

CURSO DE PROGRAMAÇÃO EM JAVA

`Introdução para Iniciantes`

`Prof. M.Sc. Daniel Calife`

Índice

1 - A programação e a Linguagem Java.

1.1 Linguagens de Programação

1.2 Java

1.3 JDK

1.4 IDE

2 - Criando o primeiro programa em Java

2.1 Hello World

2.2 Main()

2.3 Comentários

3 - Tipos de Dados e Aritmética

3.1 Tipos de Dados

3.2 Variáveis e Constantes

3.3 Comandos de Entrada e Saída

4 - Estruturas de Controle: Seleção

4.1 If

-Operadores Relacionais-

4.2 If / Else

4.3 If Aninhados

4.4 Switch

5 - Estruturas de Controle: Repetição

5.1 While

5.2 Do / while

5.3 For

6 - Classes e Objetos

6.1 Programação Orientada à Objeto

6.2 Atributos

6.3 Métodos

6.4 Métodos Construtores

6.5 Encapsulamento

7 - Métodos

7.1 Tipos de Retorno

7.2 Lista de Parâmetros

7.3 Modificador Static

8 - Arrays

8.1 Vetores

8.2 Matrizes



1. A Programação e a Linguagem Java

1.1 Linguagens de Programação

Os computadores são máquinas poderosas que podem executar uma infinidade de tarefas, limitadas apenas por nossa imaginação. Cada computador possui sua própria linguagem nativa, chamada de Código de Máquina, que são instruções, convertidas em números e depois em bits 0 e 1.

As Linguagens de Programação foram criadas para facilitar a comunicação entre os programadores e a máquina, trazendo os comandos e instruções para um nível mais próximo do nosso entendimento. Estes comandos definidos por um programador são chamados de programa e normalmente representam um algoritmo.

Depois que um programador cria seu programa utilizando uma linguagem de alto nível, este código deve ser traduzido para a linguagem do computador. De forma simplificada, isto é feito através da Compilação.

Existem diversos tipos de linguagens e paradigmas de programação, neste material vamos aprender a linguagem Java que é uma linguagem de alto nível baseada na orientação à objetos.

1.2 Java

Java é uma das linguagens mais populares nos dias de hoje, sendo empregada para desenvolver soluções nas mais diversas áreas, tanto acadêmicas quanto corporativas. Esta popularidade se dá devido à uma de suas principais características: ser o mais portátil possível. Isso significa que o mesmo código Java pode ser executado em quaisquer plataformas.

A linguagem Java ao ser compilada não gera diretamente o código de máquina específico, mas sim um bytecode, que será posteriormente interpretado pela Máquina Virtual Java (JVM – Java Virtual Machine) que converte em tempo de execução o bytecode para a linguagem de máquina.

A Sun Microsystems foi a criadora do Java, que hoje é de propriedade da Oracle. Você pode saber mais sobre o Java neste link:

<http://www.oracle.com/br/technologies/java/overview/index.html>



1.3 JDK

Para utilizarmos a linguagem Java e criarmos nossos próprios programas vamos precisar instalar o Kit de Desenvolvimento Java, o JDK. Vamos utilizar o Java SE JDK, você pode baixar a última versão deste JDK neste link:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

1.4 Ambientes de Desenvolvimento (IDE)

Para agilizar o processo de desenvolvimento de nossos programas em Java podemos utilizar alguns ambientes específicos que trazem uma série de recursos para aumentar a produtividade.

Neste material vamos utilizar um ambiente simples, direcionado para iniciantes na programação, o JCreator. A versão gratuita deste ambiente, chamada de JCreator LE Version, pode ser encontrada aqui:

<http://www.jcreator.org/download.htm>

Existem outros ambientes de programação mais sofisticados, não deixando de ser gratuitos, os mais populares são:

Netbeans

<https://netbeans.org/downloads/index.html>

Eclipse

<http://www.eclipse.org/downloads/>

2. Criando o Primeiro Programa em Java

Uma das práticas mais comuns quando nos deparamos com uma nova linguagem de programação é criarmos um programa simples de teste chamado "Hello World!".

O objetivo deste programa é simplesmente termos contato com os principais comandos e estrutura da linguagem. Para isso, criamos um programa executável que exibe uma mensagem na saída, que pode ser em uma janela ou na linha de comando do sistema operacional.

2.1 Hello World em Java

Este é o código que apresenta nosso Hello World:

```
/* Nosso primeiro programa Hello.java
   imprime uma mensagem na saída */

public class Hello {
    public static void main(String[] args) {

        // Isto é um comentário!
        System.out.println("Olá Mundo!");

    }
}
```

A saída será: **Hello World!**

Este programa deve estar definido no arquivo Hello.java.

O comando responsável para exibir um texto na saída é o

```
System.out.println("Olá Mundo!");
```

Este comando é a chamada de um método, que exibe na saída a expressão enviada entre os parênteses (). O texto "Olá Mundo!" está definido entre aspas " " para indicar que deverá ser tratado como uma String, ou seja uma cadeia de caracteres.

Todo comando em Java termina com um ponto e vírgula (;).

Java é uma linguagem baseada na programação orientada à objetos, portanto todo programa será criado em Classes. Neste exemplo a classe criada foi a Hello, o nome da classe é simplesmente um identificador que deve começar com a primeira letra maiúscula. O comando que define uma classe em Java é:

```
public class Hello {  
    ...  
}
```

Todos os comandos e definições de uma classe são criados entre as chaves ({ }) que indicam o início e fim da classe.

O nome do arquivo que contém a classe Hello deve ter o mesmo nome, portanto Hello.java.

A palavra chave public será vista com mais detalhes adiante, neste momento nós simplesmente iremos utilizá-la nas definições de classes e métodos.

Dentro da classe Hello que criamos existe apenas a definição de um método, o main()

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

O método main() é o ponto de partida de todo programa Java, o seu conteúdo é definido entre as chaves que indicam seu início e fim. Todo método possui em sua definição dois parênteses, entre estes definimos os parâmetros que o método recebe, abordaremos os parâmetros mais adiante quando falarmos sobre Métodos.

Em nosso código existem algumas linhas que não são comandos ou definições, essas linhas são ignoradas pelo compilador pois são comentários:

```
/* Nosso primeiro programa Hello.java  
   imprime uma mensagem na saída */
```

e

```
// Isto é um comentário!
```

Comentários são muito úteis na documentação do seu código, eles podem descrever a versão ou a função da sua classe ou programa, podem também explicar algum trecho de código para uma futura alteração ou melhoria. Podemos definir os comentários de duas formas:

Por bloco, onde /* indica o começo do comentário e */ indica o fim do comentário;

Por linha, onde // indica que o texto à seguir é um comentário, que vai até a quebra da linha.

2.2 Compilar e executar o programa Hello

Para compilarmos o nosso programa devemos utilizar o comando `javac`, no mesmo diretório onde está o arquivo `Hello.java`

```
javac Hello.java
```

Este comando irá gerar o arquivo `Hello.class`, podemos então executar nosso programa com o comando:

```
java Hello
```

No prompt de comando você verá a saída do nosso primeiro programa.

Se você estiver utilizando o JCreator como ambiente de desenvolvimento basta executar o projeto, utilizando Run Project (tecla F5), que a saída será exibida no painel de General Output.

3. Tipos de Dados e Aritmética

Programas, de forma geral, são criados à partir de dados e algoritmos.

Algoritmos, esses, que manipulam e transformam esses dados.

Os dados são armazenados na memória do computador. A menor unidade de dado em um computador é um bit, que pode assumir os valores 0 ou 1.

À partir destes bits, podemos agrupá-los e criarmos novos tipos de dados que facilitam a utilização da memória. Cada linguagem possui um conjunto de tipos de dados pré definidos, estes tipos de dados são chamados de tipos primitivos.

O Java possui seus Tipos de Dados Primitivos, que são:

Tipo	Finalidade	Tamanho
boolean	Valor lógico, verdadeiro (true) ou falso (false).	1 bit
char	Armazena caracteres.	16 bits
byte	Armazena o valor de um byte, -128 até 127.	8 bits
short	Valor inteiro de 16 bits.	16 bits
int	Armazena valores inteiros.	32 bits
long	Inteiros muito grandes.	64 bits
float	Números reais, com precisão simples.	32 bits
double	Números reais, com precisão dupla.	64 bits

Podemos utilizar estes tipos de dados para armazenarmos valores na memória do computador. Criamos estes espaços na memória através de Variáveis e Constantes.

Variável é um espaço na memória que utilizamos para armazenar e manipular informações.

Chamamos de variáveis pois seu valor pode ser alterado durante a execução de um programa, ao contrário das Constantes, que mantém seu valor inicial.

Veja abaixo um programa que realiza uma soma entre dois valores inteiros.

```
/* Tipos de Dados e Variáveis
Somando dois inteiros em Soma.java */

public class Soma {

    public static void main(String[] args) {
        // criando duas variáveis
        int valor1, total;
```

```

        valor1 = 10; // atribuindo um valor
        // somando dois inteiros
        total = valor1 + 5; // o resultado é atribuído à
total

        // comando de saída
        System.out.println("O total é: " + total);
    }
}

```

Dentro do método principal main() temos o seguinte comando:

```
int valor1, total;
```

Este comando cria duas variáveis do tipo int, que pode armazenar números inteiros. A sintaxe para definirmos uma variável é:

```
tipo identificador;
```

Tipo pode ser qualquer tipo de dado presente em Java ou criado através de classes. Os identificadores, ou os nomes, das variáveis pode conter caracteres ou números, mas não podem começar com números.

Podemos definir mais de uma variável no mesmo comando, separando os identificadores por vírgula (,).

A segunda instrução dentro do main realiza uma atribuição.

```
valor1 = 10; // atribuindo um valor
```

O operador = realiza atribuições da direita para a esquerda, sempre resolvendo a expressão no lado direito primeiro e depois realizando a atribuição.

Após este comando a variável valor1 passa a ter o valor 10 na memória.

O comando seguinte realiza uma operação aritmética de soma:

```
total = valor1 + 5; // o resultado é atribuído
à total
```

O operador de soma + realiza a soma entre o valor da variável valor1 e 5, o resultado 15 é atribuído para a variável total.

Veja abaixo a relação com os operadores e operações em Java e suas respectivas prioridades de resolução:

Operador	Operação	Prioridade
()	Parenteses	1º
% ou mod	Resto de uma divisão inteira	2º
*	Multiplicação	3º
/	Divisão	3º
+	Soma	4º
-	Subtração	4º

Podemos utilizar os parenteses para alterar a ordem da resolução de alguma operação.

Por fim o resultado armazenado na variável total é impresso na saída com o seguinte comando:

```
System.out.println("O total é: " + total);
```

Podemos concatenar o texto juntamente com o valor da variável total, que será convertida para String.

3.1 Comandos de Entrada e Saída

Em Java podemos realizar a leitura (entrada) e a escrita (saída) de diversas formas, já utilizamos o método `println()` para escrevermos algumas informações na saída. Abaixo segue um programa que realiza a entrada e saída utilizando janelas de uma interface gráfica.

```
/* Entrada e Saída
Area de um círculo em Circulo.java */

import javax.swing.JOptionPane;

public class Circulo {

    public static void main(String[] args) {

        int raio; // raio que será digitado pelo usuário
        float area; // será calculada

        final float PI = 3.14f; // constante

        // String temporária
        String s;

        // comando para criar janela com campo de entrada
```

```

        s = JOptionPane.showInputDialog("Digite o valor do
raio.");
        // conversão de tipos de String para int
        raio = Integer.parseInt(s);

        area = PI * raio * raio; // calculo da área

        // comando para criar uma janela com um texto
        JOptionPane.showMessageDialog(null,
            "O valor da area e: " + area);
    }
}

```

Dentro do método main() as duas primeiras linhas definem as variáveis que utilizaremos:

```

int raio; // raio que será digitado pelo usuário
float area; // será calculada

```

Logo depois criamos uma constante, ou seja um valor que não será alterado:

```

final float PI = 3.14f; // constante

```

Para isso utilizamos a palavra chave final e também criamos um identificador com todas as letras maiúsculas, assim fica mais fácil distinguir as variáveis das constantes. Devemos obrigatoriamente inicializar o valor de uma constante.

Na linha seguinte criamos a variável s que armazena uma String, ou seja um texto. Ela é utilizada para receber o valor de retorno do comando de entrada.

```

String s;

```

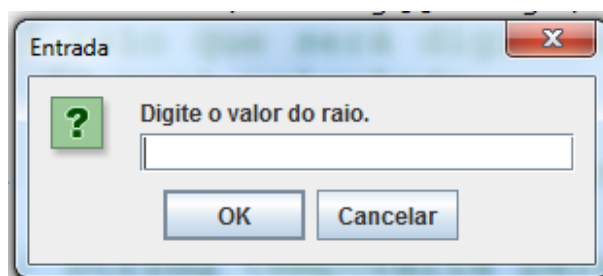
Nosso comando de entrada será o seguinte:

```

s = JOptionPane.showInputDialog("Digite o valor do
raio.");

```

O comando `JOptionPane.showInputDialog()` recebe como parâmetro dentro dos parenteses um texto que será apresentado ao usuário. Quando executado esse comando cria uma janela onde é apresentado um campo para a digitação..



Atribuímos o resultado do comando, ou seja o valor que foi digitado no campo,

para a variável `s` do tipo `String`. Isso porque, não importa o que foi digitado, o retorno do `showInputDialog()` será uma `String`. Portanto, na linha seguinte convertemos a `String s` para o tipo que queremos e atribuímos à variável `raio`:

```
raio = Integer.parseInt(s);
```

Podemos realizar toda a operação em um único comando que ficaria assim:

```
raio = Integer.parseInt(  
    s = JOptionPane.showInputDialog("Digite o valor  
do raio."));
```

Na próxima linha realizamos o cálculo da área:

```
area = PI * raio * raio;
```

E criamos uma janela para mostrarmos o resultado na tela:

```
JOptionPane.showMessageDialog(null,  
    "O valor da area e: " + area);
```

O `showMessageDialog()` recebe sempre dois parâmetros, o primeiro não utilizaremos, portanto definimos como `null` (nulo) o segundo será o texto mostrado na janela.

A classe `JOptionPane` não é nativa do Java, assim precisamos indicar que vamos utilizá-la, indicando o caminho do pacote que ela pertence. Fazemos isso no primeiro comando do programa, antes mesmo da definição da classe:

```
import javax.swing.JOptionPane;
```

4. Estruturas de Controle: Seleção

Mesmo na Programação Orientada à Objetos a base dos comandos para criarmos um programa é a Programação Estruturada, que tem como princípio o uso de três estruturas de controle para descrever qualquer algoritmo. São elas: Sequência, Seleção e Repetição.

A estrutura de Sequência apenas garante que todas as instruções sejam executadas em sequência, uma após a outra.

A estrutura de Seleção permite que seja executado um conjunto de instruções caso uma condição seja verdadeira.

Podemos implementar a Seleção de diversas formas em Java, vamos ver à seguir algumas delas.

4.1 If

O comando if testa uma condição e define um conjunto de instruções que serão executadas SE a condição for verdadeira. No exemplo à seguir temos um programa que lê um número inteiro e testa se ele é par.

```
/* Estruturas de Controle: Seleção
   if para achar numero par em Par.java */

import javax.swing.JOptionPane;

public class Par {

    public static void main(String[] args) {

        int numero, resto
        //ler a entrada do usuário
        float area; // será calculada
        numero = Integer.parseInt(
                                JOptionPane.
showInputDialog("Digite um numero"));
        resto = numero % 2; // resto da divisão por 2

        // testa se o resto é igual a 0
        if(resto == 0)
            JOptionPane.showMessageDialog(null,
                "O numero é par!");
    }
}
```

Após realizar a leitura de um número utilizando o método `showInputDialog`, realizamos a seguinte operação:

```
resto = numero % 2; // resto da divisão por 2
```

O operador `%` (módulo) retorna como resultado o resto de uma divisão, neste caso o resto do valor de `numero` dividido por 2. Quando um número é divisível por 2 o resto será 0, portanto será um número par.

Na linha seguinte esta condição será testada:

```
if(resto == 0)
```

Logo após o comando `if` devemos abrir parênteses, dentro destes parênteses iremos realizar o teste condicional, isso significa que a expressão deve retornar um resultado lógico, verdadeiro ou falso.

Neste caso a condição é: se `resto` for igual à 0. Para isso utilizamos um operador relacional de igualdade `==`. Podemos utilizar outros operadores relacionais:

Operador	Comparação
<code>==</code>	Igual
<code>!=</code>	Diferente
<code><</code>	Menor
<code><=</code>	Menor ou igual
<code>></code>	Maior
<code>>=</code>	Maior ou igual

Após o `if` temos na linha seguinte a instrução que só será executada caso a condição seja verdadeira:

```
JOptionPane.showInputDialog("Digite um  
numero"));
```

4.2 If / Else

Ao utilizarmos o `if` e selecionarmos um conjunto de instruções caso a condição seja verdadeira, podemos também incluir um conjunto de instruções que só serão executadas caso a condição for falsa, para isso utilizamos a diretiva `else`. O programa abaixo utiliza `if / else` para testar se um número é par ou ímpar.

```
/* Estruturas de Controle: Seleção  
   if / else para achar numero par em ParImpar.java */  
  
import javax.swing.JOptionPane;  
  
public class ParImpar {
```

```

public static void main(String[] args) {

    int numero;
    //ler a entrada do usuário
    numero = Integer.parseInt(
        JOptionPane.showInputDialog("Digite um
numero"));

    // expressão junto com o teste

    if(numero % 2 == 0)
        JOptionPane.showMessageDialog(null,
            "O numero é par!");

    else
        JOptionPane.showMessageDialog(null,
            "O numero é ímpar!");
    }
}

```

Neste programa realizamos a operação para retornar o resto da divisão junto com o teste de igualdade:

```

if(numero % 2 == 0)

```

Caso o resultado do teste condicional for verdadeiro o número é par:

```

JOptionPane.showMessageDialog(null,
    "O numero é PAR");

```

Com o comando else podemos definir o conjunto de instruções que serão executadas caso o resultado do teste for falso:

```

else
    JOptionPane.showMessageDialog(null,
        "O numero é ímpar!");

```

O trecho após o else só será executado caso o número seja ímpar.

4.3 If / Else Aninhados

Em Java podemos aninhar as Estruturas de Controle, ou seja, podemos utilizar uma estrutura if/else dentro de outra estrutura if/else.

O exemplo à seguir mostra um programa que lê duas notas, calcula a média e a situação do aluno:


```

/* Estruturas de Controle: Seleção
   if / else aninhados para avaliar a
   situação da média Media.java */

import javax.swing.JOptionPane;

public class Media{

    public static void main(String[] args) {

        float nota1

        //ler a entrada do usuário
        nota1 = Float.parseFloat(
            JOptionPane.showInputDialog("Digite a nota
1"));
        nota2 = Float.parseFloat(
            JOptionPane.showInputDialog("Digite a nota
2"));

        // calcula a média
        media = (nota1 + nota2) / 2;

        JOptionPane.showMessageDialog(null,
            "Sua média é!" + media );
        // avalia a situação do aluno
        if(media >= 7.0f)
        {
            JOptionPane.showMessageDialog(null,
                "Aprovado!");
        }

        else {
            if(media >= 4.0f)
            {
                JOptionPane.showMessageDialog(null,
                    "Recuperação...");
            }
            else
            {
                JOptionPane.showMessageDialog(null,
                    "Reprovado...");
            }
        }
    }
}

```

Este programa lê dois valores do tipo float, que representam as notas com valores decimais:

```
//ler a entrada do usuário
nota1 = Float.parseFloat(
    JOptionPane.showInputDialog("Digite a nota 1"));
nota2 = Float.parseFloat(
    JOptionPane.showInputDialog("Digite a nota 2"));
```

Logo à seguir calculamos a média entre elas e verificamos a situação do aluno:

```
if(media >= 7.0f)
```

Caso a média seja maior ou igual à 7.0 o aluno está aprovado, caso contrário ele pode estar de recuperação ou reprovado, para isso aninhamos outro teste condicional dentro do else:

```
else {
    if(media >= 4.0f)
    {
        JOptionPane.showMessageDialog(null,
            "Recuperação...");
    }
    else
    {
        JOptionPane.showMessageDialog(null,
            "Reprovado...");
    }
}
```

Assim, se a média for menor que 7, mas maior ou igual à 4.0 ele está de recuperação, caso contrário estará reprovado.

Notem que utilizamos chaves ({ }) neste exemplo, as chaves são necessárias quando vamos executar mais de uma instrução dentro de uma estrutura de controle, ou então para deixarmos explícitas as relações entre os ifs e elses.

Podemos escrever o mesmo trecho que avalia a situação da média de outra forma, utilizando if/ else if. Muitos programadores acham mais simples escrever desta forma:

```
// avalia a situação do aluno
if(media >= 7.0f)
    JOptionPane.showMessageDialog(null,
        "Aprovado!");
else
```

```

if(media >= 4.0f)
    JOptionPane.showMessageDialog(null,
                                   "Recuperação...");
else
    JOptionPane.showMessageDialog(null,
                                   "Reprovado...");

```

4.4 Testes Condicionais Compostos

Muitas vezes vamos precisar realizar mais de um teste condicional para executarmos um algoritmo, podemos combinar mais de um teste condicional através de operadores lógicos, E (&&) OU (||). Ainda podemos inverter um valor lógico utilizando a negação (!).

No programa à seguir utilizamos o operador E para achar qual o maior número entre três valores digitados:

```

/* Estruturas de Controle: Seleção
   testes condicionais compostos para
   achar número maior Maior.java */

import javax.swing.JOptionPane;

public class Maior{

    public static void main(String[] args) {

        int num1, num2, num3;

        //ler a entrada do usuário
        num1 = Integer.parseInt(
            JOptionPane.showInputDialog("Digite um
numero"));
        num2 = Integer.parseInt(
            JOptionPane.showInputDialog("Digite um
numero"));
        num3 = Integer.parseInt(
            JOptionPane.showInputDialog("Digite um numero));

        // encontra o maior entre os três números

        if(num1 > num2 && num1 > num3))
            JOptionPane.showMessageDialog(null,

```

```

        "O"+ num1 + "é o maior");
    else if (num2 > num3)
        JOptionPane.showMessageDialog(null,
            "O"+num2+"éomaior")
    else
        JOptionPane.showMessageDialog(null,
            "O"+num3+"éomaior")
    }
}

```

Para avaliarmos se o num1 é maior que num2 E também maior que num3 utilizamos o seguinte teste no if:

```

if(num1 > num2 && num1 > num3))

```

O operador lógico && verifica se ambas as comparações são verdadeiras, somente assim o resultado será verdadeiro. Os operadores lógicos && e || são booleanos, ou seja necessitam de dois operandos. Já a negação ! só precisa de um operando. Veja à seguir a tabela da verdade para os operadores lógicos:

A	B	A B	A && B	!A
F	F	F	F	V
F	V	V	F	V
V	F	V	F	F
V	V	V	V	F

4.5 Switch Case

O Switch Case é uma estrutura de seleção que permite de forma simples testarmos diversas possibilidades de valores de uma única variável. No exemplo à seguir temos um programa que lê dois valores e a operação matemática que queremos utilizar:

```

/* Estruturas de Controle: Seleção
   switch case para realizar operações
   Calculadora.java */

import javax.swing.JOptionPane;

public class Calculadora {

    public static void main(String[] args) {

        double operando1, operando2, total;
        char operacao;
    }
}

```

```

        //ler a entrada do usuário
        operando1 = Double.parseDouble(
            JOptionPane.showInputDialog("Digite o 1o.
valor"));
        operando2 = Double.parseDouble(
            JOptionPane.showInputDialog("Digite o 2o.
valor"));

        // ler caracter da operação
        operacao = JOptionPane.showInputDialog(
            "Digite
a operação").charAt (0));

        // realiza a operação de acordo com entrada do usuário
        switch (operacao)
        {
            case '+':
                total = operando1 + operando2;
                break;
            case '-':
                total = operando1 - operando2;
                break;
            case '*':
                total = operando1 * operando2;
                break;
            case '/':
                total = operando1 / operando2;
                break;
            default:
                JOptionPane.showInputDialog(null,
                    "Operador Inválido!");
                total = operando1 = operando2 = 0 ;
                operacao = ' ';
        }

        // mostra na saída o resultado
        JOptionPane.showMessageDialog(null,
            operando1 + " " + operacao + " " + operando2
            + " = " + total);
    }
}

```

Iremos utilizar dois valores do tipo double para armazenar os operandos, para isso utilizamos o

```
Double.parseDouble()
```

Para converter o resultado do `showInputDialog()`.

Também leremos um caractere que será a operação utilizada. Utilizamos o seguinte comando para extrair o primeiro char do retorno do `showInputDialog()`:

```
operacao = JOptionPane.showInputDialog(  
                                "Digite  
a operação").charAt (0));
```

Isto é necessário, pois o retorno seria uma `String`, que é incompatível com `char`.

Logo após avaliamos o valor da `operacao` utilizando o `switch`, a primeira etapa é indicar a variável:

```
switch (operacao)
```

Após este comando abrimos chaves (`{ }`) e utilizamos a diretiva `case` para testarmos as possibilidades:

```
case '+':  
    total = operando1 + operando2;  
    break;
```

Note que no `switch case` não testamos uma condicional, mas sim uma igualdade de valores. Logo após o `case` devemos utilizar dois pontos (`:`) e a sequência de comandos, colocando no final a instrução `break`. Devemos colocar `break` para sairmos do bloco de `switch`, caso contrário o programa irá continuar com a execução dentro do `switch`, podendo causar algum erro de lógica.

Dentro da estrutura `switch` podemos definir um conjunto de ações para quando não for encontrado nenhum valor nos `cases`:

```
default:  
    JOptionPane.showInputDialog(null,  
    "Operador Inválido!");  
    total = operando1 = operando2 = 0 ;  
    operacao = ' ';
```

Em nosso exemplo exibimos uma mensagem de erro e zeramos as variáveis para não exibir nenhuma "sujeira".

5. Estruturas de Controle: Repetição

A terceira Estrutura de Controle da programação estruturada é a Repetição. Uma das especialidades da computação é justamente automatizar as tarefas repetitivas, portanto é bastante comum o uso de estruturas de repetição.

Assim como a Seleção, podemos implementar de diversas formas a Repetição em Java.

5.1 While

While é a estrutura de repetição mais abrangente. Ela realiza um teste condicional e repete um bloco de comandos enquanto o teste for verdadeiro. Veja à seguir um exemplo de seu uso para realizarmos o lançamento de um dado 10 vezes:

```
/* Estruturas de Controle: Repetição
   while para realizar lançamento de Dados
   Dado.java */

import javax.swing.JOptionPane;

public class Dado {

    public static void main(String[] args) {

        int valor, contador;

        //devemos sempre inicializar os contadores
        contador = 1 ;

        while(contador <= 1) // condição de repetição
        {
            // valor aleatório entre 1 e 6
            valor = (int) (1+ Math.random()*6);

            // valor aleatório entre 1 e 6
            JOptionPane.showMessageDialog(null,
                "O valor do dado é:" + valor);

            // devemos incrementar o contador
            contador++;
        }
    }
}
```

Para realizar os dez lançamentos de dados vamos utilizar uma variável que será um contador, para isso devemos inicializar seu valor:

```
contador = 1 ;
```

No próximo comando definimos o laço de repetição, ou loop:

```
while (contador <= 10) // condição de repetição
```

Na sintaxe do while devemos abrir parenteses e definir uma expressão que resulte em um valor lógico (true ou false). Neste caso vamos realizar a repetição enquanto nosso contador for menor ou igual à 10. O bloco de comandos que será repetido é definido dentro das chaves ({ }) logo após o while.

Para simular o lançamento de um dado de seis faces vamos gerar um número aleatório entre 1 e 6:

```
valor = (int) (1+ Math.random()*6) ;
```

O método random() da classe matemática Math, gera um número positivo do tipo double, aleatório entre 0.0 e 1.0, incluindo o 0, mas não incluindo o 1. Portanto temos que manipular este número para que ele fique inteiro e entre 1 e 6.

Logo após a geração do número mostramos a saída na tela como valor sorteado:

```
contador++;
```

O operador de incremento ++ soma +1 à uma variável.

Devemos sempre garantir que em algum momento o laço de repetição acabe, caso contrário iremos criar um laço infinito.

5.2 do While

A estrutura de repetição do do while é uma variação da estrutura while. A diferença entre as duas é o ponto onde é realizado o teste condicional para continuar ou não o laço. Na estrutura while o teste é realizado no início do loop, já do do while o teste será realizado no final do laço.

Com a estrutura do do while podemos garantir que o laço será executado pelo menos uma vez, também podemos utilizar esta estrutura quando devemos primeiro atualizar uma série de valores e depois testar se continuamos no laço ou não.

Abaixo segue um programa que faz uso da estrutura do do while para realizar arremessos de dados até que o número 6 seja sorteado, também iremos realizar a somatória de todos os valores sorteados:


```

/* Estruturas de Controle: Repetição
   do while para realizar lançamento de Dados
   Dado2.java */

import javax.swing.JOptionPane;

public class Dado2 {

    public static void main(String[] args) {

        int valor, total;

        //devemos sempre inicializar os contadores
        total = 0 ;

        do {

            // valor aleatório entre 1 e 6
            valor = (int) (1+ Math.random()*6);

            // mostra na saída o resultado
            JOptionPane.showMessageDialog(null,
                "O valor do dado é:" + valor);
            total += valor;

        }while (valor !=6);

        // mostra na saída o total
        JOptionPane.showMessageDialog(null,
            "O total acumulado é " + total);
        }
    }
}

```

Neste exemplo não teremos um contador mas sim um acumulador, que é uma variável onde iremos totalizar a soma de todas as jogadas. Devemos sempre inicializar os acumuladores para garantir que o resultado seja correto.

```
total = 0 ;
```

Logo após temos a estrutura do while:

```

do {

    ....

}while (valor !=6);

```

Devemos iniciar com a palavra `do` e abrir chaves para definir o bloco de instruções do laço, logo após, definimos o teste condicional com o `while` e terminamos com ponto e vírgula.

Neste caso a condição é que o laço continue até que o número 6 seja sorteado. Dentro do laço após o sorteio do número, realizamos a somatória dos valores:

```
total += valor;
```

O operador `+=` é um operador de acumulação, poderíamos ter inscrito o mesmo comando da seguinte forma:

```
total += total + valor;
```

Podemos utilizar este operador com as outras operações (`*=`, `-=` e `/=`).

Quando o número 6 é sorteado o laço termina, no fim imprimimos o resultado da soma:

```
JOptionPane.showMessageDialog(null,  
    "O total acumulado é " + total);
```

5.3 for

A terceira forma de utilizarmos a estrutura de repetição em Java é o `for`. Em todo laço de repetição temos os seguintes passos:

- inicialização da variável de controle;
- teste condicional;
- atualização ou incremento da variável de controle;

A estrutura `for` é muito útil para laços que utilizam contadores, pois definimos estes três elementos de forma simples em sua sintaxe. Veja à seguir um programa que lê 5 números, realiza a somatória e calcula a média entre estes valores:

```
/* Estruturas de Controle: Repetição  
   for para ler números, somar e calcular a media  
   Media3.java */
```

```
import javax.swing.JOptionPane;
```

```
public class Media3 {
```

```
    public static void main(String[] args) {  
        //constante que define a quantidade de leituras  
        final int QTD = 5;
```

```

        double valor, total, media;

        for (int i = 1; i <= QTD; i++)
        {
            // ler a entrada do usuário
            valor = Double.parseDouble(
                JOptionPane.showInputDialog("Digite o " + i
                    + "o. valor"));

            total += valor; // somatória
        }

        //cálculo da média
        media = total / QTD;

        // mostra na saída o total e a média
        JOptionPane.showMessageDialog(null,
            "O total acumulado e: " + total
                + " a media e: " + media);
    }
}

```

A primeira definição que fazemos é a criação e inicialização da constante QTD:

```
final int QTD = 5;
```

Ao utilizarmos o modificador final indicamos que a variável QTD não poderá ser alterada após a sua inicialização, sendo assim uma constante. Utilizamos todas as letras em caixa alta para facilitar a identificação de constantes.


O interessante em usar a constante QTD é que podemos alterar o número de leituras que iremos fazer em apenas um ponto, esta técnica é uma prática que evita diversos erros de lógica.

Após a declaração e inicialização das variáveis temos a declaração do for:

```
for (int i = 1; i <= QTD; i++)
```

Dentro dos parenteses do for definimos toda a estrutura do laço. Na primeira definição temos a inicialização da variável de controle, note que podemos criar a variável na própria estrutura, assim ela só existirá dentro do bloco do for. Devemos separar as definições por ponto e vírgula (;). Na próxima parte definimos o teste condicional, também separado por ponto e vírgula. A última parte é o incremento, ou a atualização da variável de controle, uma vez que podemos executar qualquer operação nesta etapa. Assim a sintaxe do for se resume desta forma:

```
for (inicialização; teste condicional; atualização)
```



Após o for devemos abrir o bloco de instruções com chaves, seguindo as mesmas regras que utilizamos no if.

Dentro do for lemos os 5 números digitados pelo usuário e realizamos a somatória destes. Após o laço, calculamos a média e exibimos os resultados na saída.

6. Classes e Objetos

Neste capítulo iremos entender como o código fonte é organizado em um programa Java. Como já vimos o Java é uma linguagem Orientada à Objetos, isto significa que iremos organizar o código em Java através de Classes, e iremos utilizar estas classes criando instâncias de seus Objetos.

6.1 Programação Orientada à Objetos (POO)

A POO é baseada na idéia de trazermos a programação mais próxima ao mundo real. No nosso dia a dia estamos acostumados à interagir com diversos objetos, na verdade tudo ao nosso redor é objeto, como: carros, telefones, computadores, bicicletas, livros, tablets etc.

Quando utilizamos um destes objetos, como o telefone, por exemplo, não estamos preocupados em como ele funciona internamente, nós queremos simplesmente discar um número e fazer uma ligação para um amigo. Ou seja nós queremos utilizar os objetos, realizar alguma ação com ele. Na POO as ações que os objetos podem desempenhar são chamadas de Comportamento. Definimos o Comportamento em Java através de Métodos.

Além do Comportamento os objetos também apresentam características, como no caso do telefone temos: tamanho, peso, cor, marca, modelo, tecnologia etc. Estas características dos objetos na POO são chamadas de Atributos. O estado físico ou lógico de um objeto, como por exemplo se ele está ligado ou desligado, também é definido através de seus Atributos.

Antes de utilizarmos um objeto, como uma casa, devemos construí-la. Para construir uma casa devemos primeiro determinar como ela será, seu projeto. Fazemos isto através de uma planta, que contém todas as características que a casa deve ter. À partir desta planta podemos construir diversas casas. Na POO iremos definir os Atributos e Comportamento de um objeto através de uma Classe. A Classe é como uma planta de uma casa, e à partir dela podemos instanciar diversos Objetos.

6.2 Classes

Quando criamos uma Classe estamos criando um novo tipo de dado que queremos utilizar. Para definirmos uma nova Classe em Java devemos criar um novo arquivo com a extensão .java. Assim, se quisermos criar a Classe Casa, devemos criar o arquivo Casa.java, veja à seguir como definimos uma Classe:

```

/* Classes
   definição da Classe Casa no arquivo
   Casa.java */

class Casa {

    //aqui vamos definir os Atributos
    // e os métodos da Classe

}

```

Definimos uma Classe através do comando class seguido do nome da Classe, sempre utilizamos a primeira letra maiúscula e as restantes minúsculas, para facilitar a identificação de classes em um código.

Todas as definições de uma classe estarão dentro das chaves ({ }) que definem seu escopo.

Vamos melhorar nossa Casa definindo alguns atributos para ela.

6.3 Atributos

Os Atributos são as características de um Objeto. Iremos definir os atributos utilizando variáveis de tipos primitivos (int, float, boolean etc) e também podemos utilizar outras Classes/Objetos (String, Casa, Carro, Motor, etc). Estas variáveis serão membros dos objetos que criaremos.

Os Atributos criados na Classe definem o Estado do Objeto. Veja à seguir nossa Classe Casa com seus atributos:

```

/* Classes
   definição da Classe Casa no arquivo
   Casa.java */

class Casa {
    //Atributos da Classe
    int numero;
    String cor;
    boolean iluminacao;

    //aqui vamos definir os Atributos
    // e os métodos da Classe

}

```

Definimos três atributos para a classe Casa, número, cor e iluminação. A iluminação é um tipo booleano que pode armazenar verdadeiro (true) ou falso (false), e indica se a iluminação está ligado ou não.

Como vimos, uma classe é apenas uma definição, uma planta, de como o objeto será construído. Para utilizarmos a classe precisamos instanciar um Objeto. Vamos criar uma nova classe em um novo arquivo, porém esta classe terá o método main(), para que o programa seja executado:

```
/* Classes
Programa principal para testar a Casa
TestaCasa.java */

import javax.swing.JOptionPane;

public class TestaCasa {

    public static void main(String[] args) {

        Casa casa1; // referência para Casa
        casa1 = new Casa(); // instanciar objeto

        // acesso aos atributos
        casa1.numero = 10;
        casa1.cor = "amarela";
        casa1.iluminacao = false;

        // mostrar na tela o estado atual da casa1
        JOptionPane.showMessageDialog(null, "A Casa de no. "
+
            casa1.numero + " de cor " + casa1.cor + " esta
com a luz " +
            (casa1.iluminacao?"ligada":"desligada"));
    }
}
```

Dentro do main() criamos uma referência para uma Casa:

```
Casa casa1;
```

Neste momento ainda não temos um objeto de Casa, a referência casa1 ainda tem o valor nulo (null). Na linha seguinte instanciamos um objeto e o atribuímos para casa1:

```
casa1.numero = 10;
casa1.cor = "amarela";
casa1.iluminacao = false;
```

Para acessarmos os membros de um objeto utilizamos o ponto (.), assim temos acesso e podemos escrever ou ler seus atributos e chamar seus métodos.

Logo em seguida exibimos na saída um texto com o estado atual do nosso objeto casa1, ou seja os valores de seus atributos:

```
// mostrar na tela o estado atual da casa1
JOptionPane.showMessageDialog(null, "A Casa de no. "
+
    casa1.numero + " de cor " + casa1.cor + " esta
com a luz " +
    (casa1.iluminacao?"ligada":"desligada"));
```

Neste trecho utilizamos um operador ternário de seleção o "?:". Este operador pode ser utilizado quando queremos fazer um if de forma simples, ele segue a seguinte sintaxe:

```
teste condicional ? retorno verdadeiro : retorno falso ;
```

6.4 Métodos

Vamos continuar criando nossa classe Casa, agora iremos definir o seu Comportamento, ou seja as ações que ela pode executar. Os comportamentos são definidos através de deus Métodos.

Métodos são trechos de código que realizam uma tarefa específica e podem ou não retornar um valor como resultado, iremos ver os métodos com mais detalhes no próximo capítulo.

Vamos criar dois métodos para a classe Casa, um para exibir seu estado e outro para ligar ou desligar a iluminação.

```
/* Classes
definição da Classe Casa no arquivo
Casa.java */
```

```
import javax.swing.JOptionPane;
```

```
class Casa {

//Atributos da Classe
int numero;
String cor;
boolean iluminacao;
```



```

// Métodos da Classe
void mudaInterruptor()
{
    iluminacao = !iluminacao; // inverte valor
}

void exibeCasa()
{
    JOptionPane.showMessageDialog(null, "A Casa de
no. "
+ numero + " de cor " + cor + " esta com a luz "
+ (iluminacao?"ligada":"desligada"));
}
}

```

Para criamos um Método precisamos primeiro definir três elementos que definem sua assinatura: tipo de retorno, identificador e lista de parâmetros. O primeiro método que criamos foi:

```

void mudaInterruptor()
{
    iluminacao = !iluminacao; // inverte valor
}

```

Este método não tem retorno, portando utilizamos void, seu nome é mudaInterruptor, e logo após abrimos parenteses, como não temos nenhum parâmetro, dentro dos parenteses está vazio. Utilizamos o operador de negação (!) para inverter o valor lógico da iluminacao.

Criamos também o método exibeCasa, que facilita a exibição do estado do objeto. Note que acessamos diretamente os atributos da classe.

Vamos testar a nova versão da nossa Casa utilizando a classe principal, com o método main:

```

/* Classes
Programa principal para testar a Casa
TestaCasa.java */

import javax.swing.JOptionPane;

public class TestaCasa {

    public static void main(String[] args) {

        Casa casa1; // referência para Casa
        casa1 = new Casa(); // instanciar objeto

        // acesso aos atributos
    }
}

```

```

        casa1.numero = 10;
        casa1.cor = "amarela";
        casa1.iluminacao = false;

        // chamada ao métodos
        casa1.exibeCasa();
        casa1.mudaInterruptor();
        casa1.exibeCasa();
    }
}

```

Neste programa fazemos as chamadas aos métodos da casa1:

```

        casa1.exibeCasa();
        casa1.mudaInterruptor();
        casa1.exibeCasa();

```

Chamamos os métodos da mesma forma que acessamos os atributos, utilizando o operador ponto (.). Note que ao chamar os métodos utilizamos parenteses logo após o identificador.

6.5 Métodos Construtores

Em Java todas as classes possuem um tipo especial de método: os Construtores. Os métodos construtores são obrigatórios e são chamados automaticamente quando instanciamos um objeto.

O objetivo dos métodos construtores é garantir a correta inicialização de um novo objeto, definindo, por exemplo, os valores padrão de seus atributos. Vamos definir nossos Construtores para a classe Casa:

```

/* Classes
   definição da Classe Casa no arquivo
   Casa.java */

```

```

import javax.swing.JOptionPane;

```

```

class Casa {
    //Atributos da Classe
    int numero;
    String cor;
    boolean iluminacao;

    //Construtores
    Casa() // construtor Padrão
    {

```

```

        numero = 0;
        cor = "Branca";
        iluminacao = false;
    }

    // construtor com parâmetros
    // para inicializar atributos
    Casa(int n, String c, boolean i)
    {
        numero = n;
        cor = c;
        iluminacao = i;
    }

    // Métodos da Classe
    void mudaInterruptor()
    {
        iluminacao = !iluminacao; // inverte valor
    }

    void exibeCasa()
    {
        JOptionPane.showMessageDialog(null, "A Casa de
no. "
+ numero + " de cor " + cor + " esta com a luz "
+ (iluminacao?"ligada":"desligada"));
    }
}

```

Os métodos Construtores possuem o mesmo nome da classe e não tem nenhum tipo de retorno:

```

Casa(int n, String c, boolean i)
{
    numero = n;
    cor = c;
    iluminacao = i;
}

```

Dentro dos parenteses devemos definir o tipo e identificadores dos parâmetros e separá-los com vírgulas. No programa para testar nossa classe utilizamos os dois Construtores:

```

/* Classes
Programa principal para testar a Casa

```

```

TestaCasa.java */

import javax.swing.JOptionPane;

public class TestaCasa {

    public static void main(String[] args) {

        Casa casa1, casa2; // referências para Casa

        casa1 = new Casa(); // instanciar objeto
        casa2 = new Casa(15, "Roxa", true);

        // chamada ao métodos
        casa1.exibeCasa();
        casa2.exibeCasa();
    }
}

```

Podemos instanciar um objeto utilizando o construtor padrão:s:

```

casa1 = new Casa();

```

Ou utilizarmos o construtor que recebe valores como parâmetro para inicializar os atributos do novo objeto:

```

casa2 = new Casa(15, "Roxa", true);

```

Devemos respeitar a ordem e tipos dos parâmetros definidos nos métodos.

6.6 Encapsulamento

Um dos principais conceitos da POO é que as Classes devem ser independentes e garantirem a integridade de seus dados. Para isso ocorrer devemos restringir o acesso direto aos seus membros, principalmente aos seus Atributos.

Assim, iremos Encapsular os atributos de nossas Classes utilizando os modificadores de acesso. Veja como ficará nossa Classe Casa:

```

/* Classes
   definição da Classe Casa no arquivo
   Casa.java */

import javax.swing.JOptionPane;

class Casa {
    //Atributos da Classe

```

```
private int numero;
private String cor;
private boolean iluminacao;

//Construtores
public Casa() // construtor Padrão
{
    numero = 0;
    cor = "Branca";
    iluminacao = false;
}
// construtor com parâmetros
// para inicializar atributos
public Casa(int n, String c, boolean i)
{
    numero = n;
    cor = c;
    iluminacao = i;
}

//Métodos de Acesso
public void setNumero(int n)
{
    if(n > 0) // somente aceita números positivos
        numero = n;
}
public int getNumero()
{
    return numero;
}
public void setCor(String c)
{
    cor = c;
}
public String getCor()
{
    return cor;
}
public boolean getIluminacao()
{
    return iluminacao;
}
```

```

// Métodos da Classe
public void mudaInterruptor()
{
    iluminacao = !iluminacao; // inverte valor
}

public void exibeCasa()
{
    JOptionPane.showMessageDialog(null, "A Casa de
no. "
+ numero + " de cor " + cor + " esta com a luz "
+ (iluminacao?"ligada":"desligada"));
}
}

```

Note que antes de todos Atributos e Métodos utilizamos uma palavra chave, `private` ou `public`. Essas palavras são os modificadores de acesso, que liberam ou restringem o acesso direto aos membros de uma classe. Existem três modificadores:

- `public`: Acesso público, qualquer Classe pode acessar os membros públicos;
- `private`: Acesso privado, somente a própria Classe tem acesso aos seus membros;
- `protected`: Somente a própria Classe ou Classes derivadas (Herança) têm acesso aos seus membros.

Definimos os Atributos da Classe Casa como `Private`, assim nenhuma outra classe poderá acessá-los diretamente. Se desejarmos que outras Classes definam ou modifiquem estes atributos devemos criar os Métodos de Acesso. Estes métodos seguem o padrão de nomenclatura `"set"` e `"get"` antes dos nomes dos atributos, os métodos `set` permitem a escrita nos valores e os métodos `get` a leitura.

A idéia por trás desta técnica é que iremos garantir que os valores que serão incluídos estão corretos, como no nosso `setNumero()`:

```

public void setNumero(int n)
{
    if(n > 0)
        numero = n;
}

```

O método `setNumero()` garante que não será setado nenhum número negativo,

garantindo a integridade do Atributo.

Em alguns casos podemos não definir os métodos de acesso, como por exemplo o método set para iluminação, que poderá somente ser modificado através do `mudaInterruptor()`.

Para testarmos os métodos utilizaremos a seguinte classe com o `main()`:

```
/* Classes
Programa principal para testar a Casa
TestaCasa.java */

import javax.swing.JOptionPane;

public class TestaCasa {

    public static void main(String[] args) {

        Casa casa1, casa2; // referências para Casa

        casa1 = new Casa(); // instanciar objeto
        casa2 = new Casa(15, "Roxa", true);

        //métodos de acesso para setar valores
        casa1.setNumero(10);
        casa1.setCor("Amarela");

        // chamada ao métodos
        JOptionPane.showMessageDialog(null, "A Casa de no. "
            + casa1.getNumero() + " de cor " + casa1.getCor()
            + " esta com a luz " +
            (casa1.getIluminacao()?"ligada":"desligada"));

        casa2.exibeCasa();
    }
}
```

Utilizamos os métodos set para modificar os atributos do objeto `casa1`:

```
casa1.setNumero(10);
casa1.setCor("Amarela");
```

e utilizamos os métodos get para realizar a leitura e exibir na saída:

```
JOptionPane.showMessageDialog(null, "A Casa de no. "
    + casa1.getNumero() + " de cor " + casa1.getCor()
    + " esta com a luz " +
    (casa1.getIluminacao()?"ligada":"desligada"));
```

7. Métodos

Os Métodos são uma parte essencial da POO, pois eles nos ajudam a modularizar nossos programas. Isto significa separar nossos problemas em partes menores e mais específicas, permitindo assim uma melhor reutilização de nossos códigos.

Em outras linguagens os métodos podem ser chamados de Funções ou Procedimentos, estes nomes são aplicados de acordo com o seu retorno. Caso exista o retorno de um valor ou um resultado este é uma Função, caso não tenha um retorno é um Procedimento.

Os Métodos sempre devem ser definidos em Classes, eles podem ou não retornar valores e podem ou não receber parâmetros.

7.1 Tipos de Retorno

Em todos os métodos devemos definir seu tipo de retorno, o tipo faz parte da assinatura do método, conforme já vimos:

```
tipo_de_retorno identificador( parâmetros )
```

O retorno de um método é um valor único que pode ser qualquer um dos tipos primitivos do Java ou um objeto de uma Classe. Quando o método não deve retornar nenhum valor definimos seu retorno como void (vazio).

Um exemplo de método que retorna valor são os métodos de acesso: get. Assim como, os métodos de acesso set, não retornam nenhum valor, pois somente modificam os valores dos atributos.

Na nossa classe Casa, definimos métodos que retornam valores:

```
public int getNumero()  
{  
    return numero;
```

O método getNumero() retorna um valor do tipo int. Quando definimos que haverá um retorno somos obrigados à utilizar o comando return juntamente com um valor, variável ou expressão que tenha como resultado o tipo definido.

Quem fizer o chamado de um método com retorno pode ou não tratar este retorno, guardando o valor em uma variável, imprimindo na saída ou como parte de uma outra expressão.

Também na classe Casa definimos métodos que não retornam valores:


```
public void mudaInterruptor()
{
    iluminacao = !iluminacao; // inverte valor
}
```

O método `mudaInterruptor()` simplesmente inverte o valor lógico do atributo `iluminacao` e não retorna nenhum valor. Quando definimos `void` para o tipo de retorno de um método não podemos utilizar o comando `return`.

7.2 Lista de Parâmetros

Para realizar seu objetivo um método pode precisar ou não da entrada de valores. Por exemplo, um método que faz o cálculo de potência precisa receber a base e o expoente que deverá elevar.

Os métodos recebem dados, valores ou informações através de sua Lista de Parâmetros. Estes parâmetros podem ser tipos primitivos do Java ou objetos de Classes. Vale lembrar que a ordem dos parâmetros sempre será mantida em sua chamada e que os identificadores destes parâmetros só existirão dentro de seus métodos. Vamos criar uma classe `Calculadora` com alguns métodos que utilizam parâmetros:

```
/* Métodos
   definição da Classe Calculadora no arquivo
   Calculadora.java */

class Calculadora {
    public static double PI = 3.14;

    // soma dois inteiros
    public int soma(int a, int b)
    {
        return (a + b);
    }

    // soma três float
    public float soma(float a, float b, float c)
    {
        return (a + b + c);
    }

    // calcula a potencia
    public int potencia(int base, int exp)
    {
        int resultado = 1;
```

```

        for(int i = 1; i <= exp; i++)
        {
            resultado *= base;
        }
        return resultado;
    }
}

```

Definimos os parâmetros que o método irá receber indicando seu tipo e identificador, separando por vírgulas caso haja mais de um parâmetro. Desta forma sempre que este método for chamado será necessário passar os valores de a e de b.

Na mesma calculadora criamos outro método com o mesmo identificador soma(), porém com parâmetros diferentes:

```

public float soma(float a, float b, float c)
{
    return (a + b + c);
}

```

Chamamos este processo de Sobrecarga de Métodos. Podemos escrever diversas variações do mesmo método, porém todas elas necessitam de número ou tipo de parâmetros diferentes, só assim o compilador saberá qual dos métodos irá chamar.

Veja a classe principal que testa a classe Calculadora:

```

/* Métodos
Programa principal para testar a Calculadora
TestaCalculadora.java */

import javax.swing.JOptionPane;

public class TestaCalculadora {

    public static void main(String[] args) {

        Calculadora c = new Calculadora();

        JOptionPane.showMessageDialog(null, "A soma de 10 +
15 é: " +
            c.soma(10, 15) );

        JOptionPane.showMessageDialog(null,

```

```

        "A soma de 5.75 + 4.2 + 3.2 é: " +
        c.soma(5.75f, 4.2f, 3.2f) );

    JOptionPane.showMessageDialog(null,
        "O número 3 elevado ao cubo é: " +
        c.potencia(3, 3) );
}
}

```

7.3 Modificador Static

Você já deve ter reparado no modificador static no atributo PI da classe Calculadora:

```
public static double PI = 3.14;
```

e também no método main():

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

A palavra chave static indica que um método ou um atributo estará sempre presente na memória, e só terá um valor compartilhado entre todos os objetos, se este for um atributo.

Desta forma, dizemos que um atributo static ou um método static são membros da classe e não de um objeto. Isto significa que podemos acessá-los diretamente, sem a necessidade de instanciar um objeto.

A classe Math, presente no Java, tem seus métodos e atributos estáticos, assim é possível chamá-los à qualquer momento. A classe main() é estática para que a JVM chame seu programa diretamente por este método. Podemos recriar a classe Calculadora com métodos estáticos:

```

/* Métodos
   definição da Classe Calculadora no arquivo
   Calculadora.java */

class Calculadora {
    public static double PI = 3.14;

    // soma dois inteiros
    public static int soma(int a, int b)
    {
        return (a + b);
    }

    // soma três float
    public static float soma(float a, float b, float c)

```

```

    {
        return (a + b + c);
    }

    // calcula a potencia
    public static int potencia(int base, int exp)
    {
        int resultado = 1;
        for(int i = 1; i <= exp; i++)
        {
            resultado *= base;
        }
        return resultado;
    }
}

```

E assim chamar seus membros sem a necessidade de instanciar um objeto:

```

/* Métodos
   Programa principal para testar a Calculadora
   TestaCalculadora.java */

import javax.swing.JOptionPane;

public class TestaCalculadora {

    public static void main(String[] args) {

        JOptionPane.showMessageDialog(null, "A soma de 10 + 15 é: "
        + Calculadora.soma(10, 15) );

        JOptionPane.showMessageDialog(null,
        "A soma de 5.75 + 4.2 + 3.2 é: " +
        Calculadora.soma(5.75f, 4.2f, 3.2f) );

        JOptionPane.showMessageDialog(null,
        "O número 3 elevado ao cubo é: " +
        Calculadora.potencia(3, 3) );
    }
}

```

8. Arrays

Dependendo da complexidade de nosso algoritmo e principalmente do número de informação que teremos que armazenar e manipular, iremos precisar organizar melhor nossos dados. Para isso criamos as Estruturas de Dados.

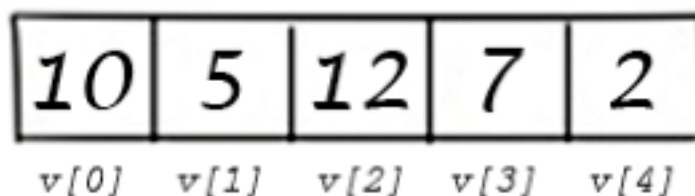
Uma das estruturas de dados mais simples são os arrays, ou, os arranjos. Que são um conjunto de dados do mesmo tipo, inteiros por exemplo, alocados na memória de forma sequencial. Estes arranjos podem ter uma dimensão, Vetores, ou mais de uma dimensão, Matrizes.

8.1 Vetores

Imagine que em um programa você necessita armazenar dez (10) valores inteiros. Para isso você poderia criar dez variáveis diferentes e armazenar seus valores. Agora imagine que não são dez, mas sim 100, 1.000 ou 1 milhão de valores. Você não iria criar 1 milhão de variáveis, não é?

Na verdade iremos criar sim, porém de uma forma diferente, vamos criar um Vetor. Um vetor é um conjunto de dados criados de forma sequencial, assim não precisamos criar uma variável para cada dado na memória, todos eles possuem o mesmo nome, acessamos cada um individualmente através da sua posição, do seu índice.

Veja à seguir uma imagem que representa um Vetor:



Nesta imagem temos um vetor chamado v com 5 posições na memória para armazenar números inteiros. Cada posição possui um índice, e é através deste índice que tratamos cada informação individualmente.

Na memória do computador começamos à contar à partir do zero (0), assim um vetor com 5 posições terá os índices de 0 à 4. Para acessarmos à primeira posição devemos indexar o vetor desta forma:

$v[0] = 10;$

Vamos criar uma classe que cria um vetor, lê seus valores e depois indica qual o maior número entre eles.

```

/* Arrays: Criação e manipulação de
   vetores em Vetor.java */

import javax.swing.JOptionPane;

public class Vetor {

    public static void main(String[] args) {

        final int TAM = 10; // tamanho do vetor

        // cria vetor de inteiros
        int valores[] = new int[TAM];

        int maior = 0; // vai armazenar o maior valor

        for(int i = 0; i < TAM; i++) // leitura do vetor
        {
            valores[i] = Integer.parseInt(
                JOptionPane.showInputDialog("Digite o "
                    + (i + 1) + "o. número"));
        }

        //procura o maior valor, maior que zero
        for(int i = 0; i < TAM; i++)
        {
            if(valores[i] > maior)
                maior = valores[i];
        }

        // imprime o maior
        JOptionPane.showMessageDialog(null, "O maior valor é:
"
            + maior );
    }
}

```

Para criarmos nosso vetor utilizamos o seguinte comando:

```

final int TAM = 10; // tamanho do vetor

```

Note que depois que criamos o vetor não é possível alterar seu tamanho na memória.

Para lermos todos os valores do vetor precisaremos percorrer todas as suas posições e armazenar cada valor individualmente, uma das formas mais práticas de percorrer um vetor é utilizar um laço for:

```
for(int i = 0; i < TAM; i++) // leitura do vetor
{
    valores[i] = Integer.parseInt(
        JOptionPane.showInputDialog("Digite o "
            + (i + 1) + "o. número"));
}
```

Repare que utilizamos a variável de controle do for para realizar o loop e também como índice para acessar uma posição no vetor:

```
valores[i]
```

Logo após, percorremos mais uma vez o vetor para encontrar qual o seu maior valor armazenado, desde que seja maior que zero:

```
for(int i = 0; i < TAM; i++)
{
    if(valores[i] > maior)
        maior = valores[i];
}
```

Utilizamos novamente o for para percorrer o vetor.

8.2 Matrizes

Podemos criar e utilizar arrays com mais de uma dimensão, estes arranjos são chamados de Matrizes.

O tipo mais utilizado de matriz é a bidimensional, que possui um certo número de linhas e de colunas, funcionando como uma tabela. Na imagem à seguir podemos ver como uma matriz é criada na memória.

	0	1	2	3
0	10	5	12	0
1	1	0	77	20
2	13	6	18	2
3	4	99	43	7

Nesta imagem temos uma matriz de nome m que possui quatro linhas e quatro colunas. Agora possuímos dois índices para cada posição na memória, portanto para acessarmos a primeira posição da matriz devemos utilizar o seguinte comando:

```
m[0][0] = 10;
```

O primeiro índice representa a linha que queremos acessar e o segundo representa a coluna.

Vamos criar um programa que cria uma matriz, lê seus valores e realiza a somatória das suas linhas e depois suas colunas:

```
/* Arrays: Criação e manipulação de
   matrizes em Matriz.java */

import javax.swing.JOptionPane;

public class Matriz {

    public static void main(String[] args) {

        final int LIN = 4; // no. de linhas
        final int COL = 4; // no. de colunas

        double total = 0; // acumulador

        // cria matriz bidimensional de double
        double matriz[][] = new double[LIN][COL];

        // realiza leitura dos valores
        for(int i=0; i < matriz.length; i++)
        {
            for(int j=0; j<matriz[i].length; j++)
            {
                matriz[i][j] = Double.parseDouble(
                    JOptionPane.showInputDialog("Digite o
valor "
                                                + "["+(i)+""]["+(j)+""]));
            }
        }

        // realiza a somatória das linhas
        for(int i=0; i < matriz.length; i++)
```



```

    {
        for(int j=0; j<matriz[i].length; j++)
        {
            total += matriz[i][j];
        }
        // imprime o total
        JOptionPane.showMessageDialog(null, "O total da
linha "
            + (i) + " e: " + total);

        // zera para somar próxima linha
        total = 0;
    }

    // realiza a somatória das colunas
    for(int j=0; j<COL; j++)
    {
        for(int i=0; i<LIN; i++)
        {
            total += matriz[i][j];
        }
        // imprime o total
        JOptionPane.showMessageDialog(null, "O total da
coluna "
            + (j) + " e: " + total);

        // zera para somar próxima coluna
        total = 0;
    }
}

```

O comando para criarmos uma matriz é semelhante ao de criarmos vetores:

```
double matriz[][] = new double[LIN][COL];
```

Precisamos definir as dimensões [][] e depois definir o tamanho de cada uma, utilizamos as constants previamente definidas.

Logo após a criação, realizamos a leitura de todos os valores da matriz. Para isso iremos utilizar dois for aninhados, um para controlar o índice da linha e o segundo, interno, para controlar o índice das colunas:

```
for(int i=0; i < matriz.length; i++)
{

```

```

        for(int j=0; j<matriz[i].length; j++)
        {
            matriz[i][j] = Double.parseDouble(
                JOptionPane.showInputDialog("Digite o
valor "
                + "["+(i)+""]"+"["+(j)+""]"));
        }
    }

```

Em Java podemos utilizar a propriedade length de um array, assim podemos criar o laço de repetição com mais segurança. Isso também é muito útil, pois também é possível criarmos arrays onde cada linha possui um tamanho diferente de colunas.

Um das principais operações realizadas em matriz é realizar sua leitura por linhas ou por colunas, o trecho seguinte realiza a soma de todos os valores por linha:

```

// realiza a somatória das linhas
for(int i=0; i < matriz.length; i++)
{
    for(int j=0; j<matriz[i].length; j++)
    {
        total += matriz[i][j];
    }
    // imprime o total
    JOptionPane.showMessageDialog(null, "O total da
linha "
        + (i) + " e: " + total);

    // zera para somar próxima linha
    total = 0;
}

```

O trecho seguinte realiza a soma por colunas, note que invertemos laços para que o externo controle o índice da coluna e o interno o índice da linha:

```

// realiza a somatória das colunas
for(int j=0; j<COL; j++)
{
    for(int i=0; i<LIN; i++)
    {
        total += matriz[i][j];
    }
    // imprime o total
}

```

```
JOptionPane.showMessageDialog(null,"O total da  
coluna "  
+ (j) + " e: " + total);  
  
// zera para somar próxima coluna  
total = 0;  
}
```

