NESNEYE DAYALI PROGRAMLAMA

PROJE-2

HAZIRLAYANLAR:

Kutay AVCI, 05180000091

Emre BALKAYA, 05180000056

Erdal Eren BAYAR, 05170000089

İÇİNDEKİLER

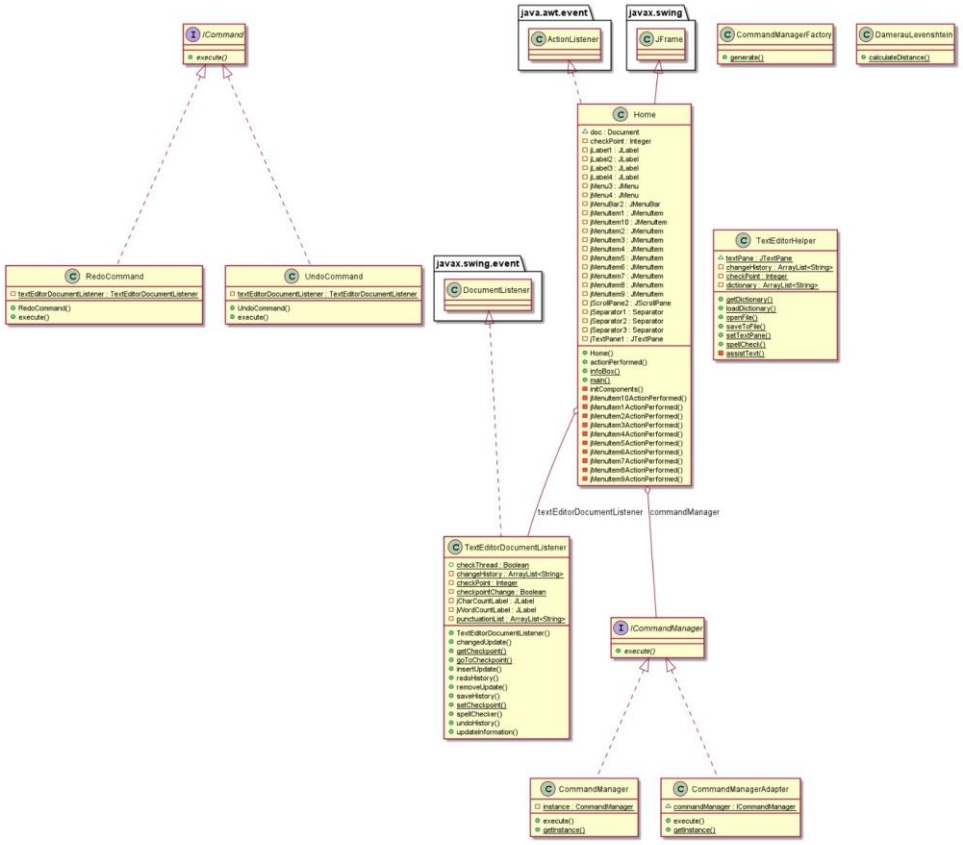
ANALİZ

- **Home sınıfı javax.swing.JFrame sınıfından kalıtım yoluyla elde edilmiştir ve Action Listener arayüzü uygulanmıştır. Fonksiyonel işlemlerin gerçekleştirimi sırasında kullanmak için integer tipinde, checkPoint adında bir değişken ve TextEditorDocumentListener sınıfından aynı adlı değişken ile Document containerından doc isimli değişken tanımlanmıştır.**
- **Arayüz bileşenleri için değişkenler tanımlanmıştır.**
- **DamareuLevensthein sınıfı iki string arasındaki uzaklık değerini döndüren bir algoritmadır. Bu sınıf Yazım Hatası Kontrolü fonksiyonun gerçekleştirimi için kullanılımıştır. CalculateDistance metodu altında bitişik harflerin yer değiştirmesi sonucu oluşan yazım hataları tespit edilmiştir.**
- **TextEditorDocumentListener sınıfımda DocumentListener arayüzü uygulanmıştır. Bu sınıfta ArrayList tipinde iki değişken bulunur. ChangeHistory değişkeni gerçekleşen son değişiklikleri tutan listedir. PunctuiationList ise noktalama işaretlerini barındıran listedir. JLabel tipindeki iki değişken kelime ve harf sayacı için oluşturulmuştur. Kontrol noktaları içinse integer tipinde checkPoint sayacı oluşturulmuştur. Kontrol noktasındaki değişimlerin takibi için iki farklı Boolean tipi değişken oluşturulmuştur.**
- **Kelime ve harf sayacı için conctructor metodu oluşturulmuştur.**
- **Kaydetme, silme ve değiştirme olayları için constructor metodlar oluşturulmuştur.**
- **SpellChecker metodu içinde Yazım Hatası Kontrolü ve değiştirilmesi işlemleri gerçekleştirilmiştir.**
- **TextEditorHelper sınıfında proje kapsamında istenilen fonksiyonel işlemlerin metodlarının gerçekleştirimi bu sınıf altında oluşturulmuştur.**

PROGRAMCI KATALOĞU

Bu projenin tasarım süreci 2-3 günlük bir süreç iken gerçekleştirim süreci 8-10 gün sürmüştür. Gerekli testler 1 gün içerisinde gerçekleştirilmiştir.

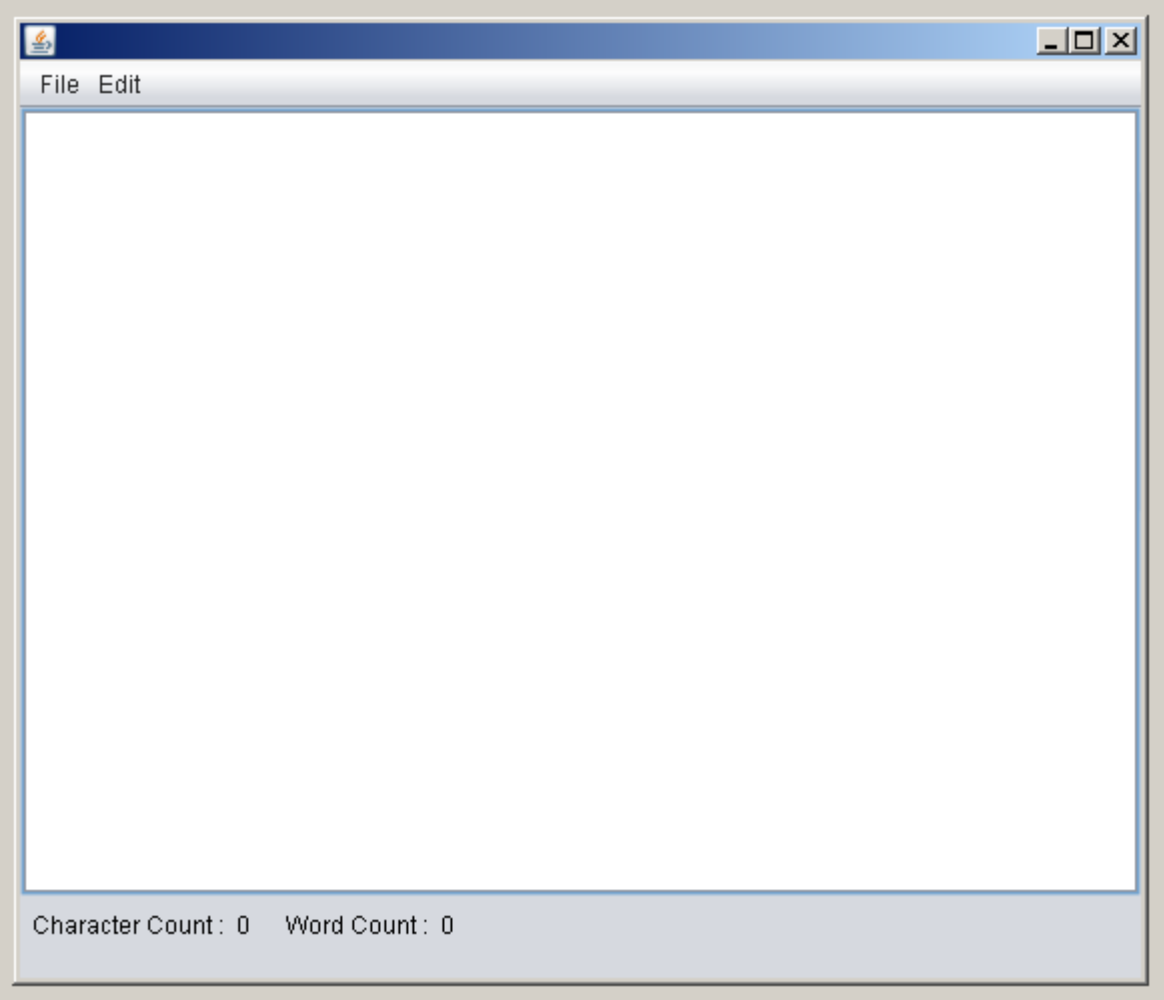
UML SINIF DİYAGRAMI



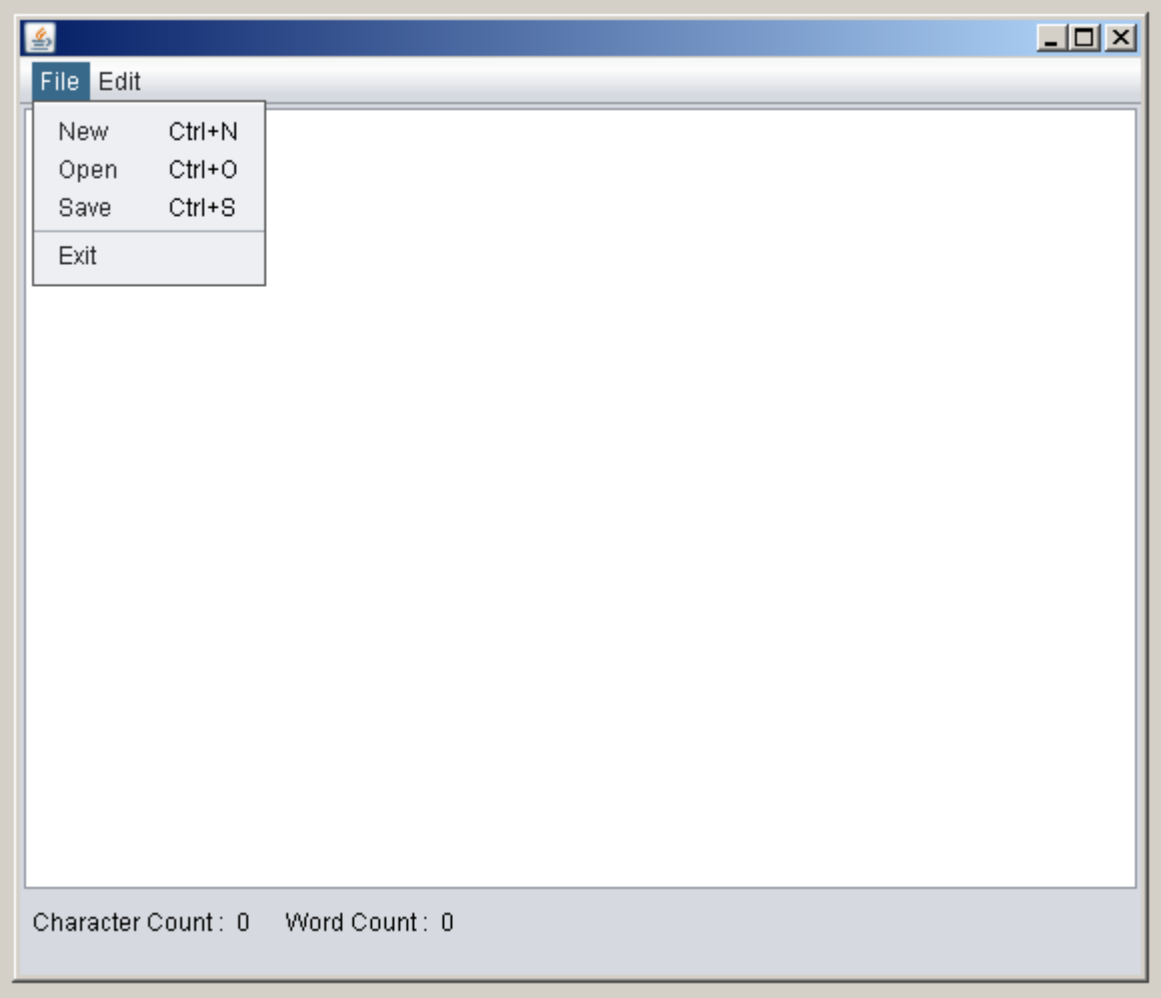
KAYNAK KOD

Proje kapsamında yazılan kaynak kodun teslimi EgeDers sistemi üzerinden oluşturulan Kaynak Kod Yükleme Klasörü'ne eklenmiştir.

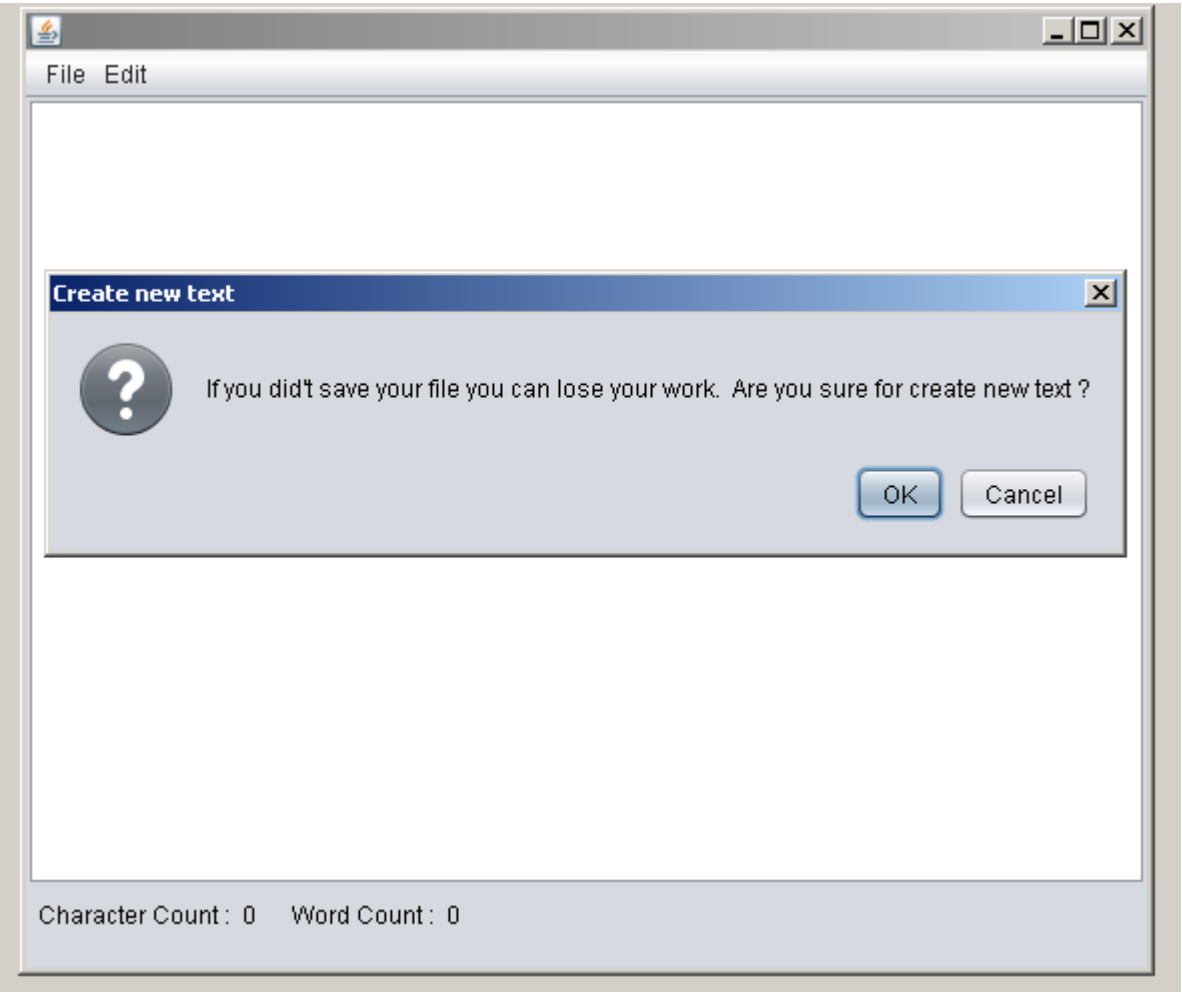
KULLANICI KATALOĞU



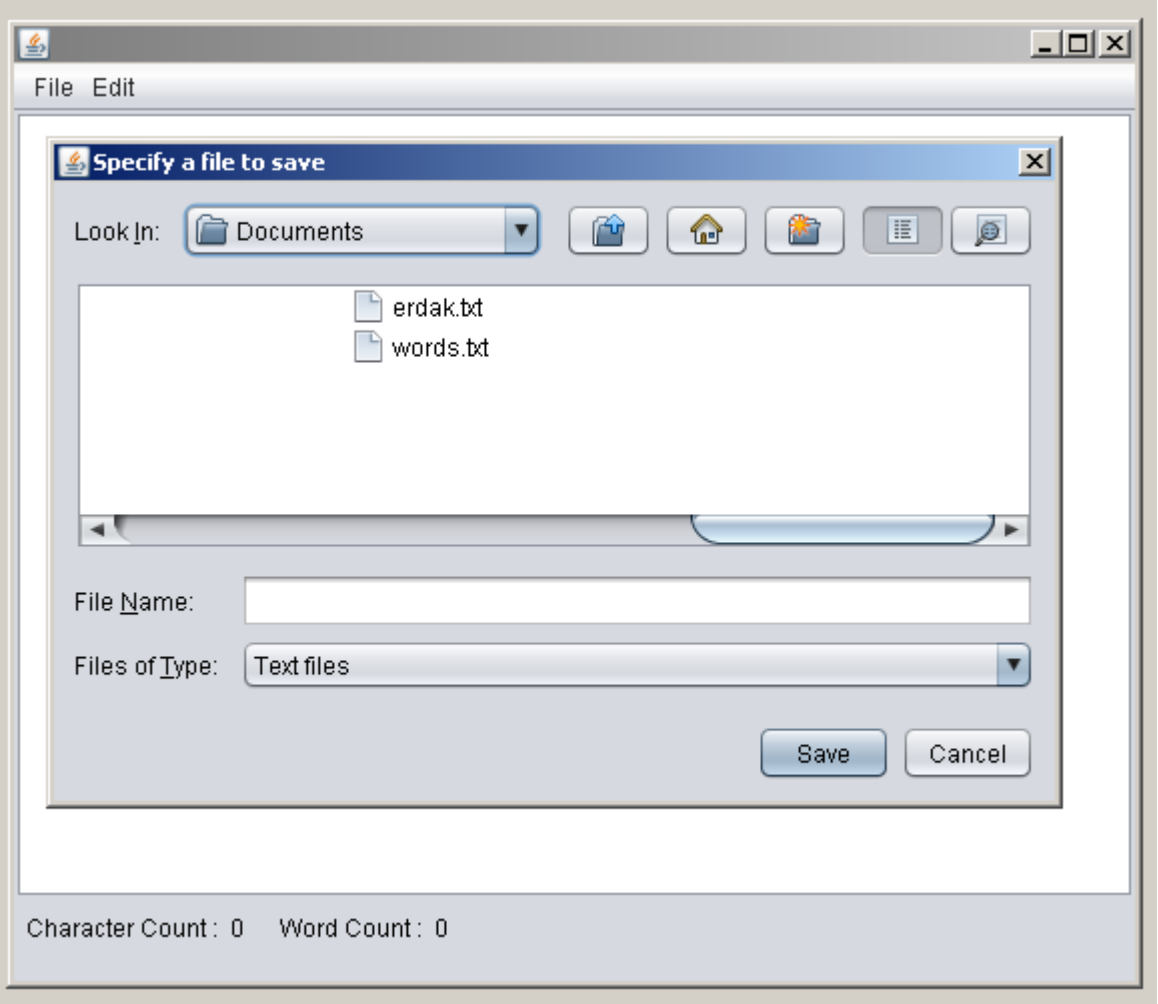
Program çalıştırıldığında kullanıcının karşılaşacağı arayüz şekildeki gibidir.



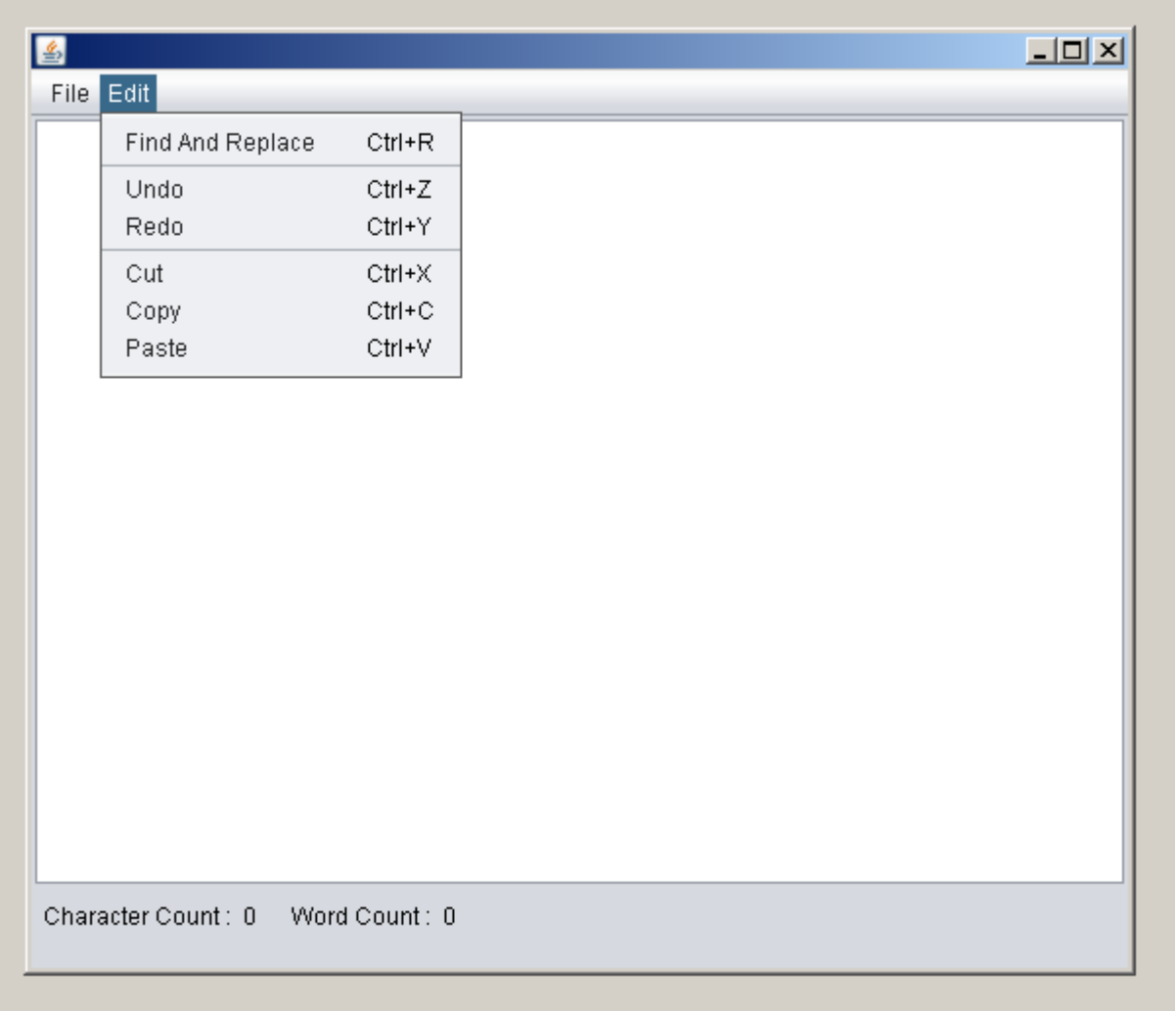
File sekmesi içerisinde New File, Open File, Save File ve Exit fonksiyonları ve kullanıcının bu fonksiyonlar için kullanabileceği kısayollar belirtilmiştir.



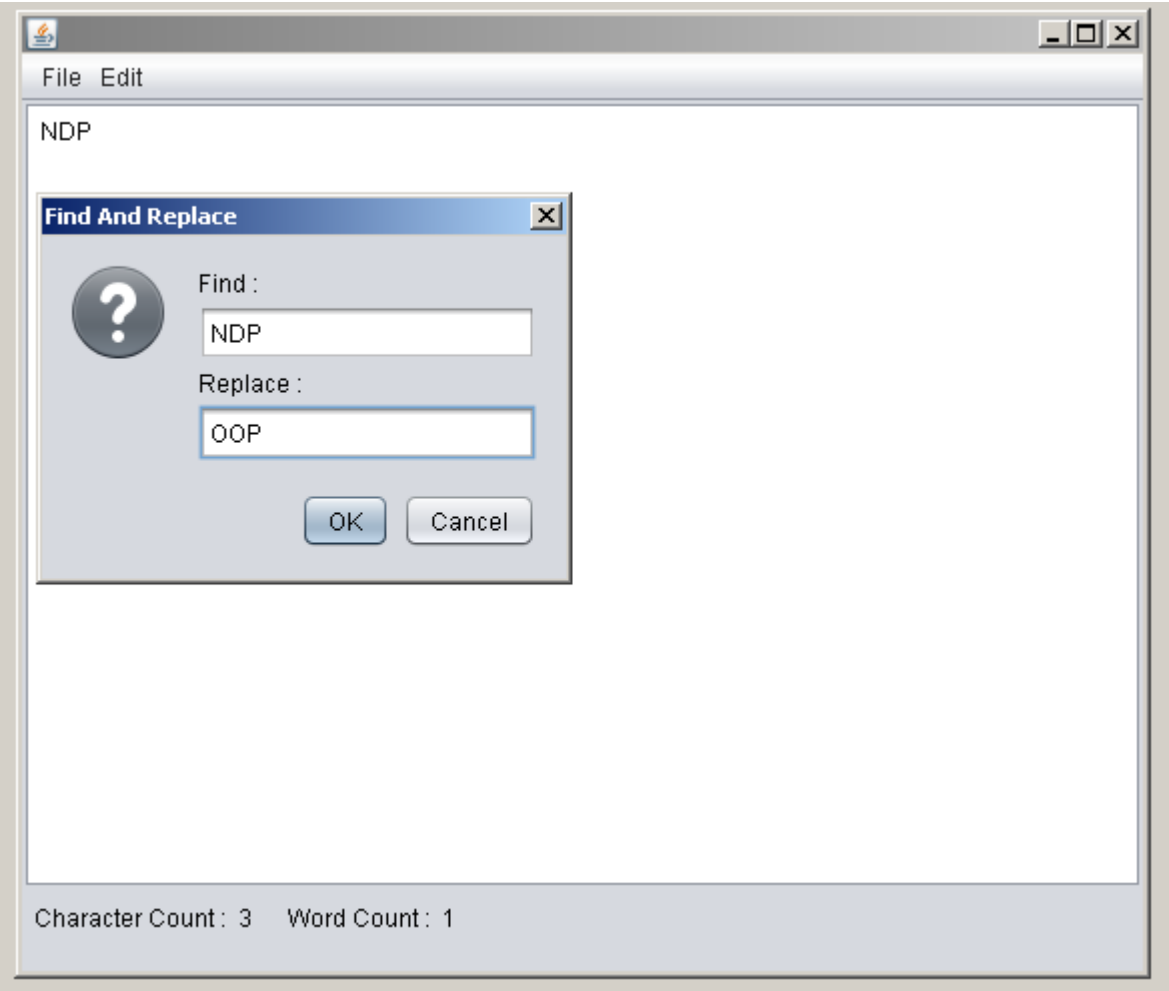
Kullanıcı New File komutunu kullandığında mevcut çalışmasını kaydetmediyse çalışmasının kaybolacağı hakkında bir uyarı verilip, yeni dosya açmak istediğine emin misin sorusu sorulmuştur. Kullanıcı mevcut çalışmasını kaydetmediyse Cancel butonuyla vazgeçebilir. Ya da OK butonuyla mevcut çalışmasını bırakarak yeni bir dosya oluşturabilir.



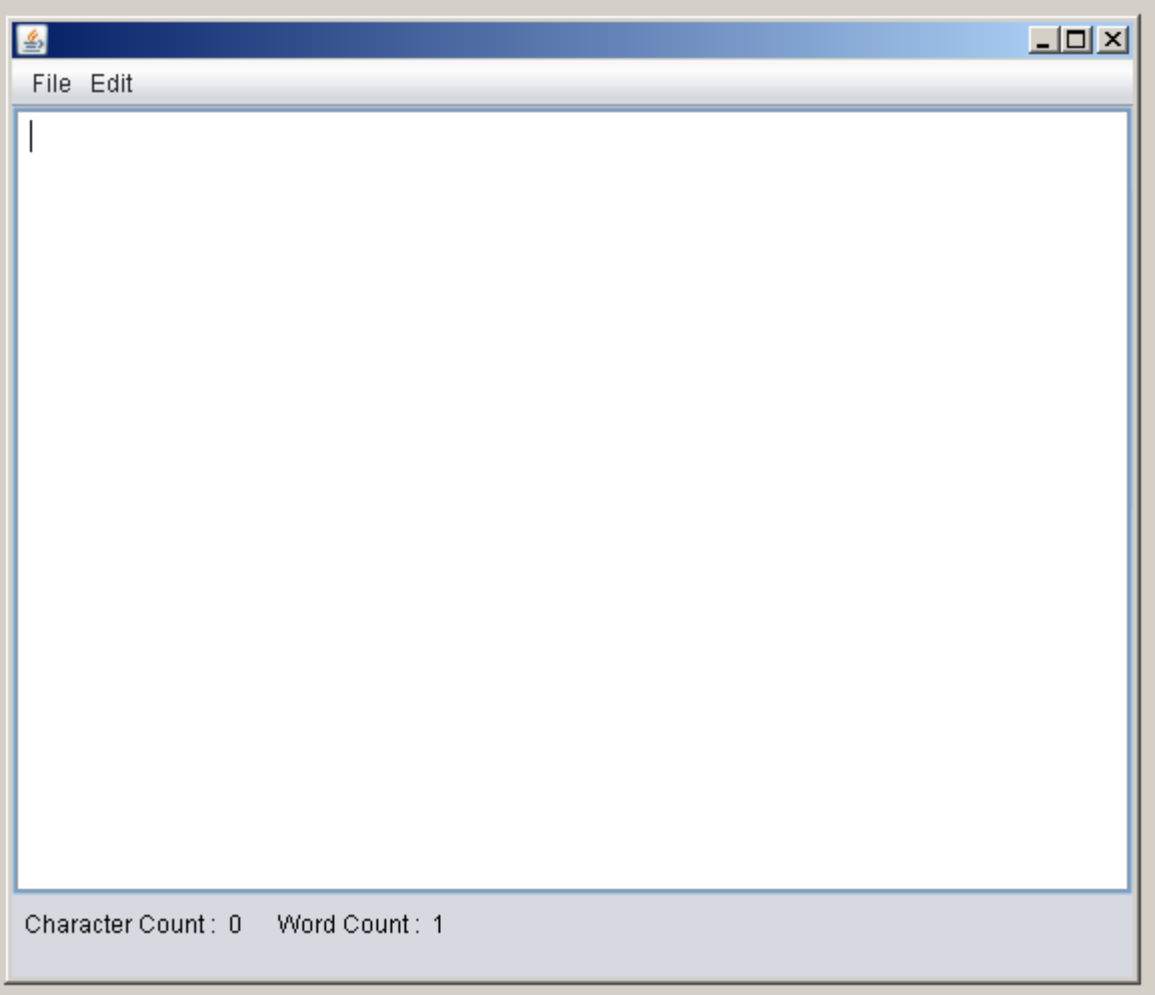
Kullanıcı Save File butonunu kullandığında çalışmasını hangi konuma kaydedeceğini belirlemesi için seçenek sunulur.



Edit sekmesi altında Find and Replace, Undo, Redo, Cut, Copy ve Paste fonksiyonları için butonlar bulunur ayrıca kullanıcının bu fonksiyonlar için kullanabilecekleri kısayollar belirtilmiştir.



Find and Replace butonu kullanıldığında kullanıcının karşısına bulmasını ve değiştirmesini istediği kelimeyi belirtmesi istenmiştir.Kullanıcı bunun dışındaki fonksiyonları ister buton kullanarak isterse kısayolları kullanarak editör üzerinde gerçekleştirebilir.



Kullanıcı sağ üstteki butonlar yardımıyla sırasıyla editörü simge durumuna küçültebilir, ekran boyutuna kaplayabilir ya da editörü kapatabilir.

```java
@Test
public void testCalculateDistance() {
    assertEquals(0, DamerauLevenshtein.calculateDistance("", ""));
    assertEquals(0, DamerauLevenshtein.calculateDistance(" ", " "));
    assertEquals(1, DamerauLevenshtein.calculateDistance("", " "));
    assertEquals(1, DamerauLevenshtein.calculateDistance(" ", ""));
    assertEquals(0, DamerauLevenshtein.calculateDistance("test", "test"));
    assertEquals(1, DamerauLevenshtein.calculateDistance("Test", "test"));
    assertEquals(1, DamerauLevenshtein.calculateDistance("test", "testy"));
    assertEquals(1, DamerauLevenshtein.calculateDistance("testy", "test"));
    assertEquals(1, DamerauLevenshtein.calculateDistance("test", "tets"));
    assertEquals(1, DamerauLevenshtein.calculateDistance("test", "test "));
}

@Test(expected = IllegalArgumentException.class)
public void testSourceNull() {
    DamerauLevenshtein.calculateDistance(null, "");
}

@Test(expected = IllegalArgumentException.class)
public void testTargetNull() {
    DamerauLevenshtein.calculateDistance("", null);
}

@Test(expected = IllegalArgumentException.class)
public void testSourceAndTargetNull() {
    DamerauLevenshtein.calculateDistance(null, null);
}
```

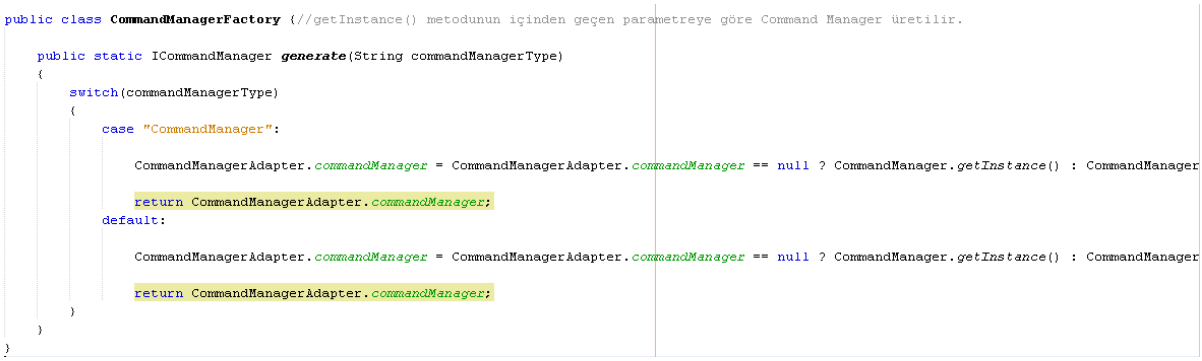
Yazım hatası kontrolü ve düzeltilmesi için kullanılan Damerau-Levensthein uzaklık algoritmasının JUnit Testinin gerçekleştirimi şekildeki gibi yapılmıştır.

```java
import java.util.List;
import java.util.Iterator;

public class CommandManager implements ICommandManager {// Command Design Pattern ile çalışan fonksiyonlar tetiklendiğinde execute

    private static CommandManager instance = null;

    static CommandManager getInstance()
    {
        CommandManager.instance = CommandManager.instance == null ? new CommandManager() : CommandManager.instance;

        return CommandManager.instance;
    }

    public void execute(List<ICommand> commandList)
    {
        Iterator<ICommand> ite = commandList.iterator(); //Iterator Design Pattern ile dolaşılmıştır.
        while(ite.hasNext()){
            ICommand item = ite.next();
            item.execute();
        }
    }
}
```

Command Design Pattern uygulaması için oluşturulan Invoker sınıfı, CommandManager adıyla oluşturuldu ve ICommandManager arayüzünden kalıtım alındı. Aynı zamanda execute methodu içinde koleksiyon dolaşılırken Iterator Design Pattern uygulandı.

```java
import java.util.List;


public class CommandManagerAdapter implements ICommandManager {//ICommandManager ve ICommand arayüzlerinin adapte edilmesi.

    static ICommandManager commandManager;

    public static ICommandManager getInstance(String commandManagerType)
    {
       return CommandManagerFactory.generate(commandManagerType);
    }

    @Override
    public void execute(List<ICommand> commandList) {
        CommandManagerAdapter.commandManager.execute(commandList);
    }

}
```

Adapter Design Pattern uygulaması için CommandManagerAdapter sınıfı ICommandManager sınıfından kalıtımla elde edildi. ICommand arayüzüyle ICommandManager arayüzleri bu sınıfla adapte edildi.



```
public class CommandManagerFactory {//getInstance() metodunun içinden geçen parametreye göre Command Manager üretilir.

    public static ICommandManager generate(String commandManagerType)
    {
        switch(commandManagerType)
        {
            case "CommandManager":

                CommandManagerAdapter.commandManager = CommandManagerAdapter.commandManager == null ? CommandManager.getInstance() : CommandManager

                return CommandManagerAdapter.commandManager;
            default:

                CommandManagerAdapter.commandManager = CommandManagerAdapter.commandManager == null ? CommandManager.getInstance() : CommandManager

                return CommandManagerAdapter.commandManager;
        }
    }
}
```

Factory Method Design Pattern uygulaması için CommandManagerFactory sınıfı oluşturuldu. Bu uygulamada getInstance() metodunun içinden geçen parametreye göre bir Command Manager üretildi.

```
public class UndoCommand implements ICommand {//Undo fonksiyonunun Command Design Pattern ile gerçekleştirilmesi.

    private static TextEditorDocumentListener textEditorDocumentListener;

    public UndoCommand(TextEditorDocumentListener textEditorDocumentListener)
    {
        UndoCommand.textEditorDocumentListener = UndoCommand.textEditorDocumentListener == null ? textEditorDocumentListener : Und
    }

    @Override
    public void execute() {//Undo işleminin gerçekleştirimi


        int changedCheckPoint = UndoCommand.textEditorDocumentListener.getCheckpoint() - 2;


        UndoCommand.textEditorDocumentListener.setCheckpoint(changedCheckPoint);


    }

}
```

Undo fonksiyonunun Command Design Pattern yoluyla uygulanması için gerekli olan metodlar UndoCommand sınıfı içerisinde ICommand arayüzünden kalıtım alınarak gerçekleştirildi.

```
private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt) {

    List<ICommand> commandList = new ArrayList<ICommand>();

    commandList.add(new UndoCommand(this.textEditorDocumentListener));

    commandManager.execute(commandList);

    this.jTextPane1.setText(TextEditorDocumentListener.goToCheckpoint());
}
```

Undo fonksiyonunun Command Design Pattern yoluyla gerçekleştirimi.