**JAVA do zero**

**Tipos primitivos: int, short, float, double, char, long, boolean**

* **Int** numero inteiro;
* **Double** numero fracionado;
* **Float** Similar ao double, mas usa 32 bits. Menos preciso, mas ocupa menos espaço. Exemplo: float altura = 1.75;
* **Char** representa um caractere ;
* **Short** Um inteiro de 16 bits. Exemplo: short ano = 2023;
* **Long** Um número inteiro de 64 bits, usado para valores maiores. Exemplo: long populacao = 7800000000L’
* **Boolean** Valor logico geralmente usado em comparações para um obter um valor true ou false;

**Não primitivos: String, Array (por enquanto)**

**OBS:** Nos tipos não primitivos podemos adicionar diversos atributos vou citar alguns como exemplo.

* **String** Pode armazenar letras, por exemplo seu nome ou um objeto ou algo do tipo

**length():** retorna o numero de caracteres em uma **string;**

**charAt(int index):** retorna o caractere na posição especificada;

**substring(int start, int end):** Retorna uma parte da string entre os indicies,

**start** e **endl**;

* **Array** Estruturas que armazenam multiplos valores do mesmo tipo

**length():** Indica o numero de elementos em uma **array**;

**Arrays.sort():** Metodos para ordenar os elementos de uma **array;**

**Arrays.copyOf():** Cria a uma copia de uma **array**  em um novo tamanho;

**Métodos e parâmetros:**

* **Método**: O método e um bloco de código onde se agrupa determinada função que você deseja que execute, um exemplo seria onde eu teria que printar diversas vezes uma variável no código e ao invés de fazer isso crio um parâmetro com essa função **print**, Exemplo:

Public static void showName() {

System.out.println(“Meu nome e Andre“)

}

Agora ao usar fica assim:

public static void main(String[] *args*) {  
  
 **showName**();  
  
}

e assim vai printar no console o que foi digitado dentro da função especificada no método.

* **Parâmetros**: O parâmetro seria determinada variável que você exige dentro dos parênteses no método, um exemplo seria se a função “**showName()**” exigisse um nome do usuario, seria mais ou menos assim:

Public static void showName(String name) {

System.out.println(“Meu nome e “ + name)

}

Ao usar ficaria assim:

public static void main(String[] *args*) {  
  
 **showPerson**("Andre");  
  
}

e no console printaria o nome digitado dentro dos parênteses no método.

**Laços de repetição:**

Existem 3 tipos de lacos de repetição que seriam o for, while e o do-while, os lacos “repetem” a acao determinada dentro dele com certas condições, vamos conhecer os 3 tipos e como cada condição deve funcionar.

* **For**: Basicamente como o próprio nome já diz para determinada acao acontecer tem que acontecer a seguinte coisa (vamos dar um exemplo no código aqui):
* for (int i = 0; i < 5; i++) {  
   **showYouNameAndAge**("Andre", 24);  
   }  
  }

Aqui Podemos ver que temos a variável “**i**” dentro de for e o valor dela começa no zero, logo após ponto e virgula temos uma condição que seria enquanto “**i”** for menor que 5, ai vem o ponto e virgula e vem a ação que deve ser tomada caso a condição seja verdadeira que no caso seria “**i++**” que seria a variável mais 1 abreviado em código, no caso o console printaria o nome e a idade determinada dentro do parâmetro 5 vezes e após a condição se tornar falsa ele termina essa ação.

O **for** geralmente e usado quando sabes quantas vezes quer que aquela acao seja repetida, então a variável já e pre estabelecida dentro do próprio laco. (e claro que podemos usar ela em diversas ocasiões e não so quando se já tem algo determinado mas para simplificar agora no começo prefiro explicar assim.)

* **While**: O **while (enquanto)** já seria o contrario do **for**, geralmente e usado quando temos uma variável já criada e não sabemos quantas interações serão necessárias no processo, vamos de exemplo:
* public static void main(String[] *args*) {  
    
   int estoqueUnidades = 0;  
    
   while (estoqueUnidades < 10) {  
   estoqueUnidades++;  
   System.out.println("Quantidade atual de unidades: " + estoqueUnidades);  
    
   }  
  }

Aqui criamos a variável **estoqueUnidade** com o valor **0** para representar um estoque, no laco de repetição **while** temos uma condição onde enquanto o estoque não chegar a 10 unidades e ele executa esse seguinte bloco de código:

estoqueUnidades++;  
 System.out.println("Quantidade atual de unidades: " + estoqueUnidades);\

Que representa que o valor do estoque soma +1 e no console printa a quantidade atual de unidades armazenada dele ate que a condição do while não seja mais verdadeira então ele pula fora.

* **Do-While**: Como nome já diz “faca enquanto ...”, então basicamente antes de ele executar o laco de repetição em si ele executa esse código primeiro uma vez e então ele verifica a condição, se ela for true passa se for false para. Exemplo:
* public static void main(String[] *args*) {  
    
   int unidades = 5;  
    
   do {  
   System.out.print("Unidades no estoque: " + unidades);  
   System.out.println();  
   unidades--;  
   } while (unidades >= 0);  
     
  }

A variável **unidades** esta com o valor 5 e o laco de repetição esta dizendo para printar as unidades no estoque e diminuir uma unidade da variável enquanto ela for maior ou igual a 0.

**Objetos**:

Objetos são classes podemos usar de exemplo uma pessoa, carro, celular, comida etc...

Esses **objeto**s possuem **atributos** que podemos dizer que são suas “características”, usando um carro por exemplo temos a cor, ano, cavalos... Se fosse uma pessoa poderíamos usar cor, idade, nome, altura, cpf ...

Vamos ver um exemplo na pratica em código do que é um **objeto** e seus **atributos** :

public class person {  
  
 String name;  
 int age;  
 double **heigth**;

}

Como Podemos ver “person” e a classe que faz referencia a uma pessoa, pode ser um cadastro e etc. Dentro dessa classe “person” temos os seguintes atributos **name, age** e **heigth**, que traduzindo fica nome, idade e altura.

Para setarmos esse nome em uma classe **main**  teríamos que primeiramente criar uma pessoa dentro dessa classe, vamos ao exemplo a baixo:

public class main {  
 public static void main(String[] *args*) {  
  
 person andre = new person();  
}

Como Podemos ver criamos uma classe **main** e para setarmos a classe **person** primeiro chamamos ela com o person e logo após vamos declarar qual sera o nome dessa pessoa que no caso foi andre, depois disso usamos sinal de igual e então usamo o new para representar que esta e uma nova pessoa e então usamos person()para finalizar a criação de uma nova pessoa com a classe **person** junto com seus atributos.

Agora para setar e preencher os atributos vamos usar o seguinte código:

andre.name = "Andre Leonardo";  
andre.age = 24;  
andre.height = 1.70;

Pode ver primeiramente setamos a pessoa com nome desejado e então em sequencia digitamos “.” e digitamos qual atributos gostaríamos de setar, após isso usamos o = para logo após preenchermos o atributo com o que ele pede geralmente, no caso da primeira linha seria o nome que no caso e uma string então uso aspas para digitar ele.

O mais legal disso tudo e que podemos criar diversas pessoas nesse código, por exemplo poderíamos logo abaixo após preencher o código criar uma outra pessoa, vejam o exemplo a baixo:

public class main {  
public static void main(String[] *args*) {  
  
 person andre = new person();  
  
 andre.name = "Andre Leonardo";  
 andre.age = 24;  
 andre.height = 1.70;  
  
 person luciano = new person();  
  
 luciano.name = "Luciano Amorim";  
 luciano.age = 45;  
 luciano.height = 1.69;  
  
 System.out.println("Nome: " + andre.name);  
 System.out.println("Idade: " + andre.age);  
 System.out.println("Altura: " + andre.height);  
 System.out.println();  
 System.out.println("Nome: " + luciano.name);  
 System.out.println("Idade: " + luciano.age);  
 System.out.println("Altura: " + luciano.height);

}

Neste caso o console ficaria assim:

Nome: Andre Leonardo

Idade: 24

Altura: 1.7

Nome: Luciano Amorim

Idade: 45

Altura: 1.69

Process finished with exit code 0

Entao assim finalizando um processo básico de criação de classe dentro de uma outra classe.

**Criando Métodos dentro dos nossos próprio objeto:**

Dentro das nossas classes e possivel criarmos **metodos/funções** próprias da mesma, como pessoa vamos usar o seguinte exemplo: Criamos a pessoa se tudo ocorrer certo após setar podemos criar uma função chamada cadastroRealizado, vamos ver isso na pratica:

public void cadastroRealizado() {  
 System.out.println("Cadastro realizado com sucesso.");  
}

Nesse exemplo dentro da classe **person** criei um método chamado cadastroRealizado,como podemos ver esse método apenas printa na tela “Cadastro realizado com sucesso”, foi apenas um exemplo de como podemos criar um método dentro de uma classe.

Agora vamos para classe main e vamos ver como usar esse método, veja abaixo:

import java.util.Scanner;  
  
public class main {  
 public static void main(String[] *args*) {  
  
 Scanner sc = new Scanner(System.in);  
  
 person andre = new person();  
  
 System.out.println("Digite seu nome: ");  
 andre.name = sc.nextLine();  
 System.out.println("Digite sua idade: ");  
 andre.age = sc.nextInt();  
 System.out.println("Digite sua altura: ");  
 andre.height = sc.nextDouble();  
 System.out.println();  
 andre.cadastroRealizado();

Primeiramente antes de declarar classe main o que fiz foi importar um **scanner** para que ele possa me dar a opção de pedir para que o usuario digite isso, após criar uma classe certinho já declaro esse mesmo **scanner** como **sc** e crio ele como se fosse uma classe normal como vimos anteriormente, logo após temos criação da pessoa **andre**, após isso vemos algo novo aqui que seria de certo modo a interação com o usuario onde peco para que digite seu nome e logo a baixo seto usuario.nome e após o sinal de igual uso o **sc.nextline()** para que o usuario digite seu código no console, logo após faco mesma coisa com **idade** e **altura** (ambos usando a a tipagem que foi declarada), e ai vem o método que criei dentro do **objeto person**, isso ira printar na tela que o cadastro foi realizado, vamos ver isso

Digite seu nome:

Andre

Digite sua idade:

24

Digite sua altura:

1,70

Cadastro realizado com sucesso.

Nome: Andre

Idade: 24

Altura: 1.7

Como podes ver ali onde o usuario digitou quando foi solicitado determinada ação marquei as letra com a cor amarelae logo após finalizar as informações a ação e executada dizendo que o cadastro foi realizado e então assim printado o resto do código, poderíamos fazer isso tudo dentro desse próprio método mas como quero so passar um exemplo simples preferi usar o próprio **main** para printar os atributos.

**Herança**

A **herança** e usada quando queremos estender uma classe, vamos supor que temos uma **pessoa** chamada **Andre Luciano** que bate um ponto de manhã, nesse ponto pede para selecionar qual e sua profissão, Andre trabalha como **metalúrgico** então ele seleciona a opção e ela retorna para o sistema que **Andre Luciano está trabalhando como metalúrgico**, agora temos outra **pessoa** chamada **Leonardo Amorim** e ele trabalha como **programador** nessa mesma empresa, então ele faz o procedimento e em vez de selecionar **metalúrgico** ele seleciona **programador** e então retorna para o sistema como **Leonardo esta trabalhando de programador**, porem temos um problema ai hipoteticamente se **Andre** decide botar que trabalha de programador e vice e versa o que podemos fazer para que isso não aconteça ? **Ai entra a herança**, pois iremos criar duas classes chamadas de **devlope**r e outra como **mettalurgical**, então vamos estender a pessoa **Andre** para **mettalurgical** e **Leonardo** para **developer** e assim não tem como um tentar “mentir” sua profissão, vamos ver um exemplo em código para simplificar, veja abaixo:

* **Classe person:**

A classe **person** esta representando a pessoa, fiz algo bem simples para entender apenas com o **atributo name** para se identificar, veja:

public class person {  
  
 String name;  
 String job;  
  
 public void cadastroRealizado() {  
 System.out.println("Login realizado com sucesso.");  
 System.out.println("Nome: " + name);  
 }

**OBS: Fiz um método para mostrar que o cadastro foi realizado e ele conferir o nome.**

* **Classe dev:**
* public class dev extends person{  
    
   public void programar() {  
   System.out.println("Esta trabalhando como programador...");  
   }  
    
  }

Como pode ver a **classe dev** estende a **classe person** então podemos usar os atributos da classe.

* **Classe mettalurgical:**

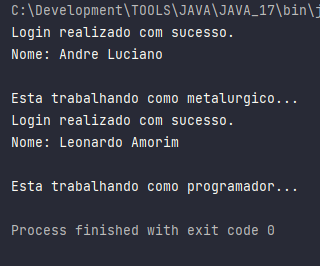
A **classe mettalurgical** mesma coisa que a **dev**:

public class metallurgico extends person {  
  
 public void trabalhando() {  
 System.out.println("Esta trabalhando como metalurgico...");  
 }  
  
}

Agora vamos para a classe **main** para ver como isso ira funcionar:

public class main {  
 public static void main(String[] *args*) {  
  
 metallurgico person1 = new metallurgico();  
  
 person1.name = "Andre Luciano";  
 person1.job = "metallurgical";  
 person1.cadastroRealizado();  
 System.out.println();  
 person1.trabalhando();  
  
 dev person2 = new dev();  
  
 person2.name = "Leonardo Amorim";  
 person2.job = "Developer";  
 person2.cadastroRealizado();  
 System.out.println();  
 person2.programar();  
  
 }  
}

Na classe **main** como pode ver em vez de criar uma “**person**” criei na verdade um **metalúrgico**, você se pergunta porque fazer isso ? isso acontece pois se a classe **person** estendesse uma das **classes de trabalhos qualquer** funcionario cadastrado poderia selecionar quaisquer trabalhos, ai esta resolvido esse problema, vamos ver como fica no console:

****

**Polimorfismo**

O polimorfismo basicamente e usar uma linha de código em varias classes com seus diferentes atributos porem cada um com sua características únicas da própria classe.

Vamos continuar usando o exemplo da ultima explicação, como cadastro. Exemplo a baixo:

Classe person:

public class person {  
  
 String name;  
 String job;  
  
 public void cadastroRealizado() {  
 System.out.println("Login realizado com sucesso.");  
 System.out.println("Nome: " + name);  
 }  
  
 public void consultaCadastro() {  
 System.out.println("Nome: " + name);  
 System.out.println("Nome: " + job);  
 }  
 //polimorfismo  
 public void trabalhando(String *cargo*) {  
  
 if(*cargo* == "developer") {  
 System.out.println("Voce nao pode se cadastrar em um cargo pois nao e um funcionario");  
 } else if (*cargo* == "metalurgico") {  
 System.out.println("Voce nao pode se cadastrar em um cargo pois nao e um funcionario");  
 } else {  
 System.out.println("Voce nao pode se cadastrar em um cargo pois nao e um funcionario");  
 }  
 }  
}

Como podem aqui temos um método dentro do objeto person chamado “trabalhando”, neste caso como essa pessoa e apenas uma pessoa sem cargo e sem login ao ela preencher o parâmetro estabelecido nesse método todas as respostas vao para “você não pode se cadastrar...” pois essa pessoa não e um funcionário, aqui e como se fosse o método principal de quando se entra em um sistema e não se tem login.

Classe Dev

public class dev extends person {  
  
 *@Override* public void trabalhando(String *cargo*) {  
  
 if (*cargo* == "programador") {  
 System.out.println("Voce foi cadastrado como desenvolvedor");  
 } else if (*cargo* == "metalurgico") {  
 System.out.println("Voce se cadastrou para o cargo incorreto");  
 } else {  
 System.out.println("Cargo invalido.");  
 }  
  
 }  
}

Aqui como pode se ver removi os métodos anteriores e fiz algo mais “real” para fácil entendimento, para usarmos o polimorfismo primeiro precisamos fazer um @Override antes do método ser criado, então após isso eu exigi um parâmetro do tipo string que se chama cargo, logo após uso uma condição para que certas acoes sejam tomadas depois de ser declarado parâmetro onde uso if e então declaro a condição que quer dizer se cargo for igual a programador ele vai fazer tal condição, que no caso seria cadastro realizado, agora usando else if uso para dizer que se cargo for igual metalúrgico ele ira dizer que se cadastrou no cargo errado pois essa e uma classe de devs, por ultimo usando o else pois suponhamos que a empresa tem apenas esses dois cargos o sistema ira retornar cargo invalido.

Classe metalúrgico

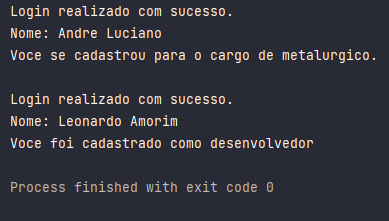
public class metallurgico extends person {  
 *@Override* public void trabalhando(String *cargo*) {  
  
 if (*cargo* == "metalurgico") {  
 System.out.println("Voce se cadastrou para o cargo de metalurgico.");  
 } else if (*cargo* == "programador") {  
 System.out.println("Voce se cadastrou no cargo incorreto.");  
 } else {  
 System.out.println("Cargo invalido");  
 }  
  
 }  
}

Como pode ver e semelhante a classe de então a explicação e a mesma.

Classe main

public class main {  
 public static void main(String[] *args*) {  
  
 metallurgico person1 = new metallurgico();  
  
 person1.name = "Andre Luciano";  
 person1.job = "metalurgico";  
 person1.cadastroRealizado();  
 person1.trabalhando(person1.job);  
 System.out.println();  
  
 dev person2 = new dev();  
  
 person2.name = "Leonardo Amorim";  
 person2.job = "programador";  
 person2.cadastroRealizado();  
 person2.trabalhando(person2.job);  
  
 }  
}

Na classe main comecamos normalmente como antes onde crio a classe e então declaro os parâmetros das variáveis normalmente mas logo após cadastrorealizado usamos o método trabalhando e declaramos o que foi pedido que no caso foi uma variável do tipo string que seja o nome do cargo. Mesma coisa ocorre com a person2.



Agora no console apresenta que o login foi realizado com sucesso, apresenta nome do funcionário e diz o cargo em que se cadastrou que foi declarado no main. E isso e o polimorfismo que basicamente são métodos com a mesma função mas com características únicas da sua própria classe. Nessa explicação usei booleans que e super importante aprender que são comparativos, no if poderíamos simplificar para || em vez de fazer um else if porem quis deixa bem claro o que estava fazendo.