AP Computer Science 12

String Class Notes

You have already used the String class to store text variables in your programs.
String s = "Hi there" is an example. But what is a *String*?

In the computer's memory, a String is one or more *characters*. We can say that a String is an *array* of characters. If the String *word* stores "education" in memory, it is actually storing e,d,u,c,a,t,i,o,n. Since it stores the individual characters that make the word, we can manipulate and access the individual characters of *word*.

To help you manipulate Strings, the String class has been developed with many useful methods. Around 50 methods are coded into the String class. The AP course requires you to only know the methods listed below:

returns	***Method name***	***What it does…***
int	length()	returns length of this String
String	substring(int from, int to)	returns a String that is from position *from* to position *to - 1*. **Counting starts at 0. This is very common in programming.**
String	substring(int from)	returns a String that is from position *from* to the end of the String
int	indexOf(String s) indexOf(String s, int pos)	returns the index position at which *s* first occurs in this String. **Returns -1 if not found. This is a very common idea as well.** You can also start your search starting from index *pos*
String	+ operator	returns a String that has concatenated two Strings.
boolean	equals(String)	returns true or false on a text match
int	compareTo(String)	returns an integer that is <0 , =0, or >0 depending on alphabetic comparison.
String	valueOf(int x)	returns a String object storing the value of the number passed in. Works for int, double, etc.

Any String object can use these methods.
Read the examples and make sure you agree with the output:

```
String word = "abcdefc";
String s="";
int x;

s = word.substring(3);  System.out.println(s);
//would print out "defc".  It grabs all the letters from position 3 on…

s = word.substring(1,3);  System.out.println(s);
//would print out "bc"  (grab the characters at index position 1 and stop BEFORE position 3.  3 is not included!)

x = word.indexOf("c");  System.out.println(x);
```

```
//would print out 2 since the C is FIRST FOUND at index position 3.

x = word.indexOf("c", 5); System.out.println(x);
//would print out 6.  The method looks for a "C" after (and including) position 5.

x = word.indexOf("z", 4);  System.out.println(x);
//would print out -1 since there is no "z" found after (and including) index 4.


s = UI.getString("Enter a single letter please");
x = word.indexOf(s);
if (x == -1)
  System.out.println("The letter you entered was not found in word");
else
  System.out.println("The letter you entered was found at position " + x);

//The above routine is a typical way to check if a certain letter exists in a String.
```

NOTE: If your String has 7 letters, you are using index positions 0,1,2,3,4,5,6. If you tried to use the command x = word.indexOf(999), you will compile fine, but get an error when running. There is NO position 999 in your String! The rule is that you may use up to indexOf(7) if you have 7 letters. Yes, that is 1 position past the last letter! I don't make the rules. Just memorize.

```
word = "fire";  s = "truck";
word = word + s;
System.out.println(word);
//would print out firetruck.  The + operator 'adds' Strings together.  Word is now equal
to firetruck.

word = "hello";  System.out.println( word.length() );
//would print out 5 since there are 5 characters in word

s = "apple";  word = "ball";
System.out.println(    s.compareTo(word)   );
//will print out a negative number (the value is not important!).  What the negative
value tells you is that the calling string, 'apple' is NOT LARGER than 'ball'.  "a" is
considered smaller than "b" and so a negative number is sent back to you.  If the calling
string IS LARGER, a positive number is sent back.  This method confuses students a lot,
but, it is useful for organizing words and letters alphabetically.  Below, I will print out
the word that is alphabetically larger.  Make sure you agree with the code!

String word1 = UI.getString("Enter word one");
String word2 = UI.getString("Enter word two");
if (word1.compareTo(word2) > 0)
   System.out.println(word1 + " is alphabetically larger");
else
```

```
   System.out.println(word2 + " is alphabetically larger");


//If you only care to check for equality, you use the equals method
if (word1.equals(word2) == true) {//do code}

//If you ever want to convert an int to a String, use the valueOf method.
int x = 5 + 3;
String s = String.valueOf(x);
//s now stores "8" as a String.  Notice how I can use any String method directly from
the String class.  You usually does this with the valueOf method.


//Lets use a loop with a String.  Follow along carefully.  What does this do?
String s = UI.getString("Enter a word");
int x = s.length();
for (int i=0; i<x; i++){
  String letter;
  letter = s.substring(i, i+1);
  System.out.println(letter);
}
//it prints out each letter of the word on a separate line


//I will store user names and passwords in a single String as follows:  "Bob#abc123"   or
"Jennifer#milkman"
//Notice how the # separates the two pieces of information.  Print out the user name
and password separately.
String data = "Jennifer#milkman";
String user; String pw;
int where;
where = data.indexOf("#");
user = data.substring(0, where);
pw = data.substring(where + 1);
System.out.println(user);
System.out.println(pw);
```

Notice how you have to be careful about where you start and stop your substrings. If you are off by one, it doesn't work correctly!