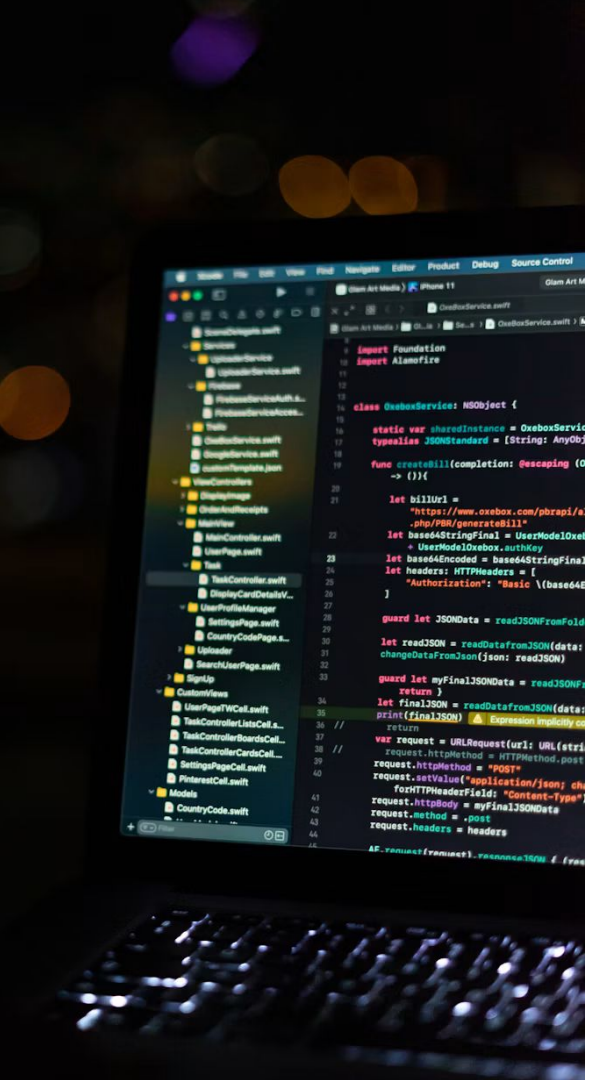

5954018 - Programação Orientada a Objetos

Aula 4 - Estruturas de Controle em Java

Prof. Dr. Denis Martins

DCM | FFCLRP | USP

Objetivos de Aprendizagem | Autoavaliação de progresso

Utilize esta lista como um guia para verificar se você consolidou os conceitos essenciais desta aula.

Decisão & Repetição

Diferenciar estruturas de decisão e de repetição.

Fluxo Condicional

Implementar o fluxo condicional usando `if`, `else if` e `switch`.

Controle de Loops

Controlar a execução de blocos de código usando loops (`for`, `while`, `do-while`).

Estruturas Aninhadas

Aplicar estruturas aninhadas para resolver problemas complexos.

Créditos: **slides baseados no material de aula original do Prof. Clever Farias.**

Estruturas de Controle

CONDICIONAL

Estruturas de Decisão

Condicionam a execução de um bloco de código ao resultado lógico (verdadeiro/falso) de uma expressão.

```
if (expressao) {
  // executa se verdadeiro
} else {
  // executa se falso
}
```

LOOPS

Estruturas de Repetição

Permitem a execução de um conjunto de comandos repetidas vezes, baseando-se em uma condição de parada.

```
while (condicao) {
  // executa repetidamente
  // enquanto verdadeiro
}
```

Estruturas de Decisão: if/else

Instrução Única

A instrução `if` normalmente espera somente **uma única instrução** no seu corpo.

Blocos de Comandos

Para incluir várias instruções dentro da mesma condição, é obrigatório o uso de blocos de comandos delimitados por `{ }`.

```
1 if (nota >= 5.0)
2     System.out.println("Aprovado, parabéns!!!");
3 else {
4     System.out.println("Reprovado");
5     System.out.println("Avalie opções de recuperação");
6 }
```

If/Else: Exemplo

```
public class AvaliacaoAluno {
    public static void main(String[] args) {
        int nota = 85;

        if (nota >= 90) {
            System.out.println("Situação: APROVADO COM EXCELENTE!");
        } else if (nota >= 70) {
            System.out.println("Situação: APROVADO.");
        } else if (nota >= 50) {
            System.out.println("Situação: APROVADO COM RESERVA.");
        } else {
            System.out.println("Situação: REPROVADO.");
        }
    }
}
```

Estruturas de Decisão: switch/case

CONDICIONAL

Fluxo de Decisão Linear

O comando `switch` executa blocos de comandos condicionalmente, avaliando de forma direta o valor de uma expressão.

Tipos Suportados na Expressão:

- **Constantes:** byte, short, int ou char
- **String:** suportado a partir da versão 7.0

```
switch (expressao){
    case constante_1:
        comando;
        break;
    ...
    case constante_n:
        comando;
        break;
    default:
        comando;
}
```

Switch/Case: Exemplo 1

```
...
int dia = 3;
switch (dia) {
    case 1:
        System.out.println("Domingo");
        break; // Sai do switch
    case 3:
        System.out.println("Terça-feira");
        break;
    default:
        System.out.println("Dia inválido.");
 }
```

Switch/Case: Exemplo 2

```
...
int nota;
...
switch (nota){
    case 10:
    case 9: System.out.println("Nota A"); break;
    case 8:
case 7: System.out.println("Nota B"); break;
case 6:
case 5: System.out.println("Nota C"); break;
default:
    System.out.println("Nota D");
    System.out.pritnln("Reprovado"); break;
}
```

Switch/Case: Exemplo 3

Estruturas Aninhadas (Nested)

Embora o `switch` seja excelente para comparações diretas, ele não consegue lidar com condições complexas. Nesses casos, usamos `if/else` dentro dos blocos para refinar a decisão.

1. Fluxo Principal (Switch)

Gerencia o fluxo de controle de nível macro (por exemplo, determinar em qual mês estamos).

2. Condição Secundária (If/Else)

Lida com regras de negócio específicas e refinadas (por exemplo, checar se fevereiro tem 28 ou 29 dias).

3. Resolução de Complexidade

O aninhamento permite modelar fluxos reais que possuem múltiplas camadas de decisão interdependentes.

```
...
int mes;
int numDias;
...
switch (mes){
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
        numDias=31; break;
    case 4:
    case 6:
    case 9:
    case 11:
        numDias=30; break;
    case 2:
        if (((ano%4 == 0) && (ano%100 != 0)) || (ano%400 == 0)) {
            numDias = 29;
        }
        else {
            numDias = 28;
        }
        break;
}
...
```

Estruturas de Repetição

Estruturas de repetição (loops) permitem a execução de **partes de um programa** de forma automatizada, **mais de uma vez**, de acordo com uma condição de parada.

`while`

Pré-teste

Avalia a condição antes de executar o bloco de código. Se a condição for falsa logo de início, o bloco nunca é executado.

`do-while`

Pós-teste

Garante que o bloco de código seja executado pelo menos uma vez, pois a condição é avaliada apenas ao final da repetição.

`for`

Variável de Controle

Ideal para situações com número definido de repetições. Integra inicialização, condição de parada e incremento em uma única linha.

Estruturas de Repetição: while

Funcionamento do Loop

Pré-teste: Testa a condição antes de executar o corpo do loop.

A repetição ocorre **até a condição se tornar falsa** (ou seja, enquanto ela for verdadeira).

Condição Inicial Falsa

Se a condição for **falsa inicialmente**, o bloco de código **nunca** será executado.

Isso diferencia o `while` de outras estruturas de pós-teste.

```
// Contagem regressiva
int contador = 5;
while (contador > 0) {
    System.out.println(contador);
    contador--; // Decrementa
}
```

Estruturas de Repetição: do-while

Funcionamento do Loop

Pós-teste

O `do-while` é um loop que garante a execução do bloco de código **pelo menos uma vez**, antes de verificar se a condição de continuação é verdadeira.

Fluxo de Execução

01. Executa o bloco `do`.

02. Testa a condição `while`.

03. Se for verdadeira, repete o ciclo. Caso contrário, encerra.

```java
// DoWhileText.java
public class DoWhileTest {
    public static void main(String args[]){
        int counter = 0;
        do {
            System.out.println(counter);
            ++counter;
        } while (counter <= 10);
    }
}
```

Estruturas de Repetição: for

O loop `for` é ideal quando se sabe o número exato de repetições, reunindo inicialização, condição e atualização de forma compacta.

Inicialização

Expressão de atribuição executada **uma única vez** no início do laço para definir o valor inicial da variável de controle.

Condição

Avaliada no **topo de cada repetição**. O laço continua enquanto for verdadeira e termina imediatamente quando o resultado for falso.

Atualização (ou Incremento)

Executada ao **final de cada repetição** para alterar a variável de controle. Incrementos mais comuns: `i++`, `i--`, `i+=n`.

```java
// ForCounter.java
public class ForCounter{
   public static void main(String args[]){
     for (int counter=1; counter <= 10; counter++){
       System.out.print(counter + " ");
       System.out.println();
   }
   ...
}
```

Exercícios

01

Fibonacci Básico

Escreva um programa Java que escreva os **10 primeiros termos** da sequência de Fibonacci.

02

Fibonacci com N

Modifique o programa anterior para imprimir os **N primeiros termos**, sendo N passado como argumento de linha de comando.

03

Soma e Média

Modifique o exercício anterior para que o aplicativo **calcule e escreva** os valores da somatória e da média aritmética.

04

Soma de Série

Calcule e escreva a soma dos **35 primeiros termos** da série:

```
5/2 - 10/3 + 15/4 - ...
```

Exercício Extra

Cenário: Você está desenvolvendo um sistema que tenta conectar a um servidor. A conexão só é bem-sucedida após um número específico de tentativas.

Tarefa: Implemente o programa para simular este processo usando do-while. O programa deve rodar até que a condição de sucesso seja atingida.

Entrada (Argumentos de Linha de Comando):

1. int maxTentativas: O número máximo de vezes que o sistema tentará conectar.
2. int sucessoTentativas: O número de tentativa que resultou no sucesso (este valor deve ser menor ou igual ao máximo).

Requisitos Técnicos:

1. O programa deve usar um loop do-while.
2. Ele deve imprimir em cada iteração: "Tentativa [X] de [Y]".
3. O loop deve parar e imprimir a mensagem "Conexão Estabelecida com Sucesso!" quando o contador interno atingir tentativas_sucesso.

Exemplo de Execução (Simulando): Se você passar 2 1, o programa deve rodar a primeira tentativa e parar. Se passar 3 3, ele deve rodar três vezes para conseguir a conexão

Conclusão

Estruturas de Controle

Decisão: Permite que o programa escolha caminhos diferentes com base em condições.

Repetição: Automatiza tarefas repetitivas, tornando o código mais conciso.

Resolução de Problemas

Desenvolvimento de raciocínio algorítmico através de práticas voltadas a cenários reais, como controle de conexões de rede e cálculo de séries matemáticas complexas.

Próximos Passos

Preparação para conceitos avançados de Programação Orientada a Objetos (POO), permitindo organizar a lógica estruturada dentro de classes e métodos reutilizáveis.

Material Adicional

📖 Leitura: **25 coisas que nós amamos no Java por JetBrains**

💻 Prática: **Pratique linha de comando com o jogo Command Line Murders**

Dúvidas e Discussão