**CS1073  
FR03B**

**Assignment #7**

**Daniyal Khan  
3765942**

**Question 1:**  
  
import java.util.Scanner;

/\*\*

\* This class represents a decryption algorithm

@author Daniyal Khan 3765942

\*/

public class Decrypt {

public static void main(String[] args) {

Scanner read = new Scanner(System.in);

String msg; // user input

int columns; //

int iterateStr = 0;

boolean alternate = true;

columns = read.nextInt();

read.nextLine(); // consumes newline character

msg = read.next();

int rows = msg.length() / columns;

String[][] patternArray = new String[rows][columns]; // 2D array for entering numbers for making up the encryption pattern

while(columns != 0) {

for (int j = 0; j < patternArray[0].length; j++) {

if (alternate){ // reading the encrypted msg and then storing the char into the 2D array alternating from bottom to top and top to bottom

for (int i = patternArray.length - 1; i >= 0; i--) {

patternArray[i][j] = "" + msg.charAt(iterateStr++);

}

alternate = !alternate;

} else {

for (int i = 0; i < patternArray.length; i++) {

patternArray[i][j] = "" + msg.charAt(iterateStr++);

}

alternate = !alternate;

}

}

iterateStr = 0;

System.out.println(decrypt(patternArray));

// NEXT INPUT

columns = read.nextInt();

// read.nextLine(); // consumes newline character

if (columns != 0) {

msg = read.next();

rows = msg.length() / columns;

patternArray = new String[rows][columns]; // new 2D array for the next encrypted msg

alternate = true;

}

}

}

public static String decrypt(String[][] array) {

boolean alternate = true;

String decryptedMsg = "";

for (int i = 0; i < array.length; i++) { // alternating from left to right and right to left in the 2D array and then storing into String

if(alternate) {

for (int j = 0; j < array[0].length; j++) {

decryptedMsg += array[i][j];

}

alternate = !alternate;

} else {

for (int j = array[0].length - 1; j >= 0 ; j--) {

decryptedMsg += array[i][j];

}

alternate = !alternate;

}

}

return decryptedMsg;

}

public static void printPartialArray(int companionVar, String[] array) {

for(int i = 0; i < companionVar; i++) {

System.out.println(array[i]);

}

}

public static void print2DArray(String[][] array) {

for(int i = 0; i < array.length; i++) {

for(int j = 0; j < array[0].length; j++) {

System.out.print(array[i][j] + " ");

}

System.out.println();

}

}

}

**Output:**

A screenshot of a computer

Description automatically generated

**Question 2:**

**SpellCaster:**

/\*\*

This class represents a Spell Caster

@author Daniyal Khan 3765942

\*/

public class SpellCaster{

/\*\*

\* Name of the spell caster

\*/

private final String name;

/\*\*

\* Level of the spell caster

\*/

private final int level;

/\*\*

\* Guild membership number of the spell caster

\*/

private final int guildMembershipNumber;

/\*\*

\* ID varaible for incrementing the guild memberhsip number everytime a new Spell Caster is added

\*/

private static int ID = 6000;

/\*\*

\* Spell book of the spell caster

\*/

private Spell[] spellBook;

/\*\*

\* Max number of spells a spell caster can hold in the spell bookjavadoc -author -private -d SpellCaster.java

\*/

private final int NUM\_SPELLS = 7;

/\*\*

\* Constructs an object of type SpellCaster

\* @param name Name of the Spell Caster

\* @param level Level of the Spell Caster

\*/

public SpellCaster(String name, int level) {

this.name = name;

this.level = level;

guildMembershipNumber = ID; // assigning guild number and changing the ID static variable everytime

ID++;

spellBook = new Spell[0]; // starting size of Spell Book is zero

}

/\*\*

\* Returns name of the spell caster

\* @return name of the spell caster

\*/

public String getName() {

return name;

}

/\*\*

\* Returns level of the spell caster

\* @return level of the spell caster

\*/

public int getLevel() {

return level;

}

/\*\*

\* Returns the guild membership id of the spell caster

\* @return guild membership id of the spell caster

\*/

public int getMembershipNum() {

return guildMembershipNumber;

}

/\*\*

\* Returns the entire spell book of the spell caster

\* @return spell book of the spell caster

\*/

public Spell[] getSpellBook() {

return spellBook;

}

/\*\*

\* Adds a spell to the spell book of the spell caster

\* @param spell spell to be added to the spell book

\* @return true if spell gets added succesfully, false otherwise

\*/

public boolean addSpell(Spell spell) {

boolean added = false;

if (spell.getLevel() <= level && spellBook.length < NUM\_SPELLS) {

Spell[] newSpellBook = new Spell[spellBook.length+1]; // whenever adding a new spell, create a new array +1 the size of the previous one

System.arraycopy(spellBook, 0, newSpellBook, 0, spellBook.length); // copy all the elements of the previous to new

newSpellBook[newSpellBook.length-1] = spell;

spellBook = newSpellBook;

added = true;

}

return added;

}

/\*\*

\* Casts/removes the spell from the spell book of the spell caster

\* @param spell spell to be casted from the spell book

\* @return true if the spell gets casted successfully, false otherwise

\*/

public boolean castSpell(Spell spell) {

boolean cast = false;

for (int i = 0; i < spellBook.length && cast != true; i++) {

if (spellBook[i].equals(spell)) {

spellBook[i] = spellBook[spellBook.length-1];

Spell[] newSpellBook = new Spell[spellBook.length-1]; // when spell gets casted, create a new array -1 the size of the previous one

System.arraycopy(spellBook, 0, newSpellBook, 0, spellBook.length-1); // copy one less element of the previous array

spellBook = newSpellBook;

cast = true;

}

}

return cast;

}

}

**SpellCasterApprentice:**

/\*\*

This class represents a Spell Caster Apprentice

@author Daniyal Khan 3765942

\*/

public class SpellCasterApprentice extends SpellCaster {

/\*\*

\* Supervisor of the apprentice

\*/

private SpellCaster supervisor;

/\*\*

\* Contructs an object of type SpellCasterApprentice

\* @param name Name of the apprentice

\* @param level Level of the apprentice

\* @param supervisor Supervisor of the apprentice

\*/

public SpellCasterApprentice(String name, int level, SpellCaster supervisor) {

super(name, level);

this.supervisor = supervisor;

}

/\*\*

\* Returns the supervisor of the apprentice

\* @return Supervisor of the apprentice

\*/

public SpellCaster getSupervisor() {

return supervisor;

}

/\*\*

\* @param spell Spell to be added

\* @return true if added successfully, otherwise false

\*/

public boolean addSpell(Spell spell) {

boolean added = false;

if(spell.hasComponents()) {

added = false;

} else {

added = super.addSpell(spell);

}

return added;

}

}

**Spell:**

/\*\*

\* This class represents a Spell

@author Daniyal Khan 3765942

\*/

public class Spell{

/\*\*

\* Name of the spell

\*/

private final String name;

/\*\*

\* Level of the spell

\*/

private final int level;

/\*\*

\* Material Comp, if the spell has it

\*/

private final boolean materialComps;

/\*\*

\* Constructs a object of type Spell

\* @param name name of the spell

\* @param level level of the spell

\* @param materialComps if it has a material component

\*/

public Spell(String name, int level, boolean materialComps) {

this.name = name;

this.level = level;

this.materialComps = materialComps;

}

/\*\*

\* Returns the name of the spell

\* @return Name of the spell

\*/

public String getName() {

return name;

}

/\*\*

\* Returns the level of the spell

\* @return Level of the spell

\*/

public int getLevel() {

return level;

}

/\*\*

\* Returns if the spell has a material component

\* @return Material component true or false

\*/

public boolean hasComponents() {

return materialComps;

}

/\*\*

\* Returns if two spells are equal

\* @param spellBook Spell to compare with

\* @return Spell equal if true or false

\*/

public boolean equals(Spell spellBook) {

return spellBook.name == name

&& spellBook.level == level

&& spellBook.materialComps == materialComps;

}

}

**Output:**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**