ETE
ESCOLA TÉCNICA ESTADUAL
PORTO DIGITAL

GOVERNO DO ESTADO
Pernambuco
Secretaria de Educação

Desenvolvimento a Desktop

Pedro Albino

Tipos de Dados

- **int**: define variáveis numéricas, sem casas decimais ou valores fracionários;
- **double/float**: define variáveis numéricas, com casas decimais ou valores fracionários;
- **char**: armazena uma única letra ou caractere;
- String: armazena um texto;
- boolean: define variáveis, com valor TRUE ou FALSE.

Tipos de Dados

Tipo	Descrição	Exemplos
int	Conjunto de inteiros	10, 1500, -10,0 -1
double / float	Conjunto dos reais	0,10,-10, 2.5, -2.67
char	Caractere ou apenas uma letra	‘F’, ‘M’, ‘V’, ‘F’
String	Armazena um texto	“Olá, Mundo”
Boolean	Pode assumir apenas dois estados	TRUE, FALSE

O que é uma Variável ?

É uma entidade destinada a guardar uma informação.

Uma variável possui três atributos: **tipo de dado**, **nome** e a i**nformação**.

Variável

Não é possível definir variáveis de diferentes tipos com o mesmo identificador.

- Exemplo: **double A** e **inteiro A** - causaria erro na programação.

Letras maiúsculas e minúsculas são tratadas de forma diferente.

- Exemplo: **Media** é diferente de **media**, como também de **MEDIA**.

Variável

Regras para nomeação de variáveis:

- Devem ser iniciadas sempre por uma letra;
- Não devem conter caracteres especiais;
- Não devem conter espaços em branco;
- Não devem conter hífen entre os nomes (utilize underline).

Operadores

Precedência na matemática, da maior para menor.

Operação	Símbolo	Prioridade
Adição	+	1
Subtração	-	1
Multiplicação	*	2
Divisão	/	2
Resto da divisão inteira	%	2

Operadores Relacionais

São utilizados para comparar valores entre variáveis e expressões do mesmo tipo;

- O retorno desta comparação é sempre um valor do tipo booleano.

Igual **==**	**Maior** **>**	**Maior ou Igual** **>=**
Diferente **!=**	**Menor** **<**	**Menor ou Igual** **<=**

Estrutura condicional

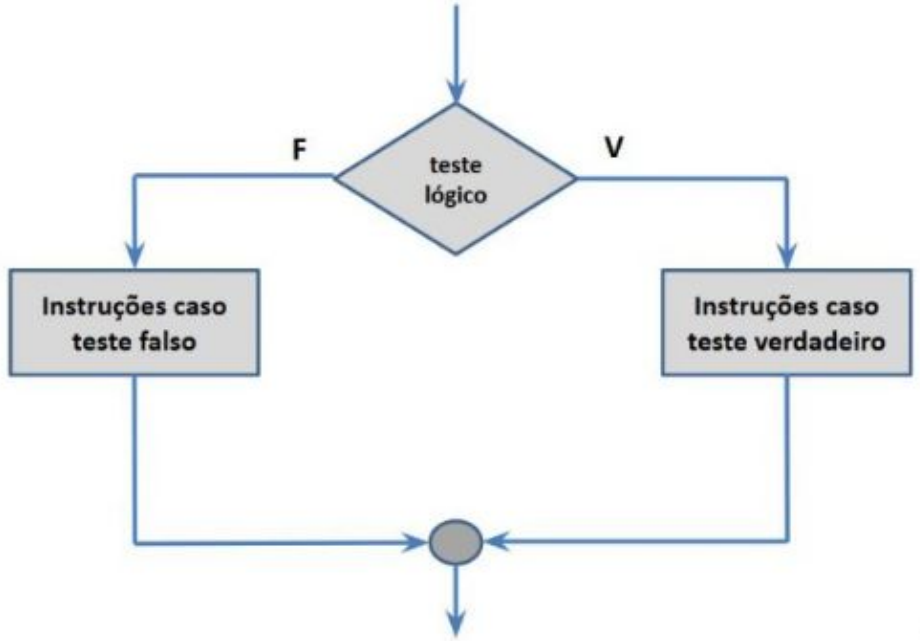
- É formulada com o uso dos operadores relacionais:
  - Igual ==
  - Diferente !=
  - Maior >
  - Menor <
  - Maior ou igual >=
  - Menor ou igual <=

Estrutura condicional simples - Sintaxe

1. Declare o tipo e inicialize a variável.
2. Faça uma condição para esta variável.
3. A sintaxe é if ( ) { } .

```java
public class minhaClasse {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int numero = 5;
        if(numero > 3) {
            System.out.println("Valor maior que 3");
        }

    }

}
```

Estrutura condicional simples - Sintaxe



Fábio dos Reis

Estrutura condicional simples - Sintaxe

- Uma condição é avaliada e, se o resultado for verdadeiro, um bloco de instruções é executado.

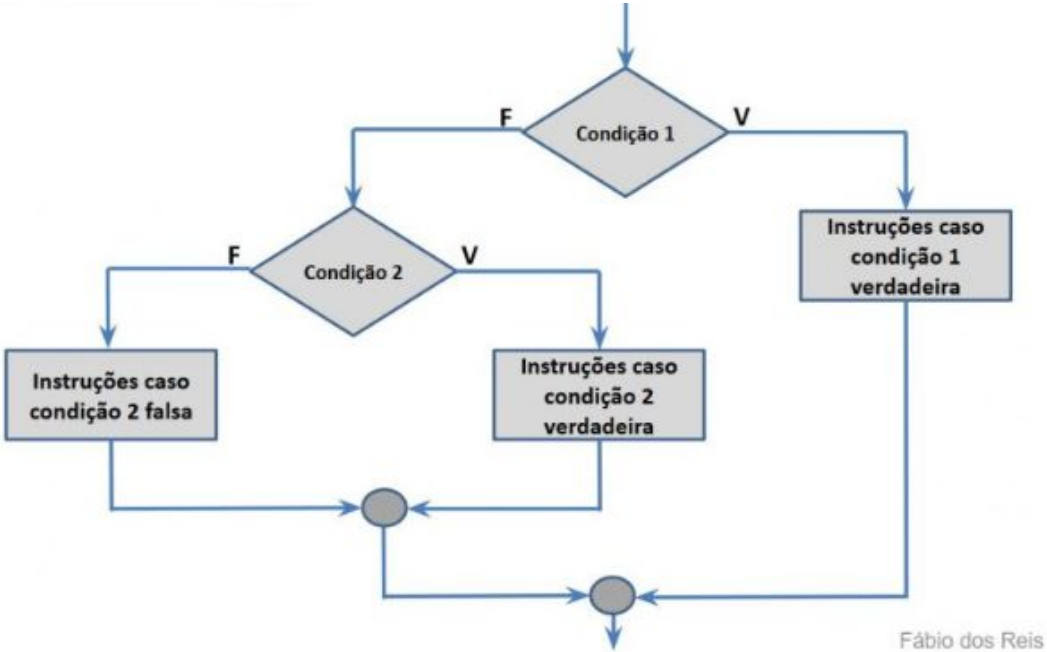
```java
public class minhaClasse {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int numero = 5;
        if(numero > 3) {
            System.out.println("Valor maior que 3");
        }

    }

}
```

Estrutura condicional simples - Sintaxe

- É usado quando é necessário verificar condições sucessivas onde uma ação será executada se um conjunto anterior de ações for satisfeito. Ou seja, permite executar múltiplos testes lógicos para decidir qual ação será tomada na sequência

```
if ( condição 1 ) {
    comando 1
    comando 2
}
else if ( condição 2 ) {
    comando 3
    comando 4
}
else if ( condição 3 ) {
    comando 5
    comando 6
}
else {
    comando 7
    comando 8
}
```

Estrutura condicional simples - Sintaxe



Estrutura condicional Encadeado - Sintaxe

```
public class Encadeado {

    public static void main(String[] args) {
        if(condicao 1) {
            //PROCESSAMENTO e SAÍDA
        }else if(condicao 2){
            //PROCESSAMENTO e SAÍDA
        }else {
            //PROCESSAMENTO e SAÍDA
        }
    }
}
```

Switch - Case

- Quando se tem várias opções de fluxo a serem tratadas com base no valor de uma variável, ao invés de várias estruturas if-else encadeadas, alguns preferem utilizar a estrutura switch-case.

```
switch (key) {
case value:

    break;

default:
    break;
}
```

Switch - Case

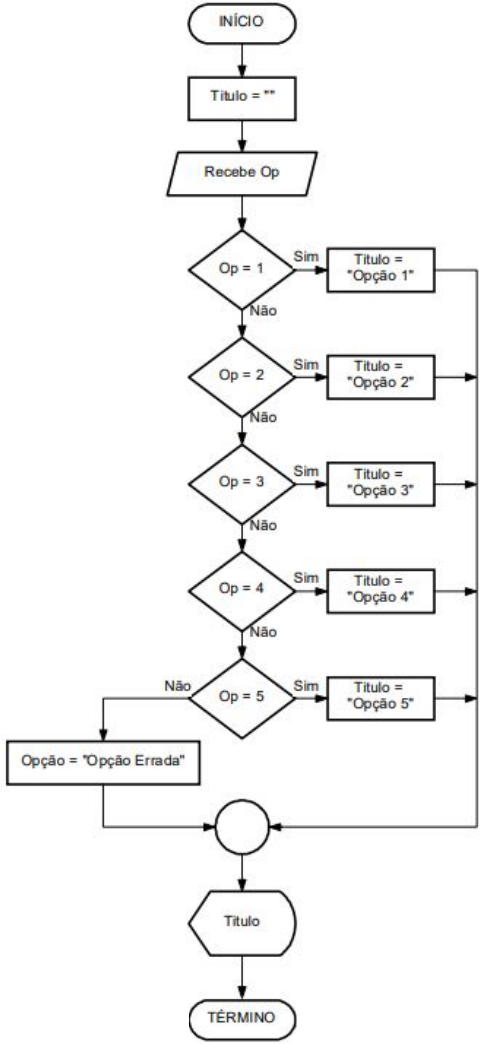
- Utiliza o mesmo raciocínio que o se e senao.
- Reduz o tempo de resposta e diminui a complexidade.
- Essa estrutura só aceita os tipos inteiros e carácter.
- Não utiliza operadores relacionais e/ou lógicos.

Switch - Case

```
public class caso {

    public static void main(String[] args) {
        int numero = 3;

        switch (numero) {
        case 1:
            System.out.println("O número foi 1 ");
            break;
        case 2:
            System.out.println("O número foi 2");
            break;
        default:
            System.out.println("Número não encontrado");
            break;
        }

    }

}
```

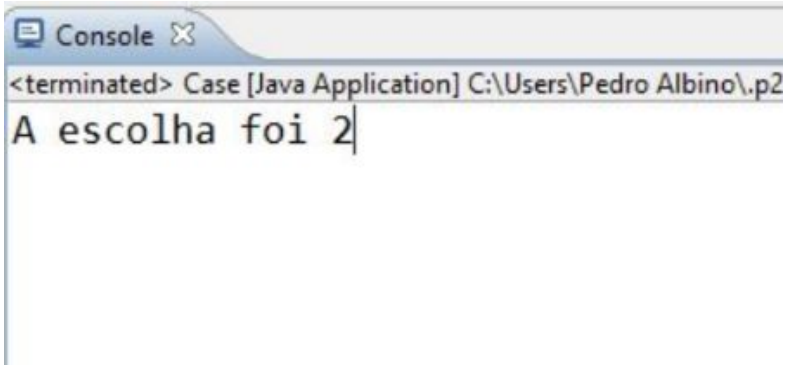
Switch - Case

- A sintaxe do switch é respectivamente o comando switch a condição a ser testada e entre chaves se coloca os casos.
- A sintaxe para se criar um caso é a palavra reservada case, o valor que a condição testada deve possuir dois pontos e suas instruções.
- Só podemos utilizar a igualdade, o que está implícita.
- Lembre-se de terminá-las com o comando break.


INÍCIO
Titulo = ""
Recebe Op
Op = 1
Sim
Titulo = "Opção 1"
Não
Op = 2
Sim
Titulo = "Opção 2"
Não
Op = 3
Sim
Titulo = "Opção 3"
Não
Op = 4
Sim
Titulo = "Opção 4"
Não
Op = 5
Sim
Titulo = "Opção 5"
Não
Opção = "Opção Errada"
Titulo
TÉRMINO

Switch - Case

```java
public class Case {

    public static void main(String[] args) {
        int valor = 2;

        switch (valor) {
        case 1:
            System.out.println("A escolha foi 1");
            break;
        case 2:
            System.out.println("A escolha foi 2");
            break;
        case 3:
            System.out.println("A escolha foi 3");
            break;
        default:
            System.out.println("Valor inválido");
            break;
        }

    }

}
```



Switch - Case

Você deseja criar um algoritmo para uma calculadora, o usuário informar primeiro o número, a operação que deseja executar e o segundo número. Dependendo do que o usuário informar como operador, o algoritmo executa um cálculo diferente (soma, subtração, multiplicação ou divisão).

Switch - Case

```java
public class Case {

    public static void main(String[] args) {
        char operacao = '-';

        double valor1 = 52;
        double valor2 = 100;
        double resultado;

        switch (operacao) {
        case '+':
            resultado = valor1 + valor2;
            System.out.println(resultado);
            break;
        case '-':
            resultado = valor1 - valor2;
            System.out.println(resultado);
            break;
        case '*':
            resultado = valor1 * valor2;
            System.out.println(resultado);
            break;
        case '/':
            resultado = valor1 / valor2;
            System.out.println(resultado);
            break;
        default:
            System.out.println("Operação inválida");
            break;
        }
        }

}
```

Estruturas repetitivas (While)

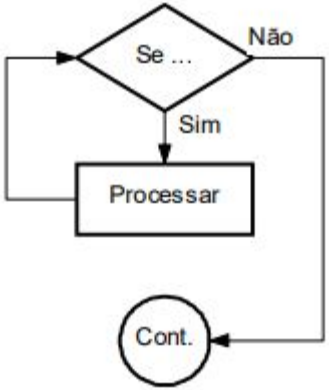
Estruturas repetitivas é um pouco mais difícil, então é necessário mais prática.

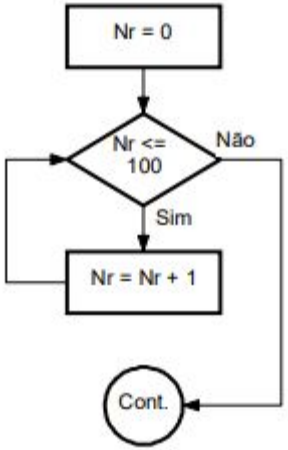
Estruturas repetitivas (While)

Neste caso, o bloco de operações será executado enquanto a condição x for verdadeira. O teste da condição será sempre realizado antes de qualquer operação.

Enquanto a condição for verdadeira o processo se repete. Podemos utilizar essa estrutura para trabalharmos com contadores.

Estruturas repetitivas (While)





Estrutura repetitiva (while)

SINTAXE / REGRA

```
while ( condição ) {
    comando 1
    comando 2
}
```

REGRA:

V: executa e volta
F: pula fora

Estrutura repetitiva (while)

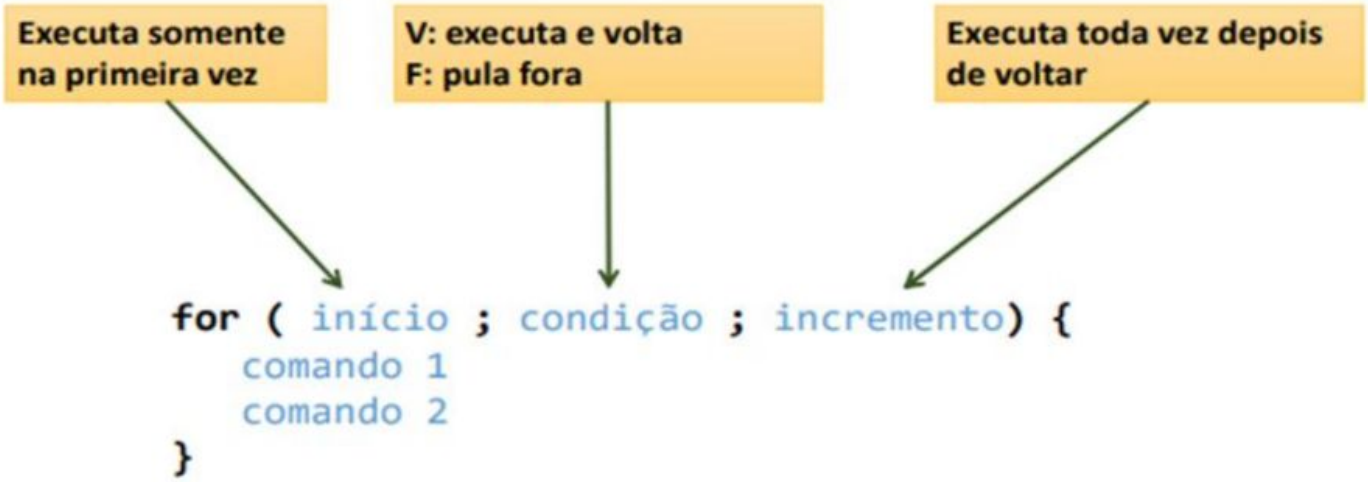
```
public class Exercicio {

    public static void main(String[] args) {

        int x = 0;
        int y= 4;

        while(x < 3 ) {
            y = y + 2;
            x = x + 1;
            System.out.println(x + " - " + y);
        }

    }
}
```

Estruturas repetitivas - for

É uma **estrutura de controle** que **repete** um bloco de comandos **para** um certo **intervalo de valores.**

Quando usar: quando se sabe previamente a **quantidade de repetições**, ou o intervalo de valores.

Estruturas repetitivas - for



Estruturas repetitivas - for

Perceba que a estrutura "para" é ótima para se fazer uma repetição baseada em uma **CONTAGEM**:

```
2
3  public class For {
4
5      public static void main(String[] args) {
6          // TODO Auto-generated method stub
7
8          for (int i=0; i<5; i++) {
9
10             System.out.println("Valor de i: " + i);
11
12         }
13     }
14
15 }
```

Console | Problems | Debug Shell

```
<terminated> For [Java Application] C:\Program Files\Java\jdk-11.0.11\bin\j
Valor de i: 0
Valor de i: 1
Valor de i: 2
Valor de i: 3
Valor de i: 4
```

Estruturas repetitivas - for

f.java

```
1
2  public class f {
3
4      public static void main(String[] args) {
5          // TODO Auto-generated method stub
6          int x = 3;
7          int y = 0;
8
9          for(int i=0; i<x; i++) {
10             System.out.print(i + ",");
11             y = y + 5;
12             System.out.println(y);
13         }
14
15     }
16
17 }
```

Console

<terminated> f [Java Application] C:\Users\Pedro Albino\.p2\pool\plugins\org.eclipse.justj.o

```
0,5
1,10
2,15
```

Exceções

Ao executar o código Java, podem ocorrer diferentes erros: erros de codificação cometidos pelo programador, erros devido a uma entrada incorreta ou outros imprevisíveis.

Quando ocorre um erro, o Java normalmente para e gera uma mensagem de erro. O termo técnico para isso é: Java lançará uma exceção (lançará um erro).

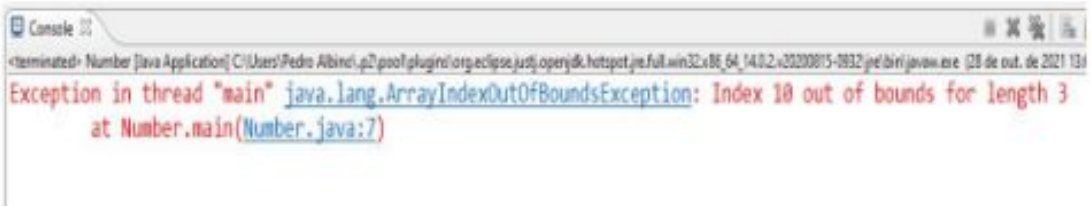
Exceções

A **try** instrução permite definir um bloco de código a ser testado quanto a erros enquanto está sendo
executado.
A **catch** instrução permite definir um bloco de código a ser executado, se ocorrer um erro no bloco try.
As palavras **try- catch** chave e vêm em pares:

```
try {
// Bloco de código
}
catch(Exception e) {
// Bloco de código para lidar com erros
}
```

Exceções

```java
public class Number {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int[] myNumbers = {1, 2, 3};
        System.out.println(myNumbers[10]);
    }

}
```



Exceções

Cath.java

```
1
2  public class Cath {
3
4      public static void main(String[] args) {
5          // TODO Auto-generated method stub
6          try {
7              int [] meuNumero = {1,2,3};
8              System.out.println(meuNumero[10]);
9          } catch (Exception e) {
10             System.out.println("Tamanho do array não encontrado, reveja novamente");
11         }
12     }
13
14 }
```

Console

<terminated> Cath [Java Application] C:\Users\Pedro Albino\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_14.0.2.v20200815-0932\jre\bin\javaw.exe (

```
Tamanho do array não encontrado, reveja novamente
```

Exercícios

A Secretaria de Meio Ambiente que controla o índice de poluição mantém 3 grupos de indústrias que são altamente poluentes do meio ambiente. O índice de poluição aceitável varia de 0,05 até 0,25. Se o índice sobe para 0,3 as indústrias do 1º grupo são intimadas a suspenderem suas atividades, se o índice crescer para 0,4 as indústrias do 1º e 2º grupo são intimadas a suspenderem suas atividades, se o índice atingir 0,5 todos os grupos devem ser notificados a paralisarem suas atividades. Faça um algoritmo que leia o índice de poluição medido e o grupo da indústria e emita a notificação adequada aos diferentes grupos de empresas.

Exercícios

Elabore um algoritmo que gera e escreve os números ímpares dos números lidos entre 100 e 200.

Exercícios

As maçãs custam R$ 1,30 cada se forem compradas menos de uma dúzia, e R$ 1,00 se forem compradas pelo menos 12. Escreva um programa que leia o número de maçãs compradas, calcule e escreva o custo total da compra.

Exercícios

Faça um algoritmo para ler: número da conta do cliente, saldo, débito e crédito. Após, calcular e escrever o saldo atual (saldo atual = saldo - débito + crédito). Também testar se saldo atual for maior ou igual a zero escrever a mensagem 'Saldo Positivo', senão escrever a mensagem 'Saldo Negativo'.

Exercícios

Faça um algoritmo para ler um número que é um código de usuário. Caso este código seja diferente de um código armazenado internamente no algoritmo (igual a 1234) deve ser apresentada a mensagem ‘Usuario inválido!’. Caso o Código seja correto, deve ser lido outro valor que é a senha. Se esta senha estiver incorreta (a certa é 9999) deve ser mostrada a mensagem ‘senha incorreta’. Caso a senha esteja correta, deve ser mostrada a mensagem ‘Acesso permitido’.

Exercícios

Faça um algoritmo para ler: a descrição do produto (nome), a quantidade adquirida e o preço unitário. Calcular e escrever o total (total = quantidade adquirida * preço unitário), o desconto e o total a pagar (total a pagar = total - desconto), sabendo-se que:

- Se quantidade <= 5 o desconto será de 2%
- Se quantidade > 5 e quantidade <=10 o desconto será de 3%
- Se quantidade > 10 o desconto será de 5%

Exercícios

Faça um algoritmo para ler: a descrição do produto (nome), a quantidade adquirida e o preço unitário. Calcular e escrever o total (total = quantidade adquirida * preço unitário), o desconto e o total a pagar (total a pagar = total - desconto), sabendo-se que:

- Se quantidade <= 5 o desconto será de 2%
- Se quantidade > 5 e quantidade <=10 o desconto será de 3%
- Se quantidade > 10 o desconto será de 5%