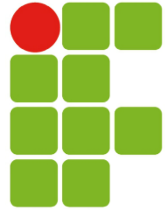


INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Programação Java para Desenvolvimento Android

Prof. Msc. Bruno Crestani Calegaro

INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Programação Java para Desenvolvimento Android

- A linguagem de programação **Java** é a base para o desenvolvimento de aplicativos **Android**

- Para ser um bom **desenvolvedor Android**, você precisa entender bem sobre programação **Java**. Contudo, a construção de **aplicativos** Android é fundamentalmente diferente de outros tipos de aplicações desenvolvidas em **Java**. Por isso, mesmo que você já tenha experiência, é necessário revisar a linguagem de programação **Java** com a perspectiva no desenvolvimento **Android**.

- Dessa forma, nessa aula vamos revisar o **Java** indo desde conceitos fundamentais de programação para iniciantes à conteúdos mais avançados. Vamos primeiro nos familiarizar com a sintaxe da linguagem **Java** e a realizar tarefas básicas de programação.

INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Linguagem de Programação Java

- O **Java** é uma linguagem de programação **fortemente tipada**
- Isso quer dizer que cada variável deve possuir explicitamente um **tipo** quando for declarada
- Por exemplo, se uma variável é um número, não será permitido guardar outra coisa nessa variável que não seja um número.
- Nesse quesito o **Java** é similar a linguagem de programação **C**

INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

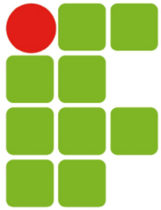
Linguagem de Programação Java

- O **Java** é uma linguagem **Orientada a Objetos**
- Uma linguagem Orientada a Objetos é baseada na modelagem de objetos e na comunicação entre eles.
- No mundo físico, por exemplo, imagine uma caneca como um objeto, ela tem propriedades como a sua cor e o quanto de café ela contém.
- Linguagens Orientadas a objetos nos permitem definir objetos como canecas e acessar suas propriedades.

INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Linguagem de Programação Java

- O **Java** é uma linguagem **Orientada a Objetos**
- Nós também podemos enviar mensagens para os objetos, assim, eu poderia perguntar para minha caneca, “Está vazia?“.
- Podemos, então, criar e manipular todos os tipos de objetos para fazer coisas diferentes em nosso aplicativo.
- Por exemplo, podemos usar o objeto Câmera para tirar uma foto. O objeto Câmera representa a câmera física em um celular Android.



Java não é Javascript

- Apesar do nome ser parecido as duas linguagens não são iguais (em muitos pontos...)
- Historicamente as duas linguagens tinham uma relação com o Java sendo projetado para ser usado em aplicações empresariais (lado servidor) e Javascript nos navegadores de internet (lado cliente)
- Mas essa relação hoje em dia não é válida pois o Javascript evoluiu e hoje pode até mesmo fazer aplicativos Android!

INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Tipos de Dados Básicos

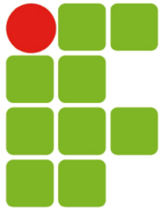
Tipo	Descrição
int	Um valor inteiro, ou seja, um número inteiro (sem decimais) que inclui zero e números negativos.
float	Um valor de ponto flutuante, um valor decimal (quebrados, ou com vírgula). Porem as casas decimais (do lado direito da vírgula) podem “flutuar”, ou seja, são imprecisas. Quando são necessários decimais precisos, como moeda, devemos usar o BigDecimal.
bolean	Um valor que pode ser verdadeiro ou falso. As palavras-chaves no Java para os valores são “true” e “false“
char	Um único caractere, como a letra ‘A’ ou o símbolo ‘#’. Cuidado, pois as maiúsculas e minúsculas são diferentes, então ‘a’ e ‘A’ são duas coisas diferentes.
String	Uma String é um conjunto de caracteres formando um texto. Por exemplo, “Android é legal”.

INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Variáveis

- Uma variável é basicamente um recipiente utilizado para armazenar dados.
- Java é uma linguagem **fortemente tipada**, o que significa que precisamos declarar o tipo de dados de uma variável.

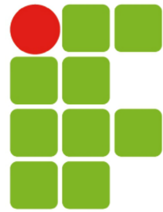
```
String mensagem = "Android é Legal";
```



Variáveis

- Observe o exemplo ao lado:
  - String é a declaração do tipo da variável
  - mensagem é o nome da variável
  - = é o sinal para atribuir um valor a uma variável
  - "Android é legal" é um valor textual, lembre que textos devem ser escritos com aspas duplas
  - ; é o final da linha. Imprescindível em cada linha de código.

```
String mensagem = "Android é Legal";
```



INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Variáveis

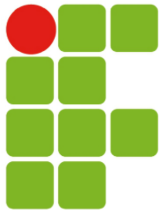
- Outros exemplos:

```
boolean existe = true;
```

```
char umaLetra = 'A';
```

```
float distancia = 13.24f;
```

```
int numero = 10;
```

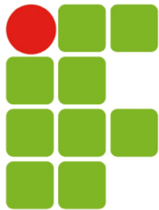


INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Métodos

- Métodos são blocos de código com nome que podem ser chamados (invocados) de outros lugares do nosso programa
- Quanto chamados irão executar alguma ação ou devolver algum tipo de resultado para podemos usar.
- Os métodos são usados para organizar o nosso código em pedaços reutilizáveis.

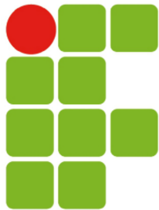
```
01.  public int length() {
02.    int tamanho = 0;
03.    ... calcula o tamanho da String ...
04.    return tamanho;
05.  }
```



Métodos

- Considere o exemplo simplificado do método **length()**, método pertencente a uma **String** que retorna o seu tamanho
- A primeira palavra na **Linha 01** declara a visibilidade do método e é, muitas vezes, **public** ou **private**
  - **public** significa que o método é acessível de qualquer lugar em nosso aplicativo.
  - **private** só estão disponíveis dentro da classe onde são definidos.

```
01.  public int length() {
02.    int tamanho = 0;
03.    ... calcula o tamanho da String ...
04.    return tamanho;
05.  }
```

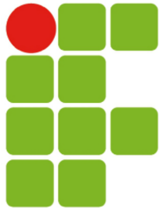


INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Métodos

- A segunda palavra no método é o tipo de dados que será devolvido.
- Neste caso, o **método** vai devolver um número, ou **int**.
- Se o **método** executado não for devolver nenhum tipo de dados, usamos a palavra-chave **void**, para indicar que aquele método não irá retornar nenhum valor.
- Em seguida é o nome do método.

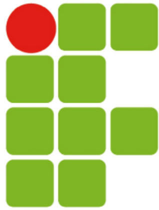
```
01. public int length() {
02.     int tamanho = 0;
03.     ... calcula o tamanho da String ...
04.     return tamanho;
05. }
```



Métodos

- Imediatamente após o nome vemos dois **parênteses** vazios.
- Os **parênteses** são obrigatórios, mas quando estão vazios, significa que o **método** não recebe nenhum dado de entrada, ou seja, não recebe **parâmetros** durante a chamada.
- Se uma ou mais variáveis fossem adicionadas entre os **parênteses**, seria preciso passar valores ou variáveis no momento da chamada do **método**.

```
01.  public int length() {
02.    int tamanho = 0;
03.    ... calcula o tamanho da String ...
04.    return tamanho;
05.  }
```

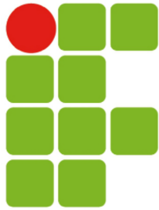


INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Métodos

- A primeira linha termina com uma **chave de abertura**, e uma **chave de fechamento** encontra-se na **Linha 05**.
- Em **Java**, blocos de código, como o código que compõem um **método**, são muitas vezes cercados por **colchetes** para designar todo o código que deve ser executado.
- Neste caso, isso significa que todas as linhas de código entre as **chaves** será executado cada vez que chamamos o **método length()**.

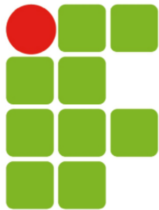
```
01.  public int length() {
02.    int tamanho = 0;
03.    ... calcula o tamanho da String ...
04.    return tamanho;
05.  }
```



Métodos

- Observe agora o conteúdo da **Linha 04**.
- O **return** é uma palavra-chave que indica que aquela variável ou valor será devolvido para quem chamou aquele **método**.
- É através desse comando que dizemos qual resposta o método vai dar.

```
01.  public int length() {
02.    int tamanho = 0;
03.    ... calcula o tamanho da String ...
04.    return tamanho;
05.  }
```



Chamar um método

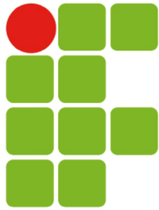
- Para usar um **método**, nós o chamamos assim "*variavel.metodo()*".
- Se ele retorna um valor, podemos armazenar o valor de retorno em algum lugar.
- Observe o exemplo chamando o **método length().**
- Primeiro, temos uma **String** que tem *15 caracteres* (incluindo espaço em branco).
- Chamamos o método na próxima linha usando a notação de ponto. Estamos dizendo para usar o **método length()** definido pela classe **String**.
- Neste caso, o método **length()** calcula a quantidade de caracteres e retorna o valor. Então, armazenamos o valor em uma nova variável **int** chamada **tamanhoDoTexto**.

```
01.    String texto = "Android é legal";
02.    int tamanhoDoTexto = texto.length();
```

INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

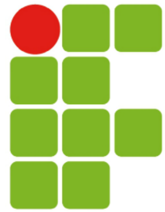
Grande Benefício do Uso de Métodos

- Imagine que temos cinco variáveis **String** diferentes que precisamos saber o tamanho do conteúdo delas.
- Se não tivesse um **método length()** nas **Strings** para usarmos, nós teríamos que escrever o código para calcular o tamanho de cada conteúdo de cada variável.
- Com os métodos nós podemos reutilizar funcionalidades, economizando linhas de programação e organizando o código.



Comentários

- **Comentários** em código **Java** são linhas de código que não têm qualquer efeito sobre a forma como o programa é executado.
- Eles são meramente informativos, destinado a nos ajudar a compreender como funciona o código ou como é organizado.
- **Comentário Simples**
  - Comentários simples começam com duas barras: **//**
  - Tudo na linha após as barras torna-se comentário e é completamente ignorado pelo programa.
- **Comentário Múltiplo**
  - Às vezes queremos comentários mais longos que se estendem por mais de uma linha. Nesse caso, use um marcador de abertura para marcar o início do comentário: **/*** , e um marcador de encerramento para marcar o fim do comentário: ***/** .



Comentário Simples

```
01.   // Este é um comentário de uma única linha
02.    String codigo = "12353454"; // Somente esta parte é um comentário
```

INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

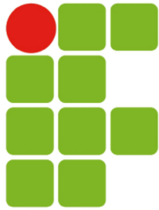
Comentário Múltiplo

```
01.    / * Este é um comentário multi-line. Embora apenas a abertura
02.     * E marcadores de fechamento são obrigados, muitas vezes adicionar um
03.     * Asterisco no início de cada linha para torná-lo
04.     * Mais legível, e terminar o comentário com o
05.     * Marcador em uma nova linha.
06.     * /
```

INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Classes e Objetos

- Em geral, os arquivos de código em **Java** são organizados como classes
  - Um arquivo (com extensão *.java*) geralmente contém uma definição de classe, e ele é compilado em um arquivo de classe (com extensão *.class*).
- Lembre que **Java** é uma linguagem **orientada a objetos**, isso significa que ele é construído em torno do conceito de objetos.
- Os **objetos** podem ser representações de objetos físicos, como uma caneca, ou objetos mais abstratos, como uma conexão para um site.
- A **classe** se destina a definir um **objeto** e como ele funciona. Deste modo, uma **classe** é como um modelo de um objeto. As **classes** definem coisas sobre **objetos** como propriedades e as habilidades do objeto são definidos como métodos.



Classes e Objetos

- Observe uma classe que poderia ser usada para representar uma simples conexão para um site.
- Neste exemplo temos uma variável de classe, url que representa a **URL** que esse objeto se destina a conectar.
- Podemos ver que no método **setURL()** usamos na frente da variável a palavra-chave **this**, isso serve para diferenciar a variável de escopo de classe com a variável que vem por parâmetro no método.

```
01. public class URLConnection {
02.     // Variáveis de classe (propriedades do o objeto)
03.     private String url;
04.     // Métodos (habilidades do objeto)
05.     public void setURL(String url) {
06.         this.url = url;
07.     }
08.     public void connect () {
09.       // Código para fazer uma conexão HTTP
10.     }
11. }
```

INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Classes e Objetos

- Temos dois métodos, o primeiro é chamado **setUrl()** e é usado para definir a variável de classe url, e o segundo é chamado **connect()** e seria onde vai o código para estabelecer uma conexão com a url.

```
01. public class URLConnection {
02.     // Variáveis de classe (propriedades do o objeto)
03.     private String url;
04.     // Métodos (habilidades do objeto)
05.     public void setURL(String url) {
06.         this.url = url;
07.     }
08.     public void connect () {
09.         // Código para fazer uma conexão HTTP
10.     }
11. }
```

INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Modificadores de Acesso

- Para as classes, variáveis de classe, e métodos, normalmente queremos especificar quem pode acessá-los.
- Em alguns casos, queremos que as coisas fiquem disponíveis publicamente, o que significa que qualquer um pode ver e utilizar as variáveis e métodos.
- Outras vezes, a gente quer manter as coisas privadas para proteger os dados do mundo exterior ou alterações acidentais. As palavras-chave **public** , **private** e **protected** são modificadores de acesso que controlam quem pode acessar a classe, variável ou método.

MODIFICADORES DE ACESSO	MESMA CLASSE	MESMO PACOTE	SUBCLASSE	OUTROS PACOTES
public	SIM	SIM	SIM	SIM
protected	SIM	SIM	SIM	NÃO
private	SIM	NÃO	NÃO	NÃO
nenhum modificador de acesso	SIM	SIM	NÃO	NÃO

INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Pacotes

- As classes são frequentemente agrupadas em **pacotes**.
- Um pacote é simplesmente uma coleção de classes relacionadas e são geralmente denominados em um formato que segue a notação de domínio reverso, ou seja, uma URL, como de um site, invertida: *ifsc.edu.br* para *br.edu.ifsc*.
- Em projeto Android podemos criar diferentes pacotes para ajudar a organizar as classes criadas em funcionalidades. Por exemplo, o pacote *ui* se refere a todas as classes referente a interface de usuário, o pacote *database* se refere a todas as classes referentes a conexão e manipulação de uma banco de dados, etc.

INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Condicionais

- Uma das coisas mais básicas de todas as linguagens de programação é o poder de verificar condições e, em seguida, fazer alguma coisa, se a condição for satisfeita.
- Em **Java** conseguimos isso usando a condicional **if**, que verifica a condição booleana como "**true**" e "**false**".

```
if (tipo.equals("Aluno")){
    Log.d( tag: "MEUAPP",  msg: "é aluno");
}
```

Condicionais

- Começamos com a palavra-chave **if** , e a condição de teste está cercada por parênteses.
- Se a condição for atendida, em seguida, o código dentro das chaves é executado. **Java**, como muitas outras línguas, usa chaves para começar e terminar **blocos de código** como este.

```
if (tipo.equals("Aluno")){
    Log.d( tag: "MEUAPP",  msg: "é aluno");
}
```

Condicionais

- Se quisermos tomar alguma ação, se a condição não for atendida, então podemos usar a palavra-chave **else** para especificar o que o código deve ser executado, como no exemplo a seguir:

```
if (tipo.equals("Aluno")){
    Log.d( tag: "MEUAPP", msg: "é aluno");
} else {
    Log.d( tag: "MEUAPP", msg: "tipo não encontrado");
}
```

Condicionais

- Além disso, podemos aninhar muitas condições juntas com **else if** , como este:

```
if (tipo.equals("Aluno")){
    Log.d( tag: "MEUAPP", msg: "é aluno");
}
else if (tipo.equals("Professor")){
    Log.d( tag: "MEUAPP", msg: "é professor");
} else {
    Log.d( tag: "MEUAPP", msg: "tipo não encontrado");
}
```

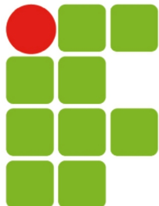
INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Explicação Extra

- O que é esse **equals()** no código? O método **equals()** está sendo usado em vez de dois sinais igual, por exemplo **if (tipo == "Aluno")**. Por quê?

- A razão é porque em Java, utilizando dois sinais de igual para objetos **String** não se compara o conteúdo dos objetos mas sim se os dois objetos são o mesmo...

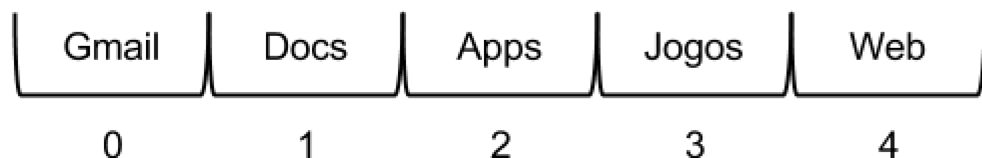
- Assim, fica a dica, sempre que forem comparar duas **Strings** usem o método **equals**, se usarem == a comparação vai dar errado!



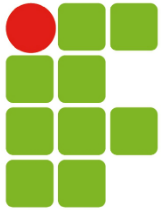
INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Arrays

- **Arrays** são estruturas usadas para organizar várias partes de dados ou outras variáveis. Basicamente são listas estruturadas em que cada parte de dados é referenciado pela sua posição na matriz. Veja este exemplo de uma lista de aplicativos.



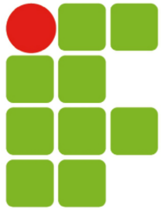
- Podemos pensar nos nomes dos aplicativos na gaveta como itens de uma matriz.
- O primeiro item da matriz, Gmail, está no índice 0. O segundo, Docs, está no índice 1, e assim por diante até o último, Web, no índice 4. O comprimento dessa matriz (ou o número de itens) é 5.



Arrays

- Quando declaramos uma variável de matriz, é preciso especificar o tipo dos itens de dados que estará na matriz.
- Por exemplo, se vamos usar uma matriz para armazenar uma lista de nomes, então teremos que declará-la como um array de **String**.
- A declaração de matrizes é semelhante à declaração de variáveis; nós simplesmente precisamos adicionar colchetes após o tipo de dados, como este:

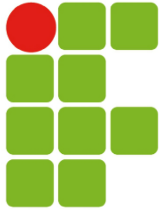
```
String[] nomes;
```



Arrays

- Podemos inicializar uma matriz em uma etapa ou várias etapas.
- Em um passo, nós simplesmente definimos todos os elementos do **array** entre chaves, como estes dois exemplos:

```
String[] cores = {"azul", "amarelo", "vermelho"};
int[] idades = {18,24,38};
```



Arrays

- Se queremos carregar os dados na matriz em um momento posterior, temos que especificar o tamanho da matriz. Itens específicos na matriz são acessados e / ou atribuídas usando o índice do array.
- O índice é especificado entre parênteses após o nome da matriz. **Arrays** em **Java** começam com índice zero, de modo que o primeiro item de uma matriz é na posição zero e é referenciado assim:

```
String[] appNomes = new String[5];
appNomes[0] = "Gmail";
appNomes[1] = "Docs";
appNomes[2] = "Apps";
appNomes[3] = "Jogos";
appNomes[4] = "Web";
String primeroApp = appNomes[0]; //primeiro será "Gmail"
```

Arrays

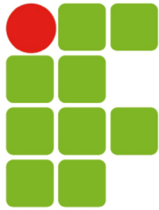
- A quantidade de itens de um array é igual ao seu comprimento.
- O objeto **Array** tem uma propriedade chamada **length** que retorna este número:

```
int quantidadeApps = appNomes.length; //será de 5
```

Loops

- **Loop** através de coleções ou conjuntos de coisas é outro comando básico de programação.
- Por exemplo, vamos dizer que queremos percorrer essa lista de nomes de aplicativos da seção **Arrays** acima e mostrar na tela.
- Se fizéssemos isso, um por um, o código ficaria assim:

```
Log.d( tag: "MEUAPP",  msg: "appNomes[0]");
Log.d( tag: "MEUAPP",  msg: "appNomes[1]");
Log.d( tag: "MEUAPP",  msg: "appNomes[2]");
Log.d( tag: "MEUAPP",  msg: "appNomes[3]");
Log.d( tag: "MEUAPP",  msg: "appNomes[4]");
```



Loops

- E se tivermos centenas de nomes? Ou se estamos lidando com milhões de registros de um banco de dados? E se a gente percorrer **20 linhas de código** para cada registro no banco de dados? Isso é um monte de código para escrever.
- A solução é fazer um loop através de um **array** (ou qualquer coleção que você está lidando) para executar um código para cada item da coleção. Vamos reescrever o código um pouco diferente.

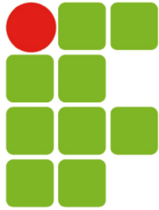
```
Log.d( tag: "MEUAPP",  msg: "appNomes[0]");
Log.d( tag: "MEUAPP",  msg: "appNomes[1]");
Log.d( tag: "MEUAPP",  msg: "appNomes[2]");
Log.d( tag: "MEUAPP",  msg: "appNomes[3]");
Log.d( tag: "MEUAPP",  msg: "appNomes[4]");
```

Loop “for”

- O **loop** “**for**” é lido como “para (algumas condições) faça o código dentro destas chaves”:

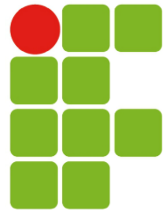
```
01.    for (inicializar declaração; compara declaração; incrementa declaração)
       {
02.     // Código a ser executado a cada passo do ciclo
03.    }
```

- Este tipo de **loop**, muitas vezes utiliza uma variável de contador (chamado “i” para “index” em nosso exemplo acima), que mantém o controle de nossa posição na coleção.



Loop “for”

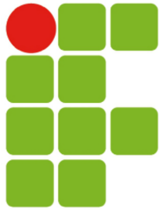
- **Inicializar Declaração**
  - A primeira instrução dentro do parêntese configura a variável do contador ou qualquer mecanismo que será usado para manter o controle de nossa posição no **loop**.
  - No nosso exemplo, a variável “i” será definida como 0, o primeiro índice do array **appNomes**.
- **Comparar Declaração**
  - A próxima declaração dentro dos parênteses (separados por ponto e vírgula) nos diz quando este ciclo deve parar o processamento e fazer a saída para a próxima linha de código após a chave de fechamento.
  - Em nosso exemplo, queremos continuar contanto que “i” é menor do que o número de itens na matriz. Desde há 5 itens na matriz, mas o último item está no índice 4 (porque começa em 0), nós queremos parar uma vez que i supera 4.
  - Caso contrário, vamos ter uma **ArrayIndexOutOfBoundsException**.



Loop “for”

Incrementar Declaração

- Essa última afirmação (mais uma vez separados por um ponto e vírgula) define como o nosso contador vai mudar depois que cada passo é executado.
- No nosso exemplo, estamos a utilizar um atalho especial “incremento”, que adiciona um para um valor inteiro. **++** pode ser usado para incrementar uma variável por um, e — pode ser utilizado para diminuir uma variável por um.



Loop “for”

Incrementar Declaração

- Temos de ter cuidado na definição dessas condições. E se a gente diminuir nosso contador ao invés de incrementá-lo, nossa condição nunca é cumprida?
- Em tais casos, o nosso programa ou aplicativo vai ser pego no que é conhecido como um “loop infinito”.
- O loop nunca vai sair e nosso programa irá usar todos os recursos indefinidamente, o que levará a um acidente, congelar, ou “aplicativo não está respondendo (ANR)” no Android.

Loop “for”

- O código da seção anterior reescrito usando **loop for** fica como:

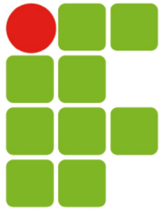
```
for(int i = 0; i < appNomes.length; i++){
    Log.d( tag: "MEUAPP", appNomes[i]);
}
```

INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Loop for-each

- O Java contém uma outra maneira mais simplificada de escrever um loop quando você quiser interagir sobre uma coleção de itens.
- A ideia por trás de um loop **for-each** é que você deseja executar um código para cada item da coleção.
- O **for-each** substitui as **três** condições do laço for com uma declaração simplificada.

```
for(String nome: appNomes){
    Log.d( tag: "MEUAPP", nome);
}
```



Loop for-each

- O lado direito do dois pontos “**:**” é a matriz ou coleção a ser percorrida completamente, e o lado esquerdo é uma variável temporária usada para guardar o item na iteração atual do loop.
- Assim, neste exemplo temos uma seqüência de cada item na matriz **appNomes** e o valor é armazenado em uma variável temporária chamada “**nome**“.
- Podemos ler isso como “para cada nome em appNomes, faça o código nestas chaves”.

```
for(String nome: appNomes){
    Log.d( tag: "MEUAPP", nome);
}
```

INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Tratamento de Erros

- Lembra do **ArrayIndexOutOfBoundsException** mencionado anteriormente?
- Esse é um possível erro que pode acontecer se você deseja acessar uma posição numa matriz que não existe. Por exemplo, o índice 10 de uma matriz de apenas 5 elementos.
- Pois então, o Java fornece uma forma para fazer o tratamento de todos os tipos de erros em nossos programas.
- E em alguns casos, ele te obriga a fazer o tratamento de erros sempre, por exemplo, abrir uma conexão de internet

INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Tratamento de Erros

- A ideia básica é que, se algo de ruim acontece em nosso programa ou aplicativo, o sistema de execução do **Java** vai levantar uma pequena bandeira que diz que algo está errado.
- Se estamos de olho nessas bandeiras podemos tratá-las adequadamente e evitar que nossos usuários passem por uma má experiência como fechar o aplicativo do nada.
- O ato de levantar esse bandeirinha é chamado de **lançar uma exceção**, e temos de pegá-las no código em nosso aplicativo.

Tratamento de Erros

- Observe o seguinte exemplo onde o código pode disparar uma exceção é envolvendo em um bloco **try-catch**:

```
01.   String url = "https://www.google.com";
02.   try {
03.     URL blogUrl = new URL (url);
04.   } catch (MalformedURLException e) {
05.     Log.e (“AndroidApp”, "Erro ao criar um objeto de URL com " + url);
06.   }
```

INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Tratamento de Erros

- Neste exemplo, o construtor **URL** (Linha 03) lança um **MalformedURLException** se uma url inválida é passada como parâmetro de entrada. Se você não capturar essa exceção, então o nosso código não irá compilar.
- O bloco **catch** requer o tipo de exceção a ser capturado e é declarado como uma variável dentro dos parênteses após a palavras-chave **catch**. Então, como de costume, o código a ser executado em caso de exceção fica entre as chaves.

```
01.   String url = "https://www.google.com";
02.   try {
03.     URL blogUrl = new URL (url);
04.   } catch (MalformedURLException e) {
05.     Log.e (“AndroidApp”, "Erro ao criar um objeto de URL com " + url);
06.   }
```

Tratamento de Erros

- Se nós precisamos verificar se há várias exceções, como se tentou abrir uma conexão **HTTP** utilizando **blogUrl**, que lança uma **IOException** , então podemos encadear declarações **catch** um sobre o outro, como este novo exemplo:

```
01.   try {
02.       // Algum código
03.    } catch (MalformedURLException e) {
04.       Log.e (“AndroidApp”, "Houve um erro na criação de um objeto URL.";
05.    } catch (IOException exception) {
06.       Log.e (“AndroidApp”,  "Erro ao conectar ao site.");
07.   }
```

INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Tratamento de Erros

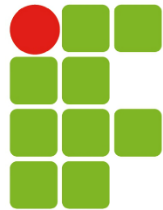
- Finalmente, há mais um bloco de código que pode ser colocado a essa sequência: o bloco **finally**. Este é o lugar onde nós colocamos o código que queremos executar mesmo que uma exceção é lançada:

```
01.    Writer writer;
02.    try {
03.      writer = new OutputStreamWriter (new FileOutputStream ("out.txt"));
04.      writer.write ("Video aula da quarentena uhul!");
05.    } catch (IOException e) {
06.      Log.e ("ANDROIDPRO", "Erro ao escrever no arquivo.");
07.    } finally {
08.     if (writer! = null) {
09.       writer.close ();
10.     }
11.    }
```

INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Conclusão

- Nesta aula vimos brevemente alguns conceitos da **Programação Orientada a Objetos**. Em suma, para o desenvolvimento Android é essencial compreender o conceito de classe, atributos e métodos
- Aprendemos sobre a sintaxe básica do **Java** com a declaração de variáveis, condicionais, loops e tratamento de erros.
- Podemos prosseguir com as etapas iniciais de programação para aplicativos móveis e construir aplicativos Android funcionais
- Contudo, para dominar o desenvolvimento Android é necessário mais aprofundamento sobre as complexidades da linguagem **Java** com conteúdos avançados de arquitetura de sistemas, *Collections,Thread,* entre outros.



INSTITUTO FEDERAL
SANTA CATARINA
Campus Geraldo Werninghaus

Programação Java para Desenvolvimento Android

Prof. Msc. Bruno Crestani Calegaro