AP Computer Science Name: ______________________________
Chapter 9 Notes (4)

Strings

Character Methods:

When you work with characters and strings, you often need to find out whether a particular character is a digit, a letter, or something else. The `Character` wrapper class has several "public service" static `boolean` methods that test whether a character belongs to a particular category. All of these take one parameter, a `char`, and return `true` or `false`. For example:

```
boolean result = Character.isDigit(c);
                    // result is set to true if c is a digit;
                    // otherwise result is set to false
```

Other character "category" methods include `isLetter`, `isLetterOrDigit`, `isUpperCase`, `isLowerCase`, and `isWhitespace` (space, tab, newline, etc.).

There are also two methods that return the uppercase and lowercase versions of a character, if these are available. These are called `toUpperCase` and `toLowerCase`. For example:

```
char c1 = Character.toUpperCase('a');   // c1 is set to 'A'
char c2 = Character.toUpperCase('*');   // c2 is set to '*'

Scanner input = new Scanner(System.in);
String firstName = input.next();

// Change the first letter in firstName to upper case:
char c = firstName.charAt(0);
firstName = Character.toUpperCase(c) + firstName.substring(1);
```

The StringBuffer class:

`StringBuffer` objects represent character strings that can be modified. Recall that `String` objects are immutable: you cannot change the contents of a string once it is created, so for every change you need to build a new string. To change one or several characters in a string or append characters to a string, it is usually more efficient to use `StringBuffer` objects.

This is especially true if you know in advance the maximum length of a string that a given `StringBuffer` object will hold. `StringBuffer` objects distinguish between the current capacity of the buffer (that is, the maximum length of a string that this buffer can hold without being resized) and the current length of the string held in the buffer. For instance, a buffer may have the capacity to hold 100 characters and be empty (that is, currently hold an empty string). As long as the length does not exceed the capacity, all the action takes place within the same buffer and there is no need to reallocate it. When the length exceeds the capacity, a larger buffer is allocated automatically and the contents of the current buffer are copied into the new buffer. This takes some time, so if you want your code to run efficiently, you have to arrange things in such a way that reallocation and copying do not happen often.

The `StringBuffer` class has several constructors. Among them:

```
StringBuffer()              // Constructs an empty string buffer with
                            //   the default capacity (16 characters)
StringBuffer(int n)         // Constructs an empty string buffer with
                            //   the capacity n characters
StringBuffer(String s)      // Constructs a string buffer that holds a
                            //   copy of s
```

The code below shows some of `StringBuffer`'s more commonly used methods at work. As in the String class, the length method returns the length of the string currently held in the buffer. The capacity method returns the current capacity of the buffer.

In addition to the `charAt(int pos)` method that returns the character at a given position, `StringBuffer` has the `setCharAt(int pos, char ch)` method that sets the character at a given position to a given value.

`StringBuffer` has several overloaded `append(sometype x)` methods. Each of them takes one parameter of a particular type: `String`, `char`, `boolean`, `int`, and other primitive types, `Object`, or a character array (Discussed Later). `x` is converted into a string using the default conversion method, as in `String.valueOf(…)`. Then the string is appended at the end of the buffer. A larger buffer is automatically allocated if necessary. The overloaded `insert(int pos, sometype x)` methods insert characters at a given position.

The `substring(fromPos)` and `substring(fromPos, toPos)` methods work the same way as in the `String` class: the former returns a `String` equal to the substring starting at position `fromPos`, the latter returns a `String` made of the characters between `fromPos` and `toPos-1`, inclusive. `delete(fromPos, toPos)` removes a substring from the buffer and `replace(fromPos, toPos, str)` replaces the substring between `fromPos` and `toPos - 1` with `str`. Finally, the `toString` method returns a `String` object equal to the string of characters in the buffer.

```
StringBuffer sb = new StringBuffer(10);     // sb is empty

int len = sb.length();                      // len is set to 0
int cap = sb.capacity();                    // cap is set to 10

sb.append("at");                            // sb holds "at"
sb.insert(0, 'b');                          // sb holds "bat"

char ch = sb.charAt(1);                     // ch is set to 'a'
sb.setCharAt(0, 'w');                       // sb holds "wat"

sb.append("er");                            // sb holds "water"
sb.replace(1, 3, "int");                    // sb holds "winter"

String s1 = sb.substring(1);                // s1 is set to "inter"
String s2 = sb.substring(1, 3);             // s2 is set to "in"
sb.delete(4, 6);                            // sb holds "wint"
sb.deleteCharAt(3);                         // sb holds "win"

sb.append(2020);                            // sb holds "win2020"
String str = sb.toString();                 // str is set to "win2020"
```