Métodos + Estruturas de Decisão e Repetição

Conteudista: Prof. Esp. Alexander Gobbato Albuquerque
Revisão Textual: Prof.ª M.ª Rosemary Toffoli

Objetivos da Unidade:

- Aprender como inserir uma decisão e deixar que o programa tenha algumas possibilidades de execução;
- Entender como inserir uma estrutura de repetição nos métodos das classes criadas.

Contextualização

Material Teórico

Material Complementar

Referências

Contextualização

Vimos na unidade anterior como encapsular as informações, vimos também como criar os métodos e funções e executá-los. Agora que já sabemos tudo sobre encapsular, métodos e funções, vamos aprender a utilizar estruturas de decisão e estruturas de repetidos nos métodos do objeto.

Material Teórico

Expressões

O conceito de expressões está associado a fórmulas matemáticas onde um conjunto de variáveis e constantes numéricas se relacionam através de operadores matemáticos formando uma expressão e resultando em um valor.

Figura 1

Por exemplo, a fórmula da área do retângulo é $A = a \times b$, está fórmula utiliza duas variáveis: $\boldsymbol{A}$, $\boldsymbol{a}$ e $\boldsymbol{b}$, em que é o lado, b a altura e A que resultará na operação de multiplicação. Em informática uma expressão é a combinação de variáveis, constantes e operadores e que quando verificadas resultam um valor.

Operadores

Os operadores são utilizados nas expressões, por exemplo, 3 + 2, os números 3 e 2 são relacionados por um operador representado pelo sinal + que significa adição.

Os operadores se classificam em aritméticos, lógicos e literais, essa divisão depende do tipo de expressão que os mesmos serão inseridos, existe, ainda, um tipo de operador que é o relacional, por meio do qual se compara informações e o resultado é um valor lógico.

Tipos de Expressões

Expressões aritméticas resultam um tipo de dado numérico inteiro ou real, somente é permitido operador aritmético e variáveis numéricas nessas expressões, vejam na Tabela a seguir as operações aritméticas:

Tabela 1 – Operadores Aritméticos

Operador	Operação
+	Adição
-	Substração
*	Multiplicação
/	Divisão
%	Mod (Resto Divisão)

Como na matemática, em computação existe uma ordem de execução dos operadores, por exemplo, primeiro são resolvidos as expressões com os símbolos * e /, em seguida + e –.

Expressões Lógicas

Expressões lógicas são aquelas que retornam ou resulta um valor lógico e só existem dois tipos de dados: verdadeiros ou falsos. Na expressão lógica utilizamos um operador lógico. A Tabela a seguir ilustra a utilização e resultado da expressão.

Tabela 2 – Tabela Verdade

A	B	! A	! B	A .OU. B (||)	A .E. B (&&)
F	F	V	V	F	F
F	V	V	F	V	F
V	F	F	V	V	F
V	V	F	F	V	V

O operador lógico .NÃO.(!) sempre inverterá o valor do seu operando. O operador .OU. resultará verdadeiro quando um dos seus operando for Verdadeiro, já o operando .E. só resultará verdade quando todos os seus operando forem Verdadeiros.

Operadores Relacionais

Ainda existe outro tipo de operadores que é o relacional, esses operadores relacionais poder ser utilizados entre as operações lógicas.

Tabela 3 – Operador Relacional

Operador	Operação
==	Igual
!=	Diferente
<	Menor
<=	Menor ou Igual
>	Maior
>=	Maior ou Igual

Desvio Condicional Simples

Uma decisão simples é composta pela instrução se... então... fim, se a condição estabelecida for verdadeira, serão executas todas as instruções definidas entre o se... então e depois serão executadas todas as instruções que existirem após o fim.

Figura 2

Para representarmos o fluxograma acima, iremos escrever um trecho de código. Será criado uma classe de nome Conta com o atributo saldo, os métodos de acesso setSaldo() e getSaldo(). O método de nome sacar, recebe um parâmetro de entrada, dentro do método é realizada a seguinte operação:

- Se o valor de entrada for maior do que o saldo, exibe uma mensagem de saldo insuficiente, caso contrário realiza a operação de saldo, retirando o valor informado da variável saldo.

```
public class Conta {

    //definição de atributos
    private double saldo = 0;

    public void setSaldo(double vSaldo)..

    public double getSaldo()..

    public void sacar(double vValor){
        if(vValor>saldo)
            System.out.println("Saldo insuficiente");
        else
            saldo=saldo-vValor;
    }
}
```

Figura 3
Fonte: Reprodução

Desvio Condicional Composta

A estrutura condicional composta é similar com a estrutura condicional simples. Sempre um comando será executado independente da condição, ou seja, caso a condição seja verdadeira os comandos da condição serão executados ou os comandos da condição falsa serão executados.

Figura 4

Para representarmos o fluxograma acima, iremos escrever um trecho de código. Será criado uma classe de nome Boletim com os atributos media, *n*1 e *n*2, os métodos de acesso setN1(), getN1(), setN2() e getN2(). O método de nome calcularMedia(), dentro do método é realizada a seguinte operação:

- Calcula-se a média das notas;
- Se a média for menor do que 4, exibir "Conceito D";
- Se a média for maior ou igual a 4 e menor do que 6, exibir "Conceito C";
- Se a média for maior ou igual a 6 e menor do que 8, exibir "Conceito B";

- Se a média for maior ou igual a 8 e menor ou igual a 10, exibir “Conceito A”.

Em *Java* podemos representar a estrutura condicional por meio do seguinte trecho de código:

```
public class Boletim {
private float media, n1,n2;

public void setN1(int varN1)..
public float getN1()..
public void setN2(int varN2)..
public float getN2()..

public void calcularMedia(){
    media=(n1+n2)/2;
    if (media >=8 && media <=10)
       System.out.println("Conceito A");
    else if(media >=6 && media < 8)
        System.out.println("Conceito B");
    else if (media >=4 && media <6)
        System.out.println("Conceito C");
    else
        System.out.println("Conceito D");
    }

}
```

Figura 5
Fonte: Reprodução

Estrutura de Decisão *Switch Case*

O comando ***switch case*** pode ser útil quando precisamos fazer diversas verificações e executar diferentes comandos de acordo com o valor de alguma variável.

Figura 6

Para representarmos o fluxograma acima, iremos escrever um trecho de código. Será criado uma classe de nome Cambio com o atributo valorRS e os métodos de acesso setValor() e getValor(). O método de nome converterMoeda(), dentro do método é realizada a seguinte operação:

- Calcula-se a média das notas;
- Se a opção for igual a 1, exibir a conversão para dólar;
- Se a média for maior ou igual a 4 e menor do que 6, exibir "Conceito C";
- Se a média for maior ou igual a 6 e menor do que 8, exibir "Conceito B";
- Se a média for maior ou igual a 8 e menor ou igual a 10, exibir "Conceito A".

Em *Java* podemos representar a estrutura condicional por meio do seguinte trecho de código:

```
import javax.swing.*;
public class Cambio {
private float valorRS;

public void setValor(float varValor){
    valorRS=varValor;
}
public float getValor(){
    return valorRS;
}

public void converterMoeda(int op){
    switch (op) {
        case 1:
            JOptionPane.showMessageDialog(null,"Total em Dólares:" + (valorRS/1.89f));
            break;
        case 2:
            JOptionPane.showMessageDialog(null,"Total em Euros:" + (valorRS/3.75f));
            break;
        case 3:
            JOptionPane.showMessageDialog(null,"Total em Guaranis:" + (valorRS/0.20f));
            break;
        case 4:
            JOptionPane.showMessageDialog(null,"Total em Bolivares:" + (valorRS/0.89f));
            break;
        default:
            JOptionPane.showMessageDialog(null,"Opção inválida");
    }
}
}
```

Figura 7
Fonte: Reprodução

Estrutura de Repetição *While*

While é uma estrutura de repetição que compara uma variável, quando o resultado da comparação é verdadeiro, ele executa o bloco de instruções. O problema com estruturas de repetição, principalmente com *while*, é o que chamamos de *looping* infinito, esse nome é dado por a linha de comando é executada repetitivamente sem nunca chegar ao resultado final, por isso é imprescindível que uma determinada variável seja modificada de acordo com cada *loop*.

Veja o exemplo:

Figura 8

Para representarmos o fluxograma acima, iremos escrever um trecho de código. Será criado uma classe de nome QtdeNumPar com o atributo numero, os métodos de acesso setQtdeNumero() e getQtdeNumero(). O método de nome retQtdeNumPar(), que recebe um parâmetro de entrada, dentro do método é realizada a seguinte operação:

- Realiza uma repetição e com cada item verifica se o número é par, executando o MOD de 2;
- Cada vez que o resto da operação for 0, contabilizamos 1 para o método retornar o valor.

Em *Java* podemos representar a estrutura condicional por meio do seguinte trecho de código:

```
public class QtdeNumPar{

    private int numero;

    public void setQtdeNumero(int vNumero){
        numero = vNumero;
    }

    public int getQtdeNumero(){
        return numero;
    }

    public int retQtdeNumPar(int vNumero){

        int contador = 1;
        int qtdeNumPar = 0;

        while(contador <= vNumero){

            if(contador%2==0)
                qtdeNumPar++;

            contador++;
        }

        return qtdeNumPar;
    }
}
```

Figura 9
Fonte: Reprodução

Estrutura de Repetição *Do While*

DO WHILE é uma estrutura de repetição, tal como o próprio *while*. A principal diferença entre os dois é que *DO WHILE* irá fazer a comparação apenas no final do bloco de código, sendo representado da seguinte forma:

Figura 10

Em *Java* podemos representar a estrutura condicional de repetição *while* por meio do seguinte trecho de código:

```
public class QtdeNumPar{

    private int numero;

    public void setQtdeNumero(int vNumero){
        numero = vNumero;
    }

    public int getQtdeNumero(){
        return numero;
    }

    public int retQtdeNumPar(int vNumero){

        int contador = 1;
        int qtdeNumPar = 0;

        do{

            if(contador%2==0)
                qtdeNumPar++;

            contador++;
        } while(contador <= vNumero)

        return qtdeNumPar;
    }
}
```

Figura 11
Fonte: Reprodução

Estrutura de Repetição *For*

É um *loop* mais "fixo", permite executar o conjunto de sentenças com um número determinado de vezes. Sua expressão geral seria como a que segue:

Figura 12
Fonte: Reprodução

Em *Java* podemos representar a estrutura condicional de repetição *while* por meio do seguinte trecho de código:

```
public class QtdeNumPar{

    private int numero;

    public void setQtdeNumero(int vNumero){
        numero = vNumero;
    }

    public int getQtdeNumero(){
        return numero;
    }

    public int retQtdeNumPar(int vNumero){

        for(int contador =1; contador <= vNumero; contador++)
        {

            if(contador%2==0)
                qtdeNumPar++;
        }

        return qtdeNumPar;
    }
}
```

Figura 13

Fonte: Reprodução

Material Complementar

Indicações para saber mais sobre os assuntos abordados nesta Unidade:

Livros

Aprenda Programação Orientada a Objetos em 21 Dias

SINTES, T. **Aprenda programação orientada a objetos em 21 dias**. 1. ed. São Paulo: Pearson Education do Brasil, 2002, v. 1.

Java como Programar

DEITEL, P.; DEITEL, H. ***Java* como programar**. 8. ed. São Paulo: Pearson Education do Brasil, 2010.

Core Java

HORSTMANN, C.S.; CORNELL, G. (2010) ***Core Java***. 8. ed. São Paulo: Pearson Education do Brasil, 2010, v. 1.

Leitura

Orientação a Objetos – Simples Assim!

Clique no botão para conferir o conteúdo.

ACESSE

Referências

SINTES, T. **Aprenda programação orientada a objetos em 21 dias**. 1. ed. São Paulo: Pearson Education do Brasil, 2002, v. 1.

DEITEL, P.; DEITEL, H. ***Java* como Programar**. 8. ed. São Paulo: *Pearson Education* do Brasil, 2010.

HORSTMANN, C. S.; CORNELL, G. ***Core Java***. 8. ed. São Paulo: *Pearson Education* do Brasil, 2010, v. 1.