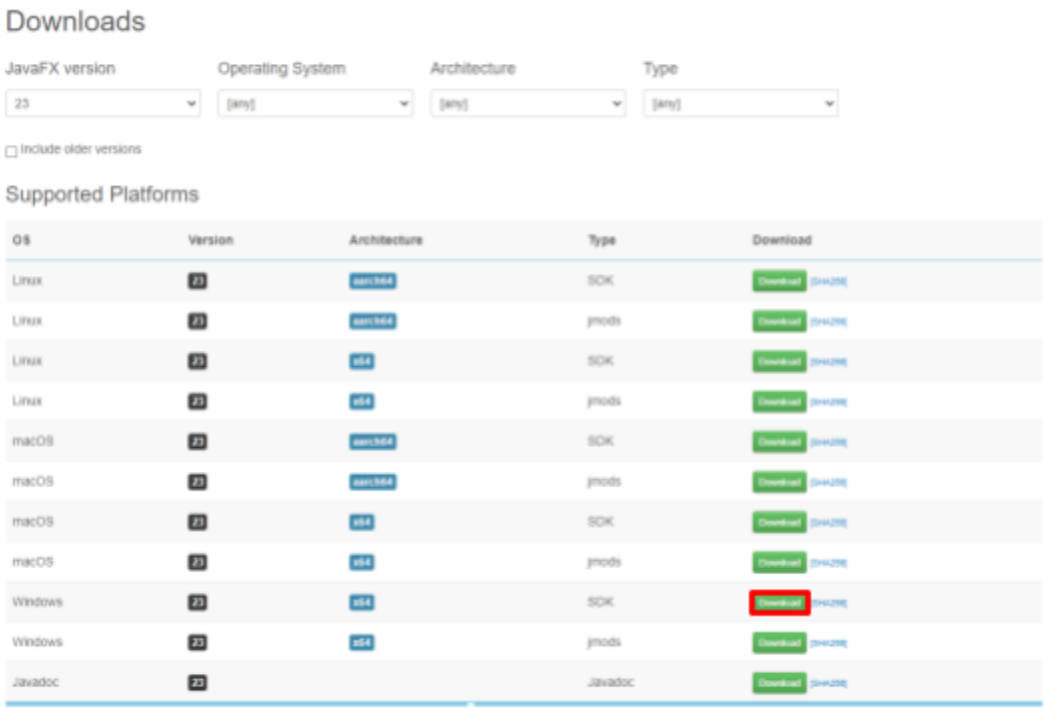
Laboratorijska vježba 7.

Uvod u JavaFX

7.1. Prvi JavaFX projekat

JavaFX predstavlja *framework* za kreiranje obogaćenih internet aplikacija (eng. *Rich Internet Applications* - RIA). Dobro je usklađen sa MVC šablonom dizajna, zbog svoje strukture i principa dizajna, koji potiču razdvajanje odgovornosti (eng. *separation of concerns*). Prije kreiranja prvog JavaFX projekta, potrebno je uraditi preuzimanje JavaFX, jer u novijim verzijama ne dolazi uz Oracle JDK niti OpenJDK. Za preuzimanje posjetiti sljedeći link. Nakon posjete JavaFX web stranice, izabrati *Download* nakon čega će se desiti preusmjeravanje na lokaciju sa koje je moguće uraditi preuzimanje. Pri dnu stranice bi se trebala nalaziti moguća preuzimanja JavaFX (kao na Slici 1, gdje je označeno dugme za preuzimanje za Microsoft Windows Operativni sistem). Izvršiti preuzimanje JavaFX klikom na opciju **Download** pored naziva odgovarajućeg operativnog sistema.



Slika 1. Preuzimanje JavaFX sa oficijelne web stranice

Nakon preuzimanja JavaFX, potrebno je raspakovati preuzetu datoteku na neku lokaciju na lokalnom računaru i zapamtiti lokaciju raspakovanog foldera (koji bi trebao imati naziv "*javafx-sdk-23*". Naredni korak je dodavanje varijable okruženja (eng. *environment variable*), koja pokazuje na **lib** direktorij u raspakovanom folderu. Varijable okruženja se mogu dodavati ručno, ali je lakše to uraditi kroz komandni prozor. Potrebno je tvoriti *command prompt* (skraćeno, *cmd*) program, tj. *terminal* program, na Microsoft Windows i Linux/Mac operativnim sistemima respektivno. U zavisnosti od operativnog sistema, potrebno je unijeti jednu od komandi čiji su osnovni oblici dati u Listingu 1 i 2. Slika 2 prikazuje način postavljanja vrijednosti varijable PATH_TO_FX, ukoliko je folder JavaFX raspakovan na lokaciju "*C:\Program Files (x86)*".

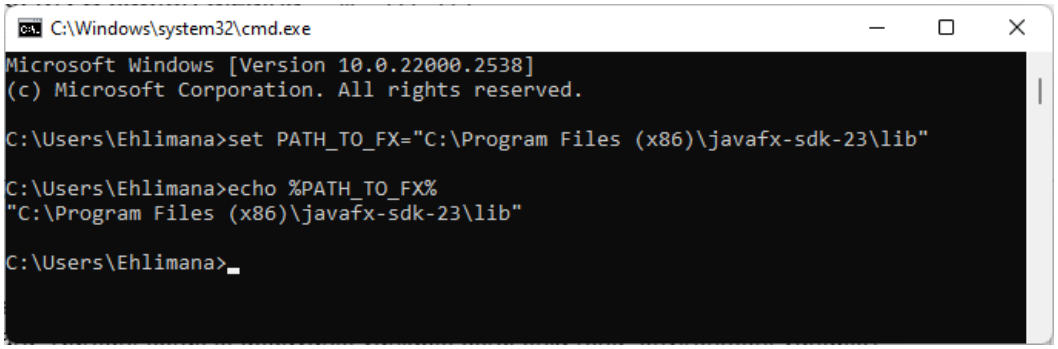
Napomena: Odgovarajuća komanda se ne treba doslovno kopirati i zalijepiti, nego je potrebno modificirati dio "*path/to/*" tako da pokazuje na direktorij u kojem se nalazi prethodno raspakovana datoteka). U slučaju ažuriranja verzije JavaFX u budućnosti, potrebno je promijeniti i naziv foldera "*javafx-sdk-23*" da odgovara preuzetoj verziji.

```
set PATH_TO_FX="path\to\javafx-sdk-23\lib"
```

Listing 1. Konzolna komanda za Microsoft Windows operativni sistem

```
export PATH_TO_FX=path/to/javafx-sdk-23/lib
```

Listing 2. Konzolna komanda za Linux i Mac operativne sisteme

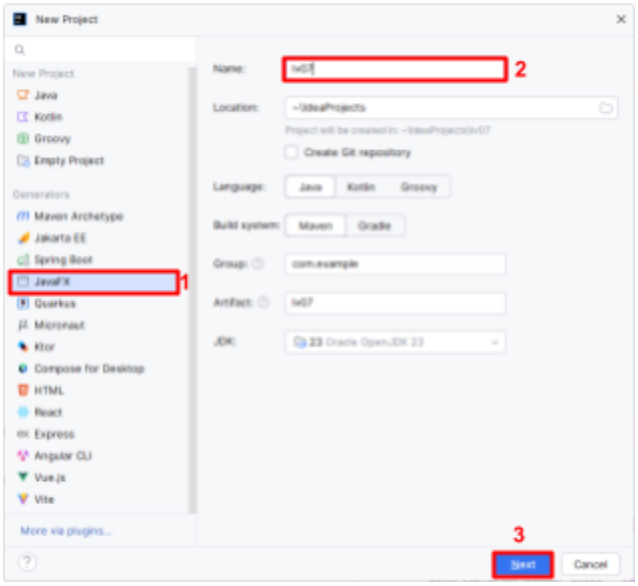


Slika 2. Postavljanje vrijednosti PATH_TO_FX varijable

Nakon postavljanja okruženja za izvršavanje JavaFX aplikacija, prelazi se na kreiranje prvog JavaFX projekta. Za kreiranje prvog JavaFX projekta kroz IntelliJ okruženje, potrebno je pratiti sljedeće korake:

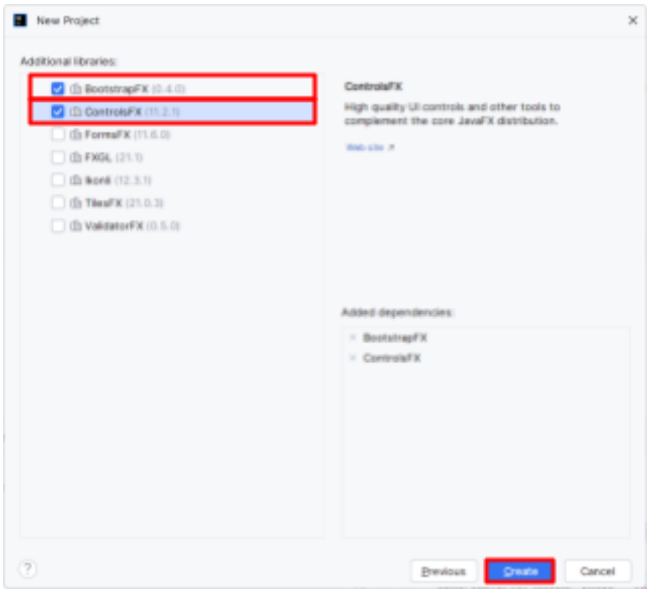
1) Nakon pokretanja IntelliJ okruženja, kreirati novi projekat (bilo izborom **New project** sa tzv. *Welcome* ekrana, bilo izborom **File** → **New** → **Project** u glavnom meniju).
2) Na lijevoj strani ekrana u dijelu *Generators* izabrati *JavaFX* kao opciju (obratiti pažnju da ne treba kreirati uobičajenu Java konzolnu aplikaciju, što je bio slučaj na prethodnim laboratorijskim vježbama), kao što je prikazano na Slici 3 (korak 1).

3) Izabrati naziv projekta (za pokazni projekat neka naziv bude *lv07* kao na Slici 3, korak 2) i lokaciju na kojoj će se projekat kreirati i spasiti, a zatim izabrati jezik i *build* sistem. U polju **Group**, navesti naziv paketa koji će se kreirati zajedno sa projektom (za pokazni projekat ostaviti polazni naziv paketa, tj. *com.example*). Iz **JDK** liste izabrati JDK koji će se koristiti za projekat. Nakon postavljanja osnovnih opcija projekta, kliknuti dugme **Next** (Slika 3, korak 3).



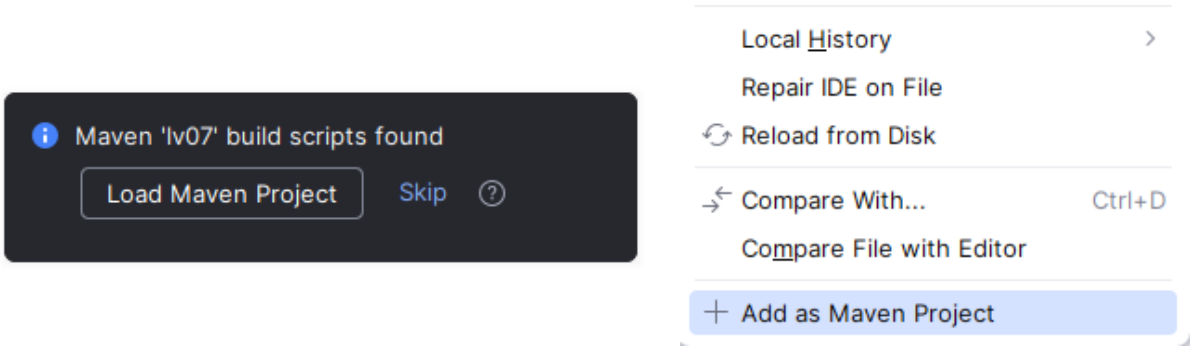
Slika 3. Osnovne opcije pokaznog JavaFX projekta

4) Na sljedećem ekranu, izabrati biblioteke koje se žele koristiti u aplikaciji. Za pokazni projekat izabrati *BootstrapFX* i *ControlsFX* biblioteke, kao što je prikazano na Slici 4. Nakon označavanja navedenih biblioteka, kliknuti na dugme **Create**.



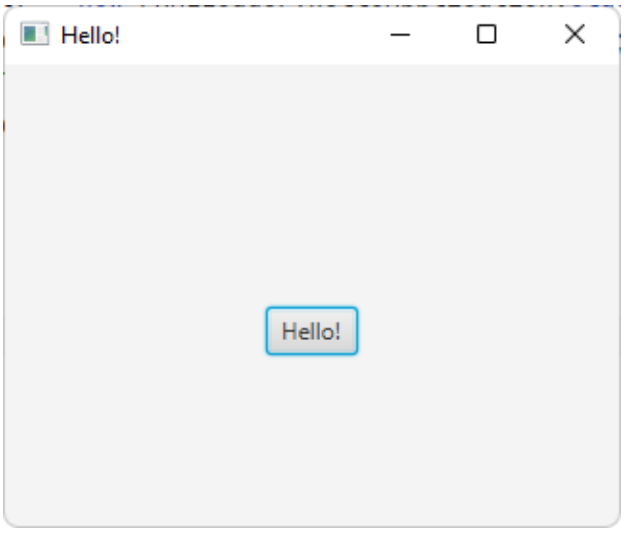
Slika 4. Izbor biblioteka koje će se korisititi u pokaznom JavaFX projektu

5) Ispravno praćenje prethodnih koraka kreiranja prvog JavaFX projekta će rezultirati kreiranjem i prikazom JavaFX *Hello world* aplikacije u okruženju, kao na Slici 5.
Napomena: Može se desiti da se, prilikom kreiranja projekta, u donjem lijevom uglu okruženja pojavi poruka sadržaja "*Maven 'lv07' build scripts found*" u donjem desnom uglu ekrana razvojnog okruženja. U ovom slučaju, potrebno je odabrati opciju "*Load Maven Project*" kako bi se novokreirani projekat uspješno učitao. Ukoliko se odabere opcija "*Skip*", onda je potrebno izvršiti desni klik miša na *pom.xml* datoteku u folder strukturi projekta i izabrati opciju "*Add as Maven Project*".



Slika 5. Odabir opcije za učitavanje JavaFX projekta (lijevo - direktno u okruženju, desno - desnim klikom na pom.xml datoteku)

6) Ako već nije otvorena, otvoriti datoteku *HelloApplication.java* koja se nalazi u folderu *src/main/java/com.example.lvo7*, a nakon toga pokrenuti aplikaciju. Rezultat pokretanja bi trebao biti kao na Slici 6. Prateći sve prethodne korake, kreirana je pokazna JavaFX aplikacija. Klikom na dugme sa natpisom *Hello!*, prikazuje se poruka *Welcome to JavaFX Application!* Čime je validacija rada početne aplikacije završena.



Slika 6. Rezultat pokretanja pokazne JavaFX aplikacije

7.2. JavaFX i MVC

Znanje stečeno u laboratorijskoj vježbi 6, a koje se tiče MVC šablona dizajna, bit će iskorišteno u ovoj vježbi za kreiranje JavaFX aplikacije prateći ovaj šablon dizajna. Jedina razlika je u tome što *View* komponenta još uvijek neće biti implementirana na način svojstven za JavaFX aplikacije, jer će to biti gradivo sljedeće laboratorijske vježbe.

7.2.1. Model u JavaFX

Po uzoru na model iz laboratorijske vježbe 6, i u ovoj vježbi će se koristiti klasa *Osoba*. Potrebno je kreirati novi paket/direktorij u folderu *src/main/java/com.example.lv07* i dati mu naziv **model**. Programski kod klase *Osoba* će se smjestiti u ovaj folder. Glavna razlika prilikom implementacije modela leži u tipu varijabli koje će se koristiti. Do sada su u klasama za definisanje atributa korišteni osnovni tipovi podataka, poput *Integer*, *String*, *Date* i sl. U JavaFX se koriste tzv. *observable properties*. Zbog toga će se umjesto *Integer* koristiti *IntegerProperty*, umjesto *String* koristit će se *StringProperty*, umjesto *Date* koristit će se private *ObjectProperty<Date>*, kao u Listingu 3. *Observable properties* pripadaju *javafx.beans.property* paketu. Glavna razlika između ova dva tipa podataka je njihova namjena. Osnovni tipovi podataka se primarno koriste za pohranu podataka, dok *observable properties* podržavaju uvezivanje (eng. *binding*), upravljanje događajima i sl., što ih čini korisnim u UI *frameworks* gdje je podatke potrebno dinamički ažurirati.

```
private IntegerProperty id;
private StringProperty ime, prezime, adresa;
private ObjectProperty<Date> datumRodjenja;
private StringProperty maticniBroj;
private ObjectProperty<Uloga> uloga;
```

Listing 3. Definisanje Property tipova atributa za model klasu

Na osnovu atributa je sada potrebno generisati *get()* i *set()* metode. Primjer ovih metoda za atribut *prezime* je prikazan u Listingu 4. Potrebno je obratiti pažnju da se za pristup vrijednosti svojstva (eng. *property*) koristi interna funkcija ovog tipa podataka *get()*. To znači da metoda *getPrezime()* nema povratnu vrijednost atributa *prezime*, nego rezultat poziva funkcije *prezime.get()*. Također, generisanje *get()* i *set()* metoda će za svaki atribut kreirati po tri metode (a ne samo dvije, što je bio slučaj u prethodnim laboratorijskim vježbama). Za ispravno povezivanje modela sa ostalim komponentama MVC šablona dizajna u JavaFX aplikacijama, neophodno je imati *get()* metodu koja vraća trenutnu vrijednost svojstva pozivom *get()* funkcije nad tim svojstvom, čime se drugim komponentama omogućava da dobiju vrijednost tog svojstva. Osim ove *get()* metode, potrebno je imati i *get()* metodu koja vraća samo svojstvo, jer je to ključno za uvezivanje i dozvoljava tzv. *event listeners* da budu povezani sa tim svojstvom, čime se omogućava korisničkom interfejsu automatsko ažuriranje kada dođe do promjene vrijednosti.

```
public String getPrezime() {
    return prezime.get();
}

public StringProperty prezimeProperty() {
    return prezime;
}

public void setPrezime(String prezime) {
    this.prezime.set(prezime);
}
```

Listing 4. get() i set() metode za atribut Property tipa

Konstruktor klase *Osoba* treba definisati tako da inicijalizira polja i radi validaciju onih polja za koja je to potrebno. Po uzoru na laboratorijsku vježbu 6, validirat će se samo polja *ime* i *maticniBroj*, koristeći funkcije *setIme()* i *setMaticniBroj()* koje sadrže validacijsku logiku na način koji je prikazan u Listingu 5.

```
public void setIme(String ime) {
    if (ime == null || ime.length() < 2 || ime.length() > 50) {
        throw new IllegalArgumentException("Ime mora imati izmedju 2
i 50 znakova.");
    }
    this.ime.set(ime);
}

public void setMaticniBroj(String maticniBroj) {
    if (maticniBroj == null || maticniBroj.trim().isEmpty() ||
maticniBroj.length() != 13) {
        throw new IllegalArgumentException("Maticni broj mora imati
tacno 13 karaktera");
    }
    else if(!ProvjeriMaticniBroj(maticniBroj)){
        throw new IllegalArgumentException("Maticni broj se ne
poklapa sa datumom rodjenja!");
    }
    this.maticniBroj.set(maticniBroj);
}
```

Listing 5. Validacijske funkcije za atribute klase Osoba

Nakon generisanja svih *get()* i *set()* metoda sa odgovarajućom validacijskom logikom, može se preći na kreiranje konstruktora koji koristi te metode pri inicijalizaciji objekta klase *Osoba*. Neophodno je obratiti pažnju da se u konstruktoru polja *property* tipa sada incijaliziraju na drugačiji način od atributa osnovnih tipova podataka. Naprimjer, umjesto dodjele *this.id = id*, koristit će se *this.id = new SimpleIntegerProperty(id)*. Kompletan programski kod konstruktora modela prikazan je u Listingu 6.

```
public Osoba(Integer id, String ime, String prezime, String adresa,
Date datumRodjenja, String maticniBroj, Uloga uloga) {
   // inicijalizacija polja
   this.id = new SimpleIntegerProperty(id);
   this.ime = new SimpleStringProperty();
   this.prezime = new SimpleStringProperty(prezime);
   this.adresa = new SimpleStringProperty(adresa);
   this.datumRodjenja = new SimpleObjectProperty<>(datumRodjenja);
   this.maticniBroj = new SimpleStringProperty();
   this.uloga = new SimpleObjectProperty<>(uloga);

   // validacija polja
   setIme(ime);
   setMaticniBroj(maticniBroj);
}
```

Listing 6. Konstruktor klase Osoba koji koristi property objekte

Ovime je završena implementacija klase *Osoba*. U nastavku će se raditi implementacija klase *OsobaModel*. Važno je napomenuti da se sloj koji je zadužen za implementaciju poslovnih pravila i operacija najčešće naziva *Service*, ali će u laboratorijskim vježbama biti korišten naziv *Model* s obzirom da se neće koristiti neke od naprednijih funkcionalnosti ovog tipa objekata. Suštinski, klasa *Osoba* je model podataka, a klasa *OsobaModel* koja će biti implementirana u nastavku omogućava usluživanje za model klasu. U klasi *OsobaModel* će u početnoj implementaciji postojati samo jedan atribut koji sadrži listu osoba, čime ova klasa zapravo postaje kontejnerska klasa za model *Osoba*. Neophodno je obratiti pažnju da se zbog rada sa JavaFX ne može koristiti obična lista, nego to mora biti *ObservableList*, kao što je prikazano u Listingu 7, uz dodatni prikaz inicijalizacije ovog tipa kolekcije u okviru konstruktora.

```
public class OsobaModel
{
   private ObservableList<Osoba> osobe;

   public OsobaModel() {
      osobe = FXCollections.observableArrayList();
   }
}
```

Listing 7. Osnovni dijelovi klase OsobaModel

U klasi *OsobaModel* će biti implementirane tzv. CRUD operacije (***C****reate,* ***R****ead,* ***U****pdate,* ***D****elete*). Prva od njih je operacija dodavanja nove osobe u kolekciju. Ova operacija će biti implementirana unutar metode *dodajOsobu()*. Parametri ove metode su svi atributi klase *Osoba*, jer su oni neophodni za inicijalizaciju novog objekta klase *Osoba*. Povratna vrijednost ove metode je tipa *String*, jer je potrebno vratiti poruku korisniku o uspješnosti dodavanja nove osobe. Ukoliko se osoba uspješno kreira, onda se ona smješta u listu *osobe* i povratna vrijednost glasi "*Osoba je uspjesno dodana!*". U suprotnom se vraća poruka greške (Listing 8).

```
public String dodajOsobu(Integer id, String ime, String prezime,
String adresa, Date datumRodjenja, String maticniBroj, Uloga uloga)
{
    try {
        Osoba newOsoba = new Osoba(id, ime, prezime, adresa,
datumRodjenja, maticniBroj, uloga);
        osobe.add(newOsoba);
        return "Osoba je uspjesno dodana!";
    } catch (IllegalArgumentException e) {
        return e.getMessage();
    }
}
```

Listing 8. Metoda za dodavanje nove osobe u kolekciju

Za pronalazak i učitavanje osoba iz kolekcije, bit će implementirane dvije metode: *dajSveOsobe()* i *dajOsobuPoId()* i koje su prikazane u Listingu 9. Prva metoda nema parametre i samo vraća cijelu listu sa svim osobama. Za razliku od nje, druga metoda ima jedan parametar, koji predstavlja id osobe koja se traži. Ukoliko u listi *osobe* postoji osoba sa datim id-em, onda će ta osoba biti povratna vrijednost ove metode. U suprotnom, povratna vrijednost će biti *null*.

```
public ObservableList<Osoba> dajSveOsobe() {
    return osobe;
}

public Osoba dajOsobuPoId(Integer id) {
    for (Osoba osoba : osobe) {
        if (osoba.getId().equals(id)) {
            return osoba;
        }
    }
    return null;
}
```

Listing 9. Metode za pronalazak svih osoba ili jedne određene osobe

Metoda za ažuriranje postojećih osoba imat će naziv *azurirajOsobu()*. Ova metoda ima isti broj parametara koliko je atributa u klasi *Osoba*, pri čemu prvi parametar predstavlja id tražene osobe i to je parametar po kojem se vrši pretraga. Dakle, ova metoda ažurira osobu na način da je u listi *osobe* pronađe po id-u i ažurira njene atribute na osnovu vrijednosti parametara proslijeđenih metodi. Povratna vrijednost ove metode je *String* i odnosi se na poruku koja se vraća korisniku. Ukoliko osoba nije pronađena, korisniku se vraća poruka *"Osoba nije pronadjena!"*. Poruka *"Osoba je uspjesno azurirana!"* se vraća korisniku u situaciji kada je osoba pronađena i polja se zatim uspješno ažuriraju, dok se u slučaju greške vraća poruka greške. Također, pojedini parametri mogu imati vrijednost *null*, što označava da dato polje nije potrebno ažurirati. Kompletan programski kod metode *azurirajOsobu()* je prikazan u Listingu 10.

```
public String azurirajOsobu(Integer id, String novoIme, String
novoPrezime, String novaAdresa, Date noviDatumRodjenja, String
noviMaticniBroj, Uloga novaUloga)
{
   Osoba trazenaOsoba = dajOsobuPoId(id);
   if(trazenaOsoba != null) {
            try {
                if (novoIme != null) {
                    trazenaOsoba.setIme(novoIme);
                }
                if (novoPrezime != null) {
                    trazenaOsoba.setPrezime(novoPrezime);
                }
                if (novaAdresa != null) {
                    trazenaOsoba.setAdresa(novaAdresa);
                }
                if (noviDatumRodjenja != null) {

trazenaOsoba.setDatumRodjenja(noviDatumRodjenja);
                }
                if (noviMaticniBroj != null) {
                    trazenaOsoba.setMaticniBroj(noviMaticniBroj);
                }
                if (novaUloga != null){
                    trazenaOsoba.setUloga(novaUloga);
                }
                return "Osoba je uspjesno azurirana!";
            }
            catch (IllegalArgumentException e) {
                return e.getMessage();
            }
   }
   return "Osoba nije pronadjena!";
}
```

Listing 10. Metoda za ažuriranje osoba

Za brisanje osoba koristit će se metoda *obrisiOsobu()*. Metoda ima jedan parametar koji predstavlja id osobe koju je potrebno ukloniti iz liste *osobe*. Osoba će biti uklonjena jedino u slučaju da postoji u listi, u koju svrhu se koristi metoda *removeIf()* sa lambda izrazom. Trenutno metoda nema povratnu vrijednost. Kompletan programski kod metode *obrisiOsobu()* je prikazan u Listingu 11.

```
public void obrisiOsobu(Integer id)
{
   osobe.removeIf(osoba -> osoba.getId() == id);
}
```

Listing 11. Metoda za brisanje osoba

Kako kolekcija osoba ne bi bila prazna na početku rada aplikacije, potrebno je implementirati metodu *napuni()*, koja listu popunjava nekim početnim podacima (Listing 12). Ovi podaci se zatim tokom rada aplikacije mogu iskoristiti kako bi se demonstriralo korištenje kreiranih metoda.

```
public void napuni(){
   osobe.add(new Osoba(1,"Neko","Nekic","Neka adresa", new
Date(97,8,25), "2509997123456", Uloga.STUDENT));
   osobe.add(new Osoba(2,"Neko 2","Nekic 2","Neka adresa 2",new
Date(97,8,25), "2509997123456", Uloga.NASTAVNO_OSOBLJE));
}
```

Listing 12. Metoda za inicijalizaciju kolekcije osoba

Za pohranu podataka se mogu koristiti tekstualne i XML datoteke (kao na laboratorijskoj vježbi 6). Metode *napuniPodatkeIzTxtDatoteke()* i *napuniPodatkeIzXmlDatoteke()* prikazane u Listingu 13 i 14 su gotovo identične metodama *ucitajOsobeIzTxtDatoteke()* i *ucitajOsobeIzXmlDatoteke()*, respektivno, iz prethodne laboratorijske vježbe. Naravno, postoje sitne razlike, ali su one zanemarive i ne zahtijevaju dodatnu pažnju. Ovime je završena implementacija modela pokazne JavaFX aplikacije.

```
public void napuniPodatkeIzTxtDatoteke(String putanjaDoDatoteke)
throws IOException, ParseException {
   osobe = FXCollections.observableArrayList();
   BufferedReader reader = new BufferedReader(new
FileReader(putanjaDoDatoteke));

   String linija;
   while ((linija = reader.readLine()) != null) {
       String[] polja = linija.split(",");
       if (polja.length == 7) {
           Integer id = Integer.parseInt(polja[0]);
           String ime = polja[1];
           String prezime = polja[2];
           String adresa = polja[3];
           Date datumRodjenja = dateFormat.parse(polja[4]);
           String maticniBroj = polja[5];
           Uloga uloga = Uloga.valueOf(polja[6].toUpperCase());

           Osoba osoba = new Osoba(id, ime, prezime, adresa,
datumRodjenja, maticniBroj,uloga);
           osobe.add(osoba);
       }
   }
   reader.close();
}
```

Listing 13. Metoda za učitavanje tekstualne datoteke u kolekciju

```
public void napuniPodatkeIzXmlDatoteke(String putanjaDoDatoteke)
throws Exception
{
   osobe = FXCollections.observableArrayList();
   File xmlFile = new File(putanjaDoDatoteke);

   DocumentBuilder builder =
DocumentBuilderFactory.newInstance().newDocumentBuilder();
   Document doc = builder.parse(xmlFile);

   doc.getDocumentElement().normalize();

   NodeList listaCvorova = doc.getElementsByTagName("osoba");

   for (int i = 0; i < listaCvorova.getLength(); i++)
   {
       Node cvor = listaCvorova.item(i);

       if (cvor.getNodeType() == Node.ELEMENT_NODE)
       {
           Element element = (Element) cvor;

           Integer id =
Integer.parseInt(element.getElementsByTagName("id").item(0).getText
Content());
           String ime =
element.getElementsByTagName("ime").item(0).getTextContent();
           String prezime =
element.getElementsByTagName("prezime").item(0).getTextContent();
           String adresa =
element.getElementsByTagName("adresa").item(0).getTextContent();
           Date datumRodjenja =
dateFormat.parse(element.getElementsByTagName("datumRodjenja").item
(0).getTextContent());
           String maticniBroj =
element.getElementsByTagName("maticniBroj").item(0).getTextContent(
);
           Uloga uloga =
Uloga.valueOf(element.getElementsByTagName("uloga").item(0).getText
Content().toUpperCase());

           Osoba osoba = new Osoba(id, ime, prezime, adresa,
datumRodjenja, maticniBroj, uloga);
           osobe.add(osoba);
       }
   }
}
```

Listing 14. Metoda za učitavanje XML datoteke u kolekciju

7.2.2. Controller u JavaFX

Sada je potrebno kreirati novi paket/direktorij naziva *controller* i u njega smjestiti *OsobaController* klasu. Ova klasa će implementirati logiku kontroler komponente MVC šablona dizajna. Kontroler se neće previše razlikovati od onoga implementiranog u laboratorijskoj vježbi 6. Dakle, atributi ove klase su *model* i *view*, a konstruktor inicijalizira instancu klase *OsobaController* na osnovu dva parametra, koji predstavljaju *Model* i *View* komponente MVC šablona dizajna. Osnovni dijelovi strukture klase *OsobaController* prikazani su u Listingu 15.

```
public class OsobaController
{
   private OsobaModel model;
   private OsobaView view;

   public OsobaController(OsobaModel model, OsobaView view)
   {
      this.model = model;
      this.view = view;
   }
}
```

Listing 15. Osnovni dijelovi klase OsobaController

Za svaku CRUD operaciju implementiranu u modelu, kontroler bi trebao imati odgovarajuću metodu koja poziva model. Zbog toga će se za metodu koja vraća korisnika po id iz modela biti implementirana i odgovarajuća metoda kontrolera *dajOsobuPoId()*. Metodu kontrolera *azurirajIme()*, implementiranu u laboratorijskoj vježbi 6, sada je moguće definisati tako da poziva metodu *azurirajOsobu()* modela koji je atribut kontrolerske klase. U suštini, jedina razlika prilikom implementacije je pozivanje metode *azurirajOsobu()* iz klase *OsobaModel*, umjesto direktnog pozivanja *set()* metode za atribut *ime* iz klase *Osoba*. Izgled metoda *dajOsobuPoId()* i *azurirajIme()* prikazan je u Listingu 16. Metode *dajOsobeIzTxtDatoteke()* i *dajOsobeIzXmlDatoteke()* je također potrebno implementirati tako da imaju gotovo identičan izgled kao na laboratorijskoj vježbi 6, s tim što se sada pozivaju prethodno definisane metode *napuniPodatkeIzTxtDatoteke()* i *napuniPodatkeIzXmlDatoteke()*.

```
public Osoba dajOsobuPoId(Integer id){
   return model.dajOsobuPoId(id);
}

public void azurirajIme(Integer id){
   try {
       model.azurirajOsobu(id, view.getUlazniTekst(), null, null,
null, null, null);
       view.setPoruka("Ime je uspjesno azurirano!");
   }
   catch(Exception e){
```

```
        view.setPoruka("Greska: " + e.getMessage());
    }
}
```

Listing 16. Metode kontrolera za CRUD operacije s osobama

7.3. Pokretanje JavaFX aplikacije

Za pokretanje JavaFX aplikacije koristi se *HelloApplication* klasa, koja se automatski kreira prilikom pravljenja projekta. Ova klasa podrazumijeva da je *View* komponenta implementirana kroz FXML datoteku, što je gradivo sljedeće laboratorijske vježbe. Zbog toga, u metodi *main()* zasad je potrebno zakomentarisati liniju koda u kojoj se poziva funkcija ***launch()***. Na ovaj način se neće pokretati grafički korisnički interfejs, već će se ispis prikazivati isključivo u konzoli. U metodi *main()* potrebno je instancirati model, izvršiti popunjavanje podataka o osobama pozivom metode *napuni()*, instancirati *View* komponentu putem koje će se ažurirati ime osobe sa id-em 1, instancirati kontroler na osnovu *Model* i *View* komponenti i provjeriti da li sve ispravno radi. Programski kod metode *main()* je prikazan u Listingu 17. Ispis u konzoli bi trebao biti kao na Slici 7, odakle je vidljivo da ispravno rade CRUD operacije, a da rad s datotekama nije moguć zbog nedostajućih datoteka. Ovime je završena implementacija pokazne JavaFX aplikacije, bez *View* komponente sa grafičkim korisničkim interfejsom.

```
OsobaModel osobaModel = new OsobaModel();
osobaModel.napuni();

OsobaView osobaView = new OsobaView();
osobaView.setUlazniTekst("Novo ime");

OsobaController osobaController = new OsobaController(osobaModel,
osobaView);
osobaController.azurirajIme(1);

System.out.println("1) View ispisuje: " + osobaView.getPoruka());
System.out.println("   Azurirana osoba je: " +
osobaController.dajOsobuPoId(1).toString());

osobaController.dajOsobeIzTxtDatoteke("src/data/osobe.txt");
System.out.println("2) View ispisuje: " + osobaView.getPoruka());

osobaController.dajOsobeIzXmlDatoteke("src/data/osobe.xml");
System.out.println("3) View ispisuje: " + osobaView.getPoruka());
```

Listing 17. Poziv svih MVC komponenti u main() metodi

```
1) View ispisuje: Ime je uspjesno azurirano!
   Azurirana osoba je: Osoba{id='1', ime='Novo ime', prezime='Nekic', adresa='Neka adresa', datumRodjenja=Thu Sep 25 00:00:00 CEST 1997, maticniBroj='2509997123456', uloga
2) View ispisuje: Greska: src\data\osobe.txt (The system cannot find the path specified)
3) View ispisuje: Greska: C:\Users\Korisnik\IdeaProjects\lv07\src\data\osobe.xml (The system cannot find the path specified)
```

Slika 7. Izlaz u konzoli nakon pokretanja JavaFX MVC aplikacije

7.4. 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.

Napraviti dorade u sljedećim metodama:

a) *obrisiOsobu()* iz *OsobaModel* klase tako da joj je povratni tip *String*, koji glasi *"Osoba nije pronadjena!"*, ukoliko osoba ne postoji a *“Osoba je uspjesno obrisana!”* ukoliko osoba postoji.
b) *dajOsobuPoId()* iz *OsobaController* klase tako da atribut *poruka* (koji je dio *View* komponente) postavlja na vrijednost *“Osoba nije pronadjena!”*, ukoliko je vraćena vrijednost *null*, odnosno na vrijednost vraćene osobe, ukoliko je osoba pronađena.

Zadatak 2.

Napisati testove za klasu *OsobaModel*, čija je implementacija prikazana u laboratorijskoj vježbi. Napisati testove koji provjeravaju da:

- Metoda *napuni()* ispravno popunjava listu *osobe;*
- Metoda *azurirajOsobu()* ispravno ažurira podatke o osobi (napisati barem tri testa, pri čemu postoje slučajevi kada id postoji, id ne postoji i kada se ažuriraju samo neka a ne sva polja).

Napomena: Za generisanje klase u kojoj će se nalaziti testovi moguće je koristiti isti princip kao i na svim laboratorijskim vježbama do sada. Međutim, nakon generisanja testne klase, potrebno je otići u datoteku *module-info.java* i dodati liniju *exports com.example.lv07.model;* čime se deklariše da je paket *com.example.lv07.model* dostupan ostalim modulima na korištenje. Bez ove linije koda, testovi ne bi mogli pristupiti modelu.

Ukoliko tokom laboratorijske vježbe ne budu uspješno izvršeni zadaci 1 i 2, moguće je steći bodove na prisustvo ukoliko se do sljedeće laboratorijske vježbe uradi i sljedeći zadatak:

Zadatak 3.

Po uzoru na implementirane komponente MVC šablona dizajna u JavaFX aplikaciji za osobu, kao i na implementaciji MVC šablona dizajna za predmet iz laboratorijske vježbe 6, izvršiti implementaciju JavaFX aplikacije za klasu *Predmet*. Sve ono što je vrijedilo u zadatku sa laboratorijske vježbe 6 vrijedi i sada.