**Java**

**Fundamentos**

**Variáveis e constantes**

Para de definir uma variável em Java se usa primeiro o tipo da variável, depois o nome, e atribui um valor para a variável.

tipo nome = valor;

Exemplo:

int raio = 3;

Nesse caso o tipo da variável será inteira.

double raio = 3.4;

Nesse caso o tipo da variável será real . Para definir uma constante em Java, basta colocar a palavra final antes do tipo, fazendo isso a variável se tornará uma constante e não poderá ser alterada.

final tipo nome = valor;

Exemplo:

final double PI = 3.14159;

Aqui foi criado uma constante do tipo double com o nome “PI”, e o padrão de conversão em Java é declarar uma constante com o nome tudo em maiúscula.

**Inferência de Tipos**

Inferência de tipo é quando você não define o tipo da variável no momento de declarar ela, ao invés de usar a palavra int , doble ou qualquer outra palavra reservada para definir o tipo da variável, se usa a palavra reservada var, fazendo assim com que o Java infira pelo tipo da variável de acordo com o valor que você atribuir a ela, ou seja, vai definir o tipo da variável pelo valor que foi atribuído á ela.

var nome = valor;

Exemplo:

var a = 4.5;

var b = “Olá”;

A variável a vai ser do tipo double(real) e a variável b vai ser do tipo String(texto). Você não pode mudar o tipo da variável em Java, declarando ela usando o var, ou definindo o tipo dela, se você declarar que uma variável é do tipo int, você não pode atribuir pra ela um valor double ou String, da mesma forma que se você declarar uma variável do tipo String, você não pode atribuir a ela depois um valor do tipo int ou double, mas se você declarar uma variável do tipo double, você pode atribuir um valor inteiro para ela, pois os números inteiros cabem dentro dos números reais, mas ela será convertida para real adicionando o “.0” no final:

double a = 4;

O resultado de a vai ser 4.0 e não apenas 4. Você também pode declarar a variável e depois inicializar ela atribuindo um valor para ela.

double a;

a = 4.5;

Mas com o var não tem como fazer isso, somente declarando com o tipo explicitamente, pois o Java precisa do valor da variável para definir o tipo dela com o var.

**Tipos Primitivos**

Em Java tem 8 tipos primitivos, sendo 6 do tipo numérico.

**Numéricos**

**Inteiros:**

**byte -> 1**

**short -> 2 vezes o tamanho do byte**

**int -> 4 vezes o tamanho do byte**

**long-> 8 vezes o tamanho do byte**

**Capacidade que cada um armazena:**

**Sendo o número negativo sempre um número a mais, e o número positivo sempre um número a menos, pois o 0 é considerado no eixo positivo.**

**byte -> -128 até +127**

**short -> 32767**

**int -> 2bilhões e alguma coisa**

**long-> um número extremamente grande**

**Reais :**

**float -> 4**

**double -> 8 vezes o tamanho do byte**

**1 Byte = 8 bits**

O padrão é sempre usar o int para inteiros e o double para os reais, se precisar de usar um valor inteiro maior do que cabe no int, use o long.

**Caractere:**

**char -> Representa uma letra, e é delimitado por aspas simples(‘’).**

**Ex:**

**char letra = ‘A’;**

**Booleano:**

**Representa verdadeiro ou falso.**

**Ex:**

**boolean verdadeiro = true;**

**boolean verdadeiro = false;**

**Classes e métodos**

**Métodos**

métodos é um comportamento, é aquele que é chamado com os “( )”, como uma função, podendo ter ou não parâmetros. Para criar um método você deve definir o tipo dele (int, double, String...) que será retornado, ou caso não retorne nada deve ser declara com o void, e se for criar parâmetros para o método, deve definir o tipo do parâmetro também.

Exemplo:

int soma(int a, int b) {

...

return a + b;

}

Ou sem passar parâmetros:

int soma() {

...

return ...;

}

E dentro do método usa o return para retorna algo dentro do método.

**Construtores**

O construtor usa a palavra reservada “new” seguido pelo nome do construtor, ex:

Produto p1 = new Produto();

Um construtor padrão é aquele que não recebe nenhum parâmetro e é definido de forma implícita pelo Java, mas da mesma forma que o método pode receber parâmetros, o construtor também pode receber parâmetros, ao se definir um novo construtor de forma explicita recebendo parâmetros, o construtor padrão que é criado pelo Java de forma implícita deixa de existir, e terá que criar um novo construtor de forma explicita que não recebe parâmetros para usá-lo sem os parâmetros. Ao se criar uma nova classe construtora, não se passa o tipo que será retornado como por exemplo(int, double, String... ou void para não retornar nada), pois o java já sabe que o retorno do construtor é uma instancia do tipo do nome da classe.

Exemplo:

class Produto {

// Construtor implícito

}

Aqui foi criado um construtor de forma implicita, ou seja, foi criado pelo próprio Java e não possui parâmetros, para usar esse construtor é so usar a palavra reservada new seguido do nome da classe construtora:

Produto p1 = new Produto();

class Produto {

Produto (int a) {

// Construtor explicito

}

}

Produto p1 = new Produto(3);

Aqui foi criado uma nova variável do tipo Produto através da classe construtora Produto, e passado o valor do parâmetro como o número inteiro 3, pois esse construtor foi declarado de forma explícita com parâmetros.

class Produto {

Produto (int a) {

// Construtor explicito com os parâmetros

}

Produto () {

// Construtor explicito sem os parâmetros

}

}

Produto p1 = new Produto(3);

Produto p2 = new Produto();

Aqui foi criado uma nova variável do tipo Produto através da classe construtora Produto, e passado o valor do parâmetro como o número inteiro 3, pois esse construtor foi declarado de forma explícita com parâmetros, mas também podemos usar o construtor sem os parâmetros, pois foi criado de forma explicita um construtor sem os parâmetros.

O retorno dos construtores do exemplo da classe Produto, será do tipo Produto. O construtor é um método especial.

Restrições do construtor:

1. Nome da Classe = Nome do Construtor
2. Um construtor NÂO TEM RETORNO

**Membros de Classe vs Instância**

Ao de criar uma classe, e chamar ela com a palavra reservada new, você estará criando uma nova instancia(Objetos), reservando a quantidade de memoria daquela instancia, de acordo com oque tem na classe, por exemplo, se criarmos a classe Data:

public class Data {

int dia;

int mes;

int ano;

}

Ao se criar uma nova instância do tipo Data, ele irá criar uma cópia do valor do atributo para cada nova instância criada com a palavra new, e assim vai acontecer para cada nova instancia do tipo Data que for criado.

public class Data {

static int dia;

int mes;

int ano;

}

Data p1 = new Data();

Ao se usar a palavra reservada static antes do tipo do atributo dentro da classe, você estará dizendo que o atributo pertencerá a classe, automaticamente o valor que está associado a instância já não vai mais estar associado a ela, e sim vai estar associado ao própria classe, e ele gerará apenas uma única copia para todos as novas variáveis que você criar, no caso objetos. Ou seja, se você dizer p1.dia = 30, todas os outros objetos do tipo data terá o dia igual a 30, e se você declarar outo objeto com o nome p2, e dizer que p2.dia = 10, tanto p1.dia quanto o p2.dia será igual a 10.

**Atribuição: Valor vs Referência**

Atribuição por Valor para tipos primitivos

Atribuição por Referência para objetos

Atribuição por Valor

Ao se atribuir o valor de uma variável para outra variável, exemplo:

int a = 8;

int b = a;

Você estará criando uma copiar de a em b, só que cada um com seu espaço reservado na memória de forma individual, se eu alterar o valor de a na linha de baixo para 10, o valor de b continuara valendo 8, pois ela já recebeu o primeiro valor de a.

Atribuição por Referência

Já se você criar um objeto, e atribuir o valor desse objeto para outro objeto, por exemplo:

Data d1 = new Data();

Data d2 = d1;

Eles vão compartilhar o mesmo espaço de memória, fazendo assim com oque você altere em d1, também seja alterado em d2 e vise e versa. Exemplo:

d1.dia = 10;

d2.dia também será igual a 10.

se você fazer o d2 receber um novo objeto Data, ele vai apontar para outra área de memoria, fazendo assim com que oque você altere em d2 não seja alterado em d1 e vice e versa.

Exemplo:

d2 = new Data();

d2.dia = 23;

O valor de d2.dia será igual a 23 e o valor de d1.dia continuará igual a 10.