Java

Iniciaremos primeiramente com o básico “Hello World”

```
import java.util.Scanner;

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

Explicando a função de cada linha:

```
import java.util.Scanner;
```

Chamada por muitos como a linha do cabeçalho, antes do código, das classes e do método será onde importaremos as bibliotecas, que é o que utilizaremos para executar algumas funções não presentes nativamente em Java.

No caso do exemplo acima a biblioteca importada foi a do Scanner, que é utilizada para receber entradas do usuário que será utilizada mais a frente.

```
public class MyClass{
    ...
}
```

Todo código que roda em Java deve estar dentro de uma classe, a qual acima nomeamos como `MyClass`. Uma classe por convenção, sempre deve se iniciar com uma letra maiúscula

> **Obs:** O nome da classe pública deve ser SEMPRE o mesmo do nome do arquivo, que nesse caso seria "MyClass.java".

```
public static void main(String []args){
    ...
}
```

Sempre que criamos um programa Java que será executado diretamente, utilizamos o método `public static void main(String [] args){}` Todo o código que estiver dentro desse método será executado. Vale destacar que nem todo código Java utiliza o método `main`, apenas aqueles que têm um ponto de entrada para serem executados.

Printar informações no Console/Terminal

```
System.out.println("Hello World");

// Saída:
/* Hello World */
```

System é uma classe embutida no Java que contém membros úteis como o `out` , que é a abreviação de “output”, que significa em português “Saída”.

O `println()`, é a abreviação de “print line”, que é usado para “printar” um valor no terminal ou console.

Toda a String (Texto) em Java deve estar entre aspas duplas “”.

> **Obs**: Devemos ter atenção também de que todo o código escrito em Java deve ser finalizado com “;”, caso seja esquecido o código apresentará um erro e provavelmente não rodará.

```
System.out.println(3);
System.out.println(232);
System.out.println(1233211);
```

```
// Saída:

/*
3
232
1233211
*/
```

Também é possível printar números utilizando a função `println`, é apenas não fazer a utilização do aspas duplo.

```
System.out.println(3+3);// Soma
System.out.println(3-3);// Subtração
System.out.println(2*5);// Multiplicação
System.out.println(4/2);// Divisão
System.out.println(25%4);// Resto da divisão
System.out.println(Math.pow(5,2));// Potenciação

// Saída:

/*
6
0
10
2
1
25
*/
```

Também existe a função `print()`, que é muito similar a função `println()`, a única diferença é que no `println()` existe uma quebra de linha no fim da saída e o `print()` não.

```
System.out.print("Hello World!");
System.out.print("Vai aparecer na mesma linha");
```

```
// Saída:
/* Hello World!Vai aparecer na mesma linha */
```

Outra função é a `printf()` , que é herdada do C.

```
System.out.printf("%s\n", "Hello World!");

// Saída:
/* Hello World! */
```

A função `printf()` segue o seguinte formato:

```
System.out.printf(formato:"", argumentos:"");
```

Nos formatos podemos utilizar:

- `%d` para decimais.
- `%s` para strings (Textos).
- `%f` para float (Números com vírgula).
- `%b` para booleano (true ou false).
- `%c` para caracteres.

Além disso também podemos utilizar o `\n` que serve para adicionar uma quebra de linha.

Exemplo:

```
int idade = 25;
String nome = "João";
double salario = 2500.50;

System.out.printf("Nome: %s, Idade: %d, Salário: %.2f%n", nome,
idade, salario);
```

```
// Saída:
/* Nome: João, Idade: 25, Salário: 2500.50 */
```

Variáveis:

Tipos de variáveis:

- `String` → Utilizadas para guardar textos, seus valores devem estar entre aspas duplas "".
- `int` → Utilizadas para guardar números inteiros, sem decimais.
- `float` → Utilizadas para guardar números "com vírgula", com decimais.
- `double` → Mesma função do "float", porém pode armazenar o dobro de variáveis.
- `char` → Utilizadas para guardar caracteres únicos, como 'a' ou 'B'. Valores char devem estar entre aspas simples ''.
- `boolean` → Utilizadas para guardar valores com 2 estados: true ou false.

Declarando Variáveis:

Sintaxe:

```
tipodaVariavel nomeDaVariavel = valorDaVariavel;
```

Exemplo:

```
String nome = "Pedro";
/* Foi criada a variável "nome" e foi definida
para ela o valor "Pedro".*/
System.out.println(nome);
// Foi "printada" a variavel "nome".

int meuNum = 15;
System.out.println(meuNum);
```

Variáveis também podem ser representadas da seguinte forma:

```
int meuNum2;
meuNum2 = 15;
System.out.println(meuNum2);
```

Note que se você definir um novo valor para uma variável existente, esse valor vai sobrescrever o valor anterior.

```
int meuNum3 = 15;
meuNum3 = 20; // O valor de meuNum3 agora é 20
System.out.println(meuNum3);
```

Combinar textos com variáveis:

Para combinar textos com variáveis na função `println()` , utilizamos o sinal “+”:

```
String nome2 = "Joao";
System.out.println("Oi " + nome2);
```

Também é possível utilizar o sinal “+” para unir uma variável a outra:

```
String primeiroNome = "Pedro ";
String segundoNome = "Alves";
String nomeCompleto = primeiroNome + segundoNome;
System.out.println(nomeCompleto);
```

Para valores numéricos, o "+" funciona como um operador matemático normalmente:

```
int x = 5;
int y = 6;
System.out.println(x + y);
// Vai printar "11", valor de 5 + 6;
```

Para declarar mais de uma variável do mesmo tipo, podemos declarar todas na mesma linha utilizando a vírgula ",":

```
int x1 = 5, y1 = 6, z = 50;
System.out.println(x1 + y1 + z);
```

Assim como podemos declarar um único valor para várias variáveis:

```
int x2, y2, z1;
x2 = y2 = z1 = 50;
System.out.println(x2 + y2 + z1);
```

Constantes:

Declaração de constantes:

Se você não quer sobrescrever um valor existente, deve-se utilizar a palavra final (Isso vai declarar a variável como "final" ou "constante", o que significa que não é possível alterar seu valor.).

```
final int meuNum4 = 15;
/* Caso escreva agora "meuNum4 = 20;", o código não irá
rodar e irá apresentar um erro:
"cannot assign a value to a final variable" */
System.out.println(meuNum4);
```

Regras para se declarar uma variável ou constante:

1. Os nomes para as variáveis são case-sensitive("minhaVar" e "MinhaVar" são variaveis diferentes);
2. Nomes deve começar com uma letra, um caractere sublinha ou underline (_) ou o símbolo cifrão ($). Os caracteres subsequentes também podem ser algarismos;
3. Não utilizar caracteres especiais, como acentos, símbolos (?/:@#.),ç entre outros;

4. As letras podem ser maiúsculas ou minúsculas;
5. Não podem ser utilizadas palavras reservadas(Ex: int, boolean).

Comandos de leitura de entrada em Java:

É necessário a utilização no inicio do código o seguinte comando para fazer a importação da biblioteca necessária:

```
import java.util.Scanner;
```

Depois disso, dentro do seu código você deve utilizar o comando:

```
Scanner nomeDoScanner = new Scanner(System.in);
```

para poder criar seu scanner dentro do código.

Exemplo:

```
Scanner kb = new Scanner(System.in);
```

Para utilizar o scanner basta escrever o seguinte comando:

```
float read = kb.nextFloat();
System.out.println("Seu float é: " + read);
```

ou

```
int read2 = kb.nextInt();
System.out.println("Seu inteiro é: " + read2);
```

ou

```
String read3 = kb.nextLine();
System.out.println("Sua String é: " + read3);
```

Operadores Relacionais:

Sempre retornam true ou false.

- == → igual.
- != → diferente.
- > → maior que.
- >= → maior que ou igual.
- < → menor que.
- <= → menor que ou igual.

Operadores lógicos:

- ! → Negação
- || → ou
- && → e
- ^ → se forem diferentes (Disjunção exclusiva)

Expressões booleanas:

Expressões que retornam como resultado valores booleanos: true ou false.

```
int x = 10;
int y = 9;
System.out.println(x > y) ;
// Saída: true, pois 10
// é maior que 9
```

Operadores Aritméticos(Provavelmente não cai em prova do 1 período):

```
System.out.println(3--);
// Subtrai 1 do valor da
```

```
System.out.println(3++);
// Adiciona 1 ao valor da variável
```

Operadores de Atribuição(Provavelmente não cai em prova do 1 período):

- x = 5 equivalente a: x = 5
- x += 3 equivalente a: x = x + 3
- x -= 3 equivalente a : x = x - 3
- x *= 3 equivalente a: x = x * 3
- x /= 3 equivalente a: x = x / 3
- x %= 3 equivalente a: x = x % 3

Operações bit a bit (bitwise):

- x &= 3 equivalente a: x = x & 3
- x |= 3 equivalente a: x = x | 3
- x ^= 3 equivalente a: x = x ^ 3
- x >>= 3 equivalente a: x = x >> 3
- x <<= 3 equivalente a: x = x << 3

Identificadores:

Todas as variáveis devem ser identificadas com um nome único. Esses nomes únicos são chamados de identificadores (ou "identifiers"). Os identificadores podem ser nomes curtos, como `x` e `y`, ou nomes mais descritivos, como `idade` ou `volumeTotal`. É recomendado o uso de nomes descritivos para tornar o código mais compreensível.

Tipos de Dados:

Como já explicado anteriormente, uma variável em Java deve ter seu tipo definido. Esses tipos podem ser divididos em dois grupos principais:

Tipos de Dados Primitivos:

Incluem: `byte`, `short`, `int`, `long`, `float`, `double`, `boolean` e `char`.

- `byte` = 1 byte → números inteiros de -128 até 127.
- `short` = 2 bytes → números inteiros de -32.768 até 32.767.
- `int` = 4 bytes → números inteiros de -2.147.483.648 até 2.147.483.647.
- `long` = 8 bytes → números inteiros de -9.223.372.036.854.775.808 até 9.223.372.036.854.775.807.
- `float` = 4 bytes → números de ponto flutuante, com precisão suficiente para armazenar de 6 a 7 dígitos decimais.
- `double` = 8 bytes → números de ponto flutuante, com precisão suficiente para armazenar até 15 dígitos decimais.
- `boolean` = 1 bit → armazena valores verdadeiros ou falsos.
- `char` = 2 bytes → armazena um único caractere/letra ou valores da tabela ASCII.

Tipos de Dados Não Primitivos (Tipos de Referência):

Incluem: `String`, `Arrays` e classes.

Eles são chamados de tipos de referência porque referem-se a objetos. A principal diferença entre tipos primitivos e tipos não primitivos é:

1. Tipos primitivos são predefinidos em Java, enquanto os não primitivos são criados pelo programador (exceto `String`).
2. Tipos não primitivos podem referir métodos para realizar operações, enquanto os primitivos não.
3. Tipos primitivos sempre têm um valor, enquanto os não primitivos podem ser nulos (`null`)

4. Tipos primitivos começam com letras minúsculas, enquanto os tipos não primitivos começam com letras maiúsculas.

Tamanho de uma string:

Uma string é na verdade um objeto, no qual contem métodos que podem que podem realizar certas operações. Por exemplo, temos o tamanho da `String`, que pode ser encontrado utilizando a função "`length()`":

```
String txt1 = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
System.out.println("O tamanho da string é: " + txt.length());
```

Outros comando com `Strings`:

```
String txt2 = "Hello World";
System.out.println(txt.toUpperCase());
// Saída: "HELLO WORLD"
System.out.println(txt.toLowerCase());
// Saída: "hello world"
```

Encontrando algo em uma `String`:

```
String txt3 = "Please locate where 'locate' occurs!";
System.out.println(txt.indexOf("locate"));
// Saída: 7
```

Concatenação de uma String:

O operador pode ser usado entre `Strings` para conseguir combina-las, assim como com os outros tipos de variáveis. Isso é chamado concatenação.

```
String primeiroNomee = "João";
String segundoNomee = "Pedro";
System.out.println(primeiroNome + " " + segundoNome);
// saída: João Pedro
```

Note que foi adicionado um texto vazio (" ") para criar um espaço entre o primeiro e o segundo nome na saída.

Você também pode utilizar o "`concat()`" para concatenar duas `Strings`:

```
String primeiroNome = "João ";
String segundoNome = "Pedro";
System.out.println(primeiroNome.concat(segundoNome));
```

> **Obs:** Java usa o operador "+" para adição e concatenação. Números são somados e `Strings` são concatenadas.

Caracteres especiais em `Strings`:

Em Java, precisamos utilizar caracteres especiais para representar aspas simples, aspas duplas e a barra invertida dentro de uma string, pois uma string é escrita entre aspas duplas. Isso significa que, se tentarmos usar esses caracteres diretamente, será gerado um erro, já que o compilador pode interpretá-los de maneira incorreta. Para evitar esse problema, utilizamos a barra invertida (`\`) como um caractere de escape.

\'	‘
\"	“
\\	\

A sequência `\"` insere uma aspas dupla em uma `string`:

```
String txt = "Nós somos assim chamados
\"Vikings\" vindos do norte.";
```

A sequência `\'` insere uma aspas simples em uma `string`:

```
String txt = "It\'s alright.";
```

A sequência `\\` insere uma barra invertida em uma `string`:

```
String txt = "O caractére \\ é chamado de barra invertida.";
```

Outras sequências comuns que são válidas em Java são:

\n	Nova linha.
\r	Carriage Return (Ato de voltar ao inicio da linha seguinte, possui basicamente a mesma função do \n, porém é uma forma de utilização mais antiga.).
\t	Tab (Funciona da mesma forma que a tecla "tab" em um editor de texto. Adiciona um espaço grande).
\b	Backspace (Funciona da mesma forma que a tecla "Backspace" do teclado, ou a tecla de apagar).
\f	Form Feed (Serve como "Quebra de página", utilizado para indicar o fim de uma página ou documento e vai para o inicio de outro).

Fundição de tipos:

Aumentando (Automaticamente): Convertendo um tipo menor para um tipo de maior tamanho:

`byte` → `float` → `char` → `int` → `long` → `float` → `double`

```
int meuInt = 9;
double meuDouble = meuInt;
// Fundição automática: int para double.

System.out.println(meuInt);     // Saída: 9
System.out.println(meuDouble);  // Saída: 9.0
```

Diminuindo(Manualmente): Convertendo um tipo maior para um tipo menor tamanho:

`double` → `float` → `long` → `int` → `char` → `short` → `byte`

```
double meuDouble1 = 9.78d;
// Obs: O sufixo "d" no final do número serve para deixar
//literal ao compilador que o número se trata de um double
//e evitar possiveis erros de compilação.
int meuInt = (int)meuDuble;

System.out.println(meuDouble);      // Saída: 9.78
System.out.println(meuInt);         //Saída: 9
```

Matemática em Java:

`Math.max(x,y)`:

A função `Math.max(x,y)` é utilizada para encontrar o maior valor de x e y:

```
Math.max(5,10);
// saída: 10
```

`Math.min(x,y)`:

A função `Math.min(x,y)` é utilizada para encontrar o menor valor de x e y:

```
Math.min(5,10);
// saída: 5
```

`Math.sqrt(x)`:

A função `Math.sqrt(x)` é utilizada para mostrar a raiz quadrada de x:

```
Math.sqrt(64);
// saída: 8.0
```

`Math.abs(x)`:

A função serve para retornar o valor absoluto (positivo) do valor de x:

```
Math.abs(-4.7);
// saída: 4.7
```

`Math.random()`:

Retorna um número aleatório entre 0.0 (Inclusivo), e 1.0 (Exclusivo):

```
Math.random();
// saída: Número aleatório entre 0.0 e 1.0
```

Para ter mais controle sobre o número aleatório, por exemplo, se você só quer um número aleatório entre 0 e 100, você pode usar a seguinte fórmula:

```
int numAleatorio = (int)(Math.random() * 101);
// 0 a 100
```

`If` ... `Else` em Java

`if`:

Utiliza-se o `if` para especificar um bloco de código a ser executado se a condição for verdadeira.

```
if (condicao) {
    // Bloco de código a ser executado
    // se a condição for verdadeira
}
```

> **Obs:** O `if` é utilizado em letras minúsculas. Se forem utilizadas letras maiúsculas será gerado um erro em seu código.

`else`:

Utiliza-se para especificar o bloco de código a ser executado caso a condição seja falsa.

```
if (condicao) {
    // Bloco de código a ser executado
    // se a condição for verdadeira
} else {
 // Bloco de código a ser executado
 // se a condição for falsa
 }
```

`else if`:

Utiliza-se para especificar uma nova condição se a primeira condição for falsa.

```
if (condicao1) {
    // Bloco de código a ser executado
    // se a condição 1 for verdadeira
} else if (condicao2) {
 /* Bloco de código a ser executado
 // se a condição 1 for falsa e a
 // condição 2 for verdadeira
 } else {
     /* Bloco de código a ser executado
     // se a condição 1 for falsa e a
     // condição 2 for falsa
 }
```

Exemplo:

```
int hora = kb.nextInt();
if (6 < hora && hora < 12){
    System.out.println("Bom dia");
} else if (12 < hora && hora < 18){
    System.out.println("Boa tarde");
} else if(18< hora && hora < 00){
    System.out.println("Boa noite");
} else {
```

```
    System.out.println("Boa madrugada");
}

// Se a hora maior que 6 e menor que 12, saída: Bom dia
// Se a hora maior que 12 e menor que 18, saída: Boa tarde
// Se a hora maior que 18 e menor que 00, saída: Boa noite
// Se todos anteriores forem falsos, então, saída: Boa madrugada
```

Forma curta de se escrever `if` ... `else` :

Existe também a forma curta do `if else`, que é conhecida como operador ternário, pelo fato que consiste em 3 operandos.

Pode ser usado para substituir múltiplas linhas de código com uma única linha, e geralmente usado para substituir utilizações simples de `if else`:

```
variavel = (condicao) ? expressaoVerdade : expressaoFalsa;
```

Invés de escrever:

```
int hora = 20
if (hora < 18) {
    System.out.println("Bom dia.");
} else {
    System.out.println("Boa noite.");
}
```

Pode-se escrever:

```
Int hora = 20
String resultado = (hora < 18) ? "Bom dia." : "Boa noite.";
System.out.println(resultado);
```

`Switch` :

Invés de escrever vários if..else, você pode usar a declaração switch.

A declaração switch seleciona um de vários blocos de código para serem executados:

```
switch(expressao) {
    case x:
        // bloco de código
    break;
    case y:
        // bloco de código
        break;
    dafault:
        // bloco de código
}
```

- O valor da expressão é comparado com os valores de cada case. Caso os valores sejam iguais, o bloco de código associado é executado.
- Quando o compilador java atinje o break, isso para a execução do switch. Assim parando a execução de mais códigos e testes de case dentro do bloco.
- Quando uma correspondência da expressão com o case é encontrada e o trabalho é feito, é hora de parar, pelo fato que não existe mais nada que seja necessário de testar.

```
int dia = 4;
switch(dia){
    case 1:
        System.out.println("Segunda");
        break;
    case 2:
        System.out.println("Terça");
        break;
    case 3:
        System.out.println("Quarta");
        break;
    case 4:
        System.out.println("Quinta");
```

```
        break;
    case 5:
        System.out.println("Sexta");
        break;
    case 6:
        System.out.println("Sábado");
        break;
    case 7:
        System.out.println("Domingo");
        break;
}
        // Saída: "Quinta" (dia 4)
```

O default especifica o bloco de código a ser executado caso não haja nenhuma correspondência com os "case".

```
int dia = 4;
switch(dia){
    case 6:
        System.out.println("Hoje é Sábado");
        break;
    case 7:
        System.out.println("Hoje é Domingo");
        break;
    default:
        System.out.println("Ansioso para o fim de semana")
}
// Saída: "Ansioso para o fim de semana".
```

Loop em Java utilizando `While`:

Um loop consegue executar um bloco de código enquanto a condição especificada for verdadeira, loops são muito úteis pelo fato de que eles economizam tempo, reduzem erros e tornam o código mais legível.

Um loop `while` roda um bloco de código enquanto a condição especificada for verdadeira.

```
while(condicao) {
    // Bloco de código a ser executado
}
```

Exemplo:

```
int i = 0;
while (i < 5){
    System.out.println(i);
    i++;
}
// O codigo acima vai rodar o loop
// enquanto que a variavel (i) for menor que 5
```

Loop Do/while:

O do/while é uma variante do comando while, esse loop vai executar o bloco de código antes de checar se a condição é verdadeira, então repetirá o loop enquanto a condição for verdadeira.

```
do{
    // Bloco de código
}
while(condicao);
```

Exemplo:

```
int i = 0;
do {
    System.out.println(i);
    i++;
```

```
}
while (i < 5);
```

Loop `for` em Java:

Quando você sabe exatamente quantas vezes vai ser executado o loop pelo bloco de código é melhor utilizar o loop for invés do `while`:

```
for (estado1; estado2; estado3) {
    // Bloco de código a ser executado
}

// O estado 1 é executado (Uma vez)
// antes da execução do bloco de código

// O estado 2 define a condição para
// executar o bloco de código

// O estado 3 é executado (todas as vezes)
// depois do bloco de código ser executado
```

Exemplo:

```
for(int i = 0; i < 5 ; i++) {
    System.out.println(i);
}

//Saída: todos os números de 0 a 4
```

Comandos `break` e `continue` :

`break`:

O comando break pode ser utilizado para sair de um loop:

```
for (int i = 0; i < 10; i++){
    if (i == 4){
        break;
    }
    System.out.println(i);
}
```

`continue`:

O continue para uma iteração no loop, se uma condição ocorre, e continua para a proxima iteração no loop.

```
for (int i = 0; i < 10; i++) {
    if (i == 4){
        continue;
    }
    System.out.println(i);
}
```

Também e possível utilizar o Break e o continue no comando while.

Arrays

Arrays são utilizadas para armazenar vários valores em uma única variável, ao invés de declarar várias variáveis para diferentes valores.

Para declarar arrays, definimos o tipo da variável com um par de colchetes:

```
tipodavariavel [] nomedavariavel;
```

Exemplo:

```
String [] cars;
```

Agora que temos uma array que guarda strings, para inserir valores devemos colocar-los dentro de chaves, dentro de aspas duplas entre vírgulas.

```
String [] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

Para criar uma array de inteiros voce pode escrever:

```
int [] myNum = {10, 20, 30, 40};
```

Acessando os elementos de uma Array:

você pode acessar os elementos de uma Array se referindo ao seu "Index number":

```
String [] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars[0]);

// Saida: Volvo
```

Obs: Os índices de uma array sempre se iniciam a partir do 0, sendo o 1 o segundo elemento.

Modificando um elemento de uma Array:

Para trocar um elemento de uma array precisamos apenas se referir ao elemento com o seu devido índice:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
cars[0] = "Opel";
System.out.println(cars[0]);

// Agora a saida será Opel e não Volvo
```

Tamanho de uma array:

Para descobrir o tamanho de uma array utilizamos a propiedade "length":

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.prinln(cars.length);
```

```
// Saida: 4
```

Loop em uma Array:

Para fazer um loop através de uma array você deve utilizar o loop for e a propriedade `length` para especificar quantas vezes o loop deve rodar.

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

for (int i = 0; i < cars.lenth; i++){
    System.out.println(cars[i]);
}
```

Loop em uma array utilizando `for-each`:

```
for (tipo variavel : nomedaarray){
    ...
}
```

Exemplo:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars) {
System.out.println(i);
}
```

O exemplo acima pode ser lido da seguinte forma: para cada elemento `String` (Chamado i - em index) em cars, printar o valor de i.

Arrays Multidimensionais:

Uma array multidimensional e uma array de arrays, que e muito util quando e necessario guardar dados em forma de tabela com linhas e colunas.

Para criar uma array multidimensional definimos da seguinte forma:

```
tipodaarray [][] = { {1, 2, 3, 4}, {5, 6, 7}};
```

Acessar elementos de uma array dimensional:

Para acessar elementos de uma array multi dimensional é necessário especificar dois índices: um para a array e outro para o elemento dentro desta array. O exemplo abaixo acessa o terceiro elemento (2) na segunda array (1) de `meusNumeros`:

```
int [][] meusNumeros = {{1, 2, 3, 4}, {5, 6, 7}};
System.out.println(meusNumeros[1][2]);
// Saída:
// 7
```

Caso comparemos essa execução com o uso de tabelas o primeiro [] indicaria qual das linhas queremos acessar e o segundo [] qual elemento dessa linha queremos acessar.

Trocando valores de elementos em arrays multidimensionais:

Para trocar um elemento de uma array multidimensional você pode fazer da seguinte forma:

```
int [][] meusNumeros = {{1, 2, 3, 4}, {5, 6, 7}};
meusNumeros [1][2] = 9;
System.out.println(meusNumeros[1][2]);
// Saida:
// 9 invés de 7
```

Loop através de uma array multidimensional:

Para criar um loop em uma array multidimensional é necessário criar um loop `for` dentro de outro para poder passar por todos os elementos das duas dimensões.

```
int [][] meusNumeros = {{1, 2, 3, 4}, {5, 6, 7}};
```

```
for (int i = 0; i < meusNumeros.length; ++i) {
    for (int j = 0; i < meusNumeros[i].length; j++) {
        System.out.prinln(meusNumeros[i][j]);
    }
}
```

Ou você pode utilizar um loop `for-each` , que pode ser considerado mais fácil de ler e escrever:

```
int [][] meusNumeros = {{1, 2, 3, 4}, {5, 6, 7}};

for (int [] row : meusNumeros) {
    for (int i : row) {
        System.out.println(i);
    }
}
```

Métodos em Java:

Um método é um bloco de código que roda apenas quando é chamado. Você pode passar os dados, conhecidos como parâmetros em um método.

Métodos são usados para executar certas ações, e eles também são conhecidos como funções.

Para que utilizar métodos? Para reutilizar códigos: Definir o código uma vez, e utiliza-lo várias vezes.

Criar um método em Java:

Um método deve ser declarado dentro de uma classe. Ele deve ser definida com o nome do método, seguido por um parênteses `()` . Java já fornece alguns métodos predefinidos, por exemplo `System.out.prinln()` , mas você pode também criar seus próprios métodos para fazer certas ações:

Exemplo:

```
public class Main {
    static void meuMetodo(){
        // código a ser executado
    }
}
```

- `meuMetodo()` é o nome do método.
- `static` significa que o método pertence a classe `Main` e não é um objeto. Isso será mais fácil de entender mais a frente.
- `void` significa que o método não retorna um valor. Isso também será explicado mais adiante.

Chamar um método:

Para chamar um método, escreva o nome do método seguido por dois parênteses `()` e um ponto vírgula `;` . No exemplo abaixo o `meuMetodo()` é usado para printar um texto (a ação), quando é chamada:

Exemplo:

Dentro da main, chama o método `meuMetodo()` :

```
public class Main {
    static void meuMetodo() {
        System.out.println("Eu fui executado");
}

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

// Saída:
// "Eu fui executado"
```

Parâmetros e Argumentos:

Informações podem ser passadas para os métodos como parâmetros. Parâmetros agem como variáveis dentro de um método.

Parâmetros são especificados depois do nome do método, dentro dos parâmetros. Você pode adicionar quantos parâmetros quiser, só precisa lembrar-se de separa-los com vírgula.

O exemplo a seguir tem um método que pega uma `String` chamada **fname** como um parâmetro. Quando o método é chamado , nos passamos um primeiro nome, que é usado dentro do método para imprimir o nome completo:

Exemplo:

```
public class Main {
    static void meuMetodo (String fname) {
        System.out.println(fname + "Alves");
    }

    public static void main(String [] args) {
        meuMetodo("Pedro");
        meuMetodo("Joao");
        meuMetodo("Luana");
    }
}

// Pedro Alves
// Joao Alves
// Luana Alves
```

> **Obs:** Quando um parâmetro é passado para o método, ele é chamado de argumento. Então, no exemplo acima, `fname` é um parâmetro, enquanto Pedro, Joao e Luana são argumentos.

Múltiplos Parâmetros:

Você também pode criar quantos parâmetros quiser:

```
public class Main {
    static void meuMetodo(String fname, int age) {
        System.out.println(fname + "is" + age);
    }

    public static void main (String [] args) {
        meuMetodo("Pedro", 30);
        meuMetodo("Joao", 29);
        meuMetodo("Luana", 17);
    }
}

// Saída:
// Pedro is 30
// Joao is 29
// Luana is 17
```

> **Obs:** Observe que, ao trabalhar com vários parâmetros, a chamada do método deve ter o mesmo número de argumentos e parâmetros, e os argumentos devem ser passados na mesma ordem.

Um método com `if…Else` :

É comum se usar `if…else` dentro de métodos:

```
public class Main {

// Criar um método checarIdade() com uma
// variável inteira chamada idade
static void checarIdade(int idade) {

// Se a idade for menor que 18, loga
// Acesso negado
```

```
    if (age < 18) {
        System.out.println("Acesso negado");
        // verifica se a idade e menor que 18
        // e loga a mensagem
    } else {
        System.out.println("Acesso garantido");
        // caso contrario ele loga a outra mensagem
    }

}

    public static void main (String [] args){
        checarIdade(20);
        // executa o metodo
    }
}

// Saída:
// "Acesso garantido"
```

Retornar Valores:

no exemplo anterior utilizamos a palavra `void` , que indica que o método não deve retornar nenhum valor.

Caso você queira que o método retorne um valor, você pode utilizar um tipo de dado (como `int`, `char`, etc.) ao invés de utilizar o `void` , e utiliza o `return` dentro do método:

Exemplo:

```
public class Main {
    static int meuMetodo(int x) {
        return 5 + x;
    }
```

```
    public static void main (String [] args) {
        System.out.println(meuMetodo(3));
    }
}

// Saida:
// 8 (5 + 3)
```

Você também pode armazenar o resultado em uma variável (recomendado, pois é mais fácil de ler e manter):

```
public class Main {
  static int meuMetodo(int x, int y) {
    return x + y;
  }

  public static void main(String[] args) {
    int z = meuMetodo(5, 3);
    System.out.println(z);
  }
}

// Saida:
// 8 (5 + 3)
```

É possível também criar mais de um método com o mesmo nome e trocar os tipos de suas variáveis, tornando o código mais limpo:

```
static int metodoSoma(int x, int y) {
  return x + y;
}

static double metodoSoma(double x, double y) {
  return x + y;
}
```

```
public static void main(String[] args) {
  int meuNumero1 = metodoSoma(8, 5);
  double meuNumero2 = metodoSoma(4.3, 6.26);
  System.out.println("int: " + meuNumero1);
  System.out.println("double: " + meuNumero2);
}
```

Recursão em Java:

Recursão é a técnica de fazer uma função chamar a ela mesma. Essa técnica fornece uma forma de transformar problemas complicados em um problema simples que é fácil de resolver.

Exemplo:

No exemplo abaixo a recursão é usada para somar um intervalo de números, dividindo-o na tarefa simples de somar dois números:

Recursão sendo utilizada para somar todos os números acima de 10:

```
public class Main {
    public static void main(String [] args) {
        int resultado  = sum(10);
        System.out.prinln(resultado);
    }
    public static int soma(int k) {
        if (k > 0) {
            return k + soma(k - 1);
            } else {
                return 0;
            }
        }
    }
```

Quando a função `soma()` é chamada, ela adiciona o parâmetro k a soma de todos os números menores que k e retorna o resultado. Quando k se torna 0, a função retorna 0. O programa quando rodando segue os seguintes passos:

💡 10 + soma(9)

10 + (9 + soma(8))

10 + (9 +(8 + soma(7)))

...

10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + sum(0)

10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 0

Desde que a função não se chama quando k é 0, o programa para e retorna o resultado.

POO → Programação orientada a objeto

criação de um método → forma de tornar o código mais limpo e de fácil manutenção

Sub Rotina (Métodos) :
Procedimento → Se não retorna valor (VOID)
Função → Se retorna valor

Modelagens Tradicionais de Software

Decomposição funcional

Quebra de um problema em partes menores e isoladas. As atividades de um cenário são agrupadas em um conjunto de atividades semelhantes que pode ser atribuída a uma pessoa em forma de cargo ou função.

Fluxo de dados

Retratam o fluxo e o conteúdo da informação (dados e controle), descrevendo.

Programação Estruturada

É uma forma de desenvolvimento em que os programas podem ser reduzidos a apenas três estruturas:

- Sequência: Descreve os passos necessários para processar determinada funcionalidade.
- Decisão: Permite a seleção do fluxo a ser percorrido dentre os passos descritos.
- Repetição: Permite a repetição de alguns passos do processamento.

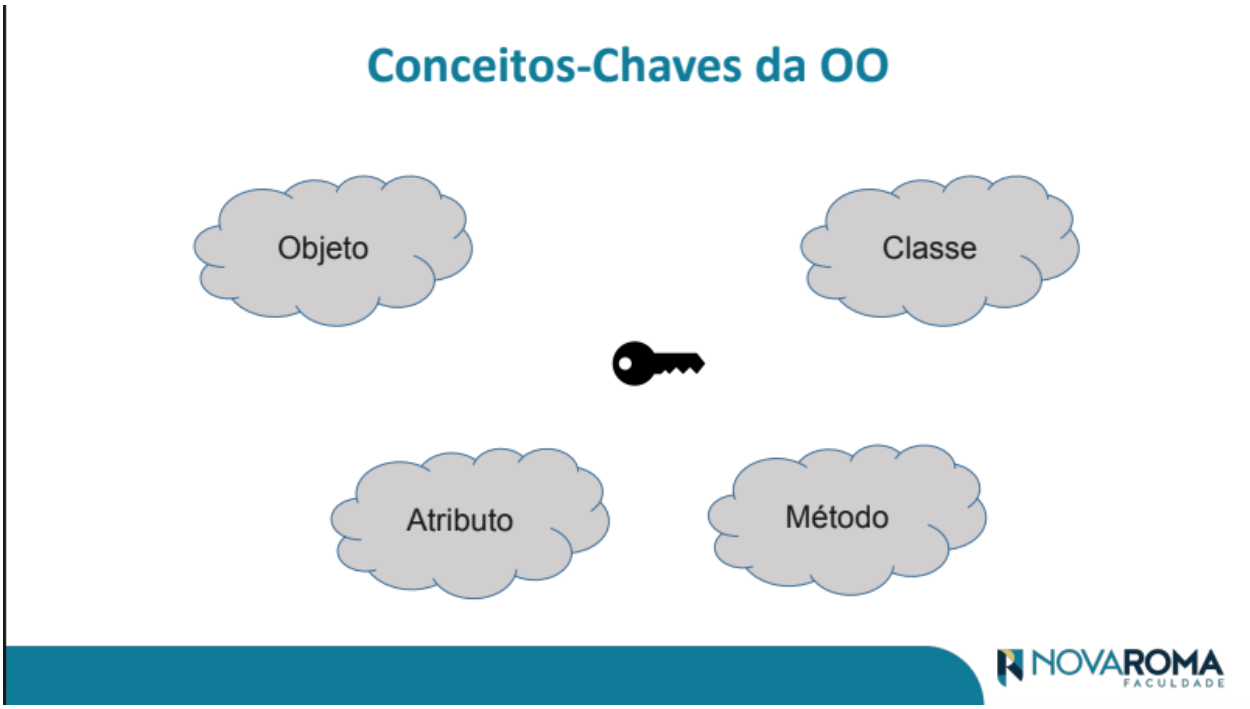
Com o aumento da extensão dos códigos fontes dos programas, aplicou-se a técnica a modularização - divisão do problema em partes menores de acordo com suas funcionalidades.

Modelagem Orientada a Objetos

O sistema é representado por módulos independentes (objetos), que podem ser facilmente reutilizados, modificados ou substituídos.

O software é um conjunto de objetos que encapsulam dados (características/atributos) e operações (ações/métodos) nesses dados para modelar objetos do mundo real. Objetos com atributos e métodos similares são agrupados em classes.

Ao invés de implementar o sistema por meio da programação estruturada baseada em processos, os objetos são criados conforme suas classes e o contexto de negócio é representado como um sistema de objetos cooperativos e colaborativos.



Classe x Objeto

Classificando os objetos por suas características (atributos) semelhantes

Podemos enxergar o mundo real como sendo composto por inúmeros objetos que podem ser agrupados por classes, de acordo com suas características (Atributos).

Atributos (características) dos objetos

- Objetos da classe PESSOA tem nome, data de nascimento, nacionalidade, ...
- Objeto da classe VEÍCULO tem placa, ano de fabricação, chassi, ...
- Objeto da classe FRUTA tem nome, peso, espécie, sabor, ...

Métodos (ações) dos Objetos

- Objetos da classe PESSOA podem caminhar, pular, correr ...
- Objeto da classe VEÍCULO encontra-se em movimento, podendo parar.
- Objeto da classe FRUTA pode crescer, amadurecer ...

As características e as ações de um objeto definem a que classe ele pertence.

Interação entre Objetos

Alguns objetos podem exercer ações sobre outros objetos da mesma classe ou de classes diferentes.

Em nosso mundo as coisas acontecem porque os objetos podem executar ações e a partir dessas ações interagir consigo mesmo, com outros objetos da mesma classe e com objetos de outras classes.

As interações entre os objetos de softwares são realizados pela troca de mensagens entre os objetos.

Visão orientada a objetos

Enxergar o mundo real na forma de OBJETOS de diversas CLASSES de acordo com suas CARACTERÍSTICAS e AÇÕES é apenas uma das maneiras de enxergar o mundo real. Essa é a visão utilizada como base na análise e programação orientada a objetos.

Abstração

Em ciência da computação, a abstração é a habilidade de concentrar nos aspectos essenciais de um contexto, ignorando características menos importantes ou acidentais.

Na orientação a objetos a abstração é utilizada por ocasião do processo de identificação das classes e seus respectivos membros, considerando-se apenas o que seja relevante para solução do problema e desconsiderando-se o que não é necessário para o contexto em questão.

Classes e Objetos em Java

Java é uma linguagem de programação orientada a objetos, logo tudo em Java está associado com classes e objetos, junto com atributos e métodos. Por exemplo: Na vida real, um carro é um objeto. O carro possui atributos, sendo esses, peso, cor e etc. Além disso o carro também possui métodos, como dirigir e frear.

Uma classe é como um construtor de objetos ou um “projeto para criar objetos”

Criar uma Classe:

para criar uma classe em java, se utiliza a palavra `class` :

> Main.java
> Criada uma classe nomeada “ `Main` ” com a variável x:

```
public class Main {
    int x = 5
}
```

> **Obs:** Lembre-se que quando falado sobre a sintaxe em Java que uma classe sempre deve começar com uma letra maiúscula, e o nome do arquivo java deve corresponder com o nome da classe.

Criar um objeto:

Em java, um objeto é criado de uma classe, Nós já criamos uma classe nomeada `Main` , então agora nós podemos utiliza-la para criar objetos.

Para criar um objeto de `Main` , especificamos o nome da classe, seguido pelo nome do objeto, e utilizamos a palavra `new` :

Exemplo:

Criar um objeto chamado “ `meuObjeto` ” e printar o valor de x:

```
public class Main {
int x = 5;

public static void main (String [] args){
        Main meuObjeto = new Main();
        System.out.println(meuObjeto.x);
    }
}
```

Múltiplos Objetos

Você pode criar múltiplos objetos de e uma classe:

Exemplo:

Criando dois objetos de `Main` :

```
public class Main {
    int x = 5;

    public static void main (String [] args){
        Main meuObjeto1 = new Main();// Objeto1
        Main meuObjeto2 = new Main();// Objeto2
        System.out.println(meuObjeto1.x);
        System.out.println(meuObjeto2.x);
    }
}
```

Usando Múltiplas Classes:

Você pode também criar um objeto de uma classe e acessar ele em outra classe. Isso é usado geralmente para ter uma melhor organização das classes (uma classe tem todos os atributos e métodos, enquanto a outra tem todo o método `main()` (código a ser executado)).

Lembre-se que o nome de um arquivo Java deve corresponder com o nome da classe. Nesse exemplo, nós criamos dois arquivos no mesmo diretório/pasta:

- Main.java
- Second.java

Main.java

```
public class Main {
    int x = 5;
}
```

Second.java

```
class Second {
    public static void main (String [] args){
        Main meuObjeto = new Main();
        System.out.prinln(meuObjeto.x);
    }
}
```

Saída:
5

Atributos de Classes em Java:

No exemplo acima, nós utilizamos a variável `x` . Essa variável pode ser considerada como um atributo da classe `Main`. Ou você pode dizer que atributos de uma classe são suas variáveis:

Exemplo:

Criando uma classe chamada "`Main`" com dois atributos: `x` e `y` :

```
public class Main {
    int x = 5;
    int y = 3;
}
```

Obs: Outro termo para os atributos de uma classe é **fields**

Acessando atributos

Você pode acessar atributos criando um objeto da classe, e usando a sintaxe do ponto (`.`):

O exemplo abaixo vai criar um objeto da classe `Main`, com o nome `meuObjeto`. E usaremos o atributo `x` no objeto para printar seu valor:

Exemplo:

Criar um objeto chamado "`meuObjeto`" e printar o valor de `x`:

```
public class Main{
    int x = 5;

    public static void main (String [] args){
        Main meuObjeto = new Main();
        System.out.println(meuObjeto.x);
    }
}
```

Modificando Atributos

Você também consegue modificar atributos:

Exemplo:

Definir o valor de `x` para 40:

```
public class Main {
    int x;

    public static void main (String [] args){
        Main meuObjeto = new Main();
        meuObjeto.x = 40;
        System.out.println(meuObjeto.x);
    }
}
```

ou sobrepor valores existentes:

Exemplo:

Trocar o valor de `x` para 25:

```
public class Main {
    int x = 10;

    public static void main (String [] args) {
        Main meuObjeto = new Main();
```

```
        meuObjeto.x = 25;
        // X agora é 25
        System.out.prinln(meuObjeto.x);
    }
}
```

Se você não quiser que o seu atributo seja modificado você pode declara-lo como `final`:

```
public class Main {
    final int x = 10;

    public static void main (String [] args) {
        Main meuObjeto = new Main();
        meuObjeto.x = 25;
        // Irá gerar um erro:
        // cannot assign a value to a final variable
        System.out.prinln(meuObjeto.x);
    }
}
```

É possível também transformar o atributo em `private` , impedindo assim também que ele seja modificado.

```
class Atributo {
    private int x = 10;
}

private class Main {
    public static void main (String [] args){
        Atributo meuObjeto = new Atributo();
        meuObjeto = 25;
        // Irá gerar um erro:
        // x has private access in Atribute
        System.out.println(meuObjeto.x);
```

```
    }
}
```

Múltiplos Objetos

Se você criar múltiplos objetos de uma classe, é possível trocar o valor do atributo em um objeto sem afetar o valor do atributo no outro:

Exemplo:

Trocar o valor de `x` para 25 no `meuObjeto2`, e deixar `x` em `meuObjeto1` sem ser modificado:

```
public class Main {
    int x = 5;

    public static void main (String [] args){
        Main meuObjeto1 = new Main();
        Main meuObjeto2 = new Main();

        meuObjeto2.x = 25;

        System.out.prinln(meuObjeto1.x); // Saída: 5
        System.out.prinln(meuObjeto2.x); // Saída: 25
    }
}
```

Múltiplos atributos

Você pode especificar quantos atributos quiser:

Exemplo:

```
public class Main {
    String fname = "Joao";
    String lname = "Alves";
    int age = 25;
```

```
    public static void main (String [] args) {
        Main meuObj = new Main();
        System.out.println("Nome:" + meuObj.fname + "" + meuObj
        System.out.prinln("Idade: " + meuObj.age);
    }
}
```

Static vs. Public

Como visto acima em , um método deve ser declarado dentro de uma classe e eles são utilizados para executar certas ações.

Você frequentemente irá ver códigos em java em que utilizam os atributos `static` e `public` em métodos. No exemplo abaixo, foi criado um método `static` , que significa que ele pode ser acessado sem precisar de um objeto da classe, diferentemente do `public` , que só pode ser acessado por objetos:

Exemplo:

```
public class Main {
    // Método static
    static void meuMetodoStatic(){
        System.out.prinln("Static podem ser chamados sem criar
    }

    // Método public
    public void meuMetodoPublic(){
        System.out.println("Public precisam de objetos para ser
    }

    public static void main (String [] args){
        meuMetodoStatic();
        // meuMetodoPublic(); Isso gera um erro

        Main meuObj = new Main(); // Cria um objeto
        meuObj.meuMetodoPublic; // Chama o metodo public
```

```
    }
}
```

Acessando Métodos com um Objeto

Exemplo:

```
public class Main {

    public void fullTPS(){
            System.out.println("O carro ta indo mais rapido poss
    }

    public void velocidade (int velMax){
        System.out.prinln("Velocidade maxima: " + velMax);
    }

    public static void main (String [] args){
        Main meuCarro = new Main();
        meuCarro.fullTPS();
        meuCarro.velocidade(180);
    }
}

// Saida:

// O carro ta indo mais rapido possivel
// Velocidade maxima: 180
```

Usando múltiplas Classes

Assim como especificado em , é uma boa prática criar um objeto de uma classe e acessa-lo em uma outra classe.

> **Obs:** Lembre-se que o nome de um arquivo Java deve condizer com o nome da classe, nesse exemplo foram criados dois arquivos no mesmo diretório:

- Main.java
- Second.java

Main.java

```
public class Main {
  public void fullThrottle() {
    System.out.println("The car is going as fast as it can!");
  }

  public void speed(int maxSpeed) {
    System.out.println("Max speed is: " + maxSpeed);
  }
}
```

Second.java

```
class Second {
  public static void main(String[] args) {
    Main myCar = new Main();     // Create a myCar object
    myCar.fullThrottle();      // Call the fullThrottle() metho
    myCar.speed(200);          // Call the speed() method
  }
}
```

Construtores:

Um construtor em java é um método especial que é utilizado para inicializar objetos. O construtor é chamado quando um objeto de uma classe é criado. Ele pode ser utilizado para setar valores iniciais para atributos de objetos.