Vježbe 10 - izrada grafičkog sučelja koriseći *Java Swing* i *JavaFX*

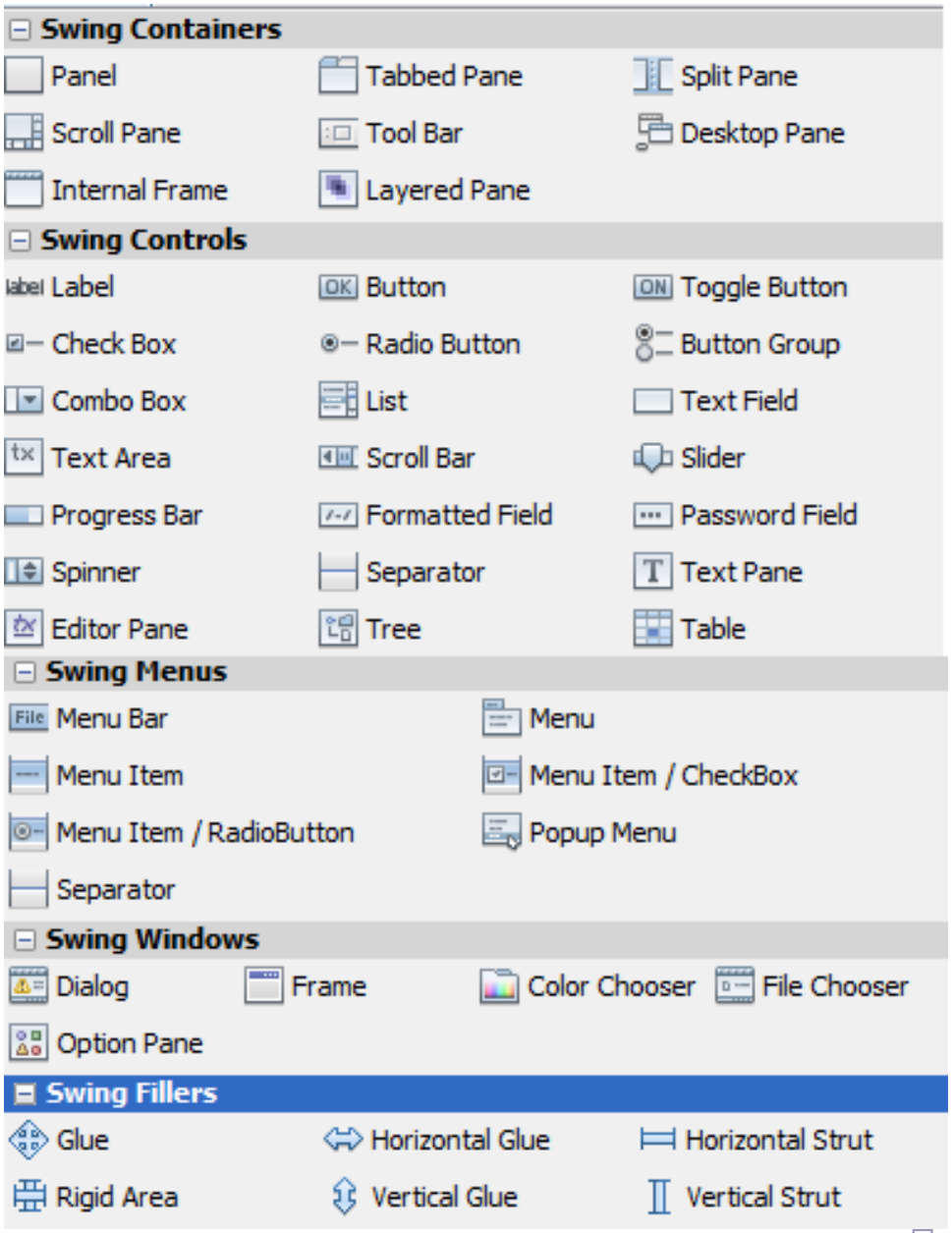
Matej Mihelčić

Prirodoslovno-matematički fakultet, Sveučilište u Zagrebu

matmih@math.hr

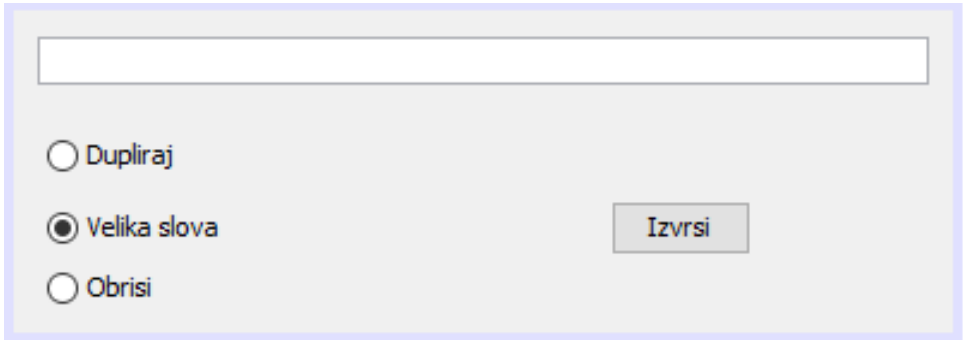
18. siječnja, 2023.

Java Swing


Swing Containers
Panel
Tabbed Pane
Split Pane
Scroll Pane
Tool Bar
Desktop Pane
Internal Frame
Layered Pane
Swing Controls
Label
Button
Toggle Button
Check Box
Radio Button
Button Group
Combo Box
List
Text Field
Text Area
Scroll Bar
Slider
Progress Bar
Formatted Field
Password Field
Spinner
Separator
Text Pane
Editor Pane
Tree
Table
Swing Menus
Menu Bar
Menu
Menu Item
Menu Item / CheckBox
Menu Item / RadioButton
Popup Menu
Separator
Swing Windows
Dialog
Frame
Color Chooser
File Chooser
Option Pane
Swing Fillers
Glue
Horizontal Glue
Horizontal Strut
Rigid Area
Vertical Glue
Vertical Strut

Zadatak 1

Kreirajte grafičko sučelje kao na slici **koristeći** pogled *dizajn Java Swing*-a. U **tekstualnu formu** na vrhu prozora upisujemo neki proizvoljni string od maksimalno 10 znakova (kada dostignemo maksimalnu duljinu **treba upozoriti korisnika na prekoračenje i ne dopustiti daljnje pisanje**). *Dupliraj*, *Velika slova i Obrisi* spadaju u grupu `Radio Button` i pridijeljeni su **istoj grupi** `Button Group` (to osigurava da **u svakom trenutku možemo izabrati samo jedan član grupe**). Opcija opisuje akciju koju treba izvršiti **pritiskom na gumb** `Izvrsi`. Novi **promijenjeni tekst** se treba **ponovo pojaviti u tekstualnoj formi**.



Nakon kreiranja nove forme, možemo definirati **naslov** (`title`), **preferiranu veličinu prozora ili ručno promjenom granica forme ili upisom veličine u polje** `Set preferred size`. Među postavkama samog okvira možemo uštimati i **operaciju koja se izvrši nakon pritiska gumba za zatvaranje** (standardno `EXIT_ON_CLOSE` - prekida izvršavanje aplikacije), ostale mogučnosti su `DO_NOTHING` (ne radi ništa), `HIDE` (sakrij prozor), `DISPOSE` (uništi trenutni prozor, međutim aplikacija i dalje radi). Mogu se definirati i **postavke automatskog fokusiranja na okvir** (standardno postavljeno na `true`) te **automatskog postavljanja prozora na vrh prozora** (standardno postavljeno na `false`). Okviru se može dodati i **ikona** (`icon image`) te postaviti **razne opcije lokacije, veličine prozora, dekoracije, tipa okvira** (standardno normalni okvir, može biti i skočni `POP_UP`, `UTILITY` - omogućuje ugradnju jednog okvira u drugi). Mijenjanjem opcije `cursor` možemo **promijeniti izgled kursora** pri radu sa okvirom.
Nakon konfiguriranja okvira, na okvir (`JFrame`) dodajemo neki od **spremnika kontrola** (npr. `Panel`).

Nad tim spremnikom možemo **definirati poredak kontrola** (tzv. `Layout`). To radimo pritiskom desnog gumba miša i **izabirom opcije** `SetLayout`. U primjeru se koristi standardni raspored `Free Design`. Nakon postavljanja spremnika i kontrola dodajemo **novi događaj** (eng. *event*) koji se **pokreće kada pritisnemo tipku na tipkovnici pri unosu znakova u tekstualnu formu**. Koristit ćemo ga da **gradimo string tekstualne forme** i prepoznamo kada je učitano više od 10 znakova. U tom trenutku ćemo **izbaciti prozor obavijesti i zanemariti daljnji unos**. Da bi to napravili u prozoru dizajna napravimo *desni klik na formu* $\rightarrow$ `events` $\rightarrow$ `key` $\rightarrow$ `keyTyped`. Automatski nam se generira funkcija za obradu događaja:

```
1 private void jTextField1KeyTyped(java.awt.event.KeyEvent
    evt) {  //dodamo kod
2     if(jTextField1.getText().length()>=10){
3         JOptionPane.showMessageDialog(this, "Prekoracen
    maksimalan broj znakova! Daljnje pisanje nema ucinka!","
    Upozorenje",JOptionPane.WARNING_MESSAGE);
4             evt.consume(); } //potrosimo unos znaka
```

Obrada događaja pritiska tipke nad tekstualnom formom.

`JOptionPane` ima tipove poruka `JOptionPane.{ERROR_MESSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE, QUESTION_MESSAGE, PLAIN_MESSAGE}`.

Da bi ispravno konfigurirali grupu gumbi za izbor opcija, kliknemo na svaki gumb i pod `buttonGroup` mu postavimo `buttonGroup1` (ime grupe `Button Group` koju smo postavili na kontrolu). **Izbor opcije možemo zabilježiti postavljanjem zastavice na neku vrijednost** (recimo $1, 2, 3$). Funkcija za obradu događaja izbora se automatski generira nakon dvostrukog klika na komponentu u pogledu dizajna.

```
1 int izbor = 2;
2  private void jRadioButton1ActionPerformed(java.awt.event.
     ActionEvent evt) { izbor = 1; }
3  private void jRadioButton2ActionPerformed(java.awt.event.
     ActionEvent evt) { izbor = 2; }
4  private void jRadioButton3ActionPerformed(java.awt.event.
     ActionEvent evt) { izbor = 3; }
```

Obrada događaja izabira gumba opcije iz grupe.

Slijedi implementiranje funkcionalnosti pritiska gumba `Izvrsi`.

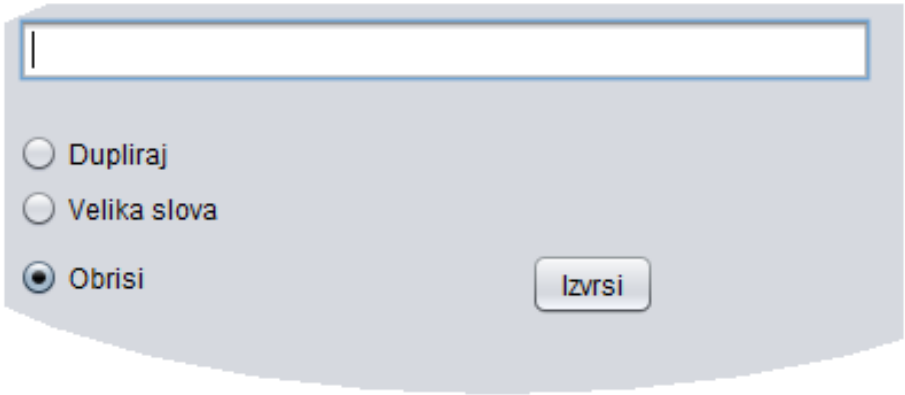
```
1  private void jButton1ActionPerformed(java.awt.event.
       ActionEvent evt) {
2          String textFieldString = jTextField1.getText();
3          if(izbor == 1){
4              textFieldString+=textFieldString;
5          }
6          else if(izbor == 2){
7              textFieldString = textFieldString.toUpperCase();
8          }
9          else if(izbor == 3){
10             textFieldString = "";
11         }
12
13         jTextField1.setText(textFieldString);
14 }
```

Obrada događaja pritiska gumba Izvrsi.

Klikom na formu i `Preview design` **možemo izabrati između nekoliko različitih izgleda okvira**. Možemo promijeniti i izgled forme:

```
1 public Zad1() {
2     initComponents(); //automatski generirani kod
3     this.setShape(new Ellipse2D.Float(-60f, -30f,550.0f,
      200.0f)); //kod za izmjenu oblika
4     this.setLocation(500, 300);  }
```

Promjena oblika forme.



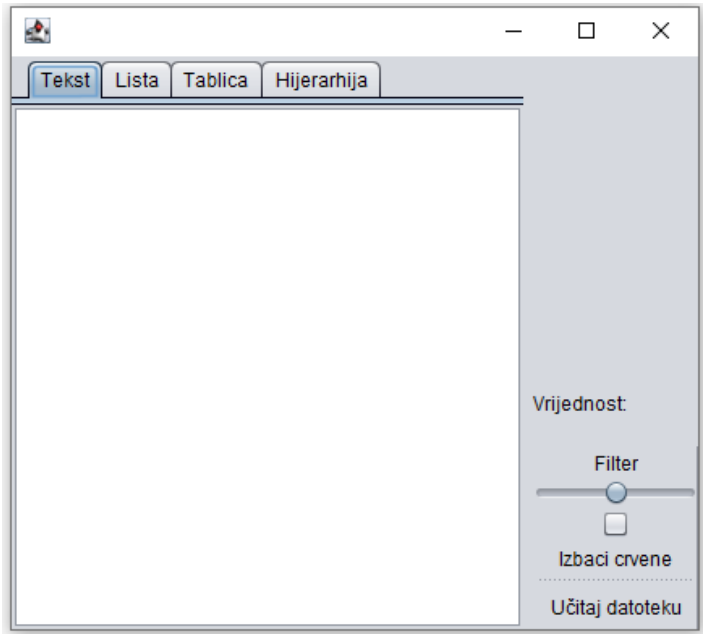
Da bi promjena oblika funkcionirala treba postaviti opciju `undecorated` na `true`. **Tim izborom gubimo mogućnost pomicanja prozora kao i gumbe za promjenu veličine, minimizaciju i zatvaranje**.

Zadatak 2

Kreirajte grafičko sučelje kao na slici **koristeći** pogled *dizajn Java Swing*-a. Sučelje treba imati 4 taba, *Tekst, Lista, Tablica, Hijerarhija*. U svakom tabu će prikazati **podatke**, **koji se učitavaju iz tekstualne datoteke**, na drugačiji način. Podaci su vezani uz cijene i boje raznih modela automobila. U prvom tabu se prikazuje **tekstualna reprezentacija datoteke** (`jTextArea`), u drugom **lista imena automobila i modela** (`jList`), u trećem **tablica** koja sadrži stupce *Proizvođač, Tip, Boja, Cijena(USD)* (`jTable`) a u četvrtom tabu se **proizvođači i njihovi modeli trebaju reprezentirati preko stabla, proizvođač je roditelj od modela** (`JTree`). Na desnoj strani se nalazi `jToolBar` koji sadrži labelu *Vrijednost*, labelu bez teksta, `jSlider`, `jCheckBox` i `jButton`. Na pritisak gumba *Učitaj datoteku* se aktivira komponenta `jFileChooser` koja **omogućava navigiranje i učitavanje odgovarajuće tekstualne datoteke**. Pri učitavanju se **ažurira maksimalna i minimalna vrijednost slider-a**.

Zadatak 2 - nastavak

Promjenom slider-a se iz tablice **filtriraju svi automobili s cijenom nižom ili jednakom vrijednosti na kliznoj komponenti**. Klizna komponenta mora raditi tako da **kada klizimo od viših vrijednosti prema nižima**, **automobili se vraćaju natrag u tablicu**. Izabirom **komponente označavanja** se **izbacuju svi automobili crvene boje** iz tablice. Ta promjena se mora dogoditi **odmah nakon selekcije** opcije.



Prvi korak je **poslagati sve navedene komponente na okvir i imenovati tabove na odgovarajući način**. Spremnik s tabovima se zove `TabbedPane`. Svaka komponenta navedena gore predstavlja **jedan tab** (automatski će se dodati nakon povlačenja komponente u spremnik). **Centralni dio aplikacije je obrada akcije gumba** *Učitaj datoteku* koji **treba ažurirati sve poglede kontejnera i sve interne varijable**. Pošto koristimo dosta kompleksne komponente kao što su `jTable`, `jTree` itd., za rješavanje ovog zadatka ćemo **koristiti modele** koji će **povezivati naše komponente s podacima**.

Za `jTextArea` koristimo **standarni model bez promjene**, kod `jTable` koristimo **standardni model ali postavljamo broj redaka na** 0 (klikom na model nam se otvara editor gdje možemo konfigurirati broj redaka), kod `jList` pod `model` izaberemo *user code* opciju i napišemo **programski kod** `new DefaultListModel()` (postavljamo klasu liste koja nam treba za daljnji rad). Za `jTree` također postavimo *user code* i dodamo: `new DefaultTreeModel(new DefaultMutableTreeNode("root"))`.

Od pomoćnih podataka nam trebaju:

```
1 boolean izbaciCrvene = false;
2 String imeDatoteke="";
3 private  JFileChooser jchooser1 = new JFileChooser();
4 private javax.swing.JTable jTable2Copy;
```

Pomoćne instance i varijable.

Na komponentu za selekciju datoteka **dodamo filter** koji omogućava otvaranje **samo datoteka s ekstenzijom** `.txt`. To **radimo definiranjem tipa filtera** `FileNameExtensionFilter filter = new FileNameExtensionFilter("TEXT FILES", "txt", "text");` i **pridjeljivanjem filtera komponenti za selekciju i otvaranje datoteka** `jchooser1.setFileFilter(filter);`. Postoje dvije vrste komponente za otvaranje datoteka: a) `showSaveDialog` (koji služi za spremanje, ima *save* gumb) i b) `showOpenDialog` (koji služi za otvaranje, ima *open* gumb). Nakon klika na gumb *Učitaj datoteku izvodi se sljedeća akcija:*

```
1  private void jButton1ActionPerformed(java.awt.event.
       ActionEvent evt) {
2      int returnVal = jchooser1.showOpenDialog((Component)evt.
       getSource()); //otvaramo dijalog za otvaranje datoteke
3      if (returnVal == JFileChooser.APPROVE_OPTION) {//
       ispitujemo je li korisnik izabrao otvaranje
4          File file = jchooser1.getSelectedFile(); //dohvacamo
        putanju do datoteke
5          try {
6              imeDatoteke = file.toString(); //spremamo putanju
        u string
7          } catch (Exception ex) {} //ignoriramo obradu
       iznimke
8
9  //format datoteke -> proizvodac automobila tip boja cijena
10         String cijelaDatoteka = "", linija = "";
11         File input = new File(imeDatoteke);
12         int brojac = 0;
13         Path p = Paths.get(input.getAbsolutePath());
```

Akcija gumba.

```
1  ArrayList<String> alist = new ArrayList<>();
2  DefaultListModel dlm = (DefaultListModel)jList2.getModel();
3  DefaultTreeModel tmodel = (DefaultTreeModel) jTree2.
       getModel();
4  tmodel.setRoot(new DefaultMutableTreeNode("Automobili"));
5  DefaultMutableTreeNode root = (DefaultMutableTreeNode)
       tmodel.getRoot();//dohvacamo sve potrebne modele
6  jTree2.getSelectionModel().setSelectionMode(
       TreeSelectionModel.SINGLE_TREE_SELECTION);
7  HashSet<String> ubaceniProizvodaci = new HashSet<>();
8  int ubaceno=0;
9  double minCijena = Double.MAX_VALUE, maxCijena = Double.
       MIN_VALUE;
10 String zaglavlje[] = null; //potrebne varijable za
       ucitavanje podataka u komponente
11
12         try{
13             BufferedReader citac = Files.newBufferedReader(p
       ,StandardCharsets.UTF_8);
```

Akcija gumba.

```
while((linija = citac.readLine())!=null){
                cijelaDatoteka+=linija+"\n";
                if(brojac == 0){
                    cijelaDatoteka+="\n";
                     zaglavlje= linija.split(" ");
                   DefaultTableModel table_model=new
    DefaultTableModel(zaglavlje,0);
                    jTable2=new JTable(table_model);//
    dodajemo pravo zaglavlje tablici (ono iz datoteke)
                  jScrollPane6.setViewportView(jTable2); //
    azuriraj izgled kontrole na okviru (koristi se kod
    kontrola koje imaju scrole bar, zapravo azurira pogled
    na tablicu koji moze biti nepotpun)
                    brojac = 1;
                }
                else{ //racunamo maks. i min. cijenu
                    String komponente[] = linija.split(" ");
          if(Double.parseDouble(komponente[3])>maxCijena)
             maxCijena = Double.parseDouble(komponente[3]);
```

Akcija gumba.

```
1  if(Double.parseDouble(komponente[3])<minCijena)
2              minCijena = Double.parseDouble(komponente[3]);
3          DefaultTableModel model = (DefaultTableModel)
     jTable2.getModel();//dohvacamo model tablice
4             model.addRow(komponente);//dodajemo redak u tablicu
5                //alist.add(komponente[0]+" "+komponente[1]);//
     moze i koristenjem pomocne liste
6          dlm.addElement(komponente[0]+" "+komponente[1]); //
     dodajemo element u listu
7                      if(insertedManufacturers.contains(
     komponente[0])){ //gradimo stablo
8                           DefaultMutableTreeNode t=null;
9
10                    for(int i=0;i<root.getChildCount();i++){
11                     if(((String((DefaultMutableTreeNode)root.
     getChildAt(i)).getUserObject()).equals(komponente[0])){
     //trazimo cvor proizvodaca u stablu
12               t = (DefaultMutableTreeNode)root.getChildAt(i);
     // dohvacamo ga
```

Akcija gumba.

```
1                                   break; } }
2                             int cc=tmodel.getChildCount(t);
3 tmodel.insertNodeInto(new DefaultMutableTreeNode(komponente
      [1]), t,cc); //dodajemo dijete proizvodaca = tip modela
4 tmodel.reload((DefaultMutableTreeNode)tmodel.getRoot());//
      ponovo ucitamo struktru stabla radi iscrtavanja
5                           }
6                           else{
7     DefaultMutableTreeNode t = new DefaultMutableTreeNode(
      komponente[0]);// cvor proizvodaca ne postoji u stablu,
      treba ga dodati
8      tmodel.insertNodeInto(t, root, root.getChildCount());
9      tmodel.reload((DefaultMutableTreeNode)tmodel.getRoot());
       //ubacujmo proizvodaca i osvjezimo stablo
10
11     int cc=tmodel.getChildCount(t);
12     tmodel.insertNodeInto(new DefaultMutableTreeNode(
      komponente[1]), t,cc);// ubacujemo model automobila
```

Akcija gumba.

```
1  tmodel.reload((DefaultMutableTreeNode)tmodel.getRoot());
2     insertedManufacturers.add(komponente[0]); //azuriramo
   stablo i zapamtimo da je proizvodac u stablu
3                   } } }
4
5              DefaultTableModel model = (DefaultTableModel)
   jTable2.getModel();
6              DefaultTableModel model1=new DefaultTableModel(
   zaglavlje,0);
7              jTable2Copy=new JTable(model1);
8             //kopiramo tablicu u pomocnu tablicu zbog
   filtriranja
9             for(int i=0;i<model.getRowCount();i++){
10                 Vector<Object> vec = new Vector<>();
11                vec.add(model.getValueAt(i, 0));
12                vec.add(model.getValueAt(i, 1));
13                vec.add(model.getValueAt(i, 2));
14                vec.add(model.getValueAt(i, 3));
15                model1.addRow(vec); }
```

Akcija gumba.

```
1
2               jSlider1.setMaximum(((int)maxCijena)+1);
3               jSlider1.setMinimum((int)minCijena);// postavi
    minimum i maksimum slidera
4               jTree2.setModel(tmodel); //dodaj model stablu
5                ((DefaultTreeModel)jTree2.getModel()).reload();
6               jScrollPane5.setViewportView(jList2);
7               jTextArea1.setText(cijelaDatoteka);
8               jScrollPane7.setViewportView(jTree2);//nacrtaj
    komponente
9            }
10           catch(IOException e){  e.printStackTrace(); } }
11    else { }  } //kraj akcije gumba
```

Akcija gumba.

Nakon implementacije akcije gumba, trebamo implementirati akciju promjene klizne komponente.

```
1  private void jSlider1StateChanged(javax.swing.event.
       ChangeEvent evt) {
2          JSlider s = (JSlider) evt.getSource();
3          jLabel3.setText(s.getValue()+""); //ispisujemo
       trenutnu vrijednost klizne komponente kroz praznu labelu
4          DefaultTableModel model=(DefaultTableModel)jTable2.
       getModel();
5          DefaultTableModel model1=(DefaultTableModel)
       jTable2Copy.getModel();//dohvacamo model tablice i
       kopije
6
7          for(int i=model.getRowCount()-1;i>=0;i--){
8             model.removeRow(i); } //izbacujemo sve elemente
       iz tablice
9
10            for(int i=0;i<model1.getRowCount();i++){ //
       dodajemo samo potrebne elemente
11                Vector<Object> vec = new Vector<>();
```

Akcija promjene klizne komponente.

```
1  if(Double.parseDouble(model1.getValueAt(i, 3)+"")>=s.
       getValue()){
2
3      if(izbaciCrvene == false || !model1.getValueAt(i, 2).
       equals("crveni")){
4                   vec.add(model1.getValueAt(i, 0));
5                   vec.add(model1.getValueAt(i, 1));
6                   vec.add(model1.getValueAt(i, 2));
7                   vec.add(model1.getValueAt(i, 3));
8                   model.addRow(vec);
9                   }      }
10                          }     } //nakon svakog poziva ove
       akcije (svake promjene klizne komponente), tablica se
       ponovo preracuna i nacrta
```

Akcija promjene klizne komponente.

Još nam preostaje implementacija komponente označavanja.

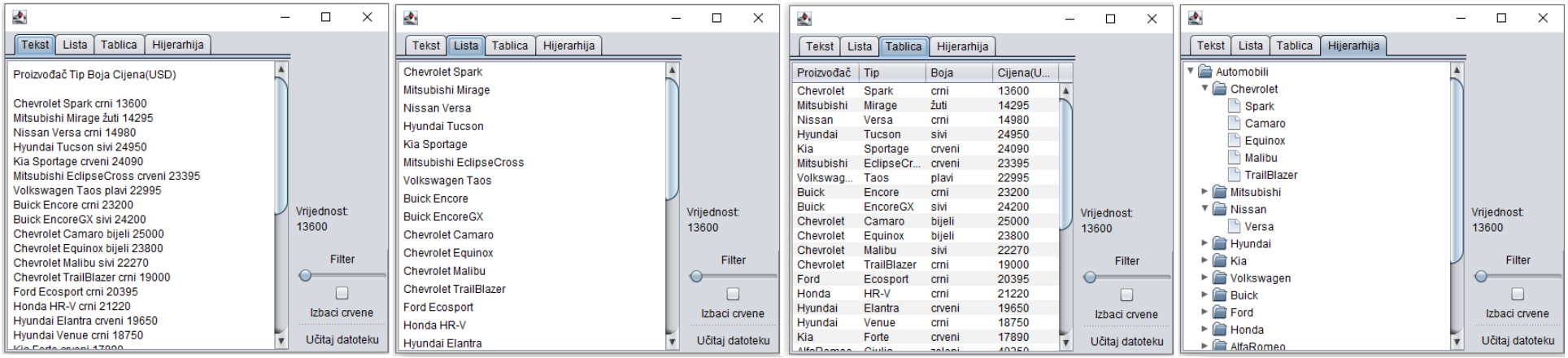
```
1 private void jCheckBox1ActionPerformed(java.awt.event.
    ActionEvent evt) {
2       if(jCheckBox1.isSelected())
3           izbaciCrvene = true; //zabiljezimo oznacavanje
4       else izbaciCrvene = false; //zabiljezimo
    odznacavanje
5
6       javax.swing.event.ChangeEvent chg = new javax.swing.
    event.ChangeEvent(jSlider1);//kreiramo dogadaj promjene
    klizne komponente
7        this.jSlider1StateChanged(chg);//izvrsimo akciju klizne
     komponente koja ce uvaziti promjenu u komponenti
    oznacavanja
8    }
```

Akcija selekcije/deselekcije komponente označavanja.

```
1 public static void main(String args[]) {
2 /*automatski generirani kod za Look and Feel*/
3 //poziv glavnog programa
4         java.awt.EventQueue.invokeLater(new Runnable() {
5             public void run() {
6                 new Zad2().setVisible(true); //poziv dretve
    koja odgovara na dogadaje
7             }
8         }); //ova dretva se izvodi sve dok ne zagasimo
    aplikaciju (standardno pritiskom na gumb zatvaranja zbog
     EXIT_ON_CLOSE opcije)
9     }
```

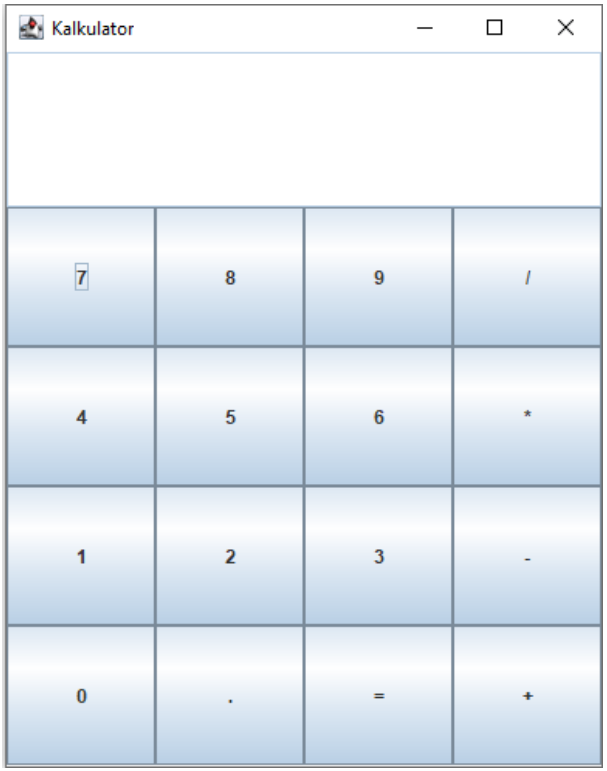
Glavni program Java Swing aplikacije.

Izgled korisničkog sučelja uz učitane podatke.



Zadatak 3.

Kreirajte grafičko sučelje i funkcionalnost **jednostavnog kalkulatora** kao na slici **bez korištenja** pogleda *dizajn Java Swing*-a. Kalkulator pamti zadnju izabranu operaciju i trenutni rezultat. Pri pozivu operanda izvrši operaciju nad trenutnim rezultatom.



Pošto u rješenju ovoga zadatka **ne koristimo** dizajn pogled, trebamo sami dizajnirati okvir, spremnik i postaviti odgovarajuće komponente na spremnik. Prvi korak će biti dizajn spremnika komponenti koji ćemo potom dodati na okvir aplikacije. Spremnik implementiramo klasom koja nasljeđuje klasu `JPanel`.

```
1  class SpremnikKalkulatora extends JPanel{
2    private JPanel spremnik; //jos jedan spremnik za sve tipke
3    private JTextField ekran; // ekran kalkulatora
4
5    public SpremnikKalkulatora(){
6      setLayout(new BorderLayout());//razmjestaj BorderLayout
       ima sjeverni, juzni i centralni dio
7
8              ekran = new JTextField();
9              ekran.setSize(400, 100);
10             ekran.setPreferredSize(new Dimension(400,100));
11             ekran.setEnabled(false); //ne dopustamo direktno
       pisanje po ekranu
```

Dizajn spremnika komponenti kalkulatora.

```
1 ekran.setFont(ekran.getFont().deriveFont(Font.BOLD, 28f));
    //povecamo velicinu znakova i otisnemo
2       add(ekran,BorderLayout.NORTH); //dodamo komponentu u
    spremnik na sjeverni dio
3     spremnik = new JPanel(); // kreiramo pomocni spremnik
4     spremnik.setLayout(new GridLayout(4,4)); //namjestimo
    mu razmjestaj mreze, sluzi za organiziranje ostalih
    tipki kalkulatora
5             }   }
```

Dizajn spremnika komponenti kalkulatora.

Nakon što smo dizajnirali generalni izgled spremnika kalkulatora, trebamo dodati gumbe na pomočni spremnik u rasporedu mreže. To ćemo napraviti pomoćnom funkcijom `dodajGumb`. Ta funkcija će primati oznaku gumba i slušača (**klasu koja implementira funkcionalnost osluškivanja akcije zadane komponente i izvrši odgovarajuću akciju**). Slušač koji odgovara gumbima $0-9$ i . treba samo zapisati znak na ekran, dok slušač za operaciju treba i pozvati funkciju za računanje, te ažurirati logičko stanje aplikacije.

```
1  private class AkcijaPisanja implements ActionListener{
2              @Override
3              public void actionPerformed(ActionEvent event){
4                  String unos = event.getActionCommand();//
       dohvacamo string pozvane kontrole
5               if(start){ ekran.setText(""); start = false; }
       //ukoliko smo u ciklusu novog racunanja, ponistimo
       prozor i postavimo da je ciklus zapoceo
6                  ekran.setText(ekran.getText()+unos); //
       zapisemo odgovarajucu vrijednost na ekran
7              }   }
8
9   private void dodajGumb(String oznaka, ActionListener slusac
       ){// dodavanje gumba na spremnik
10             JButton gumb = new JButton(oznaka);
11             gumb.addActionListener(slusac);
12             spremnik.add(gumb);
13         }
```

Dizajn akcija i funkcije dodajGumb.

Za uspješnu implementaciju akcije operacije trebamo u klasi SpremnikKalkulatora definirati još tri varijable: `private double rezultat = 0.0; private String zadnjaOperacija = "="; private boolean start = true;`.

```
1  private class AkcijaOperacije implements ActionListener{
2     @Override
3     public void actionPerformed(ActionEvent event){
4       String operacija = event.getActionCommand();
5                   //dohvacamo operaciju
6       if(start){//- se moze nalaziti i na pocetku
7          if(operacija.equals("-")){ ekran.setText(operacija);
        start = false; }
8          else  zadnjaOperacija = operacija; }
9       else{ //racunanje operacije
10            racunaj(Double.parseDouble(ekran.getText()));
11            zadnjaOperacija = operacija;//pamtimo zadnju
     operaciju, koristimo u iducem koraku
12            start = true; } } }
```

Dizajn akcije operacija.

Funkcija `racunaj` primjenjuje operaciju na novo uneseni broj i prethodni rezultat dok kod prvog unosa operacije postavlja `rezultat` na prvi učitani broj (inicijalno `zadnjaOperacija == "="`).

```
1  public void racunaj(double x){
2          if(zadnjaOperacija.equals("+")) rezultat+=x;
3          else if(zadnjaOperacija.equals("-")) rezultat-=x;
4          else if(zadnjaOperacija.equals("*")) rezultat*=x;
5          else if(zadnjaOperacija.equals("/")) rezultat/=x;
6          else if(zadnjaOperacija.equals("=")) rezultat = x;
7          ekran.setText(""+rezultat);
8      }
```

Funkcija računaj.

Sljedeći korak je dodavanje gumbi u spremnik. Nakon kreiranja instanci slušača i definiranja rasporeda spremnika `spremnik`, dodajemo kod za dodavanje ostalih kontrola kalkulatora u konstruktor `SpremnikKalkulatora()`.

```
1  ActionListener pisanje = new AkcijaPisanja();
2  ActionListener naredba = new AkcijaOperacije();
3
4  dodajGumb("7",pisanje); dodajGumb("8",pisanje);
5  dodajGumb("9",pisanje); dodajGumb("/",naredba);
6  dodajGumb("4",pisanje); dodajGumb("5",pisanje);
7  dodajGumb("6",pisanje); dodajGumb("*",naredba);
8  dodajGumb("1",pisanje); dodajGumb("2",pisanje);
9  dodajGumb("3",pisanje); dodajGumb("-",naredba);
10 dodajGumb("0",pisanje); dodajGumb(".",pisanje);
11 dodajGumb("=",naredba); dodajGumb("+",naredba);
12 add(spremnik, BorderLayout.CENTER);
```

Dodavanje gumbi spremniku.

Definiramo klasu okvira koji nasljeđuje klasu JFrame.

```
class OkvirKalkulatora extends JFrame{
     public OkvirKalkulatora(){
          setTitle("Kalkulator");
          SpremnikKalkulatora spremnik = new
    SpremnikKalkulatora();
          spremnik.setSize(400, 400);
          spremnik.setPreferredSize(new Dimension(400,400));
          add(spremnik);
          pack();//postavlja velicinu okvira tako da su sve
    komponente na preferiranoj velicini ili vece
        } }
//preferirani nacin je koristenjem awt reda dogadaja
public class Aplikacija {  //glavna klasa i glavna funkcija
 public static void main(String args[]){
  OkvirKalkulatora okvir  = new OkvirKalkulatora();
  okvir.setSize(400,500);
  okvir.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  okvir.setVisible(true); } }//OK za jednostavne aplikacije
```

Dizajn okvira aplikacije.

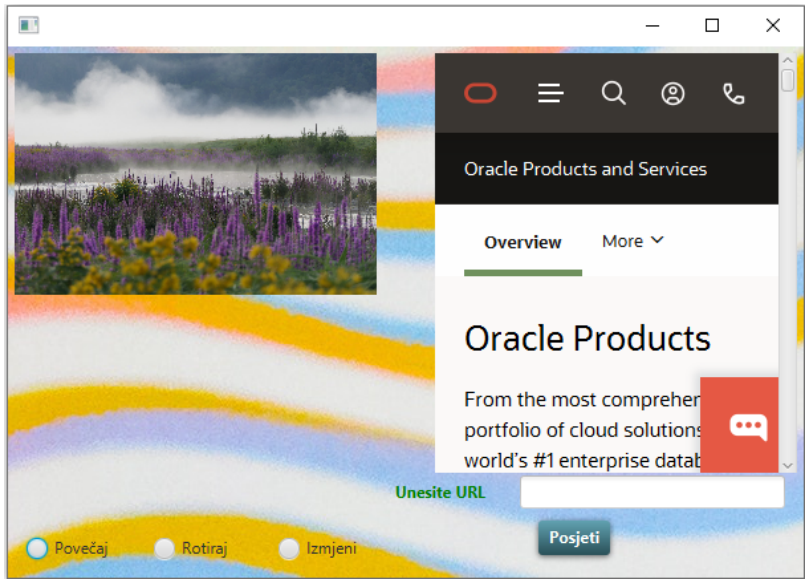
Zadatak - DZ

Proširite kalkulator dodatnim gumbima (%, CE, C, D - brisanje znaka). **Omogućite unos brojeva i operacija s tipkovnice**. Napravite da pritiskom tipke ***enter*** dobijete rezultat računanja operacije.


Accordion
Accordion (empty)
AnchorPane
BorderPane
ButtonBar (FX8)
DialogPane (empty) (FX8)
DialogPane (FX8)
FlowPane
GridPane
HBox
Pane
ScrollPane
ScrollPane (empty)
SplitPane (empty)
SplitPane (horizontal)
SplitPane (vertical)
StackPane
Tab
TabPane
TabPane (empty)
TextFlow (FX8)
TilePane
TitledPane
TitledPane (empty)
ToolBar
VBox
Button
CheckBox
ChoiceBox
ColorPicker
ComboBox
DatePicker (FX8)
HTMLEditor
Hyperlink
ImageView
Label
ListView
MediaView
MenuBar
MenuButton
Pagination
PasswordField
ProgressBar
ProgressIndicator
RadioButton
ScrollBar (horizontal)
ScrollBar (vertical)
Separator (horizontal)
Separator (vertical)
Slider (horizontal)
Slider (vertical)
Spinner (FX8)
SplitMenuButton
TableColumn
TableView
TextArea
TextField
ToggleButton
TreeTableColumn (FX8)
TreeTableView (FX8)
TreeView
WebView

Zadatak 4

Kreirajte grafičko sučelje kao na slici **korištenjem** *SceneBuilder*-a i *JavaFX*-a. Sučelje sadrži sliku koja se može: a) povećati za 10%, b) rotirati, c) izmijeniti (izmijeni kontrast, ton boje, svjetlinu, čistoću boje) pritiskom na odgovarajući gumb izbora. U desnom dijelu aplikacije je web preglednik sa odgovarajućom tekstualnom formom i gumbom koji služe za unos *url*-a web adrese. Implementirajte odgovarajuće funkcionalnosti tih *JavaFX* komponenti.



Napravimo novu *Java with Ant* klasu koja će reprezentirati naš novi *JavaFX* program, dodajmo u projekt sve ovisnosti (*JavaFX* biblioteku). Kreirajmo praznu `.fxml` datoteku i odgovarajući kontroler the CSS dokument (zadnja dva se kreiraju automatski).

Komponente **rasporedimo** u *SceneBuilder*-u te definiramo sve potrebne funkcije akcije, to su `radio{1,2,3}Handler` i `posjetiHandler` (gdje je posjeti ime gumba za dohvaćanje web stranice). Glavni spremnik komponenti je `AnchorPane`. Funkcije za obradu akcija **specificiramo** u `Code` dijelu svake komponente. Sljedeći korak je pod svojstvima uređenja (`Layout`) odrediti neke vrijednosti sidrenja (`Anchor`) jer će nam to omogučiti pravilno skaliranje komponenti pri maksimizaciji prozora.

Definirajmo funkcionalnost kontrolera:

```
1  public class Zad4FXMLController implements Initializable {
2      @FXML //anotacija FXML omogucava ubacivanje (injekciju)
       vrijednosti iz .FXML dokumenta u reference u nasem
       programu
3   RadioButton radio1, radio2, radio3; //nakon definicije,
       gumbi nisu povezani u cijelinu, kao i kod Swinga, treba
       ih grupirati.
4      @FXML
5      WebView browser;
6      @FXML
7      Button posjeti;
8      @FXML
9      ImageView slika;
10     ToggleGroup group = new ToggleGroup();//ova grupa se
       koristi za grupiranje elemenata klase RadioButton.
11     @FXML
12     TextField url;
13     WebEngine webEngine;
```

Dizajn kontrolera aplikacije.

```
1    @FXML
2            public void posjetiHandler(ActionEvent event) {
3                String hrl = url.getText();
4                if(!hrl.contains("http://"))
5                    hrl="http://"+hrl;
6                  this.webEngine.load(hrl);
7            }
8
9    @FXML
10           public void radio1Handler(ActionEvent event) {
11               slika.setScaleX(1.1); slika.setScaleY(1.1);
12           }
13
14    @FXML
15           public void radio2Handler(ActionEvent event) {
16               slika.setScaleX(1); slika.setScaleY(1);
17               slika.setRotate(slika.getRotate() + 90);
18           }
```

Dizajn kontrolera aplikacije.

```
1         @FXML public void radio3Handler(ActionEvent event) {
2              slika.setScaleX(1); slika.setScaleY(1);
3               ColorAdjust namjestiBoju = new ColorAdjust();
4
5    namjestiBoju.setContrast(0.4);      //Namjesti kontrast
6    namjestiBoju.setHue(-0.05);    //Namjesti ton boje
7    namjestiBoju.setBrightness(0.9);  //Namjesti svjetlinu
8    namjestiBoju.setSaturation(0.8);  //Namjesti cistocu boje
9    slika.setEffect(namjestiBoju);  //Primjeni na sliku
10            }
11
12    @Override
13    public void initialize(URL url, ResourceBundle rb) {
14       radio1.setToggleGroup(group);//grupiramo gumbe
15       radio2.setToggleGroup(group);
16       radio3.setToggleGroup(group);
17       webEngine = browser.getEngine(); //logika preglednika
18       this.webEngine.load("http://www.oracle.com/us/
   products/index.html"); }     }
```

Dizajn kontrolera aplikacije.

Nakon definicije kontrolera, definiramo glavnu klasu aplikacije:

```
1  public class Zad4 extends Application{
2      @Override
3    public void start(Stage primaryStage) throws IOException{
4          Parent root = FXMLLoader.load(getClass().getResource
     ("/javafx/Zad4FXML.fxml")); //citamo glavni cvor
     aplikacije (koristi se refleksija)
5          Scene scene = new Scene(root); //stvaramo scenu
6           scene.getStylesheets().add
7          (getClass().getResource("zad4fxml.css").
     toExternalForm()); //dodajemo CSS stil
8          primaryStage.setScene(scene);
9          primaryStage.initStyle(StageStyle.DECORATED); //
     dopustamo maksimizaciju, minimizaciju i slicno
10         primaryStage.show(); //prikazujemo pozornicu
11     }
12
13     public static void main(String[] args) {
14         launch(args); } }
```

Dizajn glavne klase aplikacije

Dizajn stila:

```
.root {
     -fx-background-image: url("pozadina.jpg");
     -fx-effect: dropshadow( gaussian , rgba
   (255,255,255,0.5) , 0,0,0,1 ); }

.button{
    -fx-text-fill: white;
    -fx-font-family: "Arial Narrow";
    -fx-font-weight: bold;
    -fx-background-color: linear-gradient(#61a2b1, #2A5058);
    -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6)
     , 5, 0.0 , 0 , 1 ); }

.label{
    -fx-text-fill: green;
    -fx-font-family: "Arial Narrow";
    -fx-font-weight: bold; }
```

Dizajn stila aplikacije

Zadatak 5

Kreirajte animaciju koja rotira, translatira i skalira pravokutnike različitih dimenzija i boja po ekranu dimenzija 800×600 u trajanju 100s. Dodajte gumbe za ubrzavanje, usporavanje i izvođenje animacije u obrnutom redosljedu.

```
1 public class Animacija1 extends Application{
2     double rateIncrease = 1.0;
3     @Override
4  public void start(Stage primaryStage) {
5         Group root = new Group();  //koristimo Group node kao
      korijen
6         Scene scene = new Scene(root, 800, 600, Color.BLACK);
7         primaryStage.setScene(scene);
8         Group rectangles = new Group();
```

Implementacija animacije u *JavaFX*-u.

```
1        for (int i = 0; i < 50; i++) {
2            Rectangle rectangle = new Rectangle(100 + random
    ()*50,50+random()*10, Color.web("white", 0.05));
3            rectangle.setStrokeType(StrokeType.OUTSIDE);
4            rectangle.setStroke(Color.web("white", 0.16));
5            rectangle.setStrokeWidth(4);
6            rectangles.getChildren().add(rectangle); } //
    staramo 50 cetverokuta raznih dimenzija
7        Rectangle colors = new Rectangle(scene.getWidth(),
    scene.getHeight(), new LinearGradient(0f, 1f, 1f, 0f,
    true, CycleMethod.NO_CYCLE, new Stop[]{
8        new Stop(0, Color.web("#f8bd55")),
9        new Stop(0.14, Color.web("#c0fe56")),
10       new Stop(0.28, Color.web("#5dfbc1")),
11       new Stop(0.43, Color.web("#64c2f8")),
12       new Stop(0.57, Color.web("#be4af7")),
13       new Stop(0.71, Color.web("#ed5fc2")),
14       new Stop(0.85, Color.web("#ef504c")),
15       new Stop(1, Color.web("#f2660f")),}));
```

Implementacija animacije u *JavaFX*-u.

```
1       colors.widthProperty().bind(scene.widthProperty());
2       colors.heightProperty().bind(scene.heightProperty());
3       Group blendModeGroup = new Group(new Group(new
    Rectangle(scene.getWidth(), scene.getHeight(), Color.
    BLACK), rectangles), colors); //boja ovisi o poziciji na
     ekranu
4           colors.setBlendMode(BlendMode.OVERLAY);
5           root.getChildren().add(blendModeGroup);
6
7           Button increase = new Button("+");
8           increase.setMinWidth(30);
9           root.getChildren().add(increase);
10          Button decrease = new Button("-");
11          decrease.setMinWidth(30);
12          decrease.setLayoutX(0); decrease.setLayoutY(30);
13          root.getChildren().add(decrease);//dodajemo gumbe za
     povecavanje i smanjivanje brzine animacije
```

Implementacija animacije u *JavaFX*-u.

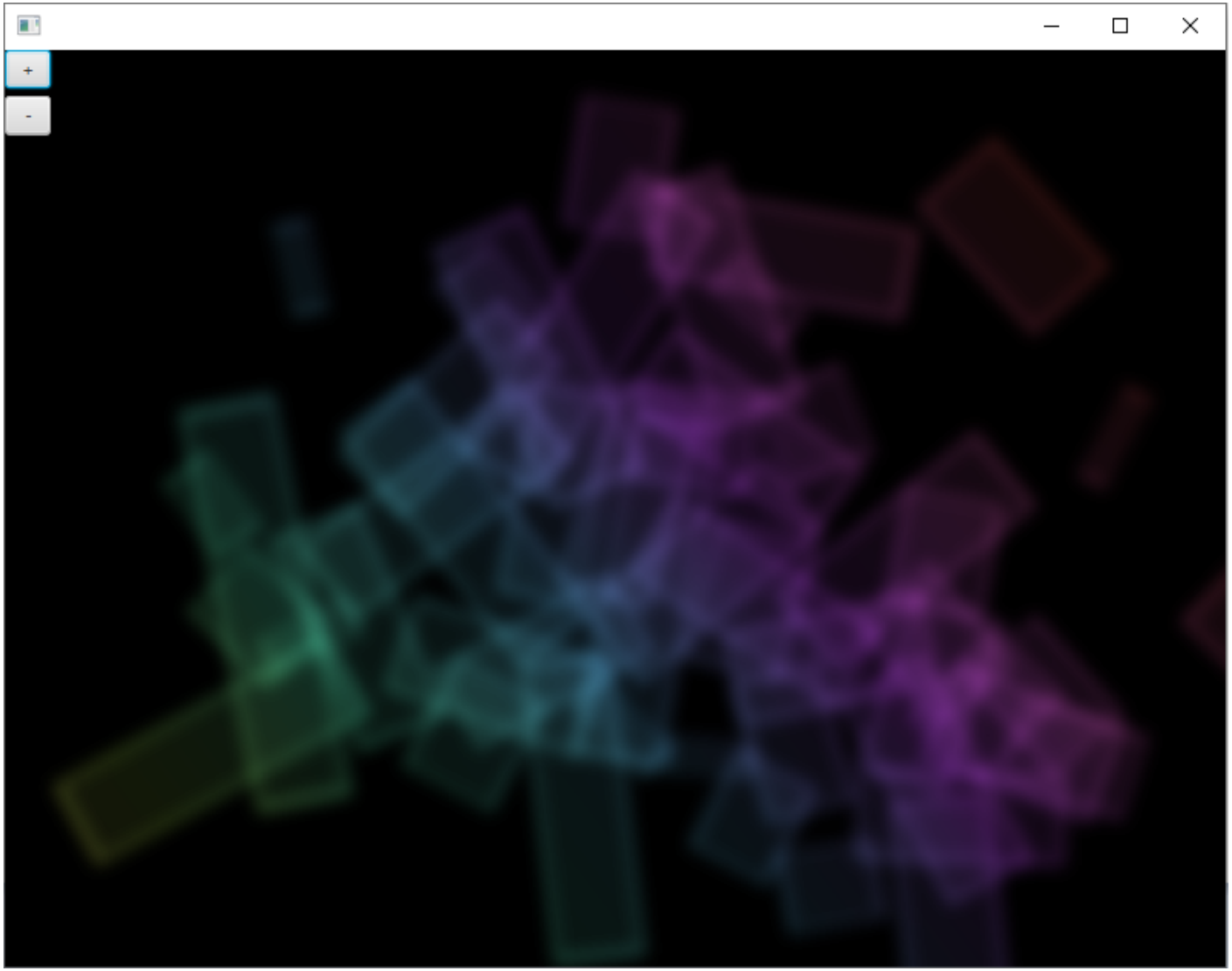
```
1         rectangles.setEffect(new BoxBlur(10, 10, 3));
2         Timeline timeline = new Timeline();
3         timeline.setRate(rateIncrease);//postavlja brzinu
      animacije
4 for (Node rectangle: rectangles.getChildren()) {
5  timeline.getKeyFrames().addAll(
6  new KeyFrame(Duration.ZERO, //pocenta pozicija - 0
7  new KeyValue(rectangle.translateXProperty(),random()*800),
8  new KeyValue(rectangle.translateYProperty(), random()*600),
9  new KeyValue(rectangle.rotateProperty(), random()*360),
10 new KeyValue(rectangle.scaleXProperty(), random()*2)),
11 new KeyFrame(new Duration(100000),//krajnja pozicija - 100s
12 new KeyValue(rectangle.translateXProperty(), random()*800),
13 new KeyValue(rectangle.translateYProperty(), random()*600),
14 new KeyValue(rectangle.rotateProperty(), random()*360),
15 new KeyValue(rectangle.scaleXProperty(), random()*2)) ); }
16        System.out.println(timeline.getCurrentRate());
17 timeline.play(); //pokrecemo 100s animacije
```

Implementacija animacije u *JavaFX*-u.

```
1  increase.setOnAction(actionEvent ->
2          {    rateIncrease+=0.5;
3               timeline.setRate(rateIncrease);  }); //
     definiramo akciju pritiska na gumb povecavanja brzine
     animacije
4
5  decrease.setOnAction(actionEvent ->
6          {    rateIncrease-=0.5;
7               timeline.setRate(rateIncrease); }); //definiramo
      akciju pritiska na gumb manjenja brzine animacije
8               primaryStage.show(); }
9
10 public static void main(String[] args) {
11      launch(args); } } //glavna funkcija koja pokrece
     aplikaciju
```

Implementacija animacije u *JavaFX*-u.

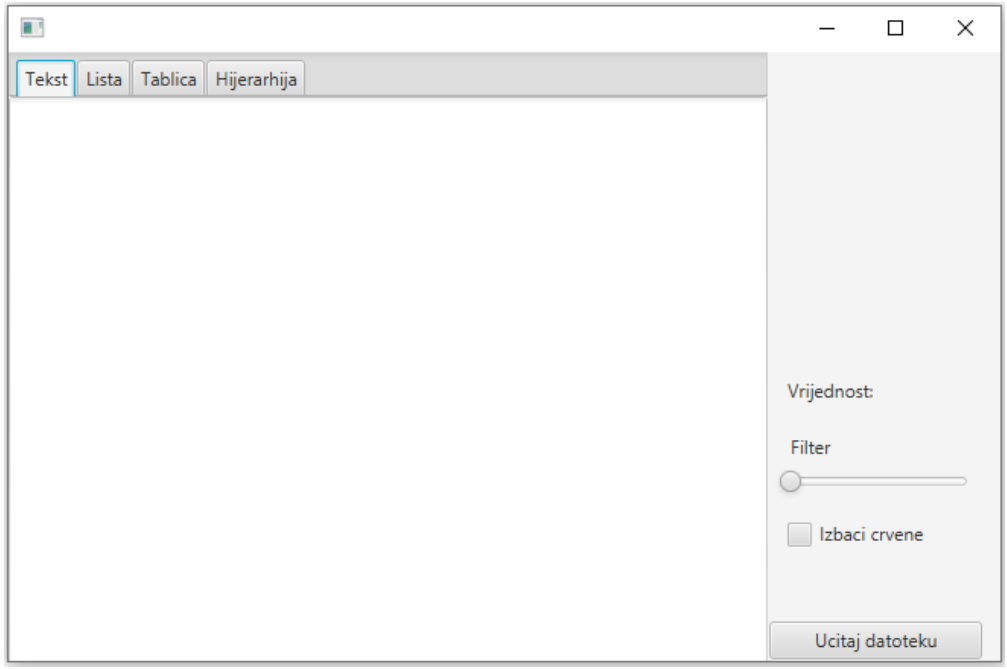
Konačni izgled aplikacije:



Zadatak 6

Riješite zadatak 2 koristeći *JavaFX* uz upotrebu *SceneBuilder*-a.

Prvi korak je napraviti funkcionalno identično sučelje u *JavaFX*-u. Možete koristiti **toolbar** ili **obični spremnik** `Pane` i na njega staviti kliznu, komponentu, komponentu označavanja i labele. Time dobijemo sučelje kao na slici.



Komponentama `textArea` (Tab 1), `ListView` (Tab 2), `TableView` (Tab 3), `TreeView` (Tab 3), kliznoj komponenti i komponenti označavanja, pridjelimo id-eve: `tekst`, `lista`, `tablica`, `hijerarhija`, `filter` i `crveni`. Sada implementiramo funkcionalnost gumba `Ucitaj datoteku`.

```
1  @FXML
2   public void ucitajHandler(ActionEvent event) {
3       File odabrana = odabirDatoteka.showOpenDialog(new
     Stage());
4           try{
5               String cijelaDatoteka = "", linija = "";
6               int brojac = 0, ubaceno = 0;
7               Path p = Paths.get(odabrana.getAbsolutePath());
8               HashSet<String> insertedManufacturers = new
     HashSet<>();
9               double minCijena = Double.MAX_VALUE, maxCijena =
      Double.MIN_VALUE;
10              BufferedReader citac = Files.newBufferedReader(p
     ,StandardCharsets.UTF_8);
```

Implementacija gumba koji učitava datoteku.

```
1   ArrayList<TableColumn> stupciFX = new ArrayList<>();
2   TreeItem<String> rootItem = null;
3
4   while((linija = citac.readLine())!=null){
5        cijelaDatoteka+=linija+"\n";
6        if(brojac == 0){
7            tablica.getColumns().clear();
8            String stupci[] = linija.split(" ");
9            for(int i=0;i<stupci.length;i++){
10             TableColumn stupac = new TableColumn(stupci[i
    ].toUpperCase());
11             if(i<stupci.length-1)
12                  stupac.setCellValueFactory(new
    PropertyValueFactory<>(stupci[i].toLowerCase()));
13             else{
14               stupci[i] = stupci[i].replaceAll("\\(", "");
15               stupci[i] = stupci[i].replaceAll("\\)", "");
16               stupac.setCellValueFactory(new
    PropertyValueFactory<>(stupci[i].toLowerCase())); }
```

Implementacija gumba koji učitava datoteku.

```
1            stupciFX.add(stupac);
2            rootItem = new TreeItem<String> ("Automobili");
   }
3            tablica.getColumns().addAll(stupciFX);
4            brojac  = 1; }
5       else{
6            String tmp[] = linija.split(" ");
7            Auto a = new Auto(tmp[0], tmp[1],tmp[2], Double.
   parseDouble(tmp[3]));
8            double anum =Double.parseDouble(tmp[3]);
9            if(minCijena>anum)
10                minCijena = anum;
11           if(maxCijena<anum)
12                maxCijena = anum;
13           tablica.getItems().add(a);
14           lista.getItems().add(a.getProizvodac()+" "+a.
   getTip());
15           TreeItem<String> proizvodac = new TreeItem<
   String> (tmp[0]);
```

Implementacija gumba koji učitava datoteku.

```
1            if(insertedManufacturers.contains(tmp[0])){
2                ObservableList<TreeItem<String>> djeca =
   rootItem.getChildren();
3                TreeItem<String> proizvodacStablo = null;
4                for(int i=0;i<djeca.size();i++){
5                 if(djeca.get(i).getValue().equals(tmp[0])){
6                     proizvodacStablo = djeca.get(i);
7                     break; }  }
8                     proizvodacStablo.getChildren().add(new
    TreeItem<String>(tmp[1]));  }
9                else{
10                   insertedManufacturers.add(tmp[0]);
11                   rootItem.getChildren().add(proizvodac);
12                   proizvodac.getChildren().add(new TreeItem
   <String>(tmp[1])); }  } }
13                   tekst.setText(cijelaDatoteka);
14                   hijerarhija.setRoot(rootItem);
15                   filter.setMax(maxCijena);
16                   filter.setMin(minCijena);
```

Implementacija gumba koji učitava datoteku.

```
1 //tablicaCopy.getColumns().addAll(tablica.getColumns());//
    NIKAKO se nesmiju kopirati stupci jer se prebaci
    poveznica komponente GUI-a na novu tablicu!
2                 for(int i=0;i<tablica.getItems().size();i++)
3                     tablicaCopy.getItems().add(tablica.
    getItems().get(i));
4                  }
5                  catch(Exception e){
6                      e.printStackTrace(); } }
```

Implementacija gumba koji učitava datoteku.

Navedena implementacija pripada klasi kontroler aplikacije. U toj aplikaciji imamo definirane sljedeće elemente: `@FXML FileChooser odabirDatoteka; @FXML TextArea tekst; @FXML ListView lista; @FXML TableView tablica; @FXML TableView tablicaCopy; @FXML TreeView hijerarhija; @FXML Slider filter; @FXML Label vrijednostF; boolean izbaciCrvene; @FXML CheckBox crveni;`

JavaFX tablici dodajemo objekte (instance) posebno definiranih klasa koje reprezentiraju objekte koji su prezentirani u tablici. Klase za svaki elemet (stupac tablice) moraju imati odgovarajuću get metodu oblika `getImeelementa()`.

```
1  public class Auto {
2      String proizvodac, tip, boja;
3      double cijenausd;
4
5      Auto(String p, String t, String b, double c){
6          proizvodac = p;  tip = t;
7          boja = b; cijenausd = c;
8      }
9
10     public String getProizvodac(){ return proizvodac; }
11     public String getTip(){ return tip; }
12     public String getBoja(){ return boja; }
13     public double getCijenausd(){ return cijenausd; }
14 }
```

Implementacija klase Auto.

Unutar inicijalizacijske funkcije kontrolera definiramo kod za osluškivanje promjena klizne komponente

```
1  @Override
2      public void initialize(URL url, ResourceBundle rb) {
3          tablicaCopy = new TableView();
4          izbaciCrvene = false;
5          odabirDatoteka = new FileChooser();
6          odabirDatoteka.setTitle("Otvori .txt datoteku");
7          odabirDatoteka.getExtensionFilters().addAll(new
       ExtensionFilter("SVE TEKSTUALNE", "*.txt"));
8          filter.valueProperty().addListener(new ChangeListener
       <Number>() {
9
10         public void changed(ObservableValue<? extends Number>
        vv, Number stara_vr, Number nova_vr) {
11         vrijednostF.setText(String.format("%.2f", nova_vr));
12
```

Implementacija inicijalizacijske metode kontrolera.

```
1     if(tablicaCopy.getItems().size()!=0){
2        for(int i=tablica.getItems().size()-1;i>=0;i--)
3            tablica.getItems().remove(i);
4
5     ObservableList<Auto> tmp = javafx.collections.
    FXCollections.observableArrayList();
6
7      for(int i=0;i<tablicaCopy.getItems().size();i++){
8           if(Double.parseDouble(((Auto)tablicaCopy.getItems
    ().get(i)).cijenausd+"")>=(double)new_val){
9
10          if(izbaciCrvene == false || !((Auto)tablicaCopy.
    getItems().get(i)).boja.equals("crveni")){
11            tmp.add((Auto)tablicaCopy.getItems().get(i));
12                } } }
13            tablica.setItems(tmp);  }   }  }); }
```

Implementacija inicijalizacijske metode kontrolera.

```
1  @FXML public void crveniHandler (ActionEvent event) {
2                  if(crveni.isSelected()){  izbaciCrvene =
     true; }
3                  else{  izbaciCrvene = false; }
4
5                  if(tablicaCopy.getItems().size()!=0){
6          for(int i=tablica.getItems().size()-1;i>=0;i--)
7              tablica.getItems().remove(i);
8
9        ObservableList<Auto> tmp = javafx.collections.
     FXCollections.observableArrayList();
10             double vrijednost = filter.getValue();
11
12         for(int i=0;i<tablicaCopy.getItems().size();i++){
13                 if(Double.parseDouble(((Auto)tablicaCopy.
     getItems().get(i)).cijenausd+"")>=vrijednost){
14
```

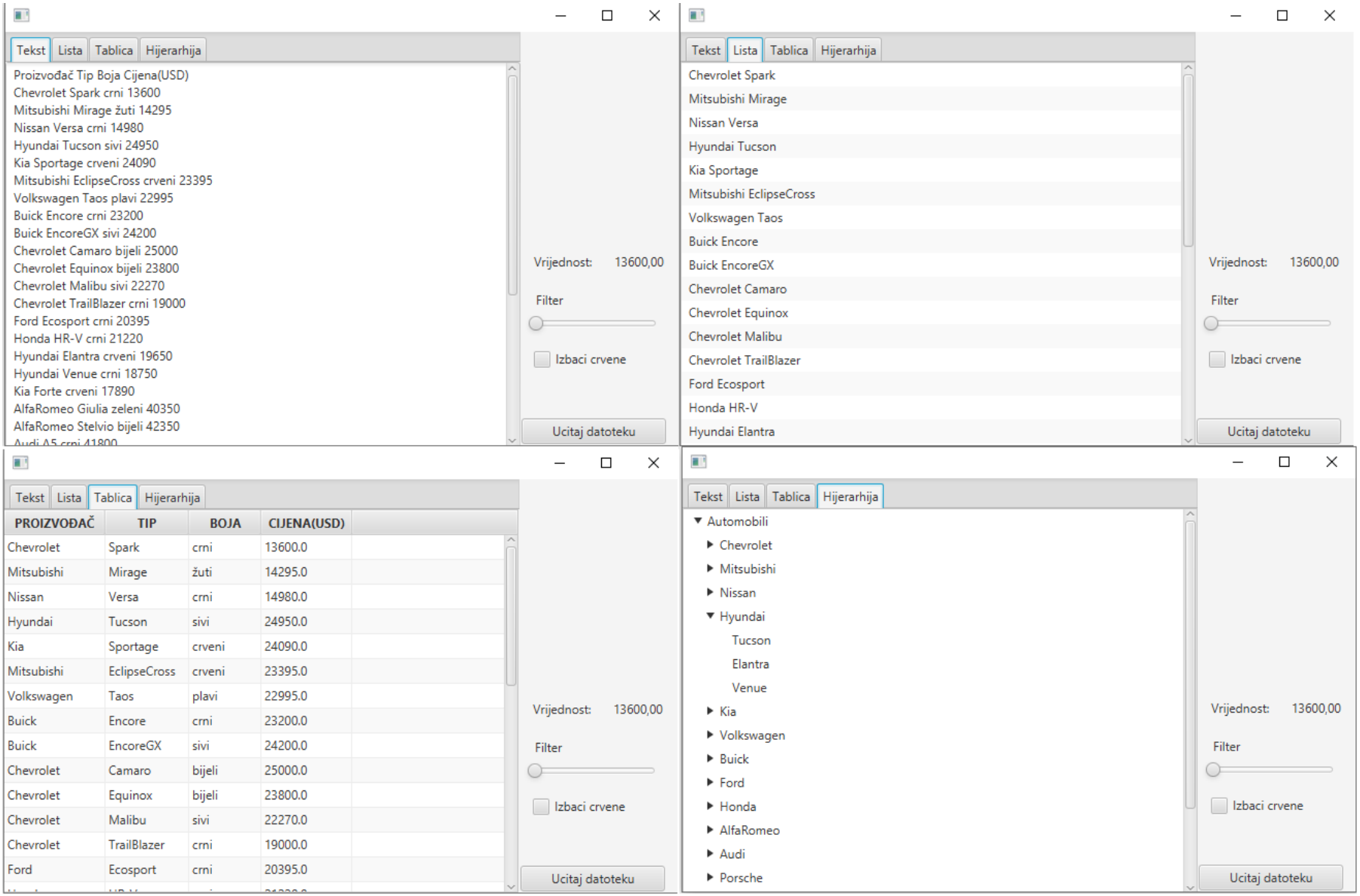
Implementacija koda za obradu promjene stanja komponente označavanja.

```
1                   if(izbaciCrvene == false || !((Auto)
    tablicaCopy.getItems().get(i)).boja.equals("crveni")){
2                   tmp.add((Auto)tablicaCopy.getItems().get(i));
      } } }
3                 tablica.setItems(tmp);  }   }
```

Implementacija koda za obradu promjene stanja komponente označavanja.

```
1 public class Zad6 extends Application{
2     @Override
3   public void start(Stage primaryStage) throws IOException{
4         Parent root = FXMLLoader.load(getClass().getResource
    ("/javafx/Zad6FXML.fxml"));
5         Scene scene = new Scene(root);
6         primaryStage.setScene(scene);
7         primaryStage.initStyle(StageStyle.DECORATED);
8         primaryStage.show(); }
9
10    public static void main(String[] args) {
11        launch(args); } }
```

Glavna klasa aplikacije.


Tekst
Lista
Tablica
Hijerarhija
Proizvođač Tip Boja Cijena(USD)
Chevrolet Spark crni 13600
Mitsubishi Mirage žuti 14295
Nissan Versa crni 14980
Hyundai Tucson sivi 24950
Kia Sportage crveni 24090
Mitsubishi EclipseCross crveni 23395
Volkswagen Taos plavi 22995
Buick Encore crni 23200
Buick EncoreGX sivi 24200
Chevrolet Camaro bijeli 25000
Chevrolet Equinox bijeli 23800
Chevrolet Malibu sivi 22270
Chevrolet TrailBlazer crni 19000
Ford Ecosport crni 20395
Honda HR-V crni 21220
Hyundai Elantra crveni 19650
Hyundai Venue crni 18750
Kia Forte crveni 17890
AlfaRomeo Giulia zeleni 40350
AlfaRomeo Stelvio bijeli 42350
Vrijednost: 13600,00
Filter
Izbaci crvene
Ucitaj datoteku
Chevrolet Spark
Mitsubishi Mirage
Nissan Versa
Hyundai Tucson
Kia Sportage
Mitsubishi EclipseCross
Volkswagen Taos
Buick Encore
Buick EncoreGX
Chevrolet Camaro
Chevrolet Equinox
Chevrolet Malibu
Chevrolet TrailBlazer
Ford Ecosport
Honda HR-V
Hyundai Elantra
PROIZVOĐAČ
TIP
BOJA
CIJENA(USD)
Chevrolet Spark crni 13600.0
Mitsubishi Mirage žuti 14295.0
Nissan Versa crni 14980.0
Hyundai Tucson sivi 24950.0
Kia Sportage crveni 24090.0
Mitsubishi EclipseCross crveni 23395.0
Volkswagen Taos plavi 22995.0
Buick Encore crni 23200.0
Buick EncoreGX sivi 24200.0
Chevrolet Camaro bijeli 25000.0
Chevrolet Equinox bijeli 23800.0
Chevrolet Malibu sivi 22270.0
Chevrolet TrailBlazer crni 19000.0
Ford Ecosport crni 20395.0
Automobili
Chevrolet
Mitsubishi
Nissan
Hyundai
Tucson
Elantra
Venue
Kia
Volkswagen
Buick
Ford
Honda
AlfaRomeo
Audi
Porsche

Zadatak - DZ

Modificirajte aplikacije iz Zadatka 2 i Zadatka 6 tako da se pri promjeni vrijednosti klizne komponente ili stanja komponente označavanja mijenjaju komponente svih tabova grafičkog sučelja.