Java

Síntese

    - **Criando um Objeto**
    - **Heap**
- **Variavéis**
    - **Primitivas**
    - **Tamanhos das Variáveis Primitivas**
    - **Os tipos de uma variável Primitiva**
        - `boolean`
        - `char`
        - `byte`
        - `short`
        - `int`
        - `long`
        - `float`
        - `double`
    - **Referência de objeto:**
    - **Criando uma Variável de Referência:**
    - **Matriz**

Observações

▼

- **Documentação java:** https://docs.oracle.com/en/java/javase/17/docs/api/index.html
- Java é uma linguagem **Orientada a Objetos**.
- O **Compilador** é um software que transcreve o código `.java` que foi escrito em um editor de texto qualquer para um código binário, possibilitando agora que o computador consiga ler o código que escrevemos.

- O arquivo criado pelo compilador é chamado de **Arquivo Binário** ou **Arquivo Executável**.
- O compilador retornar um erro se o arquivo `.java` tiver algum erro, ou seja, o arquivo binário só vai ser criado pelo compilador se o código não tiver nenhum erro.
- Depois que o arquivo binário for criado, vai ser preciso outro software para executar esse arquivo.
- O software de compilação e execução fazem parte do pacote **Java Development Kit** (**JDK**).

Instalação

▼

▼ Linux

- Um maneira de instalar o **JDK** é por meio da linha de comando.

```
sudo apt install openjdk-17-jdk
```

- Para saber se for instalado mesmo, usamos o comando:

```
java -version
```

- Agora vamos criar a variável de ambiente.
- Antes disso precisamos saber onde o java foi instalado:

```
sudo update-alternatives --config java
```

- Agora vamos inserir a variável de ambiente no arquivo `.bashrc`.
- Abrindo o arquivo `.bashrc`:

```
sudo gedit ~/.bashrc
```

- Agora ao final do arquivo vamos inserir o seguinte conteudo:

```
JAVA_HOME=caminho...
export JAVA_HOME
export PATH=$PATH:$JAVA_HOME
```

Na parte onde tem `caminho...` deve ser colocado o caminho do java que conseguimos com o comando sitado mais acima. No nosso caso estamos instalando o **JDK17**, o conteúdo que estamos inserindo no `.bashrc` é:

```
JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64
export JAVA_HOME
export PATH=$PATH:$JAVA_HOME
```

- Logo abaixo tem uma outra maneira de instalar o **JDK**, essa criar variáveis temporárias, que são destruídas sim que o terminal é fechado, ou seja, nesse método abaixo é preciso fica criando as variáveis de ambiente toda vida que o terminal for aberto.
- Você pode baixar o **JDK** de diferentes lugares. Há uma página única que sempre se refere à versão mais recente do **JDK**: https://jdk.java.net/ .
- O arquivo que vai ser baixado desse site vai vim com a extensão `.tar.gz`.
- Mova esse arquivo para o diretório desejado e extraia ele com o comando:

```
$ tar xzf *.tar.gz
```

- Feito isso, você precisa criar uma variável de ambiente chamada `JAVA_HOME` que aponta para o diretório onde você expandiu o **JDK**.
- Se você expandiu um arquivo **JDK 17** no diretório `/home/gustavo/`, o comando que você precisa digitar neste `prompt` de `shell` é o seguinte:

```
$ export JAVA_HOME=/home/gustavo/jdk-17
```

- Você pode verificar se a `JAVA_HOME` variável foi definida corretamente digitando o seguinte código:

```
$ echo $JAVA_HOME
```

- Este comando deve imprimir o seguinte:

```
/home/gustavo/jdk-17
```

- Em seguida, você precisa atualizar sua `PATH` variável para adicionar o `bin` diretório do seu diretório **JDK** a ela. Isso pode ser feito com o seguinte comando:

```
$ export PATH=$JAVA_HOME/bin:$PATH
```

- Você precisa ter muito cuidado ao configurar essas duas variáveis porque um único erro, como um espaço em branco adicionado de um ponto-e-vírgula ausente, resultará em falha.
- Não feche este `prompt` de `shell`. Se você fechá-lo e abri-lo novamente, precisará criar essas duas variáveis novamente.
- Você pode verificar se está tudo bem digitando o seguinte comando:

```
$ which java
```

- Seu `shell` deve imprimir o caminho completo para o `java` arquivo executável no `bin` diretório da distribuição que você acabou de expandir. Neste exemplo, ele imprimirá:

```
/home/gustavo/jdk-17/bin/java
```

Comandos Básicos

Criação de classe

```
public class MyFirstClass {
}
```

- **Obs:** O nome do arquivo onde a classe está sendo criada deve ter o mesmo nome da classe. Ou seja, se o nome da classe for `MyFirstClass` o nome do arquivo tem que ser `MyFirstClass.java`.
- Você pode dar a essa classe qualquer nome, desde que não comece com um número.
- Porém, há uma convenção: o nome de uma classe Java começa com uma letra maiúscula. Isso não é obrigatório, mas todos os desenvolvedores Java seguem esta convenção.

Compilando o Primeiro Comando

- Mude para o diretório onde você salvou sua primeira aula `MyFirstClass.java`. Você pode verificar se está no diretório correto digitando `dir`. Ele irá mostrar os arquivos que você tem neste diretório. Você deve ver seu `MyFirstClass.java` arquivo.
- Verifique se o seu compilador está acessível a partir deste diretório, digitando o seguinte.

```
$ java -version
```

  Ele deve informar qual versão do `javac` você está usando no momento.

- Agora você está pronto para compilar seu primeiro código. Você pode digitar o seguinte.

```
javac MyFirstClass.java
```

Duas coisas podem acontecer neste ponto. Você pode receber mensagens de erro informando que o compilador não pode compilar seu código devido a erros em seu código Java. Você precisará consertá-los antes de poder prosseguir.

Se o compilador ficar calado e não reclamar de nada: parabéns! Isso significa que seu código Java foi compilado corretamente. Verificar o conteúdo do diretório novamente deve mostrar um novo arquivo nele: `MyFirstClass.class`.

- Certifique-se de que o compilador criou o `MyFirstClass.class` para você. Para executá-lo, basta digitar o seguinte comando:

```
$ java MyFirstClass
```

▼ Ramificação Condicional (if)

- Em java, um teste `if` é basicamente o mesmo que o teste booleano de um loop `while`.

```
class IfTest{

  public static void main(String[] args){

    int x = 3;
    if (x == 3){
      System.out.println("X deve ser igual a 3");
    } else if (x > 3){
      System.out.println("x é maior que 3");
    } else {
      System.out.println("x é menor que 3");
    }
    System.out.println("Isso será executado de qualquer forma.");
  }
}
```

▼ Convenções de valores

```
int x1 = (int) 23.3;
float x2 = (float) 23;
System.out.println(x1+" "+x2);
```

```
23 23.0
```

Estruturas de Loop

- O java tem 3 tipos de estruturas de loop padrão: `while`, `do-while` e `for`.
- A principal parte de um loop é o teste condicional, esse teste é uma expressão que resulta em um valor booleano.

Tipos de Dados

String

- **Métodos**
- **.length():** Retorna o tamanho da string, pode ser usado também para saber o tamanho de uma lista.

Métodos que já vem com o Java

Math

- **.random()** retorna um número aleatório entre `0` e `1`.

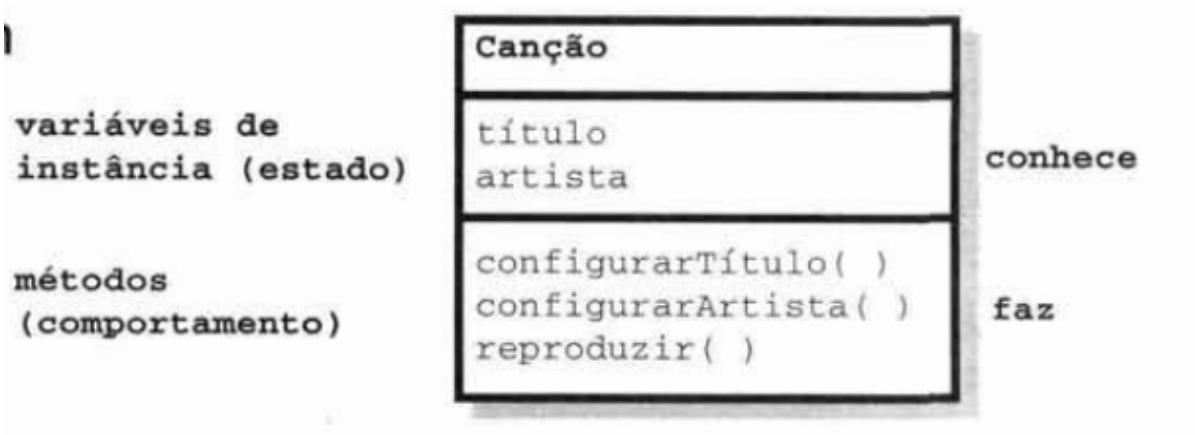
Orientação ao Objeto

- **Main:** A classe `main` em teoria é para ter só duas finalidades: **Testar** sua classe real e **Acionar/Iniciar** seu aplicativo java.

- **Variável de Instância:** Representa o estado de um objeto (os dados) e podem ter valores exclusivos para cada objeto desse tipo.

  Instância é uma outra maneira de chamar o nome Objeto.

- **Métodos:** São as coisas que um objeto fazem. Um objeto pode ter métodos que leiam ou gravem os valores das Variáveis de Instância.



- **Classe:** Uma classe não é um objeto. Mais ela serve para informar a maquina virtual como criar o objeto. Cada objeto criado a partir dessa classe terá seus próprios valores para as variáveis de instância da classe. Por exemplo, você pode usar a classe Button para criar vários botões diferentes, e cada botão poderá ter sua própria cor, tamanho, forma, rótulo e assim por diante.

- **Objetos:** Eles herdaram as variáveis de instancia e os métodos da classe.

  No tempo de execução, um programa java nada mais é do que objetos "comunicando-se" com outros objetos.

▼ **Criando um Objeto:** Para isso é preciso de duas classe. Uma para o tipo de objeto que deseja usar e outra para testar sua nova classe. Essa classe testadora vai ser a classe `main()`.

1. Criando a classe `Dog`:


```
class Dog{

  int size;
  String breed;
  String name;

  void bark(){
    System.out.println("Ruff! Ruff!");
  }
}
```

2. Criando a classe `main`:

```
class DogTestDrive(){
  public static void main(String[] args){
    //O código de teste de Dog entra aqui
  }
}
```

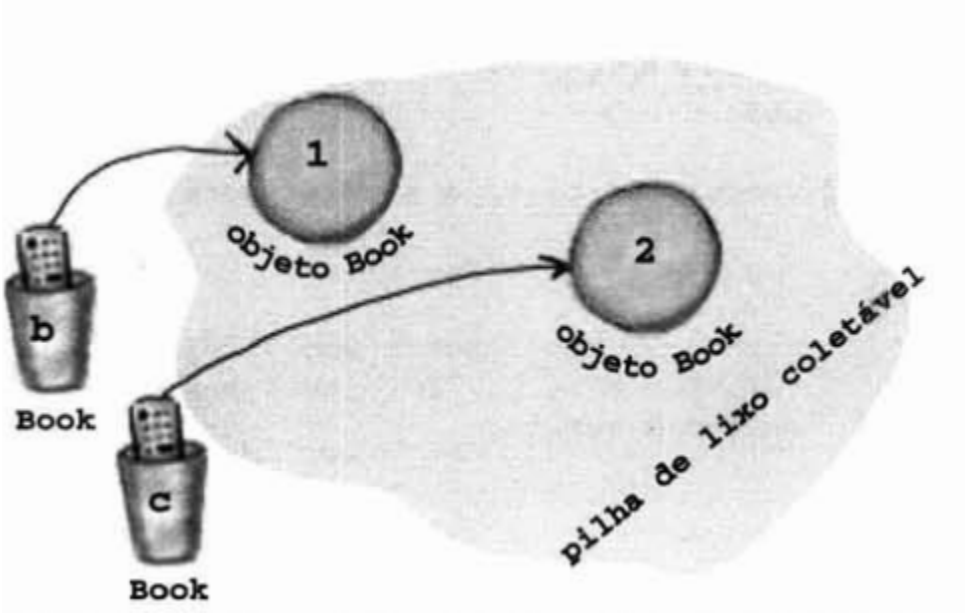
3. Na classe `main`, vamos criar um objeto que vai acessar as variáveis e métodos da classe `Dog`:

```
class DogTestDrive(){
  public static void main(String[] args){
    Dog d = new Dog();  // Objeto Dog

    d.size = 40;    //Atribuindo um valor a variável da classe Dog

    d.back();  //Chamando o médoto back() da classe Dog
  }
}
```

▼ **Heap:**

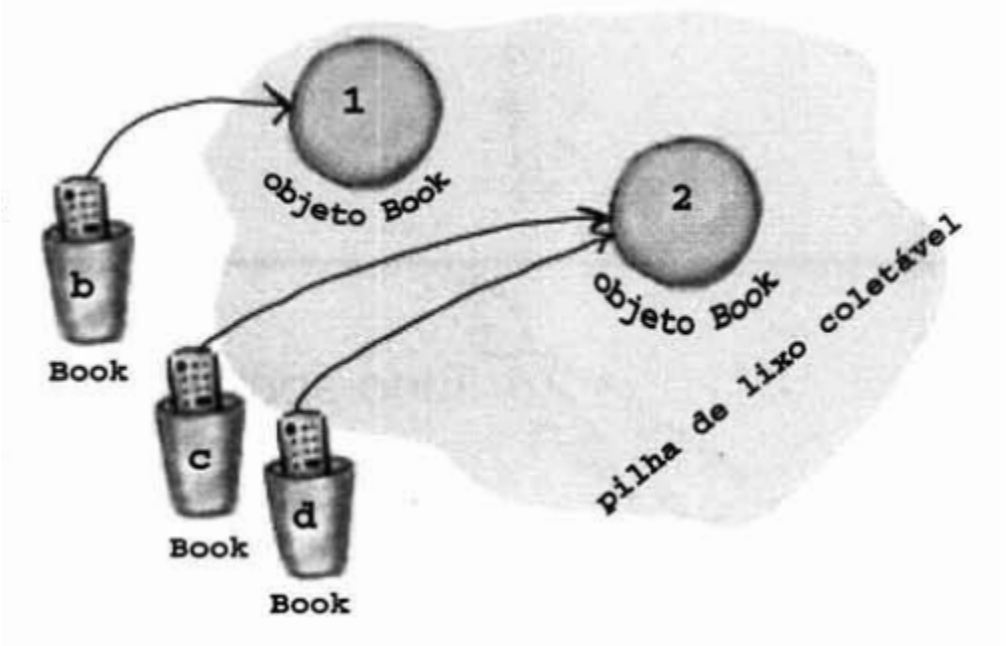
- É uma região da memória onde o java aloca todos os objetos que ele criar.
- Na verdade essa memória **Heap** Java se chama **Pilha de Lixo Coletável**. Quando um objeto é criado, o java alocará espaço na memória **Heap** de acordo com quanto esse objeto específico vai precisar.
- Quando a memória **Heap** fica cheia, o java gerenciará essa memória, apagando os objetos que ele perceber que nunca mais vai ser usado.
- **A vida na pilha de lixo coletável:**
  - Quando dois objetos da mesma classe são criados:

```
Book b = new Book();
Book c = new Book();
```



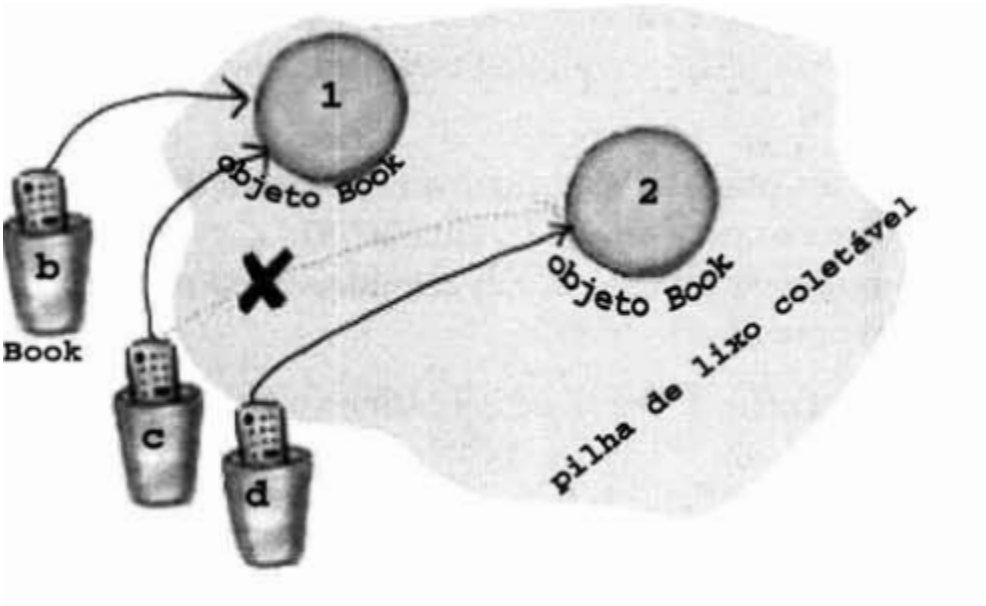
  - Atribuindo o valor de `c` a uma variável `d`:

```
Book d = c;
```



- Atribuindo o valor de `b` na variável `c`:

```
c = b;
```



O erro cometido pelo cara do vídeo foi usar os parentes em um local que não precisava "end", para que o "end" seja executado primeiro que o "or".

O problema foi que o cara do vídeo usou os parentes para executar primeiro o operador "and" do que o operador "or", mais isso não é preciso, pois por padrão o python já executar o "and" primeiro que o "or".

Variavéis

- As variáveis podem ser usadas como **Estado do Objeto** (Variáveis de instância); como **Variáveis Locais** (Variáveis declaradas dentro de um método); como **Argumentos** (Valores enviados para um método pelo código que o chamou); e como **Tipos de Retorno** (Valores retornados ao código que chamou o método).
- Existem dois tipos de variáveis: **Primitivas** e de **Referência** (Referência de objeto).

Primitivas:

Variáveis desse tipo contém valores básicos (`int`, `float`, `boolean`)

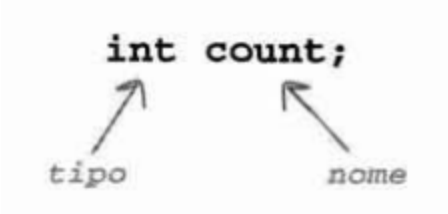
Tamanhos das Variáveis Primitivas:

Em java existem tamanhos diferentes para as variáveis primitivas. Cada variável primitiva possui uma quantidade fixa de bits.

Os tipos de uma variável Primitiva:

- `boolean`: Só recebe `true` ou `false`.
- `char:` Ela é usada para caracteres, ela tem tamanho de `16 bits`, os seus valores podem ter de `0` caracteres até `65535` caracteres.
- **Inteiro:**
  - `byte`: Usado para números inteiros, seu tamanho é de `8 bits`, ou seja, o valor atribuído a essa variável tem que está entre `-128` a `127`.

    - `short`: Usado para números inteiros, seu tamanho é de `16 bits`, ou seja, o valor atribuído a essa variável tem que está entre `-32768` a `32767`.
    - `int`: Usado para números inteiros, seu tamanho é de `32 bits`, ou seja, o valor atribuído a essa variável tem está entre `-2147483648` a `2147483647`.
    - `long`: Usado para números inteiros, seu tamanho é de `64 bits`, ou seja, o valor atribuído a essa variável tem que está entre `-enorme` a `enorme`.
  - **Ponto flutuante:**
    - `float:` Usado para números com casas decimais, seu tamanho é de `32 bits`.
    - `double:` Usado para números com casas decimais, seu tamanho é de `64 bits`.
- **Obs:** O java considera importante o tipo de uma variável. Ele não permitirá que você faça algo bizarro e perigoso como por exemplo, inserir a referência de uma girafa em uma variável coelho.
- Uma variável quando vai ser criada ela precisa ter duas coisas: Um **tipo**, e um **nome**.



▼ Referência de objeto:

- Como o próprio nome diz, essas variáveis contem *referências de objetos*. Ou seja ela contém bits que representa uma maneira de acessar um objeto.
- **Obs:** Não podemos inserir um objeto em uma variável.

Muitas vezes dizemos coisas que da a intender que estamos colocando um objeto em uma variável, exemplo: "Inseri um novo objeto Foo na variável chamada myFoo".

Isso é só um modo de fala, mais na realidade a variável só tem **bits que representam uma maneira de acessar um objeto**.

Podemos também dizer que essa variável tem um **ponteiro que aponta para o objeto**, isso também está certo, o significado é o mesmo do que foi falado mais acima.

- **Obs:** Em uma variável de referência a quantidade de bits não é relevante.
- Usamos o operador ponto . em uma variável de referência para dizer "use o que está antes do ponto para me trazer o que está depois do ponto". Por exemplo:

```
myDog.back();
```

significa "use o objeto referenciado pela variável myDog para chamar o método bark()".

- Criando uma variável referência de objeto:

```
Dog d = new Dog();
d.back();
```

▼ Criando uma Variável de Referência:

- São 3 etapas: Declaração, criação e atribuição de objetos.

1. **Declare uma variável de referência:** Solicita á JVM para alocar espaço para uma variável de referência e nomeia essa variável como `myDog`. A variável de referência será sempre do tipo Dog.

```
Dog myDog = new Dog( );
```

2. **Crie um objeto:** Solicita á JVM para alocar espaço para um novo objeto `Dog` no heap.

```
Dog myDog = new Dog( );
```

3. **Vincula o objeto a referência:** Atribui o nove objeto `Dog` à variável de referência `myDog`.

- Existe alguns nome (chamado **Palavras-Chave**) que não podem ser usadas como nome de uma variável:

boolean	byte	char	double	float	int	long	short	public	private
protected	abstract	final	native	static	stricfp	synchronized	transient	volatile	if
else	do	while	switch	case	default	for	break	continue	assert
class	extends	implements	import	instanceof	interface	new	package	super	this
catch	finally	try	throw	throws	return	void	const	goto	enum

▼ Matriz

- As matrizes sempre serão objetos. Não importa se elas foram declaradas para conter tipos primitivos ou referências de objeto.
- Cada elementos de uma matriz pode ser uma instância de um objeto, ou seja cada posição de uma matriz pode conter um objeto da classe `Dog`, logo um matriz pode conter vários objetos da classe `Dog`, ou até mesmo de outra classe.
- `.length:` Para saber o tamanho de uma matriz basta usar o `.length`.

▼ **Como criar uma matriz com objetos da classe `Dog`:**

1. Primeiro criamos a matriz, nesse caso vamos criar uma matriz de tamanho 3:

```
Dog[] myDogs = new Dog[3];
```

2. Atribuindo um objeto a primeira posição da matriz:

```
myDogs[0] = new Dog();
```

3. Acessando as variáveis de instância e os métodos da classe `Dog`:

```
myDogs[0].name = "Fido";
myDogs[0].bark();
```