HW#4

Java's primitive data types (int, double, etc.) can hold a limited range of values. int, for example, can't hold values greater than 2,147,483,647. Since we may want to work with values larger than this (say, for example, the US national debt or the number of atoms in one gram of carbon), programmers sometimes use other types (such as String) for large integers. For example:

```
String worldPopulation = "6993309969";
String rubiksCubeStates = "43252003274489856000";
```

To make use of numbers encoded this way, we need a way to add them. In grade school you learned how to add two large numbers one digit at a time (a technique appropriately called grade-school addition). You write the two numbers out, one on top of the other, then work from the right to the left adding the individual digits of the numbers and adding carry when needed. For example 137 + 864:

```
 1 1 1

   1 3 7
 + 8 6 4
 -------
 1 0 0 1
```

Your job in this problem is to write a method with this signature:

```
private String addIntegerStrings(String firstNum, String secondNum)
```

that accepts as input two Strings encoding integers with the same number of digits, then uses grade-school addition to add the two integers represented by those strings. For example:

addIntegerStrings("44", "99")	returns "143"
addIntegerStrings("1234", "4321")	returns "5555"
addIntegerStrings("137", "42")	will cause an error since 137 and 42 don't have the same number of digits and you don't need to handle that case.
addIntegerStrings("137", "042")	returns "179"
addIntegerStrings("123123123123123123123", "119119119119119119119")	returns "242242242242242242242"

You may assume the following:

- The strings firstNum and secondNum must be the same length to do the addition but the number of digits is variable. If the two strings are of different lengths return the string “error”.
- The integers encoded by firstNum and secondNum are nonnegative, so every encoded integer will be at least 0.
- All the characters in the two input strings are digits, so there are no commas, minus signs, plus signs, etc. Thus you will never get "1,000,000" or "-3" as inputs.

Your solution must add the numeric strings using the grade-school algorithm rather than converting the string to a number and using built-in arithmetic.

Write the method and a main routine to demonstrate its operation. Make sure that your name, assignment number, date, and short description are at the top of your file. Submit the .java file to D2L.

Additional info.

This assignment uses strings. Recall (see pg 159-163) that characters in a string are not accessed like an array and are immutable (can't be resized or changed) but there are methods to access and manipulate strings. Also, there is a StringBuffer class (see the box on page 163) that can be used when a variable length string is needed (such as assembling a result string as you add individual digits together).

There are many string functions in the java libraries but here are functions, and examples using them, that fit nicely with this assignment:

(assume these variable declarations have been made:)

```
int  n, number, length;
str1 = “345";
```

char charAt(index)

This is a method associated with an object of type String. It Obtains the character at the specified index and returns a char (see pg 160). A type cast can be used to convert the ASCII character code value to an integer as shown in the example below where n is the index of the desired character.

```
number = (int) str1.charAt(n);
```

Since only ASCII numerals 0 to 9 will be used, and they have numeric values 48 to 57, we can convert the ASCII character code to a number from 0 to 9 by subtracting 48.

On page 192 mention is made of classes that “wrap” primitive data types like integers in objects. Methods are defined for these objects such as the method toString() which can be used to convert an integer value 0 to 9 to a character.

Assume a variable named sum is created with type Integer (i.e. a wrapper) and that there is a string named my_string

```
Integer sum;
```

To convert an integer value from 0 to 9 to an ASCII character (or string of characters if the integer value is greater than 9) do this:

```
my_string = sum.toString();
```

While Strings objects are fixed in length at the time of creation, there is a class called StringBuffer which creates string objects that can be changed (see page 163). One of the StringBuffer constructors allows specifying how long the string is at creation:

```
StringBuffer resultString = new StringBuffer(length);
```

This class has an insert method that allows insertion of a character in the StringBuffer object. `resultString.insert(n,”5”);       // where n is the index`

Combine the conversion from integer with string insertion:

```
resultString.insert(n,sum.toString());   //
```

If you create the resultString one character longer than the two string that are being added together there will be enough space for a carry out to generate another digit.

Another handy method that comes with StringBuffer is reverse which reverses the order of characters in a string:

```
        resultString.reverse();          // no parameters needed, just this statement.
```

And the contents of a StringBuffer can be printed just like a regular string using the System.out.println or System.out.print methods.