Aula III DWeb III

Revisando e Aprendendo

Variáveis & Constantes
Atributos & Métodos
Introduzindo o Diagrama de Classe - UML
Operadores

Fontes:
https://www.devmedia.com.br/logica-de-programacao-introducao-a-algoritmos-e-pseudocodigo/37918
http://professores.dcc.ufla.br/~monserrat/download/logica.pdf
Livro de Lógica de Programação- Técnico em Informática - QI Faculdade & Escola Técnica
http://www.ams.eti.br/livros/Sandra_Puga.pdf
https://www.devmedia.com.br/java-operadores-de-atribuicao-aritmeticos-relacionais-e-logicos/38289

Revisando ....

Os dados são, na verdade, os valores que serão utilizados para a resolução de um problema. Esses valores podem ser fornecidos pelo usuário do programa, podem ser originados a partir de processamentos (cálculos) ou, então, a partir de arquivos, bancos de dados ou outros programas.

Variáveis:

Os dados são armazenados temporariamente em variáveis para que sejam processados de acordo com as especificações do algoritmo. Para que haja integridade no resultado obtido, os dados devem ser classificados de acordo com o tipo do valor a ser armazenado na variável, isto é, para evitar problemas que podem ser ocasionados devido ao fornecimento de valores inadequados à operação que será realizada. **Por exemplo, vamos supor que o algoritmo deva especificar os passos para efetuar a soma de dois números quaisquer fornecidos pelo usuário. Então, será feita uma operação aritmética (soma), que só poderá ser realizada com valores numéricos.**
Os tipos de dados são definidos, normalmente, a partir dos tipos primitivos criados em função das características dos computadores. Como os dados manipulados pelos computadores durante a execução dos programas são armazenados na memória, esses tipos seguem as características de formato e espaço disponível nessa memória. Assim, são organizados em bits e bytes e suas combinações. Por exemplo:

- para representar um número inteiro, poderiam ser usados dois bytes ou 16 bits. Isso resultaria em 216 combinações possíveis para a representação de números, ou 65.536 possibilidades, considerando os estados possíveis que um bit pode assumir: O ou 1. Lembrando que os números poderiam assumir valores negativos e positivos nessa faixa, teríamos representações que iriam de -32.768 a 32.767.

Tipos de Dados

Definir o tipo de dado mais adequado para ser armazenado em uma variável é uma questão de grande importância para garantir a resolução do problema. Ao desenvolver um algoritmo, é necessário que se tenha conhecimento prévio do tipo de informação (dado) que será utilizado para resolver o problema proposto. A partir daí, escolhe-se o tipo adequado para a variável que representará esse valor. **Na confecção de algoritmos, utilizamos os tipos de dados primitivos (literal, inteiro, real e lógico)**, uma vez que os algoritmos apenas representam a resolução dos problemas. Já na confecção de programas, existem desdobramentos para esses tipos de dados a fim de adequá-los melhor ao propósito de cada linguagem e à resolução prática dos problemas. Veja na Tabela 1 as definições dos tipos de dados primitivos e seus desdobramentos na linguagem de programação Java.

Alguns tipos de dados da linguagem de programação Java têm particularidades.
Veja a seguir quais são elas.

O tipo **long** deve ser identificado com a **letra L** para não ser 'compactado' pela linguagem em um tipo inteiro. A compactação ocorre como uma maneira de reduzir a memória gasta.
Da mesma forma que tenta usar o mínimo de memória, a linguagem Java tenta utilizar o máximo de precisão possível. Assim, se um elemento do **tipo float (7 casas de precisão após a vírgula) não for identificado com a letra f, a linguagem vai considerá-lo como do tipo double (15 casas de precisão após a vírgula),** o que pode gerar vários erros de compilação e execução.
Isso é uma característica da linguagem Java e não deve ser estendido a outras linguagens. Por exemplo:

```
int n = 4;
long numero = 456L;
float pi = 3.14f;
double tamanho = 3.873457486513793;
```

Primitivos		Específicos para linguagem de programação Java	
Tipos de dados	Definição	Tipos de dados	Capacidade de armazenamento na memória do computador, de acordo com a linguagem Java
Literal – também conhecido como texto ou caractere	Poderá receber letras, números e símbolos. *Obs.*: Os números armazenados em uma variável cujo tipo de dado é literal não poderão ser utilizados para cálculos.	`char`	16 bits – Armazena Unicodes. **NOTA:** *Também é possível armazenar dados do tipo literal na Classe String.*
Inteiro	Poderá receber números inteiros positivos ou negativos.	`byte`	8 bits – De (–128) até (127).
		`short`	16 bits – De (–32.768) até (32.767).
		`int`	32 bits – De (–2.147.483.648) até (2.147.483.647).
		`long`	64 bits – De (–9.223.372.036.854.775.808) até (9.223.372.036.854.775.807).
Real – também conhecido como ponto flutuante	Poderá receber números reais, isto é, com casas decimais, positivos ou negativos.	`float`	32 bits – De (–3,4E–38) até (–3,4E+38).
		`double`	64 bits – De (–1,7E–308) até (+1,7E+308).
Lógico – também conhecido como booleano	Poderá receber verdadeiro (1) ou falso (0).	`boolean`	8 bits – Em Java pode-se armazenar `true` ou `false`.

Fonte da imagem - Livro: Lógica de Programação e Estruturas de Dados - Sandra Puga

Resumindo o que foi visto até agora:

O **Java é uma linguagem fortemente tipada**, ou seja, todos os dados do programa devem ter um tipo de dado especificando que tipo de informação irá para a memória, se será um texto, número inteiro, um número que pode ter casas decimais, e assim por diante.

Nos algoritmos, as **variáveis são utilizadas para representar valores desconhecidos, porém necessários para a resolução de um problema e que poderão ser alterados de acordo com a situação.** Por isso dizemos que as variáveis armazenam valores (dados) temporariamente.

Quando um algoritmo é transcrito para uma determinada linguagem de programação, as variáveis também terão a função de armazenar dados temporariamente, mas na memória RAM do computador. Esses dados serão utilizados durante o processamento para a resolução do problema em questão.

Toda variável deve ser identificada, isto é, deve receber um nome ou identificador. O nome de uma variável deve ser único e deve estar de acordo com algumas regras:

- **Não utilizar espaços entre as letras.** Por exemplo, em vez de **nome do cliente,** o correto seria **nome_do_cliente** ou **nomecliente**. O caractere 'sublinha' ou 'underline' (_) pode ser utilizado para representar o espaço entre as letras.
- **Não iniciar o nome da variável com algarismos (números)**. Por exemplo: não usar 2valor. O correto seria valor2.
- **Não utilizar palavras reservadas, isto é, palavras que são utilizadas nos algoritmos para representar ações específicas. Por exemplo:**
  - **se palavra que representa uma condição ou teste lógico;**
  - **var palavra que representa a área de declaração de variáveis.**
- **Não utilizar caracteres especiais, como acentos, símbolos (?I : @# etc.), ç, entre outros.**
- **Ser sucinto e utilizar nomes coerentes.**

Exemplos de tipos de dados

- **nome: String** (pois contém letras, espaços, etc.)
- **numeroDeFilhos: byte** (pois não será um número superior a 127 e nem terá casas decimais)
- **numeroDeHabitantes: int** (pois não terá casas decimais)
- **media: double** (pode conter casas decimais, como por exemplo, 9,5)
- **aprovado: boolean** (pois só pode assumir os valores true (verdadeiro) ou false (falso))

Repetindo: em Java, os nomes para as variáveis são case-sensitive, isto é, nomes com letras maiúsculas são diferenciados de nomes com letras minúsculas.
Por exemplo: NomeCliente é diferente de nomecliente e também de nomeCliente.

- **Nomes devem começar com uma letra, um caractere 'sublinha' ou 'underline' (_)ou o símbolo cifrão ($). Os caracteres subseqüentes podem também ser algarismos.**
- **Não utilizar caracteres especiais, como acentos, símbolos (?I :@# etc.), ç, entre outros, exceto os acima citados.**
- **As letras podem ser maiúsculas ou minúsculas.**
- **Não podem ser utilizadas palavras reservadas, como final, float, for, int etc.**

Constante

São valores que não sofrem alterações ao longo do desenvolvimento do algoritmo ou da execução do programa.
Por exemplo, na expressão abaixo, o valor 3 . 1415 é atribuído à constante pi e permanecerá fixo até o final da execução. Em Java, uma constante é uma variável declarada com o modificador final.
Por exemplo:
final float pi = 3.1415f;

As constantes devem ser declaradas como variáveis cujo valor atribuído permanecerá inalterado ao longo do programa. **Por isso, são também chamadas de variáveis somente de leitura.**

Métodos

Métodos são blocos de código que pertencem a uma classe e tem por finalidade realizar uma tarefa. Ou seja, são as ações, comportamentos que nossos objetos poderão
ter. Métodos podem ou não alterar o estado (dados) de um objeto.

Tipos de Métodos

Métodos de ação (sem retorno) - void
Apenas realizam a ação sem dar nenhum resultado. A palavra void significa ausência de retorno.

Exemplo:

sentar(), levantar(), etc.

Métodos de retorno – double, int, byte, String, boolean.
Realizam a ação e ao final retornam um valor de resposta.

Exemplo:

verificarStatus(), calcularMedia(), etc.

Os métodos podem possuir ou não argumentos!

Argumentos

Métodos com ou sem argumento:
São os dados adicionais que o método requer para realizar a sua tarefa. Por exemplo, para uma pessoa realizar a ação “andar”, é necessário informar a quantidade de passos e a direção. Para uma pessoa girar, é necessário indicar a direção e os graus. Em alguns casos o método pode não ter argumento. Nesse caso, apenas utilizamos um conjunto vazio de parênteses. Para cada argumento devemos especificar o tipo de dado, assim como nos atributos.

Exemplo:

andar(passos: int, direcao: String)

Explicação:
Imagine o seguinte: Se eu falar para uma pessoa sentar, não preciso falar mais nada, isso basta, ela sabe que é para sentar-se.
Agora se eu falar para andar, a pessoa provavelmente perguntará para onde. A direção seria o argumento, para que a pessoa execute a ação, ela precisa de mais informações além da própria ação imposta.

Passagem por Parâmetro:
A passagem por parâmetro, nada mais é, do que o argumento do método, ou seja, o que deverá ser informado ao método para que o mesmo seja executado.

Exemplo:

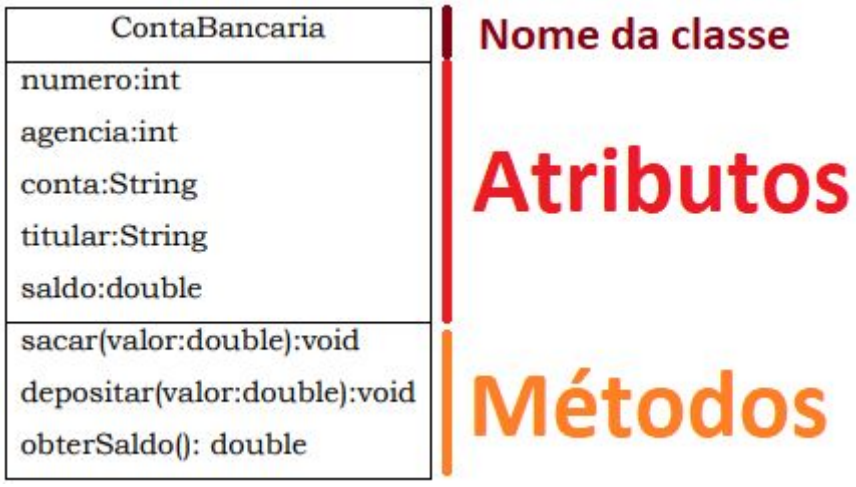
andar(direcao:String, passos:int):void

O exemplo acima é um método sem retorno e com argumento, temos dois parâmetros: **direcao e passos**.

Diagramas para representação de uma Classe no Java

Para reforçar o que falamos em aula, a linguagem Java tem como uma de suas características, trabalhar com Classes. **Classes, Objetos, Atributos e Métodos em Java**, são estruturas de um objeto que logo, logo iremos ver. As Classes podem ter características e comportamentos, permitindo assim armazenar propriedades e métodos dentro dela.

Para que possamos representar uma classe em diagramas (não fluxogramas, os fluxogramas representam o fluxo das execuções das instruções do programa), usamos o UML que é uma Linguagem Unificada de Modelagem, **neste link**, você encontra um material de apoio para leitura e entendimento, mas mais adiante, iremos trabalhar bastante com ela.



Observando a classe acima, percebemos que temos dois métodos sem retorno e com argumentos e um método com retorno e sem argumentos.

- **sacar(valor:double):void** Método sem retorno, pois o mesmo irá apenas subtrair do saldo da conta o valor pedido, não retornará nenhum resultado. O argumento é o valor que o usuário deverá informar para que seja feito o saque. Este método modifica o estado do objeto, pois irá alterar o valor do atributo "saldo".

- **depositar(valor:double):void** Método sem retorno, pois o mesmo irá apenas adicionar no saldo da conta o valor pedido para depositar, não retornará nenhum resultado. O argumento é: valor. Este método modifica o estado do objeto, pois irá alterar **o valor do atributo "saldo".**

- **obterSaldo():double** Método com retorno, pois o mesmo buscará na memória o valor que temos em saldo. E sem argumento, pois não precisamos informar nada além dos atributos que já temos na classe.

Operadores

Os operadores são utilizados para representar expressões de cálculo, comparação, condição e atribuição. Temos os seguintes tipos de operadores: de atribuição, aritméticos, relacionais e lógicos.

Operadores Aritméticos Básicos

Em nossas classes teremos métodos que irão executar alguma operação matemática. Pensando nisso, precisamos saber quais operadores matemáticos podemos utilizar em nossos cálculos. Os símbolos que representam as operações matemáticas em Java são:

Símbolo matemática	Significado	Comando/símbolo em Java
+	Adição	+
-	Subtração	-
x ou .	Multiplicação	*
÷	Divisão	/
mod	Resto da divisão	%

São executados/calculados nesta ordem de precedência:

Cálculos matemáticos

Juros e Descontos

Juros / Valor com Juros (valor a prazo)

Para calcularmos o valor de juros, basta pegarmos o valor e multiplicá-lo pelo percentual de juros, porém no programa **não poderemos utilizar o símbolo "%"** para representar percentuais, pois "%" significa módulo. Teremos que convertê-lo para um número com vírgula, para isto, basta dividi-lo por 100.

Exemplo:

10% → 0,10 → Em Java 0.1

5% → 0,05 → Em Java 0.05

Operadores de Atribuição

São utilizados para expressar o armazenamento de um valor em uma variável. Esse valor pode ser pré-definido (variante ou não) ou pode ser o resultado de um processamento.

Representação utilizando-se a notação para linguagem Java
=

Exemplo:
`nome = "Fulano de tal"`
`resultado = a + 5`
`valor = 3`

Operadores Relacionais

São utilizados para estabelecer uma relação de comparação entre valores ou expressões. O resultado dessa comparação é sempre um valor **lógico (booleano) verdadeiro ou falso.**

Operador	Representação utilizando-se a notação algorítmica	Representação utilizando-se a notação para linguagem Java	Exemplos em Java
Maior	>	>	`a > b` – Se o valor de `a` for maior do que o valor de `b`, retornará `verdadeiro`. Senão, retornará `falso`.
Maior ou igual	>=	>=	`a >= b` – Se o valor de `a` for maior ou igual ao valor de `b`, retornará `verdadeiro`. Senão, retornará `falso`.
Menor	<	<	`a < b` – Se o valor de `a` for menor do que o valor de `b`, retornará `verdadeiro`. Senão, retornará `falso`.
Menor ou igual	<=	<=	`a <= b` – Se o valor de `a` for menor ou igual ao valor de `b`, retornará `verdadeiro`. Senão, retornará `falso`.
Igual a	=	==	`a == b` – Se o valor de `a` for igual ao valor de `b`, retornará `verdadeiro`. Senão, retornará `falso`.
Diferente de	<>	!=	`a != b` – Se o valor de `a` for diferente do valor de `b`, retornará `verdadeiro`. Senão, retornará `falso`.

Operadores Lógicos

São utilizados para concatenar ou associar expressões que estabelecem uma relação de comparação entre valores. O resultado dessas expressões é sempre um valor lógico (booleano) verdadeiro ou falso.

Operador	Representação utilizando-se a notação algorítmica	Representação utilizando-se a notação para linguagem Java	Exemplos em Java
e	`.e.`	`&&`	`a = 5 && b != 9` — Caso o valor de `a` seja igual a 5 e o valor de `b` seja diferente de 9, então retornará `verdadeiro`. Caso contrário, retornará `falso`.
ou	`.ou.`	`||`	`a = 5 || b != 9` — Caso o valor de `a` seja igual a 5 ou o valor de `b` seja diferente de 9, então retornará `verdadeiro`. Se ambas as comparações retornarem `falso`, o resultado será `falso`.
não	`.não.`	`!`	`! a > 5` — Se o valor de `a` for maior do que 5, retornará `falso`. Caso contrário, retornará `verdadeiro`.

Operador	Observação
( ), [ ]	Parênteses e colchetes são usados para agrupar expressões, determinando precedência, a exemplo das expressões matemáticas.
^ ou **	Operador aritmético de potenciação.
*, /	Operadores aritméticos de multiplicação e divisão.
+, -	Operadores aritméticos de adição e subtração.
←	Operador de atribuição.
=, <, >, <=,>=, <>	Operadores relacionais.
.não.	Operador lógico de negação.
.e.	Operador lógico e.
.ou.	Operador lógico ou.