EGE ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

NESNEYE DAYALI PROGRAMLAMA

2022-2023 GÜZ YARIYILI

PROJE ÖDEVİ

TESLİM TARİHİ

06/01/2023

HAZIRLAYANLAR

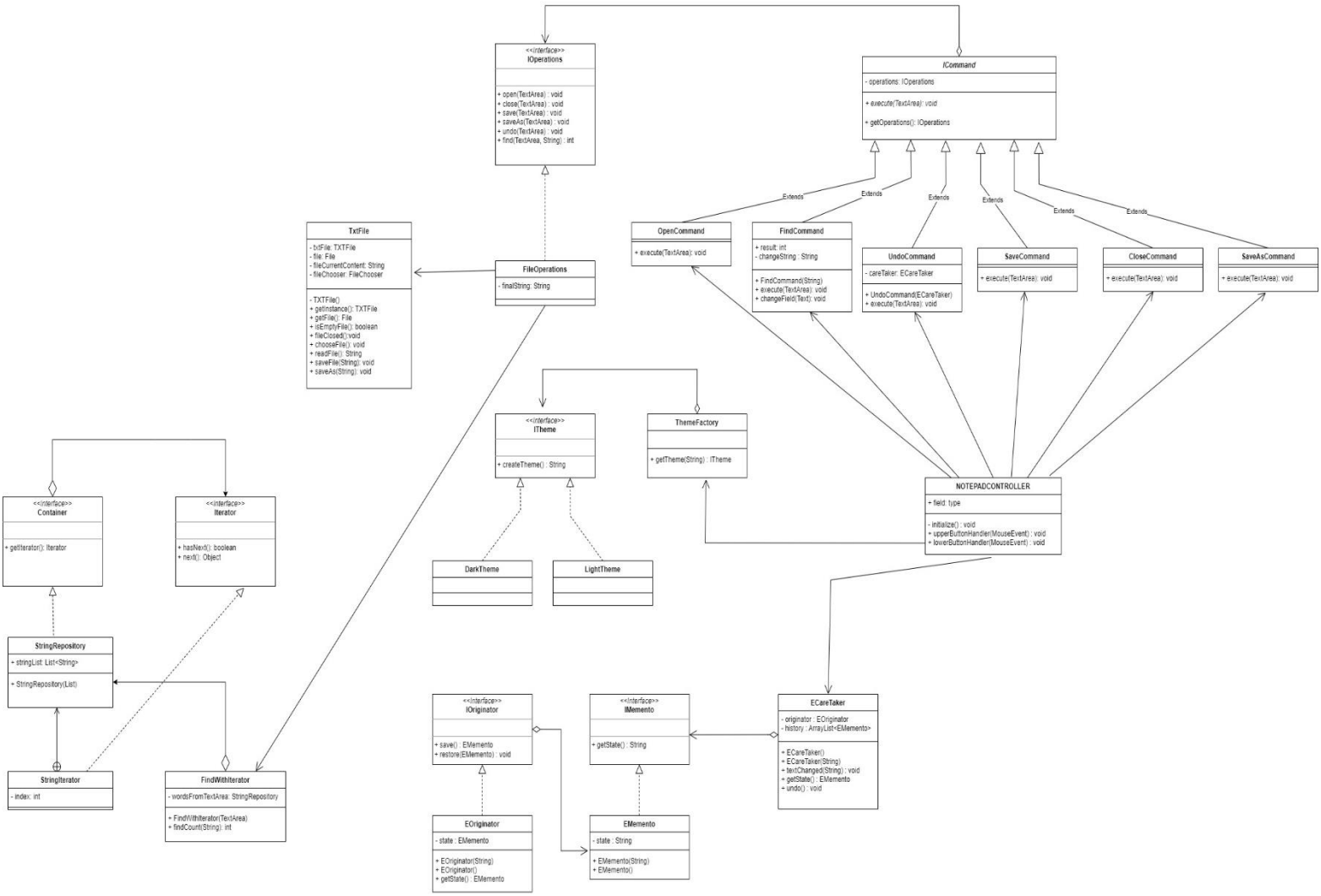
Zeynep Cerrahoğlu - 05190000033

Mahmut Çelik – 05190000114

Özgür Bayraşa – 05190000027

İçindekiler

UML Diyagramı



Design Pattern’lar

Memento

Memento Pattern’i undo işlemini sağlamak için kullanılmıştır. Bu pattern için EOriginator, EMemento ve ECareTaker sınıflarına ihtiyacımız vardır. Aynı zamanda IOriginator ve IMemento arayüzleri de kullanılır. Böylece EOriginator ve EMemento sınıflarının tanımlaması gereken metotlar belirlenmiş olur.

Undo işleminde sadece metinsel bir undo işlemi yapmak istediğimiz için EMemento sınıfımızda metinsel veri tutuldu. EMemento sınıfımız aşağıda görünmektedir.

```
package com.example.projectcode.MEMENTO;
/* IMemento interface'ine sahip olan sınıf String türünde state döndüren getState() metodunu bulundurmalıdır.*/
public interface IMemento{
   String getState();

}

package com.example.projectcode.MEMENTO;
/* EMemento sınıfı IMemento interface'ini implemente eder.*/
/* String türünde state tutar. CareTaker history'sinde saklanacak değerler string olacaktır. */
/* Çünkü undo işlemini metinsel anlamda yapmak amaçlanmaktadır. */
public class EMemento implements IMemento {
   /* Metin türünde değerleri undo etmek istediğimiz için state String türündedir.*/
   private String state;
   public EMemento(String state) {
      this.state = state;
   }
   /* String gelmeden EMemento oluşursa boş string state'e atanır. */
   public EMemento() {
      this.state = "";
   }
   /* Memento'daki string state döndürülür.*/
   @Override
   public String getState() {
      return state;
   }
}
```

Bu EMemento sınıfını kullanarak yeni Ememento lar oluşturup, statelerini EMemento sınıfı türünden tutulmasını sağlayan EOriginator tanımlanmalıdır. Bu sınıfta save ile anlık Ememento döndürülebilirken restore ile EOriginator’ın state’i (EMemento cinsindeki) değiştirlebilmektedir.

```java
package com.example.projectcode.MEMENTO;
/* IOriginator interface'ine sahip olacak olan sınıflar, EMemento döndüren save metoduyla, değer döndürmeyen ancak
*  EMemento değeri alan bir restore metodu bulundurmaldır.*/
public interface IOriginator {
    EMemento save();
    void restore(EMemento memento);
}
package com.example.projectcode.MEMENTO;
/* Originator, IOriginator interface'ini implemente eder, save ve restore metodlarını bulundurmalıdır. */
/* EMemento sınıfına restore ile veri kaydeder. Aynı zamanda EMemento sınıfına ait veri tutar. Kayıtlar */
/* bu sınıf üzerinden geri döndürülebilir veya EMemento değiştirilebilir. */
public class EOriginator implements IOriginator {
    /* EOriginator üzerinde EMemonte türünde stateler vardır.*/
    private EMemento state;
    /* Constructorlar basit olarak tanımlanmıştır. String değer varsa o değeri alan EMemento, yoksa parametresiz
    * constructor ile oluşan EMemento oluşturulur ve bu sınıfın field'ına tanımlanır.*/
    public EOriginator(String state) {
        this.state = new EMemento(state);
    }
    public EOriginator(){
        this.state = new EMemento();
    }

    /* getState ile EMemento türündeki state döndürülür.*/
    public EMemento getState() {
        return this.state;
    }
    /* Save metoduyla EMemento yine benzer şekilde EMemento türündeki state döndürülür.
    * getState'e göre daha anlamlı bir isimdir ve CareTaker kullanmaktadır.*/
    @Override
    public EMemento save() {
        return state;
    }

    /* Restore ile Yeni bir EMemento EOriginator field'ında tanımlanır.*/
    @Override
    public void restore(EMemento memento) {
        this.state = memento;
    }
}
```

ECareTaker ile EMemento listesi tutar, aynı zamanda koşullara göre bu listeye EMemento ekler ya da EOriginator'un statelerini değiştirebilir. Bizim programımızda kullanıcı her karakter girdiğinde state değişir, aynı zamnda undo işlemi de bu EOriginator içinde yapılır.

```java
package com.example.projectcode.MEMENTO;

import java.util.ArrayList;
/* EOriginator üzerinden ne hangi durumlarda kayıt yapılacağını, undo işleminin nasıl olacağını ECareTaker üzerinden
* görebiliriz. EMemento türünden elemanlar tutan bir liste ve bir originator sınıfından türemiş nesne tutulur.
* ECareTaker, kendisine gelen metine göre state'i history'e yani EMemento listesine kaydeder. Daha sonra ihtiyaç
* duyulursa undo ile bir önceki state döndürülür, ve en son state silinir.*/
public class ECareTaker {
    private EOriginator originator;
```

```
    /* history EMemento türünde bir ArrayList'tir bununla birlikte karakter karakter history tutabileceğiz.*/
    private ArrayList<EMemento> history = new ArrayList<>();
    public ECareTaker() {
        originator = new EOriginator();
    }
    public ECareTaker(String text) {
        originator = new EOriginator(text);
    }

    /* Metin null ise yeni state "", değilse gelen metin yeni state, originatorda tutulan eski state ile
    * yeni state farklı ise, yeni state memento listesine eklenir ve originator state'i yeni state ile güncellenir.*/
    public void textChanged(String text) {

        EMemento newState;
        if(text == null) {
            newState = new EMemento();
        }else {
            newState = new EMemento(text);
        }
        /* İlgili string değeri (Her karakter değiştiğinde buraya bir string gönderilir.) Mementoda tutulan
         * string'den farklı ise yeni state originator'da güncellenmelidir, eski state history'ye eklenmelidir.*/
        /* Old state Originator'un save metoduyla alındı.*/
        EMemento oldState = originator.save();
        if(oldState.getState().equals(newState.getState())){
            return;
        }

        history.add(oldState);
        /* restore metoduyle originatordaki Memento değeri yeniden atanır. Yeni state oraya atandı.*/
        originator.restore(newState);
    }
    public EMemento getState() {return this.originator.getState();}
    /* History'de zaten geçmiş stateler tutulmaktadır. Undo işlemi yapılmak istensin, ekranda AAB yazsın. Bu
durumda
    * AAB origanitorda tutulurken history'deki son eleman ise AA olacak. Dolayısıyla historydeki son eleman
    * originator'a atandı (Originator bir adım gerideki değeri aldı.) Sonrasında ise histary'den AA değeri silindi.
    * AA'dan önce ne yazılmışsa (A) artık history'nin son elemanı o oldu. */
    public void undo() {
        int size = history.size() ;
        if(size == 0) {
            return ;
        }
        originator.restore(history.get(size - 1));
        history.remove(size - 1);
    }
}
```

Kullanıcın her karakter girdiğinde, ilgili metnin ECareTaker'a gönderilmesi;

// NOTEPADCONTROLLER

```
/* Kullanıcı her karakter girdiğinde, girilen değer careTaker'a gönderilsin.
* Mesela ekranda A yazıyorsa CareTaker('A') olur. */
@FXML
private void initialize() {
    textArea.textProperty().addListener((observableValue, oldValue, newValue) ->
careTaker.textChanged(newValue));
}
```

Command Pattern ile tanımlanan undo operasyonu ile ECareTaker üzerinde undo işlemi yapılır. EOriginator State'i bir önceki State olur. Sonrasında bu değer text alanına bastırılır.

// File Operations

```
public void undo(TextArea textArea, ECareTaker careTaker) {
    /* careTaker.undo() ile originator bir önceki state'e set edildi.*/
    careTaker.undo();
    /* Sonrasında textArea'ya originator'da tutulan state (bir önceki state) bastırıldı.*/
    /* İlk getstate EMemento döndürür. Sonraki getState String döndürür (Memento.getState())*/
    textArea.setText(careTaker.getState().getState());
}
```

Factory Pattern

Factory Pattern'i NotePad uygulamamızda, Dark Tema ve Light Tema yapısı için kullandık. Programın altındaki tema buttonlarına tıklandığında factory pattern da ilgili theme oluşturuluyor ve scene in teması değişmiş oluyor.

//THEMEFACTORY CLASS

```
package com.example.projectcode.FACTORY;

/* Tıklanılan butonun ID'sine göre yeni bir tema class'ı döndürülür. Böylece Factory patterni sağlanır.
* NotePad üzerinde bulunan Dark ve Light butonlarına tıklandığında ilgili butonun ID'sine göre tema
* değişir, o temayla ilgili class döndürülür.  */

public class ThemeFactory {
    /* Buton ID'si lightThemeButton ise veya light kelimesini içeriyorsa light tema döndür. Yoksa dark tema
döndür.*/
    public static ITheme getTheme(String clickedButtonId){
        if(clickedButtonId.equals("lightThemeButton") || clickedButtonId.contains("light")){
            return new LightTheme();
        }
        else {
            return new DarkTheme();
        }

    }
}
```

//ITHEME INTERFACE

```
package com.example.projectcode.FACTORY;

/* ITheme interface'ine sahip sınıflar createTheme() metodunu kullanmak zorundadır. Bununla DarkTheme ve
LightTheme
 * oluşturulması sağlanacaktır.*/
public interface ITheme {
    String createTheme();
```

```
}
```

//DARK THEME

```
package com.example.projectcode.FACTORY;

/* ITheme interface'ini implemenete eden DarkTheme, Koyu Tema oluşturur. Koyu tema için oluşturulan CSS
* dosyasının kaynağı ile arayüz değiştirilir. */
public class DarkTheme implements ITheme{
    /* Koyu Tema için oluşturulan CSS kayak dosyasıyle arayüz değiştirildi.*/
    @Override
    public String createTheme() {
        return getClass().getResource("/com/example/projectcode/CSS/darkThemeCss.css").toExternalForm();
    }
}
```

//LIGHT THEME

```
package com.example.projectcode.FACTORY;

/* ITheme interface'ini implemenete eden LightTheme, Açık Tema oluşturur. Açık tema için oluşturulan CSS
 * dosyasının kaynağı ile arayüz değiştirilir. */
public class LightTheme implements ITheme{
    /* Açık Tema için oluşturulan CSS kayak dosyasıyle arayüz değiştirildi.*/
    @Override
    public String createTheme() {
        return getClass().getResource("/com/example/projectcode/CSS/lightThemeCss.css").toExternalForm();
    }
}
```

Iterator Pattern

Iterator pattern yapısını kullanmamızdaki amaç textArea ya girilen veya dosya açılması sonucu oluşan textArea daki veriler de kullanıcıdan aldığımız word u aramayı sağlar. Kullanmak için kullanıcıdan input aldığımız bir textfield bulunmakta. Regular expression kullanarak textArea nın içindeki kelimeleri split ile bölüyoruz.

//Container INTERFACE

```
package com.example.projectcode.ITERATOR;
/* Container interface'ini alan sınıflar getIterator metodunu kullanmalıdır.*/
public interface Container {
    Iterator getIterator();
}
```

//ITERATOR INTERFACE

```
package com.example.projectcode.ITERATOR;
/* Iterator interface'ini alan sınıflar hasNext ve next metotlarını kullanmalıdır.*/
public interface Iterator {
    boolean hasNext();
```

```
    Object next();
}
```

//FINDWITHITERATOR CLASS

```
package com.example.projectcode.ITERATOR;

import javafx.scene.control.TextArea;

import java.util.Arrays;

/* Bu sınıfta metinler otomatik olarak constructor'da regular expression ile tanımlanan şekilde ayrılır. (Iterator
* kelime kelime olacak.) */
public class FindWithIterator {
    private StringRepository wordsFromTextArea;


    /* Regular Expression ile split sonucunda array oluşur. String Repository consturctorı için Liste yapısı gerekli
    * olduğu için ilgili Array, List yapısında String Repository'ye gönderilir. Bu noktada
    * Array'de tanımlnmış olan asList metodu kullanılmıştır.*/
    public FindWithIterator(TextArea textArea) {
        this.wordsFromTextArea = new
StringRepository(Arrays.asList(textArea.getText().split("[!'^+%&/()=?_£#$½{}|*-+@`,;.:¨<> \n\t]+")));

    }

    /* String olarak gelen bir kelime, iterator yardımıyla kelimelerin regular expression ile bölünmüş listesi kelime
    kelime dolaşılır. İlgili iterator'un sonraki değeri olduğu sürece çalışan for döngüsünde, aranan kelime
yakalandığı
    * zaman count değeri arttırılır. Böylece eşleşen kelime sayısı bulunur.*/
    public int findCount(String word){
        int count=0 ;
        for(Iterator iterator = wordsFromTextArea.getIterator(); iterator.hasNext();){
            if(iterator.next().equals(word)) count++;
        }
        return count;
    }
}
```

//STRINGREPOSITORY CLASS

```
package com.example.projectcode.ITERATOR;

import java.util.List;
/* String Repository sınıfı Container interface'ini implemente eder. Dolayısıyla getItertor kullanılmalıdır.*/
public class StringRepository implements Container{
    /* String değerler alan liste tutulur. */
    List<String> stringList;

    /* String değerler tutulan liste constructor ile sınıfta tanımlanır.*/
    public StringRepository(List<String> stringList) {
        this.stringList = stringList;
    }

    /* String Iterator döndürülür. Bu bir inner sınıftır. Buna ulaşmak için bu metot önemlidir.*/
    @Override
```

```
    public Iterator getIterator() {
        return new StringIterator();
    }


    // Inner Sınıf String Iterator, Iterator interface'ini implemente eder.
    private class StringIterator implements Iterator {
        /* index değeri tutulmuştur.*/
        private int index;

        /* Iterator için, bir sonraki değer var mı? Kontrol edilir. Index size ile eşit olduğunda
        * sonraki değerin olmadığı anlaşılır. */
        @Override
        public boolean hasNext() {
            if(index < stringList.size()){
                return true;
            }
            return false;
        }

        /* Next ile iterator bir sonraki değere gider, index ile sonraki değere ulaşılır sonra
        * index değer bir arttırılır. Bu işlem için bir sonraki elemanın olması koşulu (hasNext) eklenemiştir. Aksi
durumda
        * null değer döner.*/
        @Override
        public Object next() {
            if(this.hasNext()){
                return stringList.get(index++);
            }
            return null;
        }
    }

}
```

Singleton Pattern

Projede file işlemleri yaptığımız için ve aynı anda tek dosya üzerinde işlem yapmak istediğimiz için singleton kullanmamız gerekti. İlgili yapıyı kurduktan sonra ve ilgili metodları ekleyince file işlemleri tek bir class içerisinde yapılması sağlandı.

//TXTFile CLASS

```
package com.example.projectcode.SINGLETON;

import com.example.projectcode.Demo;
import javafx.scene.control.Alert;
import javafx.scene.control.DialogEvent;
import javafx.stage.FileChooser;

import java.io.*;

/* Bu kısım Singleton örneği oluşturur. Tek bir file varlığı bütün işlemler için yeterli olur.Yapısal olarak file
işlemleri
```

```
* sadece 1 file üzerinden olması gerektiği için en uygun creational yapının singleton oldugunu dusunduk*/
public class TXTFile {
    /* SINGLETON yapısını sağlamak için TXTFile field ı oluşturmalıyız. Singleton yapısında constructor
private olması
    * gerektiği için ve bu yapının singleton olması gerektiğinden txtFile field ı getInstance() metdonda
tanımlanır*/
    private static TXTFile txtFile;
    private static File file; //Seçtiğimiz dosya File olarak saklanır
    private static String fileCurrentContent; //dosyanın anlık içeriğini saklar kontrol kısımlarında kullanıldı


    /* FileChooser yapısı ilgili button a tıklandığında GUI kısmında eklenti seçme yapısının açılmasını sağlar*/
    private static final FileChooser fileChooser = new FileChooser();

    /* Dosya seçiciye all file veya .txt uzantılı yapılar seçeneklerini ekler.*/
    static{
        fileChooser.getExtensionFilters().addAll(
                new FileChooser.ExtensionFilter("Text","*.txt"),
                new FileChooser.ExtensionFilter("All files","*.*")
        );
    }


    //SINGLETON da private constructor olması gerekli
    private TXTFile(){

    }

    /* TXT dosyası null ise yeni bir TXTfile üretilir. null değilse önceki TXTFile instance i dondurulur*/
    public static TXTFile getInstance(){
        if(txtFile == null){
            txtFile = new TXTFile();
        }

        return txtFile;

    }

    /* File objesi null ise yeni file seçtirilir. Varsa o dosya döndürülür.*/
    public File getFile() {
        if(file == null){
            chooseFile();
        }
        return file;
    }


    /* İlgili bosya boşsa true, değilse false döner.*/
    public boolean isEmptyFile(){
        if(file == null){
            return true;
        }
        return false;
    }

    /* Dosya objesi null olarak atanır. Dosya artık kapanmıştır. (Dosya kapatıldıktan sonra kullanıldı)*/
    public void fileClosed(){
        file = null;
    }
```

```
    public void chooseFile(){
            /* Dosya seçme işlemi yapılır. Seçilen dosya file'a atanır. */
            file = null;
            File selectedFile = fileChooser.showOpenDialog(Demo.globalStage);
            file = selectedFile;
            /* KULLANICI HERHANGİ BİR DOSYA SEÇMEMİŞSE HATA GÖSTERİLİR. YA DA SECILEN
DOSYA UZANTISI IZIN VERILENLERDEN DEGILSE */
            if(selectedFile == null){
                Alert alert = new Alert(Alert.AlertType.ERROR);
                alert.show();
                alert.setTitle("SELECT ONLY TXT FILES");
                alert.setContentText("SELECT ONLY TXT FILES!");
            }

    }
    /* Dosya okuma işlemi yapılır.*/
    public String readFile(){
        /* belgeİçeriği için sınıf tanımlanır.*/
        StringBuilder documentContent =new StringBuilder();

        /* Eğer dosya varsa...*/
        if(file !=null){
            /* Dosya okunmaya çalışılır.*/
            try {
                FileReader fileReader = new FileReader(file);


                int value ;
                /* Dosyanın sonuna gelene kadar okuma yapılır. Okunan değerler belgeİçeriği değişkenine eklenir.*/
                while((value = fileReader.read() )!= -1){
                    documentContent.append((char)value);    //file içerisinde bir variable a da atanabilir
                }
                /* Dosyanın içeriği, belgeİçeriği'nin dönüştürülmesi ile elde edilir. */
                fileCurrentContent = String.valueOf(documentContent.toString().toCharArray());

                /* EXCEPTION durumunda bu blok çalışır.*/
            } catch (IOException |NullPointerException e){
                e.printStackTrace();
            }


        }
        /* Dosya yoksa boş bir String döner.*/
        return documentContent.toString();
    }

    /* Save File ile notePad'daki içerik alındı.*/
    public void saveFile(String content){
        /* Dosya varsa, ve içeriği mevcut içeriğe eşit değilse işlemler devam eder.*/
        if(file !=null && ! fileCurrentContent.equals(content)){
            /* Dosyaya içerikler yazılır. EXCEPTION kontrol edilir.*/
            try {
                FileWriter writer = new FileWriter(getFile());
                writer.flush();
                writer.write(content);
                writer.close();

            } catch (IOException e) {
                throw new RuntimeException(e);
```

```
        }
    }
}

/* Dosyanın içeriği textArea'dan gelir.  */
public void saveAs(String content){
    try {
        /* Dosya seçme ekranı gösterilir.*/
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("SAVE FILE");

        /* Sadece metin belgelerine izin verilir. */
        fileChooser.getExtensionFilters().addAll(
                new FileChooser.ExtensionFilter("Text Documents","*.txt*"));

        /* Dosya seçilir. */
        file = fileChooser.showSaveDialog(Demo.globalStage);

        /* İlgili dosya .txt içermiyorsa EXCEPTION fırlatılır.*/
        if(!file.getName().contains(".txt")){
            throw new NullPointerException();
        }
        /* Dosyaya içerik kaydedilir. */

        FileWriter writer = new FileWriter(file);
        writer.flush();
        writer.write(content);
        writer.close();

        System.out.println(file);

        /* EXCEPTION durumunda hata mesajı bastırılır.*/
    }catch (NullPointerException | IOException e){
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.show();
        alert.setTitle("SELECT ONLY TXT FILES");
        alert.setContentText("SELECT ONLY TXT FILES!");
    }
}



}
```

COMMAND Pattern

File ile ilgili işlem yapacak button ların yapacağı işlemler ve button lara tıklama yapıldığında ilgili command class ından ilgili işlem çağırılır ve gerçekleştirilmiş olur.

//CLOSECOMMAND CLASS

```java
package com.example.projectcode.COMMAND;


import javafx.scene.control.TextArea;
/* CloseCommand, ICommand interface'ini implemente eder. Dolayısıyla execute metodu olmalıdır.*/
public class CloseCommand extends ICommand{
    @Override
    public void execute(TextArea textArea){
        super.getOperations().close(textArea);
    }
}
```

//FILEOPARTIONS CLASS

```java
package com.example.projectcode.COMMAND;

import com.example.projectcode.CONTROLLER.NOTEPADCONTROLLER;
import com.example.projectcode.Demo;
import com.example.projectcode.FACTORY.ThemeFactory;
import com.example.projectcode.ITERATOR.FindWithIterator;
import com.example.projectcode.MEMENTO.ECareTaker;
import com.example.projectcode.SINGLETON.TXTFile;
import javafx.scene.control.TextArea;

/* Metotlar nasıl çalışmakta, buna buradan bakabiliriz. IOperations implemente edildiğinden
* ilgili metotlar tanımlanmalıdır.*/
public class FileOperations implements IOperations{
    /* open metodu Textarea alır. */
    /* Openda seçilen txt dosyasının içeriği okunur ve textArea'ya eklenir.*/
    @Override
    public void open(TextArea textArea) {
        TXTFile.getInstance().chooseFile();
        textArea.setText(TXTFile.getInstance().readFile());
    }

    /* close metodu TextArea alır. Eğer dosya seçilmemişse save as kullanılarak kaydedilecektir çünkü elimizde
path bilgisi yok. Değilse
    * dosyanın içeriği kaydedilip sonra da kapatılacaktır. En son olarak metin içeriği null olarak atanır.*/
    @Override
    public void close(TextArea textArea) {
        if(TXTFile.getInstance().isEmptyFile()) saveAs(textArea);
        else save(textArea);
        TXTFile.getInstance().fileClosed();
        textArea.setText(null);
    }

    /* İlgili metinin içeriği ile dosya kaydedilir. */
    @Override
    public void save(TextArea textArea) {
        TXTFile.getInstance().saveFile(textArea.getText());
    }

    /* İlgili metinin içeriği ile dosya farklı kaydedilir. */
    @Override
    public void saveAs(TextArea textArea) {
        TXTFile.getInstance().saveAs(textArea.getText());
    }

    @Override
```

```
    public void undo(TextArea textArea, ECareTaker careTaker) {
        /* careTaker.undo() ile originator bir önceki state'e set edildi.*/
        careTaker.undo();
        /* Sonrasında textArea'ya originator'da tutulan state (bir önceki state) bastırıldı.*/
        /* İlk getstate EMemento döndürür. Sonraki getState String döndürür (Memento.getState())*/
        textArea.setText(careTaker.getState().getState());
    }


    /* İlgili textArea'daki aranan kelim kaç adet var? Find metodu ile bu hesabı sağlayan yapılara ilgili
    * değerler gönderilir, yapılar oluştrulur.*/
    @Override
    public int find(TextArea textArea,String searchString) {

        /* Eğer textArea boş ise, arama kısmı boş ise, text kısmı null ise 0 değeri dönsün.*/
        if(textArea.getText().isEmpty() || searchString.isEmpty() ||textArea.getText() == null) return 0;

        /* Hesaplama için FindWithPattern constructurona textArea gönderilir. Obje üretilir.*/
        FindWithIterator findWithPattern = new FindWithIterator(textArea);

        /* Bu objee üzerindeki findCount metodu hesaplamayı yapar. Değer döndürülür. Aranan metin
gönderilmelidir.*/
        return findWithPattern.findCount(searchString);

    }
}
```

//FINDCOMMAND CLASS

```
package com.example.projectcode.COMMAND;


import javafx.scene.control.TextArea;
import javafx.scene.text.Text;
/* FindCommand, ICommand interface'ini implemente eder. Dolayısıyla execute metodu olmalıdır.*/

public class FindCommand extends ICommand{
    /* Sonuç ve textArea'da aranacak string değeri belirtilmiştir.*/
    int result;
    private String changeString;

    /* Aranacak string değeri FindCommand constructur'ına gönderilmiştir.*/
    public FindCommand(String changeString) {
        this.changeString = changeString;
    }

    /* execute metoduyla, sonuç belirlenmiştir. Sonucun belirlenmesi için operationslardan find metodu
kullanılmıştır.
    * kullanılmıştır. Bu metod FileOperations içinde detaylı tanımlanmıştır.
    * find ile ilgili textArea'da aranan string değerinin adedi sonuca atanmıştır. */
    @Override
    public void execute(TextArea textArea) {
        result = super.getOperations().find(textArea,changeString);
    }

    /* Sonuç değeri notePad içerisinde ilgili alana eklenmiştir. */
    public void changeField(Text text){
```

```
        text.setText(Integer.toString(result));
    }
}
```

//COMMAND ABSTRACT CLASS

```
package com.example.projectcode.COMMAND;


import javafx.scene.control.TextArea;
/* ICommand abstract class'tır, Bu class'ı extend edenler mutlaka bir Textarea alan execute metodunu
tanımlamalıdır.*/
public abstract class ICommand {
    /* FileOperations türünden operations tanımlanmıştır. Tanımlanma türünden dolayı
    * IOperationslardaki metotlar mutlaka FileOperations sınıfında yazılmış olmalıdır.*/
    private IOperations operations = new FileOperations();

    public abstract void execute(TextArea textArea);

    public IOperations getOperations() {
        return operations;
    }
}
```

//IOPERATIONS CLASS

```
package com.example.projectcode.COMMAND;


import com.example.projectcode.MEMENTO.ECareTaker;
import javafx.scene.control.TextArea;

/* IOperations interface'i tanımlanmıştır.*/
public interface IOperations {
    /* TextArea değerleri alan open, close, save, saveAs, undo ve find metotları, bu interface'i implemente edecek
    * olan class'ların tanımlamak zorunda olduğu metotlardır. undo ve find için ek olarak sırayla CareTaker ve
    * String türünden parametreler de gereklidir.*/
    void open(TextArea textArea);
    void close(TextArea textArea);
    void save(TextArea textArea);
    void saveAs(TextArea textArea);
    void undo(TextArea textArea, ECareTaker careTaker);
    int find(TextArea textArea,String searchString);




}
```

//OPENCOMMAND CLASS

```
package com.example.projectcode.COMMAND;


import javafx.scene.control.TextArea;
```

```
/* OpenCommand, ICommand interface'ini implemente eder. Dolayısıyla execute metodu olmalıdır.*/
public class OpenCommand extends Command {


    @Override
    public void execute(TextArea textArea) {
        super.getOperations().open(textArea);

    }
}
```

//SAVEASCOMMAND CLASS

```
package com.example.projectcode.COMMAND;


import javafx.scene.control.TextArea;
/* SaveAsCommand, ICommand interface'ini implemente eder. Dolayısıyla execute metodu olmalıdır.*/

public class SaveAsCommand extends Command {


    @Override
    public void execute(TextArea textArea) {
        super.getOperations().saveAs(textArea);
    }
}
```

//SAVECOMMAND CLASS

```
package com.example.projectcode.COMMAND;

import javafx.scene.control.TextArea;
/* SaveCommand, ICommand interface'ini implemente eder. Dolayısıyla execute metodu olmalıdır.*/

public class SaveCommand extends Command {


    @Override
    public void execute(TextArea textArea) {
        super.getOperations().save(textArea);
    }
}
```

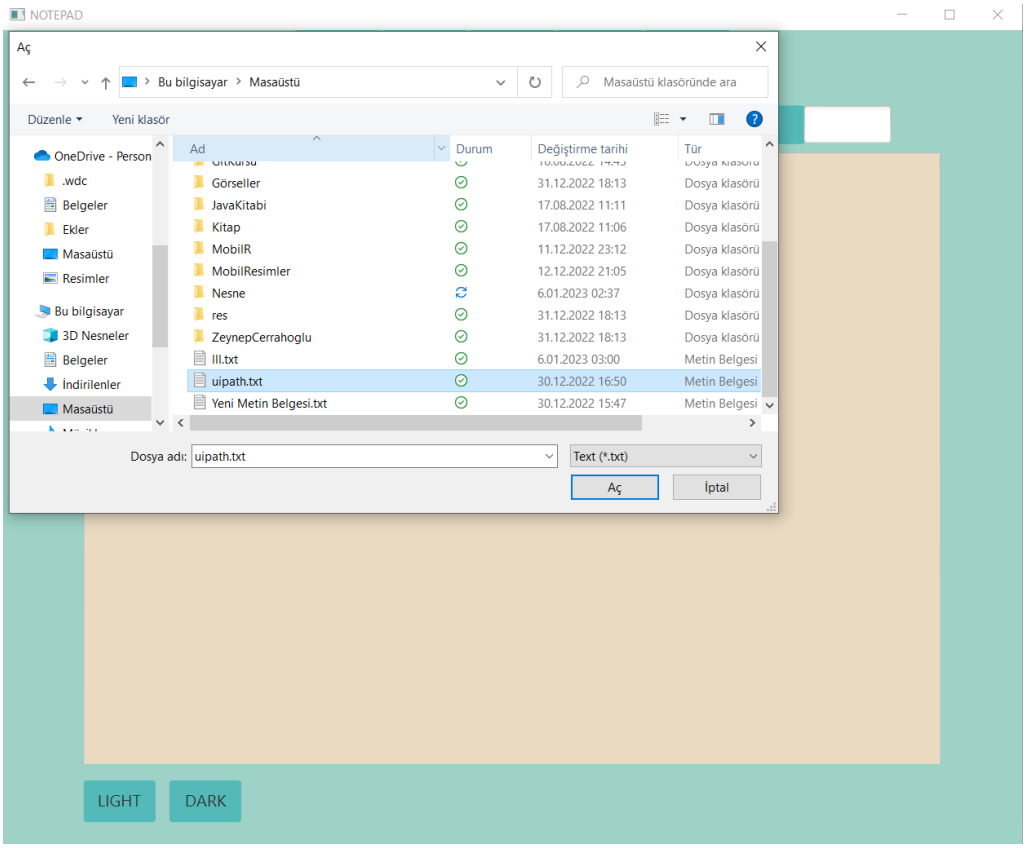
//UNDOCOMMAND CLASS

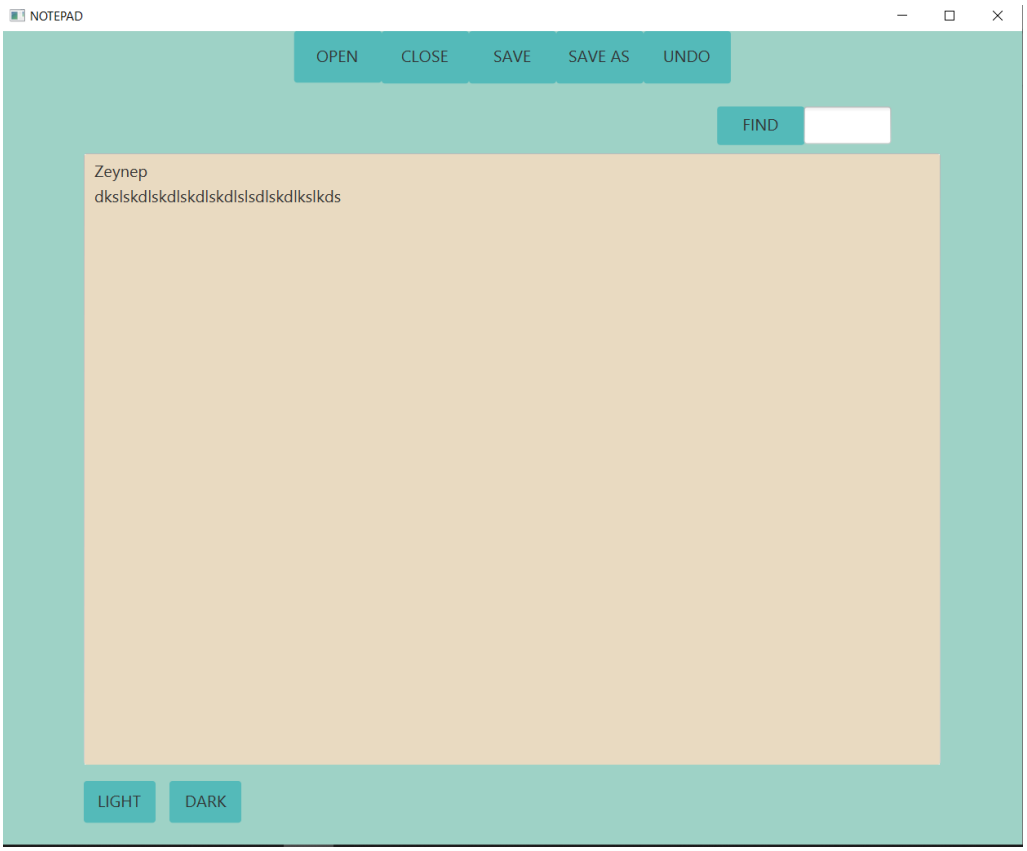
```
package com.example.projectcode.COMMAND;

import com.example.projectcode.MEMENTO.ECareTaker;
import javafx.scene.control.TextArea;
/* UndoCommand, ICommand interface'ini implemente eder. Dolayısıyla execute metodu olmalıdır.*/

public class UndoCommand extends Command {

    /* Bir careTaker oluşturulur ve UndoCommand constructor'ına gönderilir.*/
    private ECareTaker careTaker;
```

```
    public UndoCommand(ECareTaker careTaker) {
        this.careTaker = careTaker;
    }
    /* ICommand implemente edildiği için execute metodu olmalıdır. Burada FileOperations'daki
    * undo metoduna textArea ve careTaker gönderilir.*/
    @Override
    public void execute(TextArea textArea) {
        super.getOperations().undo(textArea,careTaker);
    }
}
```

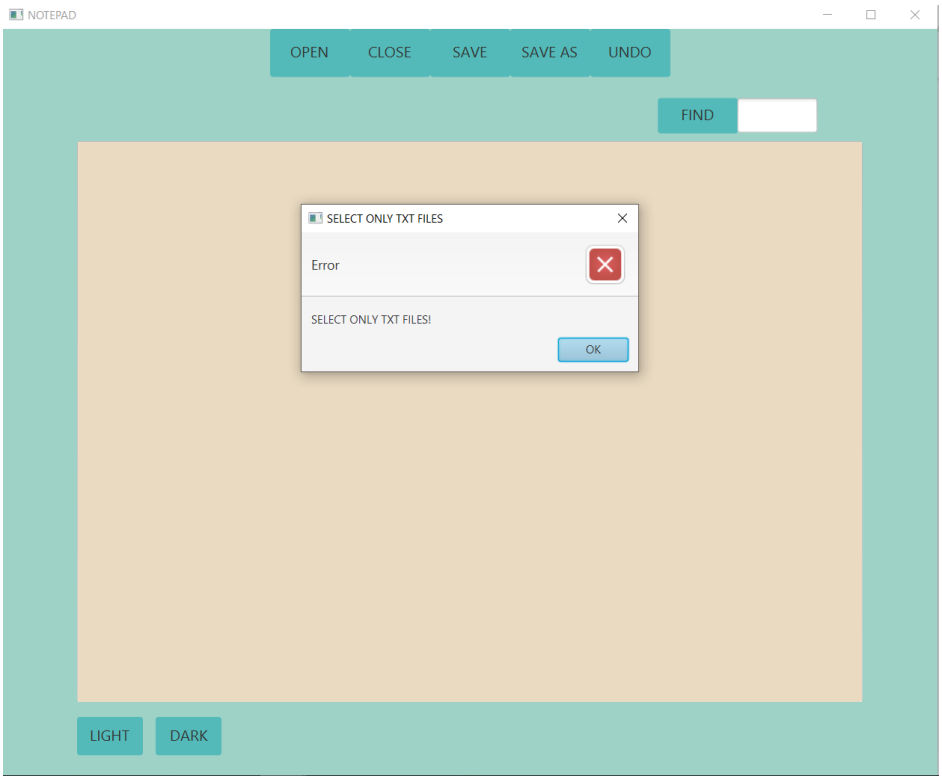
KULLANICI KILAVUZU

Mevcutta yer alan bir dosyayı açmak için OPEN butonuna tıklanılması gerekir. Butona tıkladığınızda istediğiniz dosyayı seçip Aç butonuna tıklayarak dosyanın içeriğini görebilirsiniz. Sadece txt dosyalarını seçebilirsiniz. Sistem sadece txt dosyalarını görmenizi sağlar.





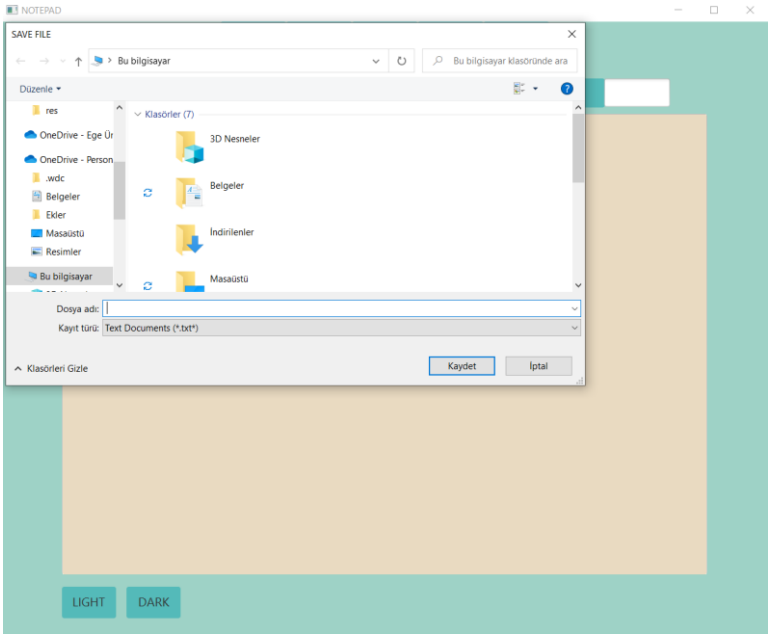
Eğer dosya seçmezseniz error alırsınız.



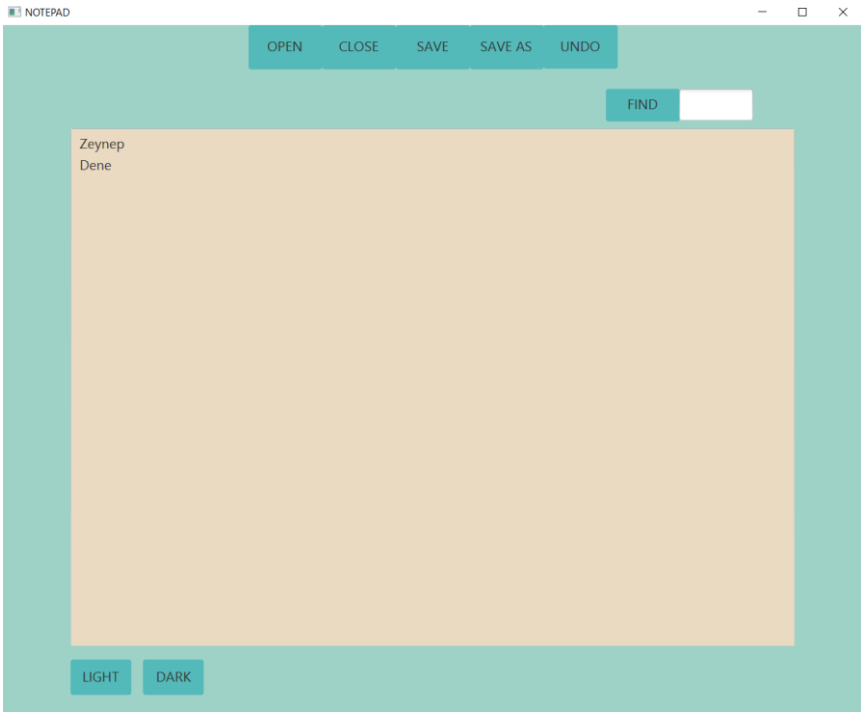
Dosyada herhangi bir değişiklik yapıp CLOSE butonuna basarsanız eğer dosyada yapılan en son değişikliklerle birlikte dosyayı aynı uzantıya kaydeder ve kapatır. OPEN butonuyla yeniden aynı dosyayı seçerseniz kaldığınız yerden dosya üzerinde işlem yapmaya devam edebilirsiniz.

SAVE butonuna tıklarsanız eğer CLOSE da olduğu gibi dosyada yapılan değişiklikler aynı dosya uzantısında kaydedilir fakat dosya kapanmaz. Eğer dosyada işlem yapmaya devam etmek istiyorsanız ve işlemlerinizin kaybolmasını istemiyorsanız SAVE butonunu kullanabilirsiniz.

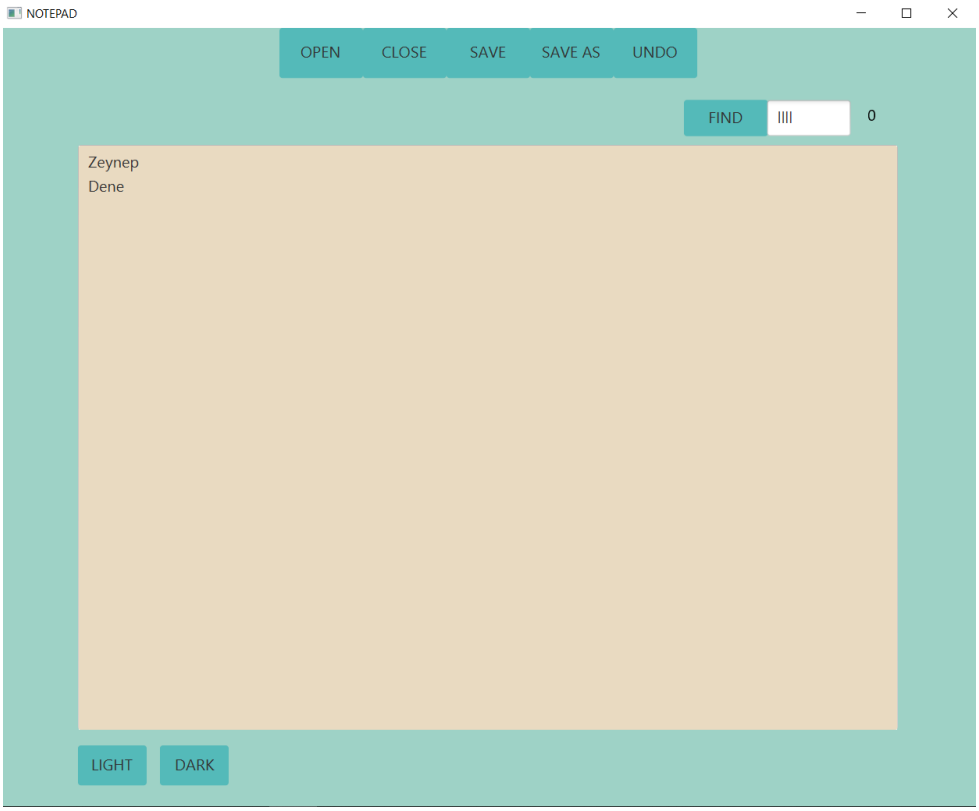
SAVE AS butonu ise dosyayı istediğiniz uzantıya yapılan değişikliklerle beraber kaydetmenizi sağlayacaktır.

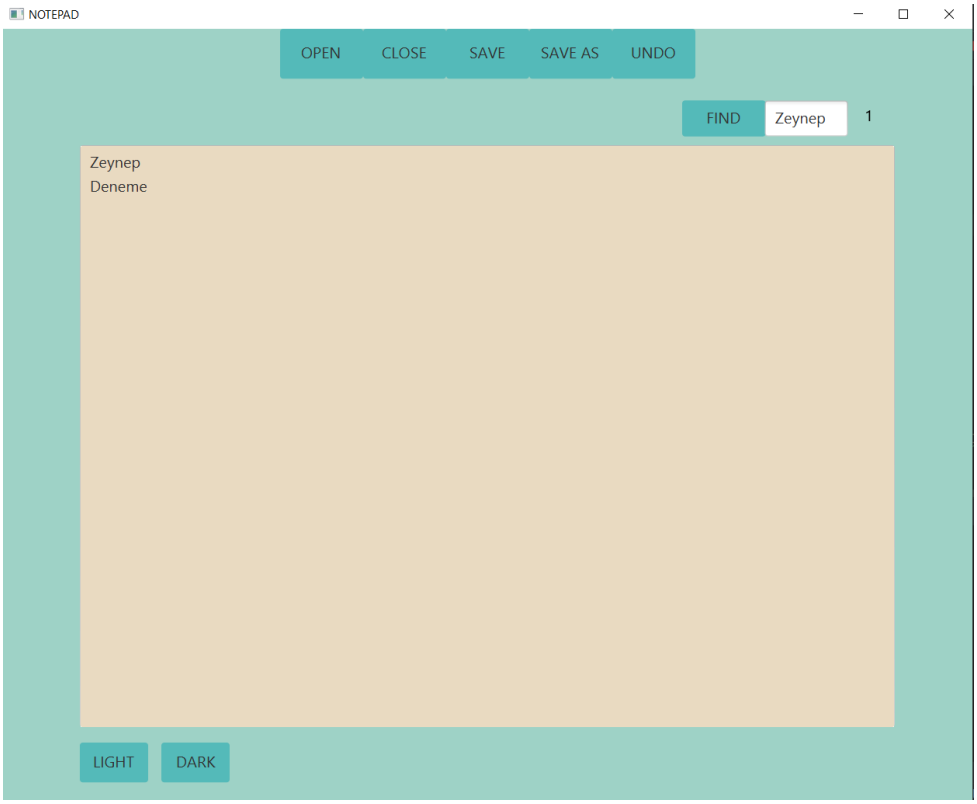


Yaptığınız işlemleri geri almak istiyorsanız UNDO butonunu kullanabilirsiniz. Yazdıklarınızı karakter karakter geri almanızı sağlayacaktır. İki kere UNDO tuşuna tıklanırsa görünecek olan metin örneği aşağıdaki gibidir.

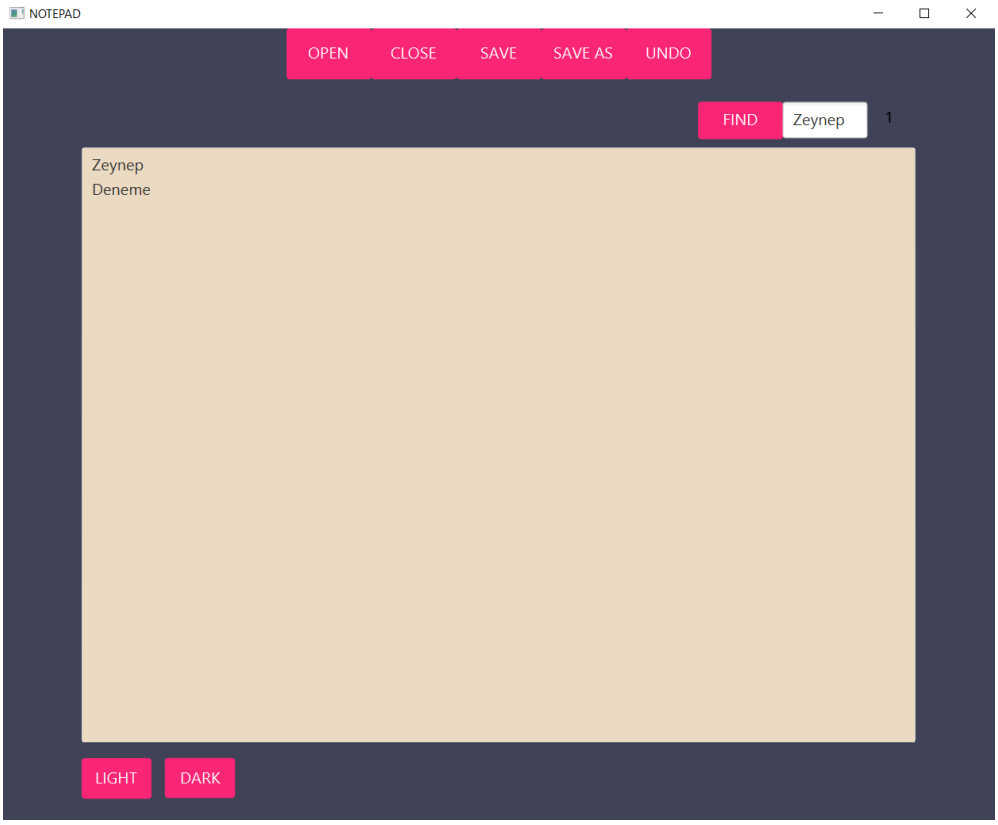


FIND butonun yanındaki boşluğa txt içinde bulmak istediğin kelimeyi girebilirsiniz. FIND butonuna tıkladıktan sonra yanında bu kelimeden txt içinde kaç tane olduğunu döndürecektir.





LIGHT VE DARK butonlarını kullanarak istediğiniz theme modunda çalışabilirsiniz. Editörünüz default olarak light modunda açılacaktır.


NOTEPAD
OPEN
CLOSE
SAVE
SAVE AS
UNDO
FIND
Zeynep
1
Zeynep
Deneme
LIGHT
DARK