Homework Turnin

Name:	Akshit K Patel
Email:	akshit@uw.edu
Student ID:	1561387
Section:	DC
Course:	CSE 143 16au
Assignment:	a1
Receipt ID:	7b9271161c8a4e4ad748659dc9c60c96

Replacing prior submission from Wed 2016/10/05 11:13pm.

Turnin Successful!

The following file(s) were received:

LetterInventory.java (7997 bytes)

```
/**
 * @author Akshit Patel
 * @Date 09/29/2016
 * CSE 143D DC
 * TA: Melissa Medsker
 * HW #1 LetterInventory
 */

/**
 * LetterInventory Class creates, updates & modifies an inventory of the no. of
 * alphabets in a string passed and provides useful methods to get its size,
 * adding or subtracting or returning a useful string of the inventory.
 */
public class LetterInventory {

    public static final int INVENTORY_SIZE = 26;// fixed size of inventory.
    private int[] inventory;// Reference to the letter inventory.
    private int size;// keeps check of the total alphabets in inventory.

    /**
     * This Constructor creates an empty inventory for the client of the fixed
     * size.
     */
    public LetterInventory() {
        this("");// create empty inventory.
    }

    /**
     * This constructor creates an inventory for the client based of the string
     * passed by client.
     *
     * preCondition: parameter passed is a string.
     *
     * @param data is a string to be passed to be processed in inventory.
     */
    public LetterInventory(String data) {
        this.inventory = new int[INVENTORY_SIZE];// create empty inventory.
        data = data.toLowerCase();// converts the string into lower case.
        // for-loop to process the string alphabets into inventory.
        for (int i = 0; i < data.length(); i++) {
            char currentChar = data.charAt(i);
            // if statement to check for only alphabets.
            if (currentChar >= 'a'
                    && currentChar <= ('a' + INVENTORY_SIZE - 1)) {
                this.inventory[currentChar - 'a']++;// adding to inventory
                this.size++;// updating the new size.
            }
        }
    }
```

```
    /**
     * This method helps the client to get data from inventory of the specific
     * letter.
     *
     * preCondition: the letter is an alphabet.
     *
     * @param letter Character whose info is needed.
     * @throws IllegalArgumentException when letter is not an alphabet.
     * @return int value of the letter in inventory.
     */
    public int get(char letter) {
        letter = Character.toLowerCase(letter);
        this.errorCheck(letter);
        return this.inventory[letter - 'a'];
    }

    /**
     * This method helps the client to set a specific value of the letter in the
     * inventory.
     *
     * preCondition: the letter is an alphabet & value is non-negative.
     *
     * @param letter Character which is needed to update its associated value.
     * @param value int data of the letter to be updated.
     * @throws IllegalArgumentException when letter is not an alphabet & value
     * is a negative number.
     *
     */
    public void set(char letter, int value) {
        letter = Character.toLowerCase(letter);
        this.errorCheck(letter);
        if (value < 0) {
            throw new IllegalArgumentException("No negative values accepted!");
        }
        int temp = this.inventory[letter - 'a'];
        this.size -= temp;// updates size by removing the value before.
        this.inventory[letter - 'a'] = value;// updates the inventory.
        this.size += value;// updates size by adding the provided value .
    }

    /**
     * This method provides the client with the total size of the inventory
     * which is the sum of the letters in inventory.
     *
     * @return the size of inventory.
     */
    public int size() {
        return this.size;
    }

    /**
     * This methods helps the client to know if the inventory is empty.
     *
     * @return true if this inventory is empty.
     */
    public boolean isEmpty() {
        return size == 0;
    }

    /**
     * This methods helps the client to get a string representation of the
     * inventory.
     *
     * @return String representation of the inventory of all letters in
     * lowercase, sorted and inside square brackets.
     */
    @Override
    public String toString() {
        String sortString = "";// empty string to store the data.
        // for loop to add data from the inventory to the string
        for (int i = 0; i < INVENTORY_SIZE; i++) {
            for (int j = 0; j < inventory[i]; j++) {
                char alphabet = (char) (i + 'a');// getting the alphabet.
                sortString += alphabet;// adding the alphabet to the string.
            }
        }
        return "[" + sortString + "]";// return the updated string.
    }

    /**
     * This method constructs and returns a new LetterInventory object that
     * represents the SUM of this(current) LetterInventory and the other given
     * LetterInventory.
     *
     * preCondition: this & other are LetterInventory object. postCondition:
     * this & other remain the same.
     *
```

```
     * @param other LetterInventory object that needs to be added to this
     * LetterInventory.
     * @return LetterInventory object that represents the sum of this and other
     * LetterInventory.
     */
    public LetterInventory add(LetterInventory other) {
        LetterInventory sum = new LetterInventory();// creates empty Inventory.
        // for loop to add the two inventory data in new one.
        for (int i = 0; i < INVENTORY_SIZE; i++) {
            // adding the data.
            sum.inventory[i] = this.inventory[i] + other.inventory[i];
            sum.size += sum.inventory[i];// updating size of the new Inventory.
        }
        return sum;// processed inventory is returned.
    }

    /**
     * This method constructs and returns a new LetterInventory object that
     * represents the DIFFERENCE of this(current) LetterInventory and the other
     * given LetterInventory.
     *
     * preCondition: this & other are LetterInventory object. postCondition:
     * this & other remain the same.
     *
     * @param other LetterInventory object that needs to be subtracted from this
     * LetterInventory.
     * @return LetterInventory object that represents the difference of this and
     * other LetterInventory. Returns null if the difference is negative.
     */
    public LetterInventory subtract(LetterInventory other) {
        LetterInventory remove = new LetterInventory();// empty Inventory made.
        // for loop to get the difference of two inventory data in newer one.
        for (int i = 0; i < INVENTORY_SIZE; i++) {
            // subtracting the data.
            remove.inventory[i] = this.inventory[i] - other.inventory[i];
            // if statement to check for negative values.
            if (remove.inventory[i] < 0) {
                return null;
            }
            remove.size += remove.inventory[i];// updates the size.
        }
        return remove;// processed inventory is returned.
    }

    /**
     * This method helps the client to know the percentage of given letter in
     * the inventory.
     *
     * preCondition: The letter is an alphabet character only.
     *
     * @param letter Character whose info is needed.
     * @throws IllegalArgumentException if letter is not an alphabet.
     * @return a double from 0.0 to 1.0.
     */
    public double getLetterPercentage(char letter) {
        letter = Character.toLowerCase(letter);
        this.errorCheck(letter);
        // if statement to check if the inventory has any letters.
        if (this.isEmpty()) {
            return 0;
        }
        // calculate and return the percent.
        return (double) (this.inventory[letter - 'a']) / size;
    }

    /**
     * This method checks for error in the letter passed by client.
     *
     * @param letter the Character passed by the client.
     * @throws IllegalArgumentException if letter does not meet the
     * preconditions.
     */
    private void errorCheck(char letter) {
        // if statement that checks for letters not in between 'a' to 'z'.
        if (letter < 'a' || letter > ('a' + INVENTORY_SIZE - 1)) {
            throw new IllegalArgumentException("Non-Alphabets not accepted!");
        }
    }
}
```

decodedCryptogram.txt (4607 bytes)

```
now, the star-belly sneetches had bellies with stars. the plain-belly sneetches had none upon thars. those stars wer
```