Laboratorijska vježba 6.

MVC šablon dizajna

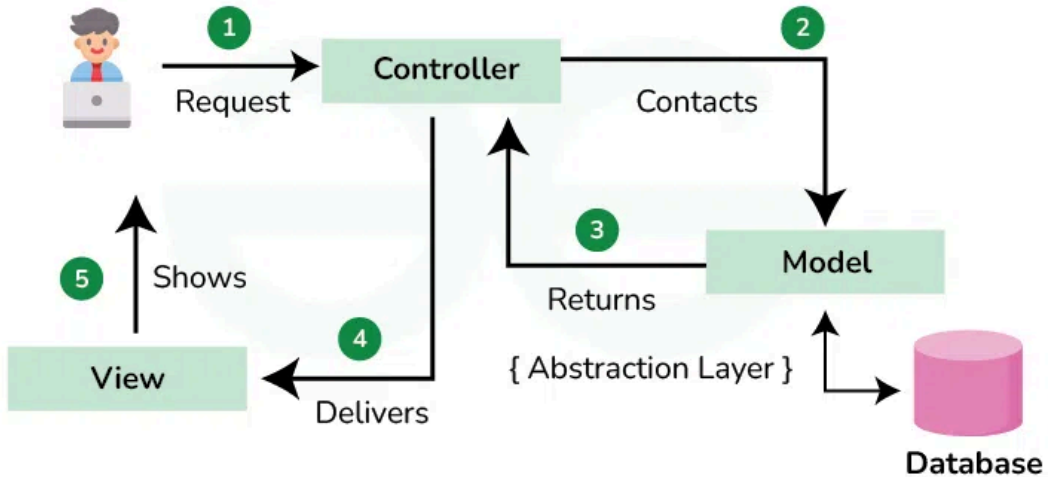
6.1. MVC šablon dizajna

Šabloni dizajna (eng. *design patterns*) predstavljaju najbolje prakse korištene od strane programera, koji se bave razvojem objektno-orijentisanog softvera. To su provjerena rješenja uobičajenih problema sa kojima se programeri suočavaju tokom razvoja softvera. Ova rješenja su dobivena principom pokušaj-pogreška od strane brojnih programera tokom prilično dugog vremenskog razdoblja. Na laboratorijskim vježbama detaljno će se obraditi ***Model View Controller*** (**MVC**) šablon dizajna, koji omogućava kreiranje aplikacija korištenjem jasne arhitekture, testiranje i održavanje aplikacija čini jednostavnijim i potiče mogućnost ponovne upotrebe komponenti (između ostalog). MVC specificira da se aplikacija sastoji od modela podataka, prezentacijskih informacija i kontrolnih informacija. Ovaj šablon dizajna zahtijeva da svaki od njih bude razdvojen u različite objekte. MVC šablon dizajna se sastoji od tri dijela (Slika 1):

1. **M** (*Model*) - predstavlja podatke i poslovnu logiku aplikacije. Ova komponenta je odgovorna za upravljanje podacima aplikacije, obradu poslovnih pravila i odgovara na zahtjeve za informacijama od drugih komponenti, kao što su *View* i *Controller*.

2. **V** (*View*) - prikazuje podatke iz modela korisniku i šalje korisničke unose kontroleru. Riječ je o “pasivnoj” komponenti, koja nema direktnu interakciju sa modelom. Umjesto toga, prima podatke od modela i šalje korisničke unose kontroleru na obradu.

3. **C** (*Controller*) - ima ulogu posrednika između *Model* i *View* komponenti. Obrađuje korisnički unos i, u skladu sa tim, ažurira *Model* komponentu. Također, ažurira *View* komponentu kako bi se odrazile promjene u modelu. Sadrži logiku aplikacije, kao što je provjera validnosti korisničkog unosa i transformacije podataka.



Slika 1. MVC komponente

6.2. Model u MVC

Prethodno je spomenuto da model predstavlja podatke i poslovnu logiku aplikacije. Dakle, tu se nalaze osnovna pravila, operacije i funkcije koje upravljaju načinom na koji se manipulira podacima, te kako se oni obrađuju i validiraju. Kada se kaže "poslovna logika" ili "biznis logika", misli se na određena poslovna pravila i validacije, koje je potrebno uraditi u *Model* komponenti. Naprimjer, u sistemu e-kupovine, biznis logika može podrazumijevati izračunavanje ukupne cijene narudžbe, proračun cijene sa popustom, provjeru dostupnosti artikala u skladištu i sl. U bankarskom sistemu, biznis logika se može odnositi na pravila kojima se određuje da li određena transakcija može biti uspješno izvršena na osnovu trenutnog stanja računa, postavljanje limita na transakcije i sl. Bez obzira na to koja su biznis pravila, *Model* je odgovoran za njih.

U ovoj laboratorijskoj vježbi će biti kreiran model po uzoru na klase koje su implementirane u sklopu laboratorijske vježbe 3, tj. koristiti će se programski kod klase *Osoba*. Ova klasa će biti modificirana kako bi bolje odgovarala modelu iz MVC šablona dizajna i na taj način bila spremna za sljedeće laboratorijske vježbe. Radi jednostavnosti, klase naslijeđene iz klase *Osoba* neće biti implementirane. Kao što je već spomenuto, različiti dijelovi MVC šablona dizajna predstavljaju različite komponente i zbog toga će svaki dio MVC šablona dizajna biti implementiran u odvojenom paketu/direktoriju. Desnim klikom na *src* folder potrebno je kreirati novi paket i dati mu naziv *model*. U navedenom paketu kreirati klasu *Osoba*. Za razliku od implementacije ove klase u laboratorijskoj vježbi 3, u klasu će biti dodani atributi *id*, *maticniBroj* i *uloga*, a koji su prikazani u Listingu 1. Obzirom da neće biti korišteno nasljeđivanje, različite uloge osoba će biti određene na osnovu enumeracijskog tipa *Uloga* (koji može imati vrijednost *STUDENT* ili *NASTAVNO_OSOBLJE*, na način prikazan u Listingu 2), a koji je potrebno kreirati u istom paketu u kome se nalazi i klasa *Osoba* (dakle, u paketu *model*).

```
private Integer id;
private String ime, prezime, adresa;
private Date datumRodjenja;
private String maticniBroj;
private Uloga uloga;
```

Listing 1. Atributi klase Osoba

```
public enum Uloga {
   STUDENT, NASTAVNO_OSOBLJE
}
```

Listing 2. Enumeracijski tip Uloga

U narednom koraku, potrebno je generisati konstruktor, *get()* i *set()* metode za sve atribute. Obzirom da model implementira validacijsku logiku, generisane metode treba modificirati tako da vrše određene provjere validnosti podataka. Ovo se najviše odnosi na *set()* metode, jer su one te koje rade promjene nad podacima. Validaciju je poželjno raditi nad svim atributima klase, ali će se radi pojednostavljenja raditi validacija samo nekih atributa na način opisan u nastavku:

- Atribut *ime* mora imati između 2 i 50 znakova. Za validiranje atributa *ime*, potrebno je modicirati metodu *setIme()*, koja sada treba imati izgled kao u Listingu 3, pri čemu se vrši provjera da li je parametar inicijaliziran, da li je string previše kratak ili previše dug.

```
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 = ime;
}
```

Listing 3. Validacija ispravnosti imena

- Atribut *maticniBroj* mora imati tačno 13 karaktera i mora se poklapati sa datumom rođenja. U laboratorijskoj vježbi 3 implementirana je metoda *ProvjeriMaticniBroj()* koja vrši validaciju matičnog broja u odnosu na datum rođenja osobe kojoj matični broj pripada. Sada je potrebno promijeniti metodu *setMaticniBroj()* tako da poziva prethodno spomenutu pomoćnu metodu, na način prikazan u Listingu 4, pri čemu se dodatno vrši provjera da li je parametar inicijaliziran i sastoji li se od tačno 13 karaktera.

```
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 = maticniBroj;
}
```

Listing 4. Validacija matičnog broja

Ovime su kompletirane validacije modela. Da bi klasa ispravno funkcionisala, potrebno je još promijeniti konstruktor tako da poziva *set()* metode, jer je u njima implementirana validacijska logika. Izgled konstruktora klase *Osoba* je prikazan u Listingu 5.

```
public Osoba(Integer id, String ime, String prezime, String adresa,
Date datumRodjenja, String maticniBroj, Uloga uloga)
{
    setId(id);
    setIme(ime);
    setPrezime(prezime);
    setAdresa(adresa);
    setDatumRodjenja(datumRodjenja);
    setMaticniBroj(maticniBroj);
    setUloga(uloga);
}
```

Listing 5. Konstruktor klase Osoba

U model će biti dodana i određena biznis logika, koja prati pravila tipična za fakultete/univerzitete. Naprimjer, neka je jedno od poslovnih pravila da nastavno osoblje uvijek može učestvovati u projektima, dok studenti mogu učestvovati samo pod uslovom da nisu voditelji projekta. Implementacija ovog poslovnog pravila je prikazana u Listingu 6. Drugo poslovno pravilo određuje da samo studenti mogu primati stipendiju, dok nastavno osoblje nema pravo na stipendiju. Implementacija ovog poslovnog pravila je prikazana u Listingu 7.

```
public boolean mozeUcestvovatiUProjektu(boolean voditeljProjekta)
{
    if(this.uloga == Uloga.NASTAVNO_OSOBLJE || (!voditeljProjekta &&
this.uloga == Uloga.STUDENT))
    {
        return true;
    }
    return false;
}
```

Listing 6. Poslovno pravilo koje određuje mogućnost učešća u projektima

```
public boolean imaPravoNaStipendiju(){
    return this.uloga == Uloga.STUDENT;
}
```

Listing 7. Poslovno pravilo koje određuje pravo na stipendiju

6.3. View u MVC

View komponenta MVC šablona dizajna je odgovorna za prezentacijski sloj aplikacije. Prikazuje podatke koje model daje korisniku i šalje korisničke naredbe *Controller* komponenti. *View* osluškuje promjene u modelu i ažurira prikaz u skladu sa time. Nema izravnu interakciju sa modelom, već sa njim komunicira putem kontrolera. Ovo je najčešće grafički korisnički interfejs (eng. *graphical user interface* - GUI), koji korisniku omogućava interakciju sa aplikacijom. Za potrebe ove laboratorijske vježbe, *View* komponenta će saržavati dva *String* atributa, koji korisniku omogućavaju da unese podatke i da dobije povratne informacije u vidu konzolnog ispisa. Ovi atributi će se zvati *ulazniTekst* i *poruka*. Najprije je potrebno kreirati paket *view* (na isti način kao paket *model* u prethodnom dijelu) i u njega smjestiti klasu *OsobaView*, koja predstavlja *View* komponentu i koja ima atribute kao u Listingu 8.

```
private String ulazniTekst;
private String poruka;
```

Listing 8. Atributi View klase

U narednom koraku potrebno je kreirati *get()* i *set()* metode za navedene atribute. Dakle, korisnik interakcijom sa *View* komponentom (u ovom primjeru interakcija podrazumijeva postavljanje vrijednosti atributa *ulazniTekst*) šalje ulazne podatke koji se obrađuju od strane kontrolera (koji će detaljno biti opisan u sljedećem poglavlju). Kada kontroler obradi podatke, vraća informaciju *View* komponenti, koja obavještava korisnika o rezultatu izvršavanja određene akcije (u ovom primjeru postavljanjem vrijednosti atributa *poruka* korisniku se prikazuje rezultat izvršavanja akcije).

6.4. Controller u MVC

Controller komponenta MVC šablona dizajna djeluje kao posrednik između *Model* i *View* komponenti. Osluškuje korisničke unose (najčešće putem informacija koje dobije od *View* komponente) i izvršava radnje na osnovu tih unosa. To znači da kontroler obrađuje unos, manipulira modelom i ažurira prikaz, odnosno ustvari "prevodi" korisničke radnje u operacije u modelu. Na primjer, kada korisnik (putem *View* komponente) ispuni formu za kreiranje novog korisničkog računa i pošalje je (klikom na dugme za potvrdu unosa), kontroler potvrđuje unos i poziva model da spremi nove korisničke podatke, nakon čega ažurira *View* komponentu kako bi ona odražavala promjene. Dakle, *Controller* je posrednik između *Model* i *View* komponenti MVC šablona dizajna. Kao takav, sadrži instance klasa koje predstavljaju ove dvije komponente. Kako bi se ostvarile ove funkcionalnosti, najprije je potrebno kreirati novi paket *controller* (na isti način kao prethodna dva paketa) i unutar njega smjestiti klasu *OsobaController*. Atributi ove klase su prikazani u Listingu 9.

```
private Osoba model;
private OsobaView view;
```

Listing 9. Atributi OsobaController klase

U narednom koraku, potrebno je kreirati konstruktor, *get()* i *set()* metode za navedene atribute. Glavna uloga kontrolera je komunikacija sa *View* i *Model* komponentama. Naprimjer, neka je korisniku omogućeno da ažurira ime. Metoda koja radi ažuriranje imena korisnika se nalazi u kontroleru i ima naziv *azurirajIme()*. Ideja ove metode je da "pokupi" korisničko ime, koje korisnik unese unutar *View* komponente postavljajući vrijednosti atributa *ulazniTekst* na odgovarajuću vrijednost. Nakon toga, kontroler poziva odgovarajuću metodu modela (u ovom slučaju riječ je o metodi *setIme()*) i prosljeđuje joj kao parametar vrijednost koju je korisnik unio (to je vrijednost atributa *ulazniTekst*). U zavisnosti od toga da li je operacija uspješno završena ili ne, kontroler ažurira *View* komponentu, postavljajući vrijednost atributa *poruka* na osnovu rezultata izvršene operacije. Kod metode *azurirajIme()* koji omogućava prethodno opisane akcije prikazan je u Listingu 10.

```
public void azurirajIme()
{
    try {
        model.setIme(view.getUlazniTekst());
        view.setPoruka("Ime je uspjesno azurirano!");
    }
    catch(Exception e) {
        view.setPoruka("Greska: " + e.getMessage());
    }
}
```

Listing 10. Metoda kontrolera za ažuriranje imena

6.5. Komunikacija između komponenti u MVC

U nastavku je prikazan tok komunikacije između komponenti u MVC šablonu dizajna, koji osigurava da je svaka komponenta odgovorna za određeni aspekt funkcionalnosti aplikacije, što dovodi do arhitekture koja se lakše održava i koja je skalabilna.

- Korisnička interakcija sa *View* komponentom:
  - Korisnik stupa u interakciju sa *View* komponentom, npr. unosom teksta u formu i klikom na dugme i sl.

- *View* komponenta prima korisnički unos:
  - *View* komponenta prima korisnički unos i prosljeđuje ga kontroleru.

- Kontroler obrađuje korisnički unos:
  - Kontroler prima korisnički unos iz *View* komponente.
  - Ova komponenta interpretira unos, obavlja sve potrebne operacije i odlučuje kako odgovoriti korisniku, što najčešće uključuje sljedeća sljedeća dva koraka:
    1. Kontroler ažurira model:
       - Kontroler ažurira model na osnovu korisničkog unosa ili logike aplikacije.
    2. Kontroler ažurira *View* komponentu:
       - Kontroler ažurira *View* na osnovu promjena u modelu ili kao odgovor na korisnički unos.

- *View* prikazuje ažurirani korisnički interfejs:
  - *View* komponenta ažurira (najčešće kroz tzv. *rendering*) ažurirani korisnički interfejs na temelju promjena koje je izvršio kontroler.

Prateći ovakav tok komunikacije, osigurano je da svaka od komponenti ima svoje zaduženje. U *main()* metodi *Main* klase programskog rješenja, koje je implementirano prateći MVC šablon dizajna, potrebno je ispravno pozvati odgovarajuće metode na osnovu opisanog toka komunikacije između komponenti. Obzirom na to da model sadrži podatke, prvi korak je kreirati instancu klase *Osoba*. U nastavku je zatim moguće provjeriti da li navedena osoba ima pravo na stipendiju, a što je objedinjeno u okviru Listinga 12. Izlaz konzolnog ispisa bi trebao biti kao na Slici 2, jer navedena osoba ima ulogu studenta.

```
Osoba osoba = new Osoba(1,"Neko","Nekic","Neka adresa", new
Date(97,8,25),"2509997123456", Uloga.STUDENT);

System.out.println("Osoba ima pravo na stipendiju: " +
osoba.imaPravoNaStipendiju());
```

Listing 11. Korištenje model klase u okviru Main klase

```
Osoba ima pravo na stipendiju: true

Process finished with exit code 0
```

Slika 2. Rezultati pri korištenju metoda model klase

Za interakciju korisnika sa aplikacijom koristi se *View* komponenta, koju je potrebno instancirati kao u Listingu 12. Neka korisnik želi ažurirati ime i neka nova vrijednosti imena glasi "*Novo ime*". Ovo znači da se u *View* komponenti vrijednost atributa *ulazniTekst* postavlja na željenu vrijednost koristeći metodu *setUlazniTekst()*, a što je također prikazano u Listingu 12.

```
OsobaView osobaView = new OsobaView();

osobaView.setUlazniTekst("Novo ime");
```

Listing 12. Korištenje view klase u Main klasi

Za komunikaciju između *Model* i *View* komponente, koristi se kontroler. Potrebno je napomenuti da se kontroler instancira koristeći konstruktor sa dva parametra, koji predstavljaju prethodno spomenute komponente klasa *Osoba* i *OsobaView*. Kada korisnik pošalje zahtjev za ažuriranjem imena, to ustvari znači da je potrebno pozvati metodu *azurirajIme()* kontrolera. Nakon promjene imena, kontroler ažurira poruku *View* komponente i ispisuje je korisniku. Sve prethodno opisano prikazano je u Listingu 13, a izlaz u konzoli prikazan je na Slici 3.

```
OsobaController osobaController = new OsobaController(osoba,
osobaView);

osobaController.azurirajIme();

System.out.println("1) View ispisuje: " + osobaView.getPoruka());
```

Listing 13. Korištenje kontroler klase u Main klasi

```
Osoba ima pravo na stipendiju: true
1) View ispisuje: Ime je uspjesno azurirano!

Process finished with exit code 0
```

Slika 3. Izlaz View komponente

6.6. Učitavanje podataka iz datoteka

Do sada se podaci unosili direktno putem programskog koda (ili putem konzole), ali je mnogo bolje rješenje imati neki vid pohrane podataka. Inicijalna ideja bi bila kreirati datoteke u kojima će se čuvati podaci. Datoteke su dobro rješenje za pohranu podataka, zbog

jednostavnosti, *cross-platform* kompatibilnosti, jednostavnosti pravljenja sigurnosne kopije (eng. *backup*) i sl. Naravno, za kompleksnije upite i transakcije, bolje je koristiti baze podataka koje će se obrađivati na nekoj od narednih laboratorijskih vježbi. Datoteke mogu imati različitu strukturu a u okviru laboratorijskih vježbi će se spominjati dva tipa datoteka: tekstualne i XML datoteke. U ovoj vježbi će se za pohranu podataka o osobama koristiti tekstualna datoteka *osobe.txt* i XML datoteka *osobe.xml*. Izgled i sadržaj ovih datoteka, kao i način rada sa njima će biti opisani u narednim dijelovima laboratorijske vježbe. Za početak je potrebno kreirati paket naziva *data* u kojem će biti smještene ove datoteke. Metode koje će raditi učitavanje podataka iz datoteka će biti implementirane u sklopu modela i te metode će biti statičke. Potrebno je napomenuti i da se najčešće za učitavanje i rad sa podacima koristi sloj koji se naziva *Service*. Riječ je o sloju koji se nalazi između modela i kontrolera, međutim zbog jednostavnosti ovaj sloj neće biti detaljnije opisan niti korišten u sklopu laboratorijskih vježbi.

6.6.1. Tekstualne datoteke

Tekstualne datoteke su datoteke koje sadrže neformatirani tekst, tj. koje se mogu kreirati i uređivati pomoću bilo kojeg uređivača teksta. Svaki red u tekstualnoj datoteci tretira se kao običan tekst, što ga čini vrlo jednostavnim i jasnim. Primjer izgleda tekstualne datoteke *osobe.txt* je prikazan u Listingu 14.

```
1,John,Doe,Some Address,1995-01-15,1501995123456,STUDENT
2,Alice,Alister,Another Address,1980-05-20,2005980444444,NASTAVNO_OSOBLJE
```

Listing 14. Primjer strukture tekstualne datoteke

Prikazana tekstualna datoteka je formatirana tako da se u svakom redu nalaze podaci o određenoj osobi, razdvojeni zarezom. Ovakav tip formatiranja se naziva CSV (eng. *Comma-Separated Values*). Metoda koja će se koristiti za učitavanje podataka o osobama iz tekstualne datoteke nosit će naziv *ucitajOsobeIzTxtDatoteke()* i ima jedan parametar koji predstavlja putanju do datoteke. Na početku spomenute metode se kreira lista koja će sadržavati učitane osobe. Ona je inicijalno prazna, ali će se popunjavati podacima kada oni budu pročitani. Za čitanje tekstualnih datoteka koristit će se klasa *BufferedReader*. Ona učitava tekst iz ulaznog toka, spremajući znakove u međuspremnik (eng. *buffer*) kako bi se omogućilo efikasno čitanje znakova, nizova i redova. Definiranje metode i inicijalizacija su prikazani u Listingu 15.

```
public static List<Osoba> ucitajOsobeIzTxtDatoteke(String
putanjaDoDatoteke) throws IOException {
   List<Osoba> osobe = new ArrayList<>();
   BufferedReader reader = new BufferedReader(new
FileReader(putanjaDoDatoteke));
}
```

Listing 15. Učitavanje svih podataka iz tekstualne datoteke

Datoteka se iščitava liniju po liniju, sve dok se ne dođe do njenog kraja. Varijabla *linija* će sadržavati svaki red pročitan iz datoteke. Metodom *readLine()* će se učitavati jedan po jedan red u *while* petlji, sve dok poziv ove metode ne vrati *null*, što označava kraj datoteke. Metoda *readLine()* čita red teksta, pri čemu se smatra da red završava bilo kojim oznakom za novi red ("\n"), oznakom za tzv. *carriage return* ("\r") ili oznakom za *carriage return* nakon kojeg odmah slijedi znak za novi red. U petlji se u svakoj iteraciji red treba podijeliti na pojedinačne podatke kako bi se oni mogli pohraniti u listu osoba. Datoteka je formatirana tako da su podaci razdvojeni znakom zarez. Zbog toga će se koristiti metoda *split()*, koja vrši podjelu varijable tipa *String* na podstringove na osnovu znaka koji se proslijedi kao parametar ove metode. Bitno je naglasiti da učitane vrijednosti treba pretvoriti u odgovarajući tip kako bi se mogle proslijediti konstruktoru klase *Osoba*. Ovo znači da se *id* treba pretvoriti u *Integer*, datum u *Date* i uloga u *Uloga* tip. Sve što je prethodno opisano objedinjeno je u Listingu 16, pri čemu je prikazan i način zatvaranja čitača datoteke putem metode *close()* nakon završetka čitanja datoteke.

```
   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 16. Učitavanje podataka za svaki pojedinačni red iz tekstualne datoteke u model

Za potrebe pretvaranja *String* varijable u varijablu tipa *Date*, koristit će se klasa *SimpleDateFormat*, koja se koristi za formatiranje i parsiranje datuma. Dakle, u model je potrebno dodati privatnu statičku varijablu za parsiranje datuma na način prikazan u Listingu 17.

```
private static final SimpleDateFormat dateFormat = new
SimpleDateFormat("yyyy-MM-dd");
```

Listing 17. Atribut za formatiranje datuma u model klasi

Bitno je obratiti pažnju na par detalja u okviru prethodno opisane metode *ucitajOsobeIzTxtDatoteke()*. Implementirana metoda može baciti dva tipa izuzetaka: *IOException* (koji se može javiti prilikom učitavanja datoteke na navedenoj putanji) i *ParseException* (koji se može javiti prilikom parsiranja datuma). Kada je završeno učitavanje

datoteke, obavezno je pozvati metodu *close()* nad instancom *BufferedReader*. Ovime se zatvara ulazni tok i oslobađaju svi sistemski resursi povezani sa njim. Ako se čitač podataka ne zatvori eksplicitno, program bi mogao nastaviti zadržavati resurse, što može dovesti do curenja memorije ili iscrpljivanja resursa.

6.6.2. XML datoteke

XML (***eXtensible Markup Language***) **datoteke** su strukturirane tekstualne datoteke koje koriste *markup* jezik za definiranje podataka. Dizajnirani su za pohranu i prijenos podataka u čitljivom formatu. XML datoteke se sastoje od elemenata, atributa i vrijednosti koje su organizovane na hijerarhijski način. Svaki podatak se zapisuje korištenjem oznaka (eng. *tags*), što podatke čini lakšim za razumijevanje i analizu. Ovaj tip datoteka se najčešće koristi za razmjenu podataka između sistema, konfiguracijske datoteke i pohranu podataka u web aplikacijama. Primjer izgleda XML datoteke *osobe.xml* je prikazan u Listingu 18.

```
<osobe>
   <osoba>
       <id>1</id>
       <ime>John</ime>
       <prezime>Doe</prezime>
       <adresa>Some Address</adresa>
       <datumRodjenja>1995-01-15</datumRodjenja>
       <maticniBroj>1501995123456</maticniBroj>
       <uloga>STUDENT</uloga>
   </osoba>
   <osoba>
       <id>2</id>
       <ime>Alice</ime>
       <prezime>Alister</prezime>
       <adresa>Another Address</adresa>
       <datumRodjenja>1980-05-20</datumRodjenja>
       <maticniBroj>2005980444444</maticniBroj>
       <uloga>NASTAVNO_OSOBLJE</uloga>
   </osoba>
</osobe>
```

Listing 18. Primjer strukture XML datoteke

U korijenu (tzv. *root* element) se nalazi element *osobe*, koji sadrži podatke o svim osobama. Na nižem nivou (tj. djeca elementi korijenskog elementa) su *osoba* elementi. Svaki od tih elemenata ima informacije o id-u, imenu, prezimenu, adresi, datumu rođenja, matičnom broju i ulozi određene osobe. Slikoviti prikaz hijerarhijske strukture XML datoteke *osobe.xml* je prikazan u Listingu 19.

```
osobe
```

```
│
├── osoba
│   ├── id: 1
│   ├── ime: John
│   ├── prezime: Doe
│   ├── adresa: Some Address
│   ├── datumRodjenja: 1995-01-15
│   ├── maticniBroj: 1501995123456
│   └── uloga: STUDENT
│
├── osoba
    ├── id: 2
    ├── ime: Alice
    ├── prezime: Alister
    ├── adresa: Another Address
    ├── datumRodjenja: 1980-05-20
    ├── maticniBroj: 2005980444444
    └── uloga: NASTAVNO_OSOBLJE
```

Listing 19. Hijerarhijska struktura XML datoteke

Metoda koja će se koristiti za učitavanje podataka o osobama iz tekstualne datoteke nosit će naziv *ucitajOsobeIzXmlDatoteke()*. Ima jedan parametar koji predstavlja putanju do datoteke. Na početku spomenute metode se kreira lista koja će sadržavati učitane osobe. Ona je inicijalno prazna, ali će se popunjavati podacima kada oni budu pročitani. Za čitanje XML datoteka koristit će se klasa *File*, koja učitava datoteku na specificiranoj putanji. XML datoteku je potrebno parsirati, za što će se koristiti *DocumentBuilder* klasa. Ova klasa omogućava dohvatanje DOM (*Document Object Model*) instance iz XML dokumenta. U narednom koraku se radi normalizacija strukture dokumenta. Ovaj korak osigurava čist prikaz XML strukture. Sve što je prethodno opisano objedinjeno je u Listingu 20.

```
public static List<Osoba> ucitajOsobeIzXmlDatoteke(String
putanjaDoDatoteke) throws Exception {
   List<Osoba> osobe = new ArrayList<>();
   File xmlDatoteka = new File(putanjaDoDatoteke);

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

   doc.getDocumentElement().normalize();
```

Listing 20. Učitavanje svih podataka iz XML datoteke

XML DOM se sastoji od čvorova, koji se čitaju na osnovu njihovih oznaka. Datoteka *osobe.xml* sadrži oznaku *osoba* za elemente koji predstavljaju pojedinačne osobe, što znači da takvi elementi sadrže neophodne podatke. Sve čvorove sa datom oznakom moguće je učitati koristeći funkciju *getElementsByTagName()*. Petljom se prolazi kroz listu čvorova, odnosno kroz

pojedinačne osobe. Svaki od čvorova u listi čvorova se sastoji od 7 elemenata sa oznakama *id, ime, prezime, adresa, datumRodjenja, maticniBroj* i *uloga*, respektivno. Za učitavanje podataka o svakoj pojedinoj osobi također se koristi metoda *getElementsByTagName()* sa odgovarajućom oznakom (naprimjer, oznaka "*id*"). U ovom primjeru, *element* predstavlja jedan čvor iz liste čvorova (ustvari, *element* je jedna osoba). Pozivom *getElementsByTagName("id")* se pronalazi element sa oznakom *id* a koji je dijete elementa koji predstavlja trenutnu osobu. Obzirom na to da svaka osoba ima samo jedan *id*, onda se preuzima element oznake *id* na poziciji 0, što je u programskom kodu prikazano sa *item(0)*. Sadržaj tog elementa se preuzima pozivom funkcije *getTextContent()*. Sve što je prethodno opisano objedinjeno je u Listingu 21.

```
   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 21. Učitavanje podataka za svaki pojedinačni red iz XML datoteke u model

6.7. Pozivanje metoda za učitavanje podataka iz kontrolera

Nakon implementacije prethodno opisanih metoda za učitavanje podataka iz datoteka u model klasi, iste je potrebno pozvati putem kontroler klase. Za tu svrhu će se koristiti metode *dajOsobeIzTxtDatoteke()* i *dajOsobeIzXmlDatoteke()*, koje će biti implementirane u *OsobaController* klasi. Ove metode samo pozivaju metode modela za učitavanje podataka iz tekstualnih i XML datoteka. Jedini im je parametar putanja do datoteke, koju proslijede odgovarajućoj metodi modela. Metoda *dajOsobeIzTxtDatoteke()* je prikazana u Listingu 22, a na isti način potrebno je kreirati metodu *dajOsobeIzXmlDatoteke()* uz jedinu razliku što se poziva odgovarajuća metoda modela za XML datoteke.

```
public void dajOsobeIzTxtDatoteke(String filePath)
{
    try
    {
        List<Osoba> osobe = Osoba.ucitajOsobeIzTxtDatoteke(filePath);
        String poruka = "Osobe ucitane iz txt datoteke su:\n";
        for (Osoba osoba : osobe)
        {
            poruka += osoba.toString() + "\n";
        }
        view.setPoruka(poruka);
    }
    catch(Exception e)
    {
        view.setPoruka("Greska: " + e.getMessage());
    }
}
```

Listing 22. Kontrolerska metoda za učitavanje podataka iz tekstualne datoteke

U obje metode za učitavanje podataka iz datoteke, kada je datoteka učitana bez problema, onda se korisniku ispisuje poruka sa informacijama o osobama čiji su podaci učitani. Slično, ako se desi greška, poruka greške će biti prikazana korisniku. Kako bi metode ispravno radile prikaz poruke, potrebno je u modelu preklopiti *toString()* metodu na način prikazan u Listingu 23, a što je već detaljnije opisano u okviru prethodnih laboratorijskih vježbi.

```
@Override
public String toString() {
    return "Osoba{" +
            "id='" + id + '\'' +
            ", ime='" + ime + '\'' +
            ", prezime='" + prezime + '\'' +
            ", adresa='" + adresa + '\'' +
            ", datumRodjenja=" + datumRodjenja +
            ", maticniBroj='" + maticniBroj + '\'' +
```

```
            ", uloga=" + uloga +
            '}';
}
```

Listing 23. Metoda za adekvatan prikaz informacija o model klasi

U *main()* metodi *Main* klase, sada se mogu pozvati metode za učitavanje podataka iz tekstualne i XML datoteke kao u Listingu 24. Prilikom navođenja putanje do datoteke, potrebno je pratiti relativnu putanju u odnosu na *root* direktorij. Ukoliko su ispravno praćeni koraci za kreiranje paketa i datoteka, onda će putanje za tekstualnu i XML datoteku biti "*src/data/osobe.txt*" i "*src/data/osobe.xml*", a dobiti će se izlaz u konzoli kao na Slici 4.

```
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 24. Poziv metoda kontrolera za učitavanje podataka iz datoteka

```
2) View ispisuje: Osobe ucitane iz txt datoteke su:
Osoba{id='1', ime='John', prezime='Doe', adresa='Some Address', datumRodjenja=Sun Jan 15 00:00:00 CET 1995, maticniBroj='1501995123456', uloga=STUDENT}
Osoba{id='2', ime='Alice', prezime='Alister', adresa='Another Address', datumRodjenja=Tue May 20 00:00:00 CEST 1980, maticniBroj='2005980444444', uloga=NASTAVNO_OSOBLJE}

3) View ispisuje: Osobe ucitane iz xml datoteke su:
Osoba{id='1', ime='John', prezime='Doe', adresa='Some Address', datumRodjenja=Sun Jan 15 00:00:00 CET 1995, maticniBroj='1501995123456', uloga=STUDENT}
Osoba{id='2', ime='Alice', prezime='Alister', adresa='Another Address', datumRodjenja=Tue May 20 00:00:00 CEST 1980, maticniBroj='2005980444444', uloga=NASTAVNO_OSOBLJE}


Process finished with exit code 0
```

Slika 4. Prikaz informacija o svim osobama učitanim iz datoteka u konzoli

6.8. 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.

Kreirati testove za model klasu, čija je implementacija prikazana u laboratorijskoj vježbi. Napisati testove koji provjeravaju:

- da je osoba ispravno kreirana;
- da je ime neispravno;
- da matični broj nema ispravnu dužinu;
- da matični broj nije podudaran sa datumom rođenja;
- da je neispravna putanja do tekstualne datoteke.

Zadatak 2.

Po uzoru na implementirane komponente MVC šablona dizajna za osobu, uraditi implementaciju istih komponenti za klasu *Predmet*. U model smjestiti atribute tipa *String* naziv i tipa *Double* ECTS. Uraditi validaciju atributa tako da *naziv* mora imati dužinu između 5 i 100 znakova, dok *ECTS* minimalno može imati vrijednost 5.0 a maksimalno 20.0. Pri tome, varijabla *ECTS* može imati samo 0 ili 5 kao vrijednost prve decimale (dakle, dozvoljene su vrijednosti tipa 7.0, 7.5, 9.5, ali ne i 8.2, 15.7 i slično). Kroz *View* i *Controller* komponente omogućiti korisniku da ažurira naziv predmeta i vrijednost ECTS kredita. Nakon implementacije MVC šablona dizajna za predmete, učitati informacije o predmetima iz tekstualne ili XML datoteke (dovoljno je izabrati samo jedan od ova dva tipa datoteka).

Napomena: *View* komponenta može biti identična kao i *OsobaView*, međutim može se i ponovo definisati kao *PredmetView*, ukoliko je to lakše i jasnije za implementaciju.