Welcome Back

Welcome back to your third year of computer programming with Java! If you're in this class, it is assumed that you have a serious interest in computer science / programming – I *hope* you spent some time coding this summer. Regardless, the problems below should serve as a decent review.

1. Create a class **Runner.java** that has a `public static void main(String[] args)` method.

2. Create a class **WelcomeBack.java** that will contain the methods below. In the `main` method of the Runner class, make a WelcomeBack object – you will use it to test / call its methods.

3. Complete the method `String getMiddle(String str)` that will return the middle character of an odd length String or the middle two characters of an even length String.

4. Complete the method `int[] sumNumbers(int n)` that returns an array containing the sum of all consecutive numbers from 0 to `n` (inclusive). The array should have a length of `abs(n)` + 1.

```
sumNumbers(5) >>> [0, 1, 3, 6, 10, 15] //0, 0 thru 1, 0 thru 2, 0 thru 3, etc.
```

5. Complete the method `int sumDigits(int num)` that will return the sum of all the digits in `num`.

```
sumDigits(234) >>> 9 //the modulus operator will help for getting individual digits
```

6. Complete the method `int keepSummingDigits(int num)` that will **repeatedly** sum all digits, until the result has only one digit.

```
keepSummingDigits(29) >>> 2 //2 + 9 == 11, 1 + 1 == 2
```

7. Complete the method `String getIntersection(int[] a, int[] b)` that will return a String containing the ***intersection*** of the two arrays. The intersection of two arrays is defined as a set of ***unique*** elements that exist in both arrays (they can be in any order).

```
getIntersection(new int[] {1,2,3,4}, new int[] {9,0,4,3,4,1}) >>> 134

/* Concatenating a String and an int will result in a String.  For example, 4 + "?"
will result in "4?".  ("" + 6137).length() would return 4 */
```

8. Complete the method `int sumDigitsRecur(int num)` that implements the `sumDigits` method seen previously in without using loops (using recursion). It should keep summing digits until there is only one digit left like `keepSummingDigits`. For the crafty, this can be done in one line, using a ***ternary operator***. Google it!

```
// ternary operator format: (testCondition)? valueForTrue : valueForFalse
//   ex 1, make a = abs(a): int a = (a < 0)? -a : a;
//   ex 2, do a null check: String value = (object == null)? null : object.getValue();
```

9. Complete the method `int sumWithoutCarry(int a, int b)` that adds two numbers without "carrying". Account for numbers of different length.

```
sumWithoutCarry(861, 3450) >>> 3211
```

10. Complete the method `int buySell1(int[] stock)`, where the element at index `i` in `stock` represents the price of a share of a particular stock on day `i`. The method should return the maximum potential profit, assuming you only made one transaction (i.e. one purchase, one sale).

```
[3, 4, 3, 2, 1, 5] >>> 4 //if you bought at 1 and sold for 5
[5, 4, 3, 2, 1, 1] >>> 0 //no profit possible
```

(Advanced) Buy / sell advanced problems

*Problems in red with an **(Advanced)** tag are uniquely challenging and completely optional (not required to get 100). Give them a shot if you have the time!*

- Complete the method `int buySell2(int[] stock)`, where the element at index `i` in `stock` represents the price of a share of a particular stock on day `i`. The method should return the maximum potential profit; however, you can make as many transactions as you like. You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again).

```
[1, 2, 7, 4, 11] >>> 13
[2, 6, 8, 7, 8, 7, 9, 4, 1, 2, 4, 5, 8] >>> 16
```

- Complete the method `int buySell3(int[] stock)`, where the element at index `i` in `stock` represents the price of a share of a particular stock on day `i`. The method should return the maximum potential profit, given ***you can make at most two transactions***. You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again). This problem is harder than it appears.

```
[1, 4, 7, 2, 11] >>> 15
[1, 2, 4, 2, 5, 7, 2, 4, 9, 0, 9] >>> 17
```

- Complete the method `int buySell4(int k, int[] stock)`, where the element at index `i` in `stock` represents the price of a share of a particular stock on day `i`. The method should return the maximum potential profit; ***however, you can make at most*** `k` ***transactions***. You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again).

```
4, [1, 2, 4, 2, 5, 7, 2, 4, 9, 0] >>> 15
2, [1, 2, 4, 2, 5, 7, 2, 4, 9, 0, 9] >>> 17
```