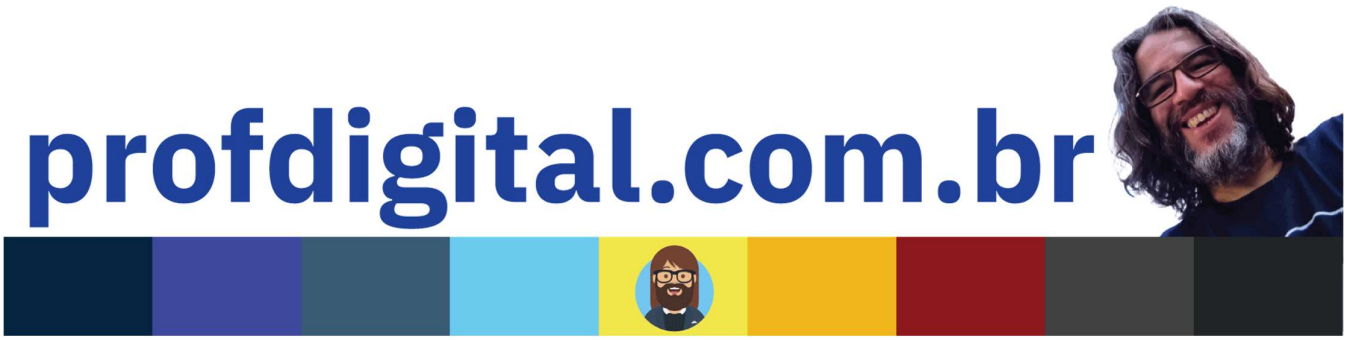
Introdução a linguagem de programação Java – Dados Primitivos

Marcos Vinícius da Silva Santos e Marcos Antonio dos Santos



ISENÇÃO DE RESPONSABILIDADE E OUTRAS INFORMAÇÕES LEGAIS

Todas as opiniões quando expressas neste documento ou no site em que é apresentado este documento não representam as opiniões de nenhuma entidade com a qual os autores tenham sido associados, são agora associados ou serão associados futuramente.

Qualquer ação envolvendo programação que você execute com as informações deste documento é feita estritamente sob a sua responsabilidade para o desenvolvimento de habilidades técnicas e profissionais, portanto, os autores não se responsabilizam por quaisquer erros ou omissões, ou pelos resultados obtidos com o uso das informações contidas nesse documento.

Todas as informações neste documento são fornecidas "no estado em que se encontram", sem a garantia dos resultados obtidos com o uso destas informações. Parte das informações aqui apresentadas foram obtidas em diversas fontes e, mesmo com todo o cuidado em sua coleta e manuseio, os autores não se responsabilizam pela publicação acidental de dados incorretos que possam levar a qualquer tipo de dano ou prejuízo.

Os autores recomendam que em caso de dúvidas o utilizador deste documento busque saná-las recorrendo à documentação oficial do fabricante que é disponibilizada nos respectivos sites.

Uma rápida visão sobre o funcionamento da memória RAM

A ***stack*** (ou pilha) é uma **forma otimizada para organizar dados** na memória RAM para que sejam alocados em sequência e **abandonados** lá durante a execução do programa. Você leu certo sim, abandonados e normalmente não há desocupação [desalocação] até o fim do programa, de certo modo isso pode custar o consumo de memória. Na ***stack*** os dados ficam em uma sequência invertida em relação a ordem da entrada.

Um dado primitivo fica em um local da memória RAM chamada ***stack*** ou ***pilha***.

Na memória RAM há também uma área que é chamado de ***heap*** (ou monte, e para constar, ninguém nunca traduz isso). O ***heap*** é a **organização de memória mais flexível** que permite o **uso de qualquer área lógica na memória que esteja disponível**.

O heap, ao contrário da stack, não impõe um modelo, um padrão de alocação de memória. Isso não é muito eficiente, mas é bastante flexível.

O heap é considerado dinâmico. Em geral você aloca ou desaloca pequenos trechos de memória, só para a necessidade do dado. Esta alocação pode ocorrer fisicamente em qualquer parte livre da memória disponível para seu processo. No heap ficam os dados de referência.

O ponto mais importante é que o heap e a pilha são termos genéricos para formas de alocação de memória RAM. Eles podem ser implementados de muitas maneiras diferentes.

Na prática, veja o seguinte trecho de código:

	`public static void main(String[] args)`
Momento 0	`{`
Momento 1	`    int i = 4;`
Momento 2	`    int y = 20;`
Momento 3	`    int soma = i + y;`
Momento 4	`    Scanner tcd = new Scanner(System.in);`
Momento 5	`}`

Observe que vai acontecer na memória RAM:

- No momento 0 avisamos que precisamos de memória stack.
- No momento 1 a variável i e seu valor são alocados na linha 1 da stack com i = 4.
- No momento 2 a variável y e seu valor são alocados na linha 2 da stack com y = 20, nesse momento a stack está sendo usada com a linha 1 contendo i = 4 e na linha 2 contendo y =20.
- No momento 3 a variável soma recebe o conteúdo da variável i adicionado com o conteúdo da variável y e o resultado deste processamento são alocados na linha 3 da stack com soma = i + y, nesse momento a stack está sendo usada com a linha 1 contendo i = 4, a linha 2 contendo y = 20 e na linha 3 contendo soma = i + y.

- No momento 4 ocorre um pedido de referência por um recurso dinâmico (Scanner) indicado por ***= new*** (em Java e na linguagem C esse pedido seria feito através da instrução ***malloc***). O ***= new*** indica uma gestão de um determinado recurso que deve ocorrer no heap. Desta forma no heap temos o tcd (que em nossas aulas representa um recurso que existe na vida real que é o teclado). Nesse momento a stack está sendo usada com a linha 1 contendo i = 4, a linha 2 contendo y = 20, a linha 3 contendo soma = i + y e na linha 4 há uma referência que aponta para o heap (= new). E no heap temos o tcd.

Como a stack é uma pilha veja como está essa pilha:

Linha	Conteúdo da stack
4	Scanner tcd = new Scanner(System.in)
3	soma = i + y
2	y = 20
1	i = 4

Já, o conteúdo do heap será apenas o tcd no exemplo anterior.

No momento 5 avisamos que não precisamos mais da stack e assim o conteúdo que estava lá é dispensado, ou seja, a stack está vazia. Porém, o heap ainda contém o tcd e certamente isso não é eficiente e com o tempo poderemos ter problemas com o uso do nosso aplicativo.

Quadro comparativo stack x heap

stack	heap
• Acesso muito rápido. • Não precisa desalocar explicitamente as variáveis. • O espaço é gerenciado de forma eficiente pela CPU, a memória RAM não se tornará fragmentada. • Variáveis locais apenas. • Limite no tamanho da pilha (dependente do sistema operacional). • Variáveis não podem ser redimensionadas.	• Variáveis podem ser acessadas globalmente. • Sem limite de tamanho de memória. • Acesso (relativamente) mais lento. • Sem garantia de uso eficiente do espaço, a memória pode ficar fragmentada com o tempo conforme os blocos de memória são alocados e, em seguida, liberados. • Você deve gerenciar a memória (você programador é o responsável por alocar e liberar variáveis). • Variáveis podem ser redimensionadas.

Lembrando que stack e heap estão disponíveis para o programa (aplicação sendo executada), CPU e Sistema Operacional.

Tipos de dados primitivos

Temos 8 tipos de dados primitivos em Java que representam dados brutos. São eles: **byte, short, int, long, float**, **double, char** e **boolean**.

Os 4 primeiros **(byte, short, int** e **long**) são utilizados para representar valores numéricos inteiros.

Os tipos numéricos **float** e **double** representam números com casas decimais (números reais ou de ponto flutuante).

O tipo **char** é indicado para representar caractere.

O tipo **boolean** é usado para indicar se uma condição é verdadeira ou não, ou, para representar dois estados, como uma luz sendo acesa (**true**) ou desligada (**false**).

		Valores possíveis (intervalo)				
Tipo	Primitivo	Menor	Maior	Valor Padrão	Tamanho	
					bits	bytes
Inteiro	byte	-128	127	0	8	1
	short	-32768	32767	0	16	2
	int	-2147483648	2147483647	0	32	4
	long	-9223372036854775808	9223372036854775807	0	64	8
Ponto flutuante	float	1.4E-45	3.4028235E38	0	32	4
	double	4.9E-324	1.7976931348623157E308	0	64	8
Caractere	char	0	65535	\0	16	2
Booleano	boolean	false (0)	true (1)	false	1	0.125

Strings

Strings (java.lang.String) são pedaços de texto armazenados em seu programa. Strings não são um tipo de dados primitivo em Java, no entanto, eles são muito comuns em programas Java.

Sugestão de leitura

https://pt.stackoverflow.com/questions/3797/o-que-s%C3%A3o-e-onde-est%C3%A3o-a-stack-e-heap

https://stackoverflow.com/questions/79923/what-and-where-are-the-stack-and-heap

https://www.codeproject.com/Articles/76153/Six-important-NET-concepts-Stack-heap-value-types#Stack%20and%20Heap

http://joinville.ifsc.edu.br/~jlcurzel/Eletronica_Digital_I/2%20-%20ELETRONICA%20DIGITAL%20DO%20CURSO%20T%C3%89CNICO/Sobre%20bits%20e%20bytes.pdf

https://www.tutorialspoint.com/display-the-minimum-and-maximum-value-of-primitive-data-types-in-java