fusberti@ic.unicamp.br

Introdução aos Aplicativos Java

prof. Fábio Luiz Usberti

MC322 - Programação Orientada a Objetos

Instituto de Computação - UNICAMP - 2014

Sumário

Palavras-chave e palavras reservadas

Palavras-chave

- As palavras-chave são utilizadas pela linguagem e não podem ser adotadas como identificadores.

Keywords in Java				
abstract	default	if	private	this
assert	do	implements	protected	throw
boolean	double	import	public	throws
break	else	instanceof	return	transient
byte	enum	int	short	try
case	extends	interface	static	void
catch	final	long	strictfp	volatile
char	finally	native	super	while
class	float	new	switch	
continue	for	package	synchronized	

Palavras reservadas

- Além das palavras-chaves, há também as palavras reservadas `true`, `false` e `null` que também não podem ser utilizadas como identificadores.

Variáveis em Java

Tipos de variáveis

- A linguagem Java exige que o programador declare as variáveis antes de usá-las.
- A declaração envolve atribuir um nome e um tipo para a variável.
- Existem dois tipos para variáveis:
  - Tipos primitivos: são pré-definidos pela linguagem com palavras-chave específicas.

Classificação	Tipo	Descrição
Lógico	boolean	Pode possuir os valores true (verdadeiro) ou false (falso)
Inteiro	byte	Abrange de -128 a 127 (8 bits)
	short	Abrange de -32768 a 32767 (16 bits)
	int	Abrange de -2147483648 a 2147483647 (32 bits)
	long	Abrange de -2^{63} a $(2^{63})-1$ (64 bits)
Ponto Flutuante	float	Abrange de 1.40239846^{-46} a 3.40282347^{+38} com precisão simples (32 bits)
	double	Abrange de $4.9406564584124654^{-324}$ a $1.7976931348623157^{+308}$ com precisão dupla (64 bits)
Caracter	char	Pode armazenar um caracteres unicode (16 bits) ou um inteiro entre 0 e 65535

  - Tipos referenciados: as variáveis referenciadas são utilizadas para o acesso e manipulação de objetos e vetores.

Variáveis em Java

Classificações de variáveis

A linguagem Java define as seguintes classificações de variáveis:

- Variáveis de instâncias
- Variáveis de classe
- Variáveis locais
- Parâmetros

```
public class Bicycle {

    // Atributos
    protected int speed = 0;
    private static int numberGears = 21;

    // Método
    public void speedUp(int increment) {
        int newSpeed = speed + increment;
        speed = newSpeed;
    }

}
```

Variáveis em Java

Variáveis de instâncias

- Correspondem aos atributos de objetos, não-estáticos, ou seja, que não são declarados utilizando a palavra-chave `static` .
- Essas variáveis têm essa denominação pois seus valores são únicos para cada instância (objeto) da classe.
- Exemplo: A velocidade de uma bicicleta é independente da velocidade de outra bicicleta.

```
public class Bicycle {

    // Atributos
    protected int speed = 0;
    private static int numberGears = 21;

    // Método
    public void speedUp(int increment) {
        int newSpeed = speed + increment;
        speed = newSpeed;
    }

}
```

Variáveis em Java

Variáveis de classe

- Correspondem aos atributos de classe declarados com a palavra-chave `static`.
- O compilador entende que existe somente uma cópia dessa variável, independente do número de objetos instanciados.
- Exemplo: O número de marchas de um modelo específico de bicicleta poderia ser marcado como estático, pois esse número se aplica a todos as instâncias (objetos).

```
public class Bicycle {

    // Atributos
    protected int speed = 0;
    private static int numberGears = 21;

    // Método
    public void speedUp(int increment) {
        int newSpeed = speed + increment;
        speed = newSpeed;
    }

}
```

Variáveis em Java

Variáveis locais

- Do mesmo modo como um objeto armazena seu estado pelos valores de seus atributos, um método armazena seu estado através de suas variáveis locais.
- A declaração de uma variável local é similar à declaração das variáveis de instância, não havendo qualquer palavra-chave especificando-as.
- Para diferenciar uma variável local de uma variável de instância basta verificar o local onde elas estão declaradas. Uma variável local está declarada dentro do escopo de um método.

```
public class Bicycle {

    // Atributos
    protected int speed = 0;
    private static int numberGears = 21;

    // Método
    public void speedUp(int increment) {
        int newSpeed = speed + increment;
        speed = newSpeed;
    }

}
```

Variáveis em Java

Parâmetros

- Os parâmetros são variáveis que são passadas como argumentos de um método.
- Variáveis de instância, de classe e locais podem ser considerados atributos de objetos, classes e métodos, respectivamente. Os parâmetros, por sua vez, são variáveis que não têm a denominação de atributo.

```
public class Bicycle {

    // Atributos
    protected int speed = 0;
    private static int numberGears = 21;

    // Método
    public void speedUp(int increment) {
        int newSpeed = speed + increment;
        speed = newSpeed;
    }

}
```

Convenção de nomes

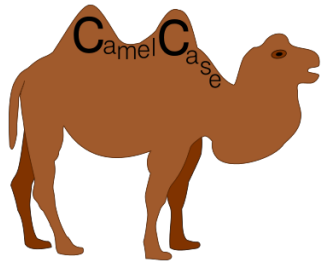
Boas práticas de programação

- A convenção de nomes tem por objetivo melhorar a legibilidade dos programas Java.
- É muito importante a adoção de nomes significativos e sem ambiguidades.
- A convenção foi elaborada para a nomeação de classes, métodos, variáveis e constantes.

Convenção de nomes

Classes

- O nome de classes e interfaces devem ser substantivos no formato *UpperCamelCase*, ou seja, TodasAsIniciaisEmMaiúsculo.



- Não abrevie as palavras e nem utilize acrônimos, a não ser nos casos onde os acrônimos são mais conhecidos do que a palavra completa (exemplo: PDF ou HTML). Obs: Ao utilizar acrônimos, somente a primeira letra deve ser maiúscula.
- Exemplos:

```
class PdfReader
class ConcurrentLinkedDeque
class ByteArrayOutputStream
```

Convenção de nomes

Métodos

- O nome de métodos devem começar por um verbo para indicar uma ação.
- O formato deve ser *lowerCamelCase*, ou seja, todasAsIniciaisEmMaiúsculoExcetoPrimeira.
- Exemplos:

```
getFirst()
removeLastOccurrence()
insertElementAt()
```

Convenção de nomes

Variáveis

- Variáveis de classe, de instância, locais e parâmetros também seguem o formato *lowerCamelCase*.
- De preferência, os nomes de variáveis devem ser mnemônicos curtos e significativos.
- Podem ser adotadas abreviaturas e siglas quando conveniente.
- Variáveis com nomes representados por um único caractere devem ser evitadas, exceto para variáveis temporárias. Nesse caso, os caracteres usuais são `i`, `j`, `k`, `l`, `m` e `n` para `int` e `c`, `d`, `e` para `char`

```
double myLenght, elapsedTime
int i
char c
```

Convenção de nomes

Constantes

- Constantes em Java são declaradas com as palavras-chave `static` e `final`
- Seus nomes devem ser escritos em maiúsculas e as palavras compostas devem ser separadas por underline.
- Os nomes de constantes podem receber dígitos numéricos quando apropriado, mas os dígitos não devem ser o primeiro caractere do nome.

```
static final int NUMBER_HOURS_IN_DAY = 24;
static final double AVOGADROS_NUMBER = 6.02214199E23;
```

Instruções de controle

Instruções de desvio condicional

- if-then, if-then-else e switch

Instruções de repetição (laço)

- while, do-while, for

Instruções de desvio incondicional

- break e continue

Instruções de controle

Instruções de desvio condicional: if-then

- O comando `if-then` executa uma seção do código se uma condicional for avaliada como `true`.
- Exemplo: A classe `Bicycle` poderia permitir a aplicação dos freios somente se a bicicleta já estiver em movimento.

```
void applyBrakes(int decrement) {
    // a cláusula "if": bicicleta deve estar em movimento
    if (speed > 0){
        // a cláusula "then": diminui a velocidade da bicicleta
        speed -= decrement;
    }
}
```

- As chaves de abertura `{` e de fechamento `}` do bloco `if-then` são opcionais, dado que a seção do código a ser executada no caso `true` contenha uma única instrução.

```
void applyBrakes() {
    // igual ao código anterior, mas sem as chaves
    if (speed > 0)
        speed -= decrement;
}
```

Instruções de controle

Instruções de desvio condicional: if-then-else

- O comando `if-then-else` permite um caminho alternativo caso a condicional seja avaliada como `false`.
- Exemplo: No caso do método `applyBrakes()`, pode-se realizar uma impressão na tela caso a bicicleta esteja parada, alertando o usuário.

```
void applyBrakes() {
    if (speed > 0) {
        speed -= decrement;
    } else {
        System.err.println("A bicicleta já se encontra parada!");
    }
}
```

- Nesse caso as chaves também são opcionais.

```
void applyBrakes() {
    if (speed > 0)
        speed -= decrement;
    else
        System.err.println("A bicicleta já se encontra parada!");
}
```

Instruções de controle

Instruções de desvio condicional: if-then-else

- Vários comandos `if-then-else` podem ser utilizados em sequência para provocar um efeito em cascata.
- Exemplo: Atribuir a nota de uma prova dependendo da porcentagem de acertos: nota A se acertou 90% ou mais, B se acertou 80% ou mais e assim sucessivamente.

```
// Determina a nota de uma prova dada a porcentagem de acerto
class IfElseExample {
    public static void main(String[] args) {

        int testscore = Integer.parseInt(args[0]); // converte o argumento em um inteiro
        char grade;

        if (testscore >= 90) {
            grade = 'A';
        } else if (testscore >= 80) {
            grade = 'B';
        } else if (testscore >= 70) {
            grade = 'C';
        } else if (testscore >= 60) {
            grade = 'D';
        } else {
            grade = 'F';
        }
        System.out.println("Nota = " + grade);
    }
}
```

Instruções de controle

Instruções de desvio condicional: switch

- Um único comando `switch` fornece um conjunto de caminhos possíveis de execução.
- O `switch` funciona com os tipos primitivos `byte`, `short`, `char` e `int` ; também funciona com tipos enumerados, classe `String`, e classes empacotadoras `Character`, `Byte`, `Short` e `Integer`.
- Exemplo 1: Obter o nome do mês, com base em seu valor numérico.
- Exemplo 2: Dado o ano e o mês, calcular o número de dias desse mês.

Instruções de controle

```
// Determina o mês dado seu valor numérico.
public class SwitchExample1 {
    public static void main(String[] args) {

        int month = Integer.parseInt(args[0]);
        String monthString;
        switch (month) {
            case 1:  monthString = "Janeiro";
                     break;
            case 2:  monthString = "Fevereiro";
                     break;
            case 3:  monthString = "Março";
                     break;
            case 4:  monthString = "Abril";
                     break;
            case 5:  monthString = "Maio";
                     break;
            case 6:  monthString = "Junho";
                     break;
            case 7:  monthString = "Julho";
                     break;
            case 8:  monthString = "Agosto";
                     break;
            case 9:  monthString = "Setembro";
                     break;
            case 10: monthString = "Outubro";
                     break;
            case 11: monthString = "Novembro";
                     break;
            case 12: monthString = "Dezembro";
                     break;
            default: monthString = "Mês Inválido";
                     break;
        }
        System.out.println(monthString);
    }
}
```

Instruções de controle

Instruções de desvio condicional: switch

```
// Calcula o número de dias de um determinado mês.
class SwitchExample2 {
    public static void main(String[] args) {

        int month = Integer.parseInt(args[0]);
        int year = Integer.parseInt(args[1]);
        int numDays = 0;

        switch (month) {
            case 1: case 3: case 5:
            case 7: case 8: case 10:
            case 12:
                numDays = 31;
                break;
            case 4: case 6:
            case 9: case 11:
                numDays = 30;
                break;
            case 2:
                if (((year % 4 == 0) &&
                     !(year % 100 == 0))
                     || (year % 400 == 0))
                    numDays = 29;
                else
                    numDays = 28;
                break;
            default:
                System.out.println("Mês Inválido.");
                break;
        }
        System.out.println("Número de Dias = " + numDays);
    }
}
```

Instruções de controle

Instruções de repetição: while

- O comando `while` executa repetidamente uma seção do código enquanto uma condição for avaliada como `true`.
- Exemplo: Imprimindo o fatorial de um número inteiro n, passado como parâmetro.

```
// Retorna o fatorial de um número.
class Factorial1 {
    public static void main(String[] args){
        int n = Integer.parseInt(args[0]);
        int fat = 1;
        while (n > 0) {
            fat *= n--;
        }
        System.out.println(fat);
    }
}
```

Instruções de controle

Instruções de repetição: do-while

- A diferença entre `while` e `do-while` consiste no momento onde a avaliação da cláusula condicional é feita.
- No `do-while` a condição fica ao final do bloco, portanto a seção do código correspondente ao `do-while` é executada pelo menos uma vez.
- Não esquecer do `;` ponto-e-vírgula ao final da condição.

```
// Retorna o fatorial de um número.
class Factorial2 {
    public static void main(String[] args){
        int n = Integer.parseInt(args[0]);
        int fat = 1;
        // Se n == 0 o código imprimirá um resultado incorreto.
        do {
            fat *= n--;
        } while (n > 0);
        System.out.println(fat);
    }
}
```

Instruções de controle

Instruções de repetição: for

- O comando `for` fornece uma maneira compacta de iterar sobre um conjunto delimitado de valores.
- Expressão de inicialização: executada uma única vez assim que o laço é inicializado.
- Condição de parada: Ao ser avaliada como `false`, encerra o laço.
- Expressão de incremento: executada a cada iteração do laço, normalmente utilizada para incrementar ou decrementar o contador.

```
// Retorna o fatorial de um número.
class Factorial3 {
    public static void main(String[] args){
        int n = Integer.parseInt(args[0]);
        int fat = 1;
        for (int i=1; i<=n; i++) {
            fat *= i;
        }
        System.out.println(fat);
    }
}
```

Instruções de controle

Instruções de repetição: for

- O comando `for` também foi projetado para iterar sobre `arrays` e `Collections`, tornando o código mais compacto e fácil de ler.
- Exemplo: Somando os números de um vetor desordenado.

```
// Soma todos os números de um vetor de inteiros.
class SumOfArray {
    public static void main(String[] args){
        int[] numbers = {1,3,5,7,9};
        int sum = 0;
        for (int item : numbers) {
            sum += item;
        }
        System.out.println("Sum is equal to " + sum);
    }
}
```

Instruções de controle

Instruções de desvio incondicional: break

- Existem duas formas de `break`: sem rótulo ou com rótulo.
- A versão sem rótulo é utilizada para terminar os laços `for`, `while` e `do-while`, além da instrução `switch`.
- Exemplo: Procurando um número em um vetor desordenado.
- A instrução `break` termina o laço assim que o valor é encontrado, transferindo o fluxo de execução para a instrução que segue o laço.

```
// Busca exaustiva por um número inteiro em um vetor.
class BreakWithoutLabel {
    public static void main(String[] args) {

        int searchFor = Integer.parseInt(args[0]);
        int[] arrayOfInts = {32,87,3,589,12,1076,2000,8,622,127};
        boolean foundIt = false;

        for (item : arrayOfInts) {
            if (item == searchfor) {
                foundIt = true;
                break;
            }
        }

        if (foundIt) {
            System.out.println("Achei " + searchfor + " no vetor.");
        } else {
            System.out.println(searchfor + " não se encontra no vetor.");
        }
    }
}
```

Instruções de controle

Instruções de desvio incondicional: break

- Um `break` sem rótulo termina a instrução `switch`, `for`, `while` ou `do-while` mais próxima, ou seja, a instrução mais interna que contém o `break`.
- Um `break` com rótulo pode terminar qualquer uma dessas instruções que o contém, mesmo a mais externa.
- Para isso, basta atribuir um rótulo à instrução que se deseja terminar e chamar o `break` com esse rótulo.
- Exemplo: Procurando um número em uma matriz.

Instruções de controle

Instruções de desvio incondicional: break

- A instrução `break search` termina o laço `for` mais externo, que foi rotulado como `search`.

```
// Busca exaustiva por um número inteiro em uma matriz.
class BreakWithLabel {
    public static void main(String[] args) {

        int searchFor = Integer.parseInt(args[0]);
        int[][] arrayOfInts = {
            { 32, 87, 3, 589 },
            { 12, 1076, 2000, 8 },
            { 622, 127, 77, 955 }
        };

        boolean foundIt = false;

    search:
        for (int i = 0; i < arrayOfInts.length; i++) {
            for (int j = 0; j < arrayOfInts[i].length; j++) {
                if (arrayOfInts[i][j] == searchfor) {
                    foundIt = true;
                    break search;
                }
            }
        }

        if (foundIt) {
            System.out.println("Achei " + searchfor + " na matriz.");
        } else {
            System.out.println(searchfor + " não se encontra na matriz.");
        }
    }
}
```

Instruções de controle

Instruções de desvio incondicional: continue

- Uma instrução `continue` também possui duas versões: com rótulo e sem rótulo.
- Na versão sem rótulo, essa instrução pula a iteração corrente do laço `for`, `while` ou `do-while` mais próximo, ou seja, o laço mais interno que contém o `continue`.
- Após pular uma interação, a expressão booleana que controla o laço passa por uma nova avaliação.
- Exemplo: Contagem do número de repetições de um caractere em uma cadeia de caracteres.

Instruções de controle

Instruções de desvio incondicional: continue

- No código abaixo, a `String` é varrida caractere a caractere em um laço `for` .
- Se um caractere da `String` não é `'r'`, a instrução `continue` pula o resto da iteração e procede para o próximo caractere, caso contrário, o contador é incrementado.

```
// Contagem do número de aparições de um caractere em uma cadeia de caracteres.
class ContinueWithoutLabel {
    public static void main(String[] args) {

        String searchMe = "A aranha arranha a rã. A rã arranha a aranha. "
                        + "Nem a aranha arranha a rã. Nem a rã arranha a aranha.";
        int max = searchMe.length();
        int numRs = 0;

        for (int i = 0; i < max; i++) {
            // interessado somente na letra r
            if (searchMe.charAt(i) != 'r')
                continue;

            // incrementa o contador
            numRs++;
        }
        System.out.println("Achei " + numRs + " aparições da letra \'r\' na String.");
    }
}
```

Instruções de controle

Instruções de desvio incondicional: continue

- Uma instrução `continue` com rótulo pula a iteração corrente de um laço marcado com um rótulo.
- Exemplo: Encontrar uma subcadeia de caracteres (*substring*) dentro de uma cadeia maior de caracteres.
- O algoritmo adota dois laços para realizar a busca. Um dos laços itera sobre a *substring* e o outro laço itera sobre a *string*.

Instruções de controle

Instruções de desvio incondicional: continue

- Uma instrução `continue outerLoop` pula uma iteração do laço externo, que itera os caracteres da string maior.

```
// Busca de uma substring em uma string.
class ContinueWithLabel {
    public static void main(String[] args) {

        String searchMe = "O tempo perguntou pro tempo quanto tempo o tempo tem. "
                        + "O tempo respondeu pro tempo que o tempo "
                        + "tem tanto tempo quanto tempo o tempo tem.";
        String substring = "tempo";
        int numSubs = 0;

        int max = searchMe.length() - substring.length();

    outerLoop:
        for (int i = 0; i <= max; i++) {
            int n = substring.length();
            int j = i;
            int k = 0;
            while (n-- != 0) {
                if (searchMe.charAt(j++) != substring.charAt(k++)) {
                    continue outerLoop;
                }
            }
            numSubs++;
        }
        System.out.println("A substring \"" + substring + "\" aparece " + numSubs + "
       vezes na string.");
    }
}
```

E/S de dados

Entrada de dados

- A Java API `Scanner` fornece um mecanismo para a entrada de dados do usuário pelo terminal.
- Para utilizar essa API, o programa deve importar a classe correspondente no código-fonte.

```
import java.util.Scanner;
```

- Antes de iniciar a leitura de dados, é necessário instanciar um objeto do tipo `Scanner` utilizando a palavra-chave `new`, passando como argumento o objeto de entrada padrão `System.in` .

```
Scanner input = new Scanner( System.in );
```

E/S de dados

Entrada de dados

- A variável `input` passa a referenciar o objeto instanciado.
- O método `nextInt()` da classe `Scanner` (consulte documentação [1]) faz com que o programe pause para aguardar que o usuário digite um número inteiro e pressione *Enter*.

```
int number = input.nextInt();
```

- Se não for digitado um número inteiro, ocorre um erro em tempo de execução.
- Esse erro pode ser evitado utilizando o método `hasNextInt()`, que retorna `true` se e somente se o usuário entrou com um número inteiro.

[1] http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html#nextInt()

E/S de dados

Saída de dados

- A impressão de dados na tela pode ser feitas pelos seguintes métodos:
  - `System.out.print()`
  - `System.out.println()`
  - `System.out.printf()`
- Esses métodos aceitam os seguintes caracteres especiais:
  - `\n` – nova linha
  - `\t` – tabulação horizontal
  - `\\` – imprime o caractere de barra invertida `\`
  - `\"` – imprime o caractere aspas duplas `"`
  - `\'` – imprime o caractere aspa simples `'`
  - `\b` – backspace

E/S dados

Exemplo de um programa utilizando E/S

```
// Addition.java
// Programa que imprime a soma de dois números.
import java.util.Scanner; // programa utiliza a classe Scanner

public class Addition
{
   // método main inicia a execução do aplicativo Java
   public static void main( String args[] )
   {
      // cria um objeto Scanner para obter a entrada do terminal
      Scanner input = new Scanner( System.in );

      int number1; // primeiro numero a somar
      int number2; // segundo numero a somar
      int sum; // soma dos dois números

      System.out.print("Entre com o primeiro número inteiro: "); // terminal
      number1 = input.nextInt(); // lê o primeiro número do usuário

      System.out.print("Entre com o segundo número inteiro: "); // terminal
      number2 = input.nextInt(); // lê o segundo número do usuário

      sum = number1 + number2; // soma os números

      System.out.printf("A soma resulta em %d\n", sum); // imprime soma

   } // fim do método main

} // fim da classe Addition
```

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.