PANEVROPSKI UNIVERZITET „APEIRON“
FAKULTET INFORMACIONIH TEHNOLOGIJA

Smjer „Programiranje i softverski inženjering“

Predmet
Objektno orjentisano programiranje u Javi

Seminarski rad na temu:

Aplikacija za analizu fajla

Predmetni nastavnik
Doc. Dr Saša Salapura

Student
Amar Badnjević
Broj indeksa: 200-18/RITP-S

Banja Luka, oktobar 2019.

Sadržaj

Uvod

Tema ovog seminarskog rada je aplikacija koja vrši analizu proslijeđenog karaktera, prikazuje grafikone i po želji korisnika, upisuje podatke analize u fajl.
Dva glavna segmenta ove aplikacije su rad sa fajlovima i prikazivanje statistike analiziranja fajla.

Prilikom pristupanja aplikaciji, korisnik je zadužen da proslijedi fajl kojeg želi analizirati, a zatim nakon analiziranja fajla dobija statistiku fajla prikazanu preko grafikona. Korisnik ima nekoliko mogućnosti pregleda statistike analiziranja fajla:

- Cijela statistika analize u jednom grafikonu,
- Statistika analize velikih slova,
- Statistika analize malih slova,
- Statistika analize brojeva, i
- Statistika analize specijalnih znakova kao što su: #, %, $, itd.

Svaka statistika analize, navedena iznad, prikazana je u obliku grafikona. Korisnik ima više mogućnosti rada nad prikazanim grafikonom analize, od kojih su neke:

- Uvećavanje i umanjivanje grafikona radi jasnijeg pregleda podataka određenih polja,
- Kopiranje grafikona,
- Spremanje grafikona kao slike u „.png“ tip fajla,
- Izmjena postavki grafikona, kao što je npr.: naziv, boja, itd.

Analiza fajla se vrši nad svakim karakterom (znaka, u daljem tekstu karakter) posebno, što na kraju korisniku daje rezultate statistike svakog karaktera i broj ponavljanja istog u fajlu. Svi karakteri su podijeljeni u četiri cjeline koje su navedene iznad: velika slova, mala slova, brojevi i specijalni znakovi.

Također, korisnik ima mogućnost zapisivanja statistike analize fajla u drugi fajl koji će se automatski generisati i biće spremljen u direktorij fajla koji je analiziran pod nazivom „stats_imeFajla.txt“. Dio naziva fajla „imeFajla“ posjeduje isti naziv kao učitani fajl, kako bi korisnik znao da se radi o statistici tog fajla.

Statistika koja će biti ispisana u fajl će u nastavku rada biti detaljno objašnjena, a sad, samo navedena:

- Fajl koji je analiziran,
- Sistemska putanja do analiziranog fajla,
- Vrijeme potrebno za analizu fajla u milisekundama,
- Sistemski datum, dan i vremenska zona iz koje je pokrenuto analiziranje fajla,
- Statistika cijelog fajla poredana po različitim grupama karaktera navedenih iznad.

Cilj seminarskog rada je da se čitaoc upozna sa jednostavnim i efikasnim načinom rada sa fajlovima, upotrebom grafikona i rad sa istim u programskom jeziku Java. Naravno, veoma je bitno i upoznavanje rada sa objektno orjentisanim programiranjem na šta se predmet svodi.

Napomena: Cijelom izvornom kodu možete pristupiti ovdje, a nakon preuzimanja fajlova i omogućavanja svih preduslova uspješnog pokretanja projekta, čitaoc je u mogućnosti mijenjati izvorni kod na svom računaru.

Korišteni alati

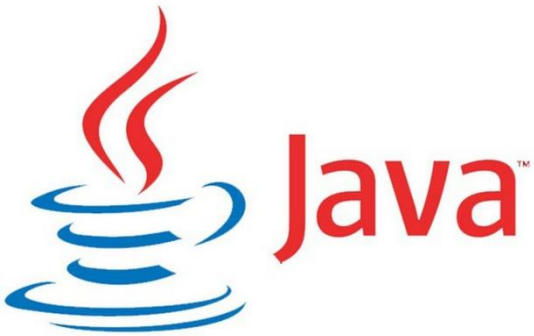
Za izradu ovog projekta, potrebno je korištenje nekoliko alata koji će u nastavku biti objašnjeni. Korišteni alati su sljedeći:

- Java programski jezik,
- Biblioteke za izradu grafikona,
- Integrisano razvojno okruženje,
- Kontrola verzije koda i,
- GitHub kao mjesto za spremanje koda projekta.

Neki od alata su neophodni za uspješnu izradu projekta dok su neki korišteni kako bi izrada projekta bila olakšana i jednostavnija.

Programski jezik Java

Programski jezik Java je veoma poznat publici i nije potrebno detaljno objašnjavanje ovog alata.



Slika 1 Logo programskog jezika Java

Bitno je napomenuti da je Java programski jezik opšte namjene, dizajniran da bude baziran na klasama i objektno orjentisanom programiranju. Veoma je moćan programski jezik i korisit se u razne svrhe. Programiranje desktop, web i Android aplikacija je samo jedna od odlika ovog programskog jezika. Također, implementacija je dizajnirana tako da aplikacije kreirane u ovom programskom jeziku, funkcionišu bez problema na različitim platformama uz prethodno omogućavanje preduvjeta.

Biblioteke za izradu grafikona

Korištene su dvije biblioteke za izradu grafikona a to su:

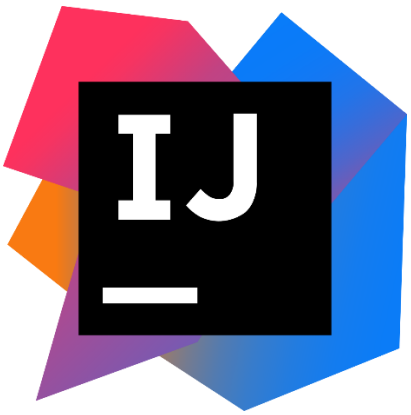
- JCommon, verzija 1.0.23
- Jfreechart, verzija 1.0.19.

Slika 2 Logo biblioteke "JFreeChart"

Obje biblioteke je moguće preuzeti ovdje. Za kreiranje grafikona koristi se JfreeChart biblioteka, međutim, ista se ne bi mogla koristiti ako nije preuzeta i druga biblioteka pod nazivom JCommon.

Integrisano razvojno okruženje

Integrisano razvojno okruženje (eng. Integrated Development Environment – IDE) je „pametno“ okruženje koje razvojnom pograneru mnogo pomaže pri izradi aplikacije.
Aplikacija se može pisati u bilo kojem tekstualnom editoru, čiji se kod nakon toga mora kompajlirati. Međutim, prednost integrisanog razvojnog okruženja je u tome što ima ugrađene alate za kompajliranje, formatiranje koda i mnogo drugih alata što proces izrade aplikacije čini mnogo bržim i jednostavnijim.



Slika 3 Logo IntelliJ IDE

Logo integrisanog razvojnog okruženja koje je korišteno, prikazano je na slici iznad. Naziv okruženja koje je korišteno je „IntelliJ IDE“ što je skraćeno od „Intelligent Java“. Ovo razvojno okruženje je konkretno namijenjeno za razvoj Java aplikacija a kompanija koja je kreirala ovo okruženje naziva se „JetBrains“ i posjeduje mnogo drugih alata i razvojnih okruženja koja možete pogledati ovdje.

Git

Git je alat koji se koristi za kontrolu verzije koda. Alat je odličan ako želimo da u bilo kojem trenutku spremimo sve što smo uradili, vratimo se na druge verzije, postavimo kod negdje na server kako bismo mogli pristupiti istom sa drugog računara i slično.

Slika 4 Git logo

Ovaj alat je veoma koristan za rad timova kako bi proces izrade softvera bio jednostavniji, kod distribuiran a isto tako, jednostavnija komunikacija i dogovor programera koji rade na razvoju softvera.
Također, Git je odličan za upotrebu kada jedna osoba radi na projektu. Razlog tome je taj što, nebitno koliko ta ista osoba poznaje programski jezik i ostale alate, veoma je lako da se izgubi u sopstvenom kodu, zatreba mu neka druga verzija i slično. Ovaj alat tada stupa na snagu.

GitHub

GitHub je online platforma koja služi kao repozitorijum za čuvanje raznih fajlova, kako projekata tako i drugih fajlova.

Slika 5 GitHub Logo

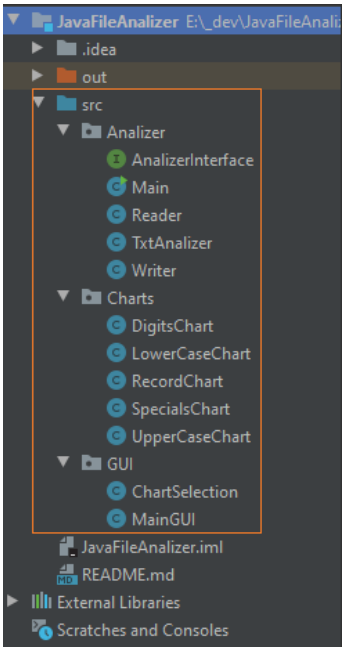
Kako biste pristupili ovoj platformi potrebno je da napravite račun nakon čega dobijate pristup vašem računu i prostoru gdje možete kreirati javne i privatne repozitorijume. Jedan od repozitorijuma je repozitorijum ovog projekta, koji je spomenut na u uvodu seminarskog a repozitorijumu možete pristupiti ovdje.

Struktura projekta

Kako bi se prikazalo neophodno znanje za predmet Objektno orjentisano programiranje u Javi, kreirani su različiti paketi i fajlovi koji zajedno surađuju kako bi aplikacija u potpunosti funkcionisala.

Struktura projekta se sastoji od sljedećih stavki:

- Tri različita paketa za različite funkcionalnosti aplikacije i,
- Fajlovi u tim paketima koji zajedno surađuju.



Slika 6 Struktura projekta

Na slici iznad je prikazana struktura projekta. Ono što je bitno za ovaj seminarski rad je folder „src“ gdje se nalazi izvorni kod projekta. U nastavku su objašnjeni paketi i fajlovi projekta.

Paketi projekta

U ovom projektu se nalaze tri različita paketa koja sadrže fajlove izvornog koda. Ti paketi su sljedeći:

- Analizer,
- Charts, i
- GUI

Analizer paket sadrži fajlove koji su zaduženi za samu analizu fajlova i kreiranje statistike iste. Pod naslovom „Fajlovi“ objašnjeni su fajlovi svih paketa.
Charts paket sadrži fajlove koji služe za kreiranje grafikona onda kada korisnik na interfejsu odabere željeni grafikon.
GUI paket sadrži fajlove koji su konkretno vezani za grafički korisnički interfejs i interakciju korisnika sa aplikacijom. Ovdje se prikazuje korisniku interfejs gdje ima mogućnost odabira različitih opcija.

Fajlovi

Fajlovi *Analizer* paketa su sljedeći:

- AnalizerInterface – u kojem se nalazi interfejs za analizu. Služi kako bi bio ugovor klasama koje ga implementiraju, što znači da te klase moraju ispoštovati kreiranje metoda koje su kreirane u interfejsu.
- Main – u ovom fajlu se nalazi Main klasa i main metoda od koje aplikacija započinje izvršavanje.
- Reader – ovaj fajl sadrži klasu Reader u kojoj se nalaze metode za otvaranje, čitanje i zatvaranje fajla.
- TxtAnalizer – je fajl koji je velikim dijelom zadužen za analizu. U njemu se, ustvari dešava svo analiziranje fajla. Sadrži više metoda koje zajedno surađuju kako bi fajl bio uspješno analiziran.
- Writer – je fajl koji sadrži klasu i metode za pisanje podataka u fajl o statistici. Metode pronalaze sistemsku putanju analiziranog fajla i na tom mjestu kreiraju fajl za statistiku i u njega upisuju podatke.

Fajlovi *Charts* paketa su većinom slični po strukturi koda ali male razlike u kodu čine da korištenje istih daje različite rezultate.
Fajlovi su sljedeći:

- *DigitsChart* – ovaj fajl sadrži klasu koja svojom instancom prikazuje grafikon analize fajla samo za *brojeve* koji se nalaze u analiziranom fajlu.
- *LowerCaseChart* – veoma slična struktura fajla ali je razlika u tome što je rezultat grafikon koji prikazuje statistiku *malih slova* analiziranog fajla.
- *SpecialsChart* – prikazuje statistiku *specijalnih* karaktera analiziranog fajla.
- *UpperCaseChart* – prikazuje statistiku *velikih slova* analiziranog fajla.
- *RecordChart* – razlika ovog grafikona i ostalih je u tome što se ovdje prikazuje statistika *svih karaktera* analiziranog fajla

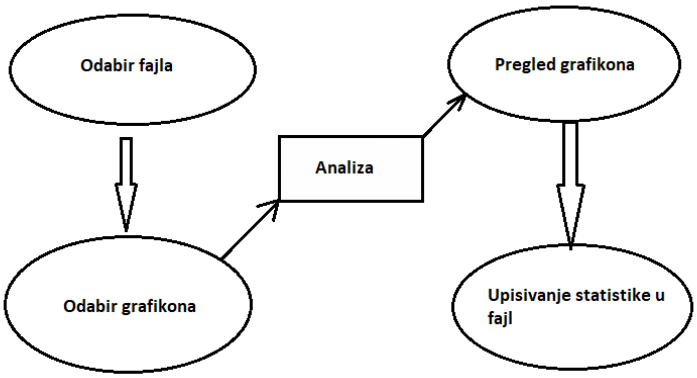
Fajlovi *GUI* paketa su sljedeći:

- MainGUI – sarži klasu koja njenim instanciranjem korisniku prikazuje početni interfejs za odabir fajla koji će se analizirati.
- ChartSelection – Nakon odabira fajla, instanciranjem klase iz ovog fajla korisniku se prikazuje interfejs u kojem će imati nekoliko mogućnosti odabira grafikona koje želi pogledati. Pored grafikona, ima mogućnost i upisivanja statistike u fajl klikom na samo jedno dugme.

U nastavku rada, svi fajlovi i paketi koji su navedeni do sad će biti prikazani. Isto tako, biće prikazan interfejs i kod aplikacije.

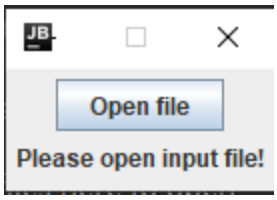
Opis procesa, interfejs i kod aplikacije

Kompleksnost trenutne aplikacije nije velika. Na slici ispod se može vidjeti proces koji korisnik prolazi kroz aplikaciju kako bi dobio potrebne informacije, odnosno grafikone.



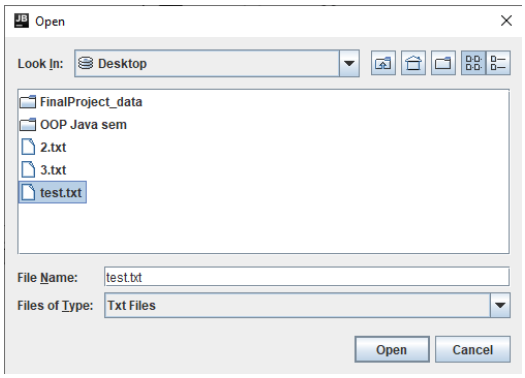
Slika 7 Proces aplikacije

Nakon što korisnik pokrene aplikaciju, prikazuje mu se jednostavan interfejs koji traži od korisnika da odabere željeni fajl za analiziranje. Taj prozor se može vidjeti na slici ispod.



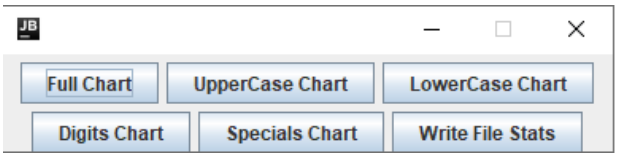
Slika 8 Prvi prozor aplikacije

Kao što se može vidjeti, dostupno je samo jedno dugme u trenutnom prozoru. Kada korisnik klikne na to dugme, otvori mu se dijalog odabira fajla koji se može vidjeti na slici ispod.



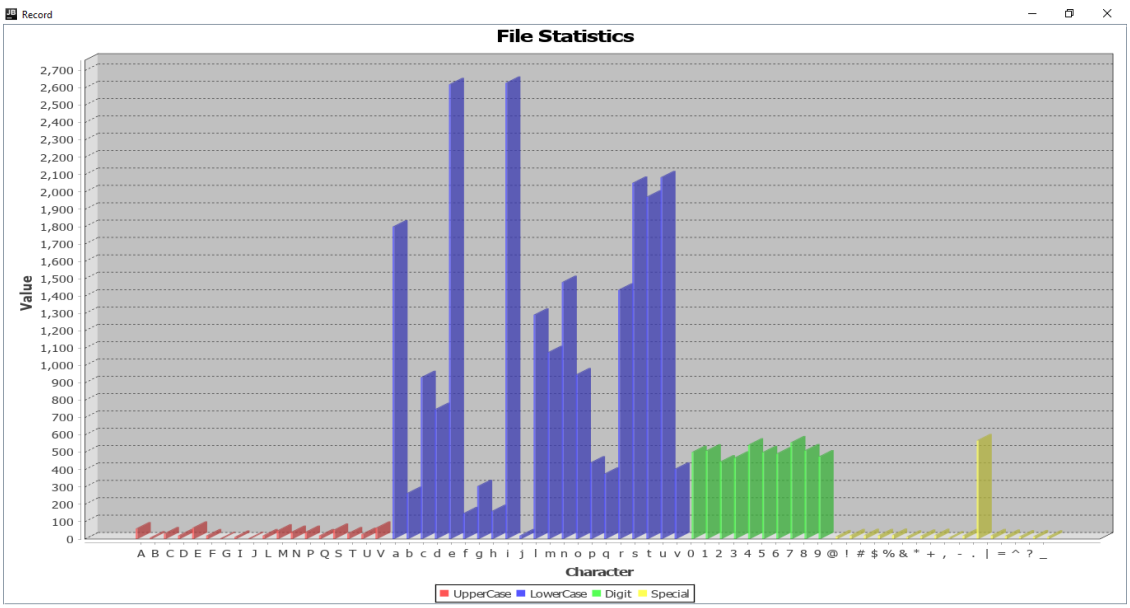
Slika 9 Dijalog odabira fajla

Nakon što korisnik odabere fajl, u trenutnom slučaju, taj fajl se naziva „test.txt“, otvara mu se prozor gdje odabira koji tip grafikona želi prikazati ili želi upisati statistiku u fajl. To se može vidjeti na slici ispod.



Slika 10 Prozor za odabir grafikona

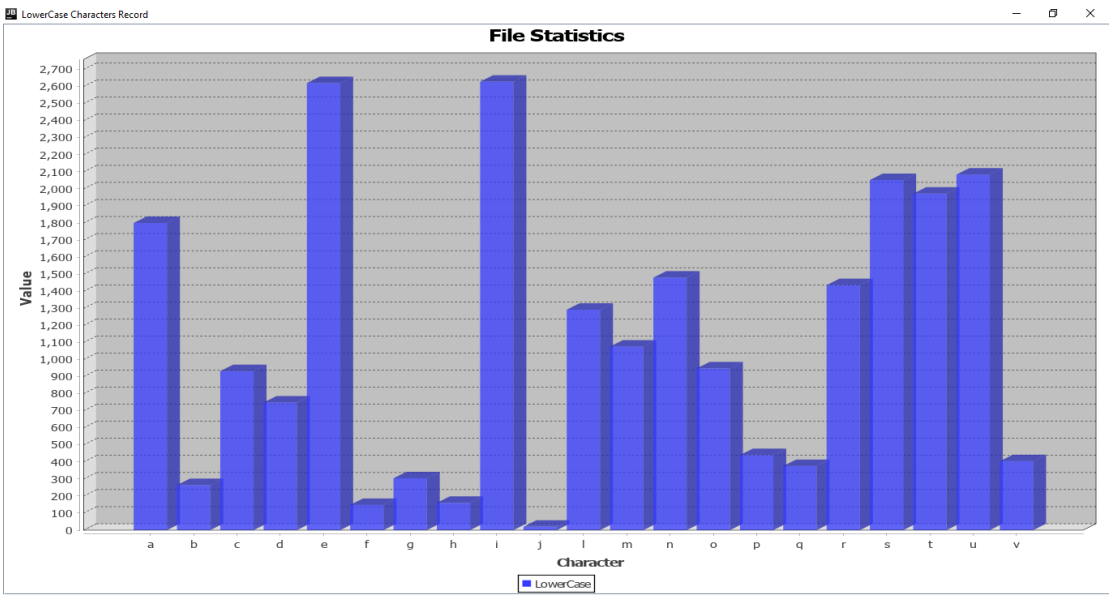
U nastavku su prikazana dva slučaja grafikona, grafikon koji obuhvata sve karaktere i grafikon koji obuhvata mala slova. Pored toga, prikazan je i slučaj pisanja statistike u fajl.
Grafikon koji obuhvata sve karaktere je prikazan na slici ispod.



Slika 11 Grafikon svih karaktera

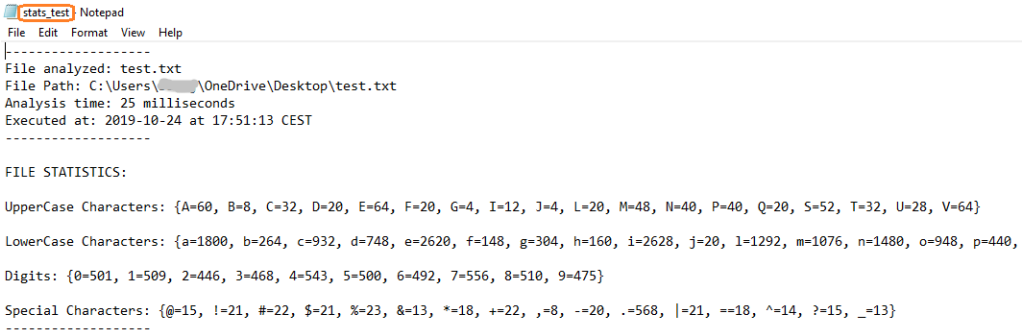
Kao što se može vidjeti na slici 11, grafikon je 3D tipa. Postoji mogućnost i 2D grafikona, ali je to stvar odabira razvojnog programera.

Grafikon je konstruisan na sljedeći način. Na dnu su prikazani svi karakteri koji se nalaze u trenutnom fajlu. Sa lijeve strane je prikazana vrijednost, odnosno koliko se puta svaki karakter ponavlja u fajlu. Vidimo da trenutno preovladavaju mala slova, što je i očekivano.
Na slici ispod je prikazan grafikon malih slova. Svi ostali grafikoni su veoma slični, samo je razlika u boji i podacima koje prikazuju.



Slika 12 Grafikon malih slova

Grafikon malih slova funkcioniše na isti način kao i svi ostali grafikoni. Na slici ispod se može vidjeti fajl statistike u kojeg su upisani svi podaci alalize. Ovo se također postiže tako što korisnik odabere ovu opciju.

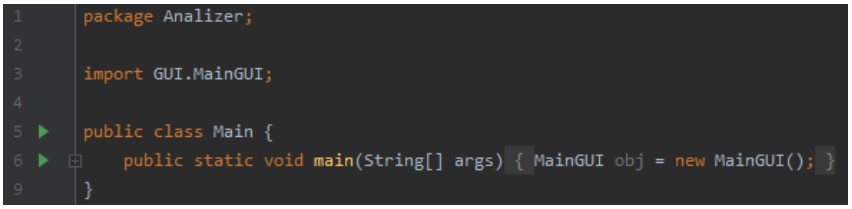


Slika 13 Fajl statistike

Bitno je napomenuti da kada korisnik odabere opciju upisivanja statistike u fajl, nakon upisivanja statistike, prikaže mu se prozor poruke da je statistika uspješno upisana. Isto tako, ako korisnik pri odabiru fajla izađe iz dijaloga, program neće prikazati prozor odabira grafikona nego će prikazati prozor da fajl nije odabran.

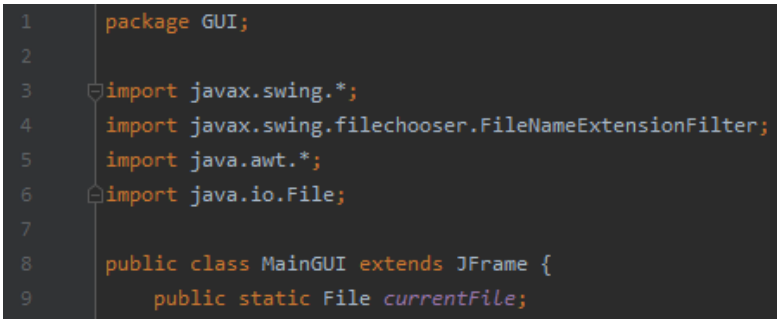
Kod aplikacije

Prilikom objašnjavanja koda aplikacije, kod svakog fajla neće biti objašnjen. Razlog tome je taj što kod nekih fajlova, kodovi su veoma slični ali su rezultati drugačiji. Razlika je samo u nekoliko linija koda.



Slika 14 Main fajl

Slika iznad prikazuje „Main.java“ fajl koji sadrži klasu Main i metodu main. Iz ove metode se pokreće program i sve što se nalazi u main metodi će prvo biti pokrenuti. Trenutno, u ovoj metodi se samo instancira objekat klase MainGUI koja će biti objašnjena u nastavku.



Slika 15 MainGUI fajl; prvi dio

MainGUI klasa se nalazi u paketu GUI. Slika iznad prikazuje importovanje (uključivanje) drugih biblioteka i klasa u ovaj fajl. Na liniji 8 vidimo da je klasa javna i da nasljeđuje klasu Jframe što joj daje mogućnost kreiranja prozora. Linija 9 definiše javni statični atribut „currentFile“ koji je tipa File i sadržavaće informacije o trenutnom fajlu.

Slika 16 MainGUI; drugi dio

Slika 16 prikazuje konstruktor klase MainGUI koji će se pozvati samim instanciranjem ove klase. U ovom konstruktoru se postavljaju parametri prozora kao što je izgled, vidljivost, veličina prozora, itd. Također, postavlja se i dugme koje preko lambda anonimne funkcije postavlja osluškivač na to dugme

i kada se to dugme pritisne ono će otvoriti dijalog odabira fajla. U try-catch bloku je obuhvaćeno spremanje trenutnog fajla u atribut „currentFile“. U slučaju da se ne proslijedi fajl, korisniku će biti prikazana poruka shodna tome.

```
38          JLabel label = new JLabel( text: "Please open input file!");
39
40          add(fileDialogButton);
41          add(label);
42      }
43  }
```

Slika 17 MainGUI; treći dio

Treći dio, tj. Slika 17 prikazuje posljednji dio trenutnog fajla. Postavlja se obična labela kao indikator šta korisnik treba da uradi i nakon toga se oba elementa dodaju u prozor pozivom funkcije add(). Ovaj prozor možete vidjeti na slici 8.

```
1   package GUI;
2
3   import Analizer.Writer;
4   import Charts.*;
5
6   import javax.swing.*;
7   import java.awt.*;
8   import java.io.IOException;
9
10  class ChartSelection extends JFrame {
11
12      ChartSelection() {
13          setLayout(new FlowLayout());
14          setVisible(true);
15          setSize( width: 400,  height: 100);
16          setResizable(false);
17
18          Toolkit toolkit = Toolkit.getDefaultToolkit();
19          Dimension screenSize = toolkit.getScreenSize();
20          int x = (screenSize.width - this.getWidth()) / 2;
21          int y = (screenSize.height - this.getHeight()) / 2;
22          this.setLocation(x, y);
23          setElements();
24      }
```

Slika 18 ChartSelection; prvi dio

U slučaju da korisnik odabere ispravan fajl, na ispravan način, biće mu prikazan prozor kao na slici 10. Slika 18 prikazuje prvi dio tog prozora. Vidimo da je na slici prikazano importovanje resursa, klasu koja nasljeđuje Jframe i njen konstruktor. Konstruktor je zadužen da postavi izgled, veličinu, centrira prozor na sredinu ekrana, i ostalo.

```
26      private void setElements() {
27          JButton fullChart = new JButton( text: "Full Chart");
28          JButton uCChart = new JButton( text: "UpperCase Chart");
29          JButton lCChart = new JButton( text: "LowerCase Chart");
30          JButton dChart = new JButton( text: "Digits Chart");
31          JButton sChart = new JButton( text: "Specials Chart");
32          JButton writeStats = new JButton( text: "Write File Stats");
33          add(fullChart);
34          add(uCChart);
35          add(lCChart);
36          add(dChart);
37          add(sChart);
38          add(writeStats);
```

Slika 19 ChartSelection; drugi dio

Slika 19 prikazuje privatnu metodu „setElements()“ koja nema parametre. Njena uloga u ovom dijelu je da kreira dugmad za odabir grafikona i dugme za ispis statistike u fajl i postavi ih u prozor.

```
fullChart.addActionListener(actionEvent -> {
    Charts.RecordChart rc = new RecordChart();
});

uCChart.addActionListener(actionEvent -> {
    UpperCaseChart ucChart = new UpperCaseChart();
});

lCChart.addActionListener(actionEvent -> {
    LowerCaseChart lcChart = new LowerCaseChart();
});

dChart.addActionListener(actionEvent -> {
    DigitsChart dchart = new DigitsChart();
});

sChart.addActionListener(actionEvent -> {
    SpecialsChart schart = new SpecialsChart();
});

writeStats.addActionListener(actionEvent -> {
    try {
        Analizer.Writer writer = new Writer(MainGUI.currentFile.toString());
        writer.write();
    } catch (IOException e) {
        JOptionPane.showMessageDialog( parentComponent: null,  message: "Internal error. Couldn't write to file");
    }

});
```

Slika 20 ChartSelection; treći dio

Slika iznad prikazuje drugi dio „setElements()“ metode. Ovaj dio je zadužen da uz pomoć lambda funkcija postavi osluškivače na svako dugme. Ako je u pitanju prikaz grafikona, lambda funkcija instancira objekat klase određenog grafikona kojeg je korisnik odabrao. U slučaju da je korisnik odabrao pisanje statistike u fajl. Tada se instancira objekat klase Writer i podaci se upisuju u fajl.

```
package Charts;

import GUI.MainGUI;
import Analizer.TxtAnalizer;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartFrame;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.CategoryPlot;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.renderer.category.BarRenderer;
import org.jfree.data.category.DefaultCategoryDataset;

import java.awt.*;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;
```

Slika 21 Grafikon; prvi dio

Vizuelni primjer grafikona je prikazan ranije, a slika iznad prikazuje prvi dio jednog od fajlova zaduženog za kreiranje grafikona. Na slici je prikazano importovanje različitih resursa.

```
public class DigitsChart {
    public DigitsChart() { showChart(); }

    private void showChart(){
        DefaultCategoryDataset dcd = new DefaultCategoryDataset();

        // Add dynamic data
        TxtAnalizer txtAnalizer = new TxtAnalizer();
        List<Map<Character, Integer>> mapList = txtAnalizer.analyze(MainGUI.currentFile.toString());
        if (!mapList.isEmpty()) {
            for (Map<Character, Integer> map : mapList) {
                Map.Entry<Character, Integer> entry = map.entrySet().iterator().next();
                Set characterSet = map.keySet();
                Iterator iterator = characterSet.iterator();
                if (Character.isDigit(entry.getKey())) {
                    while (iterator.hasNext()) {
                        Character key = (Character) iterator.next();
                        Integer value = map.get(key);
                        dcd.setValue(value,  rowKey: "Digit", key);
                    }
                }
            }
        }
```

Slika 22 Grafikon; drugi dio

Trenutni grafikon, čiji kod se nalazi na slici iznad, vezan je za grafikon brojeva. Svi grafikoni funkcionišu na veoma sličan način ali je rezultat drugačiji.

Slika iznad prikazuje klasu DigitsChart, njen konstruktor i metodu „showChart“. Ova metoda se poziva iz konstruktora a njen zadatak je sljedeći. Ona instancira objekat klase Analizer i poziva metodu „analyze“ koja ima zadatak da analizira fajl u potpunosti. Ova metoda će u nastavku biti objašnjena. Nakon što je fajl analiziran, svaki broj koji se nalazi u analiziranom fajlu dodaje u mapu koja se sastoji od seta karaktera i njihovih vrijednosti. Njihove vrijednosti znače koliko se puta taj karakter ponavljao u fajlu. Svaki karakter postavlja varijablu seta podataka koja je inicijalizovana kao prazna. Nju će grafikon koristiti da prikaže podatke.

```
45          // Creating chart, plot and modifying it's properties.
46          JFreeChart chart = ChartFactory.createBarChart3D( title: "File Statistics",  categoryAxisLabel: "Character",
47                  valueAxisLabel: "Value", dcd, PlotOrientation.VERTICAL,  legend: true,  tooltips: true,  urls: false);
48
49          CategoryPlot plot = chart.getCategoryPlot();
50          BarRenderer renderer = (BarRenderer) plot.getRenderer();
51          // set the color (r,g,b) or (r,g,b,a)
52          Color color = new Color( r: 255,  g: 166,  b: 39);
53          renderer.setSeriesPaint( series: 0, color);
54          plot.setRangeGridlinePaint(Color.black);
55          ChartFrame chartFrm = new ChartFrame( title: "Digits Record", chart,  scrollPane: true);
56          chartFrm.setVisible(true);
57          chartFrm.setSize( width: 640,  height: 490);
58      }
59  }
```

Slika 23 Grafikon; treći dio

Nakon što je analiza završena, kreira se grafikon i njegovi atributi. Zatim, svi podaci koji se nalaze u setu se dodaju u ovaj grafikon a zatim prikazuju na ekran.

```
1   package Analizer;
2
3   import javax.swing.*;
4   import java.io.File;
5   import java.io.FileNotFoundException;
6   import java.util.Scanner;
```

Slika 24 Reader; prvi dio

U slučaju da je korisnik odabrao upisivanje statistike u fajl. Poziva se kod iz Analizer paketa. Slika 24 prikazuje prvi dio koda Reader fajla.

```
8   class Reader {
9       private Scanner sc;
10
11      /**
12       * @param fileName String - name of the file. Also void method.
13       */
14      private void openFile(String fileName) {
15          try {
16              sc = new Scanner(new File(String.format("%s", fileName)));
17          }
18          catch (FileNotFoundException e) {
19              System.out.println(String.format("\n'%s' Not Found", fileName));
20              System.exit( status: 0);
21          }
22      }
```

Slika 25 Reader; drugi dio

Slika 25 prikazuje privatnu metodu „openFile“ koja prima String ime fajla kao argument. Ova metoda je zadužena da otvori postojeći fajl ili kreira fajl pod tim nazivom ako fajl ne postoji.

```
24      /**
25       * No parameters. Void method.
26       */
27      private void closeFile() { sc.close(); }
30
```

Slika 26 Reader, treći dio

Slika 26 prikazuje veoma jednostavnu metodu. Metoda „closeFile“ služi da zatvori fajl i objekat skenera koji je radio nad fajlom.

```
31      /**
32       * @param fileName String - name of the file.
33       * @return File content as String
34       */
35      String readFile(String fileName) {
36          String str = "";
37          openFile(fileName);
38          while (sc.hasNext()) {
39              str += sc.next();
40          }
41          closeFile();
42          if (str.length() == 0) {
43              JOptionPane.showMessageDialog( parentComponent: null,  message: "File is empty or not txt type.");
44              System.exit( status: 0);
45          }
46          return str;
47      }
48  }
```

Slika 27 Reader; četvrti dio

Metoda „readFile“ koja se nalazi na slici iznad, služi da analizira cijeli fajl i da vrati sadržaj fajla kao String. Funkcioniše na veoma jednostavan način tako što vrši iteraciju kroz while petlju sve dok postoji karakter u fajlu.

```
1   package Analizer;
2   import java.util.Map;
3   import java.util.List;
4
5   public interface AnalizerInterface {
6
7       public List<Map<Character, Integer>> analyze(String file);
8
9       private void mapFile(String file) {}
10
11      private void handleKey(Map<Character, Integer> map, Character key) {}
12  }
```

Slika 28 Analizer interfejs

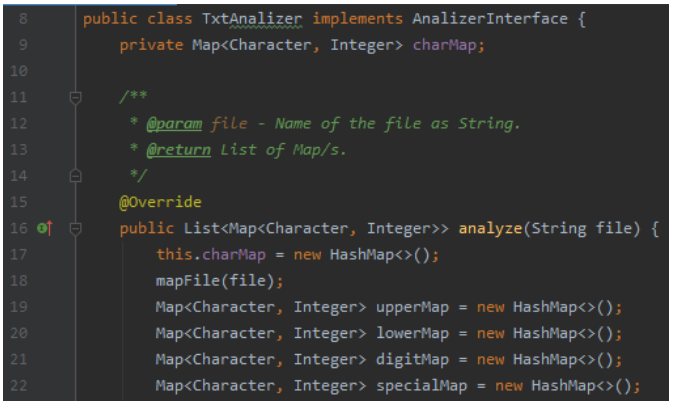
Slika 28 prikazuje interfejs kojeg će morati poštovati svaka klasa koja implementuje isti. To znači da svaku od metoda koje su navedene u ovom interfejsu, klasa koja implementuje isti, mora posjedovati tu metodu inače će se kompajler buniti da ugovor nije zadovoljen.

Interfejs služi kao ugovor svim klasama koje implementuju isti. Svaka klasa mora imati sve metode koje su navedene u interfejsu.

```
1   package Analizer;
2   import java.util.Map;
3   import java.util.HashMap;
4   import java.util.List;
5   import java.util.ArrayList;
```

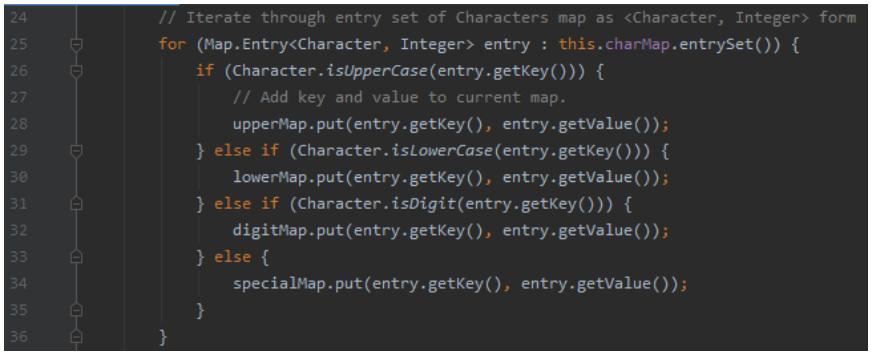
Slika 29 txtAnalizer; prvi dio

Slika iznad prikazuje prvi dio txtAnalizer fajla. U ovom dijelu se importuje Mapa, Lista i ostali potrebni resursi za rad metoda.



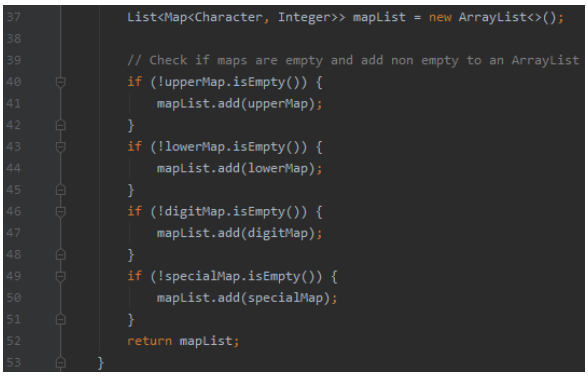
Slika 30 txtAnalizer; drugi dio

Na slici iznad je prikazana klasa TxtAnalizer. Ona ima atribut charMap koja je tipa Mape a njena polja su karakter kao ključ i cijeli broj kao vrijednost. Ova klasa implementira AnalizerInterface.
Ispod se nalazi metoda „analyze“ koja instancira charMap Mapu kao prazan HashMap objekat.
Zatim se poziva metoda *mapFile* koja će biti objašnjena u nastavku. Na kraju slike se instanciraju različite mape za tipove karaktera.



Slika 31 txtAnalizer; treći dio

U trećem dijelu ovog fajla, odnosno na slici 31, se provjerava u zajedničnoj mapi charMap koji su karakteri dostupni i isti karakteri se dodaju u posebne mape.

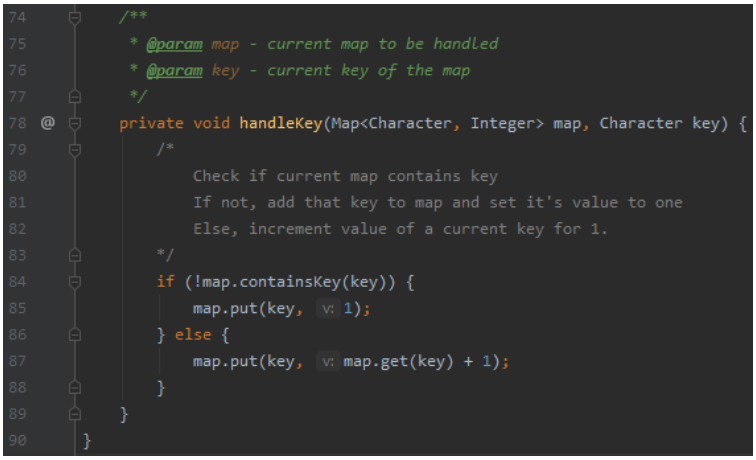


Slika 32 txtAnalizer; četvrti dio

Na kraju ove metode, instancira se lista mapa pod nazivom mapList. Zatim se svaka mapa koja nije prazna dodaje u ovu listu i na kraju se vraća ova lista mapa.

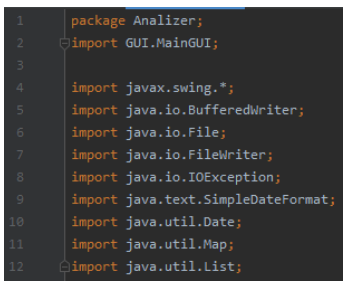
Slika 33 txtAnalizer; peti dio

Metoda mapFile je prikazana na slici iznad i njen zadatak je da pokupi podatke za koje je zadužena metoda readFile koja je ranije objašnjena i zatim da svaki karakter u dobijenom Stringu doda u zajedničku mapu karaktera.



Slika 34 txtAnalizer; šesti dio

Privatna metoda „handleKey“ je veoma jednostavna metoda koja služi da provjeri da li se ključ nalazi u trenutnoj mapi i uradi sljedeće nad njim. Ako se ključ, odnosno karakter ne nalazi u mapi onda će taj isti karakter dodati u mapu i njegovu vrijednost postaviti na 1. U slučaju da se isti ključ već nalazi u mapi, onda će samo njegovu vrijednost uvećati za 1.



Slika 35 Writer; prvi dio

Prvi dio Writer fajla je prikazan na slici iznad. Možemo vidjeti da se uključuju razni resursi kako iz drugih paketa projekta tako isto iz drugih paketa java razvojnog paketa.

```
14  public class Writer {
15
16      private String currentFileName;
17      private String fileName;
18      private BufferedWriter writer;
19
20      /**
21       * @param file - name of the current file.
22       * @throws IOException - if file is missing.
23       */
24      public Writer(String file) throws IOException {
25          currentFileName = file;
26          fileName = getFilePath(MainGUI.currentFile) + "\\stats_" + MainGUI.currentFile.getName();
27          writer = new BufferedWriter(new FileWriter(fileName));
28      }
```

Slika 36 Writer; drugi dio

Slika 36 prikazuje klasu Writer, njene atribute i konstruktor. Atributi su sljedeći:

- currentFileName – privatni atribut tipa String koji reprezentuje trenutno ime fajla.
- fileName – veoma sličan prvom atributu ali se koristi na drugačiji način. Ovaj dio koda nije dovoljno optimizovan. Optimizacija istog bi uklonila ovaj atribut i koristila samo prvi.
- Writer – privatni atribut tipa BufferedWriter koji služi za pisanje sadržaja u fajl. Nije instanciran trenutno.

Konstruktor zahtijeva argument naziv fajla kao String. Taj argument postavlja proslijeđenu vrijednost u atribut currentFilename.
fileName varijabla na osnovu njene putanje, prefiksa „stats_“ i naziva trenutnog fajla, postaje naziv fajla u kojeg će biti upisana statistika.
Također, inicijalizuje se writer.

```
30      /**
31       * @throws IOException - if file is missing.
32       * Method that writes statistics to a file.
33       */
34      public void write() throws IOException {
35          TxtAnalizer txtAnalizer = new TxtAnalizer();
36          long startTime = System.currentTimeMillis();
37          List<Map<Character, Integer>> mapList = txtAnalizer.analyze(currentFileName);
38          long endTime = System.currentTimeMillis();
39          long executionTime = endTime - startTime;
40
41          SimpleDateFormat formatter= new SimpleDateFormat( pattern: "yyyy-MM-dd 'at' HH:mm:ss z");
42          Date date = new Date(System.currentTimeMillis());
43
44          writeContent("--------------------");
45          writeContent("File analyzed: " + MainGUI.currentFile.getName());
46          writeContent("File Path: " + MainGUI.currentFile);
47          writeContent("Analysis time: " + executionTime + " milliseconds");
48          writeContent("Executed at: " + formatter.format(date));
49          writeContent("--------------------");
50          writeContent("\nFILE STATISTICS: " + "\n");
```

Slika 37 Writer; treći dio

U narednom dijelu, odnosno slici 37, nalazi se metoda *write()* koja će proslijeđeni sadržaj upisati u fajl.
U ovoj metodi se poziva analiza fajla koja je ranije objašnjena, mjeri se vrijeme u milisekundama potrebno za izvršenje analize i na slici se može vidjeti da se poziva metoda *writeContent()* koja služi da se proslijeđeni string upiše u fajl.

Upisuju se sljedeći podaci:

- Naziv analiziranog fajla,
- Sistemska putanja fajla,
- Vrijeme potrebno za analizu,
- Datum i vrijeme kad je analiza izvršena, i
- Statistika karaktera fajla.

```
52      if (!mapList.isEmpty()) {
53          for (Map<Character, Integer> map : mapList) {
54              Map.Entry<Character, Integer> entry = map.entrySet().iterator().next();
55              if (Character.isUpperCase(entry.getKey())) {
56                  writeContent("UpperCase Characters: " + map.toString() + "\n");
57              } else if (Character.isLowerCase(entry.getKey())) {
58                  writeContent("LowerCase Characters: " + map.toString() + "\n");
59              } else if (Character.isDigit(entry.getKey())) {
60                  writeContent("Digits: " + map.toString() + "\n");
61              } else {
62                  writeContent("Special Characters: " + map.toString());
63              }
64          }
65          writeContent("--------------------");
66      }
67
68      closeWriter();
69      JOptionPane.showMessageDialog( parentComponent: null,  message: "Statistics successfully written to a file!\n" +
70              "File is stored at the input file location.\n" +
71              "Name of the file is stats_" + MainGUI.currentFile.getName());
72  }
73
```

Slika 38 Writer; četvrti dio

U drugom dijelu metode *write*, što je prikazano na slici 38, upisuje se statistika karaktera fajla. Nakon toga se zatvara objekat BufferedWriter koji piše u fajl i korisniku se prikazuje poruka o uspješnom upisivanju statistike.

```
82      /**
83       * @throws IOException - if file is missing.
84       * Closes writer to prevent memory leak.
85       */
86      private void closeWriter() throws IOException {
87          this.writer.close();
88      }
```

Slika 39 Writer; peti dio

Slika 39 prikazuje veoma jednostavnu metodu *closeWriter*. Ova metoda objekat koji trenutno piše u fajl zatvara i onemogućuje njegovo dalje pisanje.

```
90      /**
91       * @param file - current file as File.
92       * @return - String representing current file path.
93       */
94  @   private String getFilePath(File file) { return file.getAbsoluteFile().getParent(); }
97  }
```

Slika 40 Writer; šesti dio

Privatna metoda *getFilePath* prikazana je na slici iznad i sadrži jednu liniju koda. Njen zadatak je da vrati sistemsku putanju trenutnog fajla. Tačnije, vraća direktorijum tog fajla a ne sami fajl.

Zaključak

Tema ovog seminarskog rada je bila prijedlog profesora. Nakon dogovora ključnih dijelova započeta je implementacija rješenja.
Tema je odlična, međutim, projekat je moguće mnogo proširiti.

Neke od stavki koje bi se mogle dodati su:

- Kreirati web aplikaciju koja bi mogla biti dostupna svima.
- Dodati analizu svih riječi u fajlu.
- Dodati analizu rečenica fajla.
- Improvizovati noviju statistiku fajlova.
- Kombinacija više fajlova u jednoj statistici, itd.

Projekat je veoma zanimljiv. Međutim, zbog opsega seminarskog rada, ovaj opseg projekta je sasvim dovoljan.

Izvori

- Članak o programskom jeziku Java: link.
- Biblioteka za kreiranje grafikona JfreeChart: link.
- Kompanija JetBrains i njihovi alati: link.
- GitHub repozitorijum trenutnog projekta: link.

Slike