Disciplina: LPOO

Prof. Anderson V. de Araujo

# Aula 03 – Controle de Fluxo, Laços e Métodos

andvicoso@facom.ufms.br
http://prof.facom.ufms.br/~andvicoso/

# Controle de Fluxo

- São comandos que possuem características especiais que alteram a execução do código
- Java provê um comando de controle de fluxo bastante poderoso
- **Seleção**: if-else, switch-case
- **Repetição**: for, while, do-while
- **Desvios** (somente em estruturas de repetição): continue, break, rótulos
- Não existe comando goto, porém, é uma palavra-reservada do Java

# if / if else / if else if

- *Sintaxe*

```
if (expressão booleana)
   instrução_simples;

if (expressão booleana) {
   instruções
}
```

```
if (expressão booleana) {
   instruções
} else if (expressão booleana) {
   instruções
} else {
   instruções
}
```

- *Exemplo*

```
if ( ano < 0) {
   System.out.println("Não é um ano!");
} else if ( (ano%4==0 && ano%100!=0) || (ano%400==0) ) {
   System.out.println("É bissexto!");
} else {
   System.out.println("Não é bissexto!");
}
```

# Switch

- Há situações em que se **sabe de antemão que as condições** assumem o valor true de forma **mutuamente exclusiva**
- **Apenas uma entre as condições** sendo testadas assume o valor **true** num dado momento
- Equivale a um conjunto de *if 's* encadeados, porém mais estruturado
- O valor resultante de expressão do switch (valor) é comparado com as constantes presentes nos comandos **case (valor)** contida dentro da estrutura switch
- A partir do java 7 é possível usar **strings** no switch

# Switch - Sintaxe

```
switch (variavel) {
        case valor_1:
        //Uma ou mais instruções;
        break;

        case valor_2:
        //Uma ou mais instruções;
        break;

        case valor_3:
        //Uma ou mais instruções;
        break;

        //Um ou mais cases...

        default:
        //Uma ou mais instruções;
        break;
}
```

**A variável sendo avaliada pode ser inteira, caractere e Strings.**

**Os valores são sempre constantes, não podem ser expressões.**

**O break é opcional. Geralmente está associado com cada case, mas nem sempre...**

**O default é opcional. Geralmente está no final, mas nem sempre...**

**Como é o último do switch também se torna opcional.**

# Switch - Exemplo

```
char letra = //ler um caractere da entrada do usuário
switch (letra) {
        case 'A':
        System.out.println("A letra lida é um 'A'");
        break;

        case 'B':
        System.out.println("A letra lida é um 'B'");
        break;

        case 'C':
        System.out.println("A letra lida é um 'C'");
        break;

        default:
        System.out.println("A letra lida é desconhecida");
}
```

# O comando break

- O comando **break interrompe a execução do bloco** de case. Continua com a próxima instrução, logo, após o bloco
  - O comando break é **opcional**
- Caso não tenha o comando break ao final de um comando **case**, o que pode acontecer?
  - **A execução irá continuar** para os cases seguintes até encontrar um break ou o final do bloco do switch

# O comando default

- Como switch pode receber várias possibilidades, pode ocorrer de algum caso estar fora do alcance ou não definido
- Caso nenhuma condição seja satisfeita, o código dentro do bloco **default será executado obrigatoriamente**
  - É o **bloco** de código **padrão** que deve ser executado quando nenhum case for satisfeito
- O comando default é **opcional** na estrutura switch
- Caso uma condição case seja satisfeita e tenha um comando break para interrompê-lo, o bloco default será executado?
  - Não, pois ele só é executado se nenhum caso for satisfeito
- Pode estar em qualquer posição dentro do switch, mas **geralmente, é colocad0 no final**

# Switch - Exemplo

```
String typeOfDay;
String dayOfWeek = s.nextLine();
switch (dayOfWeek) {
        case "Monday":
        typeOfDay = "Start of work week";
        break;
        case "Tuesday":
        case "Wednesday":
        case "Thursday":
                typeOfDay = "Midweek";
        break;
        case "Friday":
                typeOfDay = "End of work week";
        break;
        case "Saturday":
        case "Sunday":
                typeOfDay = "Weekend";
                break;
        default:
                System.out.println("Invalid day of the week: " + dayOfWeek);
}
System.out.println(typeOfDay);
```

# Qual é melhor *if* ou *switch*?

- Depende do problema
- Tente fazer com que o código fique o mais legível possível
- Na questão da velocidade, em geral, existe uma diferença muito pequena
  - Onde o switch é mais rápido
- Os compiladores mais avançados já trabalham de forma a otimizar a execução de ambos

# Lendo Valores do Console de Entrada do Usuário

- Para ler do console do usuário use o objeto **Scanner**
- Para usá-lo é **necessário importá-lo** através da linha de código no começo do arquivo .java:

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

- Exemplos:

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

System.out.print("Entre com o valor da variavel 1: ");
double var1 = sc.nextDouble(); //ou int var1 = sc.nextInt();

sc.close(); // quando terminar a leitura
```

```
char c = sc.next().charAt(0); //para ler um único caractere
```

# Exercícios Práticos

1. Leia um número inteiro do console e diga se ele é impar ou par
    1. Faça usando o comando if/else
    2. Faça usando apenas o operador condicional
2. Escreva um programa chamado Switch que realize as 4 operações (+), (-), (/) e (*) entre duas variáveis, através da escolha do usuário. **Use o comando switch**.
3. Verificar se um aluno foi aprovado na disciplina, lendo as notas de duas provas e verificando a sua média. **Use o comando if/else**.

# Laços
# (ou Estruturas de Repetição)

Repetir, repetir, repetir...

# Laços

- Estruturas que fazem com que um bloco de **instrução seja executado repetidamente**, enquanto uma condição (expressão) for verdadeira
- São definidos dois tipos de estruturas:
  - Estruturas de repetição (simples):
    - Em geral, repete um número específico de vezes, mas também está associada a uma condição
    - Exemplo: **for**
  - Estruturas de repetição condicional:
    - São estruturas de repetição que o controle é feito pela **avaliação de expressões condicionais**
    - Ou seja, o número de repetições é indeterminado na fase de programação e será conhecido durante a execução, tais como as estruturas **while** e **do–while**

# Laço for

- **for (inicialização; cond. de execução; increm./decrem.)**
  - **Inicialização:** É usada para dar valor inicial a variável de controle (contador)
    - As variáveis declaradas na inicialização têm visibilidade somente dentro do laço for
  - **Condição de execução:** É uma expressão booleana que determina a execução associada ao for, geralmente utilizando a variável de controle
  - **Incremento/Decremento:** Determina como a variável de controle será alterada a cada iteração do laço

# Laço for - Sintaxe

- Sintaxe

```
for(instrução/inicialização; expressão booleana; instrução/passo de repetição){
        //instruções
}
```

- **Todas as três** partes do laço são **opcionais**
- Exemplos:

```
for(int i=0; i<10; i++){
        System.out.println(i);
}
```

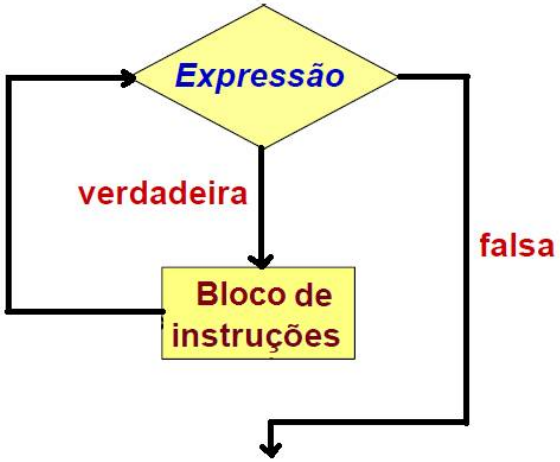
```
int cont;
for(cont=10; cont>0; cont-=2){
   System.out.println(cont);
}
```

# Laço for - Possibilidades

- Válido? `for (;;);` (laço infinito)
  - Igualmente ao laço `while (true) {};`
- Válido? `for (i=1, j=2, k=2; k<3; k++) { }`
  - não tem limite, desde que não seja declarada nenhuma variável fora da inicialização
- Válido? `for (int i=1, k=2; k<3; k++) { }`
  - é permitido ter várias variáveis na inicialização desde que sejam do mesmo tipo
- Válido? `for (int cont = 0; cont++ < 10; sysout("asdf"));`
  - é permitido ter só uma instrução após o último ';'
- Válido? `for (int i = 'A'; i <= 'Z'; i++) sysout(i);`

# Laço while

- O corpo do laço pode ser uma simples instrução, um bloco de instruções ou nenhuma
- Na estrutura **while**, a expressão sob condição é avaliada antes da execução do laço.
  - Logo, o laço poderá **nunca** ser executado



# Laço while - Sintaxe

- Sintaxe:

```
while(expressão booleana) {
        //uma ou + instruções
}
```

- Exemplos:

```
int i=0;
while(i<10){
   System.out.println(i);
   i++;
}
```

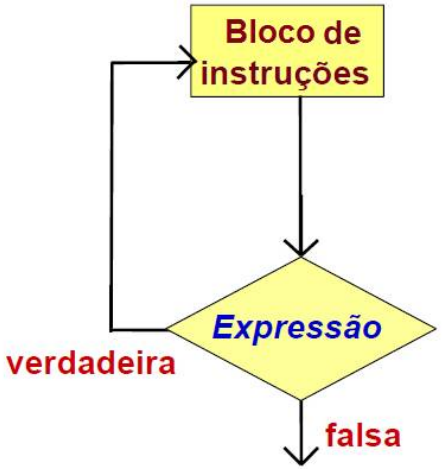
```
boolean b = true;
int x = 40;
while(b && 10 < x)
   System.out.println("De novo!");
```

# Laço while - Possibilidades

- Não é possível declarar variáveis dentro do parênteses do laço while (no for pode)
  - Deve ser uma expressão booleana
- Válido? `while (true) {};` (laço infinito)
  - Igualmente ao laço `for (;;);`

# Laço do-while

- O corpo do laço pode ser uma simples instrução, um bloco de instruções ou nenhuma
- Na estrutura **do-while**, a expressão sob condição é avaliada ao final da execução do laço
  - Logo, o laço será executado **pelo menos uma vez**



# Sintaxe do-while

- Sintaxe:

```
do{
   //uma ou + instruções
}while(expressão booleana);
```

- Exemplo:

```
class ExemploDoWhile{
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      String aux;
      int contador = 0;
      do {
         System.out.println("Digite uma letra e ENTER em seguida: ");
         aux = scanner.nextLine();
         contador++;
      } while (aux.charAt(0) != 'A'); //critério de parada
      System.out.println("Número de caracteres != A lidos:" + (contador-1));
    }
}
```

# Laço do-while - Possibilidades

- Não é possível declarar variáveis dentro do parênteses do laço do-while (no for pode)
  - Deve ser uma expressão booleana
- Válido? `do{ } while (true);` (laço infinito)
  - Igualmente ao laço `for (;;);` e `while (true) {};`

# while/do-while - Exemplo

```
int x = 0;
while (x < 10) {
    System.out.println ("item " + x);
    x++;
}
```

```
int x = 0;
do {
    System.out.println ("item " + x);
    x++;
} while (x < 10);
```

# Outros Comandos

- **break**
  - Interrompe a execução do bloco de repetição e continua com a próxima instrução logo após o bloco
  - É comumente utilizado para produzir a **parada de um laço** mediante a ocorrência de alguma condição específica, antes da chegada do final natural do laço
- **continue**
  - Interrompe a execução da iteração corrente do laço mais próximo
  - Testa a condição e **reinicia o bloco com a próxima iteração**

# break x continue - Exemplo

```
class Teste2 {
   public static void main(String args[]) {
      int soma = 0;
      for (int i = 1; i <= 5; i++) {
         if (i == 4)
            continue;
         if (i == 5)
            break;
         soma += i;
      }
      System.out.println(soma);
   }
}
```

# Rótulos

- E quando temos mais de um laço executando e queremos parar o laço externo?
- Podemos **rotular** os laços e **parar qual quisermos**!
- A execução continua a partir da próxima linha de comando após o laço quebrado
- É possível ter rótulos para os comandos continue e break

```
first: for(int i = 0; i < 10; i++) {
   second: for(int j = 0; j < 5; j ++) {
      break X;
   }
}

third: for(int a = 0; a < 10; a++) { }
```

- *X* pode ser tanto *first* quanto *second*.
  - Trocar por *third* não compila

# Rótulos - Exemplo

```
int sum = 0;
for (int i = 0; i < 3; i++) {
  labelj: for (int j = 0; j < 3; j++) {
                for (int k = 0; k < 10; k++) {
                    sum += k;
                    if (k > 5) {
                        break labelj;
                     }
                }
            }
   }
   System.out.println("sum " + sum);
```

**63**? E se **labelj** fosse adicionado antes do primeiro laço?

# Exercícios Práticos

1. Escrever um programa que leia repetidamente uma nota de um aluno fornecida pelo usuário até que o usuário digite -1 para finalizar. Calcule e imprima média de todas as notas no final.
2. Leia um inteiro do console e imprima o número binário correspondente
3. Leia um inteiro do console e imprima uma matriz nxn de 1 até n*n

# Métodos

- São blocos de código separados que pertencem a um programa* e tem por **finalidade realizar uma tarefa específica**
- Métodos geralmente correspondem à uma ação
  - somar, correr, imprimir, lerArquivo, ...
- **Evitar reescrever código** para uma mesma função toda vez que se deseja executá-la
- Podem ser declaradas em **qualquer posição** no código
  - Diferentemente de outras linguagens que obrigam que a declaração ocorra antes da sua chamada/utilização
- Sinônimos:
  - Procedimento, Função, Sub-rotina, Subprograma, ...

# Declaração

- Temos que definir suas características, tipo de retorno, nome e parâmetros
- Sintaxe geral:

```
[características] tipo_retorno nome(parâmetros) {
    // código do método
}
```

  - Exemplo:

Assinatura do método

```
public static void main(String[] args) {
}
```

- Por enquanto, vamos criamos os métodos como sendo **public static**, mas depois vamos ver o porquê...

# Retorno

- Métodos podem retornar um valor ao código chamador
- O tipo do valor de retorno é definido antes do nome do método
  - A palavra-chave **void** indica a ausência de retorno. Serve para definir um método que não retorna dado algum
- O retorno dentro do método é realizado através da palavra reservada **return**
  - O **return** serve para sair do método em que está executando e devolver o valor de retorno ao método chamador
  - Com isso, é possível ter **return** em métodos com tipo **void.** Esse retorno serve para **parar de executar o método** corrente

```
public static boolean ehPar(int a) {
    return a % 2 == 0;
}
public static String qualAltura(double altura) {
    return "Minha altura é: " + altura;
}
```

# Parâmetros

- É a sequencia de definições de variáveis separadas por vírgulas entre os parênteses na assinatura do método
- O código interno ao método pode usar as variáveis com valores passados no código chamador
- Um método pode ter nenhum, um ou mais de um parâmetro
- Exemplos:

```
public static void dataNascimento(int ano, int dia) {
        System.out.println("Eu nasci no ano de: ");
        System.out.println(ano);

        System.out.println("No dia: ");
        System.out.println(dia);
}
public static void imprimirNome() {
        System.out.println("Anderson");
   }
```

# Exemplo

```
public class OperacoesMatematicas{

    public static int somar(int num1, int num2){
        return num1 + num2;
    }

    public static int subtrair(int num1, int num2){
        return num1 - num2;
    }

    public static int multiplicar(int num1, int num2){
        return num1 * num2;
    }

    public static int dividir(int num1, int num2){
        return num1 / num2;
    }
```

# Exemplo (2)

```
        public static void main(String[] args) {
                int s1 = somar(3, 4);
                int s2 = somar(3, 5);

                int d1 = dividir(12, 3);

                int m1 = multiplicar(8, 9);

                int sub1 = subtrair(3, 1);
                int sub2 = subtrair(2, 2);
        }
}
```

O método **chamador** é o método **main.**
Os métodos chamados são somar, dividir, multiplicar e subtrair.

# Exercícios Práticos

1. Crie um método que não receba nenhum parâmetro e escreva o nome da cidade em que você nasceu no console
2. Crie um método que receba **dois parâmetros** de números de ponto flutuante (a e b) correspondendo a valores dos catetos de um triângulo retângulo e **retorne** o valor da hipotenusa
   - Use a função `Math.sqrt(x)`