



Orientação a Objetos

Aula 3 - Instruções de repetição

Daniel Porto

daniel.porto@unb.br

APRESENTAÇÃO

1. Instruções de controle - Repetição
2. Introdução de String

INSTRUÇÕES DE REPETIÇÃO

Existem 3 instruções de repetição em Java que são usadas de acordo com a correta necessidade lógica a ser aplicada em determinadas situações, em que as características de cada uma possam ser melhor aproveitadas.

```
for(<inicialização>; <condição>; <incremento>)  
    <instrução>;
```

<inicialização> é executada uma única vez, no momento em que o laço será iniciado, podendo conter declarações.

A sequência de instruções será repetida enquanto a <condição> for **verdadeira**. Quando ela for **falsa** os comandos após o laço serão executados, interrompendo a sua repetição e prosseguindo com execução do programa.

<incremento> define como será alterada a variável de controle da repetição, cada vez que ele for repetido, sendo executado logo após o fim do bloco de repetição.

INSTRUÇÕES DE REPETIÇÃO

Observe os exemplos:

```
1  for (int aux = 0; aux < 10; aux++)  
2      System.out.println("Valor = " + aux);
```

Similar as instruções anteriores, a necessidade de mais que uma instrução a ser executada durante a repetição for exige a criação do bloco de instruções com chaves.

```
1  for (int aux = 0, int cont = 1; aux <= 10; aux+=2, cont++) {  
2      System.out.print("\t" + cont + " Par = ");  
3      System.out.println("\t" + aux);  
4  }
```

A inicialização, condição e incremento podem ser nulos, possibilitando inclusive a elaboração de um laço infinito.

INSTRUÇÕES DE REPETIÇÃO

Similar as instruções condicionais, também é possível o aninhamento entre as instruções de repetição, por exemplo:

```
1  for (int aux = 0; aux < 10; aux++)  
2      for (cont = 5; cont > 1; cont--) {  
3          total = aux * cont;  
4          System.out.println("Total = " + total);  
5      }
```

O **for** com controle sobre cont não cria variável que deve ter sido declarada antes do laço, assim como total.

No **for** mais interno (controle sobre cont) é realizada a operação de decremento ao invés do incremento.

Existe um bloco de instrução somente no **for** mais interno, pois ele possui duas instruções, enquanto o primeiro **for** possui só uma (outro **for**) em seu corpo.

INSTRUÇÕES DE REPETIÇÃO

Outra instrução de repetição usa os mesmos elementos do laço **for**, mas distribuídos de forma diferente.

```
while (<condição>)  
    <instrução>;
```

No **while** a sequência de instruções é repetida enquanto a <condição> for verdadeira.

O laço se repete até que a condição se torne **falsa**, encerrando o laço e continuando a execução do programa.

O laço **while** é mais apropriado para situações que a repetição possa ser **encerrada inesperadamente**, enquanto o **for** é mais indicado para quantidades de repetições conhecidas (ou definidas).

Similar ao **for**, a instrução **while** também pode ser aninhada, ou seja, possuir um **while** dentro de outro, além de possuir um bloco de instruções com uso das chaves.

INSTRUÇÕES DE REPETIÇÃO

Observe o trecho de um programa com a repetição while como exemplo:

```
1 public static void main(String[] args) {
2     // Declarações
3     int qtde, aux;
4     Scanner ler = new Scanner(System.in);
5     DecimalFormat casas;
6     casas = new DecimalFormat("00");
7
8     // Instruções
9     System.out.print("Informe a quantidade:\n");
10    qtde = ler.nextInt();
11    aux = 0;
12    while (aux++ < qtde)
13        System.out.println("Valor= " + casas.format(aux));
14 } // encerra o método main()
```

INSTRUÇÕES DE REPETIÇÃO

A última estrutura de repetição cria um ciclo repetitivo até sua condição ser **falsa**. A condição neste laço é avaliada depois da repetição ser executada, ou seja, seu bloco será executado ao menos uma vez para ser verificado (testado).

```
do {  
    <instruções>;  
} while (<condição>;
```

As chaves não são sempre necessárias, somente quando existem mais que uma instrução em seu bloco, mas elas favorecem a legibilidade do programa.

Similar as repetições anteriores (**for** e **while**), esta instrução também aceita o aninhamento de outras repetições, inclusive dela mesmo (uma dentro da outra).

Esta instrução também é mais adequada logicamente a quantidade de repetições não definidas ou desconhecidas.

INSTRUÇÕES DE REPETIÇÃO

Observe exemplo com do...while :

```
1  /** Síntese
2   *   Objetivo: analisar a paridade de um número
3   *   Entrada: número inteiro positivo
4   *   Saída:   paridade do número informado
5   */
6  import java.util.Scanner;
7  public class Paridade {
8      public static void main(String[] args) {
9          // Declarações
10         int valor;
11         Scanner ler = new Scanner(System.in);
12         // Instruções
13         do {
14             System.out.print("Número inteiro positivo: ");
15             valor = ler.nextInt();
16             if (valor < 0)
17                 System.out.println("Valor inválido.\n");
18         } while (valor < 0);
19         System.out.print(((valor % 2) == 0) ? "\tPAR": "\tIMPAR");
20     }
21 }
```

INSTRUÇÕES DE REPETIÇÃO

Instrução break

No corpo de uma instrução de repetição o break encerra (quebra) o bloco de repetição que esta sendo executado imediatamente, passando a executar as instruções do programa que estão fora da repetição (continua o programa sem repetir mais).

`break;`

Em instruções aninhadas o break só afeta o laço que o contém, sendo somente este interrompido.

INSTRUÇÕES DE REPETIÇÃO

Instrução continue

Este comando permite que se retorne ao início do laço imediatamente e seja verificada a condição de execução da mesma novamente.

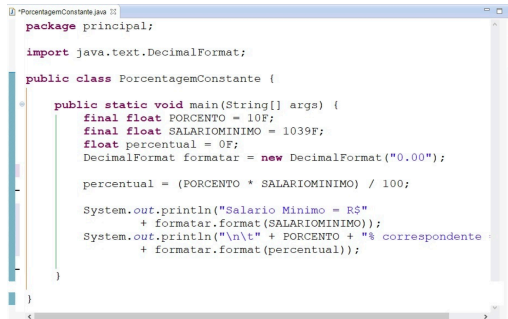
Em alguns casos que não exista a necessidade de continuar até o final do bloco de instruções do laço, pode-se usar a instrução continue.

```
continue;
```

Analise o exemplo a seguir que mostra o funcionamento das instruções break e continue.

INSTRUÇÕES DE REPETIÇÃO

As "**boas práticas**" de programação exigem um código organizado e respeitando as normas para elaboração de um código que seja fácil de entender para qualquer programador.



```
package principal;

import java.text.DecimalFormat;

public class PorcentagemConstante {

    public static void main(String[] args) {
        final float PORCENTO = 10F;
        final float SALARIOMINIMO = 1039F;
        float percentual = 0F;
        DecimalFormat formatar = new DecimalFormat("0.00");

        percentual = (PORCENTO * SALARIOMINIMO) / 100;

        System.out.println("Salario Minimo = R$"
            + formatar.format(SALARIOMINIMO));
        System.out.println("\n\t" + PORCENTO + "% correspondente
            + formatar.format(percentual));
    }
}
```

Algumas teclas de atalho agilizam a organização do código nesse IDE:

CRTL+SHIF + F

arruma endentação

CRTL +SHIF + C

comenta com // todas as linhas selecionadas

CRTL + SHIF + /

comenta como bloco as linhas selecionadas

INSTRUÇÕES DE REPETIÇÃO

Observe exemplo:

```
1  /** Síntese
2   *   Objetivo: obter um número impar
3   *   Entrada:  número inteiro impar
4   *   Saída:    número inteiro impar
5   */
6  import java.util.Scanner;
7  public class Continua {
8      public static void main(String[] args) {
9          // Declarações
10         int numero;
11         Scanner ler = new Scanner(System.in);
12         // Instruções
13         do {
14             System.out.println("Digite valor inteiro impar: ");
15             numero = ler.nextInt();
16             if ((numero % 2) == 0)
17                 continue;
18             else
19                 break;
20             System.out.print("Valor inválido!");
21         } while ((numero % 2) == 0);
22         System.out.print("Valor Impar = " + numero);
23     }
24 }
```

INSTRUÇÕES DE REPETIÇÃO

Instrução Rotulada (label)

Observe o trecho do programa Java abaixo e verifique a utilização do recurso de especificação de um rótulo (label) no processamento referenciado pela instrução continue. Este tipo de rótulo/referência também pode ser feito no break.

```
1 public class Label {
2     public static void main (String args[]) {
3         loop1 : for (int i=1; i < 10 ; i++)
4             for (int j= i+1; j < 6; j++)
5                 if (i == 2) {
6                     System.out.println("Continue i=" + i + "\tj=" + j);
7                     continue loop1; // referência ao label
8                 }
9                 else
10                    System.out.println("i="+i+"\tj="+j);
11     }
12 }
```

COMPARTILHANDO PROJETOS

As atividades de desenvolvimento e entrega (compartilhamento) dos projetos Java, solicitados por este curso, não deverão ser somente do arquivo (programa) fonte, mas de todos os aspectos definidos em seu desenvolvimento. Para isso o projeto elaborado será entregue de maneira compactada (formato **zip**), respeitando as seguintes diretrizes para entrega correta:

1. Elaborar um projeto Java que solucione o problema (exercício) proposto;
2. Fechar o projeto pelo IDE (Eclipse) usado e localizar seu diretório (pasta) específico de armazenamento;
3. Compactar no formato **zip** este diretório (pasta);
4. Renomear o arquivo compactado para o padrão exigido para entrega;
5. Entregar o arquivo compactado contendo todo o projeto elaborado.

ABRINDO PROJETOS COMPARTILHADOS

A utilização (**execução**) dos projetos Java compactados deve seguir as orientações abaixo para ser executado:

1. Copiar o arquivo compactado do projeto Java no diretório de trabalho (workspace) de seu IDE;
2. Descompactar este arquivo exatamente neste diretório de trabalho (workspace);
3. No menu de barra do IDE escolher a opção de Importar;
4. Escolher o tipo de arquivo da opção **General** denominada **Existing Projects into Workspace** e pressionar botão **Next**;
5. Indicar o caminho do diretório que foi descompactado para o IDE e deixar selecionado o projeto Java que será importado, pressionando o botão **Finish**.

Confira que o projeto foi importado e esta aberto em seu IDE para ser executado.

A partir destes esclarecimentos todo programa entregue neste curso sempre deverá acontecer por seu projeto compactado.

EXERCÍCIOS DE FIXAÇÃO

- 1) Faça um programa que analise o conjunto de dados contendo peso e sexo (masculino, feminino) de 20 pessoas (quantidade definida em constante no programa). Este programa deverá mostrar o valor do maior e menor peso informado, a média dos pesos dos homens e o número de mulheres que participaram desta análise.
- 2) Solicite ao usuário a quantidade de números inteiros que ele gostaria de informar para o cálculo do fatorial e após validar este número solicite cada um deles e mostre o seu valor fatorial.

STRING

Corresponde a um conjunto de caracteres (estrutura de dados homogênea do tipo **char**), sendo em Java uma classe disponível em sua biblioteca padrão (java.lang).

Cada instância desta classe guardará valores entre aspas, por exemplo:

String saudar = "Bom dia!";

classe Java instância String (identificador) conteúdo desta instância

A criação de um novo objeto String também pode ser realizada por meio do acionamento de seu **método construtor**, por exemplo:

```
String nome = new String("Maria"); // construtor
```

STRING

Vários são os métodos disponíveis nesta classe (String), sendo alguns deles abordados a seguir:

MÉTODO	FUNCIONALIDADE
charAt(int)	retorna o caracter da posição
indexOf(char)	retorna a posição do caracter
concat(String)	concatena String (como operador +)
equals(String)	compara se 2 Strings são idênticas
equalsIgnoreCase(String)	compara 2 Strings sem considerar Mai./min.
length()	tamanho da String instanciada
lastIndexOf(char)	retorna a última posição do caracter
replace(char, char)	troca os caracteres na String toda
substring(int inicial, int fim)	retorna um pedaço (subcadeia) da String
toLowerCase()	transforma toda String em minúsculo
toUpperCase()	transforma toda String em maiúsculo
startsWith(String)	retorna true quando String estiver no início
endsWith(String)	retorna true quando String estiver no final
trim()	retira espaços do início e final da String
valueOf(x)	converte vários tipos de dados (x) em String

INSTRUÇÕES DE REPETIÇÃO

Processo de Concatenação

O sinal `+` é usado para concatenar valores em uma String, podendo esta concatenação envolver valores de diferentes tipos, como é realizado no uso da classe `System`, por exemplo:

```
System.out.print("Salário = " + sal + "Ano = " + ano);
```

Sendo a expressão `Salário` concatenada com a variável `sal` (salário) do tipo `double` e a expressão `Ano` com a variável inteira `ano`.

Este tipo de concatenação com o sinal `+` converte os outros valores em String para serem mostrados pelo computador.

INSTRUÇÕES DE REPETIÇÃO

Comparação entre Strings

Dois métodos efetuam este tipo de comparação, sendo o `compareTo` similar a função `strcmp()` em C

```
1 String nome1 = new String("Paulo");
2 String nome2 = new String("Paula");
3 if(nome1.equals(nome2))
4     //Retorna true para Strings idênticas ou false para Strings diferentes
5
6
7     if(nome1.compareTo(nome2))
8         //Retorna zero (0) para Strings idênticas ou
9         //qualquer outro valor inteiro para diferentes
10
11
12     if(nome1 == nome2)
13         //Compara a referência ou endereço de memória dos objetos
```

EXERCÍCIOS DE FIXAÇÃO

3) Elabore um programa que cadastre uma senha de até 5 caracteres, não podendo ser menor que 3 caracteres e nem conter espaço em branco. Seu programa deverá permitir até 9 tentativas de acerto, definidas em uma constante denominada MAXIMO, por outros usuários diferentes daquele que cadastrou a senha. Caso ninguém consiga acertar até este limite, seu programa deve informar que o computador irá se auto-destruir em 10 segundos e encerrar o programa.

Use os recursos que achar adequados para solução deste problema usando String, onde os caracteres maiúsculos são diferentes dos minúsculos na averiguação da senha. Caso o usuário acerte a senha o programa deverá parabenizá-lo com uma mensagem que será apresentada depois de 22 linhas abaixo da senha informada e no centro da linha será escrita esta mensagem.

EXERCÍCIOS DE FIXAÇÃO

- 4) Faça um programa que armazene uma frase e mostre a quantidade de cada vogal contida na mesma, além do total geral de vogais. Sua solução deverá considerar as vogais independente das mesmas estarem em maiúsculo ou minúsculo. Encerre sua solução apresentando a quantidade de cada vogal, o total das vogais e o tamanho da frase digitada pelo usuário. Repita este processo de análise de uma frase enquanto o usuário desejar. Frase sem dados é inválida e deve ser solicitada novamente.
- 5) No campeonato de basquete regional existem 5 equipes e cada uma possui 10 jogadores. Faça um programa que mostre a quantidade de jogadores com idade maior que 15 anos, a média destas idades por equipe e a porcentagem de jogadores com altura menor que 1,50 metros entre todas as equipes.