Datoteke u Javi

Sadržaj

Uvod u rad s datotekama

Vrste tokova podataka u Javi

Klasa **java.io.File**

Sučelja i klase iz paketa **java.nio.file**

Klasa **Formatter**

Tokovi za čitanje i zapisivanje podataka u datoteke

Klase koje predstavljaju tokove podataka u Javi

Čitanje i zapisivanje podataka u binarne datoteke

Blok „try-with-resources”

Unapređenja bloka „try-with-resources” u Javi 9

Korištenje *streamova* u radu s datotekama

Serijalizacija i deserijalizacija u Javi

Uvod u rad s datotekama

- Datoteke se koriste za trajnije pohranjivanje podataka ili dohvat informacija iz njih koje su postojale i prije pokretanja programa
- Postoje dvije vrste datoteka: tekstualne i binarne
- Iz perspektive Java programa svaka datoteka može se razmatrati kao **niz bajtova** (engl. *stream of bytes*)
- Svaka datoteka na kraju ima „end-of-file” oznaku kojom se detektira kraj datoteke nakon čega je potrebno zaustaviti proces čitanja njenog sadržaja
- U datoteke je moguće spremati i cijele objekte postupkom koji se naziva **serijalizacija**, a čitanje podataka o objektima se naziva **deserijalizacija**

Vrste tokova podataka u Javi

- Tokovi podataka pomoću kojih se dohvaćaju i zapisuju podaci u datoteke mogu biti temeljeni na:
  - **Binarnom formatu**: ulazni i izlazni tok su predstavljeni bajtovima (char ima 2 bajta, int 4 bajta, double 8 bajtova itd.) – koristi se kod binarnih datoteka
  - **Znakovnom formatu**: ulazni i izlazni tok predstavljen je nizom znakova u kojem svaki znak ima 2 bajta – koristi se kod tekstualnih datoteka
- Prilikom otvaranja datoteka potrebno je kreirati pripadajući objekt čiji konstruktor komunicira s operacijskim sustavom
- U Javi postoje tri standardna toka podataka: **System.in** (za učitavanje podataka s tipkovnice), **System.out** (za ispis podataka na zaslon) i **System.err** (za ispis podataka o pogreškama na zaslon)

Klasa **java.io.File**

- Predstavlja element datotečnog sustava (može biti datoteka ili mapa)
- Sadrži putanju, ime i veličinu datoteke, ali ne i sam sadržaj
- Može kreirati datoteku na sljedeće načine:

```
File wf1 = new File("moj.htm");
File wf2 = new File("java\\html\\IO.html");
File wf3 = new File("D:\\java\\25.txt");
```

- Sadrži niz korisnih metoda koje provjeravaju postoji li datoteka (**exists**), je li datoteka ili mapa (**isFile** i **isDirectory**), za dohvat naziva i putanje (**getName** ili **getPath**) itd.

Sučelja i klase iz paketa **java.nio.file**

- Omogućavaju dohvat podataka iz datoteka pomoću sljedećih sučelja i klasa (uvedeni od Jave SE 6):
  - **Sučelje Path**: objekti klasa koji implementiraju to sučelje predstavljaju lokaciju datoteka ili mape (engl. *folder*), ali ne omogućavaju čitanje sadržaja datoteke
  - **Klasa Paths**: sadrži statičke metode za dohvat **Path** objekata koji predstavljaju datoteku ili mapu
  - **Klasa Files**: sadrži statičke metode za rad s datotekama ili mapama, kao što je kopiranje datoteka, kreiranje ili brisanje mapa, dohvaćanje informacija o datotekama, čitanje i promjena sadržaja datoteka itd.
  - **Sučelje DirectoryStream**: objekti klasa koji implementiraju to sučelje mogu koristiti sadržaj datoteke

Klasa **Formatter**

- Služi za oblikovanje teksta koji se može zapisivati u datoteku, po principu korištenja metode **printf**
- Objekt se kreira na sljedeći način:

```
Formatter output = new Formatter("clients.txt");
```

- Zapisivanje u datoteku se obavlja na sljedeći način:

```
output.format("%d %s %s %.2f%n", cijeliBroj, string1, string2, double);
```

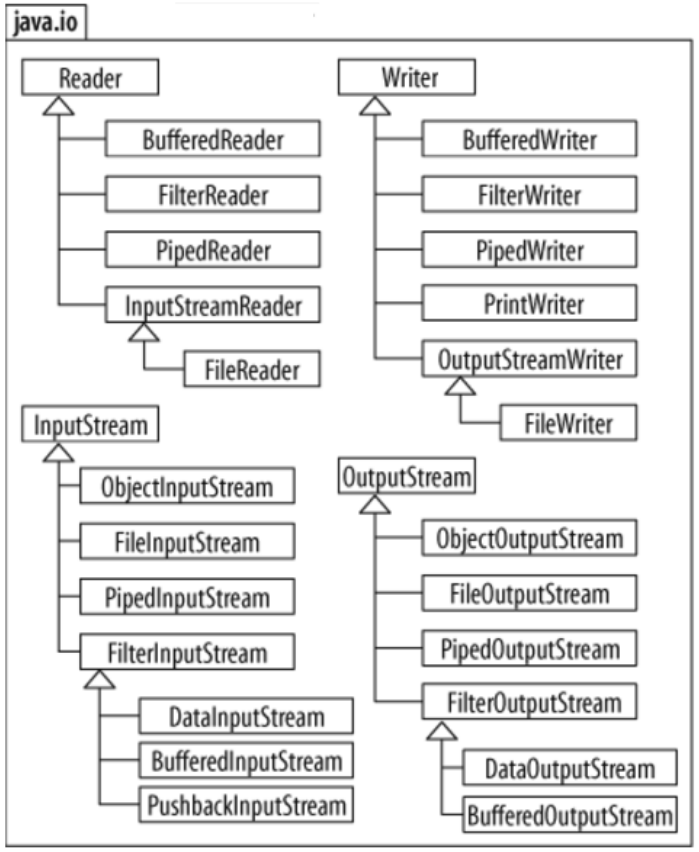
- U slučaju nedostatka prava za korištenje datoteke baca se iznimka SecurityException

Tokovi za čitanje i zapisivanje podataka u datoteke

- U Javi postoji nekoliko klasa koje predstavljaju tokove za razmjenu podataka s datotekama, a dijele se na dvije osnovne skupine:
  - Ulazni tokovi podataka
  - Izlazni tokovi podataka
- Osnovne apstraktne klase iz kojih su izvedene ostale klase za čitanje i zapisivanje toka bajtova za binarne datoteke su:
  - java.io.InputStream (šalje bajtove iz vanjskog izvora u Java program)
  - java.io.OutputStream (šalje bajtove iz Java programa u neko vanjsko odredište)
- Osnovne apstraktne klase iz kojih su izvedene ostale klase za čitanje i zapisivanje znakovnih tokova su:
  - java.io.Reader (prima znakove iz vanjskog izvora i šalje ih u Java program)
  - java.io.Writer (šalje znakove iz Java programa u vanjski izvor)

Klase koje predstavljaju tokove podataka u Javi

- Postoji niz klasa koje predstavljaju implementaciju apstraktnih klasa za čitanje i zapisivanje tokova podataka u Javi (**Reader**, **Writer**, **InputStream** i **OutputStream**):



Čitanje i zapisivanje podataka u binarne datoteke

- Osnovna metoda klase **InputStream** je

public abstract int read() throws IOException

- koja čita 1 bajt iz ulaznog toka, a vraća cjelobrojnu vrijednost tog bajta
- Metoda **read** blokira izvođenje ostatka programa (čeka) tako dugo dok se jedan bajt ne pročita
- Osnovna metoda klase **OutputStream** je

public abstract void write(int b) throws IOException

- koja šalje jedan bajt podataka preko izlaznog toka do odredišta koje taj znak interpretira na određeni način

Blok „try-with-resources”

- Svaku datoteku je nakon njenog korištenja potrebno zatvoriti pozivom metode „close” nad objektom koji predstavlja tok podataka
- Za automatiziranje tog procesa od Jave 7 uveden je poseban „try” blok koji automatski poziva metodu „close” i zove se „try-with-resources”, a izgleda ovako:

```
try (FileInputStream fis = new FileInputStream(FILENAME)) {
   for (int n = fis.read(); n != -1; n = fis.read()) {
      System.out.write(n);
   }
   System.out.flush();
}
catch (IOException ex) {
   System.err.println("Pogreška kod čitanja datoteke " + FILENAME);
   ex.printStackTrace();
}
```

Unapređenje bloka „try-with-resources” u Javi 9

- U Javi 9 je moguće koristiti blok „try-with-resources” na način da se objekti mogu definirati i prije bloka i samo se koristiti u njemu (pri čemu objekti moraju biti „final” ili „effectively final”):

```
FileInputStream fis = new FileInputStream(FILENAME);
try (fis) {
   for (int n = fis.read(); n != -1; n = fis.read()) {
      System.out.write(n);
   }
   System.out.flush();
}
catch (IOException ex) {
   System.err.println("Pogreška kod čitanja datoteke " + FILENAME);
   ex.printStackTrace();
}
```

Korištenje *streamova* u radu s datotekama

- Klasa „Files” sadrži i nekoliko metoda koje su vezane uz *streamove* i drastično olakšavaju rad s datotekama:
  - **lines(Path putanja)** – dohvaća sve linije unutar datoteke na zadanoj putanji
  - **list(Path putanja)** – dohvaća sve datoteke na zadanoj putanji
  - **newDirectoryStream(Path putanja)** – otvara mapu i omogućava dohvaćanje njenog sadržaja
  - **walk(Path putanja)** – otvara mape i sve podmape u hijerarhiji te omogućava dohvaćanje sadržaja tih mapa
  - **readString** – čita cijeli sadržaj datoteke u jedan String s opcijom definiranja „CharacterSeta”
  - **writeString** – zapisuje cijeli String u datoteku s opcijom definiranja „CharacterSeta”
- Nakon dohvaćanja podataka o datotekama, moguće je koristiti razne metode za manipulaciju tih podataka temeljene na lambda izrazima kao što su **filter**, **foreach**, **limit**, **map** itd.

Serijalizacija i deserijalizacija u Javi

- Tijekom izvođenja Java programa objekti koji se koriste su u memoriji, a nakon završetka programa se oslobađa memorija koju su koristili
- Ako je potrebno „sačuvati” objekte kako bi se mogli koristiti i nakon završetka programa ili ih je potrebno pretvoriti u „fizički” oblik koji se može koristiti za slanje objekata na druge lokacije, potrebno ih je **serijalizirati**
- **Serijalizacija** se odnosi na zapisivanje stanja objekta (vrijednosti varijabli) u binarnom obliku u datoteku
- Obrnuti proces čitanja objekata u binarnom obliku i njihovo „aktiviranje” u trenutno aktivnom programu naziva se **deserijalizacija**
- Klase čiji objekti se žele serijalizirati moraju implementirati sučelje **Serializable**

Serijalizacija i deserijalizacija u Javi

- Primjer koda za serijalizaciju objekata:

```
ObjectOutputStream out = new ObjectOutputStream(
                                        new FileOutputStream("osobe.dat"));

Zupanija zagrebacka = new Zupanija("Zagrebačka", 309696, 3078d,
       Zupanija.POZIVNI_BROJ_ZUPANIJA_ZAGREBACKA_GRAD_ZAGREB);

Osoba osoba = new Osoba("Pero", "Perić", zagrebacka, new Date());

out.writeObject(osoba);
out.close();
```

- Za serijaliziranje objekata u binarnom obliku koristi se klasa „ObjectOutputStream” i metoda „writeObject”
- Proces deserijalizacije obavlja se istim redoslijedom kojim su objekti i serijalizirani

Serijalizacija i deserijalizacija u Javi

- Primjer koda za deserijalizaciju objekata:

```
try {
        ObjectInputStream in = new ObjectInputStream(
                new FileInputStream(
                        SerijalizacijaTest.SERIALIZATION_FILE_NAME));

        Osoba procitanaOsoba = (Osoba) in.readObject();

        System.out.println("Podaci o pročitanom objektu:");
        System.out.println("Ime osobe: " + procitanaOsoba.getIme());
        ...
        in.close();
} catch (IOException ex) {
        System.err.println(ex);
} catch (ClassNotFoundException ex) {
        System.err.println(ex);
}
```

Pitanja?