Report

Experiment Report

- Exercise 3 & 4

Author: Michael Johansson
Semester: VT 2016
Subject: 1DV507

Contents

1 Exercise 3

In exercise 3 we did test on how fast StringBuilder is against using String concatenation with the + operator. We are going to do four test, two with StringBuilder and two with String concatenation, the first test on each are going to be adding one char at the time and the second one we add 80 chars at the time.

1.1 Method

We created four static methods, one for each test. In the methods we have a start and end time, inside these variables we have a for loop that either appends to the StringBuilder or concatenate to a String. The for loop iterates so many times we set in the method call. So we test how many iterations we can do while the difference between start and end time is less then one sec.

Then after we have the start time and the end time we calculate the time between them, how long the String is and how many concatenations we have done.

We run them one method at the time and just one time to get a cold start with a new Java Virtual Machine every time. With this cold start method i run each test ten times with an average on one sec.

1.2 Results

Table 1.1: Results on ten, one seconds test

Result		String Length	Number of Concatenations
Short	Concat	38.800	38.800
	Append	96.000.000	96.000.000
Long	Concat	368.000	4.600
	Append	328.000.000	4.100.000

We see from our result that using a StringBuilder to concatenate Strings together is much faster then using the + operator. I have also seen that if i did 10 test in the same JVM i got alot higher results, but i learned that is because the JVM uses "HotSpot" that are looking for code which are frequently or repeatedly executed and there for already know what the outcome should be.

1.3 StringBuilder VS + operator

The StringBuilder method by using append uses just one StringBuilder object, while the + operator uses a StringBuilder object to add two Strings. So for every loop we add one char the compiler creates a new StringBuilder object on the char to add and then finally convert the new StringBuilder into a String. So for each iteration with the + operator the compiler creates a new StringBuilder object on the char we add and then does that toString().

2 Exercise 4

In exercise 4 we did test to see how fast our sort implementations from assignment 3 is. We test our insertionSort and mergeSort implementations with one Integer and one String array. We check how large array of random Integers/Strings our sort methods can sort on one second.

2.1 Method

We have created one method to create an int array of random Integers from 0 to size of the array times 5, this is done so we get less duplicates. We also have a method that creates a String array, we got i the exercise that each string should contain ten chars. So in the method we create ten random chars using an random int between 65 and 122, with this we get a String with length ten with both lower/upper case letters and some symbols.

Then we have four methods to test each sort test. We use the same concept as the previous exercise but here i became a little lazy and did not try this exercise with cold starts of the JVM. So with these tests we run each test 10 times inside the same JVM and then calculate the average time.

2.2 Result

Table 2.2: Result of ten, one second tests

Result	InsertionSort	MergeSort
Integer	111.000	72.000.000
String	13.000	790.000

We see from our results that the mergeSort with time complexity $O(nlog(n))$ is alot faster the our insertionSort with $O(n^2)$ time complexity.