Table of Contents

[1. Abstract 3](#_Toc152611701)

[1.1 What is your project about? 3](#_Toc152611702)

[1.2 Accomplished Objectives 3](#_Toc152611703)

[2. Introduction 3](#_Toc152611704)

[2.1 Brief overview 3](#_Toc152611705)

[2.2 Problem Statement 3](#_Toc152611706)

[3. UML Diagram 3](#_Toc152611707)

[4. Visual Representation of Data Structures Used 3](#_Toc152611708)

[5. Classes in text editor 4](#_Toc152611709)

[5.1 Node 4](#_Toc152611710)

[5.2 BstNode 4](#_Toc152611711)

[5.3 FileNode 4](#_Toc152611712)

[5.4 LinkedStrings 4](#_Toc152611713)

[5.5 BinarySearchTree 4](#_Toc152611714)

[5.6 TextEditor 5](#_Toc152611715)

[5.7 FileManager 5](#_Toc152611716)

[5.8 Main Class 5](#_Toc152611717)

[6. Code 6](#_Toc152611718)

[7. Output in Console 6](#_Toc152611719)

[8. Conclusion 6](#_Toc152611720)

[9. References 6](#_Toc152611721)

# 1. Abstract

## 1.1 What is your project about?

Our project is a program that implements different data structures to create a text editor. This system can be used by a user to write, replace ,delete and save text. Our project is a program that implements different data structures to create a text editor.

## 1.2 Accomplished Objectives

Our main goal in this project was to implement data structures in Java while creating a good, useful system. After a lot of hard work and dedication, we can confidently say that we have successfully accomplished these objectives.

# 2. Introduction

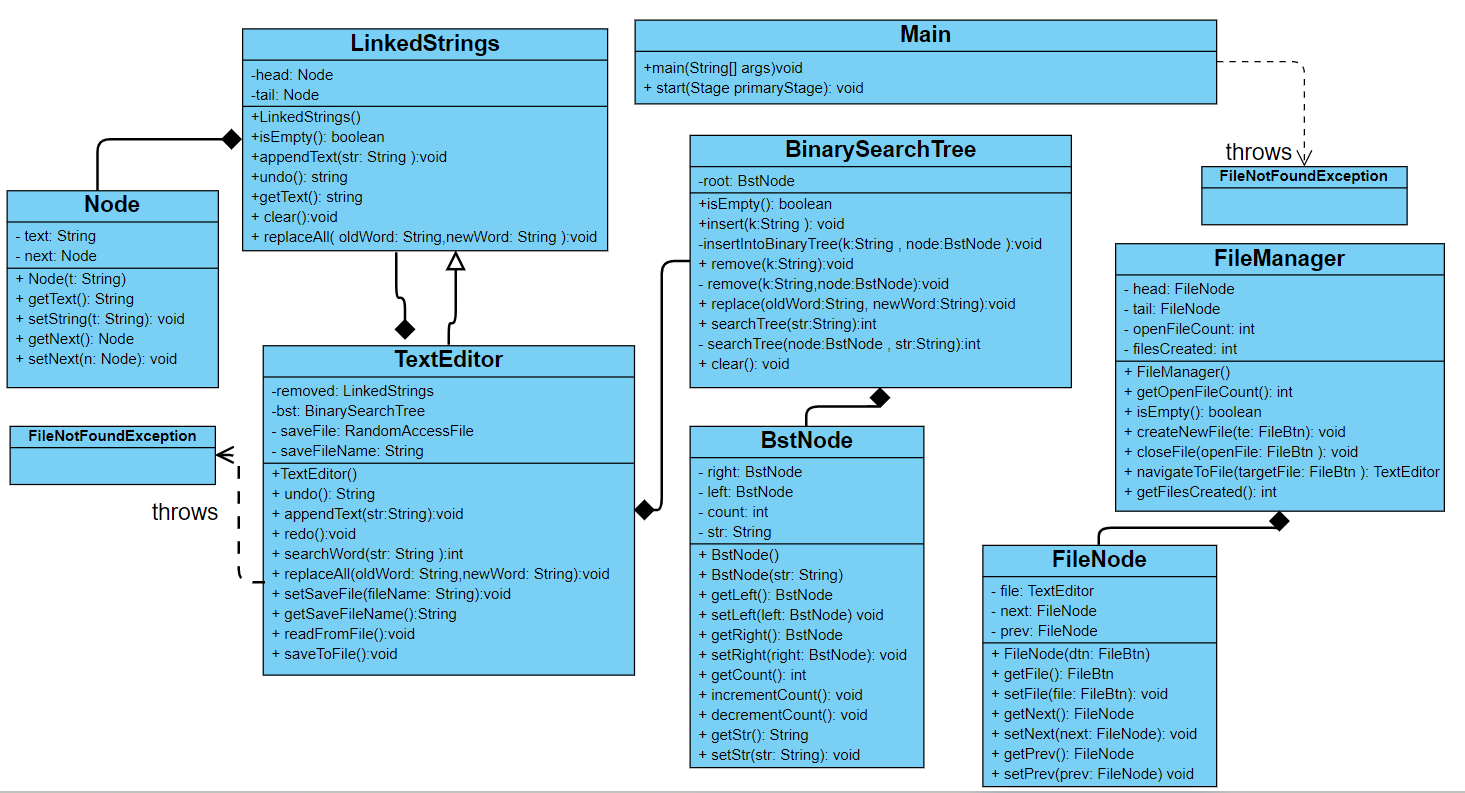
## 2.1 Brief overview

This text editor is a program that aims to allow the user to write and edit Text and save it for a later use .

## 2.2 Problem Statement

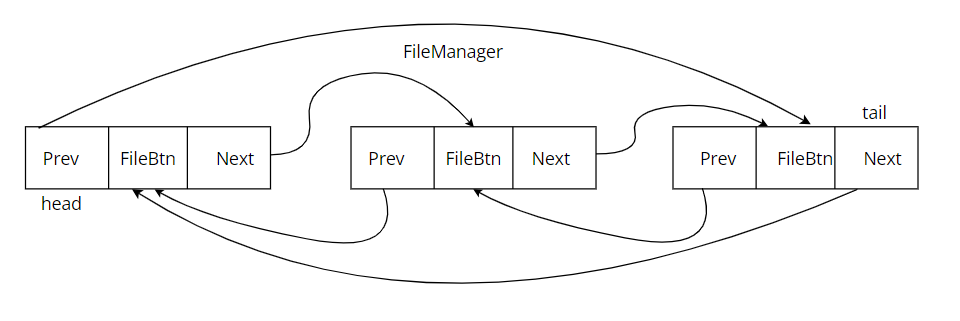
By creating this system, we give the user a way of creating, editing, and organizing written content , safeguarding important information from loss or damage without the use of paper or online sources.

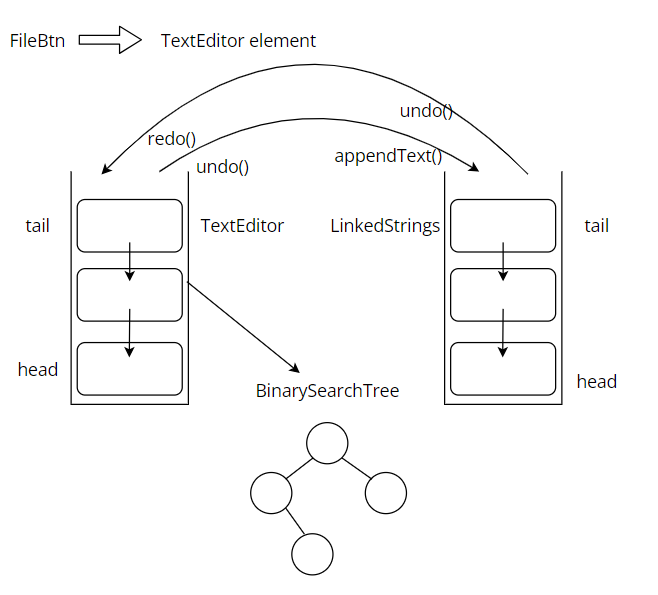
# 3. UML Diagram



# 4. Visual Representation of Data Structures Used

The data structures implemented in this system are linked lists, stacks and a binary search tree. Each individual file is organized in a circular doubly linked list ,the TextEditor class store each word separated by a space in a node inside a stack using a linked list ,and will call the BinarySearchTree class to find a specific word using a Binary Search Tree , and when deleting a word the TextEditor will store it in the LinkedStrings class that stores each one in a node inside a stack using a linked list .





# 5. Classes in text editor

## 5.1 Node

This class contains 2 fields and 1 constructor. This class is not used in the main. It is only used for implementing circular doubly linked lists.

## 5.2 BstNode

This class contains 4 fields and 1 constructor. This class is not used in the main. It is only used for implementing Binary Search Trees.

## 5.3 FileNode

This class contains 2 fields and 1 constructor. This class is not used in the main. It is only used for implementing Files.

## 5.4 LinkedStrings

This class is a linked list stack that stores the words that are deleted in the TextEditor class to insert them later . This class implements various functions described briefly below.

* LinkedStrings(): Default constructor that initializes the class attribute.
* isEmpty(): returns true if the linkedlist is empty.
* clear(): deletes everything int the linked list.
* appendText(String str): inserts an element with the value of “str” at the end of the linkedlist.
* undo(): removes the last element entered to the linkedlist.
* replace(String oldWord, String newWord): replaces the first element with a getText value equal to “oldWord “,to the value “newWord”.
* replaceAll(String oldWord, String newWord): replaces all elements with a getText value equal to “oldWord “,to the value “newWord”.
* getText(): returns the full sentence .

## 5.5 BinarySearchTree

This class is a Binary Search Tree that stores and searches for a word the user wants. This class implements various functions described briefly below.

* isEmpty(): returns true if the tree is empty.
* clear(): empties the tree.
* insert(String k): calls the recursive method insertIntoBinaryTree(String k, BstNode node):
* insertIntoBinaryTree(String k, BstNode node): recursively inserts a node with a String k into the binary search tree.
* remove(String k): calls the recursive method remove(String k, BstNode node)
* remove(String k, BstNode node): searches fore and removes the node with a str value equal to k.
* replace(String oldWord, String newWord):replaces the oldWord with a newWord.
* searchTree(String str): calls the recursive method searchTree(BstNode node, String str)
* searchTree(BstNode node, String str): recursively return true if the str exits in the binary search tree otherwise false.

## 5.6 TextEditor

This class is a linked list stack that stores and deletes words . This class implements various functions described briefly below.

* clear(): empty the TextEditor.
* undo(): removes the last node entered and preserves it in the LinkedStrings .
* appendText(String str): inserts an element with the value of “str” at the end .
* redo():calls the appendText function from the LinkedStrings class to bring back the text that was deleted .
* searchWord(String str): uses the BinarySearchTree to return true if the word “str” is present.
* replace(String oldWord, String newWord):finds the first word equal to “oldWord” and replaces it with “newWord”.
* replaceAll(String oldWord, String newWord): finds all word equal to “oldWord” and replaces them with “newWord”.
* setSaveFile(String fileName): checks which file the user wants to work with ,and if no file is called “fileName” it creates one with that name.
* getSaveFileName(): returns saveFileName.
* readFromFile():writs the text in the file.
* saveToFile():saves text in a file.

## 5.7 FileManager

This class is a circular doubly linked list that saves and allows access to files. This class implements various functions described briefly below.

* getOpenFileCount(): returns openFileCount.
* getFilesCreated(): returns filesCreated.
* isEmpty(): returns true if the list is empty.
* createNewFile(FileBtn te): create a New File named te.
* closeFile(FileBtn openFile): closes the File with which you are currently working.
* navigateToNextFile(FileBtn targetFile): search and opens the file targetFile.

## 5.8 Main Class

The main class is where everything takes place and where all methods are executed. This class has only two methods: the main and start.

# 6. Code

# Main

import java.io.File;

import java.io.FileNotFoundException;

import javafx.application.Application;

import javafx.geometry.Insets;

import javafx.geometry.Pos;

import javafx.stage.Modality;

import javafx.stage.Stage;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.ScrollPane;

import javafx.scene.control.TextArea;

import javafx.scene.control.TextField;

import javafx.scene.layout.GridPane;

import javafx.scene.layout.HBox;

import javafx.scene.layout.Priority;

import javafx.scene.layout.Region;

import javafx.scene.layout.VBox;

import javafx.scene.text.Font;

import javafx.scene.text.FontWeight;

import javafx.scene.text.Text;

public class Main extends Application {

FileManager manager = new FileManager();

TextArea mainTextArea = new TextArea();

FileBtn fileInFocus;

@Override

public void start(Stage primaryStage) {

//Global variables

mainTextArea.setFocusTraversable(false);

//Global Buttons and their pane

HBox glblBtnsPane = new HBox();

//filesPane.setPadding(new Insets(10,10,10,10));

glblBtnsPane.setSpacing(10);

Button openFile = new Button("Open");

openFile.setMinWidth(Region.USE\_PREF\_SIZE);

openFile.setFocusTraversable(false);

Button newFile = new Button("New");

newFile.setMinWidth(Region.USE\_PREF\_SIZE);

newFile.setFocusTraversable(false);

//files and their panes

HBox filesPane = new HBox();

filesPane.setSpacing(10);

filesPane.setPadding(new Insets(1,5,10,5));

ScrollPane scrollPane = new ScrollPane();

scrollPane.setContent(filesPane);

scrollPane.setFitToHeight (true);

scrollPane.setVbarPolicy (ScrollPane.ScrollBarPolicy.NEVER);

scrollPane.setHbarPolicy (ScrollPane.ScrollBarPolicy.ALWAYS);

glblBtnsPane.getChildren().addAll(openFile,newFile,scrollPane);

//File specific buttons and their pane

HBox buttonsPane = new HBox();

//buttonsPane.setPadding(new Insets(10,10,10,10));

buttonsPane.setSpacing(10);

ActionButton save = new ActionButton("Save");

save.setFocusTraversable(false);

ActionButton undo = new ActionButton("Undo");

undo.setFocusTraversable(false);

ActionButton redo = new ActionButton("Redo");

redo.setFocusTraversable(false);

ActionButton find = new ActionButton("Find and Replace");

find.setFocusTraversable(false);

ActionButton close = new ActionButton("Close");

close.setFocusTraversable(false);

buttonsPane.getChildren().addAll(save,undo,redo,find,close);

//Common elements

Stage popupStage = new Stage();

popupStage.initModality(Modality.APPLICATION\_MODAL);

popupStage.initOwner(primaryStage);

//Open popup

String openTitle = "Opening file";

Text openQuestion = new Text("Enter file name to open without the \".txt\" extension.");

TextField openField = new TextField();

Button doneOpen = new Button("Done");

doneOpen.setFocusTraversable(false);

VBox openPane = new VBox();

openPane.getChildren().addAll(openQuestion,openField,doneOpen);

openPane.setSpacing(10);

openPane.setPadding(new Insets(10,10,10,10));

openPane.setAlignment(Pos.CENTER);

Scene openScene = new Scene(openPane);

//Save popup

String saveTitle = "Saving file";

Text saveQuestion = new Text("Enter file name. Currently, it is ");

TextField saveField = new TextField();

saveField.setFocusTraversable(false);

Button doneSave = new Button("Done");

doneSave.setFocusTraversable(false);

VBox savePane = new VBox();

savePane.getChildren().addAll(saveQuestion, saveField, doneSave);

savePane.setSpacing(10);

savePane.setPadding(new Insets(10,10,10,10));

savePane.setAlignment(Pos.CENTER);

Scene saveScene = new Scene(savePane);

//Find and replace popup

String findTitle = "Find and Replace";

Text findText = new Text("Find:");

TextField findField = new TextField();

Text replaceText = new Text("Replace with:");

TextField replaceField = new TextField();

replaceField.setFocusTraversable(false);

Button findOcc = new Button("Find");

findOcc.setFocusTraversable(false);

Button replace = new Button("Replace");

replace.setFocusTraversable(false);

Button replaceAll = new Button("Replace All");

replaceAll.setFocusTraversable(false);

Button doneFind = new Button("Done");

doneFind.setFocusTraversable(false);

GridPane findPane = new GridPane();

findPane.setPadding(new Insets(10,10,10,10));

findPane.setVgap(10);

findPane.setHgap(10);

findPane.setAlignment(Pos.CENTER);

findPane.add(findText, 0, 0);

findPane.add(findField, 1, 0);

findPane.add(replaceText, 0, 1);

findPane.add(replaceField, 1, 1);

findPane.add(findOcc, 0, 2);

findPane.add(replace, 1, 2);

findPane.add(replaceAll, 1, 3);

findPane.add(doneFind, 0, 3);

Scene findScene = new Scene(findPane);

//Close popup

String closeTitle = "Closing file";

Text closeText = new Text("Do you want to save this file before closing?");

Button yes = new Button("Yes");

yes.setFocusTraversable(false);

Button no = new Button("No");

GridPane closePane = new GridPane();

closePane.setPadding(new Insets(10,10,10,10));

closePane.setVgap(10);

closePane.setHgap(10);

closePane.add(closeText, 0, 0, 2, 1);

closePane.add(yes, 0, 1);

closePane.add(no, 1, 1);

closePane.setAlignment(Pos.CENTER);

Scene closeScene = new Scene(closePane);

//Main pane, scene, and stage

double spacing = 5;

VBox mainPane = new VBox();

mainPane.setSpacing(spacing);

mainPane.setPadding(new Insets(10,10,10,10));

mainPane.getChildren().addAll(glblBtnsPane,buttonsPane,mainTextArea);

VBox.setVgrow(mainTextArea, Priority.ALWAYS);

Scene mainScene = new Scene(mainPane);

primaryStage.setScene(mainScene);

primaryStage.setTitle("Simple Notepad");

primaryStage.show();

//Event Handlers

mainTextArea.setOnMouseClicked(e ->{

mainTextArea.setEditable(!manager.isEmpty());

});

mainTextArea.textProperty().addListener(e ->{

fileInFocus.setFont(Font.font(Font.getDefault().getName(), FontWeight.BOLD, 12));

TextEditor currentEditor = fileInFocus.getTextEditor();

currentEditor.clear();

String[] strings = mainTextArea.getText().trim().split(" ");

for(String s: strings) {

currentEditor.appendText(s);

}

});

openFile.setOnAction(e ->{

//Create file button with temp name

String newFileName = "temp";

FileBtn fb = new FileBtn(newFileName);

manager.createNewFile(fb);

filesPane.getChildren().add(fb);

fb.setMinWidth(Region.USE\_PREF\_SIZE);

fb.setOnAction(e1 ->{

fileInFocus = fb;

mainTextArea.setText(fb.getTextEditor().getText());

});

fb.focusedProperty().addListener((e1, oldVal, newVal) ->{

if(newVal) {

fb.setStyle("-fx-background-color: lightgreen;");

fileInFocus = fb;

}

else {

fb.setStyle("-fx-background-color: lightgrey;");

}

});

fb.requestFocus();

//Get actual file name from user

popupStage.setScene(openScene);

popupStage.setTitle(openTitle);

popupStage.show();

doneOpen.setOnAction(e1 ->{

while(true) {

try {

File trialFile = new File(openField.getText() + ".txt");

if(!trialFile.exists() || openField.getText().contains(".txt")) {

throw new FileNotFoundException();

}

fb.setText(openField.getText());

fb.setSavedBefore(true);

TextEditor currentEditor = fb.getTextEditor();

currentEditor.setSaveFile(openField.getText());

currentEditor.readFromFile();

popupStage.close();

mainTextArea.setText(currentEditor.getText());

break;

}

catch(FileNotFoundException ex) {

openField.setText("File not found! Enter file name correctly!");

}

}

});

});

newFile.setOnAction(e ->{

String newFileName = "File " + manager.getFilesCreated();

FileBtn fb = new FileBtn(newFileName);

manager.createNewFile(fb);

filesPane.getChildren().add(fb);

fb.setMinWidth(Region.USE\_PREF\_SIZE);

fb.setOnAction(e1 ->{

fileInFocus = fb;

mainTextArea.setText(fb.getTextEditor().getText());

});

fb.focusedProperty().addListener((e1, oldVal, newVal) ->{

if(newVal) {

fb.setStyle("-fx-background-color: lightgreen;");

fileInFocus = fb;

}

else {

if(!mainTextArea.isFocused()) {

fb.setStyle("-fx-background-color: lightgrey;");

}

}

});

fb.requestFocus();

});

save.setOnAction(e ->{

if(!manager.isEmpty()) {

TextEditor te = fileInFocus.getTextEditor();

if(!fileInFocus.isSavedBefore() || fileInFocus.getText().contains("temp")) {

saveQuestion.setText(saveQuestion.getText() + fileInFocus.getText() + ".");

popupStage.setTitle(saveTitle);

popupStage.setScene(saveScene);

popupStage.show();

doneSave.setOnAction(e1 ->{

if(saveField.getText().isBlank()) {

String name = fileInFocus.getText().replaceAll(" ", "\_");

try {

te.setSaveFile(name);

te.saveToFile();

fileInFocus.setText(name);

} catch (FileNotFoundException ex) {

ex.printStackTrace();

}

}

else {

String name = saveField.getText().replaceAll(" ", "\_");

try {

te.setSaveFile(name);

te.saveToFile();

fileInFocus.setText(name);

} catch (FileNotFoundException ex) {

ex.printStackTrace();

}

}

popupStage.close();

});

fileInFocus.setSavedBefore(true);

}

else {

te.saveToFile();

}

fileInFocus.setFont(Font.font(Font.getDefault().getName(), FontWeight.NORMAL, 12));

}

});

undo.setOnAction(e ->{

if(!manager.isEmpty()) {

fileInFocus.getTextEditor().undo();

mainTextArea.setText(fileInFocus.getTextEditor().getText());

}

});

redo.setOnAction(e ->{

if(!manager.isEmpty()) {

fileInFocus.getTextEditor().redo();

mainTextArea.setText(fileInFocus.getTextEditor().getText());

}

});

find.setOnAction(e ->{

if(!manager.isEmpty()) {

TextEditor currentEditor = fileInFocus.getTextEditor();

popupStage.setTitle(findTitle);

popupStage.setScene(findScene);

popupStage.show();

findOcc.setOnAction(e1 ->{

String oldWord = findField.getText().trim();

if(!oldWord.isBlank()) {

int count = currentEditor.searchWord(findField.getText().trim());

findText.setText("Found " + count + " match(es)");

findText.setFont(Font.font(Font.getDefault().getName(), FontWeight.BOLD, 12));

}

});

replace.setOnAction(e1 ->{

String oldWord = findField.getText().trim();

String newWord = replaceField.getText().trim();

if(!oldWord.isBlank() && !newWord.isBlank()) {

currentEditor.replace(oldWord, newWord);

int count = currentEditor.searchWord(newWord);

replaceText.setText("Replaced " + count + " word(s)");

replaceText.setFont(Font.font(Font.getDefault().getName(), FontWeight.BOLD, 12));

}

mainTextArea.setText(fileInFocus.getTextEditor().getText());

});

replaceAll.setOnAction(e1 ->{

String oldWord = findField.getText().trim();

String newWord = replaceField.getText().trim();

if(!oldWord.isBlank() && !newWord.isBlank()) {

currentEditor.replaceAll(oldWord, newWord);

int count = currentEditor.searchWord(newWord);

replaceText.setText("Replaced " + count + " word(s)");

replaceText.setFont(Font.font(Font.getDefault().getName(), FontWeight.BOLD, 12));

}

mainTextArea.setText(fileInFocus.getTextEditor().getText());

});

doneFind.setOnAction(e1 ->{

findText.setText("Find:");

findText.setFont(Font.font(Font.getDefault().getName(), FontWeight.NORMAL, 12));

replaceText.setText("Replace with:");

replaceText.setFont(Font.font(Font.getDefault().getName(), FontWeight.NORMAL, 12));

popupStage.close();

});

}

});

close.setOnAction(e ->{

if(!manager.isEmpty()) {

FileBtn previous = fileInFocus;

popupStage.setScene(closeScene);

popupStage.setTitle(closeTitle);

popupStage.show();

yes.setOnAction(e1 ->{

if(fileInFocus.isSavedBefore()) {

fileInFocus.getTextEditor().saveToFile();

filesPane.getChildren().remove(fileInFocus);

fileInFocus = manager.navigateToNextFile(previous);

if(fileInFocus != null) {

mainTextArea.setText(fileInFocus.getTextEditor().getText());

}

else {

mainTextArea.setText("");

}

manager.closeFile(previous);

popupStage.close();

}

else {

//fileInFocus.getTextEditor().setSaveFile(fileInFocus.getText().replaceAll(" ", "\_"));

saveQuestion.setText(saveQuestion.getText() + fileInFocus.getText() + ".");

popupStage.setTitle(saveTitle);

popupStage.setScene(saveScene);

TextEditor te = fileInFocus.getTextEditor();

doneSave.setOnAction(e2 ->{

if(saveField.getText().isBlank()) {

String name = fileInFocus.getText().replaceAll(" ", "\_");

try {

te.setSaveFile(name);

te.saveToFile();

fileInFocus.setText(name);

} catch (FileNotFoundException ex) {

ex.printStackTrace();

}

}

else {

String name = saveField.getText().replaceAll(" ", "\_");

try {

te.setSaveFile(name);

te.saveToFile();

fileInFocus.setText(name);

} catch (FileNotFoundException ex) {

ex.printStackTrace();

}

}

filesPane.getChildren().remove(fileInFocus);

fileInFocus = manager.navigateToNextFile(previous);

if(fileInFocus != null) {

mainTextArea.setText(fileInFocus.getTextEditor().getText());

}

else {

mainTextArea.setText("");

}

manager.closeFile(previous);

popupStage.close();

});

}

});

no.setOnAction(e1 ->{

filesPane.getChildren().remove(fileInFocus);

fileInFocus = manager.navigateToNextFile(previous);

if(fileInFocus != null) {

mainTextArea.setText(fileInFocus.getTextEditor().getText());

}

else {

mainTextArea.setText("");

}

manager.closeFile(previous);

popupStage.close();

});

}

});

}

public static void main(String[] args) {

launch(args);

}

}

# TextEditor

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.RandomAccessFile;

public class TextEditor extends LinkedStrings{

private LinkedStrings removed;

private BinarySearchTree bst;

private RandomAccessFile saveFile;

private String saveFileName;

public TextEditor() {

super();

removed = new LinkedStrings();

bst = new BinarySearchTree();

saveFile = null;

saveFileName = "";

}

public void clear() {

super.clear();

bst.clear();

}

@Override

public String undo() {

String removedString = super.undo();

if(!removedString.equals("")) {

removed.appendText(removedString);

bst.remove(removedString);

}

return removedString;

}

@Override

public void appendText(String str) {

super.appendText(str);

bst.insert(str);

}

public void redo() {

if(removed.isEmpty()) {

System.out.println("You didn't undo any text!");

}

else {

String removedString = removed.undo();

appendText(removedString);

bst.insert(removedString);

}

}

public int searchWord(String str) {

if(isEmpty()) {

return 0;

}

else {

return bst.searchTree(str);

}

}

@Override

public void replace(String oldWord, String newWord) {

super.replace(oldWord, newWord);

bst.replace(oldWord, newWord);

}

@Override

public void replaceAll(String oldWord, String newWord) {

super.replaceAll(oldWord, newWord);

int count = bst.searchTree(oldWord);

while(count-- > 0) {

bst.replace(oldWord, newWord);

}

}

public void setSaveFile(String fileName) throws FileNotFoundException{

try {

saveFileName = fileName + ".txt";

saveFile = new RandomAccessFile(saveFileName,"rw");

} catch (FileNotFoundException e) {

System.out.println(e.getMessage());

throw new FileNotFoundException("File does not exist!");

}

}

public String getSaveFileName() {

return saveFileName;

}

public void readFromFile() {

try {

double fileLength = saveFile.length();

while(saveFile.getFilePointer() != fileLength) {

appendText(saveFile.readUTF());

}

} catch (IOException e) {

System.out.println(e.getMessage());

}

}

public void saveToFile() {

if(!isEmpty()) {

try {

String textToSave = getText().trim();

String[] stringsToSave = textToSave.split(" ");

saveFile.setLength(0);

for(String s: stringsToSave) {

saveFile.writeUTF(s);

}

}

catch(IOException e) {

System.out.println(e.getMessage());

}

}

}

}

# Node

public class Node {

private String text;

private Node next;

public Node(String t) {

text = t;

next = null;

}

public String getText() {

return text;

}

public void setString(String t) {

text = t;

}

public Node getNext() {

return next;

}

public void setNext(Node n) {

next = n;

}

}

LinkedStrings

public class LinkedStrings {

private Node head;

private Node tail;

public LinkedStrings() {

head = null;

tail = null;

}

public boolean isEmpty() {

return head == null;

}

public void clear() {

head = null;

tail = null;

}

public void appendText(String str) {

Node p = new Node(str);

if(isEmpty()) {

head = p;

tail = p;

}

else {

tail.setNext(p);

tail = p;

tail.setNext(null);

}

}

public String undo() {

if(isEmpty()) {

System.out.println("There is no text to undo!");

return "";

}

else if(head == tail) {

String removedString = head.getText();

head = null;

tail = null;

return removedString;

}

else {

Node current = head;

while(current.getNext() != tail) {

current = current.getNext();

}

String removedString = tail.getText();

current.setNext(null);

tail = current;

return removedString;

}

}

public void replace(String oldWord, String newWord) {

if(!isEmpty()) {

Node current = head;

while(current != null) {

if(current.getText().equals(oldWord)) {

current.setString(newWord);

break;

}

current = current.getNext();

}

}

}

public void replaceAll(String oldWord, String newWord) {

if(!isEmpty()) {

Node current = head;

while(current != null) {

if(current.getText().equals(oldWord)) {

current.setString(newWord);

}

current = current.getNext();

}

}

}

public String getText() {

if(isEmpty()) {

return "";

}

else {

Node current = head;

String text = "";

while(current != null) {

text += current.getText() + " ";

current = current.getNext();

}

return text.trim();

}

}

}

# FileNode

public class FileNode {

private FileBtn file;

private FileNode next;

private FileNode prev;

public FileNode(FileBtn btn) {

file = btn;

next = null;

prev = null;

}

public FileBtn getFile() {

return file;

}

public void setFile(FileBtn file) {

this.file = file;

}

public FileNode getNext() {

return next;

}

public void setNext(FileNode next) {

this.next = next;

}

public FileNode getPrev() {

return prev;

}

public void setPrev(FileNode prev) {

this.prev = prev;

}

}

# FileManager

import java.util.Locale;

import java.util.Scanner;

public class FileManager {

private FileNode head;

private FileNode tail;

private int openFileCount;

private static int filesCreated = 0;

public FileManager() {

head = null;

tail = null;

openFileCount = 0;

}

public int getOpenFileCount() {

return openFileCount;

}

public int getFilesCreated() {

return filesCreated;

}

public boolean isEmpty() {

return head == null;

}

public void createNewFile(FileBtn te) {

FileNode fn = new FileNode(te);

if(isEmpty()) {

head = fn;

tail = fn;

head.setNext(tail);

head.setPrev(tail);

tail.setPrev(head);

tail.setNext(head);

}

else{

tail.setNext(fn);

fn.setPrev(tail);

tail = fn;

tail.setNext(head);

}

openFileCount++;

filesCreated++;

}

public void closeFile(FileBtn openFile) {

if(head.getFile() == openFile) {

head = head.getNext();

head.setPrev(tail);

}

else if(tail.getFile() == openFile) {

tail = tail.getPrev();

tail.setNext(head);

}

else {

FileNode current = head;

while(current != tail) {

if(current.getFile() == openFile) {

current.getPrev().setNext(current.getNext());

current.getNext().setPrev(current.getPrev());

break;

}

current = current.getNext();

}

}

openFileCount--;

}

public FileBtn navigateToNextFile(FileBtn targetFile) {

if(!isEmpty()) {

FileNode current = head;

while(current != tail) {

if(current.getFile() == targetFile) {

return current.getNext().getFile();

}

current = current.getNext();

}

if(tail.getFile() == targetFile) {

return head.getFile();

}

return null;

}

return null;

}}

# BstNode

public class BstNode {

private BstNode left, right;

private int count;

private String str;

public BstNode() {

left = null;

right = null;

count = 0;

str = "";

}

public BstNode(String str) {

this.left = null;

this.right = null;

this.count = 1;

this.str = str;

}

public BstNode getLeft() {

return left;

}

public void setLeft(BstNode left) {

this.left = left;

}

public BstNode getRight() {

return right;

}

public void setRight(BstNode right) {

this.right = right;

}

public int getCount() {

return count;

}

public void incrementCount() {

this.count++;

}

public void decrementCount() {

this.count--;

}

public String getStr() {

return str;

}

public void setStr(String str) {

this.str = str;

}

}

# BinarySearchTree

public class BinarySearchTree {

private BstNode root;

public boolean isEmpty() {

return root == null;

}

public void clear() {

root = null;

}

public void insert(String k) {

BstNode p = new BstNode(k);

if(isEmpty()) {

root = p;

}

else {

insertIntoBinaryTree(k, root);

}

}

private void insertIntoBinaryTree(String k, BstNode node) {

BstNode parent = node;

if(k.compareTo(node.getStr()) < 0) {

node = node.getLeft();

if(node == null) {

parent.setLeft(new BstNode(k));

}

else {

insertIntoBinaryTree(k, node);

}

}

else if(k.compareTo(node.getStr()) > 0) {

node = node.getRight();

if(node == null) {

parent.setRight(new BstNode(k));

}

else {

insertIntoBinaryTree(k, node);

}

}

else {

node.incrementCount();

}

}

public void remove(String k) {

if(!isEmpty()) {

remove(k,root);

}

}

private void remove(String k, BstNode node) {

if(node != null) {

if(k.equals(node.getStr())) {

node.decrementCount();

}

else if(k.compareTo(node.getStr()) < 0) {

remove(k,node.getLeft());

}

else {

remove(k,node.getRight());

}

}

}

public void replace(String oldWord, String newWord) {

insert(newWord);

remove(oldWord);

}

public int searchTree(String str) {

if(isEmpty()) {

return 0;

}

else {

return searchTree(root, str);

}

}

private int searchTree(BstNode node, String str) {

if(node.getStr().equals(str)) {

return node.getCount();

}

else if(str.compareTo(node.getStr()) < 0) {

node = node.getLeft();

if(node == null) {

return 0;

}

else {

return searchTree(node, str);

}

}

else {

node = node.getRight();

if(node == null) {

return 0;

}

else {

return searchTree(node, str);

}

}

}

}

# FileBtn

import javafx.scene.control.Button;

public class FileBtn extends Button{

private TextEditor te;

private boolean savedBefore;

public FileBtn(String name) {

super(name);

te = new TextEditor();

savedBefore = false;

}

public TextEditor getTextEditor() {

return te;

}

public boolean isSavedBefore() {

return savedBefore;

}

public void setSavedBefore(boolean savedBefore) {

this.savedBefore = savedBefore;

}

}

# ActionButton

import javafx.scene.control.Button;

public class ActionButton extends Button{

private TextEditor te;

public ActionButton(String name){

super(name);

}

public boolean saveFile(TextEditor te) {

this.te = te;

if(te.getSaveFileName().isBlank()) {

return false;

}

else {

te.saveToFile();

return true;

}

}

public TextEditor closeFile() {

return te;

}

public String undo(TextEditor te) {

this.te = te;

te.undo();

return te.getText();

}

public String redo(TextEditor te) {

this.te = te;

te.redo();

return te.getText();

}

}

# 7. Output in Console

# 8. Conclusion

Text editor system is a that helps the user with writing text and nots . Its main goal is to facilitate this process and make it more secure and covenant. To implement a perfectly working system, we used several data structures in Java. To store each word and sentence we used a linked list of data and stack. To search for a word, we used a binary search tree.to store them all in files we used a circular doubly linked list . This project made us understand how to implement data structures better and how to use programming to facilitate certain processes in our lives.

## 9. References

## For reference, we used CMPS347 Data Structures course and CMPS242 OOP course.