9 Tokovi, fajlovi i serijalizacija objekata

Svi podaci smešteni u Java promenljivama, odnosno instancama klasa, su privremene prirode, i u najboljem slučaju postoje u memoriji računara dok se program izvršava. Za dugoročno čuvanje podataka, tj. čuvanje nakon prestanka rada programa, računari koriste fajlove. Fajlovi se čuvaju na *sekundarnim uređajima za skladištenje* (eng. *secondary storage devices*), kao što su hard diskovi, fleš diskovi, CD i DVD medijumi itd.

Što se tiče programskog rada sa fajlovima, značajna su dva tipa (formata) fajlova: *tekstualni* i *binarni*. U ovoj glavi opisujemo osnovne interfejse i klase za rad sa ova dva tipa. Takođe, objasnićemo i pojmove serijalizacije i deserijalizacije objekata, koji omogućavaju smeštanje, odnosno čitanje čitavih objekata iz fajlova. Započnimo sa pojmom toka.

9.1 Pojam toka. Standardni tokovi u Javi

Svaki fajl Java vidi kao sekvencijalni niz bajtova. Kad se fajl otvori, dodeljuje mu se *tok* (eng. *stream*). Tok predstavlja komunikacioni kanal između fajla i programa. U tokovima, podaci se mogu upisivati i čitati kao niz bajtova ili karaktera. U tokovima zasnovanim na bajtovima, podaci se nalaze u binarnom formatu (npr. broj 1234567 bi se predstavio sa 4 bajta, karakter `'#'` sa 2 bajta). Sa druge strane, u tekstualnom režimu, podaci se smještaju u formi sekvence karaktera, pri čemu svaki karakter zauzima 2 bajta. Fajlovi kreirani korišćenjem tokova zasnovanih na bajtovima se nazivaju *binarni fajlovi*, dok se fajlovi kreirani pomoću tokova zasnovanih na karakterima nazivaju *tekstualni fajlovi*. Prednost rada sa tekstualnim fajlovima je što su čitljivi za čoveka, dok binarni fajlovi nisu.

Svaki operativni sistem obezbeđuje mehanizam za određivanje kraja fajla, kao što je *indikator kraja fajla* (eng. *end-of-file indicator*) ili na osnovu ukupnog broja bajtova fajla o čemu operativni sistem interno vodi evidenciju. Programeri ne moraju znati na koji način operativni sistem upravlja fajlovima, odnosno kako se određuje kraj fajla. Putem odgovarajućih procedura, u programu koji radi sa fajlovima se jednostavno dobija indikacija od operativnog sistema da smo došli do kraja fajla. Na primer, C funkcija `feof` za parametar ima pokazivač na fajl i vraća nenultu vrednost (logička istina) kad dođemo do kraja tog fajla.

Java program otvara fajl kreiranjem objekta i pridruživanjem toka bajtova ili karaktera tom objektu. Konstruktor objekta komunicira sa operativnim sistemom da bi otvorio fajl. Java

može povezati tokove sa različitim uređajima. Kada Java program započne izvršavanje, on kreira tri objekta toka koji su povezani sa uređajima: `Sistem.in`, `Sistem.out` i `Sistem.err`. Objekat `Sistem.in` (standardni ulazni tok) omogućava čitanje podataka unesenih preko tastature. Objekat `Sistem.out` (standardni izlazni tok) omogućava štampanje karaktera na ekranu. Objekat `Sistem.err` (standardni tok greške) omogućava štampanje poruka greške na ekranu.

Svaki tok se može preusmeriti. Za `Sistem.in` to znači da je moguće čitati bajtove/karaktere iz drugog izvora. Za `Sistem.out` i `Sistem.err`, to znači da se karakteri mogu štampati drugde, npr. u fajl. Klasa `System` obezbeđuje metode `setIn`, `setOut` i `setErr` za preusmeravanje ulaznog, izlaznog i toka greške, respektivno.

9.2 Paket `java.io`

Za izvršenje ulazno/izlaznih (IO) operacija sa fajlovima, Java koristi klase iz paketa `java.io` i `java.nio`. Postoji nekoliko fundamentalnih razlika između ova dva paketa. Prva velika razlika je da je IO orijentisan na tok, a NIO (novi IO) na bafer. Šta to znači? Java IO orijentisan na tok znači da se podaci u toku čitaju sekvencijalno, bajt po bajt, ili više bajtova "u komadu". Ne vrši se keširanje bajtova. Takođe, ne možemo se pomerati proizvoljan broj bajtova unapred ili unazad u toku. Da bi se to postiglo, podaci se prvo moraju smestiti u bafer. Sa druge strane, kod NIO pristupa, podaci se prvo smeštaju u bafer, odakle se dalje procesiraju. Možemo se kretati unapred ili unazad u baferu koliko god je potrebno, što nam daje veću fleksibilnost tokom obrade podataka u odnosu na IO pristup. Ova pogodnost ne dolazi bez svoje cene - moramo voditi računa o podacima u baferu. Na primer, prilikom učitavanja novih podataka u bafer, moramo voditi računa da ne izgubimo podatke iz bafera koje nismo obradili.

Druga bitna razlika između IO i NIO pristupa je da je veliki broj IO tokova blokirajući, što znači da se prilikom izvršenja operacija čitanja i upisa, blokira predmetna programska nit sve dok se ne pojave podaci za čitanje, odnosno sve dok se podaci u potpunosti ne upišu u tok. Nit ne može ništa drugo da radi u međuvremenu. Javin NIO neblokirajući režim omogućava niti da pročita samo ono što se trenutno nalazi u toku ili čak ništa, ukoliko podaci trenutno nisu dostupni. Umesto da nit ostane blokirana dok se ne pojave podaci za čitanje, ona može da radi nešto drugo. Slično, prilikom upisa podataka u tok, nit ne mora da čeka upis svih podataka u tok, već može raditi nešto drugo u međuvremenu. Na primer, nit može da vrši IO operacije sa drugim tokovima u međuvremenu. Na taj način, jedna nit može upravljati IO operacijama većeg broja tokova.

Mi ćemo u nastavku obraditi neke korisne klase iz paketa `java.io`. Ovde izdvajamo klase `FileInputStream` (za čitanje podataka iz fajla na nivou bajta), `FileOutputStream` (za upis podataka u fajl na nivou bajta), `FileReader` (za čitanje podataka iz fajla na nivou

Prvo izvršenje:

```
Uneti ime fajla ili foldera: C:\Program Files (x86)\texstudio
texstudio postoji
To nije fajl
To je folder
Veličina: 4096 bajta
Poslednja izmena: 1618300520977
Roditeljski folder: C:\Program Files (x86)

Sadržaj foldera:
dictionaries
Doc.pdf
help
share
templates
texstudio.exe
TexTablet
translations
uninstall.exe
```

Drugo izvršenje:

```
Uneti ime fajla ili foldera: C:\Program Files (x86)\texstudio\Doc.pdf
Doc.pdf postoji
To je fajl
To nije folder
Veličina: 128946 bajta
Poslednja izmena: 1618234408258
Roditeljski folder: C:\Program Files (x86)\texstudio
```

Prilikom navođenja putanje do fajla/foldera, koristili smo separator `'\'`. U pitanju je separator za operativni sistem Windows. Kod operativnih sistema Linux ili Mac OS X, separator je `'/'`. Java procesira podjednako oba separatora, tj. mogli smo uneti putanju

```
C:\Program Files (x86)/texstudio
```

i program bi i dalje radio korektno. Ako želimo da budemo sigurni da koristimo pravi separator, možemo koristiti `String` konstantu `File.separator`, koja predstavlja separator karakter na datom operativnom sistemu.

9.4 Tekstualni fajlovi sa sekvencijalnim pristupom

U ovom poglavlju demonstriramo rad sa tekstualnim fajlovima sa sekvencijalnim pristupom. Kreiraćemo tekstualni fajl na osnovu unosa korisnika (prvi primer), a nakon toga otvaramo kreirani fajl, čitamo i štampamo njegov sadržaj (drugi primer). Preciznije, u prvom primeru, u svrhu upisa podataka u tekstualni fajl, kreiramo novu klasu `Radnik`, i

podatke instanci te klase upisujemo u fajl. U drugom primeru, na osnovu sadržaja fajla kreiranog u prvom primeru, kreiramo instance klase `Radnik` koje štampamo u konzoli.

Kako klasu `Radnik` koriste oba primera koji slede, definiciju klase dajemo pre primera.

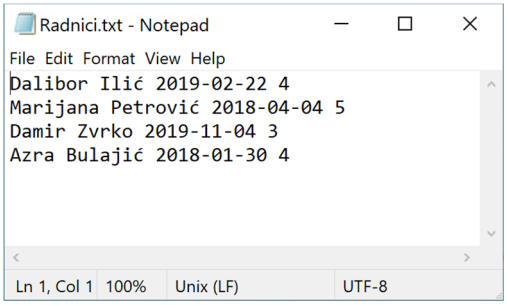
```
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;

public class Radnik {

  // Klasa koja modeluje radnika

  private String ime;
  private String prezime;
  private LocalDate pocetak;
  private int sprema;
  private static final DateTimeFormatter df =
                              DateTimeFormatter.ofPattern("dd/MM/yyyy");

  public Radnik() {
    this("", "", LocalDate.now(), 0);
  }

  public Radnik(String ime, String prezime, LocalDate pocetak, int sprema) {
    setIme(ime);
    setPrezime(prezime);
    setPocetak(pocetak);
    setSprema(sprema);
  }

  public String getIme() {
    return ime;
  }

  public void setIme(String ime) {
    this.ime = ime;
  }

  public String getPrezime() {
    return prezime;
  }

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

  public LocalDate getPocetak() {
    return pocetak;
  }

  public void setPocetak(LocalDate pocetak) {
```

```
import java.util.Scanner;
import java.io.FileNotFoundException;
import java.io.UnsupportedEncodingException;
import java.time.LocalDate;

public class UpisRadnikaUFajl {

  // Klasa koja implementira operacije kreiranja tekstualnog fajla,
  // upisa podataka o radnicima i zatvaranje fajla

  private static Formatter izlaz;

  public void otvoriFajl() {
    try {
      izlaz = new Formatter("Radnici.txt", "UTF-8");
    }
    catch(FileNotFoundException e) {
      System.err.println("Greška pri otvaranju.");
      System.exit(1);
    }
    catch(UnsupportedEncodingException e) {
      System.err.println("Kodiranje nije podržano.");
      System.exit(1);
    }
  }

  public void upisiRadnikeUFajl() {
    Radnik r = new Radnik();
    Scanner unos = new Scanner(System.in);

    System.out.print("Koliko radnika želite da unesete? ");
    int brojRadnika = unos.nextInt();
    System.out.println("Unesite radnike u formatu: <ime> <prezime> " +
                       "<datum> <sprema>");
    System.out.println("(<datum> unesite u formatu yyyy-mm-dd)");
    for(int i = 0; i < brojRadnika; i++) {
      r.setIme(unos.next());
      r.setPrezime(unos.next());
      r.setPocetak(LocalDate.parse(unos.next()));
      r.setSprema(unos.nextInt());
      izlaz.format("%s %s %s %d\n", r.getIme(),
                                    r.getPrezime(),
                                    r.getPocetak(),
                                    r.getSprema());
    }

    unos.close();
  }

  public void zatvoriFajl() {
```

klase na osnovu učitanog stringa. Učitane podatke kopiramo u odgovarajuće podatke instance `Radnik`.

Metodom `format` objekta `Formatter` vršimo upis formatiranog teksta u fajl `Radnici.txt`, na potpuno isti način kako metoda `System.out.printf` vrši štampanje teksta u konzoli. `format` baca dva neproverena izuzetka tipa `IllegalFormatException` i `FormatterClosedException`, koje nismo obradili u ovom primeru. Prvi se dešava ukoliko je sintaksa format stringa neispravna, ako specifikator formata nije kompatibilan sa prosleđenim argumentima ili ako nije prosleđen dovoljan broj argumenata format stringu. Drugi izuzetak se dešava u slučaju da je `Formatter` već zatvoren metodom `close`.

U metodi `main` klase `TestiranjeUpisaUFajl` kreiramo instancu klase `UpisRadnikaUFajl` i pozivom odgovarajućih metoda testiramo otvaranje fajla, upis podataka i zatvaranje fajla. Sadržaj tekstualnog fajla `Radnici.txt`, kreiran u tekućem folderu, je prikazan na slici 9.1.



Slika 9.1 Sadržaj tekstualnog fajla nakon unosa radnika.

9.4.2 Čitanje podataka iz tekstualnog fajla

Ovde ćemo ilustrovati otvaranje postojećeg tekstualnog fajla (kreiranog u prethodnom poglavlju), čitanje podataka iz tog fajla i kreiranje instanci klase `Radnik` na osnovu učitanih podataka. U tu svrhu, kreiraćemo klasu `CitanjeRadnikaIzFajla`, i u okviru metode `main` te klase ćemo testirati ove radnje. Realizacija ove klase data je u nastavku.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.time.LocalDate;
import java.util.Scanner;

public class CitanjeRadnikaIzFajla {

  // Klasa koja implementira otvaranje fajla i čitanje podataka iz fajla
  private static Scanner ulaz;

  public static void main(String[] args) {
    otvoriFajl();
    citajRadnikeIzFajla();
    zatvoriFajl();
```

Razlog tome je formatiranje datuma prema šablonu (videti statički podatak `df` klase `DateTimeFormatter` u klasi `Radnik`, poglavlje 9.4) u metodi `toString` klase `Radnik`.

9.4.3 Dopisivanje podataka u tekstualni fajl

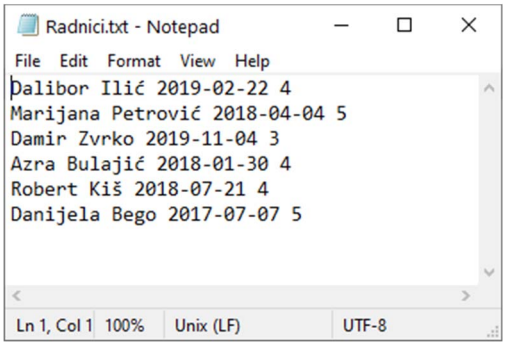
Za dopisivanje podataka u postojeći tekstualni fajl, koristićemo `FileWriter` klasu, koja omogućava upis podataka u fajl na nivou karaktera. Klasa koja implementira operaciju dopisivanja podataka u fajl je data u nastavku.

```
import java.io.FileWriter;
import java.io.IOException;
import java.time.LocalDate;
import java.util.Scanner;
import java.nio.charset.StandardCharsets;

public class DopisivanjeRadnikaUFajl {

  // Klasa koja implementira operaciju dopisivanja podataka u fajl

  private static FileWriter dopisivanje;

  public static void main(String[] args) {
    try {
      otvoriFajl();
      dopisiRadnikeUFajl();
      zatvoriFajl();
    }
    catch(IOException e) {
      System.err.println("Greška pri radu sa fajlom.");
      System.exit(1);
    }
  }

  public static void otvoriFajl() throws IOException {
    dopisivanje = new FileWriter("Radnici.txt",
                                 StandardCharsets.UTF_8,
                                 true);
  }

  public static void dopisiRadnikeUFajl() throws IOException {
    Radnik r = new Radnik();
    Scanner unos = new Scanner(System.in);

    System.out.print("Koliko radnika želite da dopišete? ");
    int brojRadnika = unos.nextInt();

    System.out.println("Unesite radnike u formatu: <ime> <prezime> " +
                       "<datum> <sprema>");
    System.out.println("(<datum> unesite u formatu yyyy-mm-dd)");
    for(int i = 0; i < brojRadnika; i++) {
```

```
        r.setIme(unos.next());
        r.setPrezime(unos.next());
        r.setPocetak(LocalDate.parse(unos.next()));
        r.setSprema(unos.nextInt());
        String red = String.format("%s %s %s %d\n",
                                   r.getIme(),
                                   r.getPrezime(),
                                   r.getPocetak(),
                                   r.getSprema());
        dopisivanje.write(red);
      }

      unos.close();
    }

    public static void zatvoriFajl() throws IOException {
      if(dopisivanje != null)
        dopisivanje.close();
    }
}
```

```
Koliko radnika želite da dopišete? 2
Unesite radnike u formatu: <ime> <prezime> <datum> <sprema>
(<datum> unesite u formatu yyyy-mm-dd)
Robert Kiš 2017-10-22 4
Danijela Bego 2018-07-07 5
```

Izgled tekstualnog fajla nakon dopisivanja radnika je dat na slici 9.2.



Slika 9.2. Sadržaj tekstualnog fajla nakon dopisivanja radnika.

Klasa `FileWriter` ima devet konstruktora, a ovde koristimo verziju sa tri parametra: imenom fajla (`String`), kôdnim skupom karaktera (`Charset`) i parametrom koji definiše dopisivanje (`boolean`). Za kôdni skup karaktera koristimo UTF-8, dobijen statičkim podatkom `UTF_8` klase `StandardCharsets` (paket `java.nio.charset`). Parametar za dopisivanje ima vrednost `true`, što omogućava dopisivanje u postojeći fajl.

Dopisivanje podataka u fajl vršimo metodom `write` klase `FileWriter`, koja za argument ima podatak tipa `String`. U pitanju je nasleđena metoda iz apstraktne klase `Writer`, koja služi za upisivanje u tokove karaktera. Ovoj metodi prosleđujemo red teksta, koji predstavlja jednog radnika, dobijen metodom `String.format`. Red je formatiran na isti način kao redovi teksta upisani nakon kreiranja fajla (metoda `UpisiRadnikeUFajl` u klasi `UpisRadnikaUFajl` u poglavlju 9.4.1).

Korišćeni konstruktor klase `FileWriter`, kao i metode `write` i `close` ove klase, bacaju izuzetak tipa `IOException`. Ove izuzetke nismo obrađivali u metodama gde se pojavljuju, već smo proglasili da date metode bacaju izuzetak tog tipa. U okviru `try` bloka metode `main` smo obradili izuzetak ispisom odgovarajuće poruke i prekidom izvršenja programa.

9.5 Serijalizacija objekata

Prilikom upisa podataka u tekstualni fajl, dolazi do gubitka informacija o tipu upisanih vrednosti. Na primer, prilikom upisa datuma, gubimo informaciju o klasi koja modeluje datum u Java programu. Klasa `LocalDate` koju smo prethodno koristili nije i jedina Javina klasa koja omogućava rad sa datumima. Takođe, podatak `2017-10-22` u fajlu može biti dobijen upisom stringa ili tri cela broja razdvojena karakterom `'-'`. Tekstualni fajl sadrži vrednost, ali ne i tip podatka.

Za upis i čitanje čitavih objekata u tokove, Java obezbeđuje *serijalizaciju objekata*. Serijalizovan objekat je objekat predstavljen sekvencom bajtova koja uključuje kako podatke objekta, tako i informaciju o tipu objekta i tipu podataka smeštenih u objektu (prisetimo se pojma kompozicije u OO programiranju). Nakon što je serijalizovani objekat smešten u tok, on se iz toka može pročitati i *deserijalizovati*, tj. na osnovu informacija o tipu objekta i vrednosti i tipu njegovih podataka, možemo kreirati instancu predmetne klase u memoriji.

9.5.1 Klase `ObjectInputStream` i `ObjectOutputStream`

Klase `ObjectOutputStream` i `ObjectInputStream` (paket `java.io`) omogućavaju serijalizaciju (upis objekta u tok) i deserijalizaciju objekata (čitanje objekta iz toka), respektivno. Ove dve klase implementiraju interfejse `ObjectOutput` i `ObjectInput`, respektivno. Da bismo za tok koristili fajl, potrebno je da objekte `ObjectOutputStream` i `ObjectInputStream` inicijalizujemo objektima tokova koji upisuju i čitaju podatke iz fajla. Inicijalizacija objekta toka sa drugim objektom toka se naziva *omotavanje* (eng. *wrapping*).

Klase `ObjectOutputStream` i `ObjectInputStream` vrše upis i čitanje bajtova, pri čemu one ne znaju gde treba da upisuju ili odakle da čitaju bajtove. Tu informaciju dobijaju od objekta toka koji se prosleđuje njihovom konstruktoru. Preciznije, objekat toka koji prosleđujemo `ObjectOutputStream` konstruktoru preuzima bajt-reprezentaciju objekta koji kreira `ObjectOutputStream` i te bajtove upisuje u tok (fajl, mrežnu konekciju itd.) Sa

```
  }

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

  public LocalDate getPocetak() {
    return pocetak;
  }

  public void setPocetak(LocalDate pocetak) {
    this.pocetak = pocetak;
  }

  public int getSprema() {
    return sprema;
  }

  public void setSprema(int sprema) {
    this.sprema = sprema;
  }

  @Override
  public String toString() {
    return String.format("%-10s %-12s sprema: %d  početak: %s",
                         this.getIme(),
                         this.getPrezime(),
                         this.getSprema(),
                         df.format(this.getPocetak()));
  }
}
```

9.5.3 Kreiranje fajla korišćenjem serijalizacije objekata

Ovde ilustrujemo kreiranje binarnog fajla i upis podataka o radnicima u taj fajl. Podatke o radnicima unosimo preko tastature i na osnovu njih kreiramo instance klase `Radnik`, koje ćemo upisujemo u fajl metodom `writeObject` interfejsa `ObjectOutput`, kako je opisano u poglavlju 9.5.1.

Za kreiranje fajla i upis podataka koristimo klasu `UpisRadnikaUFajl`. Ovu klasu testiramo koristeći klasu `TestiranjeUpisaUFajl`. Realizacija ove dve klase sledi, kao i jedno izvršenje aplikacije. Razlike u odnosu na istoimene klase iz poglavlja 9.4.1 označene su narandžastom pozadinom.

```
import java.util.Scanner;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.time.LocalDate;
```

9.5.4 Čitanje podataka iz binarnog fajla

Klasa `ObjectInputStream` omogućava deserijalizaciju fajlova, tj. kreiranje objekata određene klase na osnovu njihove bajt-reprezentacije sadržane u toku. Za tok, u sledećem primeru, koristimo binarni fajl Radnici.ser, kreiran u prethodnom poglavlju. Klasa koja otvara binarni fajl i čita objekte iz njega se naziva `CitanjeRadnikaIzFajla`. U pitanju je modifikovana verzija klase `CitanjeRadnikaIzFajla` iz poglavlja 9.4.2 i razlike u odnosu na tu klasu su označene narandžastom pozadinom.

```
import java.io.FileNotFoundException;
import java.io.EOFException;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

public class CitanjeRadnikaIzFajla {

  // Klasa koja implementira otvaranje i čitanje objekata iz binarnog fajla

  private static ObjectInputStream ulaz;

  public static void main(String[] args) {
    try {
      otvoriFajl();
      citajRadnikeIzFajla();
      zatvoriFajl();
    }
    catch(IOException e) {
      System.err.println("Greška pri radu sa fajlom.");
      System.exit(1);
    }
  }

  public static void otvoriFajl() throws IOException {
    try {
      FileInputStream fis = new FileInputStream("Radnici.ser");
      ulaz = new ObjectInputStream(fis);
    }
    catch(FileNotFoundException e) {
      System.err.println("Greška pri otvaranju.");
      System.exit(1);
    }
  }

  public static void citajRadnikeIzFajla() throws IOException {
    try {
      while(true) {
        Radnik r = (Radnik) ulaz.readObject();
        System.out.println(r);
```

```
      }
    }
    catch(ClassNotFoundException e) {
      e.printStackTrace();
    }
    catch(EOFException e) {
      System.out.println("Kraj čitanja binarnog fajla");
    }
  }

  public static void zatvoriFajl() throws IOException {
    if(ulaz != null)
      ulaz.close();
  }
}
```

```
Raša       Popov        sprema: 4  početak: 04/04/2016
Anja       Raičević     sprema: 6  početak: 21/04/2019
Nino       Taljanović   sprema: 5  početak: 18/10/2018
Kraj čitanja binarnog fajla
```

U metodi `otvoriFajl` otvaramo binarni fajl Radnici.ser naredbom

```
FileInputStream fis = new FileInputStream("Radnici.ser");
```

`FileInputStream` klasa implicira da se fajl otvara za čitanje. Objekat ove klase se prosleđuje `ObjectInputStream` konstruktoru u naredbi

```
ulaz = new ObjectInputStream(fis);
```

Objekat `ulaz` omogućava deserijalizaciju objekata klase `Radnik`, tj. čitanje objekata iz fajla Radnici.ser. Čitanje je izvršeno naredbom

```
Radnik r = (Radnik) ulaz.readObject();
```

u metodi `citajRadnikeIzFajla`. Pošto metoda `readObject` vraća `Object` instancu, potrebno ju je konvertovati naniže (*downcast*-ovati) pre upisa u referencijsku promenljivu `Radnik`.

Metoda `readObject` baca `EOFException` u slučaju pokušaja čitanja nakon kraja fajla. Za razliku od tekstualnih fajlova, kod binarnih fajlova ne postoji elegantan način određivanja kraja fajla, već kraj čitanja proglašavamo kad se desi izuzetak `EOFException`. Metoda `readObject` takođe baca i `ClassNotFoundException` ukoliko se klasa objekata koje čitamo ne može locirati, npr. ako okruženje primaoca ne sadrži tu klasu.

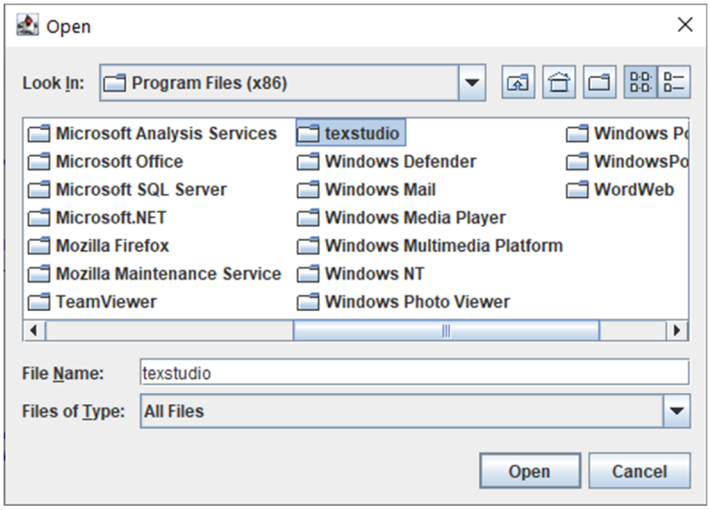
Sve operacije sa `ObjectInputStream` objektom u našem primeru bacaju izuzetak tipa `IOException`. Ove izuzetke nismo obrađivali u metodama gde se pojavljuju, već smo proglasili da date metode bacaju `IOException` izuzetak, a u okviru `try` bloka metode `main` obradili smo taj izuzetak ispisom odgovarajuće poruke i prekidom izvršenja programa.

9.6 Odabir fajlova i foldera pomoću klase `JFileChooser`

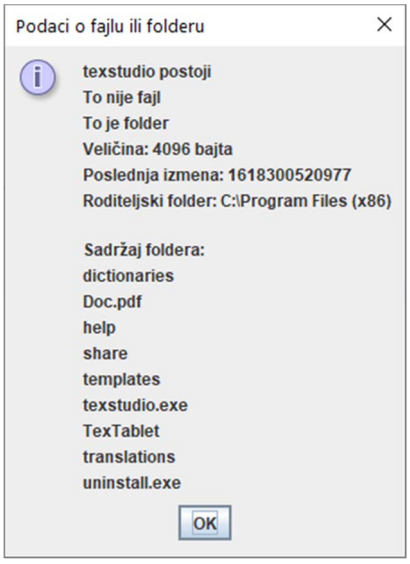
Prilikom rada sa fajlovima, poželjno je omogućiti korisniku da odabere fajl ili folder koristeći grafički, a ne tekstualni interfejs, kako je to urađeno u poglavlju 9.3. Klasa `JFileChooser` iz paketa `javax.swing` omogućava jednostavan odabir fajlova i foldera koristeći grafički korisnički interfejs operativnog sistema.

Modifikovaćemo primer iz poglavlja 9.3. Umesto unosa putanje do fajla/foldera u konzoli aplikacije, koristićemo `JFileChooser` objekat za odabir fajla/foldera. Nakon toga ćemo, ukoliko predmetni fajl/folder postoji, prikazati nekoliko opisnih poruka u okviru grafičkog prozora za prikaz poruke.

```
import java.io.File;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;

public class DemonstracijaJFileChooser {

  // Dobijanje informacija o fajlovima i folderima
  // koristeći klase JfileChooser i File

  public static void main(String [] args) {
    File ime = vratiFajlIliFolder();
    String ispis;

    if(ime.exists()) {
      ispis = String.format("%s\n%s\n%s\n%s\n%s\n%s",
                  ime.getName() + " postoji",
                  ime.isFile() ? "To je fajl" : "To nije fajl",
                  ime.isDirectory() ? "To je folder" : "To nije folder",
                  "Veličina: " + ime.length() + " bajta",
                  "Poslednja izmena: " + ime.lastModified(),
                  "Roditeljski folder: " + ime.getParent());
    }
    else
      ispis = String.format("%s ne postoji", ime);

    if(ime.isDirectory()) {
      ispis += "\n\nSadržaj foldera:\n";
      ispis += String.join("\n", ime.list());
    }

    JOptionPane.showMessageDialog(null,
                                  ispis,
                                  "Podaci o fajlu ili folderu",
                                  JOptionPane.INFORMATION_MESSAGE);
  }

  private static File vratiFajlIliFolder() {
```



Slika 9.3. `JFileChooser` prozor za odabir fajlova i foldera.



Slika 9.4. Prozor sa podacima o odabranom folderu i njegovom sadržaju.

Ukoliko je odabran folder, u prozoru biće prikazan listing njegovog sadržaja. Elegantan način da se niz stringova (imena foldera i fajlova u predmetnom folderu) konvertuje u string po principu *jedan string – jedan red*, je korišćenjem metode `join` klase `String`, kako je opisano u poglavlju 5.1.4. U našem primeru je to urađeno naredbom

```
ispis += String.join("\n", ime.list());
```

9.7 Rad sa baferizovanim tokovima

Operacija sa tokovima značajno su sporije od operacija na relaciji procesor – radna memorija. Stoga, česte operacije upisa u tok i čitanja iz njega mogu usporiti izvršenje programa. *Baferizovanje* (eng. *buffering*) predstavlja tehniku poboljšanja performansi IO operacija po kojoj se upis i čitanje ne vrši direktno iz toka, nego iz bafera, dela radne memorije. Pri radu sa baferizovanim tokovima, naredba za upis ne rezultuje fizičkim upisom u tok, već u bafer. Tek kad se bafer napuni dolazi do upisa podataka iz bafera u tok. Operacija upisa u bafer se naziva *logička izlazna operacija*, dok se upis iz bafera u tok naziva *fizička izlazna operacija*. Slično, operacija čitanja podataka, tzv. *logička ulazna operacija*, ne vrši se direktno iz toka, nego iz bafera. Kad se bafer isprazni, sledećom *fizičkom ulaznom operacijom* se podaci učitavaju iz toka u bafer. Broj fizičkih operacija sa tokom je ovako značajno manji od IO zahteva iz programa, čime se poboljšavaju IO performanse programa.

Korišćenje bafera za smeštanje podataka prilikom IO operacija može značajno poboljšati efikasnost aplikacije jer se smanjuje broj operacija sa tokovima.

Klasa koja omogućava baferizovan upis u tok je `BufferedOutputStream`. Koristi se u kombinaciji sa drugim klasama za rad sa tokovima primenom koncepta omotavanja. Na primer, ako želimo da baferizujemo upis objekata u binarni fajl (videti primer iz poglavlja 9.5.3), to možemo izvršiti sa

```
FileOutputStream fos = new FileOutputStream("Radnici.ser");
BufferedOutputStream bos = new BufferedOutputStream(fos);
ObjectOutputStream izlaz = new ObjectOutputStream(bos);
```

ili

```
ObjectOutputStream izlaz = new ObjectOutputStream(
                               new BufferedOutputStream(
                                  new FileOutputStream("Radnici.ser")));
```

Iz prethodnih naredbi vidimo da se koncept omotavanja može primeniti rekurzivno na više od dve klase. Ostatak programa, koji vrši upis objekata u binarni fajl, se ne menja. Delimično popunjen bafer može se isprazniti metodom `flush` klase `BufferedOutputStream`.

Klasa `BufferedInputStream` omogućava baferizovano čitanje iz toka. Na primer, baferizovano čitanje radnika iz binarnog fajla (primer iz poglavlja 9.5.4) se omogućava kreiranjem `ObjectInputStream` objekta na sledeći način:

```
ObjectInputStream ulaz = new ObjectInputStream(
                             new BufferedInputStream(
                                 new FileInputStream("Radnici.ser")));
```

Klase `BufferedOutputStream` i `BufferedInputStream` su izvedene iz klasa `FilterOutputStream` i `FilterInputStream`, respektivno, a ove iz klasa `OutputStream` i `InputStream`, respektivno. Klasa `PrintStream` (izvedena iz `FilterOutputStream`)

```
    catch(FileNotFoundException e) {
      System.err.println("Greška pri otvaranju.");
      System.exit(1);
    }
  }

  public static void otvoriFajlBaferizovano() throws IOException {
    try {
      ulaz = new ObjectInputStream(
                 new BufferedInputStream(
                     new FileInputStream("Radnici.ser")));
    }
    catch(FileNotFoundException e) {
      System.err.println("Greška pri otvaranju.");
      System.exit(1);
    }
  }

  public static void citajRadnikeIzFajla() throws IOException {
    try {
      while(true) {
        Radnik r = (Radnik) ulaz.readObject();
      }
    }
    catch(ClassNotFoundException | EOFException e) {
    }
  }

  public static void zatvoriFajl() throws IOException {
    if(ulaz != null)
      ulaz.close();
  }
}
```

Prvo izvršenje (nebaferizovano):

```
Vreme izvršenja: 5.477 [s]
```

Drugo izvršenje (baferizovano):

```
Vreme izvršenja: 0.753 [s]
```

Ubrzanje koje pruža baferizovani pristup je oko 7 puta!

Baferizovano čitanje radnika smo omogućili kreiranjem `ObjectInputStream` objekta na način opisan u prethodnom poglavlju.

Za merenje vremena izvršenja programa, koristili smo metodu `currentTimeMillis` klase `System`, koja vraća vreme, mereno u milisekundama, proteklo od 1.1.1970. Razlika promenljivih `pocetak` i `kraj`, podeljena sa 1000, nam daje vreme izvršenja programa u sekundama.