CS 1073

FR03A

Assignment #7

Ethan A. McCarthy

3573807

# Section 1:

import java.util.Scanner;

import java.util.Random;

/\*\*

 \* @author Ethan McCarthy

 \*/

 public class CodeEncryption {

    public static void main(String[] args){

        Scanner scan = new Scanner(System.in);

        boolean end = false;

        int rows = Integer.parseInt(scan.nextLine());

        while(!end){

            String message = scan.nextLine();

            message = message.replaceAll("[^a-zA-Z]+", "");

            message = message.toLowerCase();

            int columns = (message.length()/rows) + 1;

            if (message.length() % rows == 0){

                columns = (message.length()/rows);

            }

            Random random = new Random();

            while (message.length() != columns\*rows){

                String alphabet = "abcdefghijklmnopqrstuvwxyz";

                message = message + alphabet.charAt(random.nextInt(26));

            }

            char[][] encryptionArray = new char[columns][rows];

            int columnsDecrease = columns - 1;

            for(int i = 0; i < message.length();columnsDecrease--){

                if(columnsDecrease%2 == 0){

                      for(int x = rows - 1; x >= 0 && i < (message.length()); x--, i++){

                        encryptionArray[columnsDecrease][x] = message.charAt(i);

                    }

                }

                else{

                    for(int x = 0; (x < rows && i < (message.length())); x++, i++){

                        encryptionArray[columnsDecrease][x] = message.charAt(i);

                    }

                }

            }

            for(int i = rows - 1; i >= 0; i--){

                for(int x = 0; x < columns && i >= 0; x++){

                    System.out.print(encryptionArray[x][i]);

                }

            }

            rows = Integer.parseInt(scan.nextLine());

            System.out.println();

            System.out.println();

            if(rows == 0){

                end = true;

            }

        }

    }

 }

# Section 2:

Text

Description automatically generated

# Section 3:

/\*\*

 \* @author Ethan McCarthy 3573807

 \*/

 public class CardHolder {

    private String name;

    private int aptNum;

    private String phoneNum;

    private static int membersCount =  100000;

    private KitchenItem[] signedOutItems = new KitchenItem[8];

    /\*\*

     \* constructor

     \* @param nameIn name input

     \* @param aptNumIn apartment number input

     \* @param phoneNumIn phone number in

     \*/

    public CardHolder(String nameIn, int aptNumIn, String phoneNumIn){

        name = nameIn;

        aptNum = aptNumIn;

        phoneNum = phoneNumIn;

    }

    /\*\*

     \* method to sign out the kitchen item

     \* @param KitchenItemIn the kitchen item input

     \* @return if the sign out was successful or not

     \*/

    public boolean signOut(KitchenItem KitchenItemIn){

        boolean successful = false;

        for(int i = 0; i < 8; i++){

            if(signedOutItems[i] == null){

                signedOutItems[i] = KitchenItemIn;

                successful = true;

                break;

            }

        }

        return successful;

    }

    /\*\*

     \* method to return a signed out item

     \* @param kitchenItemIn the kitchen item that is being returned

     \* @return wether the return was successful or not

     \*/

    public boolean returnItem(KitchenItem kitchenItemIn){

        boolean successful = false;

        for(int i = 0; i < 8; i++){

            if(signedOutItems[i] != null){

                if(kitchenItemIn.getDescription() == signedOutItems[i].getDescription() && kitchenItemIn.getPrice() == signedOutItems[i].getPrice() && kitchenItemIn.isRestricted() == signedOutItems[i].isRestricted()){

                    signedOutItems[i] = null;

                    successful = true;

                }

            }

        }

        return successful;

    }

    /\*\*

     \* method to get the name of the card holder

     \* @return the name

     \*/

    public String getName(){

        return name;

    }

    /\*\*

     \* method to get the apartment number of the card holder

     \* @return apartment number of the card holder

     \*/

    public int getAptNumber(){

        return aptNum;

    }

    /\*\*

     \* method to get the phone number of the card holder

     \* @return the phone number of the card holder

     \*/

    public String getPhoneNumber(){

        return phoneNum;

    }

    /\*\*

     \* method to get the membershit number of the card holder

     \* also increases the membership number for the next member

     \* @return membership number

     \*/

    public int getMembershipNumber(){

        return membersCount++;

    }

    /\*\*

     \* mutator to change phone number

     \* @param phoneNumIn new phone number

     \*/

    public void setPhoneNumber(String phoneNumIn){

        phoneNum = phoneNumIn;

    }

    /\*\*

     \* method to retrieve a copy of the signed out items in possesion of the card holder

     \* @return array of the items that the card holder has signed out

     \*/

    public KitchenItem[] getSignedOutItems(){

        int itemsOut = 0;

        for(int i = 0; i < 8; i++){

            if(signedOutItems[i] != null){

                itemsOut++;

            }

        }

        KitchenItem[] signedOutItemsCopy = new KitchenItem[itemsOut];

        int copyLength = 0;

        for(int i = 0; i < 8; i++){

            if(signedOutItems[i] != null){

                signedOutItemsCopy[copyLength] = signedOutItems[i];

                copyLength++;

            }

        }

        return signedOutItemsCopy;

    }

 }

/\*\*

 \* @author Ethan McCarthy 3573807

 \*/

 public class JuniorCardHolder extends CardHolder{

    private String guardian;

    /\*\*

     \* constructor for the junior member

     \* @param nameIn name input

     \* @param aptNumIn apartment number input

     \* @param phoneNumIn phone number input

     \* @param guardianIn guardian input

     \*/

    public JuniorCardHolder(String nameIn, int aptNumIn, String phoneNumIn, String guardianIn){

        super(nameIn, aptNumIn, phoneNumIn);

        guardian = guardianIn;

    }

    /\*\*

     \* method to get the guardian of the junior member

     \* @return name of guardian

     \*/

    public String getGuardian(){

        return guardian;

    }

    /\*\*

     \* method to sign out items for the junior member

     \* @param the item being signed out

     \* @return wehter it was successful or not

     \*/

    public boolean signOut(KitchenItem kitchenItemIn){

        boolean successful;

        if (super.signOut(kitchenItemIn)){

            successful = true;

        }

        else{

            successful = false;

        }

        if(kitchenItemIn.isRestricted()){

            successful = false;

        }

        return successful;

    }

    /\*\*

     \* method to return an item for a junior member

     \* @param the item being returned

     \* @return whether it was successful or not

     \*/

    public boolean returnItem(KitchenItem kitchenItemIn){

        boolean successful;

        if (super.returnItem(kitchenItemIn)){

            successful = true;

        }

        else{

            successful = false;

        }

        return successful;

    }

 }

/\*\*

 \* @author Ethan McCarthy 357807

 \*/

 public class KitchenItem {

    private String itemDescription;

    private double price;

    private boolean restricted;

    /\*\*

     \* constructor method

     \* @param itemDescriptionIn description of item

     \* @param priceIn price of item

     \* @param restrictedIn whether it is restricted or not

     \*/

    KitchenItem(String itemDescriptionIn, double priceIn, boolean restrictedIn){

        itemDescription = itemDescriptionIn;

        price = priceIn;

        restricted = restrictedIn;

    }

    /\*\*

     \* method to get the description of the item

     \* @return the description

     \*/

    public String getDescription(){

        return itemDescription;

    }

    /\*\*

     \* method to get the price of the item

     \* @return the price

     \*/

    public double getPrice(){

        return price;

    }

    /\*\*

     \* method to see if the item is restricted or not

     \* @return true or false

     \*/

    public boolean isRestricted(){

        return restricted;

    }

 }

# Section 4:

\*\*\* Test case #1: Create a CardHolder object & test accessors

Name: Sally Smith

Appt #: 152

Phone: 454-1234

Member #: 100000

Correct result: Sally has zero kitchen items.

\*\*\* Test case #2: Create a JuniorCardHolder object & test accessors

Name: Tommy Smith

Appt #: 152

Phone: 454-1234

Member #: 100001

Guardian: Sally Smith

Correct result: Tommy has zero kitchen items.

\*\*\* Test case #3: Automatically generate a member number

Correct result: 100002 is the correct member number.

\*\*\* Test case #4: Create KitchenItem objects & test accessors

Description: Whisk

Orig. Price: $10.00

Restricted: false

\*\*\* Test case #5: Change phone number for both CardHolder types

Correct result: Sally's phone number successfully changed.

Correct result: Tommy's phone number successfully changed.

\*\*\* Test case #6: Sign out one KitchenItem

Correct result: Sally signed out an item successfully.

Correct result: Sally has one kitchen item.

\*\*\* Test case #7: Sign out multiple KitchenItems

Correct result: Sally signed out two more items successfully.

Correct result: Sally has three kitchen items.

\*\*\* Test case #8: Intentionally exceed the sign out limit

Correct result: Sally was prevented from signing out more than 8 kitchen items.

\*\*\* Test case #9: A junior card holder tries to sign out a restricted item

Correct result: Tommy was able to sign out an unrestricted item.

Correct result: Tommy was prevented from signing out a restricted item.

\*\*\* Test case #10: Returning the only item that was signed out

Correct result: Tommy's item was successfully returned.

Correct result: Tommy's list length changed appropriately.

\*\*\* Test case #11: Returning an item that was not signed out

Correct result: Unsuccessful attempt to return an item that was not signed out.

\*\*\* Test case #12: Returning the first item that was signed out

Correct result: Sally's first item was successfully returned.

Correct result: Sally's list length changed appropriately.

Confirm return: Whisk should be absent from the following list:

Paring Knife

Metric Measuring Spoon Set

Creme Brule Torch

Kitchen Scale

Electric Knife

16 Quart Stock Pot

Deep Fryer

\*\*\* Test case #13: Returning a mid-list item

Correct result: Creme Brule Torch was successfully returned.

Correct result: Sally's list length changed appropriately.

Confirm return: Creme Brule Torch should be absent from the following list:

Paring Knife

Metric Measuring Spoon Set

Kitchen Scale

Electric Knife

16 Quart Stock Pot

Deep Fryer

\*\*\*\*\*\*\*\*\*\*\*\*\* End of Test Cases \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*