



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
- Lista de e-mails:
bcc221-decom@googlegroups.com
- Para solicitar acesso:
<http://groups.google.com/group/bcc221-decom>



UFOP



Avisos



Avisos

UFOP



Na aula passada

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



Na aula de hoje

UFOP

- Métodos *static*
- Classe *Math*
- Promoção de Argumentos
- Sobrecarga de Métodos
- Composição
- Enumerações
 - Enumerações e Classes
- *static import*
- Criando Pacotes
- Acesso de Pacote



Métodos *static*

- Embora os métodos sejam executados em resposta a chamadas de objetos, isto nem sempre é verdade
 - Eventualmente, um método pode executar ações que não são dependentes do conteúdo de um determinado objeto;
 - Tais métodos devem ser declarados *static*.
- Métodos *static* podem ser invocados utilizando-se o nome da classe seguido de `.` e o nome do método

`classe.metodo(argumentos);`

- De fato, esta é uma boa prática, para indicar que o método é *static*.

Classe Math



Classe *Math*

UFOP

- A classe *Math* está definida no pacote *java.lang*
 - Fornece uma coleção de métodos *static* que realizam cálculos matemáticos comuns;
 - Não é necessário instanciar um objeto da classe para poder utilizar seus métodos;
 - Por exemplo:
Math.sqrt(900.00);
- Os argumentos destes métodos podem ser constantes, variáveis ou expressões.



Classe Math

UFOP

Método	Descrição	Exemplo
<code>abs(x)</code>	Valor absoluto de x	$\text{abs}(23.7)$ é 23.7 $\text{abs}(0.0)$ é 0.0 $\text{abs}(-23.7)$ é 23.7
<code>ceil(x)</code>	Arredonda x para o menor inteiro maior que x	$\text{ceil}(9.2)$ é 10.0 $\text{ceil}(-9.8)$ é -9.0
<code>cos(x)</code>	Cosseno de x (x em radianos)	$\text{cos}(0.0)$ é 1.0
<code>exp(x)</code>	Exponencial e^x	$\text{exp}(1.0)$ é 2.71828 $\text{exp}(2.0)$ é 7.38906
<code>floor(x)</code>	Arredonda x para o menor inteiro não maior que x	$\text{floor}(9.2)$ é 9.0 $\text{floor}(-9.8)$ é -10.0
<code>log(x)</code>	Logaritmo natural de x (base e)	$\text{log}(\text{Math.E})$ é 1.0 $\text{log}(\text{Math.E} * \text{Math.E})$ é 2.0



Classe Math

UFOP

Método	Descrição	Exemplo
max(x,y)	Maior valor entre x e y	max(2.3, 12.7) é 12.7 max(-2.3, -12.7) é -2.3
min(x,y)	Menor valor entre x e y	min(2.3, 12.7) é 2.3 min(-2.3, -12.7) é -12.7
pow(x,y)	x elevado a y (x^y)	pow(2.0, 7.0) é 128.0 pow(9.0, 0.5) é 3.0
sin(x)	Seno de x (x em radianos)	sin(0.0) é 0.0
sqrt(x)	Raiz quadrada de x	sqrt(900.0) é 30.0
tan(x)	Tangente de x (x em radianos)	tan(0.0) é 0.0



Classe *Math*

UFOP

Constante	Valor
Math.PI	3.14159265358979323846
Math.E	2.7182818284590452354

- Declaradas *public final static*
 - Todas as classes podem utilizar;
 - São constantes;
 - Podem ser acessadas pelo nome da classe;
 - Somente uma cópia.

Promoção de Argumentos



Promoção de Argumentos

UFOP

- A promoção de argumentos consistem em converter o tipo de um argumento
 - Por exemplo, o método *Math.sqrt* espera um *double*, mas pode ser invocado passando-se um *int* como argumento;
 - A promoção é realizada automaticamente, desde que se respeite as regras de promoção
 - Especifica quais conversões podem ser realizadas sem a perda de dados.
 - Em uma expressão com dois ou mais tipos primitivos diferentes, cada valor é promovido ao tipo “mais abrangente”.



Promoção de Argumentos

UFOP

Tipo	Promoções Válidas
<i>double</i>	Nenhuma
<i>float</i>	<i>double</i>
<i>long</i>	<i>float</i> ou <i>double</i>
<i>int</i>	<i>long</i> , <i>float</i> ou <i>double</i>
<i>char</i>	<i>int</i> , <i>long</i> , <i>float</i> ou <i>double</i>
<i>short</i>	<i>int</i> , <i>long</i> , <i>float</i> ou <i>double</i> (mas não <i>char</i>)
<i>byte</i>	<i>short</i> , <i>int</i> , <i>long</i> , <i>float</i> ou <i>double</i> (mas não <i>char</i>)
<i>boolean</i>	Nenhuma (valores booleanos não são considerados números em Java)



cast

UFOP

- Considerando a tabela anterior, não é possível realizar a promoção de argumentos de tipos “mais altos” para tipos “mais baixos”;
- No entanto, é possível realizar o *cast* explícito
 - Assumindo o risco de erros de truncamento.
- Suponha que o método abaixo só aceita valores inteiros:

```
raizQuadrada((int) valorDouble);
```

Sobrecarga de Métodos



Sobrecarga de Métodos

UFOP

- Métodos com o mesmo nome podem ser declarados dentro de uma mesma classe
 - Desde que possuam um conjunto diferente de parâmetros;
 - **Sobrecarga de métodos.**
- Quando um método sobre carregado é invocado, o compilador Java seleciona o método apropriado
 - De acordo com o número, tipo e ordem dos argumentos passados para o método.
- Desta forma, podemos ter um conjunto de métodos com o mesmo nome que realizam o mesmo tipo de operação sobre argumentos diferentes.



Sobrecarga de Métodos

- Por exemplo, os métodos *abs()*, *min()* e *max()* da classe *Math* são sobrecarregados, cada um com quatro versões:
 - Uma com dois argumentos *double*;
 - Uma com dois argumentos *float*;
 - Uma com dois argumentos *int*;
 - Uma com dois argumentos *long*.
- Vejamos um exemplo de métodos que calculam o quadrado de um número *int* ou *double*.



Sobrecarga de Métodos

UFOP

```
public class Sobrecarga
{
    int quadrado(int num)
    {
        return num*num;
    }

    double quadrado(double num)
    {
        return num*num;
    }

    public void print()
    {
        System.out.printf("Quadrado de 7.5 e: %f", quadrado(7.5));
        System.out.printf("\nQuadrado de 7 e: %d", quadrado(7));
    }
}
```



Sobrecarga de Métodos

UFOP

```
public class TesteSobrecarga
{
    public static void main(String args[])
    {
        Sobrecarga teste = new Sobrecarga();
        teste.print();
    }
}
```



Erro Comum

UFOP

- Note que somente o tipo de retorno de um método não é suficiente para que o compilador o diferencie de outro com assinatura parecida
 - Erro de compilação.
 - Exemplo:

int quadrado(**int** num)

long quadrado(**int** num)



Sobrecarga de Construtores

UFOP

- Java permite que objetos de uma mesma classe sejam inicializados de formas diferentes
 - Através da sobrecarga de construtores;
 - Basta definir múltiplos construtores com assinaturas diferentes
 - Número e tipo de argumentos.



Sobrecarga de Construtores

```
public class Tempo
{
    private int h, m, s;

    public Tempo()
    {
        h = m = s = 0;
    }

    public Tempo(int hora)
    {
        h = hora;
        m = s = 0;
    }

    public Tempo(int hora, int minuto)
    {
        h = hora;
        m = minuto;
        s = 0;
    }

    public Tempo(int hora, int minuto, int segundo)
    {
        h = hora;
        m = minuto;
        s = segundo;
    }

    public static void main(String args[])
    {
        Tempo t = new Tempo();
        Tempo t2 = new Tempo(12);
        Tempo t3 = new Tempo(12, 30);
        Tempo t4 = new Tempo(12, 30, 00);
    }
}
```

Observação sobre Construtores em Java



UFOP

- Java permite que outros métodos possuam o mesmo nome que a classe
 - Embora não se tratem de construtores;
 - Não são chamados quando um objeto da classe é criado;
 - Possuem tipo de retorno.
- Um erro comum é colocar um tipo de retorno em um método com o mesmo nome da classe e confundi-lo com um construtor.

Observação sobre Construtores em Java



UFOP

```
public class ConstrutorFalso
{
    public int ConstrutorFalso()
    {
        System.out.println("Um objeto foi criado?");
        return 1;
    }

    public ConstrutorFalso()
    {
        System.out.println("Um objeto foi criado!");
    }

    public static void main(String args[])
    {
        ConstrutorFalso obj = new ConstrutorFalso();
    }
}
```

Composição



Composição

UFOP

- Uma classe Java pode ter referências a objetos de outras classes como membros
 - Composição, ou relacionamento *tem-um*.
- Por exemplo, um despertador precisa saber o horário atual
 - É razoável embutir duas referências a objetos de uma classe *Hora* como membros da classe *Despertador*.



Composição

UFOP

```
public class Hora
{
    private int h, m, s;

    public int getH()
    {
        return h;
    }

    public int getM()
    {
        return m;
    }

    public int getS()
    {
        return s;
    }

    public void setH(int valor)
    {
        h = valor;
    }

    public void setM(int valor)
    {
        m = valor;
    }

    public void setS(int valor)
    {
        s = valor;
    }
}
```



Composição

```
public class Despertador
{
    boolean ligado;
    Hora despertador, horarioAtual;

    public void setDespertador (int h, int m, int s, boolean valor)
    {
        despertador.setH(h);
        despertador.setM(m);
        despertador.setS(s);
        ligado = valor;
    }

    public void setHorarioAtual (int h, int m, int s)
    {
        horarioAtual.setH(h);
        horarioAtual.setM(m);
        horarioAtual.setS(s);
    }
}
```



Composição

UFOP

```
public class TesteDespertador
{
    public static void main(String args[])
    {
        Despertador d = new Despertador();
        d.setHorarioAtual(11, 30, 05);
        d.setDespertador(07, 0, 0, true);
    }
}
```

Enumerações



Enumerações

UFOP

- Uma **enumeração**, em sua forma mais simples, declara um conjunto de constantes representadas por um identificador
 - É um tipo especial de classe, definida pela palavra **enum** e um identificador;
 - Como em classes, **{** e **}** delimitam o corpo de uma declaração;
 - Entre as chaves, fica uma lista de constantes de enumeração, separadas por vírgula
 - Cada uma representando um valor único.



Enumerações

UFOP

```
import java.util.Random;  
public class Baralho  
{  
    private enum Naipe {COPAS, PAUS, OUROS, ESPADAS};  
    private enum Valor {A, DOIS, TRES, QUATRO, CINCO, SEIS, SETE, OITO, NOVE, DEZ, J, Q, K};  
    public void sorteiaCarta()  
    {  
        //pode conter COPAS, PAUS, OUROS ou ESPADAS  
        Naipe cartaNaipe;  
        //pode conter uma das constantes do enum Valor  
        Valor cartaValor;  
        int numero;  
        Random aleatorio = new Random();  
        switch(aleatorio.nextInt(4))  
        {  
            case 0: cartaNaipe = Naipe.COPAS; break;  
            case 1: cartaNaipe = Naipe.PAUS; break;  
            case 2: cartaNaipe = Naipe.OUROS; break;  
            case 3: cartaNaipe = Naipe.ESPADAS;  
        }  
    }  
}
```



Enumerações

UFOP

```
int temp = 1+aleatorio.nextInt(13);

switch (temp)
{
    case 1: cartaValor = Valor.A; break;
    case 2: cartaValor = Valor.DOIS; break;
    case 3: cartaValor = Valor.TRES; break;
    case 4: cartaValor = Valor.QUATRO; break;
    case 5: cartaValor = Valor.CINCO; break;
    case 6: cartaValor = Valor.SEIS; break;
    case 7: cartaValor = Valor.SETE; break;
    case 8: cartaValor = Valor.OITO; break;
    case 9: cartaValor = Valor.NOVE; break;
    case 10: cartaValor = Valor.DEZ; break;
    case 11: cartaValor = Valor.J; break;
    case 12: cartaValor = Valor.Q; break;
    case 13: cartaValor = Valor.K; break;
}
```



Enumerações

UFOP

- Variáveis do tipo *Naipe* só podem receber valores definidos na enumeração
 - Caso contrário, ocorrerá erro de compilação.
- Cada valor é acessado como um membro, separado do nome da enumeração pelo operador `.`;
- Por padrão, utiliza-se apenas letras maiúsculas para denotar as constantes de uma enumeração;
- Uma constante de enumeração
 - Não pode ser impressa (sem *cast*);
 - Não pode ser comparada (a princípio) com tipos primitivos.



Enumerações

UFOP

- Um *enum* é implicitamente declarado como final
 - Também são implicitamente declarados como *static*;
 - Qualquer tentativa de criar um objeto de um *enum* com o operador new resulta em erro de compilação.
- Um *enum* pode ser utilizado em qualquer situação em que constantes possam ser utilizadas
 - Rótulos de *case*;
 - For aprimorado.

Enumerações e Classes



Enumerações e Classes

UFOP

- Um *enum* pode ser mais do que um simples conjunto de constantes
 - De fato, um *enum* pode ter atributos, construtores e métodos;
 - Cada constante é na verdade um objeto, com suas próprias cópias dos atributos;
 - Como em uma classe.



Book.java

UFOP

```
public enum Book
{
    //declara as constantes do enum
    JHTP6( "Java How to Program 6e", "2005" ),
    CHTP4( "C How to Program 4e", "2004" ),
    IW3HTP3( "Internet & World Wide Web How to Program 3e", "2004" ),
    CPPHTTP4( "C++ How to Program 4e", "2003" ),
    VBHTTP2( "Visual Basic .NET How to Program 2e", "2002" ),
    CSHARPHTTP( "C# How to Program", "2002" );

    //atributos
    private final String title;
    private final String copyrightYear;

    // construtor
    Book( String bookTitle, String year )
    {
        title = bookTitle;
        copyrightYear = year;
    }
}
```



Book.java

UFOP

```
//getter
public String getTitle()
{
    return title;
}

//getter
public String getCopyrightYear()
{
    return copyrightYear;
}
```



Enumerações e Classes

```
//importado por causa do método range
import java.util.EnumSet;

public class EnumTest
{
    public static void main( String args[] )
    {
        System.out.println( "Todos os livros:\n" );

        // imprime todos os livros do enum Book
        for (Book book: Book.values())
            System.out.printf( "%-10s%-45s%s\n", book, book.getTitle(),
                               book.getCopyrightYear());

        System.out.println( "\nImprimindo um intervalo de constantes:\n" );

        // imprime os quatro primeiros livros
        for (Book book : EnumSet.range(Book.JHTP6, Book.CPPHTP4))
            System.out.printf( "%-10s%-45s%s\n", book, book.getTitle(),
                               book.getCopyrightYear());
    }
}
```



Enumerações e Classes

UFOP

Todos os livros:

JHTP6 Java How to Program 6e	2005
CHTP4 C How to Program 4e	2004
IW3HTP3 Internet & World Wide Web How to Program 3e	2004
CPPHTP4 C++ How to Program 4e	2003
VBHTP2 Visual Basic .NET How to Program 2e	2002
CSHARPHTP C# How to Program	2002

Exibindo um intervalo de constantes do enum:

JHTP6 Java How to Program 6e	2005
CHTP4 C How to Program 4e	2004
IW3HTP3 Internet & World Wide Web How to Program 3e	2004
CPPHTP4 C++ How to Program 4e	2003



Enumerações e Classes

UFOP

- O método estático *values()* retorna um vetor de constantes do *enum*
 - Na ordem em que foram declaradas;
 - Criado automaticamente para cada *enum*.
- Quando uma constante é convertida para *String*, o próprio identificador é o conteúdo da *string*.
- O método *range()* da classe *EnumSet* é utilizado para determinar um intervalo dentro de um *enum*
 - Retorna um *EnumSet* que contém as constantes do intervalo, incluindo os limites;
 - Também pode ser percorrido por um for aprimorado.



UFOP



**Continua na
próxima aula...**

static import



static import

UFOP

- Uma declaração ***static import*** permite que referenciamos membros *static* importados como se fossem declarados na classe em que os usa
 - O nome da classe e o operador `.` não são necessários.
- Existem duas sintaxes para um *static import*
 - Uma que importa apenas um membro *static* em particular (***single static import***);
 - Uma que importa todos os membros *static* de uma classe (***static import on demand***).



static import

UFOP

- *Single static import*

```
import static pacote.Classe.membroStatic;
```

- *Static import on demand*

```
import static pacote.Classe.*;
```



static import

UFOP

```
//static import on demand
import static java.lang.Math.*;
public class StaticImportTest
{
    public static void main( String args[] )
    {
        System.out.printf( "sqrt( 900.0 ) = %.1f\n", sqrt(900.0));
        System.out.printf( "ceil( -9.8 ) = %.1f\n", ceil(-9.8));
        System.out.printf( "log( E ) = %.1f\n", log(E));
        System.out.printf( "cos( 0.0 ) = %.1f\n", cos(0));
    }
}
```



static import

UFOP

- Note que não é necessário utilizar o nome da classe Math para invocar os métodos *sqrt*, *ceil*, *log* e *cos*.

Criando Pacotes



Criando Pacotes

UFOP

- À medida em que as aplicações se tornam mais complexas, pacotes nos ajudam a gerenciar nossos componentes
 - Também facilitam o reuso de software ao permitir que nossos programas importem classes de outros pacotes;
 - Adicionalmente, ajudam a resolver problemas de conflito de nomes, fornecendo uma padronização.



Criando Pacotes

UFOP

- Para criar um pacote, é necessário:
 - Declare uma classe **pública**
 - Se não for pública, só poderá ser utilizada por outras classes do mesmo pacote.
 - Defina um **nome para o pacote** e adicione a declaração de pacote ao código fonte
 - Só pode haver uma declaração de pacote por código-fonte, e deve preceder todas as outras declarações no arquivo.
 - **Compilar a classe**
 - Ela será armazenada no diretório adequado.



Criando um Pacote

UFOP

```
//define a criação do pacote
package br.ufop.decom.pacote;

public class Classe
{
    //método de exemplo
    public void print()
    {
        System.out.println("Este é um pacote de exemplo!");
    }
}
```



Criando Pacotes

UFOP

- As classes que definem o pacote devem ser compiladas apropriadamente para que seja gerada a estrutura de diretórios
 - javac -d . Pacote.java*
- O **.** indica que a estrutura de diretórios deve ser criada a partir do diretório atual
 - Cada nome separado por **.** no nome do pacote define um diretório;

br

 ↳ ufop

 ↳ decom

 ↳ pacote.class



Importando o Pacote Criado

```
//importa a classe criada no pacote
import br.ufop.decom.pacote.Classe;

public class TestePacote
{
    public static void main(String args[])
    {
        //instancia um objeto da classe de exemplo
        Classe obj = new Classe();

        //invoca o método estático da classe
        //definida no pacote
        obj.print();
    }
}
```



Importando o Pacote Criado

UFOP

- Uma vez que a classe foi compilada e armazenada em seu pacote, ela pode ser importada em outros programas;
- Quando a classe que importa é compilada, o ***class loader*** procura os arquivos .class importados:
 - Nas classes padrão do JDK;
 - No pacotes opcionais;
 - No *classpath*
 - Lista de diretórios em que as classes estão localizadas.



classpath

UFOP

- Por padrão, o *classpath* consiste apenas do diretório atual, porém, pode ser modificado:
 - Através da opção –*classpath* no compilador javac;
 - Ajustando a variável de ambiente CLASSPATH
 - Uma variável especial mantida pelo sistema operacional para que as aplicações procurem pelas classes em locais especificados.
- Para maiores informações sobre como ajustar o *classpath* em Linux e Windows

java.sun.com/javase/6/docs/technotes/tools/index.html



classpath

UFOP

- Para o nosso exemplo:

javac -d . Classe.java

javac TestePacote.java –classpath ./br/ufop/decom/pacote

Acesso de Pacote



Acesso de Pacote

UFOP

- Se um modificador de acesso não for especificado para um método ou atributo de uma classe, ele terá **acesso de pacote**
 - Em um programa de uma única classe, não há efeito;
 - Caso contrário, qualquer classe do pacote poderá acessar os membros de outra classe através de uma referência a um objeto dela.
- Classes armazenadas e compiladas em um mesmo diretório são consideradas como pertencentes a um mesmo pacote
 - O pacote *default*.



Acesso de Pacote

UFOP

```
public class PackageDataTest
{
    public static void main( String args[] )
    {
        PackageData packageData = new PackageData();

        //imprime a representação em String
        System.out.printf( "Depois de instanciado:\n%s\n", packageData );

        //altera os atributos do objeto diretamente
        packageData.number = 77;
        packageData.string = "Goodbye";

        //imprime a representação em String
        System.out.printf( "\nDepois de alterar os valores:\n%s\n", packageData );
    }
}
```



Acesso de Pacote

UFOP

```
class PackageData
{
    //atributos com acesso de pacote
    int number;
    String string;

    //construtor
    public PackageData()
    {
        number = 0;
        string = "Hello";
    }

    //converte e retorna a representação em String
    public String toString()
    {
        return String.format( "numero: %d; string: %s", number, string );
    }
}
```



Acesso de Pacote

UFOP

Depois de instanciado:
numero: 0; string: Hello

Depois de alterar os valores: numero: 77;
string: Goodbye



UFOP



Perguntas?



Na próxima aula

UFOP

- Herança
- Especificadores de Acesso
- Classe Object
- Exemplo
- Construtores em Subclasses
- Compilação
- Redefinição de Métodos
- Engenharia de Software com Herança



UFOP



FIM