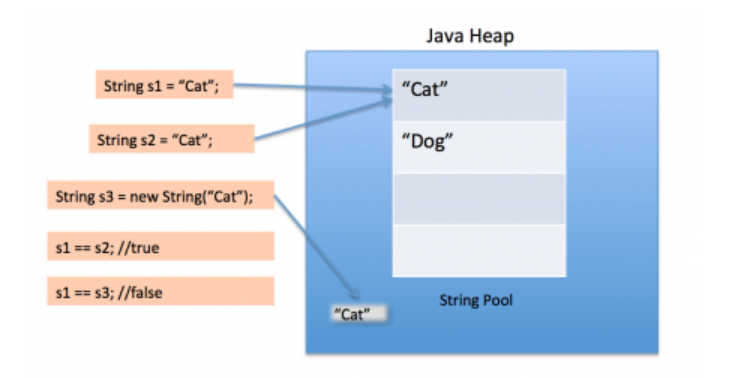
1. **Strings**: Java implements String as object rather than character array.
   1. class String implements java.io.Serializable, Comparable<String>, CharSequence
   2. Strings are immutable.
   3. **Constructors**:
      1. New String([value/charArray/strBufObj/strBuilderObj/codePoints/byteArray, start, end])
      2. We also use string literals to create string object.
   4. Since 8-bit ascii strings are common java takes byte array in constructor to initialize strings.
   5. **Methods**:
      1. **Query**: length(), isEmpty()
      2. **Extraction**: charAt(index), getChars(srcStart, srcEnd, tarArr, tarStart), getBytes(), toCharArray()
      3. **Comparision**: equals(value), equalsIgnoreCase(value), regionMatches([ignoreCase], start, str, strStart, numChars), startsWith(value), endsWith(), compateTo(value), compareToIgnoreCase(value)
         1. equals() return true if two String points to the same object or two String has same contents while ‘==’ operator returns true if two String object points to the same object but return false if two different String object contains same contents.
      4. **Searching**: indexOf(ch/string), lastIndexOf(ch/string, [startIndex])
      5. **Modifying**: substring(start, [end]), concat(), replace(origChar/origCharSeq, repChar/repCharSeq), trim(), replaceFirst(regExp, newStr), replaceAll(regExp, newStr)
         1. If beginIndex is equal to length in substring(int beginIndex), no it won't throw IndexOutOfBoundException instead it will return empty String. Same is the case when beginIndex and endIndex is equal.
         2. **Java 6**: Substring method inside String class calls String (int offset, int count, char value []) constructor to create new String object. This will also stop original string to be garbage collected, in case if doesn't have any live reference. This is clear case of memory leak in Java.
            1. **Remedy**: Trim the string, and keep size of character array according to length of substring. java.lang.String has constructor to do this.
            2. **Java 7 fix**: Apart from that String class is also changed e.g. now the char[] is not referenced when you create substring, instead a new array is created with only necessary data required by substring method.



* + 1. **Conversion**: valueOf[double/long/obj/chars]. toUpperCase(value), toLowerCase(value), split(string regExp, max)
       1. **JDK8**: join(CharSequence delim, CharSequence … strs)
  1. In java operator overloading is not supported except for + which is used for both addition and string concatenation. We can concatenate string with other datatypes also and conversion automatically handled by java. String.valueOf used to convert primate/object type to string.
  2. Object class has **public** String toString() can be override in class to return appropriate string.

1. **Compressed Strings**: From JDK 6 update 21 Strings are stored as byte[], instead of char[] – thus, saving a lot of memory.
   1. With all the info the JVM needs ready and available, the CompactString VM option is enabled by default. To disable it, we can use:
      1. +XX:-CompactStrings
   2. In case of JDK 6 Compressed Strings, a major problem faced was that the String constructor accepted only char[] as an argument. In addition to this, many String operations depended on char[] representation and not a byte array. Due to this, a lot of unpacking had to be done, which affected the performance.
2. **Compact strings**: whenever we create a String if all the characters of the String can be represented using a byte — LATIN-1 representation, a byte array will be used internally, such that one byte is given for one character. In other cases, if any character requires more than 8-bits to represent it, all the characters are stored using two bytes for each — UTF-16 representation. So basically, whenever possible, it’ll just use a single byte for each character.
   1. Most of the String operations now check the coder and dispatch to the specific implementation.
   2. With all the info the JVM needs ready and available, the CompactString VM option is enabled by default. To disable it, we can use:
      1. +XX:-CompactStrings
   3. <http://www.baeldung.com/java-9-compact-string>
3. **StringBuffer**: StringBuffer represents growable and writable character sequence.
   1. **Constructor**: StringBuffer([size/ str/chars])
      1. Default capacity is 16.
   2. **Methods**:
      1. **Query**: length(), capacity()
      2. **Modifying**: ensureCapacity(cap), setLength(len), charAt(index), setCharAt(index, ch), append(value), insert(index, str/obj/ch), reverse(), delete(start, end), deleteCharAt(loc), replace(start ,end, str)
4. **StringBuilder**: StringBuilder is same as StringBuffer but it’s not thread safe.
5. **String pools**: String Pool is possible only because String is immutable in Java and its implementation of String interning concept. String pool helps in saving a lot of space for Java Runtime although it takes more time to create the String.
   1. We can use **intern()** method to put it into the pool or refer to other String object from string pool having same value.
   2. The important difference in String pool in Java 6 and 7:
      1. String pool is relocated to Java heap space from PermGen space for storing class metadata.
      2. The default size of String pool is increased to 600013 entries from 1009 in Java 6.
      3. The -XX:StringTableSize JVM option is provided to specify the size of String pool.
      4. If you are not sure about String pool usage then you can also print string pool statistics using -XX:+PrintStringTableStatistics JVM option.
      5. All strings in the JVM string pool are eligible for garbage collection if there are no references to them from your program roots.
      6. The String pool is implemented using a HashMap. You can also customize string pool size using -XX:StringTableSize parameter. If you do provide a custom size for String pool, consider giving a prime number.
6. Important links:
   1. <http://javarevisited.blogspot.in/2012/03/why-character-array-is-better-than.html>
   2. <https://coderanch.com/t/565176/java/objects-created-string>