



Java FX - Pregled važnijih vizuelnih komponenti

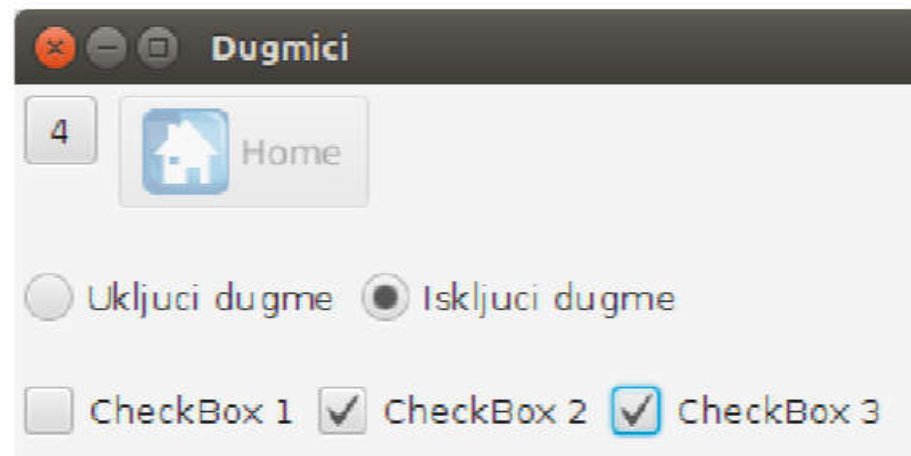


Dugmići

- U JavaFX biblioteci postoji nekoliko tipova dugmića:
 - Standardno dugme, predstavljeno klasom **Button**;
 - **RadioButton**: obično se koristi za skup opcija, pri čemu samo jedna opcija može biti selektovana;
 - **CheckBox**: slično kao RadioButton, ali više opcija može biti selektovano istovremeno;
 - **ToggleButton**: kombinacija standardnog dugmeta i RadioButton-a. Izgleda kao standardno dugme, ali može ostati pritisnuto i time označavati selekciju poput RadioButton-a.
 - Button i ToggleButton mogu prikazivati tekst, ikonicu, ili oba, dok preostala dva tipa prikazuju samo tekst.

Primer

- Dugmići su raspoređeni u tri reda, po jedan tip u zasebnom redu.
- U prvom redu se nalaze dva standardna dugmeta, u drugom dva RadioButton-a, a u trećem tri CheckBox-a.
- Upotreba ToggleButton-a nije prikazana, jer je veoma slična upotrebi RadioButton-a.



Primer

```
public class Dugmici extends Application {
    private int broj;

    @Override
    public void start(Stage stage) throws Exception {
        // inicijalizacija redova
        HBox[] redovi = new HBox[3];
        for (int i = 0; i < redovi.length; i++)
            redovi[i] = new HBox(8); // spacing = 8px

        // prvi red - dva standardna dugmeta
        // prvo povecava i prikazuje vrednost polja 'broj'
        Button brojac = new Button("0");
        brojac.setOnAction(e -> {
            ++broj;
            // postavljamo novi tekst dugmeta
            brojac.setText(broj + "");
        });
        // dume sa tekstom i slikom
        Button home = new Button("Home");
        // ucitavanje i postavljanje slike
        Image slika = new Image(getClass().getResourceAsStream("home.png"));
        // moze i direktno u konstruktoru, kao drugi parametar
        home.setGraphic(new ImageView(slika));
        // dodajemo ih u red
        redovi[0].getChildren().add(brojac);
        redovi[0].getChildren().add(home);
    }
}
```

Primer

```
// drugi red - RadioButton
// kada korisnik selektuje 'ukljuci', omoguci cemo dugme 'home'
RadioButton ukljuci = new RadioButton("Ukljuci dugme");
ukljuci.setOnAction(e -> home.setDisable(false));
// kada korisnik selektuje 'iskljuci', onemoguci cemo dugme 'home'
RadioButton iskljuci = new RadioButton("Iskljuci dugme");
iskljuci.setOnAction(e -> home.setDisable(true));
// ako vise RadioButton-a pripada istoj grupi, samo jedno
// moze biti selektovano
ToggleGroup rbgroup = new ToggleGroup();
ukljuci.setToggleGroup(rbgroup);
iskljuci.setToggleGroup(rbgroup);
ukljuci.setSelected(true);
// dodajemo ih u red
redovi[1].getChildren().add(ukljuci);
redovi[1].getChildren().add(iskljuci);

// svi CheckBox-ovi u trecem redu ce imati isti metod za obradu
// dogadjaja. u ovom slucaju, metod implementiramo u okviru
// anonimne unutrasnje klase
EventHandler<ActionEvent> obr = new EventHandler<ActionEvent>() {
```

Primer

```
@Override public void handle(ActionEvent e) {
    // koje dugme je kliknuto?
    CheckBox c = (CheckBox) e.getSource();
    if (c.isSelected())
        System.out.println(c.getText() + " je selektovan");
    else
        System.out.println(c.getText() + " nije selektovan");
}

// treci red - CheckBox
CheckBox[] check = new CheckBox[3];
for (int i = 0; i < check.length; i++) {
    check[i] = new CheckBox("CheckBox " + (i + 1));
    check[i].setOnAction(obr);
    redovi[2].getChildren().add(check[i]);
}

// dodajemo redove u VBox
VBox root = new VBox();
root.setSpacing(16);
for (HBox r : redovi)
    root.getChildren().add(r);

Scene scene = new Scene(root, 320, 120);
stage.setScene(scene);
stage.setTitle("Dugmici");
stage.show();
}

public static void main(String[] args) {
    launch(args);
}
}
```



Tekstualne komponente

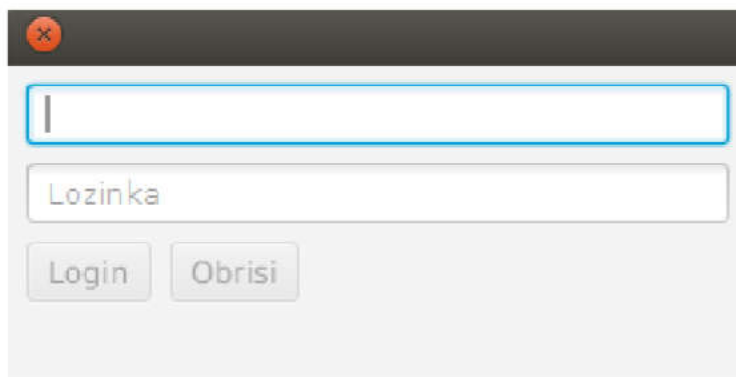
- Najčešće korišćena komponenta za prikaz statičkog teksta je Label.
- Pored toga, JavaFX nudi i nekoliko komponenti za unos teksta, od kojih su najbitnije sledeće:
 - **TextField**: može da sadrži samo jednu liniju teksta;
 - **PasswordField**: koristi se za skriven unos, na primer, kod unosa lozinki;
 - **TextArea**: može da sadrži više linija teksta;
 - **HTMLEditor**: napredni editor formatiranog teksta, koji podržava standard HTML5.



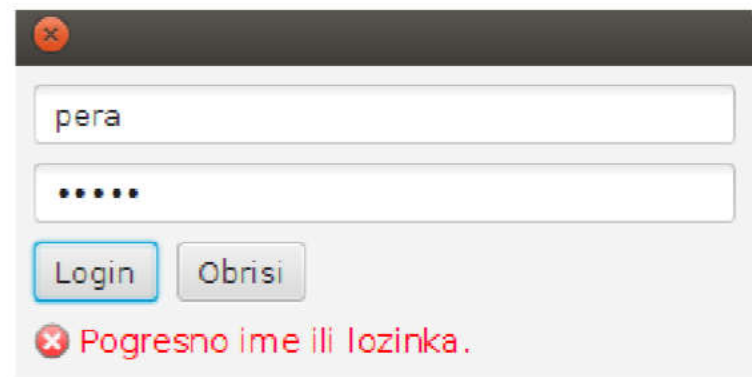
Primer

- U nastavku je dat primer aplikacije koja korisniku omogućuje da radi sa komponentom HTMLEditor.
- Međutim, korisnik se prethodno mora ulogovati, koristeći korektno ime i lozinku.
- Aplikacija sadrži dva prozora: jedan pomoćni za prijavljivanje (login) korisnika, i glavni, za unos teksta.
- Izvorni kod prozora za login je dat u nastavku. Ovaj prozor sadrži polje ime tipa TextField za unos imena, polje loz tipa PasswordField za unos lozinke, labelu za prikaz informacija o grešci, kao i dva dugmeta: login koji prijavljuje korisnika ako su podaci ispravni, i obrisi koji briše vrednosti polja ime i loz.

Primer



A login window with a dark gray title bar containing a red close button. It features two input fields: the first is empty, and the second is labeled "Lozinka". Below the fields are two buttons: "Login" and "Obrisi".



The same login window as on the left, but with the first input field containing the text "pera" and the second field containing six dots. The "Login" button is highlighted with a blue border. Below the buttons, a red error message is displayed: "Pogresno ime ili lozinka."

Login prozor nakon startovanja aplikacije (levo) i nakon pogrešnog unosa (desno).

Primer

Pomoćni prozor za login korisnika.

```
public class LoginScreen {
    private Stage stage;
    private TextField ime;
    private PasswordField loz;
    private Label greska;
    private Button login;
    private Button obrisi;
    private boolean ok;

    public LoginScreen() {
        // polje za unos korisnickog imena
        ime = new TextField();
        ime.setPromptText("Korisnicko ime"); // hint
        // kada se tekst promeni, pozovi metod 'updateControls'
        ime.textProperty().addListener(e -> updateControls());

        // polje za unos lozinke
        loz = new PasswordField();
        loz.setPromptText("Lozinka");
        loz.textProperty().addListener(e -> updateControls());

        // polje za login: proverava da li su uneti podaci ispravni
        login = new Button("Login");
        login.setOnAction(e -> {
            if (ime.getText().equals("petar") &&
                loz.getText().equals("perazdera")) { // ok, zatvori prozor
                ok = true;
                stage.hide();
            }
            else // prikazi gresku
                greska.setVisible(true);
        });
    }
}
```

Primer

Pomoćni prozor za login korisnika.

```
// dugme koje resetuje 'ime' i 'loz'
obrisi = new Button("Obrisi");
obrisi.setOnAction(e -> {
    ime.setText("");
    loz.setText("");
});

// staticki tekst koji prikazuje gresku
// polje ima sliku i crveni tekst, i koristi
// "Tahoma" font velicine 14 tacaka
greska = new Label("Pogresno ime ili lozinka.");
greska.setFont(new Font("Tahoma", 14));
greska.setTextFill(Color.RED);
Image slk = new Image(getClass().getResourceAsStream("error.png"));
greska.setGraphic(new ImageView(slk));

// dugmici ce biti u jednoj liniji
HBox dugmici = new HBox(8);
```

1

Primer

```
dugmici.getChildren().addAll(login, obrisi);

// poredjaj komponente vertikalno
VBox box = new VBox(8);
box.setPadding(new Insets(8));
box.getChildren().addAll(ime, loz, dugmici, greska);

stage = new Stage();
// ukloni dekoracije pozornice (npr. minimize i maximize dugmice)
stage.initStyle(StageStyle.UTILITY);
stage.setScene(new Scene(box, 320, 140));
// azuriraj stanja
updateControls();
// prikazuje pozornicu i ceka dok ne bude zatvorena
stage.showAndWait();
}

// pomocni metod koji azurira prikaz pojedinih komponenti
// na osnovu stanja aplikacije
private void updateControls() {
    boolean nemaIme = ime.getText().length() == 0;
    boolean nemaLoz = loz.getText().length() == 0;
    login.setDisable(nemaIme || nemaLoz);
    obrisi.setDisable(nemaIme && nemaLoz);
    // kada korisnik izmeni ime i lozinku, sakricemo gresku
    greska.setVisible(false);
}

public boolean isOk() { return ok; }
}
```



Primer

- Prilikom dizajniranja vizuelnih aplikacija, veoma je bitno da korisnik jasno zna koje operacije su mu trenutno dostupne, a koje ne.
- Konkretno, korisnik ne bi trebalo da može da klikne na dugme login ako nije uneo ime i/ili lozinku; nema smisla da klikće na dugme obriši ako su polja prazna.
- Da bismo ovo postigli, uvodimo pomoćni metod ***updateControls*** koji će ažurirati stanja vizuelnih komponenti na osnovu trenutnog stanja aplikacije.
- Ovaj pomoćni metod pozivamo kad god se promeni stanje aplikacije, konkretno, kada korisnik promeni sadržaj tekstualnih polja.
- Metod za obradu događaja promene teksta postavljamo pozivom ***textProperty().addListener(e -> ...)***.



Primer

- Klik na dugme login jednostavno proverava da li su uneto ime i prezime neke predefnisane vrednosti.
- Ukoliko jesu, login prozor se zatvara; u suprotnom, korisniku prikazujemo grešku, odnosno odgovarajuću labelu.
- Login prozor je prikazan na ekranu pozivom metoda **showAndWait**.
- Za razliku od metoda **show**, **showAndWait** će prikazati prozor/pozornicu i blokirće dalje izvršavanje aplikacije sve dok prozor ne bude zatvoren.
- Poziv **stage.initStyle(StageStyle.UTILITY)** uklanja pojedine dekoracije prozora, kao što su dugmići za minimiziranje i maksimiziranje.



Primer

- Izvorni kod glavne aplikacije je dat u nastavku.
- Najpre kreiramo (i prikazujemo) login prozor.
- Ako prijavljivanje korisnika nije bilo uspešno, izvršavanje aplikacije će biti prekinuto.
- Korektan način za prekidanje JavaFX aplikacije je poziv **Platform.exit()**, koji se može obaviti bilo gde.
- Ako je prijavljivanje uspešno, biće prikazan prozor sa komponentnom HTMLEditor i nekim početnim tekstom.

Primer

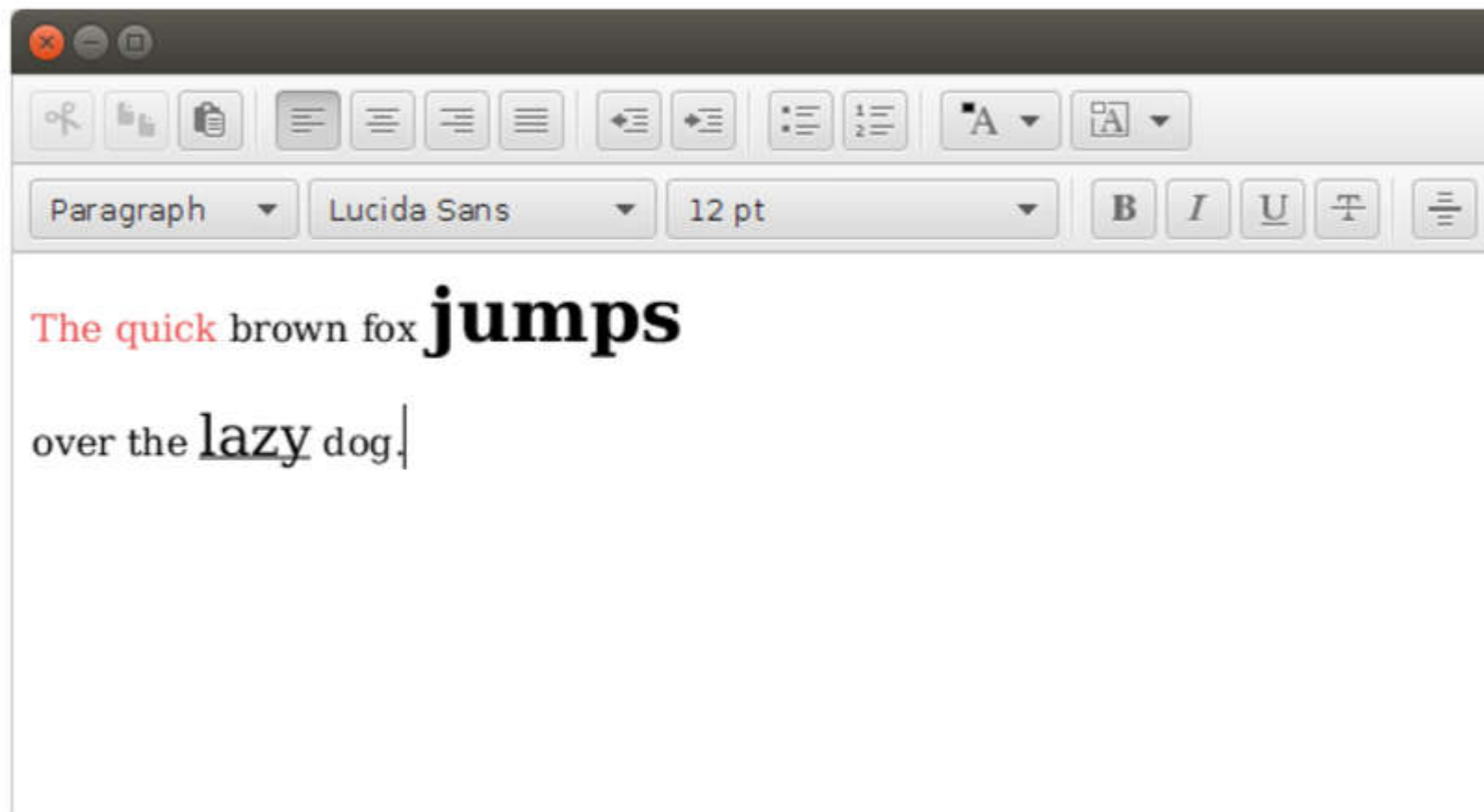
Izvorni kod glavne aplikacije.

```
public class TextDemo extends Application {
    @Override
    public void start(Stage stage) throws Exception {
        LoginScreen login = new LoginScreen();
        if (!login.isOk()) {
            Platform.exit();
            return;
        }
        HTML editor = new HTML();
        editor.setHtmlText("<html><body>The quick brown fox "
            + "jumps over the lazy dog.</body></html>");
        BorderPane root = new BorderPane(editor);
        Scene scene = new Scene(root, 640, 320);
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```


Primer

Izgled glavne aplikacije i komponente HTMLEditor.





Liste

- Postoje dva osnovna tipa listi:
 - padajuća lista **ComboBox** i
 - standardna lista **ListView**.
- Liste mogu sadržati objekte bilo kog tipa, a prikazaće njihove tekstualne reprezentacije (tj. pozivaće metod `toString` svakog objekta).
- Interno, liste čuvaju elemente u kolekciji tipa **ObservableList** iz paketa **javafx.collections**.
- Ovaj interfejs nasleđuje **java.util.List** i generiše događaje koji signaliziraju promenu, odnosno dodavanje ili uklanjanje elemenata.
- Objekte ove kolekcije je najlakše kreirati pozivom jednog od statičkih metoda klase `FXCollections`. Referenca na internu kolekciju liste se dobija pozivom metoda **getItems**.



Liste

- **ListView** omogućuje korisniku da selektuje jedan ili više elemenata istovremeno,
- dok u slučaju **ComboBox-a** korisnik može selektovati najviše jedan element.
- Selekcijom ne upravlja sama lista, već njen unutrašnji objekat, tzv. model selekcije (eng. selection model).
- Model je iz liste moguće dobiti pozivom metoda **getSelectionModel**, a funkcionalnosti modela iz ListView-a i ComboBox-a se razlikuju u tome što prvi omogućuje rad sa više elemenata istovremeno (na primer, moguće je selektovati niz elemenata odjednom).



Liste

- Neki od najbitnijih metoda modela selekcije su:
 - **getSelectedItem**: vraća selektovani element, odnosno null ukoliko lista nema selekciju. Ukoliko lista ima više selektovanih elemenata, biće vraćen onaj koji je selektovan poslednji.
 - **getSelectedIndex**: vraća indeks selektovanog elementa, odnosno -1 ukoliko lista nema selekciju. Slično, u slučaju višestruke selekcije, biće vraćen indeks poslednje selektovanog elementa.
 - **getSelectedItems/getSelectedIndices**: slično kao prethodna dva, ali postoje samo u slučaju ListView-a. Povratna vrednost je kolekcija tipa ObservableList.



Liste

- Neki od najbitnijih metoda modela selekcije su:
 - **selectFirst/Last/Next/Previous/All/Range**: selektuje, redom, prvi, poslednji, sledeći, prethodni element, sve elemente, ili zadati niz elemenata. Pri tome, poslednja dva metoda postoje samo u slučaju ListView-a.
 - **select**: selektuje prosleđeni objekat. Postoje dve varijante metoda: jedan koji prihvata sam objekat, i jedan koji prihvata indeks objekta.
 - **isEmpty**: vraća da li je selekcija prazna (tj. ništa nije selektovano).
 - **clearSelection**: de-selektuje elemente. Postoji i varijanta metoda koja prihvata jedan parametar indeks elementa koji treba de-selektovati.
 - **isSelected**: vraća da li je element sa prosleđenim indeksom selektovan.

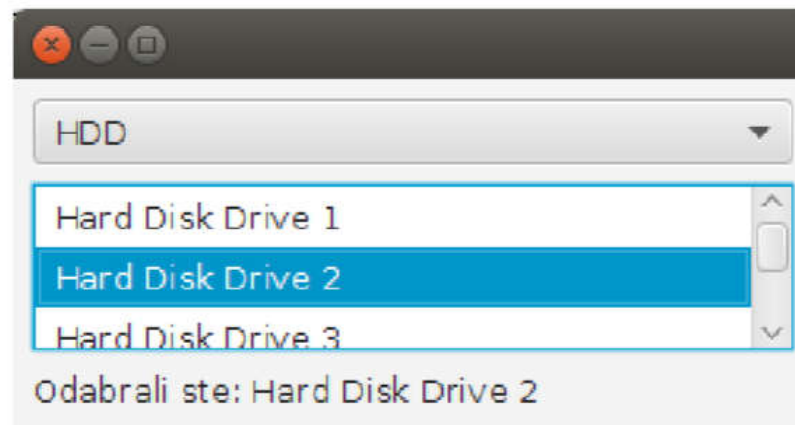


Liste

- Korisnik može selektovati više elemenata unutar ListView komponente, ali je po osnovnim podešavanjima moguće selektovati samo jedan element.
- Ovo ponašanje se menja pozivom metoda **setSelectionMode** nad modelom selekcije.
- Parametar metoda je nabrojivi tip SelectionMode sa elementima **SINGLE** i **MULTIPLE** za, redom, jednostruku, odnosno višestruku selekciju.

Primer

- Aplikacija sadrži kategorije računarskih komponenti (CPU, RAM i HDD) unutar padajuće liste.
- Kada korisnik selektuje neku kategoriju, u standardnoj listi se prikazuju sve komponente te kategorije.
- Konačno, kada korisnik selektuje neku komponentu, aplikacija prikazuje njen opis u donjoj labeli.
- ukoliko lista nije dovoljno velika da prikaže sve elemente, automatski će biti aktivirano skrolovanje.



Primer

Osnovni rad sa listama.

```
public class Liste extends Application {  
    // kljuc: kategorija , vrednost: komponente date kategorije  
    private Map<String , List<String>> komponente;  
  
    @Override  
    public void start(Stage stage) throws Exception {  
  
        ucitajPodatke();  
        // labela sa opisom selektovane komponente  
        Label odabir = new Label();  
  
        // lista komponenti  
        ListView<String> kompLista = new ListView<>();  
        // model selekcije za ListView  
        MultipleSelectionModel<String> modelListe =  
            kompLista.getSelectionModel();  
        // opis selektovanog elementa prikazujemo u labeli  
        modelListe.selectedItemProperty().addListener(e ->  
            odabir.setText("Odabrali ste: " + modelListe.getSelectedItem())  
        );  
    }  
}
```


Primer

```
// padajuca lista kategorija
ComboBox<String> kategorije = new ComboBox<>();
kategorije.getItems().addAll("CPU", "RAM", "HDD");
kategorije.setPromptText("Odaberite kategoriju");
kategorije.setMaxWidth(Double.MAX_VALUE);
// kada korisnik odabere kategoriju, prikazacemo sve njene
// komponente. u slucaju padajuce liste, o promeni selekcije
// mozemo biti obavesteni i pomocu akcija
kategorije.setOnAction(e -> {
    // model selekcije za ComboBox
    SingleSelectionModel<String> modelCB =
        kategorije.getSelectionModel();
    List<String> lista = komponente.get(modelCB.getSelectedItem());
    // prikazi novi skup komponenti
    kompLista.getItems().clear();
    kompLista.getItems().addAll(lista);
});

BorderPane root = new BorderPane();
root.setPadding(new Insets(8));
root.setCenter(kompLista);
root.setTop(kategorije);
root.setBottom(odabir);
// po 8 piksela iznad i ispod standardne liste
BorderPane.setMargin(kompLista, new Insets(8, 0, 8, 0));

stage.setScene(new Scene(root, 320, 140));
stage.show();
}
...
}
```



Tabele

- Tabela proširuje funkcionalnosti liste uvođenjem kolona.
- Svaka kolona ima svoje zaglavlje i širinu, i može prikazivati različite tipove podataka.
- U preseku reda i kolone se nalazi ćelija tabele (eng. cell).
- Da bi se lako i efikasno radilo sa tabelama u JavaFX-u, neophodno je prvo upoznati se sa konceptom JavaFX binova (eng. bean).
- JavaFX binovi su softverske (ne obavezno vizuelne) komponente.
- Razlikuju se od običnih klasa i objekata, između ostalog i po tome što drugi objekti mogu „posmatrati“ polja binova i „vezati se“ za njih, što znači da će biti automatski obavješteni kada neko polje promeni vrednost.
- Praktično sve vizuelne JavaFX komponente su definisane kao binovi.



Tabele

- Prilikom rada sa binovima se koristi nešto drugačija terminologija.
- Binovi nemaju polja kao obične klase i objekti, već tzv. osobine (eng. properties).
- U praksi to znači da koristimo drugačije tipove podataka.
- Konkretno, umesto tipova `int` i `Integer` koristimo `IntegerProperty`, umesto `String` koristimo `StringProperty`, itd.
- Za referencijalne tipove navodimo `ObjectProperty<T>`, gde `T` predstavlja konkretan tip.
- Pored toga, moramo navesti metod za pristup osobini, koji mora imati isti naziv kao i sama osobina.
- U radu, da bismo uzeli konkretnu vrednost osobine, pozivamo njen metod `get`, a za postavljanje vrednosti osobine metod `set`.




Primer

- Potrebno je napisati aplikaciju koja u tabeli prikazuje osnovne informacije o planetama Sunčevog sistema:
 - naziv,
 - rednibroj,
 - broj satelita, kao i
 - da li je gasovita ili ne.
- Pored toga, ukoliko korisnik selektuje neku planetu, aplikacija ispod tabele prikazuje njenu sliku.
- Planetu ćemo opisati binom čiji je izvorni kod dat u nastavku:

Primer

Planete Suncevog sistema				
Naziv	Redni broj	Broj satelita	Gasovita?	
Merkur	1	0	false	
Venera	2	0	false	
Zemlja	3	1	false	
Mars	4	2	false	
Jupiter	5	66	true	
Saturn	6	62	true	
Uran	7	27	true	
Neptun	8	13	true	



Primer

*Planeta Sunčevog sistema,
implementirana u formi bina.*

```
public class Planeta {
    private StringProperty naziv;
    private IntegerProperty redniBroj;
    private IntegerProperty brojSatelita;
    private BooleanProperty gasovita;
    private ObjectProperty<ImageView> slika;

    public Planeta(String naziv, int redniBroj, int brojSatelita,
        boolean gasovita, String nazivSlike) {
        this.naziv = new SimpleStringProperty(naziv);

        this.redniBroj = new SimpleIntegerProperty(redniBroj);
        this.brojSatelita = new SimpleIntegerProperty(brojSatelita);
        this.gasovita = new SimpleBooleanProperty(gasovita);
        // sliku učitavamo iz pod-paketa 'res'
        Image slika = new Image(getClass().
            getResourceAsStream("res/" + nazivSlike));
        this.slika = new SimpleObjectProperty<>(new ImageView(slika));
    }

    /** metodi za vraćanje osobina bina */

    public StringProperty naziv() { return naziv; }

    public IntegerProperty redniBroj() { return redniBroj; }

    public IntegerProperty brojSatelita() { return brojSatelita; }

    public BooleanProperty gasovita() { return gasovita; }

    public ObjectProperty<ImageView> slika() { return slika; }
}
```



Primer

- Pre kreiranja same tabele je potrebno opisati kolone.
- Kolona je predstavljena klasom TableColumn koja se zadaje sa dva tipa: tipom elemenata tabele i tipom elemenata ćelija posmatrane kolone.
- Konkretno, kolona naziv je definisana kao TableColumn<Planeta, String>, što znači da su elementi tabele planete, a elementi kolone stringovi.
- Nakon toga, koloni zadajemo tzv. fabriku ćelija (eng. cell factory).
- U našem slučaju, fabrika je lambda izraz koji govori koju osobinu bina treba koristiti za prikaz vrednosti ćelija kolone (e.getValue() vraća referencu na trenutnu planetu).
- Ukoliko korisnik može da menja vrednosti ćelija, tada će nove vrednosti automatski biti upisane u odgovarajuću osobinu bina. Dakle, ćeliju smo uz minimalni napor „vezali“ za osobinu bina. Pojedinim kolonama smo zadali i minimalnu širinu, pozivom setMinWidth.

Primer

Glavna klasa.

```
public class Planete extends Application {
    private ObservableList<Planeta> planete;

    @SuppressWarnings("unchecked")
    @Override
    public void start(Stage stage) throws Exception {
        BorderPane pane = new BorderPane();

        planete = FXCollections.observableArrayList();
        planete.add(new Planeta("Merkur", 1, 0, false, "mercury.png"));
        planete.add(new Planeta("Venera", 2, 0, false, "venus.png"));
        planete.add(new Planeta("Zemlja", 3, 1, false, "earth.png"));
        planete.add(new Planeta("Mars", 4, 2, false, "mars.png"));
        planete.add(new Planeta("Jupiter", 5, 66, true, "jupiter.png"));
        planete.add(new Planeta("Saturn", 6, 62, true, "saturn.png"));
        planete.add(new Planeta("Uran", 7, 27, true, "uranus.png"));
        planete.add(new Planeta("Neptun", 8, 13, true, "neptune.png"));

        // kolone tabele - naziv
        TableColumn<Planeta, String> naziv = new TableColumn<>("Naziv");
        naziv.setCellValueFactory(e -> e.getValue().naziv());
        // redni broj
        TableColumn<Planeta, Number> rbr = new TableColumn<>("Redni broj");
        rbr.setCellValueFactory(e -> e.getValue().redniBroj());
        rbr.setMinWidth(100.0);
        // broj satelita
        TableColumn<Planeta, Number> brSat =
            new TableColumn<>("Broj satelita");
```



```

brSat.setCellValueFactory(e -> e.getValue().brojSatelita());
brSat.setMinWidth(100.0);
// gasovita
TableColumn<Planeta, Boolean> gas = new TableColumn<>("Gasovita?");
gas.setCellValueFactory(e -> e.getValue().gasovita());
gas.setMinWidth(80.0);

// tabela
TableView<Planeta> tabela = new TableView<>(planete);
tabela.getColumns().addAll(naziv, rbr, brSat, gas);
// kada korisnik selektuje planetu, treba prikazati sliku
TableViewSelectionModel<Planeta> m = tabela.getSelectionModel();
m.selectedItemProperty().addListener(e -> {
    Planeta p = m.getSelectedItem();
    if (p == null)
        pane.setBottom(null);
    else {
        // prikazi centralno-poravnatu sliku
        ImageView slika = p.slika().get();
        pane.setBottom(slika);
        BorderPane.setAlignment(slika, Pos.CENTER);
    }
});

pane.setCenter(tabela);
stage.setScene(new Scene(pane, 640, 480));
stage.setTitle("Planete Suncevog sistema");
stage.show();
}

public static void main(String[] args) {
    launch(args);
}

```