

Java - Sobrecarga/Composição

BCC 221 - Programação Orientada a Objectos(POO)

Guillermo Cámara-Chávez

Departamento de Computação - UFOP



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* (cont.)

- ▶ 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*

- ▶ 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.

static import

- ▶ Uma declaração *static import* permite que referenciemos membros *static* importados como se fossem declarados na classe em que os usa
 - ▶ O nome da classe e o operador `.` não são necessários.

static import (cont.)

- ▶ 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 (cont.)

- ▶ *single static import*

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

- ▶ *single import on demand*

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

static import (cont.)

```
//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 (cont.)

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

Sobrecarga de Métodos

- ▶ Métodos com o mesmo nome podem ser declarados dentro de uma mesma classe
 - ▶ Desde que possuam um conjunto diferente de parâmetros;
 - ▶ Sobrecarga de métodos.

Sobrecarga de Métodos (cont.)

- ▶ Quando um método sobrecarregado 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 (cont.)

- ▶ 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*.
- ▶ Exemplo: criar os métodos que calcula o quadrado de um número *int* e *double*

Sobrecarga de Métodos (cont.)

```
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 (cont.)

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

Erro Comum

- ▶ 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)
```

Enumerações

- ▶ 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

Enumerações (cont.)

```
import java.util.Random;

public class Baralho
{
    private enum Naipes {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
        Naipes cartaNaipes;
        //pode conter uma das constantes do enum Valor
        Valor cartaValor;
        int numero;
        Random aleatorio = new Random();
    }
}
```

Enumerações (cont.)

```
switch (aleatorio.nextInt(4)) {  
    case 0: cartaNaipе = Naipе.COPAS; break;  
    case 1: cartaNaipе = Naipе.PAUS; break;  
    case 2: cartaNaipе = Naipе.OUROS; break;  
    case 3: cartaNaipе = Naipе.ESPADAS;  
}
```

Enumerações (cont.)

```
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 (cont.)

- ▶ Variáveis do tipo *Naipes* 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 `.`

Enumerações (cont.)

- ▶ 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 (cont.)

- ▶ 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 (cont.)

```
public enum Level {HIGH, MEDIUM, LOW}
```

► atribuições

```
Level level = Level.HIGH;
```

► comando condicional

```
if (level == Level.HIGH){  
    ...  
}  
else if (level == Level.MEDIUM){  
    ...  
}  
else if (level == Level.LOW){  
    ...  
}
```

Enumerações (cont.)

- comando de seleção múltipla

```
switch (level) {  
    case HIGH: ...; break;  
    case MEDIUM: ...; break;  
    case LOW: ...; break;  
}
```

- laço de repetição

```
for (Level level : Level.values()) {  
    System.out.println(level);  
}
```


Enumerações e Classes

- ▶ 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

Exemplo 1

```
public enum OpcoesMenu {  
    SALVAR(1), IMPRIMIR(2), ABRIR(3), VISUALIZAR(4), FECHAR(5);  
  
    private final int valor;  
    OpcoesMenu(int valorOpcao){  
        valor = valorOpcao;  
    }  
    public int getValor(){  
        return valor;  
    }  
}
```

Exemplo 1 (cont.)

```
public class TestadoraEnum {  
    public static void escolheOpcao(OpcoesMenu opcao){  
        if(opcao == OpcoesMenu.SALVAR){  
            System.out.println("Salvando o arquivo!");  
        }  
        else if(opcao == OpcoesMenu.IMPRIMIR){  
            System.out.println("Imprimindo o arquivo!");  
        }  
        else if(opcao == OpcoesMenu.ABRIR){  
            System.out.println("Abrindo o arquivo!");  
        }  
        else if(opcao == OpcoesMenu.VISUALIZAR){  
            System.out.println("Visualizando o arquivo!");  
        }  
        else if(opcao == OpcoesMenu.FECHAR){  
            System.out.println("Fechando o arquivo!");  
        }  
    }  
}
```

Exemplo 1 (cont.)

```
public static void escolheOpcaoSwitch(OpcoesMenu opcao){  
    switch(opcao){  
        case SALVAR:  
            System.out.println("Salvando o arquivo!"); break;  
  
        case IMPRIMIR:  
            System.out.println("Imprimindo o arquivo!"); break;  
  
        case ABRIR:  
            System.out.println("Abrindo o arquivo!"); break;  
        case VISUALIZAR:  
            System.out.println("Visualizando o arquivo!"); break;  
        case FECHAR:  
            System.out.println("Fechando o arquivo!");  
    }  
}
```

Exemplo 1 (cont.)

```
public static void main(String[] args) {  
    System.out.println("Usando Ifs");  
    for (OpcoesMenu op : OpcoesMenu.values() ) {  
        escolheOpcao(op);  
    }  
    System.out.println("Usando switch");  
    for (OpcoesMenu op : OpcoesMenu.values() ) {  
        escolheOpcaoSwitch(op);  
    }  
}
```

Exemplo 1 (cont.)

Usando `ifs`

Salvando o arquivo!

Imprimindo o arquivo!

Abrindo o arquivo!

Visualizando o arquivo!

Fechando o arquivo!

Usando `switch`

Salvando o arquivo!

Imprimindo o arquivo!

Abrindo o arquivo!

Visualizando o arquivo!

Fechando o arquivo!

Exemplo - Disciplina.java

```
package javaenum;  
  
public enum Disciplina {  
    BCC221("P00",4),  
    MTM122("Calculo", 6),  
    BCC390("Monografia I", 8),  
    BCC502("Metodologia", 2),  
    BCC265("Eletronica", 6),  
    BCC326("PDI", 4);  
  
    private final String nome;  
    private final int horas;
```

Exemplo - Disciplina.java (cont.)

```
Disciplina(String nome, int horas){  
    this.nome = nome;  
    this.horas = horas;  
}  
  
public String getNome(){  
    return nome;  
}  
  
public int getHoras(){  
    return horas;  
}  
}
```


Exemplo - DriverDisciplina.java

```
package javaenum;  
import java.util.EnumSet;  
  
public class JavaEnum {  
  
    public static void main(String[] args) {  
        System.out.println("Todas as disciplinas");  
        for (Disciplina disp : Disciplina.values())  
            System.out.printf("%-7s\t%-15s\t%2d\n",  
                               disp, disp.getNome(), disp.getHoras());  
    }  
}
```

Exemplo - DriverDisciplina.java (cont.)

```
System.out.println("\nIntervalo de disciplinas");
```

```
for (Disciplina disp : EnumSet.range(  
    Disciplina.MTM122, Disciplina.BCC502))  
    System.out.printf("%-7s\t%-15s\t%2d\n", disciplinas, disp.  
        getNome(), disp.getHoras());
```

```
Disciplina disp = Disciplina.valueOf("BCC326");  
System.out.println(disp.getNome() + " " + disp.getHoras());  
System.out.println(disp.ordinal());
```

```
}
```

```
}
```

Exemplo - DriverDisciplina.java (cont.)

Todas as disciplinas

BCC221	POO	4
MTM122	Calculo	6
BCC390	Monografia I	8
BCC502	Metodologia	2
BCC265	Eletronica	6
BCC326	PDI	4

Intervalo de disciplinas

MTM122	Calculo	6
BCC390	Monografia I	8
BCC502	Metodologia	2

PDI 4

5

Exemplo - DriverDisciplina.java (cont.)

- ▶ O método estático *values()* retorna um vetor de constantes do *enum*
 - ▶ Na ordem em que foram declaradas;
 - ▶ Criado automaticamente para cada *enum*.
- ▶ 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.

Exemplo - DriverDisciplina.java (cont.)

- ▶ Nos tipos Enum também existem outros métodos descritos abaixo.
 - ▶ `String toString()`: retorna uma String com o nome da instância (em maiúsculas).
 - ▶ `valueOf (String nome)`: retorna o objeto da classe enum cujo nome é a string do argumento.
 - ▶ `int ordinal ()`: retorna o número de ordem do objeto na enumeração.

Exemplo: Simulação de Embaralhamento

```
package casino;

public enum Naipe {
    COPAS(0), PAUS(1), OUROS(2), ESPADAS(3);
    private final int naipe;

    Naipe(int valor){
        naipe = valor;
    }

    public int getNaipe(){
        return naipe;
    }
}
```

Exemplo: Simulação de Embaralhamento (cont.)

```
package casino;

public enum Valor {
    A(1), DOIS(2), TRES(3), QUATRO(4), CINCO(5),
    SEIS(6), SETE(7), OITO(8), NOVE(9), DEZ(10),
    JACK(11), QUEEN(12), KING(13);

    private final int valor_carta;

    Valor(int valor){
        valor_carta = valor;
    }

    public int getValor(){
        return valor_carta;
    }
}
```

Exemplo: Simulação de Embaralhamento (cont.)

```
package casino;
```

```
public class Carta {  
    private Naipe naipe;  
    private Valor valor;
```

```
  
    public Carta(Naipe naipe, Valor valor) {  
        this.naipe = naipe;  
        this.valor = valor;  
    }
```

```
  
    public Naipe getNaipe(){  
        return naipe;  
    }
```

```
  
    public Valor getValor() {  
        return valor;  
    }
```


Exemplo: Simulação de Embaralhamento (cont.)

```
@Override  
public String toString() {  
    return "{" + valor + " de " + naipe + "}";  
}  
}
```

Exemplo: Simulação de Embaralhamento (cont.)

```
package casino;  
import java.util.Random;  
  
public class Deck {  
    private Carta[] cartas;  
    private static final Random randomNumber= new Random();  
  
    public Deck() {  
        cartas = new Carta[52];  
        fill();  
    }  
}
```

Exemplo: Simulação de Embaralhamento (cont.)

```
private void fill(){
    int i = 0;
    for (Naipes naipes : Naipes.values()){
        for (Valor valor : Valor.values() ) {
            cartas[i] = new Carta(naipes, valor);
            i++;
        }
    }
}
```

```
public void shuffle(){
    for (int i = 0; i < cartas.length; i++){
        int troca = randomNumber.nextInt(52);
        Carta tmp = cartas[i];
        cartas[i] = cartas[troca];
        cartas[troca] = tmp;
    }
}
```

Exemplo: Simulação de Embaralhamento (cont.)

```
@Override
public String toString() {
    String str = "";
    int num = 0;
    for (int i = 0; i < 4; i++){
        for (int j = 0; j < 13; j++){
            str += (cartas[num++].toString() + " ");
            str += '\n';
        }

        return str;
    }
}
```

Exemplo: Simulação de Embaralhamento (cont.)

```
package casino;  
  
public class Casino {  
  
    public static void main(String[] args) {  
        // TODO code application logic here  
        Deck deck = new Deck();  
        System.out.println(deck);  
        deck.shuffle();  
        System.out.println("Embaralhando ...");  
        System.out.println(deck);  
    }  
}
```

Exemplo: Simulação de Embaralhamento (cont.)

{A de COPAS} {DOIS de COPAS} {TRES de COPAS} {QUATRO de COPAS} {CINCO de COPAS} {SEIS de COPAS} {SETE de COPAS} {OITO de COPAS} {NOVE de COPAS} {DEZ de COPAS} {JACK de COPAS} {QUEEN de COPAS} {KING de COPAS}

{A de PAUS} {DOIS de PAUS} {TRES de PAUS} {QUATRO de PAUS} {CINCO de PAUS} {SEIS de PAUS} {SETE de PAUS} {OITO de PAUS} {NOVE de PAUS} {DEZ de PAUS} {JACK de PAUS} {QUEEN de PAUS} {KING de PAUS}

{A de OUROS} {DOIS de OUROS} {TRES de OUROS} {QUATRO de OUROS} {CINCO de OUROS} {SEIS de OUROS} {SETE de OUROS} {OITO de OUROS} {NOVE de OUROS} {DEZ de OUROS} {JACK de OUROS} {QUEEN de OUROS} {KING de OUROS}

{A de ESPADAS} {DOIS de ESPADAS} {TRES de ESPADAS} {QUATRO de ESPADAS} {CINCO de ESPADAS} {SEIS de ESPADAS} {SETE de ESPADAS} {OITO de ESPADAS} {NOVE de ESPADAS} {DEZ de ESPADAS} {JACK de ESPADAS} {QUEEN de ESPADAS} {KING de ESPADAS}

Embaralhando ...

Exemplo: Simulação de Embaralhamento (cont.)

{CINCO de COPAS} {CINCO de PAUS} {A de PAUS} {SEIS de COPAS} {KING de PAUS} {QUATRO de ESPADAS} {KING de OUROS} {DEZ de OUROS} {SEIS de OUROS} {CINCO de OUROS} {OITO de PAUS} {NOVE de OUROS} {DOIS de COPAS}

{NOVE de COPAS} {QUEEN de PAUS} {OITO de OUROS} {DOIS de OUROS} {SEIS de PAUS} {JACK de PAUS} {SETE de ESPADAS} {A de OUROS} {SETE de COPAS} {JACK de COPAS} {DEZ de COPAS} {KING de ESPADAS} {NOVE de ESPADAS}

{SEIS de ESPADAS} {A de COPAS} {TRES de PAUS} {CINCO de ESPADAS} {QUEEN de COPAS} {TRES de COPAS} {OITO de COPAS} {DOIS de PAUS} {JACK de OUROS} {QUATRO de OUROS} {SETE de OUROS} {SETE de PAUS} {TRES de OUROS}

{TRES de ESPADAS} {QUATRO de PAUS} {A de ESPADAS} {DOIS de ESPADAS} {DEZ de ESPADAS} {OITO de ESPADAS} {JACK de ESPADAS} {QUEEN de OUROS} {NOVE de PAUS} {QUATRO de COPAS} {KING de COPAS} {DEZ de PAUS} {QUEEN de ESPADAS}

Criando Pacotes

- ▶ 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 (cont.)

- ▶ 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 Pacotes (cont.)

```
//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 (cont.)

- ▶ 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
│   └─ decomp
│       └─ pacote.class
```

Criando Pacotes (cont.)

```
//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();
    }
}
```

Criando Pacotes (cont.)

- ▶ **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.

Criando Pacotes (cont.)

- ▶ Para compilar um pacote:

```
javac -d . Classe.java
```

```
javac TestePacote.java
```

Acesso de Pacote

- ▶ 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 : *pacote default*

Compilação

► Sem package

1. Um único arquivo java na pasta

```
javac *.java  
java  NomeDoArquivo
```

2. Mais de um arquivo na pasta (os arquivos **java** devem estar sem o comando *package NomeDoPacote!*)

```
javac *.java  
java  NomeDoArquivoComMain
```


Compilação (cont.)

► Com package

1. Criando manualmente o pacote (pasta) onde serão colocados os **.java**, e onde serão criados os **.class**

```
| -Raiz
    | -NomePacote
        | -arquivos.java
```

Estando na raiz

```
javac -cp . NomePacote/*.java
java -cp . NomePacote/NomeDoArquivoComMain
```

Compilação (cont.)

2. Sem criar manualmente o diretório de pacote

```
| -Raiz
    | -arquivos.java
```

Estando na raiz, compilar:

```
javac -d . *.java
```

O compilador cria o diretório de pacote com os .class dentro

```
| -Raiz
    | -arquivos.java
    | -NomePacote
        | -arquivos.class
```

Estando na raiz, executar:

```
java NomePacote.NomeDoArquivoComMain
```

FIM