ANÁLISE E COMPARAÇÃO DE TEMPO DE ARMAZENAMENTO DAS ESTRUTURAS DE DADOS STRING, STRINGBUFFER E STRINGBUILDER

Amanda Evangelista Lima, Ellen Tuane Silva Pinto, Wagner dos Santos Clementino de Jesus.

Universidade do Vale do Paraíba / Faculdade de Engenharias, Arquitetura e Urbanismo, Avenida Shishima Hifumi, 2911, Urbanova - 12244-000 - São José dos Campos-SP, Brasil, amandaelima1@gmail.com, ellentuanesp@gmail.com, wagner@univap.br.

Resumo - A performance de dispositivos eletrônicos é medida pelo desempenho do aparelho em relação a aplicação utilizada, tornando-se assim um fator de importância na estruturação das aplicações. Concatenação de strings via "+" realizada pela classe String, pode ser conveniente para programadores, porém essa prática pode ser custosa em termos de memória e performance. Classes de estruturas de dados como StringBuffer e StringBuilder em Java, realizam funções similares ao que se refere a concatenação de strings, mas em suas diferenças tornam-se úteis para situações divergentes. Utilizando as estruturas de dados citadas, apresentamos uma análise e comparação de tempo de armazenamento das variadas formas de concatenação de strings. Refletindo uma aplicação real, o experimento demonstra semelhanças e divergências entre as variáveis utilizadas devido às suas maneiras de utilização da memória.

Palavras-chave: String. StringBuffer. StringBuilder. Java.
Área do Conhecimento: Engenharias/Engenharia da Computação

Introdução

Atualmente com a crescente demanda de tecnologia observa-se que dispositivos eletrônicos, sendo estes aparelhos celulares, computadores, relógios inteligentes e diversos outro, esse fenômeno proporciona um processo em que a tecnologia seja cada vez mais desfrutável, ocorra a utilização de diversas aplicações de modo simultâneo, podendo causar assim lentidãos dos aparelhos e consequentemente menor desempenho de seu sistema operacional. Com isso, deve-se estimar como os dados que circundam a memória desses mecanismos podem interferir no desempenho destes aparelhos(Patterson, 2014).

A performance pode ser medida com a quantidade de aplicações acessadas simultâneas e o processamento da unidade de processamento central, CPU, compreendida no aparelho. Por intermédio desta informação pode-se afirmar que o processamento da CPU possui um valor padrão de acordo com o aparelho, a solução para obtenção de um melhor desempenho deve ocorrer na criação das aplicações. Em aplicações criadas e executadas na linguagem de programação de alto nível, como Java, há várias estruturas de armazenamento podendo estas serem específicas de numerais, palavras, letras, caracteres, listas, arrays e diversos outros. Três destas classes que são muito utilizadas são a String, StringBuffer e StringBuilder(Goodrich; Tamassia, 2013).

A String possui sua representação por intermédio de uma sequência de caracteres imutáveis, possuindo um tamanho fixo e possibilitando a concatenação de strings, a StringBuffer vem a ser similar a String em relação as suas funcionalidade, mas se diferem ao oferecer métodos que podem que possibilitam otimizar uma sequência de caracteres que são armazenado e identificados diretamente por um endereçamento de memória, em formato crescentes, ou seja, a StringBuffer pode armazenar caracteres, mas também substrings que podem ser inseridos durante ou ao final de sua execução e desta maneira apresenta crescimento dinâmico em seu comprimento. E por fim a StringBuilder também possui funcionalidades da classe String, um de seus diferenciais é de que quando se realiza concatenações não utiliza uma quantia maior de memória para seu

armazenamento, mas na realidade realiza o armazenamento mesma área de memória, diferentemente da String Buffer(Feofiloff, 2018).

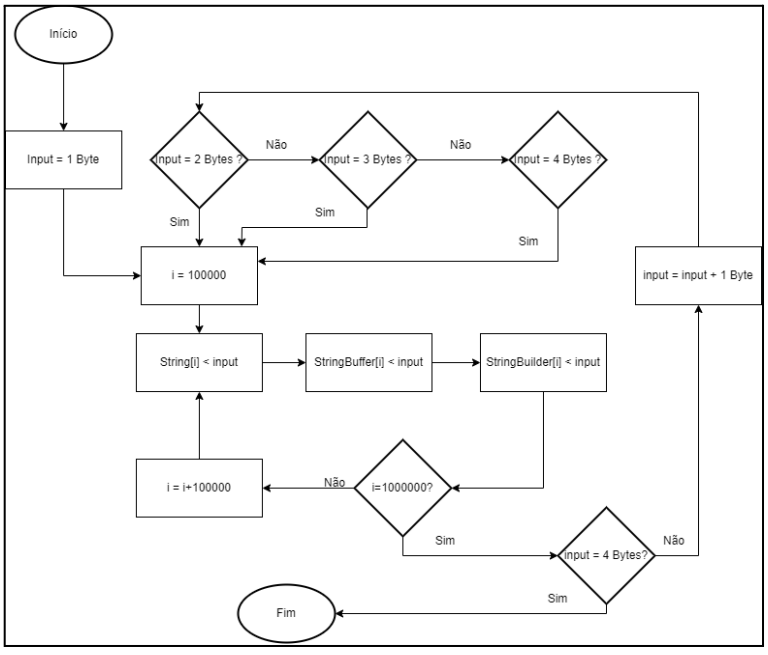
Mediante as definições anteriores e diferenças apresentadas entre as classes, este trabalho possui como proposta realizar a verificação de qual das classes é mais eficiente em seu armazenamento. Surgindo assim a ideia de criar uma codificação que verifique por intermédio de uma elevada quantidade de entradas, qual destas estruturas de dados é mais eficiente para o funcionamento de rápido armazenamento e de maior eficácia para sua utilização,efetuando métricas utilizando seus respectivos tempos de execução.

Metodologia

Os equipamentos utilizados para a realização do experimento foram um computador portátil, com processador Intel Core i7 e memória RAM de 8 gigabytes, enquanto que para o desenvolvimento do código para a realização da comparação entre as classes foi desenvolvido em linguagem Java com auxílio da IDE de desenvolvimento Eclipse versão 2019-12(Liming, 2018).

O experimento consistiu em medir o tempo de concatenação de 1 a 4 bytes. E para comparação do tempo a quantidade de concatenações foi aumentada de 100.000 em 100.000 até atingir o valor de 1.000.000 concatenações. Os resultados como classe, quantidades de concatenações, tempo e quantidade de bytes foram armazenados em arquivo csv (do inglês comma-separated values) para realização da análise dos resultados. Os dados foram analisados utilizando a linguagem de programação Python e as bibliotecas Pandas e Plotly. O fluxograma da figura 1 descreve um processo do experimento.

Figura 1 - Fluxograma que representa os procedimento experimentais do estudo



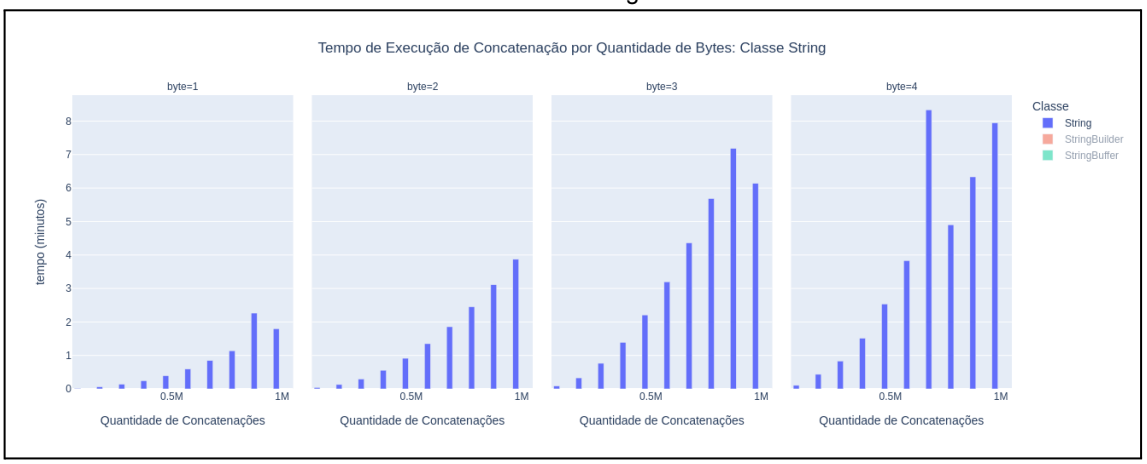
Fonte: o autor.

Importante destacar que o experimento foi realizado em modo stand-alone e monotarefa, somente o código estava sendo executado pela máquina sem nenhum outro processo acompanhando. E também, que o tempo inicial e o tempo final foram dispostos bem no início e no fim do laço ‘for’ desconsiderando qualquer outro código que venha antes ou depois, como atribuições e escritas no console, assim foram avaliados puramente a concatenação das classes.

Resultados

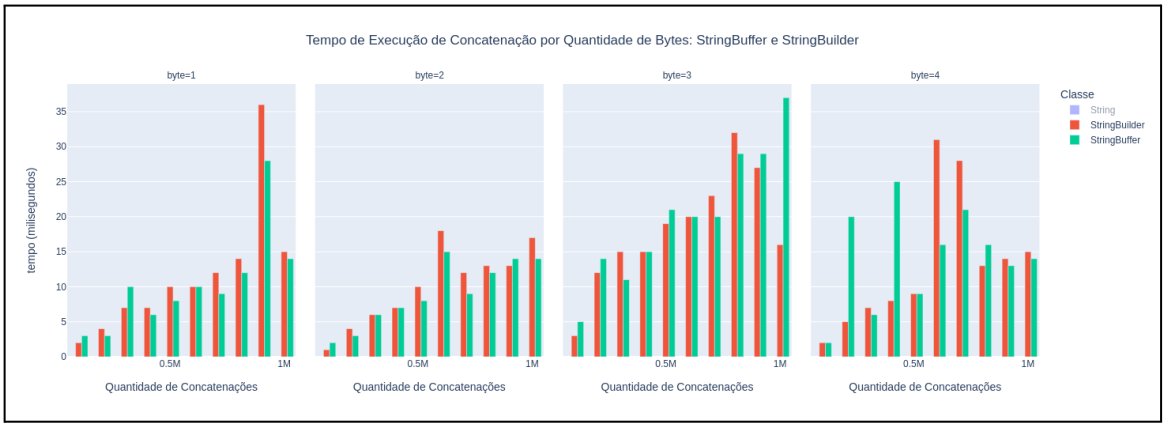
Os resultados apresentados na figura 2 e 3, tem como propósito efetuar métricas que possibilitem a visualização do tempo de execução de ocorrência das concatenações de strings. As figuras 2 e 3, logo abaixo, mostram a disposição do tempo de execução para as concentrações de 100.000 a 1.000.000 vezes, para strings de 1, 2, 3 e 4 bytes. Ao final das concatenações, as variáveis strings atingiram o tamanho de 1, 2, 3 e 4 megabyte respectivamente.

Figura 2 - Gráfico Tempo de Execução de Concatenação por Quantidade de Bytes: Classe String.



Fonte: o autor.

Figura 3 - Gráfico Tempo de Execução de Concatenação por Quantidade de Bytes: Classes StringBuffer e StringBuilder.



Fonte: o autor.

Para compreensão do comportamento do experimento como um todo, na tabela 1 são dispostas medidas estatísticas que representam cada classe. Importante ressaltar que os dados da classe String são apresentados na escala de minutos, enquanto que os dados das outras duas classes em milissegundos. Partindo disso, é possível então apontar a principal diferença entre os resultados, a classe String tem mediana de 1.37 minutos, as classes StringBuffer e StringBuilder apresentam mediana de 12.5 milissegundos. Embora a média, mediana e desvio padrão das classes StringBuffer e StringBuilder apresentam resultados próximos, há uma diferença de 4.94 milissegundos na variância entre ambos.

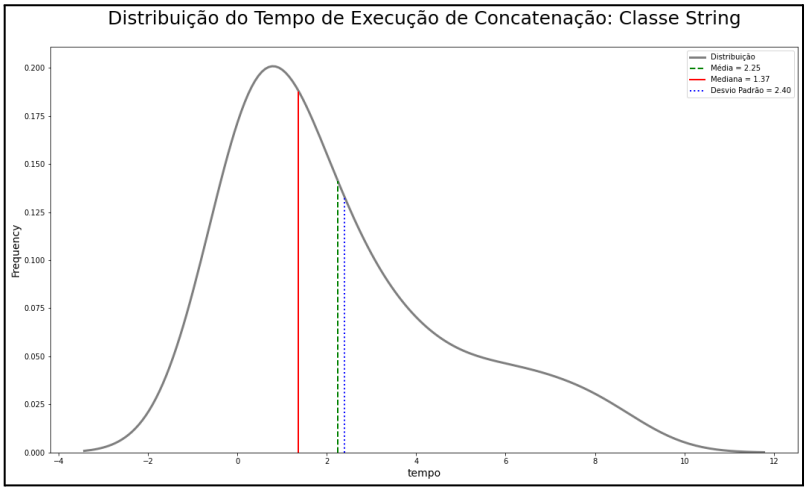
Tabela 1- Resultados Estatísticos das Classes.

Classe	Média	Desvio Padrão	Mediana	Variância
String	2.50	±2.40	1.37	5.75
StringBuffer	13.40	±8.27	12.50	68.40
StringBuilder	13.30	±8.56	12.50	73.34

Fonte: o autor.

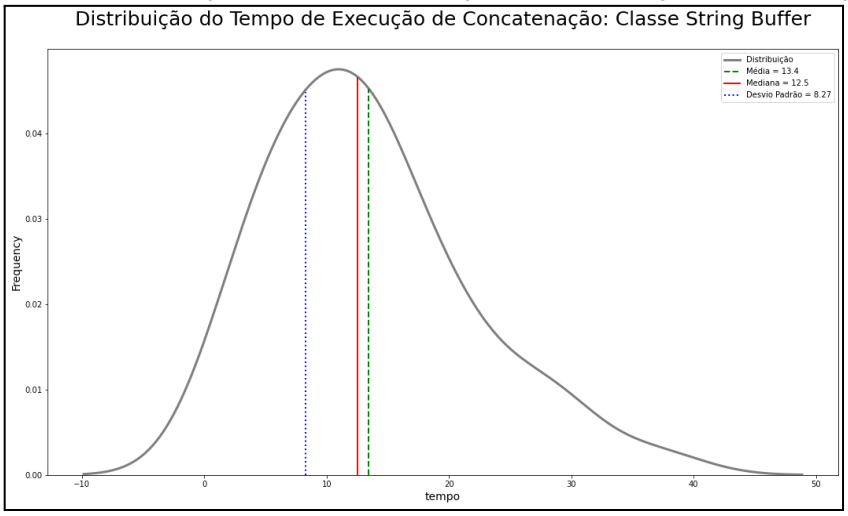
As figuras 4, 5 e 6, ilustram os resultados estatísticos descritos na tabela 1 das classes String, StringBuffer e StringBuilder, nessa ordem. Entre os valores de média e mediana apontadas nas figuras, entende-se que a mediana descreve melhor o comportamento dos dados, visto que a linha de representação em vermelho se dispõe próximo ao centro da distribuição e também devido ao fato de que a média é sensível a dados extremos ou a valores atípicos. É possível observar também, que a linha de representação de desvio padrão da Classe String está à direita do centro da distribuição, enquanto que para as outras duas classes esta mesma representação encontra-se à esquerda do centro.

Figura 4 - Gráfico Distribuição do Tempo de Execução de Concatenação: Classe String.



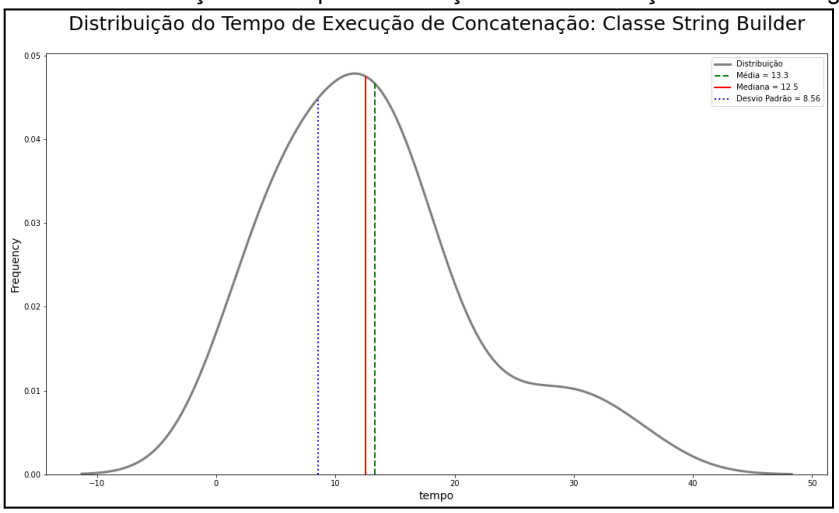
Fonte: o autor.

Figura 5 - Gráfico Distribuição do Tempo de Execução de Concatenação: Classe StringBuffer.



Fonte: o autor.

Figura 6 - Gráfico Distribuição do Tempo de Execução de Concatenação: Classe StringBuilder.



Fonte: o autor.

Discussão

O primeiro comportamento observado e esperado em comum para as três classes é que o tempo, apesar haver alguns valores atípicos, seguem uma crescente à medida que há um aumento no número de concatenações e de tamanho em bytes. O segundo, é que a partir da observação das figuras 2 e 3 é possível notar que há uma diferença considerável em relação ao tempo de concatenação da classe String em relação às outras duas classes. Uma evidência disso é que o tempo final está na escala de minutos, visto que se mantido em milissegundos os valores finais de tempo dificultariam a leitura e análise. Comparando uma amostra, 600.000 concatenações de 4 bytes na classe String leva aproximadamente 8 minutos, enquanto que para as classes StringBuilder e StringBuffer o tempo para o mesmo tamanho é de 30 e 15 milissegundos respectivamente. A justificativa para este resultado é que a classe String (possui um comportamento estático) é imutável, ou seja, o valor não pode ser mudado após a primeira atribuição. Desta maneira, ao concatenar

strings com diversas novas áreas de alocação de memória são criadas e as strings antigas perdem referência, mas continuam ocupando o espaço na memória em sua reminiscência. Portanto, a concatenação dessa forma é considerada prejudicial à performance da aplicação.

Ao realizar a comparação dos resultados entre as classes StringBuffer e StringBuilder, podemos observar que a execução da mesma função, o Stringbuilder apresenta uma ligeira rapidez sendo evidenciada na média de 13.3 milissegundos contra 13.4 milisegundos da Stringbuffer, mas apresenta medianas idênticas. Porém, a StringBuffer apresenta uma vantagem quando analisada a variância do tempo na tabela 1, 68.40 milissegundos contra 73.34 milissegundos da StringBuilder. Estatisticamente, essa diferença de 4.94 milisegundo indica o que os resultados dos experimentos da classe StringBuffer mantém-se mais próximo à média. E tecnicamente, a principal diferença entre as duas classes é que o StringBuffer é sincronizado em relação a classe StringBuilder, possibilitando assim maior eficiência na aplicação quando há diversas leituras ou modificação na mesma string. Portanto, a partir dessas evidências, pode-se dizer que a classe StringBuffer apresenta melhor desempenho quanto a concatenação de string.

Conclusão

O estudo contempla uma breve abordagem de comparação de consumo de tempo de concatenação utilizando as classes String, StringBuffer e StringBuilder em Java, visando a performance da aplicação. O experimento mostra a importância de otimizar a concentração de strings utilizando as classes em Java corretamente e com eficácia. Análises estatísticas foram combinadas junto com a disposição dos dados do experimento para providenciar informações importantes para a discussão do experimento e assim provar que os resultados experimentais mostram que a utilização da classe correta apresenta uma estratégia de otimização. O tempo de execução dos experimentos possibilitou também entender a importância da redução na alocação de memória dado à classe utilizada. Portanto, quanto à performance e otimização dos recursos de memória, os resultados encorajam a utilização da classe StringBuffer no uso direto de concatenação de strings.

Referências

Ascencio, Ana Fernanda **Estrutura de dados.** 2011. Editora Pearson / Prentice HAll

Boldi, Paolo; Vigna, Sebastiano **Mutable strings in Java: design, implementation and lightweight text-search algorithms.** Disponível em: https://www.sciencedirect.com/science/article/pii/S0167642304001005 Acesso em: 11/08/2022.

Chixiang Zhou; Phyllis Frankl **Mutation Testing for Java Database Applications.** Disponível em: https://ieeexplore.ieee.org/document/4815373 Acesso: 05/08/2022

Feofiloff, Paulo. **Strings e cadeias de caracteres.** Disponível em: https://www.ime.usp.br/~pf/algoritmos/aulas/string.html. Acesso em: 15/08/2022

Goodrich, Michael T. Tamassia, Roberto **Estrutura de Dados e Algoritmos em Java.** 5. ed. 2013

Patterson, D.A. **Organização e Projeto de Computadores: A Interface Hardware e Software.** Elsevier, 2014

Liming, Sean. Malin, John. **Java and Eclipse for Computer Science**, 2018.

Myalapalli, Vamsi Krishna; Sunitha Geloth **High performance JAVA programing.** Disponível em:https://ieeexplore.ieee.org/abstract/document/7087004?casa_token=ssgrm1MnJWoAAAAA:1GDa9t3Q8pSJqMjGBiFOf4Q0SNPnmqJB-mfgQg3_nhjDeaTtG34_35hEXjgOYfQeXV0VAEEe1V1P1zs Acesso: 05/08/2022