**Shopping Cart**

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. At this point, it's probably a good idea to move to a next-level IDE (if you haven't already). The "How to use the Eclipse IDE" powerpoint from AP CS will get you up and running; check the "More Eclipse tricks and tips" for more info on everything Eclipse can do.
2. Create a class **Runner.java** that has a public static void main(String[] args) method.
3. 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.
4. 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. For the crafty, this can be done in one line, using a ***ternary operator***. Google it!

/\* ternary operator format: (testCondition)**?** valueForTrue **:** valueForFalse \*/

1. 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.

1. 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

1. 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

1. 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 \*/

1. Complete the method Map<Integer, Integer> mapLengths(String[] words) that returns a Map containing the number of times each length of String occurs in words. The order of the Map is irrelevant. The key will be the length of the String; the value will be the number of times it occurs.

mapLengths(new String[] {"a", "b", "hello", "hi", "yo", "I"}) >>> {1=3, 2=2, 5=1}

1. (Riddle) How can the number four be half of five?
2. Complete the method sumDigitsRecur() that implements the sumDigits() method seen previously without using loops (using recursion).
3. Complete the method keepSummingDigitsRecur() that implements the keepSummingDigits() method seen previously without using loops.
4. Complete the method int sumWithoutCarry(int a, int b) that adds two numbers without "carrying". You can assume the two numbers will be the same length.

sumWithoutCarry(861, 450) >>> 211

1. 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

1. Zeckendorf's theorem states that every positive integer can be represented as the sum of one or more *distinct* Fibonacci numbers in such a way that the sum does not include any two consecutive Fibonacci numbers. For example, the Zeckendorf representation of 100 is: 89 + 8 + 3.

Write a method void zeck(String fileName) that will print (to the console) the Zeckendorf representation for numbers the number in the file specified by fileName. You can find sample input in the **"zeck.txt"** text file, in the lab folder. Check the AP powerpoints if you can't remember how to read from a file. This file contains a number N, how many lines to read, followed by N positive integers. Your output should match the output shown below:

120 = 89 + 21 + 8 + 2

34 = 34

88 = 55 + 21 + 8 + 3 + 1

90 = 89 + 1

320 = 233 + 55 + 21 + 8 + 3

**Shopping Cart**

In this lab, you will create a graphical "shopping cart" style application. Import the **ShoppingMain.java** and **ShoppingFrame.java** files from the starter code folder into your project. ShoppingMain is a client class, for testing, with a main() method; the ShoppingFrame class provides the graphical user interface (GUI).

Begin by adding the following classes:

**Item.java** – basic description of a single item in the shopping cart

|  |  |
| --- | --- |
| Item(String name, double price) | Constructor with only name and price parameters. Should utilize the four-parameter constructor below (with a call to this()). |
| Item(String name, double price,  int bulkQty, double bulkPrice) | Also takes a bulk quantity and a bulk price as arguments, representing the discounted price if the user buys bulkQty or more items. Throws an IllegalArgumentException if any number is negative, as shown below:  throw new IllegalArgumentException("error"); |
| double priceFor(int quantity) | Returns the price for a given quantity of Item (considering bulk price, if applicable). Should throw an IllegalArgumentException if quantity is negative. |
| boolean equals(Object obj) | *<overridden>* Returns true if this Item has the same name as the supplied Item.  /\* add the @Override annotation to assert to the compiler that you're overriding a super-class method \*/ |
| String toString() | *<overridden>* Returns a String representation of this item: name followed by a comma and space, followed by $price.  If this Item has a bulk price, then you should append an extra space and a parenthesized description of the bulk pricing that has the bulk quantity, the word "for", and the bulk price. |

*If you haven't yet, check the "Eclipse tips and tricks" powerpoint for some ways Eclipse can make your life easier (the code generation slides will be especially useful as you're writing a lot of simple classes).*

**Catalog.java** - stores information about a collection (list) of Items for sale

|  |  |
| --- | --- |
| Catalog(String name) | Constructor that takes the name of this catalog as a parameter. |
| void add(Item item) | Adds an Item to the catalog (list). |
| int size() | Returns the number of items in this list. |
| Item get(int index) | Returns the Item at the supplied index. |
| String getName() | Returns the name of this catalog. |

**ItemOrder.java** - bundles together an item and the quantity ordered for that item

|  |  |
| --- | --- |
| ItemOrder(Item item, int qty) | Constructor that creates an item order for the given item and given quantity. |
| double getPrice() | Returns the cost for this item order. |
| Item getItem() | Returns a reference to the Item in this order. |
| boolean equals(Object obj) | *<overridden>* Returns true if this ItemOrder contains the same Item as the supplied ItemOrder. |

**ShoppingCart.java** - stores information about the user's orders

|  |  |
| --- | --- |
| ShoppingCart() | Constructor that creates an empty list of ItemOrders. |
| void add(ItemOrder newOrder) | Adds an ItemOrder to the list, **replacing any previous order for this item with the new order.**  Used when the user updates the quantity of an order for an Item. Should use calls to corresponding overridden equals() methods.  /\* ArrayList's *contains()* method uses a call to the *equals()* method of the type it's storing. The *equals()* method is the way Java tests objects for equivalence – if you haven't overridden a class' *equals()* method, it will use the version inherited from Object! \*/ |
| double getTotal() | Returns the total cost of the shopping cart. |

You should not introduce any other public methods to these classes, although you can add your own private methods if needed. You can override toString() in any of these classes (you might find that helpful in testing and debugging your code).

For example, create Items for bulk and non-bulk cases, update item totals, etc.

*Based on the Shopping Cart Project*

[*garfieldcs.com/wordpress/wordpress/wp-content/uploads/2013/03/Project-Shopping-Cart.pdf*](http://www.garfieldcs.com/wordpress/wordpress/wp-content/uploads/2013/03/Project-Shopping-Cart.pdf)

**(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