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.

Classe *Math* (cont.)

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

Classe *Math* (cont.)

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* (cont.)

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

Constante	**Valor**
Math.PI	3.14159265358979323846
Math.E	2.7182818284590452354

Promoção de Argumentos

- 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 (cont.)

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

Cast

- 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

- 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)
```

Sobrecarga de Construtores

- 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

Sobrecarga de Construtores (cont.)

```
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;
    }
```

Sobrecarga de Construtores (cont.)

```
    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

- 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 (cont.)

```
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

- 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.

Hora.java

```
package testedespertador;

public class Hora {
    private int h, m, s;
    public Hora(){
        this(0,0,0);
    }
    public Hora(int h, int m, int s){
        setH(h);
        setM(m);
        setS(s);
    }
    public Hora(Hora h){
        this(h.getH(), h.getM(), h.getS());
    }
    public void setH(int hora){
        h = hora > 0 & hora <24 ? hora : 0;
    }
```

Hora.java (cont.)

```
public int getH(){
    return h;
}
public void setM(int minutos){
    m = minutos > 0 & minutos <60 ? minutos : 0;
}

public int getM(){
    return m;
}

public void setS(int segundos){
    s = segundos > 0 & segundos <60 ? segundos : 0;
}

public int getS(){
    return s;
}
```

Hora.java (cont.)

```
    public String toUniversalString(){
        return String.format("%02d:%02d:%02d",
            getH(), getM(), getS());
    }

    public String toString(){
        return String.format("%d:%02d:%02d %s",
            (getH() == 0 || getH() == 12) ? 12 :
            getH() % 12, getM(), getS(),
            (getH() < 12 ? "AM" : "PM"));
    }
}
```

Despertador.java

```
package testedespertador;

public class Despertador {
    boolean ligado;
    Hora alarma, horaAtual;

    public Despertador(){
        alarma = new Hora();
        horaAtual = new Hora();
    }

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

Despertador.java (cont.)

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

        public String getAlarma(){
            return alarma.toString();
        }

        public String getHoraAtual(){
            return horaAtual.toUniversalString();
        }
    }
```

TesteDespertador.java

```
package testedespertador;

public class TesteDespertador {

    public static void main(String[] args) {
        Despertador d = new Despertador();
        d.setHoraAtual(14, 56, 20);
        d.setAlarma(18, 15, 0);
        System.out.println("Hora atual: " + d.
            getHoraAtual());
        System.out.println("Alarma: "
            + d.getAlarma());
    }

}
```

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 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();
```

Enumerações (cont.)

```
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 (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 *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 **.**;

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 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 - Disciplina.java

```
package javaenum;

public enum Disciplina {
    BCC221("POO",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, disciplinas.getNome(),
                disciplinas.getHoras());
    }
}
```

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
```

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.

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 static import*

```
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*

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.)

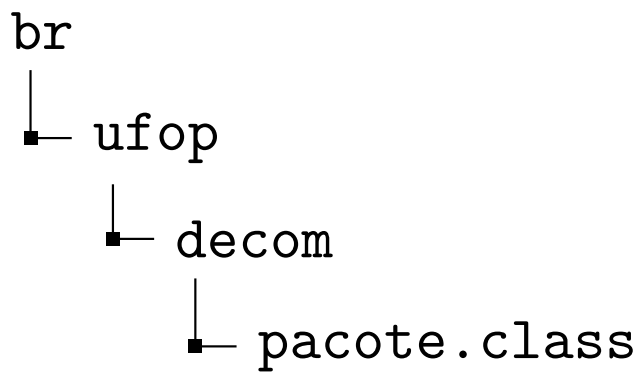
```
//define a criação do pacote
package br.ufop.decom.pacote;

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;



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*

FIM