DIZAJN I RAZVOJ SOFTVERA

Tokovi, fajlovi i serijalizacija objekata

Pojam toka

- Svi podaci smješteni u Java promjenljivim, odnosno instancama klasa, su privremeni, tj. postoje u memoriji računara dok se program izvršava. Za dugoročno čuvanje podataka (nakon prestanka rada programa) računari koriste fajlove.
- Fajlovi se čuvaju na sekundarnim uređajima za skladištenje, kao što su hard diskovi, fleš diskovi, CD i DVD medijumi itd.
- Java svaki fajl vidi kao sekvencijalni niz bajtova. Kad se fajl otvori, dodjeljuje 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 se predstavlja sa 4 bajta, karakter `'#'` sa 2 bajta).
- 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 čovjeka, dok binarni fajlovi nisu.

Pojam toka

- Svaki operativni sistem obezbjeđuje 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 određuje kraj fajla. U programu koji radi sa fajlovima se, pomoću odgovarajućih funkcija, jednostavno dobija indikacija od operativnog sistema da smo došli do kraja fajla (npr. C funkcija `feof` za parametar ima pokazivač na fajl i vraća nenultu vrijednost (logička istina) kad dođemo do kraja 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`.
- `Sistem.in` (standardni ulazni tok) omogućava čitanje podataka unesenih preko tastature. `Sistem.out` (standardni izlazni tok) omogućava štampanje karaktera na ekranu. `Sistem.err` (standardni tok greške) omogućava štampanje poruka greške na ekranu.

Paketi java.io i java.nio

- 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 pomjerati proizvoljan broj bajtova unaprijed ili unazad u toku. Da bi to postigli, podaci se prvo moraju smjestiti u bafer.
- Sa druge strane, kod NIO pristupa, podaci se prvo smještaju u bafer, odakle se dalje procesiraju. Možemo se kretati unaprijed 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 cijene - moramo voditi računa o podacima u baferu. Na primjer, prilikom učitavanja novih podataka u bafer, moramo voditi računa da ne izgubimo podatke iz bafera koje nismo obradili.

Paketi java.io i java.nio

- Druga bitna razlika između IO i NIO je da je veliki broj IO tokova blokirajući - prilikom izvršenja read ili write operacija, blokira se 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. Umjesto 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 primjer, 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 obraditi neke klase iz paketa `java.io`: `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 karaktera) i `FileWriter` (za upis podataka u fajl na nivou karaktera).
- Klase `ObjectInputStream` i `ObjectOutputStream` omogućavaju upis i čitanje čitavih objekata u tokove (tzv. serijalizacija), čime nas izdižu na viši nivo apstrakcije od bajtova i karaktera.
- Klase za baferizovan rad sa tokovima (`BufferedReader`, `BufferedWritter`, `BufferedInputStream`, `BufferedOutputStream`) poboljšavaju performanse upisa i čitanja.

Rad sa fajlovima i folderima. Klasa File

- Klasa **`File`** (paket `java.io`) pruža niz korisnih metoda u radu sa fajlovima i folderima (direktorijumima), od kojih je jedan broj dat u tabeli ispod.

Metoda	Opis
`exists`	Vraća `true` ako fajl ili folder predstavljen `File` objektom postoji, `false` u suprotnom.
`isFile`	Vraća `true` ako ime prosllijeđeno `File` konstruktoru odgovara fajlu, `false` u suprotnom.
`isDirectory`	Vraća `true` ako ime prosllijeđeno `File` konstruktoru odgovara folderu, `false` u suprotnom.
`isAbsolute`	Vraća `true` ako argumenti prosllijeđeni i `File` konstruktoru odgovaraju apsolutnoj putanji do fajla ili foldera, `false` u suprotnom.
`getAbsolutePath`	Vraća `String` sa apsolutnom putanjom do fajla ili foldera.
`getName`	Vraća `String` sa imenom fajla ili foldera.
`getPath`	Vraća `String` sa putanjom do fajla ili foldera.
`getParent`	Vraća `String` sa roditeljskim (prvim iznad) folderom datog fajla ili foldera.
`length`	Vraća veličinu fajla, izraženu u bajtovima. Ako `File` objekat predstavlja folder, vraća se nedefinisana vrijednost.
`lastModified`	Vraća vrijeme u `long` formatu (vrijednost zavisi od platforme) posljednje modifikacije fajla ili foldera. Vraćena vrijednost se može koristiti samo za poređenje sa drugim vrijednostima koje vraća ova metoda.
`delete`	Briše predmetni fajl ili folder. U slučaju brisanja foldera, njegov sadržaj se mora izbrisati pre brisanja samog foldera. Vraća `true` ako je brisanje uspješno izvršeno i `false` u suprotnom.
`list`	Vraća niz `String` objekata koji predstavlja sadržaj foldera, odnosno `null` ako `File` objekat ne predstavlja folder.

Klasa File

- Koristeći objekte klase `File` ne možemo otvoriti fajlove, niti ih procesirati na bilo koji način. Ovi objekti se obično koriste u kombinaciji sa drugim `java.io` klasama da specificiraju sa kojim fajlovima i folderima se konkretno radi.
- `File` klasa ima četiri konstruktora.
- Pomenimo ovdje samo konstruktor sa parametrom stringom koji određuje ime fajla ili foldera koje će biti pridruženo `File` objektu. Ime može sadržati i putanju do fajla/foldera. Putanja može uključiti samo neke ili sve foldere do željenog počev od korijenog foldera (npr. `"C:\Prvi\Drugi\Treci"`). Putanja koja sadrži sve foldere počev od korijenog foldera se naziva apsolutna putanja. Relativna putanja se računa u odnosu na tekući folder, tj. onaj u kom je aplikacija počela da se izvršava.
- U nastavku dajemo primjer korišćenja klase `File`. Od korisnika ćemo tražiti unos putanje do fajla ili foldera, a nakon toga, ukoliko predmetni fajl/folder postoji, odštampaćemo nekoliko opisnih poruka vezanih za fajl/folder. Dodatno, ako je u pitanju folder, odštampaćemo i njegov sadržaj.

Klasa File - Primjer

```
import java.util.Scanner;
import java.io.File;

public class DemonstracijaKlaseFile {

   public static void main(String [] args) {
      Scanner unos = new Scanner(System.in);

      System.out.print("Unijeti ime fajla ili foldera: ");
      File ime = new File(unos.nextLine());
      if(ime.exists()) {
         System.out.printf("%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",
                           "Posljednja izmejna: " + ime.lastModified(),
                           "Roditeljski folder: " + ime.getParent());
      }
      else
         System.out.printf("%s ne postoji", ime);

      if(ime.isDirectory()) {
         System.out.println("\n\nSadržaj foldera:");
         for(String f: ime.list())
            System.out.println(f);
      }

      unos.close();
   }
}
```

Klasa File - Primjer

Izvršenje 1

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

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

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

Izvršenje 2

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

Tekstualni fajlovi sa sekvencijalnim pristupom

- Demonstrirajmo rad sa tekstualnim fajlovima sa sekvencijalnim pristupom.
- Sekvencijalni pristup podrazumijeva da se podaci upisuju i čitaju jedan za drugim. Nasuprot sekvencijalnom imamo slučajni pristup, gdje bilo kad možemo pristupiti proizvoljnom podatku u toku.
- U prvom primjeru ćemo kreirati tekstualni fajl na osnovu unosa korisnika.
- U drugom primjeru ćemo otvoriti kreirani fajl i odštampati njegov sadržaj.
- Pomenute radnje ćemo demonstrirati pomoću novokreirane klase `Radnik`, prikazane na sljedećem slajdu.

Klasa Radnik

`LocalDate` klasa predstavlja datume, bez vremenske zone, u ISO-8601 kalendarskom sistemu, npr. 2021-04-23.

Klasa `DateTimeFormatter` se koristi za formatiranje datuma. Metoda `ofPattern` ove klase kreira formater datuma na osnovu stringa šablona. Šablon je kombinacija predefinisanih slova i simbola za prikaz datuma. Na primjer, šablon `"d MMM uuuu"` će formatirati datum `2021-04-23` kao `"23 Apr 2021"`.

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

public class Radnik {
   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) { 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()));
   }
}
```

Kreiranje txt fajla i upis u fajl

```
import java.util.*;
import java.io.*;
import java.time.LocalDate;

public class UpisRadnikaUFajl {

   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() {
      if(izlaz != null)
         izlaz.close();
   }
}
```

Instanca klase `Formatter` omogućava upis formatiranih podataka u tekstualni tok

Kreiranje instance klase `LocalDate` na osnovu učitanog stringa

Upis formatiranog teksta u fajl, isto kako metoda `System.out.printf` štampa tekst u konzoli

Kreiranje txt fajla i upis u fajl

```
public class TestiranjeUpisaUFajl {

   public static void main(String[] args) {
      UpisRadnikaUFajl fajlOper = new UpisRadnikaUFajl();
      fajlOper.otvoriFajl();
      fajlOper.upisiRadnikeUFajl();
      fajlOper.zatvoriFajl();
   }

}
```

Izvršenje

```
Koliko radnika želite da unesete? 4
Unesite radnike u formatu: <ime> <prezime> <datum> <sprema>
(<datum> unesite u formatu yyyy-mm-dd)
Dalibor Ilić 2019-02-22 4
Marijana Petrović 2018-04-04 5
Damir Zvrko 2019-11-04 3
Azra Bulajić 2018-01-30 4
```

Kreiranje txt fajla i upis u fajl. Klasa Formatter

- Pored klasa iz `java.io` paketa, upis i čitanje na nivou karaktera se može vršiti pomoću klasa `Formatter` i `Scanner`, što je slučaj u ovom primjeru.
- Klasu `Scanner` smo ranije koristili, za učitavanje karaktera sa tastature. Pored tastature, za koju je vezan *standardni ulazni tok*, klasu `Scanner` možemo koristiti i za čitanje podataka iz fajla.
- Vrlo korisna je i klasa `Formatter` koja omogućava upis formatiranih podataka u bilo koji tekstualni tok, slično metodi `System.out.printf`.
- U metodi `otvoriFajl` kreiramo instancu klase `Formatter` sa:

```
izlaz = new Formatter("Radnici.txt", "UTF-8");
```

- Konstruktor (ima ih 17) dobija dva stringa: ime fajla i ime skupa karaktera.
- Konstruktor kreira fajl ukoliko fajl ne postoji, odnosno briše sadržaj postojećeg fajla.
- Ovaj konstruktor baca provjerene izuzetke tipa `FileNotFoundException` i `UnsupportedEncodingException`, koje smo obradili u odgovarajućim `catch` blokovima. U slučaju oba izuzetka, prekidamo izvršenje programa pozivom statičke metode `exit` klase `System`. Nulta vrijednost `exit` argumenta označava da je program uspješno završio izvršavanje, nenulta da je došlo do neke greške prilikom izvršavanja.

Kreiranje txt fajla i upis u fajl

- U metodi `upisiRadnikeUFajl` vršimo unos pojedinačnih radnika pomoću klase `Scanner`. Prvo unosimo broj radnika, a zatim radnike. Prije unosa radnika, prikazujemo poruku o formatu datuma (podatak `pocetak` klase `Radnik`) koji očekuje klasa `LocalDate`.
- Datum učitavamo kao string, a pomoću metode `parse` klase `LocalDate` kreiramo instancu ove 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.
- Metoda `format` baca neprovjerene izuzetke tipa `IllegalFormatException` i `FormatterClosedException`, koje nismo obradili u ovom primjeru. Prvi se dešava ukoliko je sintaksa format stringa neispravna, ako specifikator formata nije kompatibilan sa proslijeđenim argumentima ili ako nije proslijeđen dovoljan broj argumenata format stringu. Drugi izuzetak se dešava u slučaju da je `Formatter` instanca već zatvorena metodom `close`.

Čitanje podataka iz txt fajla

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

public class CitanjeRadnikaIzFajla {

   private static Scanner ulaz;

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

   public static void otvoriFajl() {
      try { ulaz = new Scanner(new File("Radnici.txt")); }
      catch(FileNotFoundException e) { System.err.println("Greška pri otvaranju."); System.exit(1); }
   }

   public static void citajRadnikeIzFajla() {
      Radnik r = new Radnik();

      while(ulaz.hasNext()) {
         r.setIme(ulaz.next());
         r.setPrezime(ulaz.next());
         r.setPocetak(LocalDate.parse(ulaz.next()));
         r.setSprema(ulaz.nextInt());

         System.out.println(r);
      }
   }
```

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

Predmetnu klasu koristimo za testiranje čitanja iz fajla

Čitanje podataka iz txt fajla

- U metodi `otvoriFajl`, kreiramo instancu `Scanner` objekta `ulaz`, kojoj za argument prosljeđujemo `File` objekat. Dakle, `Scanner` objekat koristimo za učitavanje podataka iz fajla. Do sad smo `Scanner` objektu prosljeđivali `System.in`, čime smo podatke učitavali sa tastature.
- Konstruktor klase `Scanner`, sa `File` objektom kao parametrom, baca provjereni izuzetak tipa `FileNotFoundException`, koji smo obradili u metodi `otvoriFajl`.
- U metodi `citajRadnikeIzFajla`, vršimo učitavanje podataka iz fajla u `while` petlji, koja se izvršava sve dok postoje podaci za učitavanje, tj. sve dok metoda `ulaz.hasNext()` vraća `true`. Pretpostavljamo da znamo strukturu tekstualnog fajla, tj. da su u svakom redu upisana četiri podatka: ime (`String`), prezime (`String`), datum početka (`String`) i sprema radnika (`int`).
- Metodom `parse` kreiramo instancu klase `LocalDate` na osnovu učitanog stringa. Nakon kreiranja tekuće instance klase `Radnik`, štampamo je metodom `println`.

Dopisivanje podataka u txt fajl

```
import ...;

public class DopisivanjeRadnikaUFajl {

   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();
   }
}
```

append parametar omogućava dopisivanje

Instanca klase FileWriter omogućava dopisivanje podataka u tekstualni tok

Dopisivanje formatiranog teksta u fajl

Dopisivanje podataka u txt fajl

- Za dopisivanje u postojeći tekstualni fajl, koristićemo `FileWriter` klasu, koja omogućava upis podataka u fajl na nivou karaktera.
- Klasa `FileWriter` ima 9 konstruktora. Koristimo verziju sa tri parametra: imenom fajla (`String`), kôdnim skupom karaktera (`Charset`) i parametrom koji definiše dopisivanje (`boolean`). Za skup karaktera koristimo UTF-8, dobijen statičkim podatkom `UTF_8` klase `StandardCharsets` (paket `java.nio.charset`). Parametar za dopisivanje ima vrijednost `true`, što omogućava dopisivanje u postojeći fajl.
- Dopisivanje podataka u fajl vršimo metodom `write` klase `FileWriter`, koja za argument ima `String` podatak. U pitanju je naslijeđena metoda iz apstraktne klase `Writer`, koja služi za upisivanje u tokove karaktera. Ovoj metodi prosljeđujemo red teksta, koji predstavlja jednog radnika, dobijen metodom `String.format`. Red je formatiran na isti način kao redovi teksta upisani nakon kreiranja fajla.
- Korišćeni konstruktor klase `FileWriter`, kao i metode `write` i `close` ove klase, bacaju izuzetak tipa `IOException`. Ove izuzetke nismo obrađivali u metodama gdje se pojavljuju, već smo proglasili da date metode bacaju izuzetak tog tipa. U okviru `try` bloka metode `main` smo obradili taj izuzetak ispisom odgovarajuće poruke i prekidom izvršenja programa.

Serijalizacija objekata

- Prilikom upisa podataka u tekstualni fajl, dolazi do gubitka informacija o tipu upisanih vrijednosti. Na primjer, prilikom upisa datuma, gubimo informaciju o klasi koja modeluje datum u Java programu.
- Klasa `LocalDate` koju smo 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 cijela broja razdvojena karakterom `'-'`. Tekstualni fajl sadrži vrijednost, ali ne i tip podatka.
- Za upis i čitanje čitavih objekata u tokove, Java obezbjeđuje serijalizaciju objekata!
- Serijalizovan objekat je objekat predstavljen sekvencom bajtova koja uključuje kako podatke objekta, tako i informaciju o tipu objekta i tipu podataka smještenih u tom objektu (prisjetimo se pojma kompozicije u OO programiranju).
- Nakon što je serijalizovani objekat smješten u tok, on se iz toka može pročitati i deserijalizovati, tj. na osnovu informacija o tipu objekta i vrijednosti i tipu njegovih podataka, možemo kreirati instancu predmetne 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. Da bi za tok koristili fajl, potrebno je da `ObjectOutputStream` i `ObjectInputStream` objekte 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 gdje treba da upisuju ili odakle da čitaju bajtove. Tu informaciju dobijaju od objekta toka koji se prosljeđuje njihovom konstruktoru.
- Preciznije, objekat toka koji prosljeđujemo `ObjectOutputStream` konstruktoru preuzima bajt-reprezentaciju objekta koji kreira `ObjectOutputStream` i upisuje te bajtove u tok (fajl, mrežnu konekciju itd.) Sa druge strane, objekat toka koji prosljeđujemo `ObjectInputStream` konstruktoru daje bajt-reprezentaciju objekta `ObjectInputStream`-u koju ovaj konvertuje u odgovarajući objekat.

Interfejs Serializable

- Za ilustraciju serijalizacije, koristićemo ranije kreiranu klasu `Radnik`. Da bi se instance ove klase mogle serijalizovati, potrebno je da klasa implementira interfejs `Serializable`.
- U pitanju je tagging interfejs ili prazni interfejs, koji ne sadrži nijednu metodu ni konstantu. Ovi interfejsi se koriste da klasi dodaju vezu tipa jeste.
- `ObjectOutputStream` neće upisati objekat u tok ukoliko objekat nije `Serializable`.
- U `Serializable` klasi, svaki objekat podatak mora biti `Serializable`. Ukoliko neki od objekata podataka nije `Serializable`, to se mora naglasiti ključnom rečju `transient` kako bi se ti podaci ignorisali tokom serijalizacije.
- Podrazumijevano su svi primitivni podaci `Serializable`. Za referencijske tipove je potrebno provjeriti u dokumentaciji da li su `Serializable` (ukoliko nisu, treba provjeriti da li su im superklase `Serializable`).

Podatak serialVersionUID

- Prilikom serijalizacije, Javino izvršno okruženje dodjeljuje svakoj `Serializable` klasi broj verzije, `static final long` podatak `serialVersionUID`, koji se koristi za provjeru da li se serijalizacija i deserijalizacija sprovode sa istom verzijom klase.
- Vrijednost `serialVersionUID` je smještena zajedno sa ostalim podacima objekta tokom serijalizacije. Ukoliko primalac i pošiljalac serijalizovanog objekta koriste verzije `Serializable` klase sa različitim `serialVersionUID`, dolazi do izuzetka `InvalidClassException`. U klasi se ovaj podatak može deklarisati eksplicitno, npr:

```
private static final long serialVersionUID = 15L;
```

- Kod eksplicitne deklaracije, obično je vrijednost prve verzije naše `Serializable` klase `1L`, i tu vrijednost povećavamo za 1 pri svakoj strukturnoj promjeni klase.
- Ukoliko ne izvršimo eksplicitnu deklaraciju `serialVersionUID` podatka, izvršno okruženje će izračunati njegovu podrazumijevanu vrijednost na osnovu raznih aspekata klase, kako je objašnjeno u specifikaciji serijalizacije Java objekata. Ipak, preporučuje se eksplicitna deklaracija jer je podrazumijevana vrijednost ovog podatka vrlo osetljiva na detalje klase koji mogu varirati zavisno od implementacije kompajlera, što može rezultovati `InvalidClassExceptions` izuzetkom tokom deserijalizacije. Takođe, preporučuje se korišćenje `private` modifikatora pristupa za podatak `serialVersionUID`, čime se on ne bi nasljeđivao.

Serializable klasa Radnik

- Izmijenjena klasa `Radnik`, u skladu sa potrebom serijalizacije, je data u nastavku (izmjene u odnosu na originalnu klasu su označene crvenom bojom teksta).

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

public class Radnik implements Serializable {

   private static final long serialVersionUID = 1L;
   private String ime;
   private String prezime;
   private LocalDate pocetak;
   private int sprema;
   private static final DateTimeFormatter df = DateTimeFormatter.ofPattern("dd/MM/yyyy");

   ... // ostalo je isto
}
```

Kreiranje fajla korišćenjem serijalizacije

```
import ...;

public class UpisRadnikaUFajl {

   private static ObjectOutputStream izlaz;

   public void otvoriFajl() {
      try {
         FileOutputStream fos = new FileOutputStream("Radnici.ser");
         izlaz = new ObjectOutputStream(fos); }
      catch(IOException e) { System.err.println("Greška pri otvaranju fajla."); System.exit(1); }
   }

   public void upisiRadnikeUFajl() {
      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++) {
         Radnik r = new Radnik();
         r.setIme(unos.next());
         r.setPrezime(unos.next());
         r.setPocetak(LocalDate.parse(unos.next()));
         r.setSprema(unos.nextInt());
         try { izlaz.writeObject(r);}
         catch(IOException e) { System.err.println("Greška pri upisu u fajl."); }
      }
      unos.close();
   }
```

```
   public static void zatvoriFajl() {
      try { if(izlaz != null) izlaz.close(); }
      catch (IOException e) { System.err.println("Greška"); }
   }
}
```

Otvaranje binarnog fajla za upis

Omotavanje

Upis serijalizovanog objekta u tok

Kreiranje fajla korišćenjem serijalizacije

```
public class TestiranjeUpisaUFajl {
   public static void main(String[] args) {
      UpisRadnikaUFajl fajlOper = new UpisRadnikaUFajl();
      fajlOper.otvoriFajl();
      fajlOper.upisiRadnikeUFajl();
      fajlOper.zatvoriFajl();
   }
}
```

Izvršenje

```
Koliko radnika želite da unesete? 3
Unesite radnike u formatu: <ime> <prezime> <datum> <sprema>
(<datum> unesite u formatu yyyy-mm-dd)
Raša Popov 2016-04-04 4
Anja Raičević 2019-04-21 6
Nino Taljanović 2018-10-18 5
```

Kreiranje fajla korišćenjem serijalizacije

- U metodi `otvoriFajl` kreiramo binarni fajl Radnici.ser naredbom

```
FileOutputStream fos = new FileOutputStream("Radnici.ser");
```

- Pošto je u pitanju klasa `FileOutputStream`, fajl se otvara za upis. Verzija konstruktora ove klase koju koristimo ima jedan parametar, string koji definiše ime (opciono i putanju do) fajla koji otvaramo. Postoji i konstruktor sa dodatnim `boolean` parametrom `append` koji omogućava otvaranje fajla za dopisivanje. Izbor ekstenzije .ser je proizvoljan i ovdje se koristi samo da bi asocirao da dati binarni fajl sadrži serijalizovane objekte.
- `FileOutputStream` objekat se proslijeđuje `ObjectOutputStream` konstruktoru (prisjetimo se da se ovo naziva omotavanje) u naredbi

```
izlaz = new ObjectOutputStream(fos);
```

- Objekat `izlaz` omogućava serijalizaciju `Radnik` objekata, tj. njihov upis u fajl Radnici.ser, što je izvršeno naredbom `izlaz.writeObject(r)` u `try` bloku metode `upisiRadnikeUFajl`.
- Sve operacije sa `ObjectOutputStream` objektom u našem primjeru bacaju izuzetak tipa `IOException`, koji je obrađen u metodama klase.
- Binarni fajlovi nisu čitljivi za čovjeka, pa ih nema smisla otvarati u tekst editoru.

Čitanje podataka iz binarnog fajla

```
import ...;

public class CitanjeRadnikaIzFajla {

   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
```

Izvršenje

Čitanje podataka iz 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 prosljeđuje `ObjectInputStream` konstruktoru u naredbi

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

- Objekat `ulaz` omogućava deserijalizaciju `Radnik` objekata, 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 downcast-ovati prije upisa u `Radnik` promjenljivu.

- 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 `EOFException` izuzetak. Metoda `readObject` takođe baca i ClassNotFoundException ukoliko se klasa objekata koje čitamo ne može locirati.
- Sve operacije sa `ObjectInputStream` objektom u našem primjeru bacaju izuzetak tipa `IOException`. Ove izuzetke smo obradili u `try` naredbi metode `main`.

Rad sa baferizovanim tokovima

- Operacija sa tokovima su značajno 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, dijela 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, se ne vrši direktno iz toka, nego iz bafera. Kad se bafer isprazni, sljedećom fizičkom ulaznom operacijom se podaci učitavaju iz toka u bafer.
- Broj fizičkih operacija sa tokom je ovako značajno manji od IO zahtjeva iz programa, čime se poboljšavaju IO performanse programa.

Rad sa baferizovanim tokovima

- Klasa koja omogućava baferizovan upis u tok je `BufferedOutputStream`. Koristi se u kombinaciji sa drugim klasama za rad sa tokovima primjenom koncepta omotavanja. Na primjer, ako želimo da baferizujemo upis objekata u binarni fajl, to možemo izvršiti sa:

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

dok se nebaferizovan upis vršio sa:

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

- U ostatku programa koji vrši upis objekata u binarni fajl, ništa se ne mijenja.
- Klasa `BufferedInputStream` omogućava baferizovano čitanje iz toka. Na primjer, baferizovano čitanje radnika iz binarnog fajla se omogućava kreiranjem `ObjectInputStream` objekta na sledeći način:

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

Dodatne pogodnosti baferizovanog čitanja

```
import ...;

public class BaferizovanoCitanjeRadnikaIzFajla {

   private static BufferedReader 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 {
      ulaz = new BufferedReader(new FileReader("Radnici.txt", StandardCharsets.UTF_8));
   }

   public static void citajRadnikeIzFajla() throws IOException {
      Radnik r = new Radnik();
      String linija;
      Scanner skener;

      while((linija = ulaz.readLine()) != null) {
         skener = new Scanner(linija);
         r.setIme(skener.next());
         r.setPrezime(skener.next());
         r.setPocetak(LocalDate.parse(skener.next()));
         r.setSprema(skener.nextInt());
         System.out.println(r);
         skener.close();
      }
   }
```

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

Kreiranje `BufferedReader` instance na osnovu `FileReader` instance

Čitamo liniju po liniju

Liniju teksta dajemo `Scanner` konstruktoru da bismo mogli izdvojiti potrebne podatke

Dodatne pogodnosti baferizovanog čitanja

- Klasa `FileReader` omogućava čitanje sadržaja tekstualnog fajla isključivo na nivou karaktera. Metoda `read`, naslijeđena iz klase `InputStreamReader`, vraća pročitani karakter iz toka (prva verzija metode) ili upisuje pročitane karaktere u niz (druga verzija metode). Na programeru je da obezbijedi logiku kojom se tumače pročitani karakteri.
- Klasa `BufferedReader` dodaje metodu `readLine`, koja čita liniju teksta iz fajla.
- Naredba

```
ulaz = new BufferedReader(
            new FileReader("Radnici.txt", StandardCharsets.UTF_8));
```

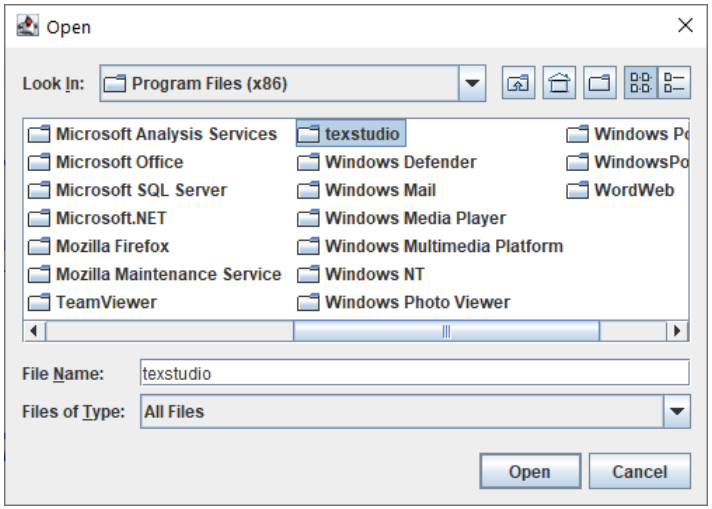
  kreira `BufferedReader` instancu na osnovu `FileReader` instance.
- Klasa `FileReader` ima pet konstruktora, a ovdje koristimo verziju sa dva parametra: imenom fajla (`String`) i kôdnim skupom karaktera (`Charset`). Za kôdni skup karaktera koristimo UTF-8. Klasa `BufferedReader` ima dva konstruktora. Ovdje koristimo konstruktor sa parametrom `Reader`. Drugi konstruktor ima dodatni parametar – veličinu bafera, koja ima podrazumijevanu vrijednost kod prvog konstruktora.
- U metodi `citajRadnikeIzFajla`, u `while` petlji čitamo red po red (radnika po radnika), Da bismo iz tekućeg reda izdvojili ime, prezime, datum početka i spremu radnika, kreiramo `Scanner` instancu na osnovu tekućeg reda teksta i pomoću njenih metoda `next` i `nextInt` dobijamo ove podatke.

Odabir fajlova i foldera pomoću JFileChooser-a

```
import ...;

public class DemonstracijaJFileChooser {

   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",
                                    "Posljednja izmjena: " + ime.lastModified(),
                                    "Roditeljski folder: " + ime.getParent());
      else ispis = String.format("%s ne postoji", ime);

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

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

   private static File vratiFajlIliFolder() {
      JFileChooser fc = new JFileChooser("C:\\Program Files (x86)\\");
      fc.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
      int result = fc.showOpenDialog(null);
      if (result == JFileChooser.CANCEL_OPTION)  System.exit(1);

      File ime = fc.getSelectedFile();
      if(ime == null || ime.getName().equals("")) {
         JOptionPane.showMessageDialog(null, "Pogrešno ime fajla ili foldera",
                                       "Pogrešno ime", JOptionPane.ERROR_MESSAGE);
         System.exit(1);
      }
      return ime;
   }
}
```

Odabir fajlova i foldera pomoću JFileChooser-a

- Odabir fajla ili foldera vršimo u metodi `vratiFajlIliFolder`. Konstruktoru `JFileChooser` klase prosljeđujemo string sa putanjom do foldera čiji će sadržaj biti prikazan u `Open` prozoru. Primjer `Open` prozora je prikazan na slici ispod.
- U metodi `setFileSelectionMode` određujemo da `JFileChooser` objekat prikaže fajlove i foldere (statička konstanta `FILES_AND_DIRECTORIES`). Za prikaz samo fajlova ili samo foldera, naveli bi konstante `FILES_ONLY` i `DIRECTORIES_ONLY`.
- Metoda `showOpenDialog` prikazuje `JFileChooser` prozor. Parametar ove metode je roditeljska komponenta (prozor) u odnosu na koju će biti prikazan `JFileChooser` prozor. U našem slučaju, vrijednost `null` označava da će prozor biti prikazan na sredini ekrana.



- Metoda `getSelectedFile` klase `JFileChooser` vraća odabrani fajl/folder. Ukoliko je ime korektno, vraćamo ga nazad pozivajućoj metodi. Ako nije korektno (`null` ili prazan string), prekidamo izvršenje fajla uz prikaz prozora sa porukom greške.

Metoda showMessageDialog klase JOptionPane

- Metoda `showMessageDialog` klase `JOptionPane` prikazuje prozor sa tekstualnom porukom. Prvi parametar ove metode je roditeljska komponenta u odnosu na koju će biti prikazan prozor (`null` označava da će prozor biti prikazan na sredini ekrana). Drugi i treći parametar su tekst poruke i naziv prozora, respektivno, dok je četvrti tip poruke.
- Mogući tipovi poruke su `JOptionPane` konstante `ERROR_MESSAGE`, `WARNING_MESSAGE`, `INFORMATION_MESSAGE`, `QUESTION_MESSAGE` i `PLAIN_MESSAGE`.
- Primjer prozora sa tekstom poruke koja sadrži podatke o odabranom folderu i listing njegovog sadržaja je dat na slici desno.