

Java Básico

Igor Ebrahim (ies@cin.ufpe.br)



Módulo 4

Referências, Strings e Arrays

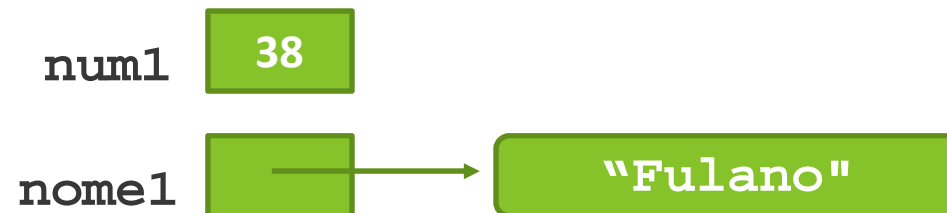
+ Tipos

Tipos
primitivos
int, float, ...

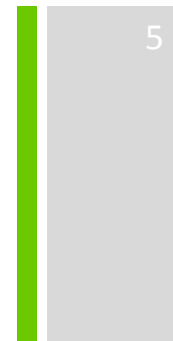
Tipos de
referência
Classes,
interfaces, ...

+ Referências

- Variável primitiva guarda seu valor;
- Variável de objeto guarda um endereço para ele;
- Uma referência pode ser imaginada como um apontador para o local onde está armazenado o objeto;



+ Referências



■ O que acontece?

Antes:

num1	38
num2	96

```
num2 = num1;
```

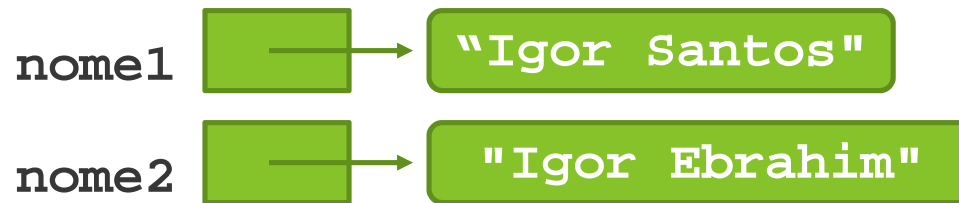
Depois:

num1	38
num2	38

+ Referências

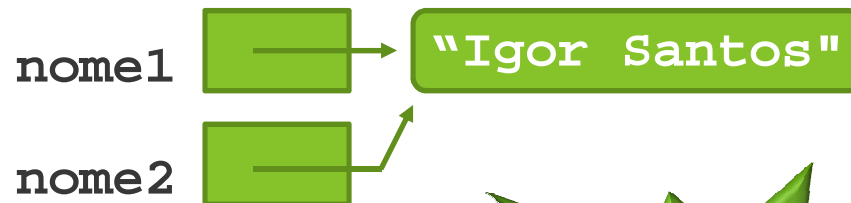
■ O que acontece?

Antes:



```
nome2 = nome1;
```

Depois:



Aliasing

+ Referências

- Em Java não se trabalha diretamente com os objetos, mas com as referências para os objetos;
- Isso altera a maneira como copiamos e comparamos os objetos;

+ Strings

- Seqüência de caracteres;
- Em Java, String não é tipo primitivo;
- É tipo de referência → são objetos!

```
String str = "Olá Mundo!";
```

```
String str = new String("Olá Mundo!");
```


+ Igualdade de Strings

- O operador `==` não deve ser usado para compararmos se duas Strings são iguais;
- Deve-se usar o método `equals` da classe `String`



```
str1 == str2
```

```
str1.equals(str2)
```

+ Strings

- `boolean equals(Object o)`

```
String x = "Igor";  
String y = "Ebrahim";  
boolean b = x.equals(y); // b == false
```

+ Strings

- `int indexOf(String s):`
 - Retorna o índice do primeiro encontrado;
 - Índices em Java começam em 0;
 - Caso não encontre, retorna -1;

```
String x = "Clique para adicionar anotações";  
String y = "adicionar";  
int i = x.indexOf(y); // i == 12
```

+ Strings

- `String substring(int inicio, int fim):`
 - Retorna a substring do índice inicio (inclusivo) até o índice fim (exclusivo);

```
String x = "Clique para adicionar anotações";  
String y = x.substring(2, 5);  
// y é "iqu"
```

+ Strings

- `char charAt(int indice):`
 - Character da posição índice;

```
String x = "Clique para adicionar anotações";  
char c = x.charAt(0); // c == 'C'
```

+ Strings

- `boolean equalsIgnoreCase(Object o)`
 - igualdade ignorando diferenças entre maiúsculas e minúsculas
- `int length()`
 - comprimento da strings; quantidade de caracteres

+ Prática

1. Refatore as classe Conta e Cliente, implementando os seus respectivos métodos equals.
 - Lembre que o método equals deve ter a seguinte assinatura: `public boolean equals(Object o);`
2. Utilize os métodos equals implementados na questão anterior para comparar duas instâncias de conta e duas instâncias de cliente.

+ Tipo Enumerável

- Adicionado na versão 5.0 de Java;
- Coleção de constantes que pertencem a uma única “ideia”;
- Em Java, todo tipo enum é uma classe;
- Todo enum herda implicitamente de `java.lang.Enum`:
 - Não pode herdar de mais nenhuma outra classe;
 - Veremos herança mais pra frente.

+ Tipo Enumerável

- Os elementos devem ser separados por vírgula

```
public enum TipoCliente {  
    NORMAL,  
    ESPECIAL,  
    OURO  
}
```

```
Cliente c = new Cliente(TipoCliente.OURO);
```

+ Tipo Enumerável

- Como todo enum é uma classe, podemos adicionar métodos e atributos a ele

```
public enum TipoCliente {  
    NORMAL        (0.1F),  
    ESPECIAL      (0.2F),  
    OURO          (0.3F)  
  
    private float taxa;  
  
    TipoCliente(float taxa) {  
        this.taxa = taxa;  
    }  
  
    public float getTaxa() {  
        return this.taxa;  
    }  
}
```

+ Prática

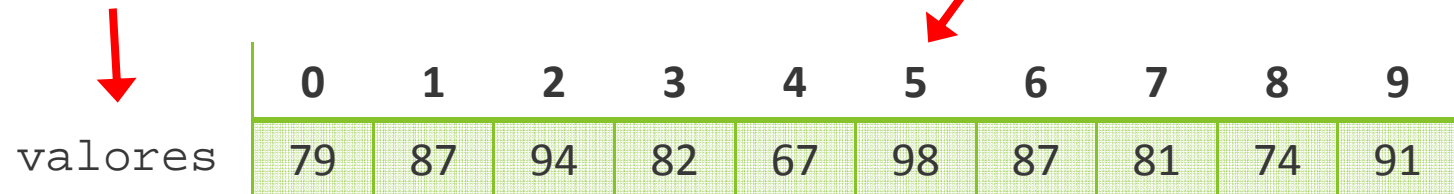
1. Implemente o enum TipoCliente. Este deve ser composto pelas constantes NORMAL, ESPECIAL e OURO.
2. Refatore a classe Cliente para que ela possua uma referência para um TipoCliente.

+ Arrays

- São listas ordenadas de valores que permitem armazenar uma grande quantidade de informação;

Nome do array
inteiro

Cada valor tem um índice numérico



	0	1	2	3	4	5	6	7	8	9
valores	79	87	94	82	67	98	87	81	74	91

Um array de tamanho N é indexado de 0 a N-1

Esse array guarda 10 valores indexados de 0 a 9

+ Arrays

- Também são Objetos → tipos de referência;
- Todos os elementos de um array são do mesmo tipo;
- Arrays têm tamanho fixo depois de criados;

+ Arrays

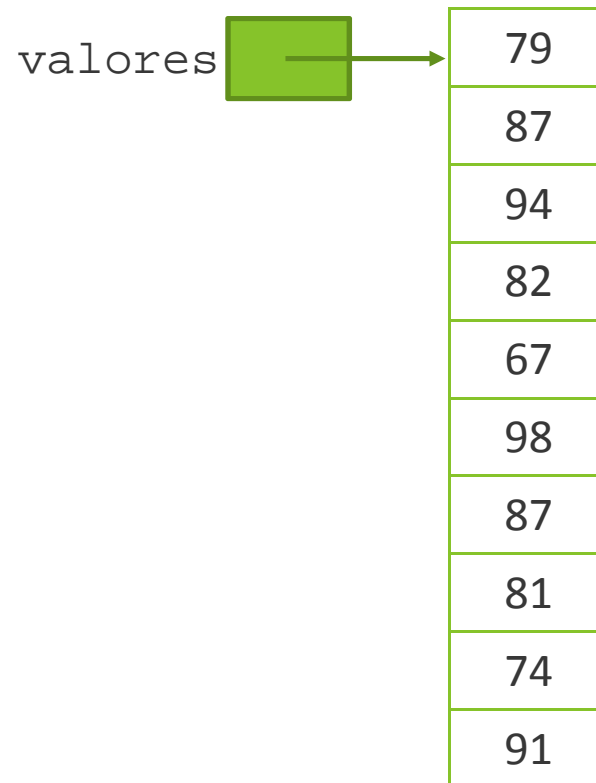
0	1	2	3	4	5	6	7	8	9
79	87	94	82	67	98	87	81	74	91

```
int i = valores[3]; // i == 82
```

- Para referenciar um determinado valor armazenado em um array, usa-se o nome do array seguido do índice entre colchetes
- Essa expressão representa um inteiro e pode ser usada em qualquer lugar que uma variável de tipo inteiro pudesse também

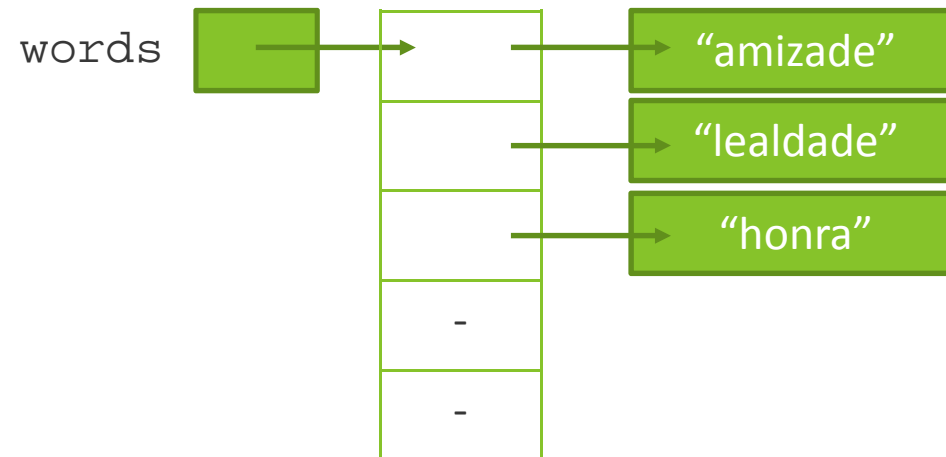
+ Arrays

■ Outra maneira de descrever um array



+ Arrays

■ E para tipos referência



+ Declarando Arrays

```
int[] valores = new int[10];
```

- O tipo da variável valores é int[] (um array de inteiros)
- A variável valores referencia um novo objeto array que pode armazenar 10 inteiros
- Outros exemplos:

```
float[] precos = new float[100];  
String[] nomes = new String[21];
```

+ Estruturas de Controle [2]

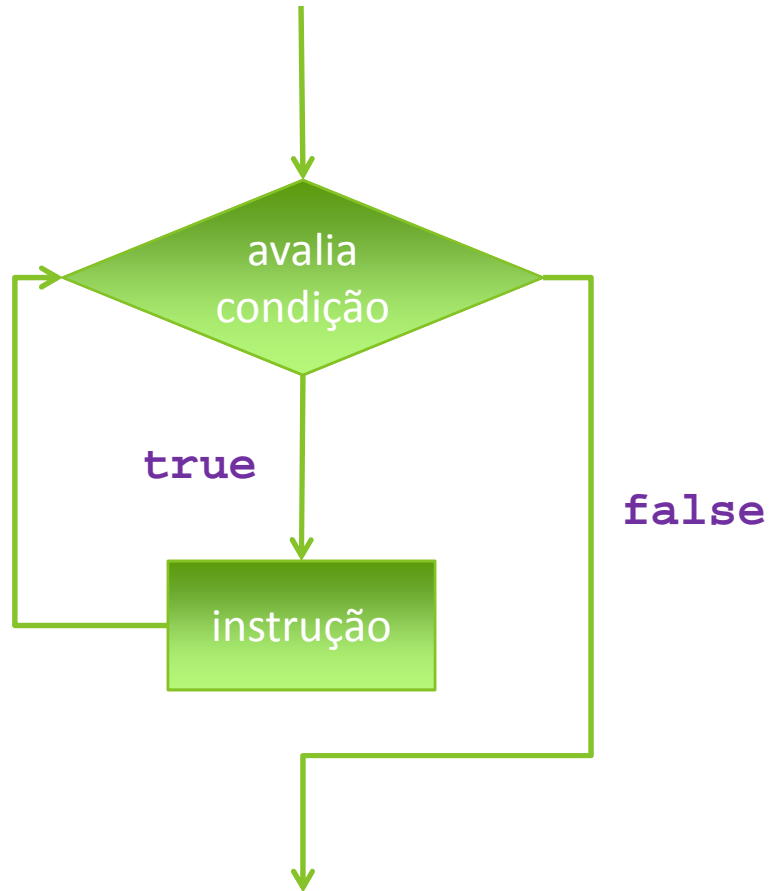
- Estruturas de repetição permitem executar um trecho de código várias vezes;
- Também podem ser chamados de **loops**;
- Assim como as estruturas condicionais, estruturas de repetição também são controladas por expressões booleanas;
- Tipos de estruturas de repetição em Java
 - while
 - do ... while
 - for
 - for (foreach)
- Cada situação pede um tipo de loop diferente!

+ while

```
while (condicao) {  
    // instrução  
}
```

- Se condicao for true, então a instrução é executada
- Após a execução ter sido realizada, se a condicao continuar true, instrução é realizada novamente
- A execução se repete indefinidamente até a condicao tornar-se false

+ while



+ while

```
int x = 10;
int[] a = {4, 2, 5, 1, 10};
boolean achou = false;
int i = 0;

while (achou == false && i < a.length) {
    if (a[i] == x) {
        achou = true;
    }
}

System.out.println(achou);
```

+ Loop infinito

■ CUIDADO!

- Caso a condição do loop nunca seja false, então aquele laço será executado “para sempre”;
- Erro de programação muito comum;

+ Loop Aninhado

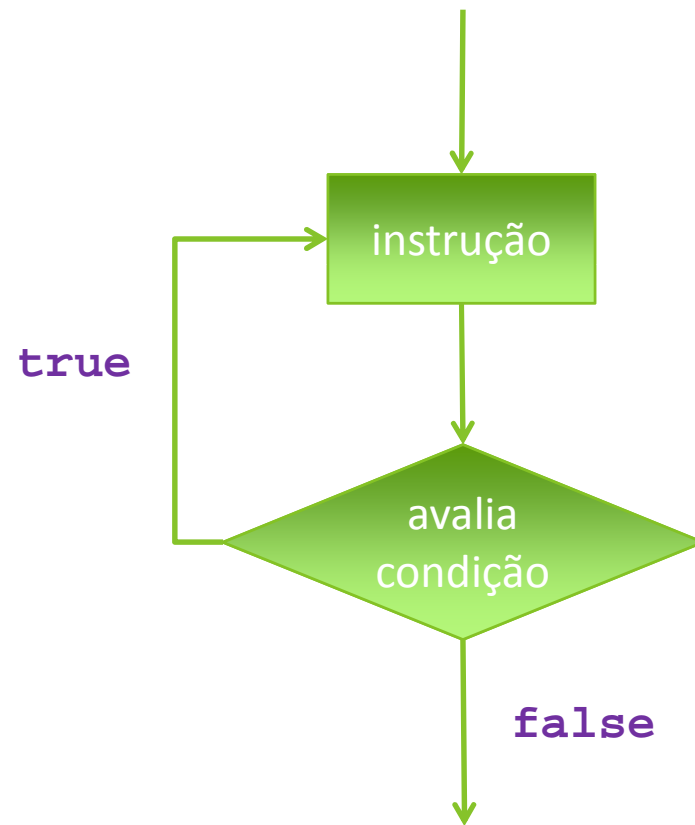
- Um loop pode conter uma outra estrutura de repetição dentro do seu corpo;
- Para cada iteração externa, deve-se completar a interna;

+ do ... while

```
do {  
    // instrução  
} while(condicao);
```

- A instrução é executada uma primeira vez
- A condicao é avaliada e caso seja true, a instrução é executada novamente
- A instrução se repete indefinidamente até a condicao tornar-se false

+ do ... while



+ do ... while

```
int i = 0;
do {
    i = i + 1;
    System.out.println(i);
} while (i < 5);
```

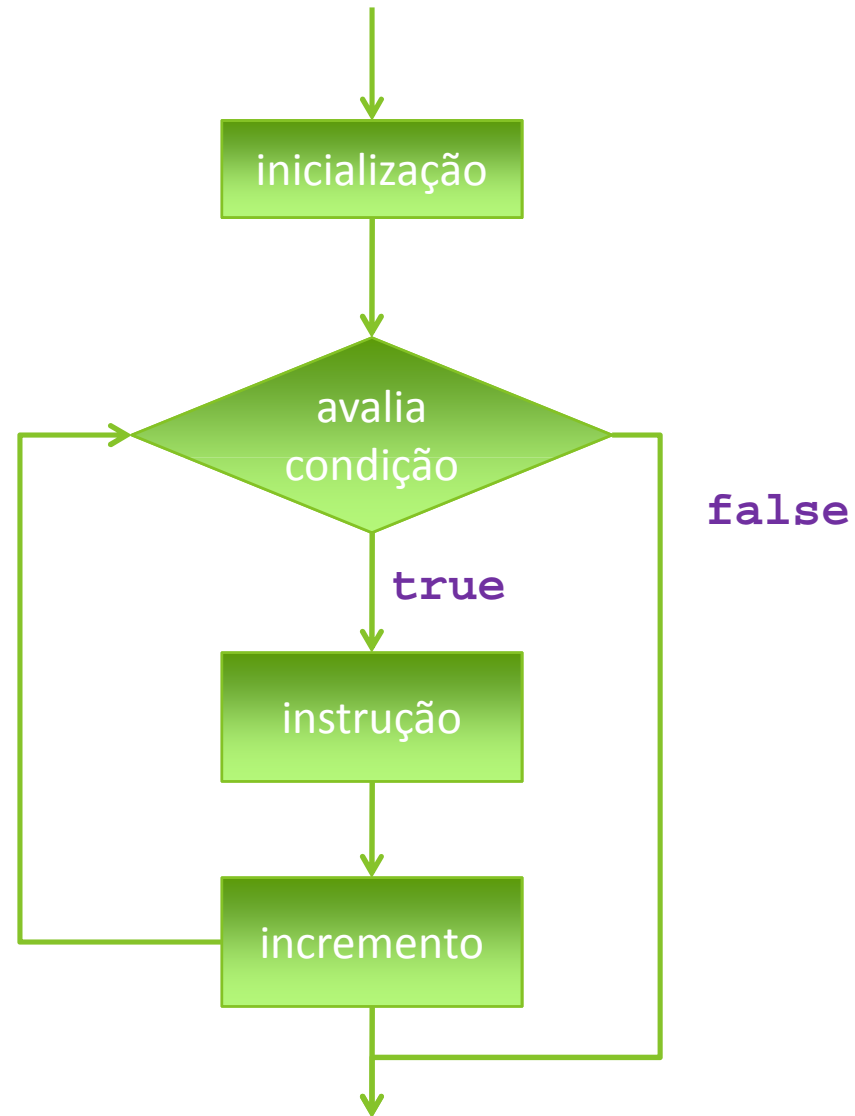
- O corpo do do ... while é executado pelo menos uma vez

+ for

```
for (inicializacao; condicao; incremento) {  
    // instrução  
}
```

- A inicializacao é executada uma vez antes do loop começar
- A instrução é executada até a condicao tornar-se false
- O incremento é executado ao final de cada iteração

+ for



+ for

```
int[] a = new int[10];  
for (int i = 0; i < a.length; i++) {  
    a[i] = i * 2;  
}
```

- O campo de inicialização pode ser usado para declarar uma variável
- O campo de incremento pode ser calculado da forma que o programador quiser
- Usa-se o for quando sabemos o número de iterações ou quando ele é facilmente calculado

+ for (foreach)

```
for (Tipo variavel : colecao) {  
    // instrução  
}
```

- Introduzido na versão 5.0;
- “Para cada” elemento...

+ for (foreach)

```
int[] numeros = {1, 2, 5, 7, 11, 13, 17, 19};  
for (int i : numeros) {  
    System.out.println(i);  
}
```

- Para cada elemento da coleção realiza-se a(s) instrução(ões) do corpo do for;

+ Inicializando Arrays

- Existem várias formas de fazê-lo

```
int[] numeros = {13, 43, 1, 123} ;

String[] animais = {"cachorro", "macaco",
    "cavalo"};

float[] precos = new float[5];
for (int i = 0; i < precos.length; i++) {
    precos[i] = 10 * (i + 1);
}
```


+ Arrays Multidimensionais

- Arrays unidimensionais armazenam uma lista de elementos;
- Arrays bidimensionais podem ser imaginados como uma tabela (matriz), com linhas e colunas;
- Arrays podem ter quantas dimensões forem necessárias;

+ Arrays Multidimensionais

- Em Java um array multidimensional não é nada mais do que um array de arrays;
- Um array bidimensional pode ser declarado especificando-se o tamanho de cada dimensão separadamente;

```
int[][] matriz = new int[3][3];
```

+ Arrays Multidimensionais

- O elemento é referenciado usando dois índices:

```
int valor = matriz[3][1];
```

- Um array armazenado em uma linha, pode ser referenciado usando um índice:

```
int[] linha = matriz[2];
```

+ Arrays

- Se é feito um acesso a um elemento do array fora dos seus limites, é gerada um exceção:
 - `IndexOutOfBoundsException`

```
String[] animais = {"Cavalo", "Cachorro"};  
System.out.println(animais[2]);
```

+ Prática

1. Implemente a classe `RepositorioConta`. Esta deve ser responsável por armazenar um conjunto de contas num array. Além disto deve possuir os métodos `inserir` (insere uma conta no array), `remover` (remove uma conta do array), `pesquisar` (retorna uma conta do array que possua o mesmo número).
2. Faça o mesmo para a classe `Cliente`, retornando no método `pesquisar` o cliente que possuir o mesmo cpf.