ICS4U - STRING VARIABLES

Java is an object-oriented language. There are eight primitive data types (byte, short, int, long, float, double, char, boolean), but an unlimited number of potential objects (or referenced data type). One of the most useful objects, which we will use extensively, is the String object.

We can declare a string variable and assign it a value in a manner which looks identical to the other data types.

int int1;
int1 = 25;
 	
String firstName, secondName;
firstName = "Robert";
secondName = new String("Lisa");
	// declares an integer variable called int1
	// assigns a value of 25 to int1
	
	// declares two string variables: firstName & secondName
	// assigns the value "Bob" to the string firstName
	// another way of assigning a value to a string variable

COMPARING STRINGS - DO NOT USE "==" (double equal) to compare strings, you must use the method "equals"

firstString.equals(secondString) → produces boolean value

if (firstname.equals(secondName)) {
  System.out.println("first and second are the same");
}
else if (firstname.equalsIgnoreCase(secondName) {
  System.out.println("first and second same case ignored");
}
	// compares firstname to secondname
	
	// compares firstname to secondname ignoring uppercase or lowercase

Using the method “compareTo”

firstString.compareTo.(secondString) → produces an integer; compares the two strings based on the unicode values of the characters in the string; compares the first mismatched pair of characters

```
public class StringExample2 {
  public static void main(String[] args) {
    String s1 = "swing";
    String s2 = "swag";

    int diff = s1.compareTo(s2);
    System.out.println (diff);
    if (diff == 0) {
      System.out.println ("your strings are the same");
    }
    else if (diff > 0) {
      System.out.println (s2 + " is alphabetically before  " + s1);
    }
    else if (diff < 0) {
      System.out.println (s1 + " is alphabetically before  " + s2);
    }
  }
}
```

```
// the "s" are the same and so are "w"
// in this example the "i" is compared to the "a" and the
difference in the unicode values is assigned to the variable
"diff"


// diff = unicode s1 minus unicode s2

//  a useful way to compare two strings for sorting
alphabetically
```

STRINGS – LENGTH OF A STRING & ONE CHARACTER OF A STRING

Using the length() method to find string length.

```
public class StringExample2 {
  public static void main(String[] args) {
   String sentence = "This is a sample string";
     System.out.println  ( sentence.length()  );                    // this would print 23
   }
}
```

Using the charAt() method to access individual characters in a string.

```
String sentence = "abcdefg";
 System.out.println(sentence.charAt(3));                    // would output d
 System.out.println(sentence.charAt(10));                  // would throw an exception
```

An example to count the number of spaces in a string:

```
String sentence = "Count the number of spaces in this String";
int spaceCount = 0;

for (int index = 0; index < sentence.length(); index++)
{
  if (sentence.charAt(index) == ' ')
    spaceCount++;
}
System.out.println("There are " + spaceCount + " spaces in " + sentence);
```

SUB-STRINGS – using the substring() method

```
String str = "abcdefgh";
 String newStr;
 newStr = str.substring (3);              // sets newStr to "defgh"   (index 3 to the end of the string)
 newStr = str.substring (1,3);            // sets newStr to "bc"       (index 1 & 2, not 3)
 newStr = str.substring (0,3);            // sets newStr to "abc"      (index 0 to 2, not 3)
 newStr = str.substring (1,4);            // sets newStr to "bcd"      (index 1 to 4, not 5)

                                          // to remove "def" from str to get "abcgh"
newStr = str.substring(0,3) + str.substring(6);
```

SEARCHING STRINGS – indexOf() & lastIndexOf()

indexOf() – searches the string in a forward manner; searches for the first occurrence

lastIndexOf() – searches the string in a backward manner (from the end)

When calling these methods you need to give at least one parameter to indicate what you are looking for. This could be either a string or a character.

You can also add a second optional parameter that specifies the index that you want to start searching from. If this second parameter is not given indexOf() starts searching from the start of the string and lastIndexOf() starts searching from the end of the string.

```
                //          1          2
                // 012345678901234567890123456
        String fruits = "apple orange pineapple kiwi";
        int findPos;
        findPos = fruits.indexOf('p');                  // would return 1
        findPos = fruits.indexOf('p', 5);               // would return 13
        findPos = fruits.lastIndexOf(' ');              // would return 22
        findPos = fruits.lastIndexOf(' ', 21);          // would return 12

        findPos = fruits.indexOf("apple");              // would return 0
        findPos = fruits.indexOf("apple", 3);           // would return 17
        findPos = fruits.lastIndexOf("pp");             // would return 18
        findPos = fruits.lastIndexOf("pp", 14);         // would return 1

        findPos = fruits.indexOf("banana");             // would return -1
        findPos = fruits.indexOf("orange", 12);         // would return -1
        findPos = fruits.lastIndexOf("peach");          // would return -1
        findPos = fruits.lastIndexOf("kiwi", 12);       // would return -1
```

OTHER STRING METHODS

String s1 = " ExAm pLe "; String s2 = s1.toUpperCase(); String s3 = s1.toLowerCase(); String s4 = s1.trim();	Uppercase & lowercase conversion s2 = " EXAM PLE " s3 = " exam ple " s4 = "ExAm pLe" → removes leading and trailing white spaces.
int number = 123; String s1 = "Number = " + number;	A string with a primitive type – converts int1 (integer) to string and joins them together to form a new string ("Number = 123")

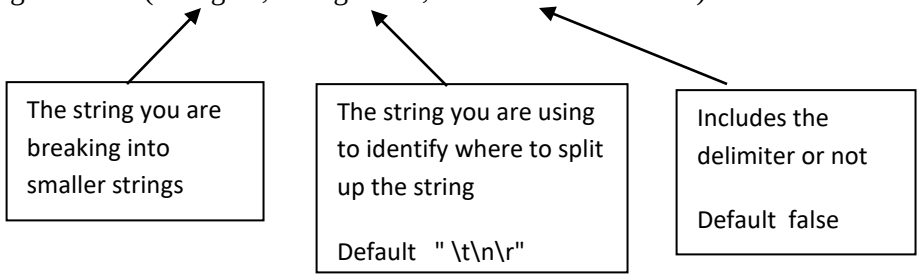
int i = 345;
double d = 45;
 	
String s1 = String.valueOf(i);
String s2 = String.valueOf(d);
	// use the String.valueOf() method to convert primitive data type to a string.
	
	// assigns the string"345" to s1
	// assigns the string"45.0" to s2 since d is a double its value is 45.0

Date currentDate = new Date(); String dateAsString = currentDate.toString();	To convert non-primitive variables to strings we can use the toString() method available for most objects

String s1 = "51";
int x = Integer.parseInt(s1);
String s2 = "4.6";
double y = Double.parseDouble(s2);
	To convert a string to an integer
	
	To convert a string to a double

THE StringTokenizer CLASS – breaks down large strings into smaller ones; useful for converting a full line of text into separate words

StringTokenizer(String str, String delim, boolean includeDelim)



StringTokenizer phrase = new StringTokenizer("One, Two, Three, Four!", "\n\t\r,!?.", true);

Once you have created a StringTokenizer object, you can retrieve the tokens from this object by calling the **nextToken()** method. The first time you call this method it will return the first token. Each subsequent call returns the next token in the list. If there are no tokens available, nextToken()will throw a NoSuchElementException. To avoid these exceptions we can call the boolean method **hasMoreTokens()**to check if there are any more tokens left. Combining these two methods we can look at all of the tokens in a StringTokenizer using a simple loop.

StringTokenizer phrase
= new StringTokenizer("Words in a String");
 	
while (phrase.hasMoreTokens())
{
System.out.println(phrase.nextToken());
}
	No delimiter provided – default delimiters if not provided are " \t\n\r" (space, tab, newline, return)
	
	The code would output the following:
	
	Words
	in
	a
	String

System.out.println(phrase.countToken());	countTokens() will tell you how many tokens or words there are. In this example the output would be 4.