

Laboratorijska vježba 8.

JavaFX View i Scene Builder

8.1. View u JavaFX

View komponenta je odgovorna za prezentacijski sloj aplikacije. Njena glavna uloga je definisanje načina na koji se prikazuje korisnički interfejs (eng. *user interface* - UI) i prikaz podataka iz modela korisniku. U JavaFX aplikacijama, View komponenta pomoću elemenata kao što su dugmadi, labele, tekstualna polja, tabele i sl. definira raspored elemenata (eng. *layout*) i izgled UI-a. View komponenta je u prošlim laboratorijskim vježbama implementirana kao konzolni ispis, dok će se u ovoj vježbi implementirati grafički korisnički interfejs korištenjem mogućnosti koje pruža JavaFX. Najprije će biti analiziran polazni kod, koji se automatski generiše prilikom kreiranja bilo koje JavaFX aplikacije. Konkretno, razmatrat će se klase *HelloController* i *HelloApplication*.

U Listingu 1 je prikazan programski kod automatski generisane klase *HelloController*, koja predstavlja kontroler komponentu *Hello world* aplikacije. Anotacija **@FXML** se koristi za označavanje polja i metoda definisanih u FXML datoteci, što predstavlja deklarativni način definisanja izgleda korisničkog interfejsa u JavaFX-u (kada se kaže "deklarativni način", misli se da su struktura i izgled korisničkog interfejsa opisani korištenjem *markup* jezika ili strukturiranog formata, umjesto da se grade programski, tj. imperativno, u programskom kodu). FXML je jezik temeljen na XML-u koji se koristi za opisivanje korisničkog interfejsa, a **@FXML** povezuje Java kod s elementima definiranim u FXML datoteci. U Listingu 1 anotacija **@FXML** se koristi za atribut tipa *Label welcomeText*, kao i uz metodu *onHelloButtonClick()*, što ukazuje da su ti elementi definisani u odgovarajućoj FXML datoteci (koja predstavlja samu View komponentu) i da će biti povezani sa kontrolerom.

```
public class HelloController {  
    @FXML  
    private Label welcomeText;  
  
    @FXML  
    protected void onHelloButtonClick() {  
        welcomeText.setText("Welcome to JavaFX Application!");  
    }  
}
```

Listing 1. Automatski generisana klasa *HelloController*

Deklaracija JavaFX *Label* elementa je obavljena korištenjem polja *welcomeText*. Ovo polje odgovara *Label* elementu definiranom u FXML datoteci, pri čemu taj element ima id "*welcomeText*" i prikazat će poruku dobrodošlice korisniku. Anotacija *@FXML* osigurava da je ovaj element povezan s odgovarajućom komponentom grafičkog korisničkog interfejsa predstavljenog u FXML datoteci. Metoda *onHelloButtonClick()* je metoda kontrolera koja će se pozvati kada se na korisničkom interfejsu desi klik na dugme. Izabrani naziv je proizvoljan, ali je dobra praksa koristiti naziv koji označava radnju koja se izvršava. Ova metoda je povezana sa *Button* elementom u FXML datoteci, tako da će se metoda izvršiti kada se desi klik na to dugme.

Klikom na *hello-view.fxml* datoteku moguće je vidjeti sadržaj FXML datoteke, koja predstavlja UI polazne JavaFX aplikacije (Listing 2). Raspored elemenata korisničkog interfejsa je definisan koristeći **VBox** *layout* tj. način rasporeda elemenata. *VBox* je kontejnerski element koji vertikalno raspoređuje svoju djecu elemente (tj. elemente korisničkog interfejsa). Postavljanjem atributa *alignment* na vrijednost "*CENTER*" se određuje da će svi djeca elementi unutar *VBox* elementa biti centrirani horizontalno. Postavljanjem atributa *spacing* na vrijednost 20.0 piksela (px), određuje se vertikalni razmak između elemenata koji se nalaze unutar *VBox* elementa. Dakle, svaki element koji se nalazi unutar *VBox* elementa će imati razmak od 20 piksela (vertikalno) od sljedećeg elementa. Atribut vrijednosti *xmlns:fx="http://javafx.com/fxml"* osigurava da JavaFX prepozna i ispravno obrađuje FXML sintaksu. Atributom vrijednosti *fx:controller="com.example.javafxdemo.controller.HelloController"* FXML datoteka se povezuje sa klasom odgovarajućeg Java kontrolera. U ovom slučaju, FXML datoteka je povezana sa klasom *HelloController* koja se nalazi u paketu *com.example.lv08*.

```
<VBox alignment="CENTER" spacing="20.0"
xmlns:fx="http://javafx.com/fxml"
    fx:controller="com.example.lv08.HelloController">
    <padding>
        <Insets bottom="20.0" left="20.0" right="20.0" top="20.0"/>
    </padding>

    <Label fx:id="welcomeText"/>
    <Button text="Hello!" onAction="#onHelloButtonClick"/>
</VBox>
```

Listing 2. FXML kod polaznog korisničkog interfejsa

Element **padding** je ugniježđeni element unutar *VBox* elementa, koji dodaje *padding* oko *VBox* kontejnera, tj. dodaje prostor između sadržaja *VBox* elementa i njegove granice. U ovom primjeru, *padding* je određen elementom **Insets**. Ovaj element određuje prostor koji treba dodati oko sadržaja na sve četiri strane elementa (donju, lijevu, desnu i gornju), a sve vrijednosti su (u ovom primjeru) postavljene na 20.0 piksela. Element **Label** će sadržavati tekst koji će se prikazati korisniku. Atribut *fx:id* povezuje *Label* element sa poljem u klasi kontrolera, dopuštajući kontroleru da ga referencira i njime programski manipulše (kao što je i navedeno u kontroleru, ovaj element će se referencirati pomoću varijable *welcomeText*). Inicijalno, ovaj *Label* element ne prikazuje nikakav tekst (budući da nije naveden *text* atribut u FXML datoteci). Međutim,

kontroler može dinamički ažurirati tekst na osnovu korisničkih radnji ili logike rada same aplikacije. Konačno, **Button** element predstavlja dugme na koje se može kliknuti na korisničkom interfejsu. Postavljanjem vrijednosti atributa *text* na vrijednost *Hello!*, određuje se tekst koji će prikazati unutar dugmeta. Atribut *onAction* specificira metodu koja upravlja događajima (eng. *event handler*) u kontroleru. Kada se dugme klikne, poziva se metoda *onHelloButtonClick()* iz klase *HelloController*. Simbol # označava da je riječ o referenci na metodu u klasi kontrolera.

Klasa *HelloApplication* predstavlja glavnu klasu JavaFX aplikacije i prikazana je u Listingu 3. Metoda *start()* predstavlja ključnu metodu bilo koje JavaFX aplikacije. Kada se JavaFX aplikacija pokrene, ovu metodu automatski poziva JavaFX *runtime*. Ova metoda je mjesto gdje se podešava glavni prozor (poznat kao *scene*) i sadržaj koji će se prikazati. Kao što je vidljivo u programskom kodu, metoda *start()* je naslijeđena iz klase *Application* i kao parametar prima *Stage* objekat, koji predstavlja glavni prozor JavaFX aplikacije; to je mjesto gdje će se prikazati korisnički interfejs. Metoda baca *IOException* izuzetak, zbog toga što se može desiti da učitavanje FXML datoteke dovede do bacanja ovog izuzetka ako se datoteka ne može pronaći ili pročitati.

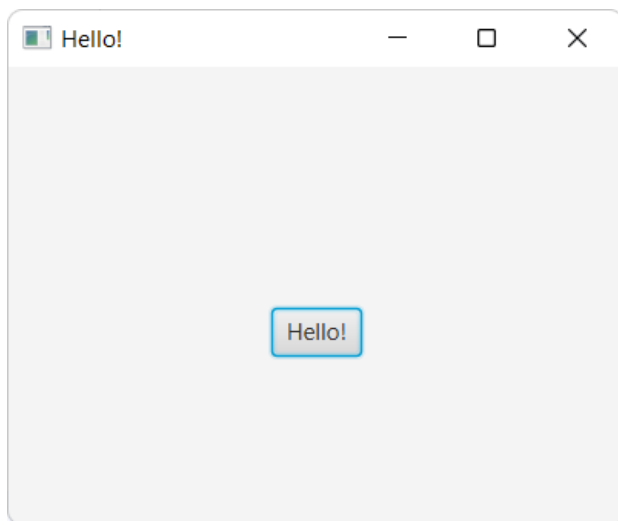
```
public class HelloApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException
    {
        FXMLLoader fxmlLoader = new
FXMLLoader(HelloApplication.class.getResource("hello-view.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 320, 240);
        stage.setTitle("Hello!");
        stage.setScene(scene);
        stage.show();
    }

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

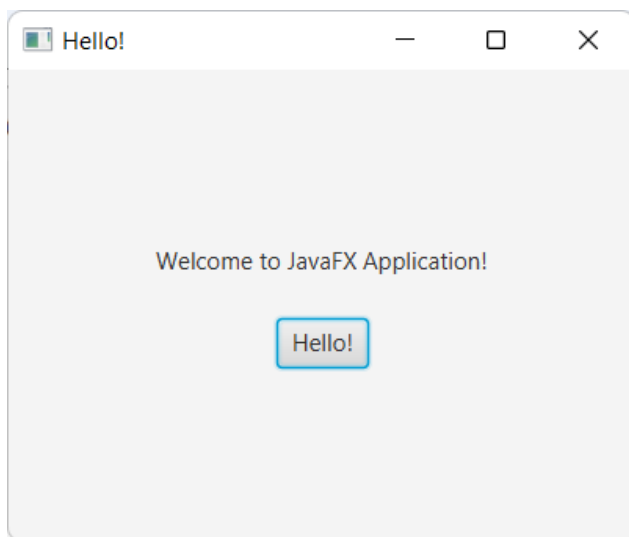
Listing 3. Klasa HelloApplication koja pokreće JavaFX aplikaciju

FXMLLoader je u metodi *start()* odgovoran za učitavanje FXML datoteke. Pomoću poziva metode *HelloApplication.class.getResource("hello-view.fxml")* se locira FXML datoteka pod nazivom *hello-view.fxml*. Metoda *getResource()* se koristi za učitavanje resursa iz *classpath* lokacije. Zatim, *FXMLLoader* analizira FXML datoteku i kreira korisnički interfejs na način koji je naveden u toj datoteci. U JavaFX *Scene* predstavlja spremnik za sav vizuelni sadržaj u JavaFX aplikaciji. Sadrži elemente koji određuju raspored elemenata (npr. *VBox*, *HBox*, *GridPane* itd.) i sve elemente korisničkog interfejsa kao što su dugmadi, labele, tekstualna polja, itd. Dakle, raspored elemenata se postavlja na "scenu" (*scene*) nakon čega "scenu" postavljaju na "pozornicu" (*stage*).

Metoda `FXMLLoader.load()` učitava i vraća korijenski čvor korisničkog interfejsa kako je to definirano u FXML datoteci (obično je riječ o elementu koji određuje raspored elemenata, poput `VBox` ili `GridPane`). Ovaj korijenski čvor se proslijeđuje kao prvi parametar `Scene` konstruktoru. Scena se kreira na osnovu korijenskog čvora iz FXML datoteke (kao prvog parametra), te širine i visine (kao drugog i trećeg parametra, respektivno). U ovom primjeru, prozor je postavljen na 320 piksela širine i 240 piksela visine. Pozivom `setTitle()` metode nad objektom scene se postavlja naslov prozora. U ovom slučaju, u naslovnoj traci prozora aplikacije će pisati "Hello!". Pozivom `stage.show()` metode se prozor ("pozornica", `stage`) čini vidljivom, tj. na ekranu se prikazuje prozor koji sadrži postavljenu scenu. Metoda `main()` klase `HelloApplication` sadrži samo poziv metode `launch()` koja pokreće JavaFX aplikaciju postavljanjem JavaFX `runtime`-a i pozivanjem metode `start()`. Ovoj metodi nije potrebno proslijeđivati bilo kakve argumente. Pokretanje polazne aplikacije (tj. pokretanje datoteke `HelloApplication`) otvara prozor prikazan na Slici 1, a klikom na dugme sa tekstom `Hello!` prikazuje se tekst u `Label` elementu (Slika 2).



Slika 1. Prikaz polazne JavaFX aplikacije



Slika 2. Klik na dugme sa tekstom `Hello!` prikazuje tekst unutar labele

Nakon analize polaznog primjera FXML datoteke i povezivanja *Controller* i *View* komponenti, u nastavku će biti opisana implementacija ovih komponenti na pokaznom primjeru iz laboratorijske vježbe 7. Početni programski kod za ovu laboratorijsku vježbu je implementacija pokazne JavaFX aplikacije sa laboratorijske vježbe 7. Neka *View* komponenta treba uraditi prikaz podataka o osobama u listi i omogućiti ažuriranje podataka pojedinih osoba iz te liste kroz formu sa poljima za unos (tj. ažuriranje) podataka. Najprije će biti kreirana FXML datoteka, koja sadrži sve elemente korisničkog interfejsa. Pritom će se koristiti sljedeći elementi:

- *VBox* za kreiranje rasporeda elemenata;
- *ListView* za prikaz podataka o svim osobama u vidu liste;
- *TextField* za polja za unos tekstualnih podataka (u ovom slučaju, to su ime, prezime, adresa i matični broj);
- *DatePicker* za unos datuma rođenja;
- *ChoiceBox* za izbor uloge;
- *Button* za potvrdu unosa i ažuriranje osobe u modelu;
- *Label* za jasniji prikaz različitih dijelova prikazanog ekrana, kao i za prikaz poruka korisniku.

Inicijalni sadržaj FXML datoteke je prikazan u Listingu 4. Osim već opisanih elemenata (poput *VBox*, *Label*, *Button*), u FXML datoteci se koriste i neki novi elementi, koji imaju specifične atribute. *ListView* element prikazuje listu elemenata. Korisnik može izabrati svaki od elemenata iz te liste. Atributi *prefHeight* i *prefWidth* postavljaju preferiranu visinu i širinu elementa. Ovi atributi nisu jedinstveni za *ListView* tip elementa, već se mogu koristiti i sa drugim tipovima elemenata.

```
<VBox alignment="CENTER" spacing="20.0"
xmlns:fx="http://javafx.com/fxml" xmlns="http://javafx.com/javafx">

    <Label fx:id="ucitavanjeLabel" />

    <Label text="Lista osoba" />
    <ListView fx:id="osobeListView" prefHeight="300.0"
prefWidth="300" />

    <Label text="Forma za pregled i azuriranje detalja osobe" />

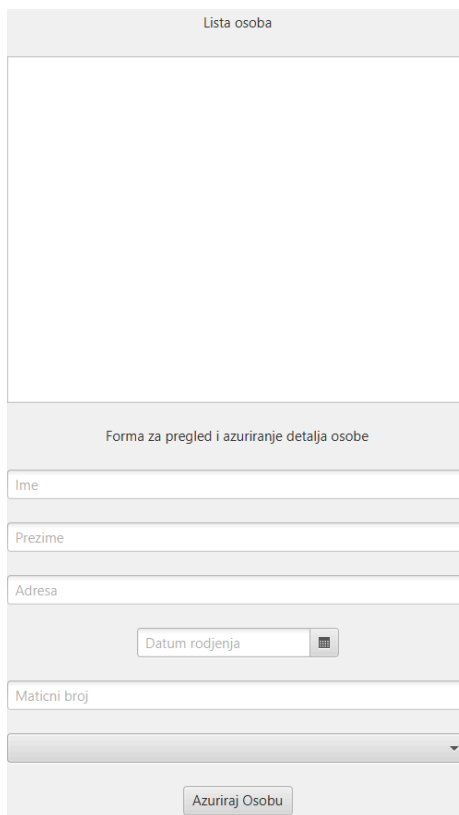
    <TextField fx:id="imeField" promptText="Ime" />
    <TextField fx:id="prezimeField" promptText="Prezime"/>
    <TextField fx:id="adresaField" promptText="Adresa"/>
    <DatePicker fx:id="datumRodjenjaPicker" promptText="Datum
rodjenja"/>
    <TextField fx:id="maticniBrojField" promptText="Maticni broj"/>

    <ChoiceBox fx:id="ulogaChoiceBox" prefWidth="400.0"/>
```

```
<Button fx:id="azurirajOsobuButton" text="Azuriraj Osobu"/>
<Label fx:id="porukaLabel" visible="false"/>
</VBox>
```

Listing 4. FXML datoteka koja definiše izgled korisničkog interfejsa

Element *TextField* omogućava korisniku unos jednog reda teksta. Obično se koristi za unos tekstualnih podataka, kao što je korisničkog ime ili šifra a u ovom primjeru koristiti će se za unos imena, prezimena, adrese i matičnog broja. Atribut *promptText* se odnosi na tekst koji je će se prikazati u polju kada je ono prazno (tzv. *placeholder*). Element *ChoiceBox* predstavlja vrstu padajućeg menija, koja korisniku omogućava izbor jedne stavke iz unaprijed definiranog spiska. Ovakva FXML datoteka proizvodi raspored elemenata kao na Slici 3. Važno je napomenuti da se za prikaz rasporeda FXML elemenata kao na Slici 3 koristi okruženje *SceneBuilder*, koje će biti opisano u nastavku laboratorijske vježbe.



Slika 3. Prikaz izgleda kontrola za unos podataka o osobi

Potrebno je obratiti pažnju da element *VBox* nema postavljen atribut *fx:controller*, što znači da *View* komponenta nije povezana sa odgovarajućim kontrolerom. Povezivanje će se, u ovom slučaju, uraditi u *start()* metodi *HelloApplication* klase, zbog specifičnosti instanciranja kontrolera na osnovu modela. Izgled ove metode nakon dodavanja navedene izmjene je prikazan u Listingu 5.

```
public void start(Stage stage) throws IOException
{
    OsobaModel osobaModel = new OsobaModel();

    FXMLLoader fxmlLoader = new
FXMLLoader(HelloApplication.class.getResource("hello-view.fxml"));
    fxmlLoader.setController(new OsobaController(osobaModel));

    Scene scene = new Scene(fxmlLoader.load(), 300, 700);
    stage.setTitle("Dodaj osobu!");
    stage.setScene(scene);
    stage.show();
}
```

Listing 5. start() metoda u kojoj se instancira model i postavlja za kontroler FXML forme

8.2. Povezivanje Controller i View komponenti u JavaFX

Kontroler treba modificirati tako da sadrži ispravne reference na polja View komponente. Dakle, kontroler treba imati reference na sve elemente za koja je definiran atribut *fx:id*. Pojedina polja u FXML datoteci nemaju postavljen ovaj atribut, ali je to urađeno zbog toga što ta polja neće doživljavati promjene u toku rada same aplikacije (npr. *Label* element u kojem je sadržan tekst “*Lista osoba*” se koristi samo kako bi poboljšao čitljivost korisničkog interfejsa, što znači da ga nije potrebno referencirati u kontroleru). Sva polja se referenciraju korištenjem anotacije *@FXML*. Osim polja View komponente, u kontroleru se trebaju nalaziti i atributi koji su bili prisutni i prilikom implementacije kontrolera u laboratorijskoj vježbi 7. Novododani atribut je i *izabranaOsoba*, koji označava osobu koju je korisnik izabrao u *ListView* elementu. U Listingu 6 prikazani su svi atributi kontrolerske klase, kao i konstruktor koji sada prima samo jedan parametar, koji predstavlja model klasu.

```
public class OsobaController
{
    @FXML
    private Label ucitavanjeLabel;
    @FXML
    private ListView<Osoba> osobeListView;
    @FXML
    private TextField imeField;
    @FXML
    private TextField prezimeField;
    @FXML
    private TextField adresaField;
    @FXML
    private DatePicker datumRodjenjaPicker;
    @FXML
    private TextField maticniBrojField;
    @FXML
```



```
private ChoiceBox<Uloga> ulogaChoiceBox;  
@FXML  
private Button azurirajOsobuButton;  
@FXML  
private Label porukaLabel;  
  
private OsobaModel model;  
  
private ObservableList<Osoba> osobeObservableList =  
FXCollections.observableArrayList();  
  
private Osoba izabranaOsoba;  
  
public OsobaController(OsobaModel model) {  
    this.model = model;  
}
```

Listing 6. Atributi i konstruktor kontrolerske klase

JavaFX aplikacija je osmišljena na način da korisnik izabere jednu osobu iz liste osoba (tj. iz *ListView* elementa), nakon čega se polja za unos popunjavaju odgovarajućim podacima te osobe. Kako bi korisnik mogao birati objekte iz liste, podaci se učitavaju u model pozivom metode *napuni()* (a mogu se pozivati i metode *napuniPodatkeIzTxtDatoteke()* i *napuniPodatkeIzXmlDatoteke()*). Nakon što se polja ispune odgovarajućim podacima, korisnik može ažurirati podatke izabrane osobe. Klikom na dugme *azurirajOsobuButton()* se poziva metoda za ažuriranje osobe. Rezultat izvršavanja ove metode će se ispisati kao poruka korisniku unutar labela *porukaLabel*.

U JavaFX aplikaciji kontroler može posjedovati posebnu metodu koja se naziva ***initialize()*** (metoda inicijalizacije kontrolera). Ova metoda se automatski poziva od strane JavaFX *framework*-a nakon što je FXML datoteka učitana i nakon što su sva polja sa ***@FXML*** anotacijom ubačena (tj. povezana s odgovarajućim UI komponentama u FXML datoteci). Ova metoda se obično koristi za postavljanje početnog stanja komponenti korisničkog interfejsa, popunjavanje podataka, dodavanje *event listeners* i izvođenje svih drugih zadataka postavljanja koji se trebaju desiti prije nego što se korisnički interfejs prikaže korisniku. Metodu nije potrebno eksplicitno pozivati, jer je JavaFX automatski poziva tokom faze inicijalizacije kontrolera. Dakle, ova metoda se poziva nakon učitavanja FXML datoteke, ali prije nego što se prozor aplikacije (*stage*) prikaže korisniku.

Prilikom inicijalizacije kontrolera, najprije se model treba ispuniti podacima, na način prikazan u Listingu 7. Ispravno završena operacija punjenja podataka postavlja tekst labela *ucitavanjeLabel* na "Ucitani podaci" i boju pozadine labela na zelenu. Kada je riječ o postavljanju teksta, koristi se metoda *setText()*, dok se za postavljanje stila koristi metoda *setStyle()*. Metoda *setStyle()* omogućava primjenu ugrađenih stilova iz CSS (*Cascading Style Sheets*) direktno na JavaFX UI komponentu. Ova metoda kao parametar prima niz koji sadrži svojstva (eng. *properties*) i vrijednosti (eng. *values*) CSS stila i primjenjuje te stilove na element JavaFX scene. CSS stilovi se pišu u obliku "svojstvo:vrijednost". Ukoliko je potrebno koristiti više

stilova nad istim elementom, onda svaki od njih treba razdvojiti znakom tačka-zarez. Npr. `"-fx-font-size: 20px; -fx-text-fill: blue;"`. Detalji CSS-a neće biti obrađeni u okviru laboratorijskih vježbi, tako da je dovoljno znati samo osnove. U ovom primjeru će se za labelu koristiti CSS svojstvo `-fx-background-color`, koje će biti postavljeno na vrijednost *green*.

```
@FXML
public void initialize()
{
    model.napuni();

    ucitavanjeLabel.setText("Ucitani podaci");
    ucitavanjeLabel.setStyle("-fx-background-color: green;");

    azurirajOsobuButton.setText("Azuriraj");

    ulogaChoiceBox.getItems().addAll(Uloga.STUDENT,
    Uloga.NASTAVNO_OSOBLJE);

    osobeObservableList.addAll(model.dajSveOsobe());
    osobeListView.setItems(osobeObservableList);
}
```

Listing 7. Inicijalizacija podataka prilikom pokretanja kontrolera

Nakon učitavanja labele, tekst dugmeta *azurirajOsobuButton* će biti dinamički postavljen na vrijednost "Azuriraj" koristeći metodu *setText()*. U *ChoiceBox* elementu će biti smještene dvije opcije, koje predstavljaju uloge korisnika. Za popunjavanje ovog elementa odgovarajućim opcijama koristi se kombinacija metoda *getItems()* (koja vraća listu stavki sadržanih unutar ovog elementa; ova lista je tipa *ObservableList* i sadrži sve stavke koje korisnik može izabrati) i *addAll()* (koja dodaje više stavki u *ChoiceBox* listu). Lista osoba *osobe* treba sadržavati kolekciju osoba koje se nalaze u modelu. Kako bi se dohvatile osobe iz modela, poziva se metoda *dajSveOsobe()*. Nakon toga se *ListView* element (referenciran varijablom *osobeListView*) povezuje sa listom *osobeObservableList* koristeći metodu *setItems()*. Nakon pozivanja metode *setItems()*, element *ListView* će prikazati stavke iz liste *osobeObservableList*, a sve promjene napravljene nad varijablom *osobeObservableList* (npr. dodavanje ili uklanjanje osoba) automatski će se odraziti na *ListView* zbog prirode ponašanja ovog tipa liste.

U metodi *initialize()* neophodno je implementirati i *event listeners* za klik na dugme za ažuriranje i za izbor osobe iz *ListView* elementa. Za dugme *azurirajOsobuButton* potrebno je dodati akciju koja se dešava kada korisnik klikne na to dugme. Metoda *setOnAction()* dugmetu dodjeljuje *event listener*, dok se u metodi *handle* anonimne unutrašnje klase *EventHandler<ActionEvent>* nalazi programski kod koji se izvršava kada se desi klik dugmeta, kao što je prikazano u Listingu 8 (u ovom slučaju to je poziv metode *azurirajOsobu()*).

```
azurirajOsobuButton.setOnAction(new EventHandler<ActionEvent>() {  
    @Override  
    public void handle(ActionEvent event) {  
        azurirajOsobu();  
    }  
});
```

Listing 8. Dodavanje event listenera za dugme

Kao što je prethodno spomenuto, klik dugmeta samo poziva metodu za ažuriranje podataka o izabranoj osobi, koja je nazvana *azurirajOsobu()*. Riječ je o pomoćnoj metodi, koja provjerava da li je na korisničkom interfejsu u okviru liste izabrana (tj. označena mišem) neka osoba. Ukoliko jeste, metoda čita vrijednosti koje se nalaze u poljima za ime, prezime, adresu, datum rođenja, matični broj i ulogu, validira navedena polja i poziva metodu *azurirajOsobu()* iz modela koristeći učitane podatke. Osim toga, šalje korisniku poruku da je ažuriranje uspješno završeno ili da je došlo do neke greške. Poruka se ispisuje u *porukaLabel* elementu, koji se prvo čini vidljivim pozivom metode *setVisible()* sa parametrom *true*. Tekst poruke se postavlja pozivom metode *setText()* sa parametrom koji sadrži *String* varijablu teksta poruke. Na samom kraju metode *azurirajOsobu()*, poziva se metoda *refresh()* nad varijablom koja predstavlja *ListView* element (tj. *osobeListView*), kako bi se promjene ispravno prikazale na korisničkom interfejsu. Kompletan programski kod metode *azurirajOsobu()* je prikazan u Listingu 9.

```
private void azurirajOsobu() {  
    if(izabranaOsoba != null) {  
        // procitaj sadržaj input polja  
        String ime = imeField.getText();  
        String prezime = prezimeField.getText();  
        String adresa = adresaField.getText();  
        LocalDate datumRodjenjaLocal =  
        datumRodjenjaPicker.getValue();  
        String maticniBroj = maticniBrojField.getText();  
        Uloga uloga = ulogaChoiceBox.getValue();  
        // validacija polja forme  
        if (ime.isEmpty() || prezime.isEmpty() || adresa.isEmpty()  
        || maticniBroj.isEmpty() || datumRodjenjaLocal==null ||  
        uloga==null) {  
            porukaLabel.setVisible(true);  
            porukaLabel.setText("Sva polja moraju biti popunjena!");  
            return;  
        }  
        Date datumRodjenja =  
        Date.from(datumRodjenjaLocal.atStartOfDay(ZoneId.systemDefault()).to  
        Instant());  
        String poruka = model.azurirajOsobu(izabranaOsoba.getId(),  
        ime, prezime, adresa, datumRodjenja, maticniBroj, uloga);  
        porukaLabel.setVisible(true);  
        porukaLabel.setText(poruka);  
        osobeListView.refresh();  
    }  
}
```

```
}
```

Listing 9. Metoda za ažuriranje podataka o odabranoj osobi

U metodi *initialize()* je ostalo još da se doda *event listener* na *osobeListView* varijablu, koji odgovara na promjene u odabiru stavki iz liste. Kada korisnik odabere drugu stavku u *ListView* elementu, poziva se definisani *event listener*, koji obavlja određene radnje ažuriranja korisničkog interfejsa i internih podataka, i to pozivom *getSelectionModel()* metode nad *osobeListView*, koji je odgovoran za praćenje koja je stavka trenutno odabrana u listi. *SelectionModel* objekat pruža metode i svojstva za upravljanje odabirom u *ListView* elementu, uključujući dohvaćanje odabrane stavke, programski odabir određenih stavki ili osluškivanje promjena u odabiru. U ovom primjeru će se koristiti metoda *selectedItemProperty()*, koja vraća svojstvo koje predstavlja trenutno odabranu stavku u *ListView* elementu. Ovo svojstvo se mijenja svaki put kada korisnik odabere novu stavku ili poništi odabir postojeće. S obzirom da je ovo svojstvo *observable*, to znači da se na njega može dodati *event listener* koji će pratiti i reagovati na promjene izbora stavki. *Event listener* se dodaje metodom *addListener()*, koja će biti implementirana kao lambda izraz sa tri parametra:

- *observable* (predstavlja *observable* vrijednost koja se prati radi promjena - u ovom slučaju, to je *selectedItemProperty*),
- *staraVrijednost* (prethodno odabrana stavka u *ListView*; predstavlja stavku koja je odabrana prije promjene),
- *novaVrijednost* (trenutno odabrana stavka u *ListView*; predstavlja stavku koju je korisnik upravo odabrao).

U samom *event listener*-u se provjerava da li je izabrana ijedna osoba iz liste. Ukoliko jeste, parametar *izabranaOsoba* se postavlja na vrijednost parametra *novaVrijednost*, popunjavaju se polja forme sa podacima o novoj osobi, a sakriva se poruka sadržana u varijabli *porukaLabel*. Sve što je prethodno opisano prikazano je u okviru Listinga 10.

```
osobeListView.getSelectionModel().selectedItemProperty().addListener((observable, staraVrijednost, novaVrijednost) -> {  
    if (novaVrijednost != null) {  
        // azuriranje varijable koja predstavlja trenutno izabranu osobu  
        izabranaOsoba = novaVrijednost;  
        // ispunjavanje polja detaljima izabrane osobe  
        ispuniPolja(novaVrijednost);  
        // sakrij labelu koja sadrzi poruku  
        porukaLabel.setVisible(false);  
    }  
});
```

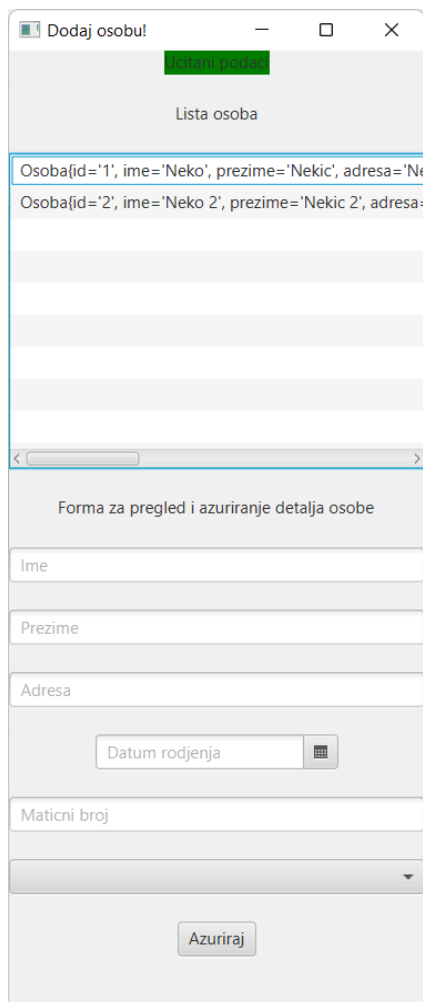
Listing 10. Event listener za ListView objekat

Metoda *ispuniPolja()* (Listing 11) je pomoćna i prima jedan parametar koji predstavlja osobu. Na osnovu podataka te osobe, metoda ispunjava odgovarajuća polja *View* komponente. Ovime su implementirane i povezane *Controller* i *View* komponente, što znači da je aplikaciju moguće

pokrenuti. Pokretanje aplikacije otvara prozor kao na Slici 4. Izborom bilo koje osobe iz *ListView* komponente, moguće je ažurirati podatke o njoj. Međutim, raspored elemenata nije dobro osmišljen. Naime, elementi su vertikalno “naredani” jedan na drugi što prozor čini “izduženim” po visini. Zbog ovoga, potrebno je veću pažnju posvetiti dizajnu korisničkog interfejsa. Naravno, sve promjene korisničkog interfejsa se trebaju raditi u FXML datoteci koja predstavlja *View* komponentu JavaFX aplikacije. Ipak, iako moguće, nezgodno je pisati sadržaj FXML datoteka i ručno dizajnirati korisnički interfejs. Zbog toga će se u narednom dijelu laboratorijske vježbe detaljnije obraditi alat za dizajniranje JavaFX korisničkog interfejsa *Scene Builder*.

```
private void ispuniPolja(Osoba osoba) {  
    imeField.setText(osoba.getIme());  
    prezimeField.setText(osoba.getPrezime());  
    adresaField.setText(osoba.getAdresa());  
  
    datumRodjenjaPicker.setValue(osoba.getDatumRodjenja().toInstant().atZone(ZoneId.systemDefault()).toLocalDate());  
    maticniBrojField.setText(osoba.getMaticniBroj());  
    ulogaChoiceBox.setValue(osoba.getUloga());  
}
```

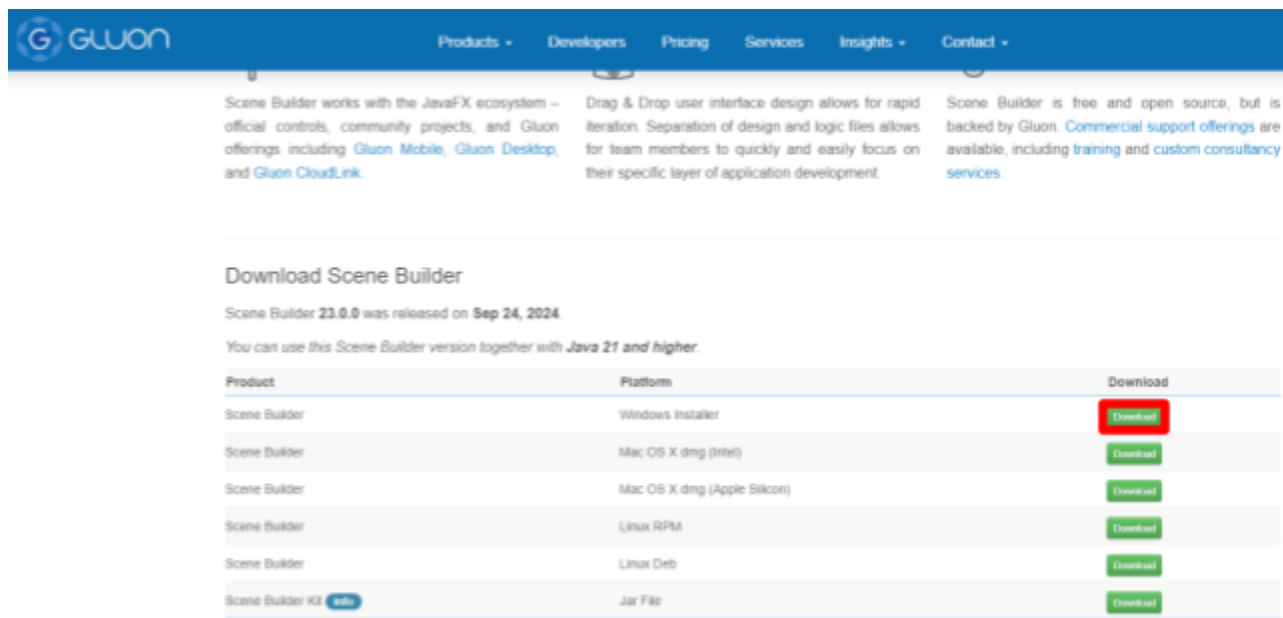
Listing 11. Pomoćna metoda za popunjavanje polja korisničkog interfejsa za datu osobu



Slika 4. Prikaz izgleda korisničkog interfejsa nakon pokretanja JavaFX aplikacije

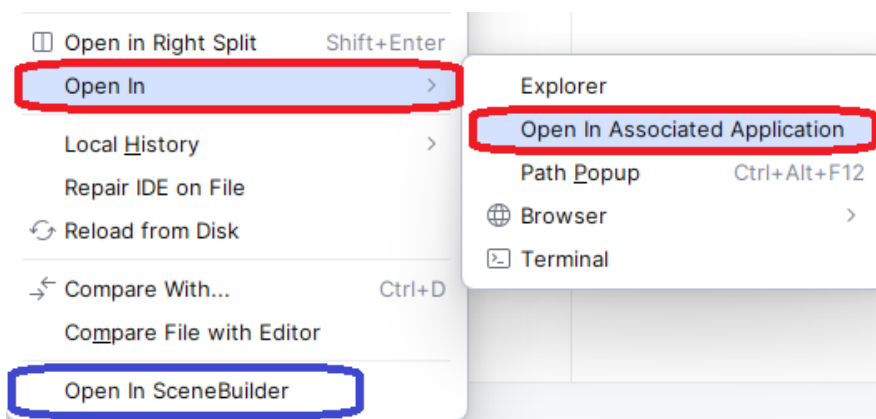
8.3. Scene Builder

Scene Builder je alat za vizualni dizajn JavaFX aplikacija, koji omogućava kreiranje korisničkog interfejsa tzv. *drag-and-drop* komponenti (kao što su dugmadi, oznake, tekstualna polja itd.) na scenu, umjesto ručnog pisanja FXML koda. Ovime se pojednostavljuje proces dizajniranja JavaFX korisničkih interfejsa. *Scene Builder* automatski generiše FXML kod za dizajn korisničkog interfejsa, koji je vizualno kreiran putem ovog alata. Iako se FXML kod ne mora ručno pisati, uvijek je moguće uraditi pregled kreirane FXML datoteke i izmijeniti je, ako postoji potreba za tim. Kako bi se instalirao *Scene Builder*, prvo je preuzeti ga koristeći [sljedeći link](#). U prvom koraku potrebno je odabrati odgovarajući operativni sistem, a zatim odabrati opciju **Download**, kao što je prikazano na Slici 5.



Slika 5. Odabir opcije za preuzimanje Scene Builder alata

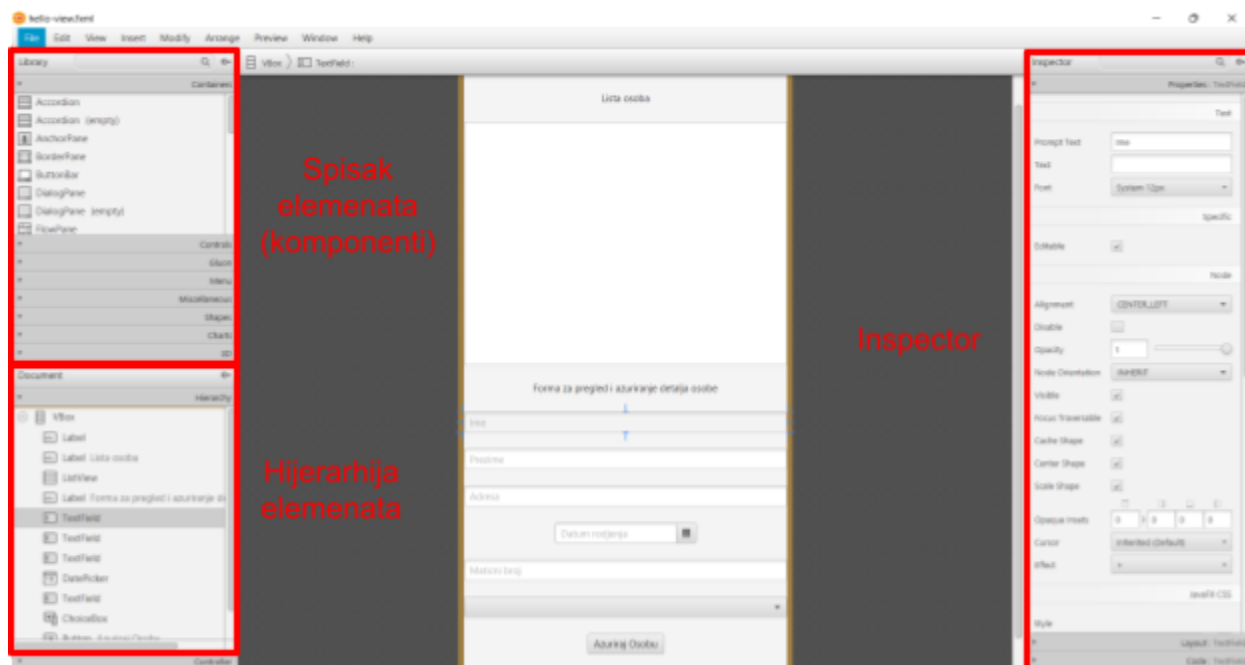
Nakon preuzimanja alata, potrebno je pratiti korake instalacije. Nakon uspješne instalacije, moguće je koristiti *Scene Builder* za otvaranje FXML datoteka na dva načina. Prvi način podrazumijeva konfiguraciju da se *Scene Builder* automatski pojavljuje kao mogućnost za otvaranje kada se izvrši desni klik na neku FXML datoteku (opcija **Open in Scene Builder** označena plavom bojom na Slici 6). Da bi se to postiglo, pratiti uputstvo na [sljedećem linku](#). Druga, jednostavnija mogućnost je odabir opcije **Open in** → **Open in Associated Application** kada se izvrši desni klik na neku FXML datoteku (označena crvenom bojom na Slici 6), pri čemu se nakon toga treba navigirati u instalacijski direktorij *Scene Builder* alata i odabrati odgovarajuću izvršnu datoteku.



Slika 6. Odabir opcija za otvaranje FXML datoteka u Scene Builder alatu

Nakon odabira jedne od dvije moguće opcije za otvaranje FXML datoteka koristeći *Scene Builder* alat, na ekranu bi se trebao prikazati prozor kao na Slici 7. Na istoj slici prikazani su najvažniji dijelovi *Scene Builder* alata. Sa lijeve strane (u gornjem uglu) se nalazi spisak

elementa koje je moguće dodati na scenu. Elementi su sortirani po tipovima (npr. *containers*, *controls* itd.) a mogu se i pretraživati unosom odgovarajućeg naziva kontrole u polje za pretragu. Sa lijeve strane (u donjem uglu) su prikazani svi elementi i njihov hijerarhijski poredak. Dakle, ovdje je moguće vidjeti redoslijed u kojem će elementi biti prikazani, kao i odnose roditelj-dijete. U desnom dijelu okruženja je *Inspector* koji pruža uvid u raspored elemenata i njihova svojstva. Klikom na određeni element, koji je postavljen na scenu (tj. u centralnom dijelu *Scene Builder*-a), u desnom dijelu je moguće vidjeti i modificirati određena svojstva, raspored i slično (npr. sadržaj tekstualnog polja, poravnanje, udaljenost elemenata od ivice, font itd.).



Slika 7. Prikaz FXML datoteke u *Scene Builder* alatu

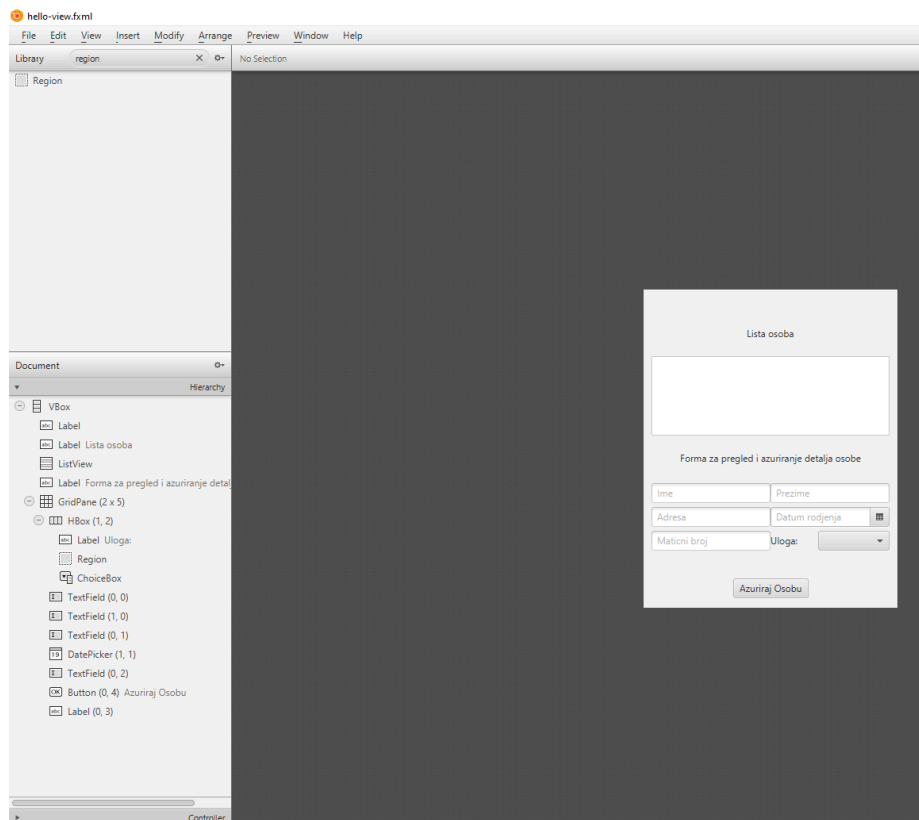
Kako bi prikaz elemenata bio kompaktniji, korištenjem *Scene Builder* alata bit će urađene modifikacije *View* komponente. Najprije, za raspored elemenata, koji predstavljaju polja forme, koristit će se **GridPane** element. Ovaj element omogućava kreiranje rasporeda u obliku mreže (eng. *grid*). Mreža se sastoji od određenog broja redova i kolona. Redove i kolone je moguće dodati desnim klikom na *GridPane* element, prelaskom kursorom miša preko *Grid Pane* i izborom jedne od opcija *Add Row Above/Below*, *Add Column Before/After*. Neka se sada želi formirati mreža tako da sadrži 5 redova i jednu kolonu:

- U prva tri reda potrebno je smjestiti polja za unos svih podatka o osobi sa po dva polja u svakom redu (dakle, prvi red sadrži polja za unos imena i prezimena, drugi red polja za unos adrese i izbor datuma rođenja, treći red polje za unos matičnog broja i izbor uloge).
- U četvrti red potrebno je smjestiti labelu za ispis poruke korisniku.
- U peti red potrebno je smjestiti dugme za ažuriranje podataka o osobi.

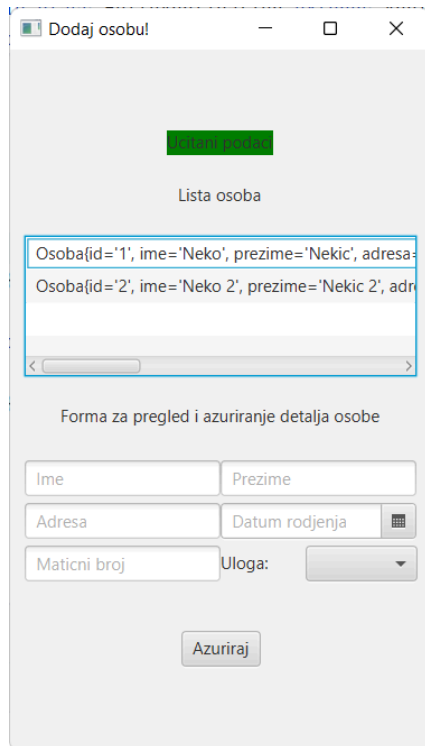
Iako se elementi mogu smjestiti u odgovarajuće ćelije mreže korištenjem *drag-and-drop* operacije, isto je moguće uraditi klikom na odgovarajući element i otvaranjem *Layout* opcija u

Inspector dijelu. Među *Layout* opcijama, nalazit će se sljedeće opcije: *Row/Column Index* (koja govori o tome koji je indeks reda/kolone u kojem počinje element) i *Row/Column Span* (koja govori o tome na koliko redova/kolona se prostire element). Potrebno je voditi računa da indeksiranje redova počinje od 0, što znači da će npr. polje za unos adrese imati *Row Index* = 1 a *Column Index* = 0, a što odgovara drugom redu i prvoj koloni). Samo će elementi koji predstavljaju labelu za ispis poruke korisniku i dugme za ažuriranje podataka korisnika imati *Row Span* jednak 2, dok će svi ostali elementi imati ovu vrijednost postavljenu na 1 (što je i inicijalna vrijednost).

Obzirom na to da padajući meni za izbor uloge nema nikakav opis pored sebe (niti neku vrstu *placeholder* teksta), njemu je potrebno dodati labelu. Labela i padajući meni moraju biti smješteni u istoj ćeliji mreže, tako da je u tu ćeliju, ustvari, potrebno staviti kontejnerski element **HBox**, koji elemente raspoređuje horizontalno jedan do drugog. Labela bi trebala biti lijevo a padajući meni desno centriran. Stoga se između ova dva elementa uvodi **Region** element za popunjavanje prostora. Modificiranjem vrijednosti minimalne i preferirane širine pojedinih elemenata unutar *HBox*, postiže se željeni raspored elemenata. Labelu za ispis poruke korisniku i dugme za ažuriranje podataka o korisniku horizontalno centrirati izborom opcije *HorizontalAlignment* u *Inspector* dijelu pod nazivom *Layout*. Kako bi elementi bili ljepše prikazani unutar *VBox*, potrebno je dodati *padding* klikom na *VBox* element i izborom opcije *padding* u *Inspector* dijelu pod nazivom *Layout*. Konačni raspored elemenata *View* komponente je prikazan na Slici 8, dok je izgled pokrenute aplikacije prikazan na Slici 9. Generisana FXML datoteka je prikazana u Listingu 12. U metodi *start()* klase *HelloApplication* moguće je modificirati dimenziju scene tako da ona sada iznosi, naprimjer 300x500.



Slika 8. Konačni raspored elemenata JavaFX aplikacije u Scene Builder alatu



Slika 9. Izgled JavaFX aplikacije nakon pokretanja

```

<VBox alignment="CENTER" spacing="20.0"
xmlns:fx="http://javafx.com/fxml" xmlns="http://javafx.com/javafx">

    <Label fx:id="ucitavanjeLabel" />

    <Label text="Lista osoba" />
    <ListView fx:id="osobeListView" prefHeight="100.0"
prefWidth="300" />

    <Label text="Forma za pregled i azuriranje detalja osobe" />
    <GridPane>
        <columnConstraints>
            <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0"
prefWidth="100.0" />
            <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0"
prefWidth="100.0" />
        </columnConstraints>
        <rowConstraints>
            <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
            <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
            <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
        </rowConstraints>
    </GridPane>

```

```

        <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
        <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
    </rowConstraints>
    <children>
        <HBox alignment="CENTER_LEFT" prefHeight="100.0"
prefWidth="200.0" GridPane.columnIndex="1" GridPane.rowIndex="2">
            <children>
                <Label minWidth="50.0" text="Uloga:" />
                <Region prefHeight="200.0" prefWidth="200.0"
HBox.hgrow="ALWAYS" />
                <ChoiceBox fx:id="ulogaChoiceBox" prefWidth="400.0"
HBox.hgrow="ALWAYS">
                    <HBox.margin>
                        <Insets left="10.0" />
                    </HBox.margin>
                </ChoiceBox>
            </children>
        </HBox>
        <TextField fx:id="imeField" promptText="Ime" />
        <TextField fx:id="prezimeField" promptText="Prezime"
GridPane.columnIndex="1" />
        <TextField fx:id="adresaField" promptText="Adresa"
GridPane.rowIndex="1" />
        <DatePicker fx:id="datumRodjenjaPicker" promptText="Datum
rodjenja" GridPane.columnIndex="1" GridPane.rowIndex="1" />
        <TextField fx:id="maticniBrojField" promptText="Maticni
broj" GridPane.rowIndex="2" />

        <Button fx:id="azurirajOsobuButton" text="Azuriraj Osobu"
GridPane.columnSpan="2" GridPane.halignment="CENTER"
GridPane.rowIndex="4" />
        <Label fx:id="porukaLabel" visible="false"
GridPane.columnSpan="2" GridPane.halignment="CENTER"
GridPane.rowIndex="3" />
    </children>
</GridPane>
<padding>
    <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />
</padding>
</VBox>

```

Listing 12. Automatski generisan FXML kod koji odgovara izgledu JavaFX aplikacije

8.4. Diskusija

8.4.1. Principi povezivanja

U JavaFX, povezivanje (eng. *binding*) je mehanizam koji omogućava svojstvima da ostanu sinhronizirana, što znači da kada se jedno svojstvo promijeni, drugo svojstvo se automatski ažurira. Postoji nekoliko vrsta povezivanja: jednosmjerno (eng. *unidirectional*), dvosmjerno (eng. *bidirectional*) i putem izraza povezivanja (eng. *binding expressions*). Prilikom implementacije *initialize()* metode kontrolera, korišten je *event listener*, koji “osluškuje” promjene *ListView* elementa i poziva pomoćnu metodu *ispuniPolja()*. Međutim, korištenjem principa povezivanja programski kod može postati jasniji i nema potrebe za ručnim postavljanjem vrijednosti u poljima. U tu svrhu, postojeći programski kod će biti modificiran tako da se koristi jednosmjerno i dvosmjerno povezivanje polja za unos i prikaz podataka o izabranoj osobi. Radi jednostavnosti, razmatrat će se samo polja za unos imena i prezimena. Prvo je potrebno zakomentarirati (ili obrisati) liniju koda koja poziva metodu *ispuniPolja()* prilikom implementacije *eventListener*-a nad varijablom *osobeListView*. Nakon toga, varijablu *izabranaOsoba* neophodno je izmijeniti tako da bude tipa *ObjectProperty<Osoba>* (Listing 13). Ovo će zahtijevati i izmjenu metode *azurirajOsobu()* tako da se sada koristi metoda *izabranaOsoba.get().getId()*, umjesto metode *izabranaOsoba.getId()*, kao i izmjenu *eventListener*-a tako da se sada koristi metoda *izabranaOsoba.setValue(novaVrijednost)*, umjesto dodjele *izabranaOsoba = novaVrijednost*.

```
private ObjectProperty<Osoba> izabranaOsoba = new  
SimpleObjectProperty<>();
```

Listing 13. Izmjena tipa varijable u kontroleru

Promjene nad atributom *izabranaOsoba* će se pratiti dodavanjem *eventListener*-a tako da se metodom *bindBidirectional()* poveže odgovarajuće polje za unos i odgovarajuća vrijednost smještena u ovom svojstvu. Ukoliko je prethodna vrijednost bila različita od *null* (tj. prethodno je bila izabrana neka osoba), onda je potrebno pozvati metodu *unbindBidirectional()* da se ukloni veza između odgovarajućeg polja za unos i vrijednosti koja je prethodno bila smještena u atributu *izabranaOsoba*. U Listingu 14 je prikazan način na koji se dodaje *eventListener*, koji osluškuje promjene atributa *izabranaOsoba*.

```
izabranaOsoba.addListener((observable, oldValue, newValue) -> {  
    if(oldValue != null) {  
  
imeField.textProperty().unbindBidirectional(oldValue.imeProperty())  
;  
  
prezimeField.textProperty().unbindBidirectional(oldValue.prezimePro  
perty());  
    }  
    if(newValue != null) {  
  
imeField.textProperty().bindBidirectional(newValue.imeProperty());  
    }  
});
```

```
prezimeField.textProperty().bindBidirectional(newValue.prezimeProperty());  
    }  
});
```

Listing 14. Event listener za atribut izabranaOsoba sa dvosmjernim povezivanjem

Dodavanje ovako definisanog *event listener*-a će dovesti do toga da se bilo koja promjena izabrane osobe iz *list**View* elementa reflektuje na vrijednosti koje se nalaze u poljima za unos. Međutim, promjene u poljima za unos se reflektuju i na vrijednosti izabrane osobe iz *list**View* elementa. Dakle, za razliku od slučaja kada se koristilo ručno popunjavanje vrijednosti smještenih u svojstvu *izabranaOsoba*, pozivom metode *ispuniPolja()*, sada se polja automatski ažuriraju bez da se poziva metoda *azurirajOsobu()*.

Pitanje za diskusiju: Nakon što su navedena polja dvosmjerno povezana, da li će se baciti izuzetak, kojim se korisnik obavještava da ime mora imati između 2 i 50 znakova, kada korisnik ukuca ime koje ima samo 1 znak?

Za razliku od dvosmjernog povezivanja, jednosmjerno povezivanje znači da je jedno svojstvo vezano za drugo, a sve promjene u prvom svojstvu automatski se odražavaju na drugo svojstvo. Međutim, promjene drugog svojstva ne utječu na prvo svojstvo. Ovaj tip povezivanja se radi pozivom metode *bind()*, dok se uklanjanje povezanosti radi pozivom metode *unbind()* (bez parametara), kao u Listingu 15.

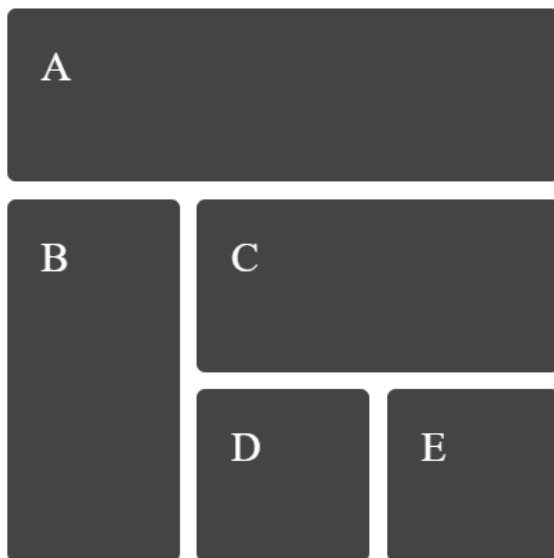
```
izabranaOsoba.addListener((observable, oldValue, newValue) -> {  
    if(oldValue != null) {  
        imeField.textProperty().unbind();  
        prezimeField.textProperty().unbind();  
    }  
    if(newValue != null) {  
        imeField.textProperty().bind(newValue.imeProperty());  
        prezimeField.textProperty().bind(newValue.prezimeProperty());  
    }  
});
```

Listing 15. Event listener za atribut izabranaOsoba sa jednosmjernim povezivanjem

Pitanje za diskusiju: Kakvo ponašanje će proizvesti korištenje jednosmjernog povezivanja? Da li je to željeno ponašanje u ovom slučaju?

8.4.2. Kompleksni raspored elemenata

Pitanje za diskusiju: Popuniti tabele ispod, tako da se iz putem definicije elemenata tabele može kreirati raspored FXML elemenata kao na Slici 10.



Slika 10. Primjer kompleksnijeg rasporeda FXML elemenata

Element	Broj redova	Broj kolona
GridPane	?	?

Element	Row Index	Column Index	Row Span	Column Span
A	?	?	?	?
B	?	?	?	?
C	?	?	?	?
D	?	?	?	?
E	?	?	?	?

8.5. Zadaci za samostalni rad

Za sticanje bodova na prisustvo laboratorijskoj vježbi, potrebno je uraditi sljedeće zadatke tokom laboratorijske vježbe i postaviti ih u repozitorij studenta na odgovarajući način:

Zadatak 1.

Modificirati *toString()* metodu modela, tako da se sada ispisuju podaci o osobi razdvojeni zarezom.

Zadatak 2.

Po uzoru na implementaciju *View* komponente za osobu, kao i implementaciju model i kontroler komponenti JavaFX aplikacije za predmet iz laboratorijske vježbe 7, izvršiti implementaciju *View* komponente JavaFX aplikacije za predmet. Neka je putem *View* komponente moguće dodati novi predmet. Podaci koje je potrebno unijeti se trebaju prikazivati jedni ispod drugih a svaki od njih, osim polja za unos, treba imati i odgovarajuću labelu. Dugme za potvrdu unosa treba biti desno poravnato sa poljima za unos. Ukoliko su podaci ispravno uneseni, obrisati sadržaj polja tako da je moguće unijeti podatke o novoj osobi i korisniku ispisati poruku o uspješno završenom unosu. Ako dođe do greške, ispisati poruku greške. Nakon svakog dodavanja nove osobe, konzolno ispisati (upotrebom *System.out.println()*) podatke o svakoj osobi koja se nalazi u listi osoba.