COP 3330
Homework 1

Out: 1/9/12 (Monday)
Due: 1/27/12 (Friday) at 11:55 PM WebCourses time
Late submissions accepted until 1/29/12 (Sunday) at 11:55 PM (with penalty)

Objective

1. To help you become comfortableusing the basic control structures in Java.
2. To familiarize you with basic Java I/O.
3. To familiarize you with using arraysand Strings in Java.

Notes

The sample input and output provided for each problem is not comprehensive. When we grade your programs, we will test them using *different* data. You should test your program using different data, too.

Code

You may discuss high-level ideas of how to solve these problems with your classmates, but do not discuss actual source code (except for boiler-plate code such as opening a file for reading, reading user input, etc.). All source code should be yours and yours alone. **Do not:**

- Share your source code with anyone
- Ask anyone to share his/her source code with you
- Ask/pay a programming forum/community to write your homework for you
- Pair program
- Do anything else that you know is deceptive, dishonest, or misleading in any way

Problem A: FizzBuzz

In this question, you will implement a class *FizzBuzz*, to solve a variant of the FizzBuzz problem, which turned into a minor internet sensation among the programming community a few years ago. (http://imranontech.com/2007/01/24/using-fizzbuzz-to-find-developers-who-grok-coding/)

Input (Standard input)

The input contains multiple data cases, each consisting of a single line. Each line contains two positive integers *a* and *b*, separated by a single space. These integers will not exceed 10000, and *a* will never be greater than *b*. End of input will be indicated by a case with $a = 0$ and $b = 0$. Your program should detect this situation and stop immediately, without processing this case.

Output (Standard output)

For each test case, your program must print the integers in the range *a* to *b*, one per line. However, if the integer is a multiple of 3, you must **print 'Fizz' instead of the integer**, and if it is a multiple of 5, you must **print 'Buzz' instead of the integer**. For integers that are multiples of both 3 and 5, **print 'FizzBuzz' instead of the integer**. Leave a single blank line between the outputs for each test case (see sample).

Your program must follow the sample output EXACTLY to receive full credit.

Sample I/O

See `FizzBuzz.sample.in` and `FizzBuzz.sample.out`. Your output to screen must match **exactly** with `FizzBuzz.sample.out` in order to receive full credit.

Problem B: Kth Elements

In this question, you will implement a class *KthElements* to print every *kth* element of a given array for a given value of *k*. For instance, for the following input:

```
Array: 12, -2, 5, 42, 6, 15, 25
    k: 2
```

You would print the 2nd, 4th, and 6th elements of the array (for this problem you may assume you always start counting at 1).

```
-2, 42, 15
```

If *k* were instead 3, you would print the 3rd and 6th elements:

```
5, 15
```

Naturally, if *k* is 1, you would print every element of the array. Remember when solving this problem that arrays in Java start at index 0 and not 1.

Input (Standard input)

The input contains multiple data cases, each consisting of three lines. Each case will begin with a single integer *n* indicating the number of elements in the array, where *n* will be at most 100. This will be followed a line containing exactly *n* integers between -100 and 100, each separated by a single space. On the next line will be a single integer *k* between 1 and *n*, inclusive. End of input will be indicated by a case with *n* = 0. Your program should detect this situation and stop immediately, without processing this case.

Output (Standard output)

For each test case, your program must print every *kth* element of the given array, separated by commas and a single space (but be careful: the last element should *not* be followed by a comma or any trailing spaces). The bounds on the input guarantee that each test case will have at least 1 element printed. Each test case should be printed on its own line.

Your program must follow the sample output EXACTLY to receive full credit.

Sample I/O

See `KthElements.sample.in` and `KthElements.sample.out`. Your output to screen must match **exactly** with `KthElements.sample.out` in order to receive full credit.

Problem C: Stringing

Implement a class *Stringing* to perform various basic operations on an array of Strings. Your program will be given a list of strings, which you must read into an array. Then it will be given a list of commands, indicating various operations you must perform on one or more strings in the array.

Input (Standard input)

The input will begin with a single integer *n*, the size of the array of Strings. The next *n* lines will each contain a single string, representing the elements of the array in order.

Each line that follows will be a series of space-separated integers of the form <*command*> <*parameters*>. That is, the first integer on each line represents the command: 1 = length, 2 = substring, 3 = count, 4 = concatenate. Depending on which of these commands is used, some integers will follow as parameters for that command (*k, a, b,* etc., in the explanation below). In particular, each line will be one of these four types:

1 *k*
Here *k* will be an integer between *0* and *n-1* – or to put it simply, a valid index into the array. This command should be interpreted as 'print the length of the *k*th element of the array'.

2 *k a b*
Here *k* is a valid index into the array, and *a* and *b* are valid start and end indices into the *k*th element of the array. This should be interpreted as, 'Print the substring of the kth array element that starts at index *a* and and ends before index *b'*. (Basically, *a* and *b* are the arguments to the substring() function – see the Javadocs to understand the details.)

3 *l*
Here *l* can be any positive integer no greater than 100. This should be read as 'count the number of strings in the array that have at most *l* characters'. (That includes strings with exactly *l* characters.)

4 *i j*
Here *i* and *j* will be valid indices into the array, with i <= j. This should be interpreted as 'Print the concatenation of the strings from index *i* to index *j* in the array.

The input is terminated by the end of file. All input is guaranteed to be valid, per the above commands and parameters.

Output (Standard output)

For each command, print a single line corresponding to the output of the line. See the sample output for any clarifications. Note that you must follow the sample output **exactly** to receive full credit.

Sample I/O

See `Stringing.sample.in` and `Stringing.sample.out` for sample input and output. Your output to screen must match **exactly** with sample output in order to receive full credit.

Problem D: Caesar Cipher

Implement a class *Cipher* that performs a Caesar cipher (also known as a shift cipher) on a given message in plaintext.A Caesar cipher takes as input a shift parameter *k*and a message in plaintext and produces as output the message in ciphertext. In particular, each character is shifted forward by *k* letters, where the last and first letter of the alphabet are considered adjacent. For instance, for $k = 3$, the encryption scheme is:

```
Plaintext:  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Ciphertext: D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
```

The message "`A SAMPLE PLAINTEXT`" would be encrypted as "`D VDPSOH SODLQWHAW`"

Input (Standard input)

There are multiple test cases in the input, one per line. Each line corresponds to one Caesar cipher to perform, which contains, in order, the shift parameter *k* and the plaintext. The shift parameter and the plaintext are separated by a single space. The shift parameter is an integer, where $0 \leq k \leq 100$, and the plaintext will contain only the characters 'A' – 'Z' (ASCII codes 65-90) and the space character (ASCII code 32). Each word in the plaintext will be separated by a single space. Each plaintext message will contain at most 1,000 characters, and will not contain any leading or trailing spaces (except the space between the shift parameter and the first word of the plaintext). The input is terminated by the end of file.

Output (Standard output)

For each test case, print the ciphertext. You must follow the sample output **exactly** to receive full credit.

Sample I/O

See `Cipher.sample.in` and `Cipher.sample.out` for sample input and output. Your output to screen must match **exactly** with sample output in order to receive full credit.

Deliverables

Submit each of source code files (**FizzBuzz.java, KthElements.java, Stringing.java, Cipher.java**) over Webcourses as a separate attachment (i.e. do *not* create an archive/zip of the files). **In particular, *do not* submit any .class files!!! This will result in 0 credit for the assignment.** You must send your source files as an attachment using the "Add Attachments" button. Assignments that are typed into the submission box will **not** be accepted.

Restrictions

Your program must compile using Java 6.0 or later on the command prompt. It's okay to develop your program using the IDE of your choice, but the TAs will use the command prompt to compile your code and run the programs. Your code should include a header comment with the following information:

Your name
Course number, section number
Description of the code

You **WILL** lose credit if this information is not found in the beginning of each of your programs. You should also include **inline comments** to describe different parts of your program.

Execution and Grading Instructions

1. Download the source *.java* files and place in a folder.
2. Check the source code of each program to make sure it contains header comments, inline comments and reasonable use of variable names.
3. Copy the sample files (everyone has these) and judging files (only TAs have these) into the directory.
4. Run the sample execution script (posted with assignment, students can do this part too).
5. Run the judge execution script (only TAs have this, useless for students since they don't have the judge files).
6. If all sample and judge output matches the student's program output, give full credit for execution. Otherwise, give partial credit base on the situation.

Points Breakdown

Total Points: 160

1. FizzBuzz.java
    - Execution: 30
    - Documentation: 10
2. KthElements.java
    - Execution: 30
    - Documentation: 10
3. Stringing.java
    - Execution: 30
    - Documentation: 10
4. Cipher.java
    - Execution: 30
    - Documentation: 10