**DOKUZ EYLUL UNIVERSITY**

**ENGINEERING FACULTY**

**DEPARTMENT OF COMPUTER ENGINEERING**

**CME1251 PROJECT BASED LEARNING – II**

**FINAL REPORT**

**PROJECT – III**

**COLUMNS**

**by**

**Gürkan Bıyık**

**Duru Çapar**

**Deniz Katayıfçı**

**Şahin Öztürk**

**Lecturers**

**Prof.Dr. Derya Birant**

**Res.Asst. Göksu Tüysüzoğlu**

**IZMIR**

**27.05.2022**

# CHAPTER ONE

PROGRESS DESCRIPTION

Columns have been turned into Multi Linked List data type. Transfer operations have been done. Checked if the cards are setted.

# CHAPTER TWO

TASK SUMMARY

## Completed Tasks

All tasks completed.

## Incomplete Tasks: Reasons and Explanations

All tasks completed.

## Additional Improvements

The number of columns can be changed according to the player's request, and as the number of columns increases, the number of cards also increases. A menu for the game has been created.

# CHAPTER THREE

EXPLANATION of algorıthms

## Screenshots

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu



metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

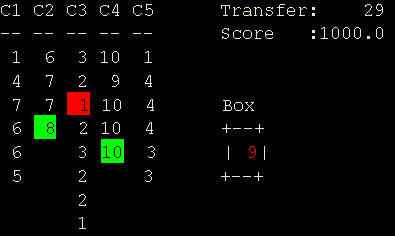
## Functions

Game.java

* public void keyTyped(KeyEvent e)
* public void keyPressed(KeyEvent e)
* public void keyReleased(KeyEvent e)
* public Game()
* public void start()
* public void menuKeys()
* public void printMenu()
* public void printHowToPlay()
* public void printPreparedBy()
* public void optionsMenu()
* public void play()
* public void resetGameVariables()
* public void print(enigma.console.Console console, String str, int x, int y, TextAttributes attr)
* public void clear(enigma.console.Console console)
* public void gameKeys()
* public void printGameScreen(int column\_width)
* public void clearValues(int column\_width)
* public void createBox(SinglyLinkedList box)
* public void createLines(MultiLinkedList columns, int column\_amount)
* public void dealTheCards(SinglyLinkedList box, int number)
* public boolean canTransferToColumn(MultiLinkedList columns, int line\_index, int card)
* public void printHighScoreTable(double playerScore)
* public void checkColumns()
* public DoublyLinkedList highscoretableData(double playerScore)
* public TextAttributes randomTextAttributesBacColor(Color BackgroundColor)
* public TextAttributes randomTextAttributesForColor(Color ForegroundColor)
* public Color rgbToColor(int r, int g, int b)
* public Color hexToColor(String hex)

## Algorithms and Solution Strategies

Gürkan Bıyık: A main menu created with options. In the gameplay, the selected number’s color is changed to red. Highlighted the back of the number that cursor on with red. If the player selects a number the last number of possible columns that can be transferred will be highlighted as green. When the cursor comes to edges, it shifts to the beginning or end of the columns depending on the way.



Duru Çapar: As the numbers are transferred, the checkColumns function in Game class has been written to continuously check each column to check whether there is a set.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Inside this function, isThereAnySet and linePop functions called from Multi Linked list class are used. isThereAnySet, checks if there is a set in this column. If there is a set, Line Pop function is called and the column's elements are deleted.

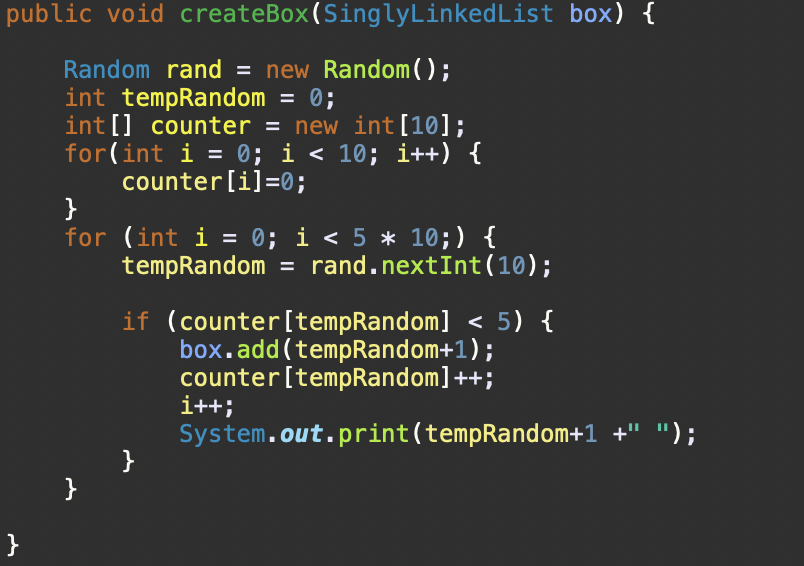
metin içeren bir resim

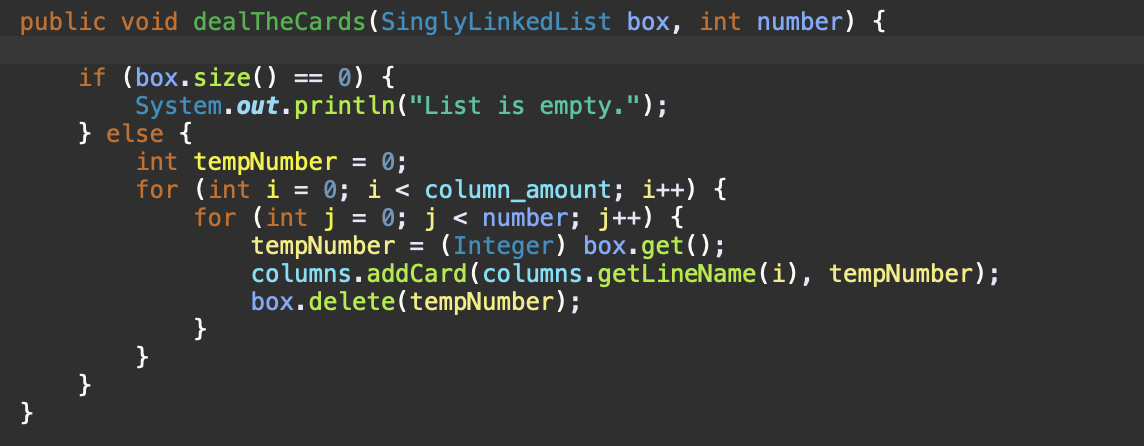
Açıklama otomatik olarak oluşturuldu

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Deniz Katayıfçı: I used random class to create box and deal the cards. I limited each column with 6 at first dealing process with constants. I used SLL functions to create returnboxnumber and deleteboxnumber functions. I returned the first element in sll and if player decided to use the number, I delete the first number.







Şahin Öztürk: I used the MultiLinkedList class and its functions to create and print all the columns. With the MultiLinkedList class, it was much easier to print the card deck containing that number of columns according to the number of columns in the options. Different functions have also been added to the MLL class.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

# CHAPTER FOUR

PROBLEMS ENCOUNTERED

No problems were encountered.

# CHAPTER FIVE

conclusıon

* Columns project was an educational journey for our team.
* Our team finished all tasks correctly, and fully functional.
* The team worked with good communication, and we learned to work as a team.

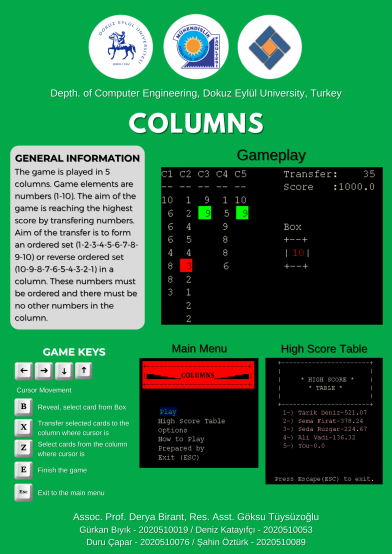
REFERENCES

* 1) [https://howtodoinjava.com](https://howtodoinjava.com/)

2) [https://java-programming.mooc.fi](https://java-programming.mooc.fi/)

3) https://stackoverflow.com

**AppendIx A**



**AppendIx B**

Code of the Project

**MultiLinkedList.java**

**public class MultiLinkedList {**

**private LineNode head;**

**public MultiLinkedList() {**

**this.head = null;**

**}**

**public void addLine(String lineName) {**

**LineNode newNode = new LineNode(lineName);**

**if (head == null) {**

**head = newNode;**

**} else {**

**LineNode temp = head;**

**while (temp.getDown() != null) {**

**temp = temp.getDown();**

**}**

**temp.setDown(newNode);**

**}**

**}**

**public void addCard(String lineName, int cardNumber) {**

**if (head == null) {**

**System.out.println("There is no line");**

**} else {**

**ColumnNode newCNode = new ColumnNode(cardNumber);**

**LineNode temp = head;**

**while (temp != null && !temp.getData().toString().equals(lineName)) {**

**temp = temp.getDown();**

**}**

**if (temp != null) {**

**if (temp.getRight() == null)**

**temp.setRight(newCNode);**

**else {**

**ColumnNode tempC = temp.getRight();**

**while (tempC.getNext() != null) {**

**tempC = tempC.getNext();**

**}**

**tempC.setNext(newCNode);**

**}**

**} else {**

**System.out.println("There is no line like " + lineName);**

**}**

**}**

**}**

**public void deleteLine(String lineName) {**

**if (head == null) {**

**System.out.println("There is no line");**

**} else {**

**if (head.getData().toString().equals(lineName))**

**head = head.getDown();**

**else {**

**LineNode temp = head;**

**LineNode prev = null;**

**while (temp != null && !temp.getData().toString().equals(lineName)) {**

**prev = temp;**

**temp = temp.getDown();**

**}**

**if (temp != null) {**

**prev.setDown(temp.getDown());**

**} else {**

**System.out.println("There is no line like" + lineName);**

**}**

**}**

**}**

**}**

**public void deleteCard(String lineName, int cardNumber) {**

**if (head == null) {**

**System.out.println("There is no line");**

**} else {**

**LineNode temp = head;**

**while (temp != null && !temp.getData().toString().equals(lineName)) {**

**temp = temp.getDown();**

**}**

**if (temp != null) {**

**if (temp.getRight() == null) {**

**System.out.println("There is no card to delete in the line");**

**} else {**

**if (temp.getRight().getData().toString().equalsIgnoreCase(String.valueOf(cardNumber))) {**

**temp.setRight(temp.getRight().getNext());**

**} else {**

**ColumnNode tempC = temp.getRight();**

**ColumnNode prevC = null;**

**while (tempC != null**

**&& !tempC.getData().toString().equalsIgnoreCase(String.valueOf(cardNumber))) {**

**prevC = tempC;**

**tempC = tempC.getNext();**

**}**

**if (tempC != null) {**

**prevC.setNext(tempC.getNext());**

**} else {**

**System.out.println("There is no card like " + cardNumber);**

**}**

**}**

**}**

**} else {**

**System.out.println("There is no line like " + lineName);**

**}**

**}**

**}**

**public void display() {**

**if (head == null) {**

**System.out.println("There is no data to display.");**

**} else {**

**LineNode temp = head;**

**while (temp != null) {**

**System.out.print(temp.getData().toString() + " | ");**

**ColumnNode tempC = temp.getRight();**

**while (tempC != null) {**

**System.out.print(tempC.getData().toString() + " ");**

**tempC = tempC.getNext();**

**}**

**System.out.println();**

**temp = temp.getDown();**

**}**

**}**

**}**

**public void linePop(String line\_name) {**

**if (head == null) {**

**System.out.println("There is no line.");**

**} else {**

**LineNode temp = head;**

**while (temp.getDown() != null && !line\_name.equals(String.valueOf(temp.getData()))) {**

**temp = temp.getDown();**

**}**

**if (!line\_name.equals(String.valueOf(temp.getData()))) {**

**System.out.println("There is no line named '" + line\_name + "'.");**

**return;**

**}**

**ColumnNode tempC = temp.getRight();**

**if (tempC == null) {**

**System.out.println("There is no data to pop in line '" + line\_name + "'.");**

**} else if (tempC.getNext() == null) {**

**temp.setRight(null);**

**} else {**

**ColumnNode previous = null;**

**ColumnNode current = tempC;**

**while (current != null) {**

**if (current.getNext() == null) {**

**previous.setNext(current.getNext());**

**break;**

**}**

**previous = current;**

**current = current.getNext();**

**}**

**}**

**}**

**}**

**public void linePop(String line\_name, int index) {**

**if (head == null) {**

**System.out.println("There is no line.");**

**} else {**

**LineNode temp = head;**

**while (temp.getDown() != null && !line\_name.equals(String.valueOf(temp.getData()))) {**

**temp = temp.getDown();**

**}**

**if (!line\_name.equals(String.valueOf(temp.getData()))) {**

**System.out.println("There is no line named '" + line\_name + "'.");**

**return;**

**}**

**ColumnNode tempC = temp.getRight();**

**if (tempC == null) {**

**System.out.println("There is no data to pop in line " + line\_name + ".");**

**} else if (index == 0) {**

**temp.setRight(tempC.getNext());**

**} else {**

**if (index >= lineSize(line\_name)) {**

**System.out.println("Index out of bounds.");**

**System.out.println("Index:" + index + " Size:" + lineSize(line\_name) + " Line:" + line\_name);**

**}**

**int count = 0;**

**ColumnNode previous = null;**

**ColumnNode current = tempC;**

**while (current != null) {**

**if (index == count) {**

**previous.setNext(current.getNext());**

**break;**

**}**

**previous = current;**

**current = current.getNext();**

**count++;**

**}**

**}**

**}**

**}**

**public String getLineName(int index) {**

**if (head == null) {**

**System.out.println("There is no line.");**

**return null;**

**} else {**

**LineNode temp = head;**

**for (int i = 0; i < index; i++) {**

**temp = temp.getDown();**

**}**

**return String.valueOf(temp.getData());**

**}**

**}**

**public int lineSize(String line\_name) {**

**if (head == null) {**

**System.out.println("There is no line.");**

**return -1;**

**} else {**

**LineNode temp = head;**

**while (temp.getDown() != null && !line\_name.equals(String.valueOf(temp.getData()))) {**

**temp = temp.getDown();**

**}**

**if (!line\_name.equals(String.valueOf(temp.getData()))) {**

**System.out.println("There is no line named '" + line\_name + "'.");**

**return -1;**

**}**

**ColumnNode tempC = temp.getRight();**

**int count = 0;**

**while (tempC != null) {**

**count++;**

**tempC = tempC.getNext();**

**}**

**return count;**

**}**

**}**

**public int lineSize(int index) {**

**return lineSize(getLineName(index));**

**}**

**public int getCardFromLine(String line\_name, int index) {**

**if (head == null) {**

**System.out.println("There is no line.");**

**return -1;**

**} else {**

**LineNode temp = head;**

**while (temp.getDown() != null && !line\_name.equals(String.valueOf(temp.getData()))) {**

**temp = temp.getDown();**

**}**

**if (!line\_name.equals(String.valueOf(temp.getData()))) {**

**System.out.println("There is no line named '" + line\_name + "'.");**

**return -1;**

**}**

**ColumnNode tempC = temp.getRight();**

**for (int i = 0; i < index; i++) {**

**tempC = tempC.getNext();**

**}**

**return (int) tempC.getData();**

**}**

**}**

**public boolean isThereAnySet(String line\_name) {**

**boolean isSet = true;**

**LineNode temp = head;**

**while (temp.getDown() != null && !line\_name.equals(String.valueOf(temp.getData()))) {**

**temp = temp.getDown();**

**}**

**ColumnNode tempC = temp.getRight();**

**if ((Integer) tempC.getData() == 1) {**

**for (int i = 0; i < 10; i++) {**

**if (tempC != null && (Integer) tempC.getData() != i + 1) {**

**isSet = false;**

**break;**

**}**

**tempC = tempC.getNext();**

**}**

**} else if ((Integer) tempC.getData() == 10) {**

**for (int i = 0; i < 10; i++) {**

**if (tempC != null && (Integer) tempC.getData() != 10 - i) {**

**isSet = false;**

**break;**

**}**

**tempC = tempC.getNext();**

**}**

**} else {**

**isSet = false;**

**}**

**return isSet;**

**}**

**public int maxLineSize() {**

**int maxSize = -1; // means no line**

**LineNode temp = head;**

**while (temp != null) {**

**if (maxSize < lineSize((String) temp.getData())) {**

**maxSize = lineSize((String) temp.getData());**

**}**

**temp = temp.getDown();**

**}**

**return maxSize;**

**}**

**public int getCardFromLine(int x, int y) {**

**String lineName = (String) getLineName(x);**

**return getCardFromLine(lineName, y);**

**}**

**}**

**Main.java**

**public class Main {**

**public static void main(String[] args) throws Exception {**

**new Game();**

**}**

**}**

**ColumnNode.java**

**public class ColumnNode {**

**private Object data;**

**private ColumnNode next;**

**public ColumnNode(Object data) {**

**this.data = data;**

**this.next = null;**

**}**

**public Object getData() {**

**return data;**

**}**

**public void setData(Object data) {**

**this.data = data;**

**}**

**public ColumnNode getNext() {**

**return next;**

**}**

**public void setNext(ColumnNode next) {**

**this.next = next;**

**}**

**}**

**DLL\_Node.java**

**public class DLL\_Node {**

**private Object data;**

**private DLL\_Node prev;**

**private DLL\_Node next;**

**public DLL\_Node(Object dataToAdd) {**

**data = dataToAdd;**

**prev = null;**

**next = null;**

**}**

**public Object getData() {**

**return data;**

**}**

**public void setData(Object data) {**

**this.data = data;**

**}**

**public DLL\_Node getPrev() {**

**return prev;**

**}**

**public void setPrev(DLL\_Node prev) {**

**this.prev = prev;**

**}**

**public DLL\_Node getNext() {**

**return next;**

**}**

**public void setNext(DLL\_Node next) {**

**this.next = next;**

**}**

**}**

**DoublyLinkedList.java**

**public class DoublyLinkedList {**

**private DLL\_Node head;**

**private DLL\_Node tail;**

**public DoublyLinkedList() {**

**head = null;**

**tail = null;**

**}**

**public DLL\_Node getHead() {**

**return head;**

**}**

**public void setHead(DLL\_Node head) {**

**this.head = head;**

**}**

**public DLL\_Node getTail() {**

**return tail;**

**}**

**public void setTail(DLL\_Node tail) {**

**this.tail = tail;**

**}**

**public void addAsSorted(Object name, double scoredata) {**

**DLL\_Node newNode = new DLL\_Node(name + "-" + scoredata);**

**if (head == null) {**

**head = newNode;**

**tail = newNode;**

**} else {**

**DLL\_Node temp = head;**

**if (scoredata >= Double.parseDouble(temp.getData().toString().split("-")[1])) {**

**newNode.setNext(head);**

**head.setPrev(newNode);**

**head = newNode;**

**} else {**

**boolean flag = false;**

**while (flag == false && temp.getNext() != null**

**&& scoredata < Double.parseDouble(temp.getData().toString().split("-")[1])) {**

**if (flag == false**

**&& Double.parseDouble(temp.getNext().getData().toString().split("-")[1]) < scoredata) {**

**flag = true;**

**} else {**

**temp = temp.getNext();**

**}**

**}**

**newNode.setPrev(temp);**

**newNode.setNext(temp.getNext());**

**if (temp.getNext() != null) {**

**temp.getNext().setPrev(newNode);**

**} else**

**tail = newNode;**

**temp.setNext(newNode);**

**}**

**}**

**}**

**public int size() {**

**int count = 0;**

**if (head == null) {**

**System.out.println("List is empty.");**

**} else {**

**DLL\_Node temp = head;**

**while (temp != null) {**

**count++;**

**temp = temp.getNext();**

**}**

**}**

**return count;**

**}**

**public void display() {**

**if (head == null) {**

**System.out.println("List is empty.");**

**} else {**

**DLL\_Node temp = head;**

**while (temp != null) {**

**System.out.println(temp.getData() + " ");**

**temp = temp.getNext();**

**}**

**System.out.println();**

**}**

**}**

**}**

**SinglyLinkedList.java**

**public class SinglyLinkedList {**

**private SLL\_Node head;**

**public void add(Object data) {**

**if (head == null) {**

**SLL\_Node newNode = new SLL\_Node(data);**

**head = newNode;**

**} else {**

**SLL\_Node temp = head;**

**while (temp.getLink() != null) {**

**temp = temp.getLink();**

**}**

**SLL\_Node newNode = new SLL\_Node(data);**

**temp.setLink(newNode);**

**}**

**}**

**public int size() {**

**if (head == null) {**

**return 0;**

**} else {**

**int count = 0;**

**SLL\_Node temp = head;**

**while (temp != null) {**

**temp = temp.getLink();**

**count++;**

**}**

**return count;**

**}**

**}**

**public void display() {**

**if (head == null) {**

**System.out.println("List is empty.");**

**} else {**

**SLL\_Node temp = head;**

**while (temp != null) {**

**System.out.print(temp.getData() + " ");**

**temp = temp.getLink();**

**}**

**}**

**}**

**public void remove(Object dataToDelete) {**

**if (head == null) {**

**System.out.println("Linked list is empty.");**

**} else {**

**while ((Integer) head.getData() == (Integer) dataToDelete)**

**head = head.getLink();**

**SLL\_Node temp = head;**

**SLL\_Node previous = null;**

**while (temp != null) {**

**if ((Integer) temp.getData() == (Integer) dataToDelete) {**

**previous.setLink(temp.getLink());**

**temp = previous;**

**}**

**previous = temp;**

**temp = temp.getLink();**

**}**

**}**

**}**

**public int findMax() {**

**if (head == null) {**

**System.err.println("The Linked List is empty");**

**return Integer.MIN\_VALUE;**

**} else {**

**int maxVal = Integer.MIN\_VALUE;**

**SLL\_Node temp = head;**

**while (temp != null) {**

**if ((int) temp.getData() > maxVal) {**

**maxVal = (int) temp.getData();**

**}**

**temp = temp.getLink();**

**}**

**return maxVal;**

**}**

**}**

**public boolean search(Object data) {**

**if (head == null) {**

**System.out.println("List is empty.");**

**return false;**

**} else {**

**SLL\_Node temp = head;**

**while (temp != null) {**

**if ((Integer) temp.getData() == (Integer) data) {**

**return true;**

**}**

**temp = temp.getLink();**

**}**

**return false;**

**}**

**}**

**// returns head data of the sll**

**public Object get() {**

**if (head == null) {**

**System.out.println("List is empty.");**

**return -1;**

**} else {**

**return head.getData();**

**}**

**}**

**// returns data due to index number**

**public Object get(int index) {**

**if (head == null) {**

**return -1;**

**} else {**

**SLL\_Node temp = head;**

**int count = 0;**

**while (temp.getData() != null && index != count) {**

**temp = temp.getLink();**

**count++;**

**}**

**return temp.getData();**

**}**

**}**

**// removes the number that first found**

**public void delete(int dataToDelete) {**

**if (head == null) {**

**System.out.println("Link is empty.");**

**return;**

**}**

**if ((Integer) head.getData() == (Integer) dataToDelete) {**

**head = head.getLink();**

**} else {**

**SLL\_Node previous = head;**

**SLL\_Node current = head.getLink();**

**while (current != null) {**

**if ((Integer) current.getData() == (Integer) dataToDelete) {**

**previous.setLink(current.getLink());**

**break;**

**}**

**previous = current;**

**current = current.getLink();**

**}**

**}**

**}**

**// delete the last node from sll**

**public void pop() {**

**if (head == null) {**

**System.out.println("Link is empty.");**

**return;**

**}**

**if (head.getLink() == null) {**

**head = null;**

**} else {**

**SLL\_Node previous = null;**

**SLL\_Node current = head;**

**while (current != null) {**

**if (current.getLink() == null) {**

**previous.setLink(current.getLink());**

**break;**

**}**

**previous = current;**

**current = current.getLink();**

**}**

**}**

**}**

**// delete the data from sll due to index**

**public void pop(int index) {**

**if (head == null) {**

**System.out.println("Link is empty.");**

**return;**

**}**

**if (head.getLink() == null) {**

**head = null;**

**} else {**

**SLL\_Node previous = null;**

**SLL\_Node current = head;**

**int count = 0;**

**while (current != null) {**

**if (count == index) {**

**previous.setLink(current.getLink());**

**break;**

**}**

**count++;**

**previous = current;**

**current = current.getLink();**

**}**

**}**

**}**

**}**

**LineNode.java**

**public class LineNode {**

**private Object data;**

**private LineNode down;**

**private ColumnNode right;**

**public LineNode(Object data) {**

**this.data = data;**

**this.down = null;**

**this.right = null;**

**}**

**public Object getData() {**

**return data;**

**}**

**public void setData(Object data) {**

**this.data = data;**

**}**

**public LineNode getDown() {**

**return down;**

**}**

**public void setDown(LineNode down) {**

**this.down = down;**

**}**

**public ColumnNode getRight() {**

**return right;**

**}**

**public void setRight(ColumnNode right) {**

**this.right = right;**

**}**

**}**

**SLL\_Node.java**

**public class SLL\_Node {**

**private Object data;**

**private SLL\_Node link; // next**

**public SLL\_Node(Object dataToAdd) {**

**data = dataToAdd;**

**link = null;**

**}**

**public Object getData() {**

**return data;**

**}**

**public void setData(Object data) {**

**this.data = data;**

**}**

**public SLL\_Node getLink() {**

**return link;**

**}**

**public void setLink(SLL\_Node link) {**

**this.link = link;**

**}**

**}**

**Game.java**

**import enigma.console.TextAttributes;**

**import enigma.core.Enigma;**

**import java.awt.Color;**

**import java.awt.event.KeyEvent;**

**import java.awt.event.KeyListener;**

**import java.io.FileNotFoundException;**

**import java.io.FileReader;**

**import java.util.Scanner;**

**import java.util.Random;**

**public class Game {**

**enigma.console.Console cn = Enigma.getConsole("Columns", 50, 39, 15);**

**KeyListener klis = new KeyListener() {**

**public void keyTyped(KeyEvent e) {**

**}**

**public void keyPressed(KeyEvent e) {**

**if (keypr == 0) {**

**keypr = 1;**

**rkey = e.getKeyCode();**

**}**

**}**

**public void keyReleased(KeyEvent e) {**

**}**

**};**

**int keypr; // key pressed?**

**int rkey; //**

**boolean isEnterPressed = false;**

**boolean isEscapePressed = false;**

**// Text Attributes**

**TextAttributes DEFAULT = new TextAttributes(hexToColor("#ECE6CE"), Color.BLACK);**

**TextAttributes MENUCURSOR = new TextAttributes(Color.GREEN, rgbToColor(0, 0, 80));**

**TextAttributes GAMECURSOR = new TextAttributes(Color.BLACK, Color.RED);**

**TextAttributes SELECTED = new TextAttributes(Color.RED, Color.BLACK);**

**TextAttributes HIGHLIGHTED = new TextAttributes(Color.BLACK, Color.GREEN);**

**TextAttributes SCORETABLE = new TextAttributes(Color.pink, Color.black);**

**// variables**

**String[] MENU = { "Play", "High Score Table", "Options", "How to Play", "Prepared by", "Exit (ESC)" };**

**int transfer\_number = 0;**

**double score = 0; // for try**

**int completed\_sets = 0;**

**int column\_amount = 5; // how many columns game has**

**int box\_number = -1; // -1 if it is invisible else revealed number**

**boolean isBoxSelected = false;**

**boolean isGameOver = false;**

**int x = 0;**

**int y = 0;**

**int selected\_x = -1;**

**int selected\_y = -1;**

**int[] selected\_cards = new int[200];**

**int line = 0;**

**SinglyLinkedList box = new SinglyLinkedList();**

**MultiLinkedList columns = new MultiLinkedList();**

**// constructor**

**public Game() throws Exception {**

**cn.getTextWindow().addKeyListener(klis);**

**start();**

**}**

**// game initliaze**

**public void start() throws FileNotFoundException, InterruptedException {**

**Boolean exit = false;**

**do {**

**printMenu();**

**menuKeys();**

**if (isEnterPressed) { // enter**

**clear(cn);**

**if (line == 0) { // play**

**play();**

**} else if (line == 1) { // highscoretable**

**printHighScoreTable(score);**

**} else if (line == 2) { // MENU**

**optionsMenu();**

**} else if (line == 3) { // how to play**

**printHowToPlay();**

**} else if (line == 4) { // prepared by**

**printPreparedBy();**

**} else if (line == 5) { // exit**

**exit = true;**

**}**

**// esc to exit to menu**

**boolean flag = false;**

**while (!flag) {**

**if (keypr == 1) {**

**if (rkey == KeyEvent.VK\_ESCAPE) {**

**flag = true;**

**break;**

**}**

**keypr = 0;**

**}**

**}**

**isEnterPressed = false;**

**}**

**Thread.sleep(100);**

**clear(cn);**

**} while (!exit);**

**}**

**public void menuKeys() {**

**if (keypr == 1) { // if keyboard button pressed**

**keypr = 0;**

**if (rkey == KeyEvent.VK\_UP) {**

**if (line > 0) {**

**line--;**

**}**

**} else if (rkey == KeyEvent.VK\_DOWN) {**

**if (line < MENU.length - 1) {**

**line++;**

**}**

**} else if (rkey == KeyEvent.VK\_ENTER) {**

**isEnterPressed = true;**

**}**

**keypr = 0; // last action**

**}**

**}**

**public void printMenu() {**

**TextAttributes random = randomTextAttributesForColor(Color.BLACK);**

**print(cn, "+------------------------+", 11, 1, random);**

**print(cn, "|", 11, 2, random);**

**print(cn, "▆▅▃▂▁𝐂𝐎𝐋𝐔𝐌𝐍𝐒▁▂▃▅▆", 12, 2, random);**

**print(cn, "|", 36, 2, random);**

**print(cn, "+------------------------+", 11, 3, random);**

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

**if (line == i) {**

**print(cn, MENU[i], 15, 6 + i, MENUCURSOR);**

**} else {**

**print(cn, MENU[i], 15, 6 + i, DEFAULT);**

**}**

**}**

**}**

**public void printHowToPlay() {**

**print(cn, "+-----------------------------+", 9, 1, new TextAttributes(Color.black, Color.pink));**

**print(cn, "|", 9, 2, new TextAttributes(Color.black, Color.pink));**

**print(cn, " HOW TO PLAY ", 10, 2, new TextAttributes(Color.white, Color.red));**

**print(cn, "|", 39, 2, new TextAttributes(Color.black, Color.pink));**

**print(cn, "+-----------------------------+", 9, 3, new TextAttributes(Color.black, Color.pink));**

**print(cn, "GENERAL INFORMATION", 15, 5, new TextAttributes(Color.red, Color.black));**

**print(cn, "The game is played in 5 columns. Game elements", 1, 6, DEFAULT);**

**print(cn, "are numbers (1-10). The aim of the game is", 1, 7, DEFAULT);**

**print(cn, "reaching the highest score by collecting number", 1, 8, DEFAULT);**

**print(cn, "sets.", 1, 9, DEFAULT);**

**print(cn, "GAME ELEMENTS ", 17, 11, new TextAttributes(Color.red, Color.black));**

**print(cn, "Number Set = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.", 1, 12, DEFAULT);**

**print(cn, "GAME KEYS", 19, 14, new TextAttributes(Color.red, Color.black));**

**print(cn, "Up, Down, Left and Right Arrows: Arrow keys", 1, 15, DEFAULT);**

**print(cn, "Arrow keys helps player to travel from column", 1, 16, DEFAULT);**

**print(cn, "to column.", 1, 17, DEFAULT);**

**print(cn, "Z-key: Pressing Z key, makes player select a", 1, 19, DEFAULT);**

**print(cn, "number from column by moving the cursor to that", 1, 20, DEFAULT);**

**print(cn, "number.", 1, 21, DEFAULT);**

**print(cn, "X-key: Pressing X key, makes player select a", 1, 23, DEFAULT);**

**print(cn, "column by moving the cursor to that column and", 1, 24, DEFAULT);**

**print(cn, "by moving the cursor to ", 1, 25, DEFAULT);**

**print(cn, "the drawn number is transferred to the end", 1, 26, DEFAULT);**

**print(cn, "of the column.", 1, 27, DEFAULT);**

**print(cn, "B-key: Pressing B key, makes player select and", 1, 29, DEFAULT);**

**print(cn, "draw a numberfrom the box. ", 1, 30, DEFAULT);**

**print(cn, "Press Escape(ESC) to exit.", 12, 32, DEFAULT);**

**}**

**public void printPreparedBy() {**

**print(cn, "+-------------------+", 14, 1, DEFAULT);**

**print(cn, "| PROJECT III |", 14, 2, DEFAULT);**

**print(cn, "| COLUMNS |", 14, 3, DEFAULT);**

**print(cn, "| PREPARED BY |", 14, 4, DEFAULT);**

**print(cn, "+-------------------+", 14, 5, DEFAULT);**

**print(cn, " --- LAB6 GROUP4 --- ", 14, 6, DEFAULT);**

**print(cn, " -GURKAN BIYIK ", 14, 7, DEFAULT);**

**print(cn, " -DURU CAPAR ", 14, 8, DEFAULT);**

**print(cn, " -DENIZ KATAYIFCI ", 14, 9, DEFAULT);**

**print(cn, " -SAHIN OZTURK ", 14, 10, DEFAULT);**

**print(cn, " Izmir/Turkey ", 16, 12, DEFAULT);**

**print(cn, " May 2022 ", 16, 13, DEFAULT);**

**print(cn, "Press Escape(ESC) to exit.", 12, 15, DEFAULT);**

**}**

**public void optionsMenu() throws InterruptedException {**

**// esc to exit to menu**

**boolean flag = false;**

**while (!flag) {**

**print(cn, "+------------------------------+", 9, 1, DEFAULT);**

**print(cn, "| OPTIONS |", 9, 2, DEFAULT);**

**print(cn, "+------------------------------+", 9, 3, DEFAULT);**

**print(cn, " ", 10, 5, MENUCURSOR);**

**print(cn, "Column Amount (Default: 5): " + column\_amount, 10, 5, MENUCURSOR);**

**print(cn, "Use left and right arrow keys", 10, 7, DEFAULT);**

**print(cn, "to change option. (<- ->)", 10, 8, DEFAULT);**

**print(cn, "Press Escape(ESC) to exit.", 10, 10, DEFAULT);**

**Thread.sleep(60);**

**if (keypr == 1) {**

**if (rkey == KeyEvent.VK\_ESCAPE) {**

**flag = true;**

**break;**

**} else if (rkey == KeyEvent.VK\_LEFT) {**

**if (column\_amount > 3) {**

**column\_amount--;**

**}**

**} else if (rkey == KeyEvent.VK\_RIGHT) {**

**if (column\_amount < 7) {**

**column\_amount++;**

**}**

**}**

**keypr = 0;**

**}**

**}**

**}**

**// game loop**

**public void play() throws InterruptedException, FileNotFoundException {**

**resetGameVariables();**

**createLines(columns, column\_amount);**

**createBox(box);**

**dealTheCards(box, 6);**

**while (completed\_sets != column\_amount) {**

**clearValues(3);**

**printGameScreen(3);**

**gameKeys();**

**checkColumns();**

**if (isGameOver) {**

**break;**

**}**

**Thread.sleep(30);**

**}**

**if (completed\_sets == column\_amount) {**

**score = (score / transfer\_number) + (completed\_sets \* 100);**

**score = (double) (Math.round(score \* 100.0) / 100.0);**

**print(cn, "End-Game-Score: " + score, 8, 15, DEFAULT);**

**} else {**

**score = 0;**

**}**

**print(cn, "Press Escape(ESC) to exit.", 8, 16, DEFAULT);**

**}**

**public void resetGameVariables() {**

**transfer\_number = 0;**

**score = 0;**

**completed\_sets = 0;**

**box\_number = -1; // -1 if it is invisible else revealed number**

**isBoxSelected = false;**

**isGameOver = false;**

**x = 0;**

**y = 0;**

**selected\_x = -1;**

**selected\_y = -1;**

**selected\_cards = new int[200];**

**box = new SinglyLinkedList();**

**columns = new MultiLinkedList();**

**}**

**// string printer with TextAttributes**

**public void print(enigma.console.Console console, String str, int x, int y, TextAttributes attr) {**

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

**console.getTextWindow().output(x + i, y, str.charAt(i), attr);**

**}**

**}**

**// console clear**

**public void clear(enigma.console.Console console) {**

**int col = console.getTextWindow().getColumns();**

**int row = console.getTextWindow().getRows();**

**int size = (col \* row) - 1;**

**String str = "";**

**for (int i = 0; i < size; i++) {**

**str += " ";**

**}**

**console.getTextWindow().setCursorPosition(0, 0);**

**cn.setTextAttributes(DEFAULT);**

**System.out.print(str);**

**console.getTextWindow().setCursorPosition(0, 0);**

**}**

**public void gameKeys() throws InterruptedException {**

**if (keypr == 1) { // if keyboard button pressed**

**if (rkey == KeyEvent.VK\_LEFT) {**

**if (x == 0 && y >= columns.lineSize(column\_amount - 1)) {**

**x = column\_amount - 1;**

**y = columns.lineSize(column\_amount - 1) - 1;**

**} else if (x == 0) {**

**x = column\_amount - 1;**

**} else if (x > 0 && columns.lineSize(x - 1) == 0) {**

**x--;**

**y = 0;**

**} else if (x > 0 && (y >= columns.lineSize(x - 1))) {**

**y = columns.lineSize(x - 1) - 1;**

**x--;**

**} else if (x > 0) {**

**x--;**

**}**

**} else if (rkey == KeyEvent.VK\_RIGHT) {**

**if (x == column\_amount - 1 && y >= columns.lineSize(0)) {**

**x = 0;**

**y = columns.lineSize(0) - 1;**

**} else if (x == column\_amount - 1) {**

**x = 0;**

**} else if (x < column\_amount - 1 && columns.lineSize(x + 1) == 0) {**

**x++;**

**y = 0;**

**} else if (x < column\_amount - 1 && (y >= columns.lineSize(x + 1))) {**

**y = columns.lineSize(x + 1) - 1;**

**x++;**

**} else if (x < column\_amount - 1) {**

**x++;**

**}**

**} else if (rkey == KeyEvent.VK\_UP) {**

**if (y == 0 && columns.lineSize(x) != 0) {**

**y = columns.lineSize(x) - 1;**

**} else if (y > 0) {**

**y--;**

**}**

**} else if (rkey == KeyEvent.VK\_DOWN) {**

**if (y < columns.lineSize(x) - 1) {**

**y++;**

**} else if (y == columns.lineSize(x) - 1) {**

**y = 0;**

**}**

**;**

**} else if (rkey == KeyEvent.VK\_Z) {**

**// selected\_cards[0] == 0 --> no selected card via key z**

**if (!isBoxSelected && selected\_cards[0] == 0) {**

**int index = 0;**

**selected\_x = x;**

**selected\_y = y;**

**int len = columns.lineSize(x);**

**for (int i = y; i < len; i++) {**

**int card = (int) columns.getCardFromLine(columns.getLineName(x), i);**

**selected\_cards[index] = card;**

**index++;**

**}**

**} else { // cancel the selection**

**selected\_cards = new int[200];**

**selected\_x = -1;**

**selected\_y = -1;**

**}**

**} else if (rkey == KeyEvent.VK\_B) {**

**if (box\_number == -1) { // box number not revealead**

**box\_number = (int) box.get(box.size() - 1);**

**} else { // box number revealead**

**// selected\_cards[0] == 0 --> no selected card via key z**

**if (!isBoxSelected && selected\_cards[0] == 0) {**

**isBoxSelected = true;**

**box.pop();**

**selected\_x = -1;**

**selected\_y = -1;**

**selected\_cards[0] = box\_number;**

**} else { // cancel the selection**

**isBoxSelected = false;**

**box.add(selected\_cards[0]);**

**selected\_cards[0] = 0;**

**}**

**}**

**} else if (rkey == KeyEvent.VK\_X) { // adds selected cards to columns**

**if (canTransferToColumn(columns, x, selected\_cards[0])) {**

**// adds selected cards to column if conditions are true**

**for (int i = 0; i < selected\_cards.length; i++) {**

**int card = selected\_cards[i];**

**if (card == 0) {**

**break;**

**} else {**

**String line\_name = columns.getLineName(x);**

**columns.addCard(line\_name, card);**

**}**

**}**

**transfer\_number++;**

**if (!isBoxSelected) { // if card not came from box deletes the cards where taken from**

**for (int j = 0; j < selected\_cards.length; j++) {**

**if (selected\_cards[j] == 0) {**

**break;**

**}**

**String line\_name = columns.getLineName(selected\_x);**

**columns.linePop(line\_name);**

**}**

**}**

**// if added card is came from box resets the box\_number**

**if (isBoxSelected) {**

**box\_number = -1;**

**isBoxSelected = false;**

**}**

**// resets selected cards**

**selected\_cards = new int[200];**

**selected\_x = -1;**

**selected\_y = -1;**

**}**

**} else if (rkey == KeyEvent.VK\_E) { // exit the game**

**isGameOver = true;**

**}**

**keypr = 0; // last action**

**}**

**}**

**public void printGameScreen(int column\_width) {**

**print(cn, "Transfer:" + String.format("%6s", transfer\_number), column\_amount \* column\_width + 5, 0, DEFAULT);**

**print(cn, "Score :" + String.format("%6s", score), column\_amount \* column\_width + 5, 1, DEFAULT);**

**// columns**

**for (int i = 0; i < column\_amount; i++) {**

**print(cn, "C" + (i + 1), i \* column\_width, 0, DEFAULT);**

**print(cn, "--", i \* column\_width, 1, DEFAULT);**

**int len = columns.lineSize(i);**

**if (len == 0 && i == x) {**

**print(cn, " ", i \* column\_width, 2, GAMECURSOR);**

**} else if (len == 0 && i != x) {**

**print(cn, " ", i \* column\_width, 2, DEFAULT);**

**} else {**

**for (int j = 0; j < len; j++) {**

**int num = (int) columns.getCardFromLine(columns.getLineName(i), j);**

**String str = String.valueOf(num);**

**str = String.format("%2s", str);**

**if (j == y && i == x) {**

**print(cn, str, i \* column\_width, j + 2, GAMECURSOR);**

**} else if (!isBoxSelected && j == selected\_y && i == selected\_x) {**

**print(cn, str, i \* column\_width, j + 2, SELECTED);**

**} else {**

**print(cn, str, i \* column\_width, j + 2, DEFAULT);**

**}**

**if (j == selected\_y && i == selected\_x && len == selected\_y) {**

**print(cn, " ", i \* column\_width, 2, GAMECURSOR);**

**}**

**}**

**}**

**}**

**// highligth columns can be added**

**for (int i = 0; i < column\_amount; i++) {**

**if (i != selected\_x && (i != x || (i == x && y != (columns.lineSize(i) - 1)))**

**&& canTransferToColumn(columns, i, selected\_cards[0])) {**

**String st;**

**if (columns.lineSize(i) == 0) {**

**st = " ";**

**if (i == x && y == 0) {**

**print(cn, st, i \* column\_width, 2, GAMECURSOR);**

**} else {**

**print(cn, st, i \* column\_width, 2, HIGHLIGHTED);**

**}**

**} else {**

**int n = (int) columns.getCardFromLine(columns.getLineName(i),**

**columns.lineSize(i) - 1);**

**st = String.valueOf(n);**

**st = String.format("%2s", st);**

**print(cn, st, i \* column\_width, columns.lineSize(i) + 1, HIGHLIGHTED);**

**}**

**}**

**}**

**// box**

**print(cn, "Box", column\_amount \* column\_width + 5, 4, DEFAULT);**

**print(cn, "+--+", column\_amount \* column\_width + 5, 5, DEFAULT);**

**if (box\_number == -1) {**

**print(cn, "| |", column\_amount \* column\_width + 5, 6, DEFAULT);**

**} else {**

**if (isBoxSelected) {**

**print(cn, "|", column\_amount \* column\_width + 5, 6, DEFAULT);**

**print(cn, String.format("%3s", box\_number), column\_amount \* column\_width + 5, 6, SELECTED);**

**print(cn, "|", column\_amount \* column\_width + 5, 6, DEFAULT);**

**} else {**

**print(cn, "|" + String.format("%3s", box\_number + "|"), column\_amount \* column\_width + 5, 6, DEFAULT);**

**}**

**}**

**print(cn, "+--+", column\_amount \* column\_width + 5, 7, DEFAULT);**

**}**

**public void clearValues(int column\_width) {**

**print(cn, " ", column\_amount \* column\_width + 14, 0, DEFAULT); // transfer**

**print(cn, " ", column\_amount \* column\_width + 14, 1, DEFAULT); // score**

**// columns**

**for (int i = 0; i < column\_amount; i++) {**

**for (int j = 0; j < 37; j++) {**

**print(cn, " ", i \* column\_width, j + 2, DEFAULT);**

**}**

**}**

**// box**

**print(cn, " ", column\_amount \* column\_width + 6, 6, DEFAULT);**

**}**

**public void createBox(SinglyLinkedList box) {**

**Random rand = new Random();**

**int tempRandom = 0;**

**int[] counter = new int[10];**

**for (int i = 0; i < 10; i++) {**

**counter[i] = 0;**

**}**

**for (int i = 0; i < column\_amount \* 10;) {**

**tempRandom = rand.nextInt(10);**

**if (counter[tempRandom] < column\_amount) {**

**box.add(tempRandom + 1);**

**counter[tempRandom]++;**

**i++;**

**}**

**}**

**}**

**public void createLines(MultiLinkedList columns, int column\_amount) {**

**for (int i = 0; i < column\_amount; i++) {**

**columns.addLine("C" + (i + 1));**

**}**

**}**

**public void dealTheCards(SinglyLinkedList box, int number) {**

**if (box.size() == 0) {**

**System.out.println("List is empty.");**

**} else {**

**int tempNumber = 0;**

**for (int i = 0; i < column\_amount; i++) {**

**for (int j = 0; j < number; j++) {**

**tempNumber = (Integer) box.get();**

**columns.addCard(columns.getLineName(i), tempNumber);**

**box.delete(tempNumber);**

**}**

**}**

**}**

**}**

**public boolean canTransferToColumn(MultiLinkedList columns, int line\_index, int card) {**

**if (card == 0) {**

**return false;**

**}**

**if (columns.lineSize(line\_index) == 0 && (card == 1 || card == 10)) {**

**return true;**

**} else if (columns.lineSize(line\_index) == 0 && (card != 1 && card != 10)) {**

**return false;**

**} else {**

**String line\_name = (String) columns.getLineName(line\_index);**

**int column\_number = (int) columns.getCardFromLine(line\_name, columns.lineSize(line\_index) - 1);**

**return column\_number - 1 == card || column\_number == card || column\_number + 1 == card;**

**}**

**}**

**// prints score from DLL high\_score\_table**

**public void printHighScoreTable(double playerScore) throws FileNotFoundException {**

**print(cn, " +-----------------------+", 12, 1, DEFAULT);**

**print(cn, " | |", 12, 2, DEFAULT);**

**print(cn, " | \* HIGH SCORE \* |", 12, 3, DEFAULT);**

**print(cn, " | \* TABLE \* |", 12, 4, DEFAULT);**

**print(cn, " | |", 12, 5, DEFAULT);**

**print(cn, " +-----------------------+", 12, 6, DEFAULT);**

**String[] arr = new String[5];**

**DLL\_Node temp = highscoretableData(playerScore).getHead();**

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

**arr[i] = temp.getData().toString();**

**temp = temp.getNext();**

**}**

**for (int j = 0; j < arr.length; j++) {**

**print(cn, " ", 14, 7 + j, SCORETABLE);**

**print(cn, (j + 1) + "-) " + arr[j], 14, 7 + j, SCORETABLE);**

**}**

**print(cn, "Press Escape(ESC) to exit.", 12, 15, DEFAULT);**

**}**

**// checks columns if there is set pattern**

**// updates columns and increases completed\_sets and score**

**public void checkColumns() {**

**for (int i = 0; i < column\_amount; i++) {**

**String lineName = columns.getLineName(i);**

**if (columns.lineSize(lineName) == 10) {**

**boolean set\_exist = columns.isThereAnySet(lineName);**

**if (set\_exist) {**

**for (int j = 0; j < 10; j++) {**

**columns.linePop(lineName);**

**}**

**score += 1000;**

**completed\_sets++;**

**}**

**}**

**}**

**}**

**// creates a DoublyLinkedList type high score table**

**public DoublyLinkedList highscoretableData(double playerScore)**

**throws FileNotFoundException {**

**DoublyLinkedList highScoreTable = new DoublyLinkedList();**

**FileReader scoreTable = new FileReader("highscoretable.txt");**

**Scanner file = new Scanner(scoreTable);**

**while (file.hasNextLine()) {**

**String fileData = file.nextLine();**

**String[] arr = fileData.split(" ");**

**String fullname = arr[0] + " " + arr[1];**

**Double score = Double.parseDouble(arr[2]);**

**highScoreTable.addAsSorted(fullname, score);**

**}**

**if (completed\_sets == column\_amount) {**

**highScoreTable.addAsSorted("You", playerScore);**

**} else {**

**highScoreTable.addAsSorted("You", 0);**

**}**

**file.close();**

**return highScoreTable;**

**}**

**public TextAttributes randomTextAttributesBacColor(Color BackgroundColor) {**

**Color[] arr = { Color.GREEN, Color.BLUE, Color.YELLOW, Color.RED };**

**int random = (int) (Math.random() \* arr.length);**

**return new TextAttributes(arr[random], BackgroundColor);**

**}**

**public TextAttributes randomTextAttributesForColor(Color ForegroundColor) {**

**Color[] arr = { Color.GREEN, Color.BLUE, Color.YELLOW, Color.RED };**

**int random = (int) (Math.random() \* arr.length);**

**return new TextAttributes(ForegroundColor, arr[random]);**

**}**

**public Color rgbToColor(int r, int g, int b) {**

**if (r < 0 || r > 255 || g < 0 || g > 255 || b < 0 || b > 255) { // returns color red for errors**

**return Color.RED;**

**} else {**

**Color myColor = new Color(r, g, b);**

**return myColor;**

**}**

**}**

**public Color hexToColor(String hex) {**

**int r = Integer.valueOf(hex.substring(1, 3), 16);**

**int g = Integer.valueOf(hex.substring(3, 5), 16);**

**int b = Integer.valueOf(hex.substring(5, 7), 16);**

**if (r < 0 || r > 255 || g < 0 || g > 255 || b < 0 || b > 255) { // returns color red for errors**

**return Color.RED;**

**} else {**

**Color myColor = new Color(r, g, b);**

**return myColor;**

**}**

**}**

**}**

**HighScoreTable.txt**

**Sema Firat 378.24**

**Tarik Deniz 521.07**

**Ali Vadi 136.32**

**Seda Ruzgar 224.67**