

Minicurso de Java

Introdução a Java

João Paulo Taylor Ienczak Zanette

22 de Fevereiro de 2017

O que é programação?

Introdução a Java

1. Deve ser "simples, orientada a objetos, e familiar";
2. Deve ser "robusta e segura";
3. Deve ser "de arquitetura neutra e portátil";
4. Deve executar com "alta performance";
5. Deve ser "interpretada, dinâmica e com suporte a threads ";

Características gerais da linguagem

Multiplataforma: O mesmo código pode rodar em mais de uma plataforma sem a necessidade de recompilar;

Biblioteca padrão extensa: Oferece diversos recursos já na biblioteca padrão, desde interfaces gráficas até manipulação de áudio/vídeo;

Alocação não-determinística: Recursos alocados não são liberados explicitamente pelo programador, isso é feito por um *Garbage-Collector*;

Suporte a JIT: Alguns trechos de código são otimizados em tempo de execução (*Just In Time Compilation*).

“Java é uma mãezona: não deixa você fazer algo errado e ainda limpa tudo para você.”

Onde é utilizada

- Aplicações Desktop no geral (com ou sem interface gráfica);
- Comunicação com Banco de Dados em servidores de aplicações Web (*back-end*);
- Aplicações para dispositivos móveis Android;
- Sistemas embarcados e de tempo-real.

Foi bastante utilizada também para aplicações de celulares antigos (JavaME).

Toda aplicação Java roda na JVM a partir de uma sequência de instruções geradas de um código Java. Essas instruções se chamam ***Bytecode***.

Desenvolvendo uma aplicação Java simples


1. Crie um arquivo de código Java (.java);
2. Escreva o código (ver exemplo adiante), definindo o procedimento `main`;
3. Gere o *Bytecode* da aplicação, que será lido pela JVM. Isso é feito pelo processo de *Compilação*;
4. Execute o código chamando a JVM e indicando qual classe deve ser carregada. A JVM vai procurar e executar o procedimento `main` dessa classe.

Exemplo de código Java

```
public class Application {  
    public static void main(String... args) {  
        System.out.println("Hello, World!");  
    }  
}
```

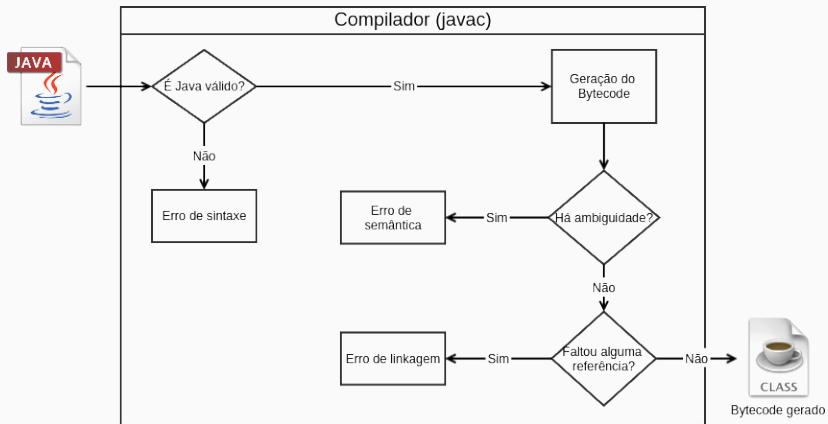
OBS: Toda aplicação Java inicia pelo main.

Utilizando o compilador de Java

☯ Eclipse: . Executar a aplicação também chama o compilador.

➤ Terminal: `javac Application.java`

Processo de compilação de Java



Bytecode gerado

```
public class Application {
```

```
    public Application();
```

```
        Code:
```

```
        0: aload_0
```

```
        1: invokespecial #1 // java/lang/Object."<init>"
```

```
        4: return
```

```
public static void main(java.lang.String...);
```

```
    Code:
```

```
        0: getstatic    #2 // java/lang/System.out
```

```
        3: ldc         #3 // Hello, World!
```

```
        5: invokevirtual #4 // java/io/PrintStream.println
```

```
        8: return
```

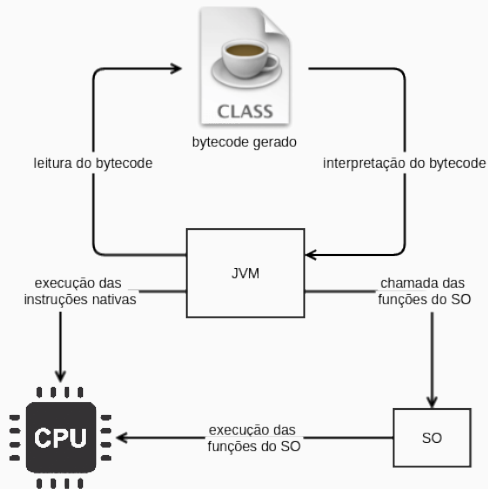
```
}
```

Executando uma aplicação Java

 Eclipse: Pressionar  +  ( Run).

 Terminal: `java Application`

Execução do Bytecode Java



Execução do *Bytecode* Java

Importante: A execução do código é sequencial, na ordem em que o código foi escrito. Portanto o seguinte trecho de código:

```
System.out.println("Linha 1");  
System.out.println("Linha 2");  
System.out.println("Linha 3");  
System.out.println("Linha 4");
```

Produzirá a saída:

```
Linha 1  
Linha 2  
Linha 3  
Linha 4
```

Trabalhando com programação

Problema: Como guardar valores?

Não queremos fazer programas que apenas mostrem textos na tela. Porém, se quisermos programas mais complexos, precisamos de mais recursos. Para fazer uma loja, por exemplo, precisamos:

- Guardar as informações dos produtos (nomes, preços, etc.);
- Guardar a lista de compras do cliente;
- Calcular o total da compra para que o cliente possa pagar.

Variáveis são valores guardados na memória (geralmente RAM).
Toda variável possui:

Identificador: Nome para referenciá-la no código. Deve ser único dentro de um mesmo escopo.

Tipo: Define o tipo de dado que ela guarda.

Há também, em programação, o conceito de *constantes*, que funcionam semelhante às variáveis, porém não podem ser alteradas.

Criando uma variável

Para criar uma variável, basta *declarar* ela, ou seja, escrever:

```
Tipo nome;
```

É possível ainda criar uma variável e já atribuir um valor a ela, da forma:

```
Tipo nome = valor;
```

Os tipos de dados que podem ser guardados são divididos em:

Tipos Primitivos: Valores indivisíveis, geralmente números;

Tipos Definidos por Usuário: Tipos personalizados, compostos por diferentes valores internos.

Tipos primitivos

Numéros inteiros:

Tipo	Tamanho	Intervalo
byte	1 Byte	-128 a 127
short	2 Bytes	-2^{15} a $2^{15} - 1$
int	4 Bytes	-2^{31} a $2^{31} - 1$
long	8 Bytes	-2^{63} a $2^{63} - 1$

Numéros reais:

Tipo	Tamanho	Intervalo
float	4 Bytes	2^{-149} a $(2 - 2^{-23}) \cdot 2^{127}$
double	8 Bytes	2^{-1074} a $(2 - 2^{-52}) \cdot 2^{1023}$

Para caracteres há o tipo char (2 Bytes).

Operações: Soma

```
int a = 5;  
int b = 7;  
int sum = a + b;
```

Após executar o trecho acima, o valor de `sum` será 12.

Operações: Subtração

```
int a = 5;  
int b = 7;  
int diff = a - b;
```

Após executar o trecho acima, o valor de `diff` será `-2`.

Operações: Multiplicação

```
int a = 5;  
int b = 7;  
int product = a * b;
```

Após executar o trecho acima, o valor de `product` será 35.

Operações: Divisão

```
int a = 5;  
int b = 7;  
int quotient = a / b;
```

Esse é um caso especial em que, após executar o trecho acima, o valor de `quotient` será 0. Isso se dá porque estamos utilizando números *inteiros*.

Operações: Divisão real

Se quisermos um resultado em números reais, é preciso que *pelo menos um dos operandos* seja um número real (float ou double):

```
int a = 5;  
double b = 7;  
double quotient = a / b;
```

(OBS: Se o resultado da divisão for colocada em uma variável que represente um inteiro, o resultado será *truncado*, ou seja, o valor após a vírgula é ignorado: $1.1 = 1$, $1.9 = 1$).

Representando os produtos de uma loja

Suponhamos que a tabela de preços da loja seja a seguinte:

Produto	Preço
Pão	R\$0.50
Leite	R\$3.00
Suco	R\$5.00
Pizza	R\$9.99

Assim, precisaremos guardar o preço dos produtos em variáveis. Qual o tipo de dado seria mais adequado para representar dinheiro/preço?

Escolhendo o tipo de dado

- A primeira vista, `double` e `float` parecem ótimas decisões, afinal representam números reais.
- **Porém**, ambos possuem erros de arredondamento, que podem fazer ou você ou o cliente perder/ganhar dinheiro por uma simples má escolha de tipo de dado.
- Por exemplo, às vezes você escreve no código que o valor de um `float` é 4.5, porém o que efetivamente será carregado é 4.4999999999, que seria o número mais próximo representável por um `float`.

Escolhendo o tipo de dado

Para resolver esse problema, algumas linguagens utilizam um tipo de dado especial que possui um número fixo de casas decimais, chamado `Decimal` (ou "Fixed Point", em contrapartida com "Floating Point"), que pode servir para representar dinheiro (que sempre terá duas casas decimais apenas).

Infelizmente Java não possui esse tipo. Utilizaremos, portanto, o tipo `int`.

Declarando as variáveis

Assim, a declaração dos preços dos produtos será:

```
int breadPrice = 50;  
int milkPrice = 300;  
int juicePrice = 500;  
int pizzaPrice = 999;
```

Interagindo com o usuário

Em nossa loja será necessário informar a quantidade de cada produto que o cliente está levando (para poder calcular o total a ser pago).

Para isso, é possível pedir ao usuário que digite um texto no Console (ou Terminal), ler esse texto e guardar em uma variável:

```
System.console().readLine("Digite algo: ");
```

Conversão: Texto para número

Vale apontar, porém, que o tipo de dado retornado pelo `readLine(...)` é de texto (chamado `String`). Porém, queremos utilizar um tipo numérico, e precisamos converter esse texto para um número inteiro. Isso pode ser feito com:

```
Integer.parseInt(texto);
```

E em seguida jogar o resultado em uma variável do tipo inteiro.

Interação com o usuário

Assim, para pedir ao usuário que digite quantos pães está comprando, basta utilizar:

```
String answer = System.console().readLine(  
    "Quantidade de pães comprados: "  
);  
int breads = Integer.parseInt(answer);
```

Ou ainda:

```
int breads = Integer.parseInt(  
    System.console().readLine(  
        "Quantidade de pães comprados: "  
    )  
);
```

Em nosso programa, queremos mostrar na tela apenas os produtos que forem comprados (afinal, uma loja geralmente tem centenas de produtos cadastrados). Porém, como dizer para nosso programa apenas executar um trecho de código se e somente se uma condição for satisfeita?

Ou seja, como podemos fazer nosso programa tomar decisões?

Variáveis lógicas

Em programação, há um tipo de dado chamado "Lógico", que possui apenas dois valores possíveis: **verdadeiro** e **falso**. Em Java, esse tipo se chama `boolean`:

```
boolean a = true;  
boolean b = false;  
boolean c = 5 > 3; // true  
boolean d = 5 < 3; // false
```

Operadores lógicos

Operador	Descrição
<code>!x</code>	true se x for false (inversor/NOT).
<code>x == y</code>	true se os valores forem iguais.
<code>x != y</code>	true se os valores forem diferentes.
<code>x > y</code>	true se x for maior do que y.
<code>x < y</code>	true se x for menor do que y.
<code>x >= y</code>	true se x for maior ou igual a y.
<code>x <= y</code>	true se x for menor ou igual a y.
<code>x && y</code>	true se ambos x e y forem true (AND).
<code>x y</code>	true se pelo menos um dos dois valores forem true (OR).
<code>x ^ y</code>	true se apenas x ou apenas y for true, mas não ambos ao mesmo tempo . (XOR).

Estrutura condicional if

Para que nosso programa possa "escolher" se vai ou não executar determinado trecho, há uma estrutura de código chamada `if`, em que é entregue qualquer valor do tipo `boolean` entre parênteses e, se durante a execução esse valor for `true`, o trecho de código dentro das chaves será executado.

```
if (condição) {  
    /*  
        trecho a ser executado caso a condição  
        seja verdadeira  
    */  
}
```

Exibindo texto no console

Assim, para que o produto "Pão" seja apenas exibido se for comprado mais de um pão, podemos escrever:

```
if (breads > 0) {  
    System.out.println(  
        "Foram comprados " + breads + " pães"  
    );  
}
```

OBS: Perceba que é possível juntar textos (Strings) com outros valores (incluindo outras Strings) utilizando "+".

Escolhendo a forma de pagamento

Ao final da compra, o cliente irá querer escolher a forma de pagamento. Podemos estabelecer três opções:

- Se o usuário digitar "C", o pagamento será efetuado com Cartão;
- Se o usuário digitar "D", o pagamento será efetuado em dinheiro.
- Para qualquer texto diferente que for digitado o programa irá alertar que a opção é inválida.

If-Else

Para isso, podemos fazer uma opção extra caso um if falhe, através de else:

```
String paymentForm = System.console().readLine(
    "Escolha a forma de pagamento " +
    "(C = Cartão, D = Dinheiro): "
);
if (paymentForm.equals("C")) {
    System.out.println("Escolhido: Cartão.");
} else if (paymentForm.equals("D")) {
    System.out.println("Escolhido: Dinheiro.");
} else {
    System.out.println("Forma de pagamento inválida.");
}
```


- Exercícios com correção automática:
`http://pet.inf.ufsc.br/~jptiz/academico/`
- Material do minicurso (explicação, códigos de exemplo, etc.):
`https://github.com/JPTIZ/java_minicourse/`

Exercícios da aula de hoje:

- POO-I - Unidade 1: Variáveis, Operadores e Estruturas Condicionais (`http://pet.inf.ufsc.br/~jptiz/academico/exercicio/?course=pooi&unit=0`)