SPARC String Density Report (July 9, 2015)

The full set of micro-benchmarks were executed on a SPARC T5 system against a baseline JDK 9 (1.9.0-ea-b72) versus a String Density JDK built on July 8 from the String Density JDK 9 sandbox repository.

The command line options used for both the JDK 9 baseline and String Density JDK 9 are as follows:
-Xms4g
-Xmx4g
-Xmn3500m (doesn't need much old gen to run micro-benchmarks)
-XX:-UseCompressedOops (disabled to remove the compressed oops shifting instructions to make looking at generated code a little easier)
-XX:-BackgroundCompilation (disabled to help achieve more consistent and repeatable results)
-XX:+UseSerialGC (used to help achieve more consistent and repeatable results)

Summary

There are three distinct categories of observations from running the full set of micro-benchmarks and comparing the results of a JDK 9 baseline to a String Density JDK:

1. A regression observed – a micro-benchmark that appears to expose an introduced regression in the String Density JDK
   There are micro-benchmarks that clearly suggest a regression was introduced with the changes in the String Density JDK. Many of these regressions are likely overlapping issues. For example, a fix for one of the regressions may address a regression in other micro-benchmarks too.
2. On par, or String Density better — a micro-benchmark that appears to experience no regression, or an improvement in performance. In other words, micro-benchmarks that have either no regression, or better performance with the String Density JDK.
3. Exception occurred – a micro-benchmark that experienced an Exception that prevented the micro-benchmark from executing
   There are number of micro-benchmarks that experience NullPointerException and fail to execute. These issues are being addressed in the String Density repository or in the String Density micro-benchmarks suite at the time of this writing, and will be executed once the fixes have been committed to the repository.

What follows in the sections below shows the micro-benchmarks, by String API or StringB*er concatenation API, results from executing the String Density micro-benchmarks.
For the regressions, it does not go into an analysis of the root cause of a given regression. This work is on-going as of the writing of this document.

Also, for micro-benchmarks which do not have a regression, identification of the reason for a faster result when executing with the String Density JDK is not included.

This document will continue to be updates as improvements are made.

Regressions

This section contains the list of micro-benchmark tests that have been identified as having regressions and require further investigating as what can be done to close the gap of the regression.

new String(char[] c)

- ***ConstructBench.cmp2beg (size 1) - regression ~ 45% slower***
  - Constructing a new String with a non-compressible single char
- ***ConstructBench.cmp2end (size 1) - regression ~ 45% slower***
  - Constructing a new String with a non-compressible single char
- ***ConstructBench.cmp2end(size 64) - regression ~ 50% slower***
  - Constructing a new String of size 64 with a non-compressible character at the end
- ***ConstructBench.cmp2end(size 4096) regression ~ 40% slower***
  - Constructing a new String of size 64 with a non-compressible character at the end
- ***FromCharArray.test - 50% regression***
  - Constructing a new String with a non-compressible char
- **Hypothesis**: Since the regression is about the same for each sized incoming char[], it appears the regression is somewhere in the overhead introduced in getting to the String coder for UTF-16, i.e. Could it be that expensive to figure out we have a compressible String? Or, is it something else before we get to that point?

String.toCharArray()

- ***ToCharArray.test (cmp = 0.5) - 40% regression***
  - Getting a copy of a String's char[] where the char[] has a non-compressible char

- **StringBuilder.append()** variants
  - ***ConcatCharBench.test_char2_cmp1 (size 4096) - ~ 45% regression***
    - Concatenating to a non-compressible char, a compressible char[], i.e. like doing a new StringBuilder.append(non-compressibleChar).append(compressibleChars).toString()
  - ***ConcatCharBench.test_cmp1_char2 (size 4096) - ~ 45% regression***
    - Concatenating to a compressible char[], a non-compressible char, i.e. like doing a new StringBuilder.append(compressibleChars).append(non-compressibleChar).toString()

  - ***ConcatLongBench.test_long_cmp2 (size 1) - ~ 45% regression***
  - ***ConcatLongBench.test_long_cmp2 (size 64) - ~ 45% regression***
  - ***ConcatLongBench.test_long_cmp2 (size 4096) - ~ 45% regression***
    - Concatenating to a long, char[] with the non-compressible char at the end of the char[]. Like doing new StringBuilder.append(long).append(compressibleChars).toString()
  - ***ConcatLongBench.test_cmp2_long (size 1) - ~ 20% regression***
  - ***ConcatLongBench.test_cmp2_long (size 64) - ~20% regression***
  - ***ConcatLongBench.test_cmp2_long (size 4096) - ~20% regression***
    - Concatenating to a non-compressible char[] with the non-compressible char at the end, a long of the char[]. Like doing new StringBuilder.append(non-compressibleChars).append(long).toString()

- ***ConcatStringBench.test_cmp1_cmp2 (size 4096) - ~ 15% regression***
  - Concatenating to a compressible char[], a non-compressible char[] with the non-compressible char at the end of the char[]. Like doing new StringBuilder.append(compressibleChars).append(non-CompressibleChars).toString()
- ***ConcatStringBench.test_cmp2_cmp1 (size 4096) - ~ 15% regression***
  - Concatenating to a non-compressible char[] with the non-compressible char at the end, a compressible char[]. Essentially doing new StringBuilder.append(non-compressibleChars).append(compressibleChars).toString()

String.indexOf(String str)

- ***IndexOfString.img1_base1__img1 (img 64, size 1) - regression ~ 20%***
- ***IndexOfString.img1_base1__img1 (img 64, size 64) - regression ~ 35%***
- ***IndexOfString.img1_base2__img1 (img 1, size 1) - regression ~ 30%***
- ***IndexOfString.img1_base2__img1 (img 64, size 64) - regression ~ 30%***
- IndexOfString.base2_img1__img1 (img 1, size 64) - slight regression ~ 10%
- ***IndexOfString.base2_img1__img1 (img 64, size 64) - regression ~ 20%***
- ***IndexOfString.img1_base1__img1 (img 1, size 1) - regression ~ 25%***
- IndexOfString.img1_base1__img1 (img 1, size 64) - slight regression ~ 8%
- IndexOfString.img1_base1__img1 (img 1, size 4096) - slight regression ~ 8%
- IndexOfString.img1_base1__img1 (img 1, size1) - slight regression ~ 10%
- IndexOfString.img1_base1__img1 (img 1, size 64) - slight regression ~ 10%
- IndexOfString.img1_base1__img1 (img 1, size 4096) - slight regression ~ 10%
- IndexOfString.img1_base1__img1 (img 64, size1) - slight regression ~ 10%
- IndexOfString.img1_base1__img1 (img 64, size 64) - slight regression ~ 10%
- IndexOfString.base2_img2__img1 (img 1, size 64) - slight regression ~ 10%
- IndexOfString.base2_img2__img1 (img 1, size 64) - slight regression ~ 10%
- IndexOfString.img2_base1__img1 (img 1, size 64) - slight regression ~ 8%

String.compareTo()

- CompareTo.test - slight regression ~ 10%
- CompareToBench.cmp1_cmp1 (size 1) - slight regression ~ 10%
- CompareToBench.cmp1_cmp1 (size 64) - on par
- CompareToBench.cmp1_cmp2 (size 1) - slight regression ~ 10%
- CompareToBench.cmp1_cmp2 (size 64) - slight regression ~ 10%
- ***CompareToBench.cmp1_cmp2 (size 4096) - regression ~ 25%***
- CompareToBench.cmp2_cmp1 (size 1) - slight regression ~ 10%
- CompareToBench.cmp2_cmp1 (size 64) - slight regression ~ 10%
- ***CompareToBench.cmp2_cmp1 (size 4096) - regression ~ 25%***
- CompareToBench.cmp2_cmp2 (size 1) - slight regression ~ 10%
- CompareToBench.cmp2_cmp2 (size 64) - on par
- Note: The cmp1_cmp2 variant and cmp2_cmp1 variants are ~ 10% faster w/ String Density

String.codePointBefore(int index)

- ***CodePointBefore.stream - regression ~ 15%***
- CodePointBefore.spoiled - slight regression ~ 10%

String.codePointAt(int index)
- CodePointAt.spoiled - slight regression ~ 10%

String.codePointCount(int beginIndex, int endIndex)
- CodePointCount.test - slight regression ~ 10%

String.charAt(int index)
- CharAtBench.test_cmp1 (size 1, 64, 4096) - slight regression ~ 10%
- CharAtBench.test_cmp2 (size 1, 64, 4096) - slight regression ~ 10%

- CharAtStreamBench.test_cmp1 (size 1) - slight regression ~ 5%
- CharAtStreamBench.test_cmp2 (size 1) - slight regression ~10%

String.compareTo(String anotherString)
- CompareToBench.cmp1_cmp1 (size 1) - slight regression ~ 8%
- CompareToBench.cmp1_cmp2 (size 1) - slight regression ~10%
- CompareToBench.cmp2_cmp1 (size 1) - slight regression ~ 10%
- CompareToBench.cmp2_cmp2 (size 1) - slight regression ~ 8%

No Regressions
This section contains a list of micro-benchmark tests have been identified as having no regressions along with an observation note which may offer a bit more information about the test result.

- **String.charAt(int index)**
  - CharAt.stream - on par

- **String.hashCode()**
  - HashCode.test - String Density ~ 10% faster

  - HashCodeBench.cmp1 (size 1) - String Density ~ 10% faster
  - HashCodeBench.cmp1 (size 64) - on par
  - HashCodeBench.cmp1 (size 4096) - on par
  - HashCodeBench.cmp2 (size 1) - on par
  - HashCodeBench.cmp2 (size 64) - on par
  - HashCodeBench.cmp2 (size 4096) - String Density ~10% faster

new String(char[] c)
- ConstructBench.cmp1 (size 1) - on par
- ConstructBench.cmp1 (size 64) - String Density 20% faster
- ConstructBench.cmp1 (size 4096) - String Density 10% faster
- ConstructBench.cmp2beg (size 64) - String Density 20% faster
- ConstructBench.cmp2beg (size 4096) - String Density 10% faster

- **String.charAt(int index)**
  - CharAtStreamBench.test_cmp2 (size 64) - on par

- **StringBuilder.append()** variants
  - ConcatCharBench.test_char1_cmp1 (size 1) - on par
  - ConcatCharBench.test_char1_cmp1 (size 64) - String Density slightly faster
  - ConcatCharBench.test_char1_cmp1 (size 4096) - String Density 2x faster
  - ConcatCharBench.test_char1_cmp2 (size 1) - on par
  - ConcatCharBench.test_char1_cmp2 (size 64) - on par
  - ConcatCharBench.test_char1_cmp2 (size 4096) - on par
  - ConcatCharBench.test_char2_cmp1 (size 1) - on par
  - ConcatCharBench.test_char2_cmp1 (size 64) - on par
  - ConcatCharBench.test_char2_cmp2 (size 1) - on par
  - ConcatCharBench.test_char2_cmp2 (size 64) - on par
  - ConcatCharBench.test_char2_cmp2 (size 4096) - on par
  - ConcatCharBench.test_cmp1_char1 (size 1) - on par
  - ConcatCharBench.test_cmp1_char1 (size 64) - String Density slightly faster
  - ConcatCharBench.test_cmp1_char1 (size 4096) - String Density 2x faster
  - ConcatCharBench.test_cmp1_char2 (size 1) - on par
  - ConcatCharBench.test_cmp1_char2 (size 64) - on par
  - ConcatCharBench.test_cmp2_char1 (size 1) - on par
  - ConcatCharBench.test_cmp2_char1 (size 64) - on par
  - ConcatCharBench.test_cmp2_char1 (size 4096) - on par
  - ConcatCharBench.test_cmp2_char2 (size 1) - on par
  - ConcatCharBench.test_cmp2_char2 (size 64) - on par
  - ConcatCharBench.test_cmp2_char2 (size 4096) - on par

  - ConcatStringBench.test_cmp1 (size 1) - on par
  - ConcatStringBench.test_cmp1 (size 64) - String Density ~ 20% faster
  - ConcatStringBench.test_cmp1 (size 4096) - String Density ~ 2x faster
  - ConcatStringBench.test_cmp1_cmp1 (size 1) - on par
  - ConcatStringBench.test_cmp1_cmp1 (size 64) - String Density ~ 15% faster
  - ConcatStringBench.test_cmp1_cmp1 (size 4096) - String Density almost 2x faster
  - ConcatStringBench.test_cmp1_cmp2 (size 1) - on par
  - ConcatStringBench.test_cmp1_cmp2 (size 64) - on par
  - ConcatStringBench.test_cmp2 (size 1) - on par
  - ConcatStringBench.test_cmp2 (size 64) - String Density ~ 20% faster
  - ConcatStringBench.test_cmp2 (size 4096) - String Density ~ 2x faster
  - ConcatStringBench.test_cmp2_cmp1 (size 1) - on par
  - ConcatStringBench.test_cmp2_cmp1 (size 64) - on par
  - ConcatStringBench.test_cmp2_cmp2 (size 1) - on par
  - ConcatStringBench.test_cmp2_cmp2 (size 64) - on par
  - ConcatStringsBench.test_cmp2 _cmp2 (size 4096) - on par

  - ConcatIntBench.test_cmp1_int (size 1) - on par
  - ConcatIntBench.test_cmp1_int (size 64) - String Density slightly faster

- ConcatIntBench.test_cmp1_int (size 4096) - String Density 2x faster
- ConcatIntBench.test_cmp2_int (size 1) - on par
- ConcatIntBench.test_cmp2_int (size 64) - on par
- ConcatIntBench.test_cmp2_int (size 4096) - on par
- ConcatIntBench.test_int_cmp1 (size 1) - on par
- ConcatIntBench.test_int_cmp1 (size 64) - String Density slightly faster
- ConcatIntBench.test_int_cmp1 (size 4096) - String Density 2x faster
- ConcatIntBench.test_int_cmp2 (size 1) - looks good
- ConcatIntBench.test_int_cmp2 (size 64) - looks good
- ConcatIntBench.test_int_cmp2 (size 4096) - looks good

- ConcatLongBench.test_cmp1_long (size 1) - String Density slightly faster
- ConcatLongBench.test_cmp1_long (size 64) - String Density slightly faster
- ConcatLongBench.test_cmp1_long (size 4096) - String Density 2x faster
- ConcatLongBench.test_long_cmp1 (size 1) - String Density slightly faster
- ConcatLongBench.test_long_cmp1 (size 64) - String Density 30% faster
- ConcatLongBench.test_long_cmp1 (size 4096) - String Density 2x faster

- ConcatSimpleBench.base1 (size 1) - on par
- ConcatSimpleBench.base1 (size 64) - on par
- ConcatSimpleBench.base1 (size 4096) - on par
- ConcatSimpleBench.base2 (size 1) - on par
- ConcatSimpleBench.base2 (size 64) - on par
- ConcatSimpleBench.base2 (size 4096) - on par
- ConcatSimpleBench.cmp1 (size 1) - String Density 20% faster
- ConcatSimpleBench.cmp1 (size 64) - String Density 10% faster
- ConcatSimpleBench.cmp1 (size 4096) - String Density 2x faster
- ConcatSimpleBench.cmp2 (size 1) - on par
- ConcatSimpleBench.cmp2 (size 64) - on par
- ConcatSimpleBench.cmp2 (size 4096) - on par

- **String.equals(Object anObject)**
  - equals - String Density slightly faster

  - equals.EqualsBench.cmp1_cmp1 (size 1) - String Density slightly faster
  - equals.EqualsBench.cmp1_cmp1 (size 64) - String Density 2x faster
  - equals.EqualsBench.cmp1_cmp1 (size 4096) - String Density 2x faster
  - equals.EqualsBench.cmp1_cmp2 (size 1) - String Density 2x faster
  - equals.EqualsBench.cmp1_cmp2 (size 64) - String Density 8x faster
  - equals.EqualsBench.cmp1_cmp2 (size 4096) - String Density 300x faster
  - equals.EqualsBench.cmp2_cmp1 (size 1) - String Density 2x faster
  - equals.EqualsBench.cmp2_cmp1 (size 64) - String Density 8x faster
  - equals.EqualsBench.cmp2_cmp1 (size 4096) - String Density 300x faster
  - equals.EqualsBench.cmp2_cmp2 (size 1) - on par
  - equals.EqualsBench.cmp2_cmp2 (size 64) -on par
  - equals.EqualsBench.cmp2_cmp2 (size 4096) - on par

  - equals.EqualsDiffLenBench.cmp1_cmp1 - on par

- equals.EqualsDiffLenBench.cmp1_cmp2 - String Density slightly faster
- equals.EqualsDiffLenBench.cmp2_cmp1 - String Density slightly faster
- equals.EqualsDiffLenBench.cmp2_cmp2 - on par

- **String.indexOf(char c)**
  - IndexOfChar.test - on par
  - indexof.IndexOfChar.base1_img1__img1 (size 1) - on par
  - indexof.IndexOfChar.base1_img1__img1 (size 64) - on par
  - indexof.IndexOfChar.base1_img1__img1 (size 4096) - on par
  - indexof.IndexOfChar.base1_img2__img1 (size 64) - on par
  - indexof.IndexOfChar.base1_img2__img2 (size 64) - on par
  - indexof.IndexOfChar.base2_img1__img1 (size 1) - on par
  - indexof.IndexOfChar.base2_img1__img1 (size 64) - on par
  - indexof.IndexOfChar.base2_img1__img2 (size 64) - on par
  - indexof.IndexOfChar.base2_img2__img1 (size 64) - on par
  - indexof.IndexOfChar.base2_img2__img2 (size 1) - on par
  - indexof.IndexOfChar.base2_img2__img2 (size 64) - on par
  - indexof.IndexOfChar.img1_base1__img1 (size 1) - on par
  - indexof.IndexOfChar.img1_base1__img1 (size 64) - on par
  - indexof.IndexOfChar.img1_base1__img1 (size 4096) - on par
  - indexof.IndexOfChar.img1_base1__img2 (size 1) - on par
  - indexof.IndexOfChar.img1_base2__img1 (size 1) - on par
  - indexof.IndexOfChar.img1_base2__img1 (size 64) - on par
  - indexof.IndexOfChar.img1_base2__img1 (size 4096) - on par
  - indexof.IndexOfChar.img1_base2__img2 (size 64) - on par
  - indexof.IndexOfChar.img2_base1__img1 (size 64) - on par
  - indexof.IndexOfChar.img2_base1__img2 (size 1) - on par
  - indexof.IndexOfChar.img2_base1__img2 (size 64) - on par
  - indexof.IndexOfChar.img2_base1__img2 (size 4096) - on par
  - indexof.IndexOfChar.img2_base2__img1 (size 64) - on par
  - indexof.IndexOfChar.img2_base2__img2 (size 1) - on par
  - indexof.IndexOfChar.img2_base2__img2 (size 64) - on par
  - indexof.IndexOfChar.img2_base2__img2 (size 4096) - on par
  - indexof.IndexOfChar.img1_base1__img1 (size 1) - on par
  - indexof.IndexOfChar.img1_base1__img1 (size 64) - on par
  - indexof.IndexOfChar.img1_base1__img1 (size 4096) - on par
  - indexof.IndexOfString.img1_base1__img1 (img 64, size 4096) - on par
  - indexof.IndexOfString.img1_base1__img2 (img 1, size 1) - at least 2x faster on String Density
  - indexof.IndexOfString.img1_base1__img2 (img 1, size 64) - at least 2x faster on String Density
  - indexof.IndexOfString.img1_base1__img2 (img 1, size 4096) - at least 2x faster on String Density
  - indexof.IndexOfString.img1_base1__img2 (img 64, size 1) - at least 2x faster on String Density
  - indexof.IndexOfString.img1_base1__img2 (img64, size 64) - at least 2x faster on String Density

- indexof.IndexOfString.img1_base1__img2 (img64, size 4096) - at least 2x faster on String Density
- indexof.IndexOfString.base1_img2__img1 (img 1, size 1) - on par
- indexof.IndexOfString.base1_img2__img1 (img 1, size 64) - on par
- indexof.IndexOfString.base1_img2__img1 (img 1, size 4096) - String Density ~ 25% faster
- indexof.IndexOfString.base1_img2__img1 (img 64, size 1) - String Density ~ 25% faster
- indexof.IndexOfString.base1_img2__img1 (img 64, size 64) - on par
- indexof.IndexOfString.base1_img2__img1 (img 64, size 4096) - String Density ~ 20% faster
- indexof.IndexOfString.base1_img2__img2 (img 1, size 1) - on par
- indexof.IndexOfString.base1_img2__img2 (img 1, size 64) - on par
- indexof.IndexOfString.base1_img2__img2 (img 1, size 4096) - String Density ~ 20% faster
- indexof.IndexOfString.base1_img2__img2 (img 64, size 1) - String Density ~ 20% faster
- indexof.IndexOfString.base1_img2__img2 (img 64, size 64) - String Density ~ 20% faster
- indexof.IndexOfString.base1_img2__img2 (img 64, size 4096) - String Density ~ 20% faster
- indexof.IndexOfString.base2_img1__img1 (img 1, size 1) - String Density ~ 25% faster
- indexof.IndexOfString.base2_img1__img1 (img 1, size 4096) - String Density ~ 20% faster
- indexof.IndexOfString.base2_img1__img1 (img 64, size 1) - String Density ~ 20% faster
- indexof.IndexOfString.base2_img1__img1 (img 64, size 4096) - String Density ~ 20% faster
- indexof.IndexOfString.base2_img1__img2 (img 1, size1) - String Density ~ 50% faster
- indexof.IndexOfString.base2_img1__img2 (img 1, size 64) - on par
- indexof.IndexOfString.base2_img1__img2 (img 1, size 4096) - String Density ~ 20% faster
- indexof.IndexOfString.base2_img1__img2 (img 64, size1) - String Density ~ 20% faster
- indexof.IndexOfString.base2_img1__img2 (img 64, size 64) - String Density ~ 40% faster
- indexof.IndexOfString.base2_img1__img2 (img 64, size 4096) - String Density ~ 20% faster
- indexof.IndexOfString.base2_img2__img1 (img 1, size 1) - on par
- indexof.IndexOfString.base2_img2__img1 (img 1, size 4096) - ~ String Density ~ faster
- indexof.IndexOfString.base2_img2__img1 (img 64, size1) - String Density ~ 10% faster
- indexof.IndexOfString.base2_img2__img1 (img 64, size 64) - on par
- indexof.IndexOfString.base2_img2__img1 (img 64, size 4096) - String Density ~ 20% faster
- indexof.IndexOfString.base2_img2__img2 (img 1, size 1) - on par
- indexof.IndexOfString.base2_img2__img2 (img 1, size 64) - on par
- indexof.IndexOfString.base2_img2__img2 (img 1, size 4096) - String Density ~ 20% faster
- indexof.IndexOfString.base2_img2__img2 (img 64, size 1) - String Density ~ 30% faster
- indexof.IndexOfString.base2_img2__img2 (img 64, size 64) - String Density ~ 35% faster
- indexof.IndexOfString.base2_img2__img2 (img 64, size 4096) - String Density ~ 20% faster
- indexof.IndexOfString.img1_base1__img1 (img 64, size 4096) - String Density 50x faster
- indexof.IndexOfString.img1_base1__img2 (img 1, size 1) - String Density ~ 50% faster
- indexof.IndexOfString.img1_base1__img2 (img 1, size 64) - String Density ~ 10x faster
- indexof.IndexOfString.img1_base1__img2 (img 1, size 4096) - String Density ~ 50x faster
- indexof.IndexOfString.img1_base1__img2 (img 64, size 1) - String Density ~ 50% faster
- indexof.IndexOfString.img1_base1__img2 (img 64, size 64) - String Density ~ 40% faster
- indexof.IndexOfString.img1_base1__img2 (img 64, size 4096) - String Density ~ 50x faster
- indexof.IndexOfString.img1_base2__img1 (img 1, size 64) - String Density ~ 10% faster
- indexof.IndexOfString.img1_base2__img1 (img 1, size 4096) - on par
- indexof.IndexOfString.img1_base2__img1 (img 64, size 1) - String Density ~ 10% faster
- indexof.IndexOfString.img1_base2__img1 (img 64, size 4096) - String Denisty ~ 25% faster
- indexof.IndexOfString.img1_base2__img2 (img 1, size 1) - String Denisty ~ 40% faster
- indexof.IndexOfString.img1_base2__img2 (img 1, size 64) - String Density ~ 8% faster
- indexof.IndexOfString.img1_base2__img2 (img 1, size 4096) - String Density ~ 8% faster

- indexof.IndexOfString.img1_base2__img2 (img 64, size 1) - String Denisty ~ 20% faster
- indexof.IndexOfString.img1_base2__img2 (img 64, size 64) - String Density ~40% faster
- indexof.IndexOfString.img1_base2__img2 (img 64, size 4096) - String Density ~15% faster
- indexof.IndexOfString.img2_base1__img1 (img 1, size 1) - on par
- indexof.IndexOfString.img2_base1__img1 (img 1, size 4096) - String Density ~ 20% faster
- indexof.IndexOfString.img2_base1__img1 (img 64, size 1) - String Denisty 10% faster
- indexof.IndexOfString.img2_base1__img1 (img 64, size 64) - on par
- indexof.IndexOfString.img2_base1__img1 (img 64, size 4096) - String Density ~ 20% faster
- indexof.IndexOfString.img2_base1__img2 (img 1, size 1) on par
- indexof.IndexOfString.img2_base1__img2 (img 1, size 64) String Density ~ 20% faster
- indexof.IndexOfString.img2_base1__img2 (img 1, size 4096) on par
- indexof.IndexOfString.img2_base1__img2 (img 64, size 1) String Density ~ 20% faster
- indexof.IndexOfString.img2_base1__img2 (img 64, size 64) String Density ~ 20% faster
- indexof.IndexOfString.img2_base1__img2 (img 64, size 4096) String Density ~ 50% faster
- indexof.IndexOfString.img2_base2__img1 (img 1, size 1) - on par
- indexof.IndexOfString.img2_base2__img1 (img 1, size 64) - on par
- indexof.IndexOfString.img2_base2__img1 (img 1, size 4096) - String Density ~ 20% faster
- indexof.IndexOfString.img2_base2__img1 (img 64, size 1) - String Density ~ 15% faster
- indexof.IndexOfString.img2_base2__img1 (img 64, size 64) - on par
- indexof.IndexOfString.img2_base2__img1 (img 64, size 4096) - String Density ~ 30% faster
- indexof.IndexOfString.img2_base2__img2 (img 1, size 1) - on par
- indexof.IndexOfString.img2_base2__img2 (img 1, size 64) - String Density ~ 20% faster
- indexof.IndexOfString.img2_base2__img2 (img 1, size 4096) - on par
- indexof.IndexOfString.img2_base2__img2 (img 64, size 1) - String Density ~ 15% faster
- indexof.IndexOfString.img2_base2__img2 (img 64, size 64) - String Density ~ 20% faster
- indexof.IndexOfString.img2_base2__img2 (img 64, size 4096) - String Density ~ 45% faster

- **String.length()**
  - length.LengthBench.test (bias 0, count 4096) - on par
  - length.LengthBench.test (bias 0.25, count 4096) - on par
  - length.LengthBench.test (bias 0.50, count 4096) - on par
  - length.LengthBench.test (bias 0.75, count 4096) - on par
  - length.LengthBench.test (bias 1, count 4096) - on par

- **String.compareTo()**
  - CompareToBench.cmp1_cmp1 (size 4096) - String Density ~ 10% faster
  - CompareToBench.cmp2_cmp2 (size 4096) - String Density ~ 10% faster

- **String.coder selection**
  - PairSelect.baseline (bias 0) - on par
  - PairSelect.baseline (bias 0.25) - on par
  - PairSelect.baseline (bias 0.50) - on par
  - PairSelect.baseline (bias 0.75) - on par
  - PairSelect.baseline (bias 1.00) - on par
  - PairSelect.baselineRef - (bias 0) - on par
  - PairSelect.baselineRef - (bias 0.25) - on par

- PairSelect.baselineRef - (bias 0.50) - on par
- PairSelect.baselineRef - (bias 0.75) - on par
- PairSelect.baselineRef - (bias 1.00) - on par
- PairSelect.selectByFirst (bias 0) - on par
- PairSelect.selectByFirst (bias 0.25) - on par
- PairSelect.selectByFirst (bias 0.50) - on par
- PairSelect.selectByFirst (bias 0.75) - on par
- PairSelect.selectByFirst (bias 1.00) - on par
- PairSelect.selectByFirstUnsafe (bias 0) - on par
- PairSelect.selectByFirstUnsafe (bias 1.00) - on par
- PairSelect.selectByID (bias 0) - on par
- PairSelect.selectByID (bias 1.00) - on par
- PairSelect.selectByLen (bias 0) - on par
- PairSelect.selectByLen (bias 1.00) - on par

With Exceptions

The following are the micro-benchmarks that when executed result in a NullPointerException. These are known issues in the implementation and are being addressed. Once the changes are in the String Density repository, a String Density JDK will be built and these micro-benchmarks will be executed.

StringBuilder.append() with a final static String constant

- ConcatStringsConstBench.test_cmp[1|2]_constcmp[1|2] - all size variants
- ConcatStringsConstBench.test_constcmp[1|2]_constcmp[1|2] - all size variants
- ConcatStringsConstBench.test_constcmp[1|2]_cmp[1|2] - all size variants