec
{
    //cria um único item para todos os objetos
    private static int n = 0;

    public Rec()
    {
        //incrementa o atributo static
        n++;
    }

    protected void finalize()
    {
        n--;
        System.out.println("Finalizou um objeto");
    }

    public static int getRec()
    {
        return n;
    }
}
```



# Membros *static*

UFOP

```
class RecTest
{
    public static void main(String args[])
    {
        System.out.println(Rec.getRec());
        Rec r1 = new Rec();
        Rec r2 = new Rec();
        Rec r3 = new Rec();
        System.out.println(Rec.getRec());
        //faz com que a JVM marque os objetos como
        //sem referência
        r1 = null;
        r2 = null;
        r3 = null;
        //faz com que a coleta do lixo seja realizada
        System.gc();
        System.out.println(Rec.getRec());
    }
}
```



# Membros *static*

UFOP

- Se uma variável *static* não for inicializada, o compilador atribui um valor padrão;
- O método *static gc* do pacote *System* faz uma chamada explícita ao coletor de lixo
  - Não necessariamente todos os objetos serão coletados.
- Um método *static* não pode acessar membros não *static*
  - Um método *static* pode ser chamado mesmo quando não houver um objeto instanciado;
  - Pelo mesmo motivo, o ponteiro *this* não pode ser utilizado em um método *static*.
- Um método *static* deve ser invocado pelo nome da classe seguido de **.** ou por um objeto.

# Outros



# Geração de Números Aleatórios

UFOP

```
import java.util.Random;

public class RolaODado
{
    public static void main(String Args[])
    {
        //cria um objeto da classe Random
        Random aleatorio = new Random();

        for(int i = 0; i<10; i++)
            //O método nextInt gera um número inteiro entre 0 e n-1
            System.out.printf("%d\n", aleatorio.nextInt(6));
    }
}
```



UFOP



# Perguntas?



# Na próxima aula

UFOP

- Classes e Métodos
  - Um olhar mais profundo



UFOP



**FIM**