JAVA PROGRAMMING

(340)

Regional – 2016

**Production Portion:**

Program 1: Base K Addition (335 points)

 ***TOTAL POINTS (335 points)***

**Judge/Graders: Please double check and verify all scores and answer keys!**

Property of Business Professionals of America.

May be reproduced only for use in the Business Professionals of America

*Workplace Skills Assessment Program* competition.

You will have 90 minutes to complete your work.

Your name and/or school name should NOT appear on any work you submit for grading.

Create a folder on the flash drive provided using your contestant number as the name of the folder. Copy your entire solution/project into this folder. You must submit your entire solution/project so that the graders may open your project to review the source code. You must ensure that the files required to run your program are present and will execute on the flash drive provided. Note that the flash drive letter may not be the same when the program is graded as it was when you created the program. It is recommended that you use relative paths rather than absolute paths to ensure that the program will run regardless of the flash drive letter. The graders will *not* compile or alter your source code to correct for this. Submissions that do *not* contain source code will *not* be graded.

Assumptions to make when taking this assessment:

* The input file(s) will contain only ASCII characters.
* A test input file will be available and will be named, “basekaddition.txt”.

Development Standards:

* Standard name prefixes must be utilized for variables.
* All subroutines, functions, and methods must be documented with comments explaining the purpose of the method, the input parameters (if any), and the output (if any). If you create a class, then you must use Javadoc comments.

Your application will be graded on the following criteria:

**Solution and Project**

The project is present on the flash drive \_\_\_\_ 10 points

The project main class is named baseKaddition \_\_\_\_ 10 points

**Program Execution**

The program runs from the USB flash drive \_\_\_\_ 15 points

If the program does *not* execute, then the remaining items in this section receive a score of zero.

The program runs and reads the input file \_\_\_\_ 20 points

The program displays an error message if the command line is empty \_\_\_\_ 20 points

The program displays an error message if the file cannot be found \_\_\_\_ 20 points

The program displays the base of the numbers \_\_\_\_ 20 points

The program displays the first number to add \_\_\_\_ 20 points

The program displays the second number to add \_\_\_\_ 20 points

The program displays the correct addition of the numbers \_\_\_\_ 30 points

The program displays all the output for all of the numbers in the file \_\_\_\_ 40 points

**Source Code Review**

The source code is properly commented

 A comment containing the contestant number is present \_\_\_\_ 10 points

 Methods and code sections are commented \_\_\_\_ 20 points

A method exists to perform the display of output \_\_\_\_ 20 points

A method exists to perform the addition \_\_\_\_ 20 points

Code uses try…catch for file exception \_\_\_\_ 20 points

Code uses a consistent variable naming convention \_\_\_\_ 10 points

Methods are used appropriately \_\_\_\_ 10 points

**Total Points: 335 points**

# Base K Addition

The simple idea is that the program will be able to perform addition, in decimal, binary, octal, hexadecimal (base 16), or any other base that it is given.

In this problem, you will be given two numbers in base K and asked to add them.

**Input:**

The input to this problem consists of a text file. The first line of the text file contains a single decimal integer that indicates the number of times the program should perform the addition. The next three lines will repeat for the number indicated in the first line.

The first line of these three numbers contains a single decimal integer indicating the base in which the arithmetic is to be done. This integer will be between 2 and 20 inclusive. The second and third lines of text each consists of a single non-negative number in that base.

In keeping with mathematical tradition, numerals 10-19 (if needed) will be indicated by the letters A-J respectively – upper case only. The sum of the numbers represented in the second and third lines will be smaller than the decimal value 2,014,003,021.

**Output:**

The output consists of four lines of text. The first line describes the base of the numbers. The second line shows the first number. The third line shows the second number. The fourth line shows the result of the addition.

*Example:*

The base of the numbers to add is 10

The first number is 173

The second number is 169

The sum is: 342

**Requirements:**

1. You must create an application named baseKaddition.
2. Your name and contestant number must appear as a comment at the top of the main source code file.
3. The program must read the name of the input file from the command line. If the file does not exist, then the program needs to display an appropriate message and exit. If the file name is not entered on the command line, then the program should display an appropriate message and exit.
4. The program will perform the addition correctly for however many numbers are in the input file.
5. The program must implement methods for output and the addition.
6. The program will display the output like the example above for all of the values in the input file.

Note to Graders: The contestants have been given a file named basekaddition.txt. You will run their program with a file named graderaddition.txt. This will produce different results than the file they were given. Thus you will be able to detect if they hardcoded the name of the file into their program. You must copy graderaddition.txt into the folder where their program runs from. Listed below are test cases and their results. The first line is what you type at the command prompt. The second line is the output produced by the program. The wording does *not* have to be exact.

When running the program, perform the following test cases:

Test Case #1:

baseKaddition

Please enter the name of the text file to use on the command line.

Test Case #2

baseKaddition grader.txt

The file grader.txt was not found.

Test Case #3

baseKaddition graderaddition.txt

The base of the numbers to add is 10

The first number is 173

The second number is 169

The sum is: 342

The base of the numbers to add is 8

The first number is 173

The second number is 147

The sum is: 342

The base of the numbers to add is 14

The first number is 2BC

The second number is 84D

The sum is: B2B

The base of the numbers to add is 2

The first number is 00101101

The second number is 11100101

The sum is: 100010010

The base of the numbers to add is 16

The first number is 000BBBB

The second number is 8888888

The sum is: 8894443

The base of the numbers to add is 3

The first number is 222222

The second number is 111111

The sum is: 1111110

The base of the numbers to add is 9

The first number is 804

The second number is 027

The sum is: 832

The base of the numbers to add is 7

The first number is 0

The second number is 0

The sum is: 0

The base of the numbers to add is 4

The first number is 123123

The second number is 000123

The sum is: 123312

The base of the numbers to add is 20

The first number is ABCDEF

The second number is 0GH0IJ

The sum is: AB8ECD

The base of the numbers to add is 15

The first number is 965DF

The second number is 01120

The sum is: 97710

Sample Code, student code should and will vary:

**import** java.io.FileNotFoundException;

**import** java.io.FileReader;

**import** java.util.Scanner;

/\*\*

 \* BPA Java Programming Contest Problem : Base K Addition

 \*

 \* **@author** contestant number goes here

 \* **@version** 2014 September 10

 \*/

**public** **class** Add {

 **public** **static** **void** main(String[] args) **throws** FileNotFoundException {

 String sum = "";

// if the file was not put onto the command line prompt show an error message

 **if** (args.length > 0) {

 // try to open the file that was entered on the command prompt

 **try** {

 FileReader fr = **new** FileReader(args[0]);

 Scanner sc = **new** Scanner(fr);

 //get the number of times to do the summation

 **int** numberOfTimesToRun = sc.nextInt();

 **for** (**int** j = 0; j < numberOfTimesToRun; j++) {

 //get the base and two numbers

 **int** radix = sc.nextInt();

 String first = sc.next();

 String second = sc.next();

 //ensure the two numbers are the same length

 **while** (first.length() < second.length()) {

 first = "0" + first;

 }

 **while** (second.length() < first.length()) {

 second = "0" + second;

 }

 sum = *sumNumbers*(radix, first, second);

 *showOutput*(radix,first,second,sum);

 }

 //if the file can not be found, show an error message

 } **catch** (FileNotFoundException ec) {

 System.*out*.println("The file " + args[0] + " was not found.");

 }

 } **else** {

 //Display an erro if there was not a file entered on the command line

 System.*out*.println("Please enter the name of the text file to use on the command line.");

 }

 }

 // This method will produce the output for all of the data pre-processing

 // It takes as input the base of the numbers, the first and second number, and the result

 // It does not return anything

 **public** **static** **void** showOutput(**int** radix, String first, String second, String sum)

 {

 System.*out*.println("The base of the numbers to add is " + radix);

 System.*out*.println("The first number is " + first);

 System.*out*.println("The second number is " + second);

 System.*out*.println("The sum is: " + sum + '\n');

 }

 // This method will perform the summation of the two numbers

 // It takes as input the base, and the two numbers

 // It returns the sum of the addition

 **public** **static** String sumNumbers(**int** radix, String first, String second)

 {

 String numerals = "0123456789ABCDEFHIJ";

 String sum="";

 **int** carry = 0;

 //loop for the length of the numbers

 **for** (**int** i=first.length()-1; i>=0; i--) {

 **int** a = numerals.indexOf(first.charAt(i));

 **int** b = numerals.indexOf(second.charAt(i));

 //add the characters together

 **int** c = a+b+carry;

 //adjust for greater than base

 **if** (c>=radix) {

 carry = 1;

 c -= radix;

 } **else** {

 carry = 0;

 }

 //sum up the numbers using the numbers and letters

 sum = numerals.charAt(c) + sum;

 }

**if** (carry == 1) {

 sum = "1" + sum;

 }

 **return** sum;

 }

}