1. Experiments

There are two ways concatenations can be done. One way is by using the plus operator, for example *str = str + "… "*. Another way is by using the StringBuilder class and repeatedly use the method *append("… ")*. The task is to find the fastest approach by measuring the number of concatenations, and the length of the final string, each of them can compute in 1 second. The tests are made by concatenating and appending short strings containing only one character and long strings representing a row with 80 characters.

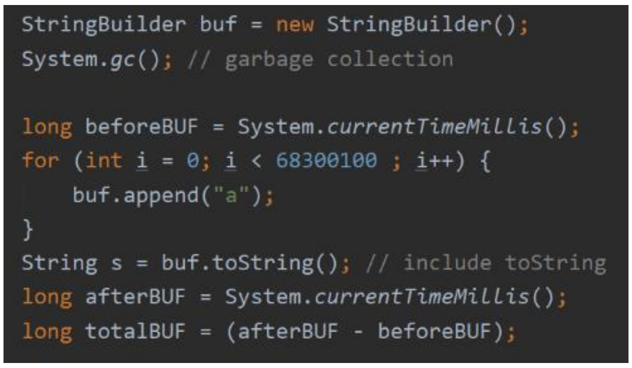
2. Experimental Setup

All experiments were done on a Lenovo ThinkPad t460s with an Intel Core i5 processor (2.50Ghz) with 12GB of memory. To avoid memory problems, garbage collection was also run just before and after each test. All other applications were closed while performing the experiments. The clock *System.currentTimeMillis()* was used in all tests.

To make sure that the JVM is fully optimized when we do the tests, we go through with a few practice tests to ensure accurate results. Each test was executed 10 times to get an average result.

Before I could do my tests, I had to check how many elements I could concatenate with the StringBuilder and with the plus-operation during 1 second. I created a for-loop and changed the termination until I got a result with approximately 1000 milliseconds (1 second).

I saved the current time in a variable, to know when the test began. Then I used my for-loop I created by the explanation above. When I executed the tests with the StringBuilder, I had to convert it to a String using *String s = buf.toString()*, to include thin in the time it took. When the for-loop and conversion was done, I saved the current time in another variable. The difference between the current time before the execution and the current time after, I could calculate the time it took to do the test. As an example, the code in the image below shows the code used for measuring the time to test the StringBuilder class by appending a short string:



```
StringBuilder buf = new StringBuilder();
System.gc(); // garbage collection

long beforeBUF = System.currentTimeMillis();
for (int i = 0; i < 68300100 ; i++) {
    buf.append("a");
}
String s = buf.toString(); // include toString
long afterBUF = System.currentTimeMillis();
long totalBUF = (afterBUF - beforeBUF);
```

3. Experimental Results

3.1 Experiment with strings containing only one character

The table below to the left shows the time in milliseconds for each test using the plus-operation approach. Every test performed 34 000 operations and every final string had the length of 34 000 characters. The average for the time was 1089 milliseconds. This means that 34 000 short strings can be concatenated for approximately 1089 milliseconds, using the plus-operator.

The table to the right shows the time in milliseconds for each test using the StringBuilder approach. Every test performed 68 300 100 appends and every final string had the length of 68 300 100 characters. The average for the time was 1007 milliseconds. This means that 68 300 100 short strings can be concatenated for approximately 1007 milliseconds, using the StringBuilder.

Test	Time
1	1072
2	1081
3	1058
4	1063
5	1176
6	1138
7	1071
8	1066
9	1068
10	1101
Average time:	1089

Test	Time
1	1057
2	1085
3	983
4	978
5	986
6	994
7	997
8	997
9	966
10	1031
Average time:	1007

3.2 Experiment with strings containing 80 characters.

The table below to the left shows the time in milliseconds and the final length of the string using the plus-operation. Every test performed 3200 concatenations and every final string had the length of 256 000 characters. The average for the time was 1367 milliseconds. This means that 3200 long strings can be concatenated during approximately 1367 milliseconds using the plus-operation.

The table below to the right shows the time in milliseconds for each test using the StringBuilder approach. Each test performed 2 149 580 concatenations and every final string had the length of 171 966 480 characters. The average for the time was 1128 milliseconds. This means that 2 149 580 long strings can be concatenated during approximately 1128 milliseconds using the StringBuilder.

Test	Time
1	1183
2	1212
3	1186
4	1195
5	1584
6	1719
7	1545
8	1591
9	1249
10	1209
Average time:	1367

Test	Time
1	1142
2	1164
3	988
4	943
5	1194
6	1122
7	1400
8	1033
9	1300
10	991
Average time:	1128

4. Discussion

The result is clearly that the StringBuilder approach is much faster than the plus-operator both concatenating the short string, containing only one character, and the longer string containing 80 characters.

This can be explained by the fact that when you concatenate two strings using the plus-operator, it produces a new String object. It does this because a String is immutable in Java. However, StringBuilders are used to create a mutable String, which means that it won’t produce a new String object every time we do a concatenation. Therefore, the StringBuilder is way faster than when you concatenate using the plus-operation approach.

A problem that occurred was when I tested how many times a long string could be concatenated for approximately 1000 milliseconds using the StringBuilder, it was very hard to get close to 1000 milliseconds. The termination 2 149 580 in the for loop gave me an average of 1128 milliseconds, and the termination 2 149 581, only one more, gave me an average of 1552 milliseconds. I chose to use the termination 2 149 580 and keep the margin of error in mind. This means that using a StringBuilder, you can concatenate strings containing 80 characters 2 149 580 times during 1128 milliseconds. Apart from this you can still see that using the StringBuilder is much better.

Additional experiments I could have done, are to measure the time it takes for the StringBuilder and for the plus-operation to perform for example 1000 concatenations with a longer string. It would be interesting to know which one is better or if they are nearly equal with fewer concatenations using both longer and shorter strings.