ICE 1: Welcome Java!

Resource: http://blue.smu.edu.sg/is442/ice1-resource.zip.

For all exercises, difficulty level is shown by using * beside the question (* Easy, ** Medium, *** Hard).

1. **[Difficulty: *]** Implement the method `sumOfDigits` that returns the sum of all the digits of that number. For example, `sumOfDigits(123)` returns `6` (1 + 2 + 3). For negative numbers, the sum is the same value as that for a positive number. For example, `sumOfDigits(-354)` returns `12` (3 + 5 + 4).

2. **[Difficulty: *]** Write a program that repeatedly prompts the user for two words. It then checks whether the first character of the two words (assume case sensitive) are the same. If they are the same, the program displays `"bingo"` and terminates the program. Otherwise it displays `"try again"` and prompts for inputs again from the user. Sample run of the program is shown below:

```
D:\IS442\ICE1>java Q2
Enter first word:Java
Enter second word:Python
try again

Enter first word:Java
Enter second word:Jack
Bingo

D:\IS442\ICE1>
```

3. **[Difficulty: **]** Write a program that reverses the words inside a sentence. Any punctuation mark can be treated as part of a word. You can also assume that there is a single space between any two consecutive words, and there is no extra space in the beginning or the end of the sentence. Two sample runs of the program are shown below:

```
D:\IS442\ICE1>java Q3
Enter sentence:this is difficult
difficult is this

D:\IS442\ICE1>java Q3
Enter sentence:but i've figured it out
out it figured i've but
```

4. **[Difficulty: *]** Write a program that repeatedly prompts the user for numbers. Once the user types any number less than zero, this program must terminate and display the product of all non-negative numbers typed. Two sample runs of the program are below:

```
D:\IS442\ICE1>java Q4
Enter number:4
Enter number:2
Enter number:3
Enter number:-1
4 x 2 x 3 = 24

D:\IS442\ICE1>java Q4
Enter number:-2
No positive number is entered
```

5. **[Difficulty: **]** Write code that contains only 1 "for" loop to produce each of the following outputs. Below shows the sample Run 1 for part (A):

```
D:\IS442\ICE1>java Q5
Enter size:3
a b c

D:\IS442\ICE1>java Q5
Enter size:27
a b c d e f g h i j k l m n o p q r s t u v w y x z a
```

Hint for part A:

```
jshell> char x = 'a'
x ==> 'a'

jshell> (int)x   // note: char can be cast to int, not String
$2 ==> 97

jshell> char x = 'b'
x ==> 'b'

jshell> (int)x
$4 ==> 98
```

	when size is 3	**when size is 5**	**when size is 7**
(A)	a b c	a b c d e	a b c d e f g
(B)	123 234 345	123 234 345 456 567	123 234 345 456 567 678 789
(C)	1 12 123	1 12 123 1234 12345	1 12 123 1234 12345 123456 1234567

6. **[Difficulty: ***]** You need to implement a static method that attempts to align two strings. This method takes in two parameters which are both of type String. Let's call the first parameter `str1` and the second `str2`. You can assume that `str1` is longer than or of the same length as `str2`. The method should try to align `str2` with `str1` and insert spaces for additional characters in `str1`. In the end the method ***returns*** `true` if there is an alignment, and meanwhile the method ***prints out*** the alignment. If no alignment can be found, the method ***returns*** `false` and ***prints out*** nothing.

   An alignment between `str1` and `str2` means by inserting spaces into `str2`, each character in `str2` (excluding the inserted spaces) can be matched in `str1` in the same position. For example, if `str1` is `"Recess Week"` and `str2` is `"RcWk"`, then we can align `str2` to `str1` as follows:

   ```
   Recess Week

   R c    W  k
   ```

   Another example would be `"Java Programming Course"` and `"amg"`:

   ```
   Java Programming Course

    a         m   g
   ```

   Notice that in this case there are multiple ways to align `"amg"` to `"Java Programming Course"` (e.g. the `'a'` in `"amg"` could be matched to the first, the second or the third `'a'` in `"Java Programming Course"`). The method only needs to print out one valid alignment.

7. **[Difficulty: *]** Write a program that outputs the song of the "12 days of Christmas" using switch-case. Hint: Take advantage of fall-through cases.

   On the first day of Christmas my true love sent to me
   A partridge in a pear tree

   On the second day of Christmas my true love sent to me
   Two turtle doves, and
   A partridge in a pear tree

   On the third day of Christmas my true love sent to me
   Three french hens
   Two turtle doves, and
   A partridge in a pear tree
   ...

   Reference: https://www.dltk-holidays.com/xmas/midi/twelve_days.htm

   Hint:

```
// an array is like a list in Python
String[] days = {
        "first", "second", "third", "fourth", "fifth", "sixth", "seventh",
        "eighth", "ninth", "tenth", "eleventh", "twelfth" };
System.out.println(days[0]);  //prints first
System.out.println(days.length); // prints the number of elements
```

8. **[Difficulty: **]** Counting in binary is similar to counting in any other number system. Beginning with a single digit, counting proceeds through each symbol, in increasing order. Decimal (or base-10) counting uses the symbols **0** through **9**, while binary only uses the symbols **0** and **1.** Read more about how a decimal number can be converted to its binary equivalent: http://www.wikihow.com/Convert-from-Decimal-to-Binary

   Write a program called **Q8BinaryConverter.java** that converts a positive integer number to its binary equivalent. You can check if your conversion is correct using your windows' calculator (select "programmer mode").

   This shows a sample run of your program:

```
D:\IS442\ICE1>java Q8BinaryConverter
Enter decimal number:32
Binary equivalent of 32 is 100000

D:\IS442\ICE1>
```

   Let's do some spy work here. Spies encrypt textual messages before sending them over to their correspondent. A very simple algorithm proposed here is to convert a String message into a series of zeros and ones (binaries). The recipient of the binary message will need to convert it back to the original textual message.

   Using the ASCII chart, characters are mapped to their corresponding decimal values. What you want here is the binary value of the characters. A sample of ASCII characters and their binary equivalent are shown below:

Note: The binary values need to be padded with zeros in front if they are not of length 8.

Character	ASCII value (in decimal)	Binary equivalent of decimal value
A	65	01000001
B	66	01000010
C	67	01000011
D	68	01000100
E	69	01000101
D	70	01000110

For this exercise, write a program called `Q8TextToBinary.java` that takes any textual message and prints out its binary equivalent. Write a similar program called `Q8BinaryToText` which takes in a binary message and prints out the textual representation. This shows a sample run of your programs:

```
D:\IS442\ICE1>java Q8TextToBinary
Enter the text message:This is funny =)
0101010001101000011010010111001100100000011010010111001100100000011001100111010
0
1011011100110111001111001001000000011110100101001

D:\IS442\ICE1>java Q8BinaryToText
Enter the binary message:
0101010001101000011010010111001100100000011010010111001100100000011001100111010
1011011100110111001111001001000000011110100101001
This is funny =)
```

Enter the following binary representation into your working program to find out what it says (copy from this document; right-click and select Paste in your cmd window):

```
0100000101100011011000110110111110111001001100100011010010110111001100111001000
0001110100011011110010000001010100011010010110111101100010011001010010110000010
0000010010100110000101110110011000010010000001101000011000010111001100100000001
1000100110010101100101011011100010000001110100011010000110010100100000011011100
1110101011011010110001001100101011100100010000000110001001000000110111011100
1000100000001100100010000001101101011011110111011001101110100001000000111000001100
1111011100000111010101101100011000010111001000100000011011000110000101101110011
1001110111010101100001011001110110010100100000011000100110000101110011011010010
0110001101100001011011000110110001111001001000000111001101101001011011100110000
1101100101001000000110100101110100011100110010000001100011011100100110010101100
0001011101000110100101101111011011100010000001101001011011100010000001110100010
1010000110010100100000011011010110100101100100001011010011100100110000000100111
0111001100101110
```

Hint: Check out the `toBinaryString` and `parseInt` methods from the `Integer` class.

9. **[Difficulty: **]** This question explores the differences between short-circuit and normal logical operators. Given the following:

```
char c = 'a';
int age = 9;
```

Do you think that the following fragments will produce different outputs?

```
// fragment A
if (c == 'a' && ++age == 10){
    age++;
}
System.out.println(age);
```

```
// fragment B
if (c == 'a' & ++age == 10){
    age++;
}
System.out.println(age);
```

```
// fragment C
if (c != 'a' && ++age == 10){
    age++;
}
System.out.println(age);
```

What about these?

```
// fragment D
if (c != 'a' || ++age == 10){
    age++;
}
System.out.println(age);
```

```
// fragment E
if (c == 'a' || ++age == 10){
    age++;
}
System.out.print(age);
```

```
// fragment F
if (c != 'a' | ++age == 10){
    age++;
}
System.out.print(age);
```

Try them out in a program and check the output against your expected output.

There is a difference between the && (boolean "short-circuit" AND) and the & (boolean AND) operators, as well as the || (boolean "short-circuit" OR) and | (boolean OR) operators. Read about the differences and understand why the value of age is different for the code fragments above. This may help:
https://users.drew.edu/bburd/JavaForDummies4/ShortCircuitEval.pdf

Note:
It is a bad idea to write confusing code. This is an example of something that you shouldn't do:

```
if (c == 'a' && ++age == 10){...
```

because whether age is incremented depends on whether the first condition (`c == 'a'`) is true or false. If you want to increase the value of age if (c == 'a') is true, the following makes it explicit and clear to the reader:

```
if (c == 'a'){
    ++age;
}
if (c == 'a' && age == 10){...
```

10. **[Difficulty: **]** This question introduces you to the break and continue keywords. When dealing with loops, you can use the break and continue keywords to get out of the loop. Read about how to use these keywords in statements within a loop block. It's important to know the difference between these 2 keywords. This may help:
https://docs.oracle.com/javase/tutorial/java/nutsandbolts/branch.html

(a) Consider the following code:

```
01  public class Q10 {
02      public static void main (String [] args){
03          for (int i = 0; i < 3; i++){
04              System.out.println("A");
05              for (int j = 0; j < 3; j++){
06                  System.out.println("B");
07                  if (i == 1 || j == 1){ //breaks the loop when i or j = 1
08                      break;
09                  }
10                  System.out.println("i: " + i + ", j: " + j);
11              }
12              System.out.println("C");
13          }
14      }
15  }
```

Trace through the code manually and determine what will be printed out. Then compile and execute it to check if the output matches your expected one.

(b) Change line 8 to

```
continue;
```

Trace through the code to predict the output again.

11. **[Difficulty: **]** This question introduces you to labels.

Using the break or continue keyword within a nested (inner) loop can only bring you out of the inner loop. You can actually break or continue to the outer loop immediately by labeling the outer loop with an identifier and breaking or continuing directly to the outer loop.

Read about using labels for loops:
https://www.cs.umd.edu/~clin/MoreJava/ControlFlow/break.html

(a) Consider the following code:

```
01  public class Q11 {
02      public static void main (String [] args){
03          outer:
04          for (int i = 0; i < 3; i++){
05              System.out.println("A");
06              for (int j = 0; j < 3; j++){
07                  System.out.println("B");
08                  if (i == 1) {
09                      break outer;
10                  }
11                  if (j == 1) {
12                      break;
13                  }
14                  System.out.println("i: " + i + ", j: " + j);
15              }
16              System.out.println("C");
17          }
18      }
19  }
20
```

Trace through the code manually and determine what will be printed out. Then compile and execute it to check if the output matches your expected one.

(b) Change line 9 to

```
continue outer;
```

Trace through the code to predict the output again.

12. **[Difficulty: **]** This question introduces you to enums.

    One limitation of the switch/case feature of Java is that it can only be used to compare equality of variables of the following types only: primitive type - byte, short, int and char & the reference type - String.
    You can use enumerated types in a switch statement as well.

    Enumerated types (or enums) is a new feature introduced since Java 5. It is a more intuitive and flexible replacement of integer constants. You can try the following:

```
// Shape.java
public enum Shape{
    RECTANGLE,
    CIRCLE,
    TRIANGLE
}
```

```
// Test.java
public class Q12 {
    public static void main (String[] args){
        Shape s = Shape.CIRCLE;
        switch (s){
            case RECTANGLE:
                System.out.println("it has 4 edges");
                break;
            case CIRCLE:
                System.out.println("its round");
                break;
            case TRIANGLE :
                System.out.println("it has 3 edges");
        }
    }
}
```

Read up about enums in Java and learn how to use them. These links may help:

1. http://download.oracle.com/javase/1.5.0/docs/guide/language/enums.html
2. https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html

13. **[Difficulty: ***]** This question introduces you to bit manipulation in Java.

    Computers work on ones and zeros(binary numbers). Understand more about it here:

    a) https://www.youtube.com/watch?v=Xpk67YzOn5w
    b) https://youtu.be/ewokFOSxabs

    Bit manipulation operations allow us to perform low-level operations directly on the binary representation used in integers.

    Unary bitwise complement operator (~)
    It makes every "0" a "1" and every "1" a "0". For example, a byte contains 8 bits; applying this operator to a value whose bit pattern is "00000000" would change its pattern to "11111111".

```
jshell> int x = 0b0 // 0b means that we are declaring a binary value
x ==> 0

jshell> ~x
$2 ==> -1

jshell> Integer.toBinaryString(~x)
$3 ==> "11111111111111111111111111111111"
```

    Bitwise AND operator (&)
    All the bits of both numbers are compared one by one and it results in 1 only when the two compared bits are both equal to 1. Otherwise, it results in 0.

```
  0110010
& 1100000
---------
  0100000
```

```
jshell> int a = 0b0110010
x ==> 50

jshell> int b = 0b1100000
y ==> 96

jshell> Integer.toBinaryString(a & b)
$6 ==> "100000"
```

    To check for odd/even: if num & 1 not equals zero then num is ODD otherwise it is EVEN.

Bitwise OR operator (|)

All the bits of both numbers are compared one by one and it results in 1 only when either one of the two compared bits is 1. Otherwise, it results in 0.

```
  0110010
| 1100000
---------
  1110010
```

```
jshell> int a = 0b0110010
x ==> 50

jshell> int b = 0b1100000
y ==> 96

jshell> Integer.toBinaryString(x | y)
$7 ==> "1110010"
```

Bitwise Exclusive OR (XOR) operator (^)

All the bits of both numbers are compared one by one and it results in 1 only when **ONLY** one of the two compared bits is 1. Otherwise, it results in 0.

```
  0110010
^ 1100000
---------
  1010010
```

```
jshell> int a = 0b0110010
x ==> 50

jshell> int b = 0b1100000
y ==> 96

jshell> Integer.toBinaryString(x ^ y)
$8 ==> "1010010"
```

Reference:

1. Read more about bit flags here: https://eddmann.com/posts/using-bit-flags-and-enumsets-in-java/

Left-Shift Operator (<<)

It moves each digit in a number's binary representation left. The last bit in the direction of the shift is lost, and a 00 bit is inserted on the other end.

```
2467       00000000000100110100011
2467 << 3  00000001001101000110000
2467 << 5  00000100110100011000000
```

```
jshell> int x = 0b100110100011
x ==> 2467

jshell> Integer.toBinaryString(x << 3) // leading 0s are not shown
$11 ==> "100110100011000"

jshell> Integer.toBinaryString(x << 5)
$12 ==> "10011010001100000"
```

What is the output for the following piece of code? **Doing a left-shift is equivalent to performing a mathematical operation. What is that operator (+, -, /, *)?**

```
int x = 3;
System.out.println(x << 1); // x <operator> 2
System.out.println(x << 2); // x <operator> 2 <operator> 2
System.out.println(x << 3); // x <operator> 2 <operator> 2 <operator> 2
```

Signed Right-Shift Operator (>>)

It moves each digit in a number's binary representation right. The last bit in the right direction of the shift is lost. If the number is negative, then 1 is used as a filler and if the number is positive, then 0 is used as a filler on the left.

```
2467       00000000000000000000100110100011
2467 >> 3  00000000000000000000000100110100 ("011" is pushed off)
2467 >> 5  00000000000000000000001001101("00011" is pushed off)
```

```
jshell> int x = 0b100110100011
x ==> 2467

jshell> Integer.toBinaryString(x >> 3)
$13 ==> "100110100"

jshell> Integer.toBinaryString(x >> 5)
$14 ==> "1001101"
```

```
-2467      11111111111111111111011001011101
-2467 >> 3 11111111111111111111111011001011("101" is pushed off)
-2467 >> 5 11111111111111111111111110110010     ("11101" is pushed off)
```

Reference:

1. https://www.youtube.com/watch?v=NLKQEOgBAnw
2. https://www.cs.cornell.edu/~tomf/notes/cps104/twoscomp.html

Unsigned Right-Shift Operator (>>>)

It moves each digit in a number's binary representation right. The last bit in the direction of the shift is lost, and zeros are inserted on the other end.

```
2467       00000000000000000000100110100011
2467 >>> 3 00000000000000000000000100110100 ("011" is pushed off)
2467 >>> 5       00000000000000000000001001101("00011" is pushed off)
```

```
jshell> int x = 0b100110100011
x ==> 2467

jshell> Integer.toBinaryString(x >>> 3)
$13 ==> "100110100"

jshell> Integer.toBinaryString(x >>> 5)
$14 ==> "1001101"
```

```
-2467           11111111111111111111011001011101
-2467 >>> 3           00011111111111111111111011001011 ("101" is pushed
off)
-2467 >>> 5     00000111111111111111111110110010 ("11101" is pushed off)
```

Reference:

1. https://medium.com/@codingfreak/bit-manipulation-interview-questions-and-practice-problems-27c0e71412e7