Conceitos Básicos

Haaa lek ! Ai ..Resumão do Java e declaração de vars com pitada de array

Resumo de Java pra estudo pessoal

(by Daniel)

Vou separar em tópicos entende? Pra ficar mais maneiro de estudar

- Conceitos básicos
- Condicionais e Loops
- Arrays (listas, vetores e Matrizes)

funcionalidade

Mais de 3 bilhões de dispositivos são executados em Java. Java é usado para desenvolver aplicativos para o sistema operacional Android do Google, várias aplicações de desktop, como players de mídia, programas antivírus, aplicativos da Web, aplicativos corporativos (ou seja, bancos) e muito mais!

Primeiro programa Java

Comecemos por criar um programa simples que imprima "Olá Mundo" na tela.

```
classe MyClass*{

public static void main (String [] args) {

   System.out.println ("Olá Mundo");
                              }
}
```

Em Java, cada linha de código que realmente pode ser executada precisa estar dentro de uma classe.

No nosso exemplo, chamamos a classe MyClass.

Em Java, cada aplicação tem um ponto de entrada, ou um ponto de partida, que é um método chamado main. Junto com o main, as palavras-chave públicas e estáticas também serão explicadas mais tarde.

Como um resumo:

- Todo programa em Java deve ter uma classe.

- **Todo programa Java começa a partir do método principal.**
- **Em Java cada classe sempre maiúscula p/ kda palavra, se eu fosse criar uma classe que fritasse omelete por exemplo :**
- **class EssaClasseServeParaFritarUmOmelete{ };**

Review 1

- O método principal

- Para executar o nosso programa, o método principal deve ser idêntico a esta assinatura:
- public static void main (String [] args)

- - público: qualquer um pode acessá-lo
- - estático: o método pode ser executado sem criar uma instância da classe que contenha o método principal
- - vazio: o método não retorna nenhum valor
- - main: o nome do método

- Por exemplo, o código a seguir declara um método chamado teste, que não retorna nada e não possui parâmetros:

```
teste vazio ()
```

- Os parâmetros do método são declarados dentro dos parênteses que seguem o nome do método.
- Para principal, é uma série de strings chamadas args. Vamos usá-lo em nossa próxima lição, então não se preocupe se você não entende tudo agora.

O “escreval” do Java:

System.out.println ()

Em seguida, é o corpo do método principal, fechado em chaves :

```
{
System.out.println ("Hello World!");
}
```

O método println imprime uma linha de texto na tela.

A classe System e seu fluxo de saída são usados para acessar o método println.

Nas classes, métodos e outros códigos de controle de fluxo, o código está sempre incluído em chaves curvas { }.

Ponto e vírgula em Java

Você pode passar um texto diferente como o parâmetro para o método println para imprimi-lo.

```
classe MyClass {
  public static void main (String [] args) {
    System.out.println ("Estou aprendendo Java");    //saída: “Estou aprendendo Java”
  }
}
```

Em Java, cada declaração de código deve terminar com um ponto e vírgula.

Lembre-se: não use pontos-e-vírgulas após o método e as declarações de classe que se seguem, com o corpo definido usando as chaves .

Comentários

O objetivo de incluir comentários em seu código é explicar o que o código está fazendo.

O Java suporta comentários de uma única e multi-linha. Todos os caracteres que aparecem dentro de um comentário são ignorados pelo compilador Java.

Um comentário de linha única começa com duas barras laterais e continua até chegar ao final da linha.

Por exemplo:

```
// este é um comentário de uma única linha
x = 5; // um comentário de linha única depois do código
```

A adição de comentários à medida que você escreve código é uma boa prática, porque eles fornecem esclarecimentos e entendimentos quando você precisa se referir a ele, bem como para outros que talvez precisem lê-lo.

Comentários de várias linhas

O Java também suporta comentários que abrangem múltiplas linhas.

Você inicia esse tipo de comentário com uma barra diagonal seguida de um asterisco e termina com um asterisco seguido de uma barra inclinada para a frente.

Por exemplo:

```
/ * Este também é um
   comentário abrangente
   várias linhas * /
```

Observe que o Java não suporta comentários aninhados em várias linhas.

No entanto, você pode aninhar comentários de linha única em comentários de várias linhas.

```
/ * Este é um comentário de linha única:

   // um comentário de linha única

 * /
```

Variáveis

Exemplos de declarações variáveis:

```
classe MyClass {
  public static void main (String [] args) {
    String name = "David";
    int idade = 42;
    pontuação dupla = 15,9;
    grupo char = 'Z';
  }
}
```

Tente você mesmo

O char está como caractere e representa um único caractere(‘Z’)

Outro tipo é o tipo booleano, que tem apenas dois valores possíveis: verdadeiro e falso.

Este tipo de dados é usado para sinalizadores simples que rastreiam condições verdadeiras / falsas.

Por exemplo:

```
boolean online = true;
```

Você pode usar uma lista separada por vírgulas para declarar mais de uma variável do tipo especificado. Exemplo: int a = 42, b = 11;

Os operadores de matemática

O Java fornece um conjunto rico de operadores para usar na manipulação de variáveis. Um valor usado em ambos os lados de um operador é chamado de operando.

Por exemplo, na expressão abaixo, os números 6 e 3 são operandos do operador mais:

```
int x = 6 + 3;
System.ou.println(x)                                    //saída: 9
```

Operadores aritméticos Java:

+ adição

- subtração

* multiplicação

/ divisão

% módulo

Os operadores aritméticos são usados em expressões matemáticas da mesma maneira que são usados em equações algébricas.

Bem simples neh?

Adição

O operador + acrescenta dois valores, como duas constantes, uma constante e uma variável, ou uma variável e uma variável. Aqui estão alguns exemplos de adição:

```
int sum1 = 50 + 10;       //60
int sum2 = sum1 + 66;     //126
int sum3 = sum1+ sum2;    //166
```

Subtração

O operador - resta um valor de outro.

```
int sum1 = 1000 - 10;                  //990
int sum2 = sum1 - 5;                   //985
int sum3 = sum1 - sum2;                 //5
```

Multiplicação

O operador * multiplica dois valores.

```
int sum1 = 1000 * 2;          //2.000
int sum2 = sum1 * 10;         //20.000
int sum3 = sum1 * sum2;       //40.000.000
```

Divisão

O / operador divide um valor por outro.

```
int sum1 = 1000/5;                  //250
int sum2 = sum1 / 2;                //125
int sum3 = sum1 / sum2;             //2
```

No exemplo acima, o resultado da equação de divisão será um número inteiro, como int é usado como o tipo de dados. Você pode usar o dobro para recuperar um valor com um ponto decimal. Ex: Sum4= 7.5 Sum5=Sum4*2 //15

Modulo

A operação matemática modulo (ou restante) executa uma divisão inteira de um valor por outra e retorna o restante dessa divisão.

O operador para a operação do módulo é o caractere porcentual (%).

Exemplo:

valor int = 23;

int res = valor% 6; // res é 5

pq 6*3 = 18 para 23 ainda faltam 5.. Logo res=5;

Dividindo 23 por 6 retorna um quociente de 3, com um restante de 5. Assim, o valor de 5 é atribuído à variável res.

Incrementar Operadores

Um operador de incremento ou decremento fornece uma maneira mais conveniente e compacta de aumentar ou diminuir o valor de uma variável por um.

Por exemplo, a afirmação x = x + 1; pode ser simplificado para ++ x;

Exemplo:

int test = 5;

teste ++; // teste é agora 6 e no próximo loop será 7 e assim por diante

O operador de decremento (-) é usado para diminuir o valor de uma variável por um.

int test = 5;

--teste; // teste é agora 4 ... enquanto aquele soma esse subtrai

Prefixo & Pós fixo

Duas formas, prefixos e Pós fixo, podem ser usados com os operadores de incremento ou diminuição.

Com a forma de prefixo, o operador aparece antes do operando, enquanto na forma de posfixo, o operador aparece após o operando. Abaixo está uma explicação sobre como as duas formas funcionam:

Prefixo: Incrementa o valor da variável e usa o novo valor na expressão.

Exemplo:

int x = 34;

int y = ++ x; // é 35

O valor de x é primeiro incrementado para 35, e então é atribuído a y, então os valores de x e y agora são 35.

Posfix: o valor da variável é usado pela primeira vez na expressão e, em seguida, é aumentado.

Exemplo:

int x = 34;

int y = x ++; // é 34

Tente você mesmo

x primeiro é atribuído a y, e então é incrementado por um. Portanto, x torna-se 35, enquanto a y é atribuído o valor de 34.

O mesmo se aplica ao operador do decremento.

Operadores de atribuição

Você já conhece o operador de atribuição (=), que atribui um valor a uma variável.

valor int = 5;

Isso atribuiu o valor 5 a uma variável chamada valor do tipo int.

O Java fornece uma série de operadores de atribuição para facilitar a escrita do código.

Adição e atribuição (+ =):

int num1 = 4;

int num2 = 8;

num2 + = num1; // num2 = num2 + num1; ou seja ... num2 é 12 e num1 é 4

Subtração e atribuição (- =):

int num1 = 4;

int num2 = 8;

num2 - = num1; // num2 = num2 - num1;

// num2 é 4 e num1 é 4

Da mesma forma, o Java suporta multiplicação e atribuição (* =), divisão e atribuição (/ =), e restante e atribuição (% =).

Concatenação

O operador + entre strings os adiciona para criar uma nova string. Esse processo é chamado de concatenação.

A seqüência resultante é a primeira string colocada junto com a segunda string.

Por exemplo:

String firstName, lastName;

firstName = “James”;

lastName = “Bond";

System.out.println ("Meu nome é" + firstName + "" + lastName);

// Imprime: meu nome é James Bond

Scanner, o “Leia” do visual G em Java

Embora o Java ofereça muitos métodos diferentes para obter a entrada do usuário, o objeto Scanner é o mais comum, e talvez o mais fácil de implementar.
Importe a classe Scanner para usar o objeto Scanner, como visto aqui:

importar java.util.Scanner;

Para usar a classe Scanner, crie uma instância da classe usando a seguinte sintaxe:

Scanner myVar = new Scanner (System.in);

Agora você pode ler em diferentes tipos de dados de entrada que o usuário digita.

Aqui estão alguns métodos disponíveis na classe Scanner:

Leia um byte - nextByte () Leia um breve - nextShort () Leia um int - nextInt () Leia um longo - próximoLong () Leia um flutuador - nextFloat ()

Leia um duplo - nextDouble () Leia um booleano - nextBoolean () Leia uma linha completa - nextLine () Leia uma palavra - next ()

Exemplo de um programa usado para obter entrada do usuário:

```
importar java.util.Scanner;
classe MyClass {
  public static void main (String [] args) {
    Scanner Leia = new Scanner (System.in);
    System.out.println (Leia.nextLine ());
  }
}
```

Isso aguardará o usuário inserir algo e imprimir essa entrada.

Condicionais e loops

IF, tomando decisões

Declarações condicionais são usadas para executar diferentes ações com base em condições diferentes.

A instrução if é uma das instruções condicionais mais utilizadas.

Se a expressão de condição da declaração if for verdadeira, o bloco de código dentro da instrução if é executado. Se a expressão for falsa, o primeiro conjunto de código após o fim da instrução if (após a correia curvada de fechamento) é executado.

Sintaxe:

```
se (condição)                                                    // Executa quando a condição é verdadeira
}
```

Qualquer um dos seguintes operadores de comparação pode ser usado para formar a condição:

< menor do que , > superior a , ! = não é igual a, == igual a

<= menor ou igual a, > = maior ou igual a

Por exemplo:

```
int x = 7;
se (x <42) {
  System.out.println ("Hi");
}                    //saída:  Hi
```

Lembre-se de que você precisa usar dois sinais iguais (==) para testar a igualdade, uma vez que um único sinal igual é o operador de atribuição.

Else if, ..mais decisões

Uma instrução if pode ser seguida por uma instrução opcional else, que é executada quando a condição é avaliada como falsa.

Por exemplo:

```
idade = 30;


se (idade <16) {
   System.out.println ("Too Young");
} else {
   System.out.println ("Welcome!");
}
// Saídas "Bem-vindo"
```

À medida que a idade é igual a 30, a condição na instrução if é avaliada como falsa e a instrução else é executada.

Aninhado de declarações if e else

Você pode usar uma declaração if-else dentro de outra declaração if or else.

Por exemplo:

```
int age = 25;
if (idade> 0) {
   if (idade> 16) {
      System.out.println (“Bem-vindo!!");
   } else {
      System.out.println (“Novinha");
   }
} else {
   System.out.println ("Erro  ;( ");
}
// Saída  "Bem-vindo!“       p.s    vc pode usar quantos ifs quiser.
```

Operadores lógicos

Os operadores lógicos são usados para combinar múltiplas condições.

Digamos que você queria que seu programa exibisse "Bem-vindo!" somente quando a variável idade for maior que 18 e a variável moeda for superior a 500.

Uma maneira de conseguir isso é usar instruções if juntas, aninhadas:

```
if (idade> 18) {
  if(dinheiro> 500) {
      System.out.println ("Welcome!");
   }
}
```

No entanto, usar o operador AND logic (&&) é uma maneira melhor:

```
if (idade> 18  &&  dinheiro> 500) {
   System.out.println ("Welcome!");
}
```

Se ambos os operandos do operador AND forem verdadeiros, a condição torna-se verdadeira.

O Operador OR

O operador OR (||) verifica se qualquer uma das condições é verdadeira.

A condição torna-se verdadeira, se qualquer um dos operandos se avaliar como verdadeiro. Por exemplo:

```
int age = 25;
int money = 100;
if (idade> 18 || dinheiro> 500) {
  System.out.println ("Welcome!"); // Saídas "Bem-vindo!"
}
```

O código acima imprimirá "Bem-vindo!" se a idade for maior que 18 ou se o dinheiro for superior a 500.

O operador lógico **NOT (!)** É usado para reverter o estado lógico de seu operando. Se uma condição for verdadeira, o operador NOT logico fará isso falso.

Exemplo:

```
int age = 25;
if (! (idade> 18)) {
  System.out.println (“novinha");
} else {
  System.out.println ("Bem-vindo");
}
// Saída:  "Bem-vindo"
```

! (idade> 18) lê como "se a idade não é maior que 18".

While Loops

Uma instrução de loop que permite executar repetidamente uma declaração ou grupo de instruções.

Uma instrução loop while repetidamente executa uma declaração de destino, desde que uma determinada condição seja verdadeira. Exemplo:

```
int x = 3;
while (x> 0) {
  System.out.println (x);
  x--;
}
/ *
Saídas
 3
 2
 1
* /
```

O while loops verifica a condição x> 0. Se ele for verdadeiro, ele executa as declarações dentro do corpo. Em seguida, verifica a declaração novamente e repete.

Observe a declaração x--. Isso diminui x cada vez que o loop é executado, e faz o loop parar quando x atinge 0.

Sem a declaração, o loop seria executado para sempre.

While Loops

Quando a expressão é testada e o resultado é falso, o corpo do loop é ignorado e a primeira declaração após o loop while é executada. Exemplo:

```
int x = 6;
while (x <10)
{
  System.out.println (x);
  x ++;
}
System.out.println ("Loop terminou");
//saída:
/ *
6
7
8
9
Loop terminou
* /
```

For Loops

Outra estrutura de loop é o loop for. Um loop for permite que você escreva eficientemente um loop que precisa executar um número específico de vezes.
Sintaxe:

para (inicialização; condição; incremento / decremento) {

  (afirmações)

}

Inicialização: a expressão é executada apenas uma vez durante o início do ciclo

Condição: é avaliada cada vez que o loop é executado. O loop executa a declaração repetidamente, até que esta condição retorna falsa. O exemplo a seguir imprime os números 1 a 5.

Increment / Decrement: Executa após cada iteração do loop.

```
for (int x = 1; x <= 5; x ++) {
  System.out.println (x);
}
/ * Saídas
1
2
3
4
5
* /
```

Isso inicializa x para o valor 1 e imprime repetidamente o valor de x, até que a condição x <= 5 se torne falsa. Em cada iteração, a declaração x ++ é executada, incrementando x por uma.

Observe o ponto-e-vírgula (;) após a inicialização e a condição na sintaxe.

For Loops

Você pode ter qualquer tipo de condição e qualquer tipo de declaração de incremento no loop for.

O exemplo abaixo imprime apenas os valores pares entre 0 e 10:

```
for (int x = 0; x <= 10; x = x + 2) {
  System.out.println (x);
}
/ *
0
2
4
6
8
10
* /
```

P.S Um loop for é melhor quando os números inicial e final são conhecidos.

Arrays

Arrays

Uma matriz é uma coleção de variáveis do mesmo tipo.

Quando você precisa armazenar uma lista de valores, como números, você pode armazená-los em uma matriz, em vez de declarar variáveis separadas para cada número.

Para declarar um vetor ou vetor, você precisa definir o tipo de elementos com colchetes.

Por exemplo, para declarar uma série de inteiros:

int [] Vet;

O nome da vetor é Vet. O tipo de elementos que ele irá realizar é int.

Agora, você precisa definir a capacidade dele, ou o número de elementos que ela irá manter. Para fazer isso, use a palavra-chave nova.

int [] Vet = new int [5];

O código acima declara uma matriz de 5 inteiros.

Em uma matriz, os elementos são ordenados e cada um tem uma posição específica e constante, que é chamado de índice.

Para referenciar elementos em uma matriz, digite o nome da matriz seguida pela posição de índice dentro de um par de colchetes.

Exemplo:

Vet[2] = 42;

Isso atribui um valor de 42 ao elemento com 2 como índice.

Observe que os elementos na matriz são identificados com números de índice baseados em zero, o que significa que o índice do primeiro elemento é 0 em vez de um. Então, o índice máximo da matriz int [5] é 4.

Inicializando Arrays

Java fornece um atalho para instanciar matrizes de tipos primitivos e strings.

Se você já sabe quais valores inserir na matriz, você pode usar matriz unilateral(vetor).

Exemplo de uma matriz unilateral:

```
String [] myNames = {"A", "B", "C", "D"};
System.out.println (myNames [2]);

// Saídas "C“
```

Coloque os valores em uma lista separada por vírgulas, incluídas em chaves.

O código acima inicializa automaticamente uma matriz contendo 4 elementos e armazena os valores fornecidos.

Às vezes, você pode ver os colchetes colocados após o nome da matriz, que também funciona, mas a maneira preferida é colocar os colchetes após o tipo de dados da matriz.

Comprimento da matriz ou Vetor

Você pode acessar o comprimento de uma matriz (o número de elementos que armazena) através da sua propriedade de comprimento(length).

Exemplo:

```
int [ ] Vetor = new int [5];

System.out.println (Vetor.length);


// Saída 5
```

Arrays

Agora que sabemos como configurar e obter elementos de matriz, podemos calcular a soma de todos os elementos em uma matriz usando loops.

O loop for é o loop mais utilizado ao trabalhar com arrays, pois podemos usar o comprimento da matriz para determinar quantas vezes executar o loop.

```
int [] Vetor = {6, 42, 3, 7};
int sum = 0;
for (int x = 0; x <Vetor.length; x ++) {
   soma + = Vetor [x];
}
System.out.println (soma);
//saída :  58
```

No código acima, declaramos uma soma variável para armazenar o resultado e atribuí-lo a 0.

Então usamos um loop for para pesquisar através da matriz e adicionamos o valor de cada elemento à variável.

A condição do loop for é x <Vetor.length(comprimento do vetor), pois o índice do último elemento é Vetor.length-1.

Matrizes multidimensionais

Arrays multidimensionais são matrizes que contêm outros arrays. A matriz bidimensional é a matriz multidimensional mais básica.

Para criar matrizes multidimensionais, coloque cada matriz dentro de seu próprio conjunto de colchetes. Exemplo de uma matriz bidimensional:

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

Isso declara uma matriz com dois arrays como seus elementos.

Para acessar um elemento na matriz bidimensional, forneça dois índices, um para a matriz e outro para o elemento dentro dessa matriz.

O exemplo a seguir acessa o primeiro elemento na segunda matriz de amostra.

```
int x = amostra [1] [0];
System.out.println (x);
// Saídas 4
```

Os dois índices da matriz são chamados índice de linha e índice de coluna.