BIF812 Assignment 1:
Testing the Efficiency of String Concatenation Methods in Java

Christopher Eeles

February 24, 2019

1 Introduction

In this report the computational efficiency of concatenation using the "Built-in", "StringBuilder"[1] and "StringWriter"[2] class methods were compared for various string sizes and number of concatenations. The results will be interpreted in the context of memory allocation to hypothesize what may have caused observed differences in execution times for each respective method.

2 Result

Table 1: **Execution Times for Concatenation Using Three Java Concatenation Methods**

Parameters	**Built-in '+' Operator**	**StringBuilder**	**StringWriter**
100,000 times for 1k_Sample.txt	2,176,131,628,300 ns	29,441,560,000 ns	23,473,505,400 ns
10,000 times for 10k_Sample.txt	198,203,696,300 ns	3,592,489,900 ns	2,450,167,200 ns
1000 times for 100k_Sample.txt	18,417,240,500 ns	319,983,800 ns	303,948,800 ns
100 times for 1000k_Sample.txt	1,826,808,400 ns	93,045,700 ns	82,194,400 ns
10 times for 10000k_Sample.txt	226,440,300 ns	84,685,300 ns	74,745,100 ns

3 Discussion

Based on these results, it is clear that the data structures and methods used to implement a feature can have a significant effect on performace. Observations suggest that the "+" operator is relatively inefficient compared to the "StringBuilder.append()" and "StringWriter.write()" class methods. Execution times for 100,000 iterations of "+" concatenation were on the order of 10^{12} ns, while those of "StringBuilder" and "StringWriter" were on the order of 10^{10}—a difference of two orders of magnitude. The gap for 10 iterations was smaller, with one order of magnitude difference between the "+" and other methods; this suggests that the number of concatenations has a greater effect on execution time than string length.

Given that "+" operates on the string level, and strings are immutable in Java, new memory must be allocated every time a string is changed. Thus as the number of iterations increases, so too does the number of allocation events, each of which requires CPU resources that would otherwise be available to complete the concatenation operation. Additionally, the Java garbage collector has to deal with the discarded memory locations, further compounding the computational load. To avoid this wasteful memory allocation, both "StringBuilder" and "StringWriter" objects act like mutable strings, although their underlying mechanics differ. This behavior is likely achieved by use of a list object to store the characters of the string, allowing new characters to be added (i.e., concatenated) without assigning new memory locations for existing list members.

References

Oracle. (2018). Class StringBuilder. In Java API Documentation. Retrieved from `https://docs.oracle.com/javase/7/docs/api/java/lang/StringBuilder.html`

Oracle. (2018). Class StringWriter. In Java API Documentation. Retrieved from `https://docs.oracle.com/javase/7/docs/api/java/io/StringWriter.html`