

Sit-in Lab #3

CS1101 AY2009/10 Semester 1

Date: 23 October 2009, Friday

School of Computing, National University of Singapore

0 Instructions

Please follow the instructions on the first page.

There are **two** exercises in this lab. Time limit is **1 hour 30 minutes**.

Do NOT:

- Use recursion
- Create any additional .java programs (they will not be uploaded onto CM)

Please do NOT reveal the question to anybody until tomorrow. Anybody found to have leaked the question in full or in part will be subject to disciplinary actions.

1 Exercise 1: Using ArrayList

[40 marks]

There are two programs in this exercise: **Inventory.java** and **Stocktake.java**. The program **Inventory.java** is a complete program, which you are not supposed to modify even a bit. You are required to complete only **Stocktake.java**, which is an application program using the Inventory class.

You may write additional method(s) in **Stocktake.java** if you wish. (But do not create additional class(es) or program(s).)

We will ignore your **Inventory.java** and use ours (which is the one given to you) to test your **Stocktake.java** program. Hence, please treat **Inventory.java** as a write-protected file.

The complete **Inventory.java** program and the **Stocktake.java** skeleton program are loaded into the working directory, and are shown below (some lines in the print-out below are merged into a single line to save space) for your convenience.

```

/**
 * CS1101 AY2009/10 Sit-in Lab #3 Set B Ex 1
 * Inventory.java
 * This defines the Inventory class that contains name and quantity.
 * Note: You are NOT to change this program a bit.
 */
class Inventory {

    // Data members
    String name;
    int    quantity;

    // Default constructor
    public Inventory() {
        this("unknown", 0);
    }

    // Alternative constructor
    public Inventory(String newName, int newQuantity) {
        name = newName;
        quantity = newQuantity;
    }

    // Accessors
    public String getName() { return name; }

    public int getQuantity() { return quantity; }

    // Mutators
    public void setName(String newName) { name = newName; }

    public void setQuantity(int newQuantity) { quantity = newQuantity; }

    // Returns string representation of Inventory object
    public String toString() {
        return name + " (" + quantity + ")";
    }

    // Returns true if this object and obj object
    // contain the exact content
    public boolean equals(Object obj) {
        if (obj instanceof Inventory) {
            Inventory inv = (Inventory) obj;
            return name.equals(inv.name) &&
                quantity == inv.quantity;
        }
        else
            return false;
    }
}

```

```

/**
 * CS1101 AY2009/10 Sit-in Lab #3 Set B Ex 1
 * Stocktake.java
 * This program asks user to enter a list of items to be
 * inserted, deleted or changed in the inventory.
 * It uses the Inventory class.
 *
 * <Fill in your name and discussion group here>
 */

// import appropriate package(s)

class Stocktake {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        List<Inventory> list = new ArrayList<Inventory>();
        Inventory inv;

        System.out.println("Enter instruction and relevant data,");
        System.out.println("end by pressing enter for instruction:");

        // Fill in your code
        // You must use the ArrayList list in your code

        displayList(list);
    }

    // Display all elements in the list.
    // This method is provided.
    public static void displayList(List<Inventory> aList) {
        Inventory inv;
        Iterator<Inventory> itr = aList.iterator();

        while (itr.hasNext()) {
            inv = itr.next();
            System.out.println(inv);
        }
    }
}

```

Task

You are to write **Stocktake.java** to read a list of instructions from the user. There are 3 types of instructions, and for each type, different data are required:

- **Insert** instruction: User needs to provide the name of the item which is a string (eg: "Pencil sharpener") and a quantity which is a positive integer (eg: 12). The instruction "insert", item name and item quantity are to be read on 3 lines, for example:

```
insert
Pencil sharpener
12
```

The item is to be inserted to the end of the ArrayList **list**.

- **Delete** instruction: User needs to provide the position of the item to be deleted. If the position is invalid (i.e. no item in that position in the **list**), this instruction is to be ignored. Your program must not crash. The instruction "delete" and the position are to be read on 2 lines, for example:

```
delete
5
```

The item at position 5 (if there is such an item) is to be removed from the ArrayList **list**.

- **Change** instruction: User needs to provide the position of the item whose quantity is to be changed. If the position is invalid (i.e. no item in that position in the **list**), this instruction is to be ignored. Your program must not crash. The instruction "change", the position and the new quantity (a positive integer) are to be read on 3 lines, for example:

```
change
3
10
```

The quantity of the item at position 3 (if there is such an item) of the ArrayList **list** is to be changed to 10.

The user enters an empty instruction to terminate the input. The program then displays the elements of the ArrayList **list** by calling the given **displayList()** method.

Sample Run

A sample run of the program is given below. Essential outputs are shown in bold.

```
Enter instruction and relevant data,  
end pressing enter for instruction:  
insert  
School bag  
10  
insert  
English Textbook 1A  
5  
insert  
Ruler  
20  
delete  
5  
delete  
-1  
delete  
1  
insert  
Eraser  
30  
change  
-2  
8  
change  
6  
8  
change  
1  
8  
    ← user pressed enter here to end list of instructions  
School bag (10)  
Ruler (8)  
Eraser (30)
```

Marking Scheme

This exercise constitutes 40 marks. 30 marks are allocated to the correctness, and 10 marks to design and style.

Advice: Do not spend more than 35 minutes on Exercise 1, so that you have enough time for Exercise 2.

2 Exercise 2: Using 2-dimensional Array

[60 marks]

You are to write a single program **FillTable.java** to perform the following:

- Read the number of rows and columns (both positive integers), and create a two-dimensional integer array with that number of rows and columns.
- Read a positive integer value k .
- Fill the array with multiples of k (i.e. $0, k, 2k, 3k$, etc.) from topmost row (row 0) to bottom, and for each row, from leftmost column (column 0) to right. Display the array.
- Read two values $corner1$ and $corner2$, and sum up all the values in the rectangular portion of the array cornered by $corner1$ and $corner2$. You may assume that:
 - $corner1$ and $corner2$ are distinct
 - $corner1 < corner2$
 - $corner1$ and $corner2$ are values present in the array, $corner1$ being the top-left of the rectangular portion and $corner2$ being the bottom-right

You must use array and not ArrayList for this exercise.

The skeleton **FillTable.java** program is loaded into the working directory. A **printArray()** method is provided in the skeleton program. Do NOT modify **printArray()**. You may write your own additional method(s) if you wish. (But do not create additional class(es) or program(s).)

Sample Runs

Three sample runs of the program are given below (with explanatory note). Essential outputs are shown in bold.

```
Enter number of rows and columns: 4 6
Enter k: 5
0      5      10     15     20     25
30     35     40     45     50     55
60     65     70     75     80     85
90     95     100    105    110    115
Enter values at two corners: 35 80
Sum = 460
```

Rectangular portion cornered by 35 and 80.

```
Enter number of rows and columns: 3 4
Enter k: 12
0      12     24     36
48     60     72     84
96     108    120    132
Enter values at two corners: 48 72
Sum = 180
```

Rectangular portion cornered by 48 and 72.

```
Enter number of rows and columns: 4 3
Enter k: 3
0      3      6
9      12     15
18     21     24
27     30     33
Enter values at two corners: 12 30
Sum = 63
```

Marking Scheme

This exercise constitutes 60 marks. 40 marks are allocated to the correctness of the methods that you are supposed to complete or write. 20 marks are allocated to design and style.

=== End of paper ===