



Curso de Java

PARTE 03

Estruturas da Linguagem

Classes Wrappers
Operadores Relacionais/Lógicos
Estruturas de Decisão
Estruturas de Repetição
Coleções: List e Map

por: Mário Sergio

mariosergio30@gmail.com

[linkedin.com/in/mario-sergio-a1125831](https://www.linkedin.com/in/mario-sergio-a1125831)

Classes Wrappers (java.lang)

As Classe envólucro (Wrappers) : *encapsulam tipos primitivos*

<i>Boolean</i>	<i>Byte</i>	<i>String</i>
<i>Short</i>	<i>Integer</i>	<i>Long</i>
<i>Double</i>	<i>Float</i>	

Métodos de conversão a partir de uma String

Para	Método	Exemplo de expressão
int	<i>parseInt</i>	<i>Integer.parseInt(s)</i>
long	<i>parseLong</i>	<i>Long.parseLong(s)</i>
float	<i>parseFloat</i>	<i>Float.parseFloat(s)</i>
double	<i>parseDouble</i>	<i>Double.parseDouble(s)</i>
boolean	<i>valueOf,</i> <i>boolean Value</i>	<i>Boolean.valueOf(s).booleanValue()</i>

Ex: `Double numero = Double.parseDouble("13.5");`

Todas essas Classes possuem o método **toString()** para realizar a operação inversa, entre outros métodos úteis.

Lembre-se: F1 – para ajuda on-line

Métodos da Classe String

Principais métodos da classe **String**

Exemplos:

```
texto.equals("Accenture");  
texto.equalsIgnoreCase("accenture");  
texto.trim();  
texto.toUpperCase();  
texto.toLowerCase();  
texto.contains("Accenture");  
texto.replace("Brasil", "Japão");  
texto.substring(5, 8);  
texto.length();  
texto.charAt(0);  
texto.indexOf("t");
```

Todos esses métodos retornam valores !

Quais são esses valores para a string abaixo ???

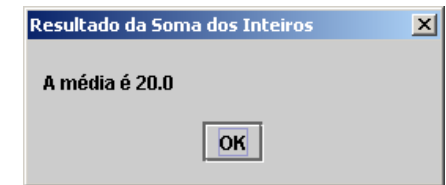
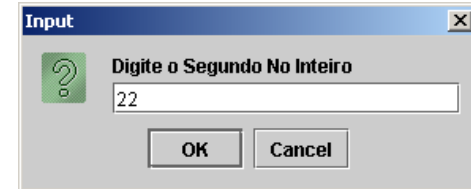
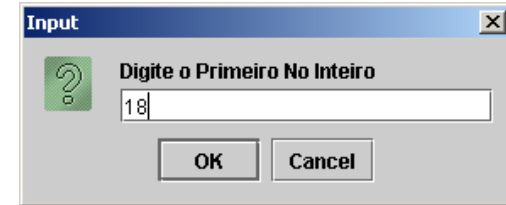
```
String texto = "Accenture do Brasil";
```

Saber mais:

<https://www.devmedia.com.br/java-string-manipulando-metodos-da-classe-string/29862>

Diálogos com javax.swing

```
// Meu Segundo Programa JAVA
// Trabalhando com Números e Operadores Aritméticos
// Baseado em Deitel & Deitel, 2003
// Pacote de extensão Java
import javax.swing.JOptionPane; // import class JOptionPane
public class Adicao {
    public static void main( String args[] )    {
        String primeiroNumero; // 1o string informado pelo usuário
        String segundoNumero;  // 2o string informado pelo usuário
        int numero1;           // primeiro operando da adição
        int numero2;           // segundo operando da adição
        int media;             // Resultado da Adição
        // ler o primeiro número (na forma string)
        primeiroNumero = JOptionPane.showInputDialog("Digite o Primeiro No Inteiro" );
        // ler o segundo número (na forma string)
        segundoNumero = JOptionPane.showInputDialog( "Digite o Segundo No Inteiro" );
        // convertendo os strings em números inteiros
        numero1 = Integer.parseInt(primeiroNumero);
        numero2 = Integer.parseInt(segundoNumero);
        // Somando os números
        media = (numero1 + numero2)/2;
        // Apresentando os resultados
        JOptionPane.showMessageDialog(null, "A media é "+media,"Resultado da media: ",
            JOptionPane.PLAIN_MESSAGE);
    } // fim do método main()
} // fim da classe Adicao
```



Diálogos com **javax.swing**

O Pacote swing possui uma biblioteca de Classes para **interface gráfica com o usuário GUI**, por enquanto utilizaremos apenas a classe ***JOptionPane*** que oferece caixas de diálogo gráficas predefinidas que permitem aos programas exibir mensagens aos usuários:

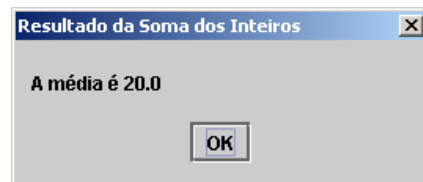
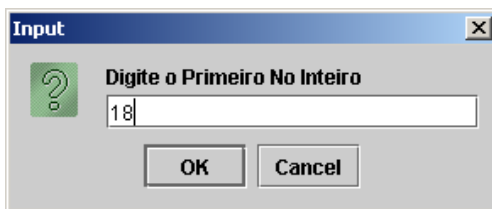
- O método ***showMessageDialog()*** *exibe um aviso com interface gráfica para o usuário.*
- O método ***showConfirmDialog*** *exibe uma pergunta ao usuário com botões OK, NÃO E CANCELAR, e retorna um código de resposta clicada pelo usuário.*
- O método ***showInputDialog()*** *combina a montagem da janela de edição com o *prompt* de digitação do string fornecido pelo usuário.*
- As variáveis informadas pelo ***showInputDialog()*** *aos programas Java são sempre Strings e devem ser convertidas com o uso das **classes wrappers** do pacote java.lang.*

Mãos à Obra



Com base no exercício anterior,
melhore seu “**Sistema Escolar**”,

Utilize *JOptionPane* para todas as ações
de *interface com usuário*.



Operadores de Igualdade e Relacionais

Operador algébrico de igualdade padrão ou operador relacional	Operador de igualdade ou relacional em Java	Exemplo de condição em Java	Significado da condição em Java
<i>Operadores de igualdade</i>			
=	==	x==y	x é igual a y
≠	!=	x!=y	x não é igual a y
<i>Operadores relacionais</i>			
>	>	x>y	x é maior que y
<	<	x<y	x é menor que y
=	>=	x>=y	x é maior que ou igual a y
=	<=	x<=y	x é menor que ou igual a y

- Todos os operadores relacionais têm o mesmo nível de precedência e associam da esquerda para a direita.
- São largamente utilizados nas **estruturas de controle de decisão e repetição**.
- Resultam SEMPRE em um **valor lógico**, que pode ser atribuído a uma variável do tipo booleano, ex.: **boolean** lAprovado = (media >= 6);

Estruturas de Controle Condicional

Também conhecidas como **estruturas de decisão, seleção** ou de **desvio de fluxo**

```
1  if (condition) {  
      
    } else {  
    }  
}
```

Seleção

Simples/Composta/Encadeada

```
2  condition ? true : false;
```

Immediate IF

```
3  switch (variable) {  
    case value:  
        break;  
    default:  
        break;  
}
```

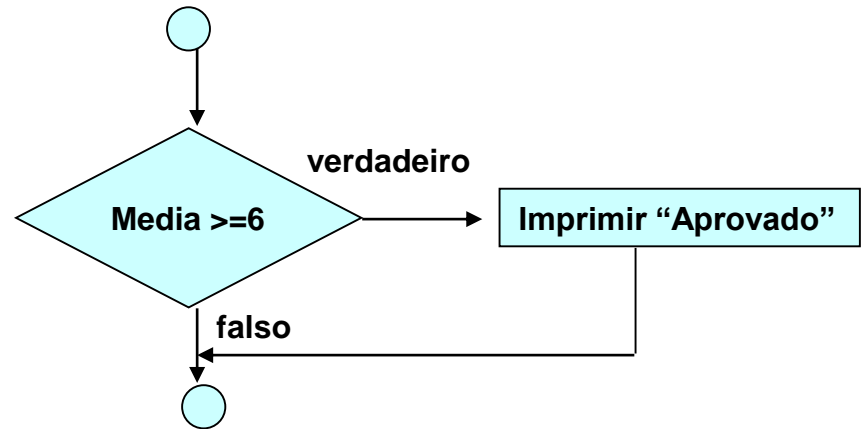
Seleção Múltipla



Estrutura Condicional: **if**

Sintaxe

```
if (condição)  
    comando Java;  
[ou {bloco de comandos Java;}]
```



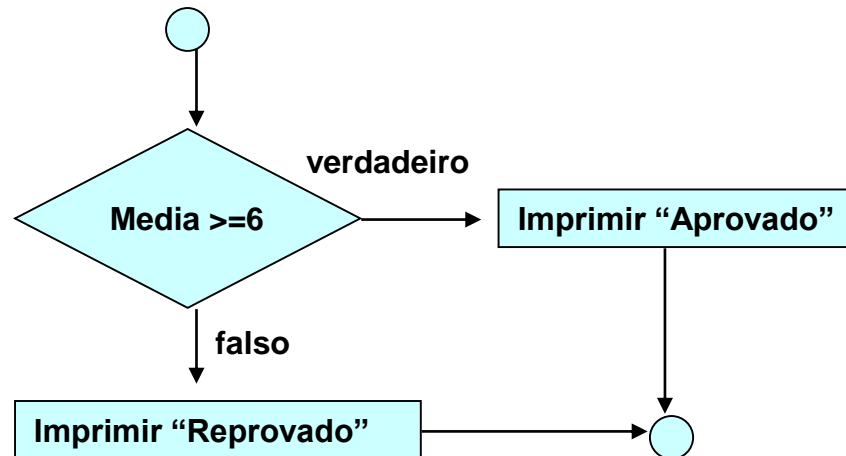
O Exemplo:

```
if (media >= 6)  
    System.out.println("Aprovado"); // Instrução de apenas 1 linha
```

```
if (media >= 6) {  
    System.out.print("Oi Fulano !");  
    System.out.println("Você foi Aprovado");  
} // fim do bloco if
```

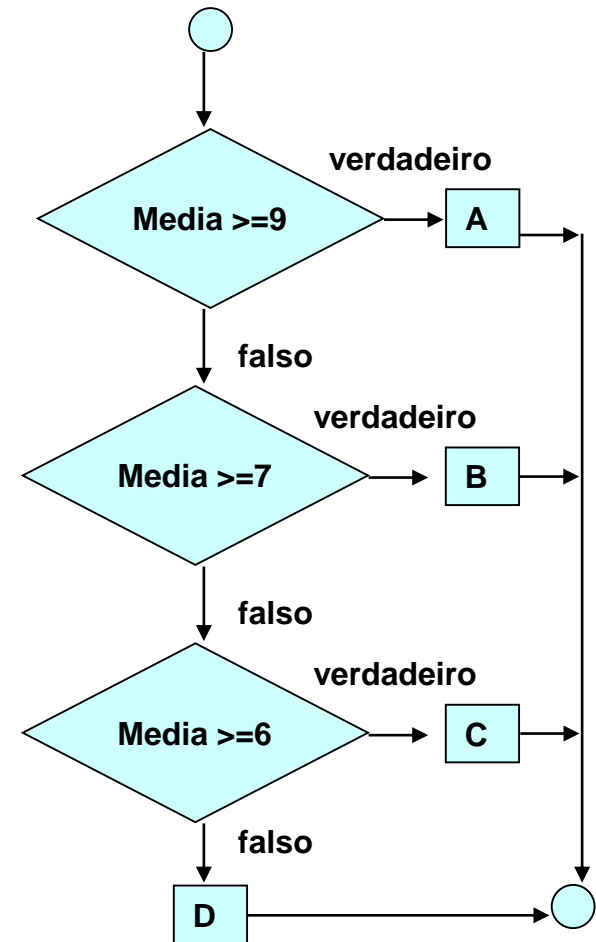
Estrutura Condicional: **if/else**

```
if (media >= 6){  
    System.out.print("O Aluno está");  
    System.out.println("Aprovado");  
} // fim do bloco if  
  
else {  
    System.out.print("O Aluno está");  
    System.out.println("Reprovado");  
} // fim do bloco else
```



Estrutura Condicional: **if/else** (encadeada)

```
if (media >= 9)
    System.out.print("O Conceito é A");
else if (media >= 7)
    System.out.print("O Conceito é B");
else if (media >= 6)
    System.out.print("O Conceito é C");
else
    System.out.print("O Conceito é D");
```



Mãos à Obra



Com base no exercício anterior, melhore seu “**Sistema Escolar**”, informando o resultado final do aluno, conforme os critérios:

Média:

maior ou igual a 7 - Aprovado

de 4 a 6.99999999.. - Recuperação (**não usar operador lógico**)

menor que 4 - Reprovado



Operadores de Lógicos

Operador	Ação
&&	And (E)
	Or (Ou)
!	Not (Não)

considerando:

```
int age;  
boolean aprovado;
```

OPERATORS	NAME	DESCRIPTION	EXAMPLE
&&	logical AND	It will return true when both conditions are true	If (age > 18 && age <= 25)
	logical OR	It will returns true when at-least one of the condition is true	If (age == 35 age < 60)
!	logical NOT	If the condition is true, logical NOT operator makes it false	if (!aprovado)

Operadores lógicos são largamente utilizados em conjunto com estruturas de controle condicionais.

Saber mais:

<https://www.tutorialgateway.org/java-logical-operators/>

Relembrando: Tabela Verdade

Lógica Proposicional: Tabela Verdade

Para quem tem
dúvidas...

A	B	A e B	A ou B	não A
F	F	F	F	V
F	V	F	V	V
V	F	F	V	F
V	V	V	V	F

Ex: temos as proposições A e B

A: Brasília é a ^Vcapital do Brasil **E** B: Lima é a ^Vcapital do Peru. \rightarrow ^V

A: Gato é um ^VAnimal **E** B: Peixe é um ^Fmamífero. \rightarrow ^F

A: Recife é um ^Fpaís **OU** B: Olinda é ^Vuma cidade \rightarrow ^V

NÃO A: O ^VJapão fica na Ásia \rightarrow ^F

Saber mais:

<https://www.tutorialgateway.org/java-logical-operators/>

Mãos à Obra



Com base no exercício anterior,
melhore seu “**Sistema Escolar**”,
adicionando os critérios:

Quantidade de Faltas e trabalho noturno:

Só é permitido ficar em Recuperação se a quantidade de faltas for menor que 10.

Só é permitido ser aprovado com nota 7 se o aluno tiver menos que 3 faltas ou comprovar que trabalha a noite. Caso contrário, só será aprovado direto com média a partir de 9.



Precedência de operadores

3º: conector ||

16

Estrutura Condicional: Immediate-IF

O operador Ternário **?**: retorna um valor dependendo do resultado da condição

Sintaxe:

(condição) **?** {valor para condição **true**} : {valor para condição **false**}

Exemplos:

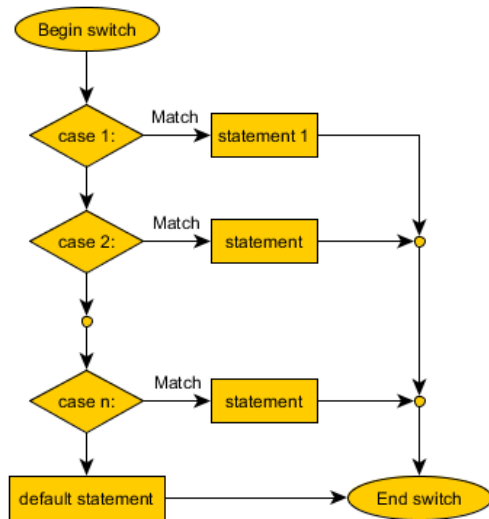
```
resultado = (media >= 6) ? "Aprovado" : "Reprovado";  
System.out.println((media >= 6) ? "Aprovado" : "Reprovado");
```

Equivalente a:

```
if (media >= 6)  
    resultado = "Aprovado";  
else  
    resultado = "Reprovado";
```

Estrutura Condicional: `switch` / `case`

Também conhecido como estrutura de seleção múltipla



```
System.out.println("Qual operacao deseja realizar ?");
System.out.println("Opções: 1-Soma, 2-Subtração, 3-Multiplicacao, 4-Divisão");
int operacao = teclado.nextInt();

switch (operacao) {
case 1:
    resultado = n1 + n2;
    break;
case 2:
    resultado = n1 - n2;
    break;
case 3:
    resultado = n1 * n2;
    break;
case 4:
    resultado = n1 / n2;
    break;
default: ← É OPCIONAL
    System.out.println("Operação inválida");
}

System.out.println("Resultado: " + resultado);
```

- O valor a ser testado (condicional) aparece apenas uma vez, diferentemente do que ocorreria com **múltiplos if else**
- Utilize o comando **break** para finalizar a execução do **switch** quando um dos **case's** for atendido, caso contrário, todos os blocos **case** abaixo serão também executados.

Mãos à Obra



Baseado no exemplo anterior, crie um novo programa: “**Calculadora**”

Utilizando a estrutura de **seleção múltipla**

```
switch (key) {  
  case value:  
  
    break;  
  
  default:  
    break;  
}
```



Estrutura Condicional: `switch` / `case`

Um motivo para não usar `break`

```
System.out.println("Versão do seu carro: " + versao);
System.out.println("Itens opcionais disponíveis:");
System.out.println("");

switch (versao.toUpperCase()) {
case "PREMIUM":
    System.out.println("Bancos em Couro");
    System.out.println("Pneus de Liga Leve");
case "ADVANCED":
    System.out.println("Ar Condicionado");
    System.out.println("Direção elétrica");
    System.out.println("Pintura metálica");
case "BASIC":
    System.out.println("Tapetes");
    System.out.println("Porta copos");
    break;
default:
    System.out.println("Versão inexistente");
}
```

Estruturas de Controle de Repetição

Executa trechos de código repetidamente (loops)

Principais Tipos de estruturas de repetição:

- Repetição condicional (indefinida)

```
while (condition) {
```

← Com teste lógico no início

```
}
```

```
do {
```

```
} while (condition);
```

← Com teste lógico no final (executa ao menos 1 vez)



- Repetição contada (definida) Número pré-determinado

```
for (int i = 0; condition over i; i++) {
```

← Iteração contada

```
}
```

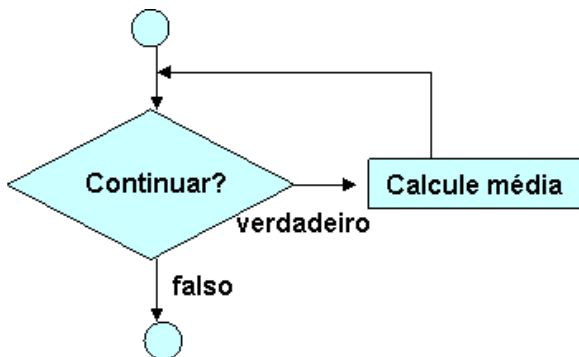
```
for (String item : list) {
```

← Iteração sobre elementos de uma lista considerando: List<String> **list**;

```
}
```

Estrutura de Repetição: **while** e **do/while**

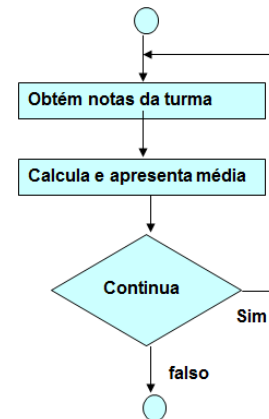
- Essas estruturas de repetição permitem especificar uma ação ou um bloco de ações que devem permanecer sendo repetidas **enquanto determinada condição for verdadeira**.
- O corpo da estrutura pode ser uma **instrução única ou um bloco de comandos**.
- Quando a condição do comando **while** ou **do/while** se tornar falsa, a ação (ou bloco) do comando será pulada. O programa continuará com a ação imediatamente após o comando **while**.
- **IMPORTANTE:** você deve **sempre prever** a ação que **tornará falsa** a condição do comando **while**. **Caso contrário seu programa entrará em loop infinito.**



```
boolean lTemVida = true;
```

```
while (lTemVida) {  
    System.out.println("Há Esperança !");  
}
```

Pode **nunca** ser executado



```
int nQtdvidas = 7;
```

```
do {  
    System.out.println("Há Esperança !");  
} while (nQtdvidas > 0);
```

Será executado ao **menos 1 vez**

Nesses dois exemplos temos problema de **looping infinito**, pois o valor da **VARIÁVEL DE CONTROLE** não está sendo **alterado** dentro do escopo da estrutura.

Mãos à Obra



Com base no exercício anterior,
melhore seu “**Sistema Escolar**”:

Após a exibição do resultado da média do primeiro aluno,
pergunte ao usuário se ele deseja digitar
dados para um novo aluno.

Obs: por enquanto não se preocupe
em armazenar o histórico de todos
alunos/médias



Estrutura de Repetição: **for**

- A estrutura de repetição **for** permite repetir uma ação ou um bloco de ações em **um número definido de vezes**.
- O comando **for** possui três seções:

```
for (inicializadores; condição de continuação; incremento)  
{  
    ação ou bloco de ações no comando;  
}
```

Chamada de **Variável de Controle**

```
for (int x = 0; x < 1000; x++) {  
    System.out.println("Repetição de numero: " + x);  
}
```

Será executada **EXATAMENTE** 1000 vezes

Mãos à Obra



Com base no exercício anterior,
melhore seu “**Sistema Escolar**”:

Precisamos obter mais notas para compor a média, não apenas duas.
Assim, antes de digitar as notas
o usuário precisará informar
quantas notas ele pretende digitar
para cálculo da média do aluno.

Obs: por enquanto não se preocupe
em armazenar o histórico de todos
alunos/médias




Coleções: Listas (java.util.List)

- Uma **Lista** representa uma **coleção de objetos**, sendo mais flexível que os arrays pois podem **variar de tamanho dinamicamente**.
 - Permitem elementos duplicados
 - Permitem valores nulos
 - Mantém uma ordenação posicional (por índice)

Exemplos:

```
List<String> listaEstados = new ArrayList<String>();  
listaEstados.add("São Paulo");  
listaEstados.add("Rio de Janeiro");  
listaEstados.add("Minas Gerais");
```



```
System.out.println("Segundo elemento da lista :" + listaEstados.get(1));
```

Saber mais:

<https://www.caelum.com.br/apostila-java-orientacao-objetos/collections-framework/>

Coleções: Listas (java.util.List)

Métodos da classe **List**:



add – Adiciona um item no final da coleção

get – retorna (pega) um item em determinada posição

set – altera o valor de um item em uma determinada posição

size – retorna o tamanho atual da coleção

clone – Duplica a ArrayList

contains – busca um valor no array, e retorna true, se o elemento estiver no array;

indexOf – busca um valor no array, mas retorna o índice do elemento encontrado;

lastIndexOf – o mesmo que indexOf mas retorna o último elemento encontrado;

remove – Remove um item da coleção.

Estrutura de Repetição: **for** (com listas)

A estrutura de repetição **for** também permite **realizar iterações sobre elementos de uma lista** de uma forma mais direta, sem a necessidade de uma variável de controle (contador).

```
List<String> list;  
  
for (String item : list) {  
    System.out.println("Item da lista: " + item);  
}
```

Para cada iteração, o próximo item da lista é colocado na variável **item**, seguindo a ordem que o item ocupa na lista (índice).

Exemplo:

```
List<String> listFrutas = new ArrayList<String>();  
  
listFrutas.add("Morango");  
listFrutas.add("Abacaxi");  
listFrutas.add("Uva");  
  
System.out.println("<<<< Lista de Frutas >>>>>>");  
  
for (String umaFruta : listFrutas)  
    System.out.println("Fruta: " + umaFruta);
```

Mãos à Obra



Com base no exercício anterior,
melhore seu “**Sistema Escolar**”,

1- *Guardar o histórico dos alunos:
nomes, médias, idades, e escolaridade
dos alunos, em quatro objetos do tipo List.*

2- *Mostrar relatório com nomes, médias, idades,
e escolaridade de TODOS os alunos
no final da execução do programa.*

Obs: A escolaridade tem como dominio:
FUNDAMENTAL
MEDIO
SUPERIOR

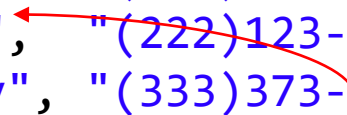


Coleções: Mapas (java.util.Map)

- Um **Map** representa uma **coleção de objetos**, numa estrutura (key,value). Também conhecida como um PAR (chave,valor). Assim como as listas, podem **variar de tamanho dinamicamente**.
 - Não permitem elementos duplicados
 - Permitem apenas um valor nulo
 - Não mantem uma ordenação posicional
 - Encontra um valor a partir de sua chave

Exemplos:


```
Map<String,String> mapContatos = new HashMap<>();  
mapContatos.put("Tom", "(111)123-4567");  
mapContatos.put("Dick", "(222)123-7890");  
mapContatos.put("Harry", "(333)373-3703");
```



```
String fone = mapContatos.get("Dick");
```

Valor retornado:

(222)123-7890



Coleções: Mapas (java.util.Map)

Iteração sobre todos os elementos de um Map:

```
for (Map.Entry<String,String> umContato : mapContatos.entrySet()) {  
  
    System.out.println("Chave: " + umContato.getKey());  
    System.out.println("Valor: " + umContato.getValue());  
    System.out.println("-----");  
}
```

Busca por uma chave em um Map:

```
if (mapContatos.containsKey("Harry"))  
    System.out.println("Telefone:" + mapContatos.get("Harry"));
```

saída

Telefone: (333)373-3703

Mãos à Obra



Com base no exercício anterior,
melhore seu “**Sistema Escolar**”,

1- Precisamos incluir permissões de acesso para usar o Sistema escolar. Desejamos que os funcionarios da escola Realizem login, utilizando usuario e senha antes de iniciar o cadastro dos alunos.

***Dica:** Utilize um Map pré-cadastrado (hard-code) para cadastrar os acessos:*

Ex: Chave/valor -> Usuario/senha

PROFESSOR->25413

DIRETOR->felicidade

SECRETARIA->amor1000

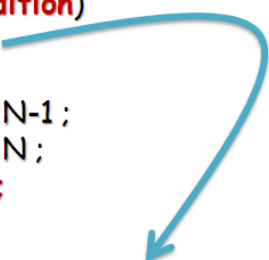
COORDENADOR->2018




Instruções: **break** e **continue**

- As instruções **break** e **continue** modificam o comportamento das estruturas de repetição **while**, **for**, **do/while** ou **switch**.
- A instrução **break** interrompe o laço (no caso das estruturas de repetição) e impede a execução de outros casos de um comando **switch**.
- A instrução **continue** permite o salto do conjunto de operações, com retorno à expressão condicional do laço, reiniciando o mesmo (portanto, ao contrário do **break**, não interrompe o laço).

```
Initialization;  
while (condition)  
{  
    Statement 1;  
    Statement 2;  
    Statement 3;  
    .....  
    .....  
    if ( If Condition)  
        break;  
  
    Statement N-1;  
    Statement N;  
    Increment;  
}  
OutsideStatement 1;
```

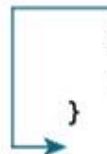


```
while (isOk)  
{  
    ...  
    if (aCondition)  
        continue;  
    ...  
}
```

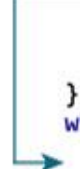


Instruções: **break** e **continue**


```
while (testExpression) {  
    // codes  
    if (condition to break) {  
        break;  
    }  
    // codes  
}
```




```
do {  
    // codes  
    if (condition to break) {  
        break;  
    }  
    // codes  
} while (testExpression);
```




```
for (init; testExpression; update) {  
    // codes  
    if (condition to break) {  
        break;  
    }  
    // codes  
}
```



```
while (testExpression) {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
}
```



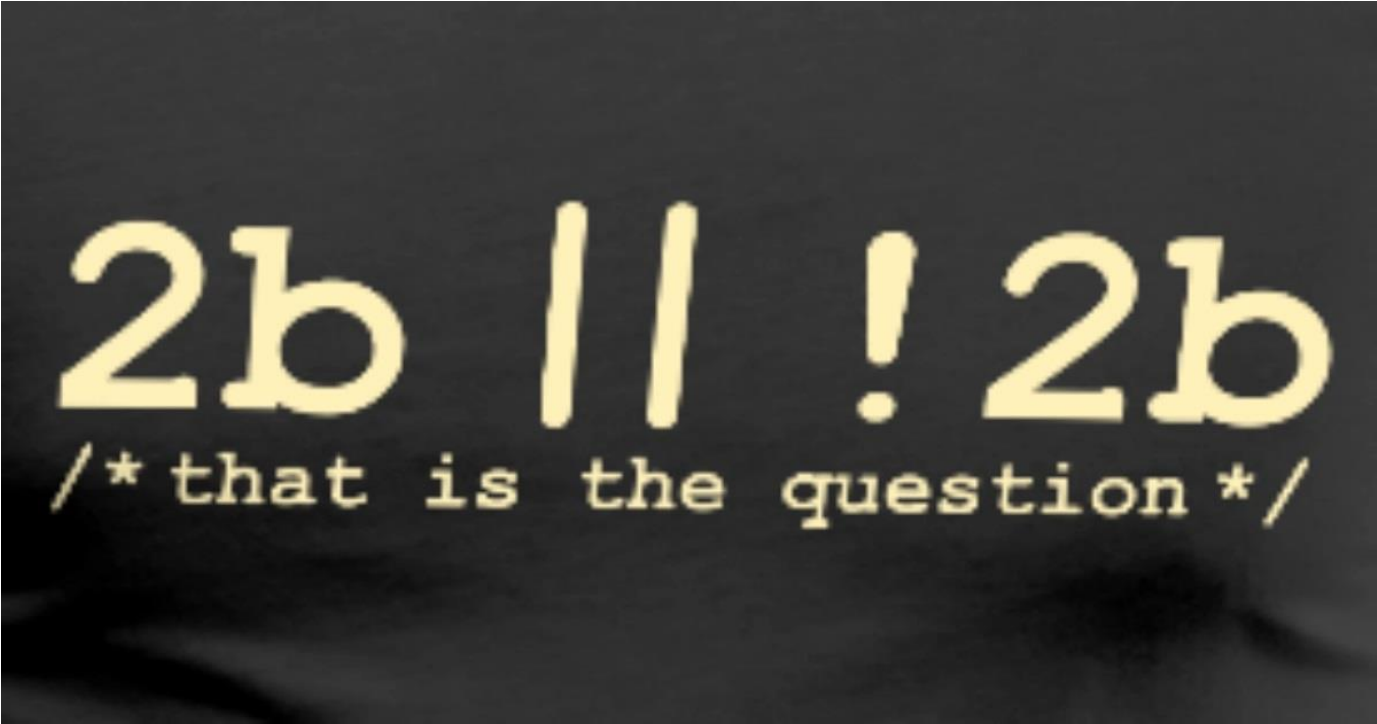
```
do {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
} while (testExpression);
```



```
for (init; testExpression; update) {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
}
```



Obrigado !



2b || ! 2b
/*that is the question*/

Referências

Programação de computadores
em Java

Rui Rossi dos Santos

Java 8: Programação de
Computadores - Guia Prático de
Introdução, Orientação e
Desenvolvimento - José Augusto
N. G. Manzano

Slides do Prof. Roberto Pacheco
INE – CTC – UFSC