fusberti@ic.unicamp.br

Métodos e Variáveis

prof. Fábio Luiz Usberti

MC322 - Programação Orientada a Objetos

Instituto de Computação - UNICAMP - 2014

Sumário

Módulos

Módulos de um aplicativo Java

- Os blocos fundamentais de um programa Java são os métodos, classes e pacotes.
- Aplicativos Java são desenvolvidos a partir de classes e métodos escritos pelo programador combinados com classes e métodos predefinidas (Java APIs ou outras).
- Assim como as Java APIs se encontram agrupadas em pacotes, as classes e métodos desenvolvidos por um programador também devem ser relacionadas e agrupadas em pacotes para serem importadas e reutilizadas por novos programas.

Módulos

Módulos de um aplicativo Java

- Os métodos, também denominados funções ou procedimentos, modularizam um programa ao dividir tarefas em unidades autocontidas.
- As instruções de um método, apesar de serem escritas uma única vez, podem ser utilizadas muitas vezes e em diversas localizações de um programa.
- Uma motivação para modularizar um programa em métodos está na abordagem dividir para conquistar, pois facilita a construção de programas a partir de blocos menores e mais simples.

Módulos

Módulos de um aplicativo Java

- A reutilização do código é outra motivação da modularização de um programa por métodos, pois novos programas podem ser criados a partir de métodos existentes.
- Para promover a reutilização, teste e depuração de um aplicativo, todos os métodos devem estar limitados à realização de uma única tarefa bem-definida e o nome do método deve expressar essa tarefa efetivamente.
- Se você não consegue escolher um nome conciso que expresse a tarefa de um método é possível que esse método esteja realizando tarefas demais. Nesse caso, tente particionar o método em tarefas menores.

Métodos estáticos

Declaração e chamada

- Frequentemente, a tarefa exercida por um método não está relacionada com o estado de um objeto.
- Nesse caso, é possível declarar esse método de modo estático (palavra-chave `static`) para informar ao compilador que o método se aplica à classe em que está declarado.
- Por exemplo, um método estático `myStaticMethod()` pertencente a uma classe importada `MyClass`, pode ser chamado da seguinte forma:

```
MyClass.myStaticMethod(<argumentos>);
```

- Se o método `myStaticMethod()` for chamado dentro da própria classe `MyClass`, então é possível omitir o nome da classe da chamada.

Métodos estáticos

Classe `Math`

- A classe `Math` contém muitos métodos estáticos para realizar operações matemáticas como exponenciação, logaritmo, raiz quadrada e funções trigométricas.
- A classe `Math` faz parte do pacte `java.lang` , que é implicitamente importado pelo compilador, assim não é ncessário importar a classe `Math` para utilizar seus métodos.
- A chamada de uma função estática da classe `Math` segue o mesmo modelo de chamada de qualquer outro método estático.

```
Math.<nome da função>(<argumentos>);
```

Métodos estáticos

Classe `Math`

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 não menor que *x*	`ceil( 9.2 )` é `10.0` `ceil( -9.8 )` é `-9.0`
`cos( x )`	co-seno trigonométrico de *x* (*x* em radianos)	`cos( 0.0 )` é `1.0`
`exp( x )`	método exponencial e^x	`exp( 1.0 )` é `2.71828` `exp( 2.0 )` é `7.38906`
`floor( x )`	arredonda *x* para o maior 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`
`max(x,y)`	maior valor de *x* e *y*	`max( 2.3, 12.7 )` é `12.7` `max( -2.3, -12.7 )` é `-2.3`
`min(x,y)`	menor valor de *x* e *y*	`min( 2.3, 12.7 )` é `2.3` `min( -2.3, -12.7 )` é `-12.7`

Métodos estáticos

Classe `Math`

Método	Descrição	Exemplo
pow(*x*,*y*)	*x* elevado à potência de *y* (isto é, x^y)	`pow( 2.0, 7.0 )` é `128.0` `pow( 9.0, 0.5 )` é `3.0`
`sin( x )`	seno trigonométrico 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 trigonométrica de *x* (*x* em radianos)	`tan( 0.0 )` é `0.0`

- Além dos métodos para cálculos matemáticos, a classe também possui dois atributos declarados como `public` , `static` e `final` que representam constantes matemáticas comumentes utilizadas.
- `Math.PI` : o valor `double` que mais se aproxima da constante π.
- `Math.E` : o valor `double` que mais se aproxima do número de Euler *e*.

Métodos estáticos

```
// MaximumFinder.java
// Determina o máximo dentre três números.
import java. util .Scanner;

public class MaximumFinder {

        // método que determina o máximo dentre três números passados como parâmetros
        public static double maximum(double x, double y, double z) {
                double maximumValue = x;
                if (y > maximumValue)
                        maximumValue = y;
                if (z > maximumValue)
                        maximumValue = z;
                return maximumValue;

        } // fim método maximum

        public static void main(String[] args) {

                Scanner input = new Scanner(System.in);

                System.out.print("Entre com três números: ");
                double number1 = input.nextDouble();
                double number2 = input.nextDouble();
                double number3 = input.nextDouble();

                // chama método estático para determinar o valor máximo.
                double result = MaximumFinder.maximum(number1, number2, number3);

                System.out.println("Maximo é " + result);

        } // fim main
} // fim classe MaximumFinder
```

Métodos estáticos

Exemplo `MaximumFinder`

- O método `maximum()` por ser estático, foi chamado sem a necessidade de instanciar um objeto.
- Pelo fato da chamada do método `maximum()` ocorrer na mesma classe onde ele está declarado, não é necessário especificar a classe, ou seja, a alternativa abaixo é igualmente válida:

```
double result = maximum(number1, number2, number3);
```

- Antes de implementar um método, verifique se não é possível reutilizar um código existente.

Métodos estáticos

Exemplo `MaximumFinder`

- Métodos retornam no máximo um valor, porém o valor retornado pode ser uma referência para um objeto que contém múltiplos valores armazenados.
- Não utilize variáveis de instância ou de classe quando elas são utilizadas em um único método. Ao invés disso, utilize variáveis locais.
- Declarar parâmetros de um mesmo tipo como `float x, y` resulta em erro de compilação. Cada parâmetro deve ter declarado seu próprio tipo na lista de parâmetros `float x, float y`.
- Cuidado ao utilizar o operador `+` de concatenção de strings. Lembre-se que em um operador os operandos são avaliados da esquerda para a direita.

```
System.out.println("x = " + 2 + 2);
```

Métodos estáticos

Por que o método `main` é declarado estático?

- Em um terminal, ao executar o comando `java` para uma certa classe, a JVM tenta chamar o método `main` dessa classe.

```
java MyClass <argumento1 ... argumentoN>;
```

- Declarar o método `main` como `static` permite ao JVM executar esse método sem necessidade de instanciar um objeto.
- A lista de argumentos informada pelo usuário será passada para o vetor de `String` do método `main` .

Variáveis estáticas

Variáveis de classe

- Todo objeto possui sua própria cópia de todas as variáveis de instância pertencentes à classe.
- Em certos casos, uma única cópia da variável, compartilhada por todos os objetos da classe, é suficiente.
- Nesses casos, uma variável de classe (ou variável estática, ou atributo estático) pode ser utilizada.
- As variáveis estáticas existem mesmo que nenhum objeto da classe exista. Elas se encontram disponíveis assim que a classe é lida para a memória.

Variáveis estáticas

Variáveis de classe

- Para acessar uma variável estática pública quando nenhum objeto da classe foi instanciado (ou mesmo depois), basta chamar o nome da classe, seguido de um ponto (`.`) e o nome da variável, como em `Math.PI` .
- No caso de uma variável estática privada, a chamada deve ocorrer dentro da classe em que ela está declarada. Alternativamente, é possível recuperar ou atribuir valor de uma variável estática privada utilizando os métodos acessores `get` e `set` .

Variáveis estáticas

Exemplo de uso: classe Employee

```
// Employee.java
// Exemplo de uso de variáveis estáticas.
public class Employee {

        private String name;
        private static int count = 0; // número de empregados criados

        // construtor
        public Employee(String name) {
                this.name = name;
                ++count; // incrementa o contador de empregados
                System.out.printf("construtor da classe Employee: %s ; count = %d\n", name, count);
        } // fim construtor

        public String getName() {
                return name;
        } // fim método getName

        public static int getCount() {
                return count;
        } // fim método getCount
} // fim classe Employee
```

Variáveis estáticas

Exemplo de uso: classe EmployeeDriver

```
// EmployeeDriver.java
// Testando a classe Employee.
public class EmployeeDriver {
        public static void main(String[] args) {

                // mostrando que o contador de empregados se inicia em zero.
                System.out.printf("Número de empregados antes da instanciação: %d\n", Employee.getCount());

                // criando dois empregados
                Employee e1 = new Employee("Fulano da Silva");
                Employee e2 = new Employee("Beltrano de Souza");

                System.out.println("\nEmpregados após as instanciações: ");
                System.out.printf("Employee.getCount() = %d\n", Employee.getCount());
                // nomes dos empregados
                System.out.printf("\nEmployee 1: %s\nEmployee 2: %s\n",
e1.getName(), e2.getName());

        } // fim main
} // fim classe EmployeeDriver
```

Variáveis finais

Palavra-chave `final`

- Seguindo o princípio do menor prilégio, um código deve conceder a visibilidade e o acesso necessários e suficientes para realizar as tarefas requisitadas.
- Isso torna o programa mais robusto, prevenindo modificações acidentais (ou maliciosas) de variáveis e a chamada de métodos que não deveriam estar acessíveis ao meio externo.
- A palavra-chave `final` diz ao compilador que o valor de uma variável deve ser mantido constante e qualquer tentativa de modificá-lo deverá resultar em erro.

Variáveis finais

Palavra-chave `final`

- Esse princípio se aplica tanto a variáveis de classe quanto de instância.
- O valor das variáveis finais deve obrigatoriamente ser atribuído ou durante a declaração, ou no método construtor.
- Atribuir valor a variáveis finais no método construtor permite que cada objeto possua sua própria constante.
- Se o valor de uma variável final for igual para todos os objetos, então a convenção é que ela seja declarada como estática (constante da classe).
- A palavra-chave `final` também se aplica a métodos e classes no contexto de herança.

Importação estática

Importação de métodos estáticos

- A importação estática permite que métodos e atributos estáticos de uma classe ou interface possam ser acessados diretamente pelos seus nomes.
- A sintaxe para uma importação estática é dada por:

```
import static packageName.ClassName.*;
```

- Um erro de compilação ocorre quando um programa tenta importar de duas ou mais classes métodos estáticos com a mesma assinatura ou atributos estáticos com o mesmo nome.

Importação estática

Exemplo de uso:

```
// StaticImportTest.java
// Importação estática de métodos da classe Math.
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( "E = %f\n", E );
                System.out.printf( "PI = %f\n", PI );
        } // fim main

} // fim classe StaticImportTest
```

Referências

1. Java: Como Programar, Paul Deitel & Heivey Deitel; Pearson; 7a. Ed. (no. chamada IMECC – 05.133 D368j)
2. Data Structures and Algorithms with Object Oriented Design Patterns in Java, Bruno Preiss; (http://www.brpreiss.com/books/opus6/)
3. The Java Tutorials (Oracle) (http://docs.oracle.com/javase/tutorial/)
4. Guia do Usuário UML, Grady Booch et. al.; Campus(1999)
5. Java Pocket Guide - Robert Liguori & Patricia Liguori; O'Reilley, 2008.