[AT Ralph] Java Practice 2 - String

Merry Christmas and Happy New Year!

Target: Enjoy the day, begin to love the most frequently use data type "String"

For Challenge: Finish 4 practice regarding the String

Part 1: Introduce To String

Strings, which are widely used in Java programming, are a sequence of characters. In Java programming language, strings are treated as objects.

The Java platform provides the String class to create and manipulate strings.

The way to initial the String:

```
String str = "Hello World!";

String str = "Hello ";
str += "World!";
```

```
String url = new String("Hello World!");
```

Slight different, the first way is better it only use one memory space.

Practice 1:

we have:

> *int a = 1;*
> *double b = 2.0D;*
> *char c = 'c';*
> *float d = 0.1010101f;*
> *String e = "hello";*

try combine those variable with different order, see what will happened.

(I expect the result "102.0hello0.1010101")

Part 2: String API

https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

Modifier and Type	Method and Description
`char`	**`charAt`**`(int index)` Returns the `char` value at the specified index.
`int`	**`codePointAt`**`(int index)` Returns the character (Unicode code point) at the specified index.
`int`	**`codePointBefore`**`(int index)` Returns the character (Unicode code point) before the specified index.
`int`	**`codePointCount`**`(int beginIndex, int endIndex)` Returns the number of Unicode code points in the specified text range of this `String`.

int	**compareTo**(String anotherString) Compares two strings lexicographically.
int	**compareToIgnoreCase**(String str) Compares two strings lexicographically, ignoring case differences.
String	**concat**(String str) Concatenates the specified string to the end of this string.
boolean	**contains**(CharSequence s) Returns true if and only if this string contains the specified sequence of char values.
boolean	**contentEquals**(CharSequence cs) Compares this string to the specified CharSequence.
boolean	**contentEquals**(StringBuffer sb) Compares this string to the specified StringBuffer.
static String	**copyValueOf**(char[] data) Returns a String that represents the character sequence in the array specified.
static String	**copyValueOf**(char[] data, int offset, int count) Returns a String that represents the character sequence in the array specified.
boolean	**endsWith**(String suffix) Tests if this string ends with the specified suffix.
boolean	**equals**(Object anObject) Compares this string to the specified object.
boolean	**equalsIgnoreCase**(String anotherString) Compares this String to another String, ignoring case considerations.
static String	**format**(Locale l, String format, Object... args) Returns a formatted string using the specified locale, format string, and arguments.
static String	**format**(String format, Object... args) Returns a formatted string using the specified format string and arguments.
byte[]	**getBytes**() Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.
byte[]	**getBytes**(Charset charset) Encodes this String into a sequence of bytes using the given charset, storing the result into a new byte array.
void	**getBytes**(int srcBegin, int srcEnd, byte[] dst, int dstBegin) **Deprecated.** *This method does not properly convert characters into bytes. As of JDK 1.1, the preferred way to do this is via the getBytes()method, which uses the platform's default charset.*
byte[]	**getBytes**(String charsetName) Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array.
void	**getChars**(int srcBegin, int srcEnd, char[] dst, int dstBegin) Copies characters from this string into the destination character array.
int	**hashCode**() Returns a hash code for this string.
int	**indexOf**(int ch) Returns the index within this string of the first occurrence of the specified character.
int	**indexOf**(int ch, int fromIndex) Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
int	**indexOf**(String str) Returns the index within this string of the first occurrence of the specified substring.
int	**indexOf**(String str, int fromIndex) Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

String	**intern**() Returns a canonical representation for the string object.
boolean	**isEmpty**() Returns true if, and only if, length() is 0.
int	**lastIndexOf**(int ch) Returns the index within this string of the last occurrence of the specified character.
int	**lastIndexOf**(int ch, int fromIndex) Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
int	**lastIndexOf**(String str) Returns the index within this string of the last occurrence of the specified substring.
int	**lastIndexOf**(String str, int fromIndex) Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
int	**length**() Returns the length of this string.
boolean	**matches**(String regex) Tells whether or not this string matches the given regular expression.
int	**offsetByCodePoints**(int index, int codePointOffset) Returns the index within this String that is offset from the given index by codePointOffset code points.
boolean	**regionMatches**(boolean ignoreCase, int toffset, String other, int ooffset, int len) Tests if two string regions are equal.
boolean	**regionMatches**(int toffset, String other, int ooffset, int len) Tests if two string regions are equal.
String	**replace**(char oldChar, char newChar) Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.
String	**replace**(CharSequence target, CharSequence replacement) Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence.
String	**replaceAll**(String regex, String replacement) Replaces each substring of this string that matches the given regular expression with the given replacement.
String	**replaceFirst**(String regex, String replacement) Replaces the first substring of this string that matches the given regular expression with the given replacement.
String[]	**split**(String regex) Splits this string around matches of the given regular expression.
String[]	**split**(String regex, int limit) Splits this string around matches of the given regular expression.
boolean	**startsWith**(String prefix) Tests if this string starts with the specified prefix.
boolean	**startsWith**(String prefix, int toffset) Tests if the substring of this string beginning at the specified index starts with the specified prefix.
CharSequence	**subSequence**(int beginIndex, int endIndex) Returns a new character sequence that is a subsequence of this sequence.
String	**substring**(int beginIndex) Returns a new string that is a substring of this string.
String	**substring**(int beginIndex, int endIndex) Returns a new string that is a substring of this string.
char[]	**toCharArray**() Converts this string to a new character array.

`String`	**`toLowerCase`**`()` Converts all of the characters in this `String` to lower case using the rules of the default locale.
`String`	**`toLowerCase`**`(Locale locale)` Converts all of the characters in this `String` to lower case using the rules of the given `Locale`.
`String`	**`toString`**`()` This object (which is already a string!) is itself returned.
`String`	**`toUpperCase`**`()` Converts all of the characters in this `String` to upper case using the rules of the default locale.
`String`	**`toUpperCase`**`(Locale locale)` Converts all of the characters in this `String` to upper case using the rules of the given `Locale`.
`String`	**`trim`**`()` Returns a copy of the string, with leading and trailing whitespace omitted.
`static String`	**`valueOf`**`(boolean b)` Returns the string representation of the `boolean` argument.
`static String`	**`valueOf`**`(char c)` Returns the string representation of the `char` argument.
`static String`	**`valueOf`**`(char[] data)` Returns the string representation of the `char` array argument.
`static String`	**`valueOf`**`(char[] data, int offset, int count)` Returns the string representation of a specific subarray of the `char` array argument.
`static String`	**`valueOf`**`(double d)` Returns the string representation of the `double` argument.
`static String`	**`valueOf`**`(float f)` Returns the string representation of the `float` argument.
`static String`	**`valueOf`**`(int i)` Returns the string representation of the `int` argument.
`static String`	**`valueOf`**`(long l)` Returns the string representation of the `long` argument.
`static String`	**`valueOf`**`(Object obj)` Returns the string representation of the `Object` argument.

Scared of them? No, there are the same as SQL query method, and the IDE will prompt you, it is very easy to use

```
select LENGTH('Merry Christmas and Happy New Year! ');
select CONCAT('Merry Chrismas', ' and Happy New Year! ');
select SUBSTRING('Merry Christmas and Happy New Year! ', 20);
select TRIM('Merry Christmas and Happy New Year! ');
select REPLACE('Merry Christmas and Happy New Year! ', 'and', 'and all');
select REVERSE('Merry Christmas and Happy New Year! ');
select LOWER('Merry Christmas and Happy New Year! ');
select * from insight_user where upper(user_name) = 'AUTOMATION_NOBODY';
select * from insight_user where user_name like '%automation%';
```

Let's try it

Practice 2:

We have a String

String happy = "Merry Christmas and Happy New Year! ";

please print below result:

1) trim the string

2) check if the string contains "Happy"

3) get the substring start with index 20

4) tell me the length of the happy string

5) replace the "and" with "and all"

If you have time, try below also:

6) ignore the case, is the happy string equals to the "merry christmas AND happy new year! "

7) split the happy string to happy array by space, print first split string

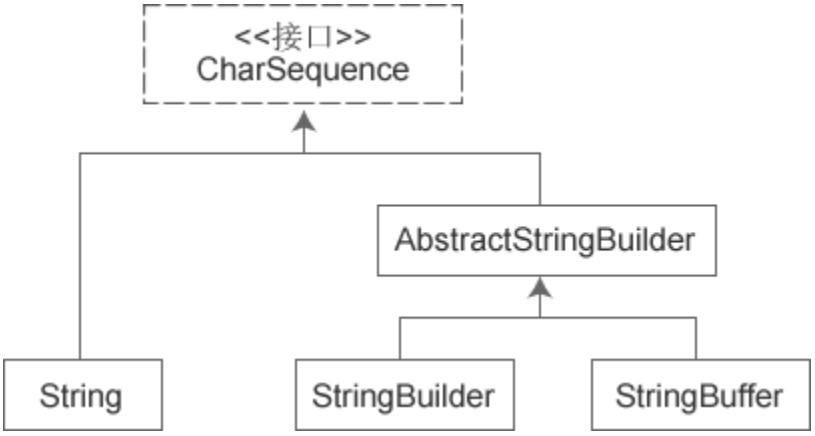
Tips: I have mark red color for the method we may need.

Part 3: StringBuilder & StringBuffer

What is StringBuilder and StringBuffer:

The **StringBuffer** and **StringBuilder** classes are used when there is a necessity to make a lot of modifications to Strings of characters.

Unlike Strings, objects of type StringBuffer and String builder can be modified over and over again without leaving behind a lot of new unused objects.




```
public class Test{
  public static void main(String args[]){
     StringBuffer sBuffer = new StringBuffer("Merry");
  sBuffer.append(" Christmas");
  sBuffer.append(" ");
  sBuffer.append("Happy");
  sBuffer.append(" ");
  sBuffer.append("New Year");
  System.out.println(sBuffer);
  }
}
```

Compare to:

```
public class Test{
  public static void main(String args[]){
     String bBuffer = new String("Merry");
  bBuffer += " Christmas";
  bBuffer += " ";
  bBuffer += "Happy";
  bBuffer += " ";
  bBuffer += "New Year";
  System.out.println(bBuffer);
  }
}
```

Practice 3:

We have a String

String fragment = "abcdefghijklmnopqrstuvwxyz";

please:

1. *add it self 20000 times by String methods*
2. *add it self 20000 time by StringBuilder Method*
3. *add it self 20000 time by StringBuffer Method*

check how long they takes, what's the different

How we choose which one to use:

Usually, for performance StringBuilder > StringBuffer > String

The main difference between the StringBuffer and StringBuilder is that StringBuilders methods are not thread safe (not synchronised).

It is recommended to use **StringBuilder** whenever possible because it is faster than StringBuffer. However, if the thread safety is necessary, the best option is StringBuffer objects.

So,

1) Process small amount of string --> use String

2) Single thread process big amount of data --> use StringBuilder

3) Multiple threads process big amount of data --> user StringBuffer

Part 4: Convert among String & StringBuilder & StringBuffer

How we convert:

```
public static void main(String[] args) {
  String s = "Hello, How are you";
  System.out.println(s);

  StringBuilder sbd = new StringBuilder(s);
  System.out.println(sbd);

  String s2 = sbd.toString();
  System.out.println(s2);

  StringBuffer sbf = new StringBuffer(s);
  System.out.println(sbf);

  String s3 = sbd.toString();
  System.out.println(s3);
 }
```

StringBuider API: https://docs.oracle.com/javase/7/docs/api/java/lang/StringBuilder.html

StringBuffer API: https://docs.oracle.com/javase/7/docs/api/java/lang/StringBuffer.html

Practice 4:

write a method "StringReverse()" to reverse the string, and write the test to test your method

for example: given string "Where is my santa gift?", after reverse, it return "?tfig atnas ym si erehW"