Lab 4 Arrays

Tracking Sales

File *Sales.java* contains a Java program that prompts for and reads in the sales for each of 5 salespeople in a company. It then prints out the id and amount of sales for each salesperson and the total sales. Study the code, then compile and run the program to see how it works. Now modify the program as follows:

1. Compute and print the average sale. (You can compute this directly from the total; no loop is necessary.)
2. Find and print the maximum sale. Print both the id of the salesperson with the max sale and the amount of the sale, e.g., "Salesperson 3 had the highest sale with $4500." Note that you don't need another loop for this; you can do it in the same loop where the values are read and the sum is computed.
3. Do the same for the minimum sale.
4. After the list, sum, average, max and min have been printed, ask the user to enter a value. Then print the id of each salesperson who exceeded that amount, and the amount of their sales. Also print the total number of salespeople whose sales exceeded the value entered.
5. The salespeople are objecting to having an id of 0—no one wants that designation. Modify your program so that the ids run from 1–5 instead of 0–4. Do not modify the array—just make the information for salesperson 1 reside in array location 0, and so on.
6. Instead of always reading in 5 sales amounts, at the beginning ask the user for the number of sales people and then create an array that is just the right size. The program can then proceed as before.

Grading Quizzes

Write a program (*Grades.java*) that grades arithmetic quizzes as follows:

1. Ask the user how many questions are in the quiz.
2. Ask the user to enter the key (that is, the correct answers). There should be one answer for each question in the quiz, and each answer should be an integer. They can be entered on a single line, e.g., 34 7 13 100 81 3 9 10 321 12 might be the key for a 10-question quiz. You will need to store the key in an array.
3. Ask the user to enter the answers for the quiz to be graded. As for the key, these can be entered on a single line. Again there needs to be one for each question. Note that these answers do not need to be stored; each answer can simply be compared to the key as it is entered.
4. When the user has entered all of the answers to be graded, print the number correct and the percent correct.

When this works, add a loop so that the user can grade any number of quizzes with a single key. After the results have been printed for each quiz, ask "Grade another quiz? (y/n)."

Reversing an Array

Write a program (*Revers.java*) that prompts the user for an integer, then asks the user to enter that many values. Store these values in an array and print the array. Then reverse the array elements so that the first element becomes the last element, the second element becomes the second to last element, and so on, with the old last element now first. Do not just reverse the order in which they are printed; actually change the way they are stored in the array. Do not create a second array; just rearrange the elements within the array you have. (Hint: Swap elements that need to change places.) When the elements have been reversed, print the array again.

Adding To and Removing From an Integer List

File *IntegerList.java* contains a Java class representing a list of integers. The following public methods are provided:

- IntegerList(int size)—creates a new list of size elements. Elements are initialized to 0.
- void randomize()—fills the list with random integers between 1 and 100, inclusive.
- void print()—prints the array elements and indices

File *IntegerListTest.java* contains a Java program that provides menu-driven testing for the IntegerList class. Compile and run I*ntegerListTest.java* to see how it works.

It is often necessary to add items to or remove items from a list. When the list is stored in an array, one way to do this is to create a new array of the appropriate size each time the number of elements changes, and copy the values over from the old array. However, this is rather inefficient. A more common strategy is to choose an initial size for the array and add elements until it is full, then double its size and continue adding elements until it is full, and so on. (It is also possible to decrease the size of the array if it falls under, say, half full, but we won't do that in this exercise.)

1. Add this capability to the IntegerList class. You will need to add an increaseSize method plus instance variables to hold the current number of integers in the list and the current size of the array. Since you do not have any way to add elements to the list, you won't need to call increaseSize yet.
2. Add a method void addElement(int newVal) to the IntegerList class that adds an element to the list. At the beginning of addElement, check to see if the array is full. If so, call increaseSize before you do anything else.

   Add an option to the menu in IntegerListTest to test your new method.
3. Add a method void removeFirst(int newVal) to the IntegerList class that removes the first occurrence of a value from the list. If the value does not appear in the list, it should do nothing (but it's not an error). Removing an item should not change the size of the array, but note that the array values do need to remain contiguous, so when you remove a value you will have to shift everything after it down to fill up its space. Also remember to decrement the variable that keeps track of the number of elements.

Add an option to the menu in IntegerListTest to test your new method.

4. Add a method removeAll(int newVal) to the IntegerList class that removes all occurrences of a value from the list. If the value does not appear in the list, it should do nothing (but it's not an error).
   Add an option to the menu in IntegerListTest to test your new method.

A Shopping Cart

In this exercise you will complete a class that implements a shopping cart as an array of items. The file *Item.java* contains the definition of a class named Item that models an item one would purchase. An item has a name, price, and quantity (the quantity purchased). The file *ShoppingCart.java* implements the shopping cart as an array of Item objects.

1. Complete the ShoppingCart class by doing the following:
   a. Declare an instance variable cart to be an array of Items and instantiate cart in the constructor to be an array holding capacity Items.
   b. Fill in the code for the increaseSize method. Your code should increase size by 3 elements.
   c. Fill in the code for the addToCart method. This method should add the item to the cart and update the totalPrice instance variable (note this variable takes into account the quantity).
   d. Compile your class.
2. Write a program that simulates shopping. The program should have a loop that continues as long as the user wants to shop. Each time through the loop read in the name, price, and quantity of the item the user wants to add to the cart. After adding an item to the cart, the cart contents should be printed. After the loop print a "Please pay ..." message with the total price of the items in the cart.

A Shopping Cart Using the ArrayList Class

In this exercise you will implement a shopping cart using the ArrayList class. The file *Item.java* contains the definition of a class named Item that models an item one would purchase (this class was used in an earlier lab). An item has a name, price, and quantity (the quantity purchased). The file Shop.java is an incomplete program that models shopping.

1. Complete *Shop.java* as follows:
   a. Declare and instantiate a variable cart to be an empty ArrayList.
   b. Fill in the statements in the loop to add an item to the cart and to print the cart contents (using the default toString in the ArrayList class). Comments in the code indicate where these statements go.
   c. Compile your program and run it.
2. You should have observed two problems with using the default printing for the cart object: the output doesn't look very good and the total price of the goods in the cart is not computed or

printed. Modify the program to correct these problems by replacing the print statement with a loop that does the following:

   a. gets each item from the cart and prints the item
   b. computes the total price of the items in the cart (you need to use the getPrice and getQuantity methods of the Item class). The total price should be printed after the loop.

3. Compile and run your program.