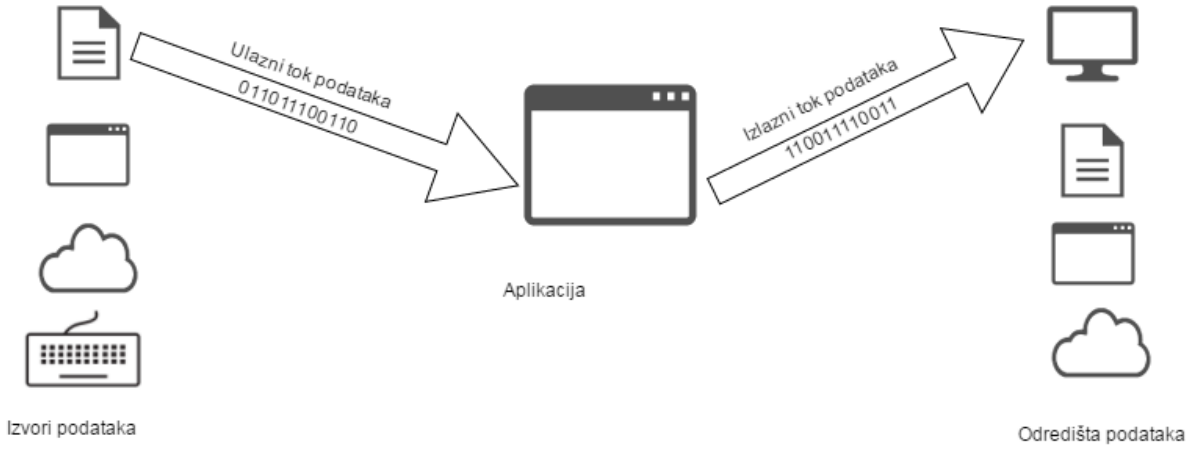
Fajlovi

Autori:
Milan Segedinac
Goran Savić

1. Ulazni i izlazni tokovi

Uvođenjem standardnog ulaza i standardnog izlaza omogućili smo da naš program komunicira sa spoljašnjim svetom: kroz standardni ulaz unosili smo podatke sa tastature, a kada smo ih slali na standardni izlaz prikazivali smo ih na ekran. Naravno, podatke je moguće preuzimati i iz drugih izvora osim sa tastature i moguće ih je slati na druga odredišta osim ekrana. *Ulazni i izlazni tokovi* (Input/Output [I/O] Streams) omogućuju nam da izvorima i destinacijama podataka pristupamo na uniforman način.

Na slici ispod vidimo kako se izvorima podataka i odredištima podataka pristupa kroz tokove podataka. Podaci koji se preuzimaju iz ulaznog toka u programu su dostupni kao niz binarnih vrednosti (nula i jedinica) bez obzira na implementaciju izvora podataka. Takođe, podaci koji se šalju na odredište kroz izlazni tok u programu se interpretiraju kao niz binarnih vrednosti nezavisno od konkratne implementacije odredišta. Time su sami podaci apstrahovani od implementacije izvora/odredišta. Obratite pažnju da isti objekat (na primer fajl) može da bude i izvor podataka (kada čitamo iz fajla) i odredište (kada snimamo u fajl).



Slika – Ulazni i izlazni tokovi podataka

Osnovni način pristupa podacima je da ih posmatramo kao nizove binarnih vrednosti bez ikakvog predefinisanog formata. Takav pristup realizovan je kroz *tokove bajtova*. Java programi koriste klase *InputStream* i *OutputStream* za pristup izvorima i odredištima podataka, pri čemu se podaci prenose kao bajtovi (8 bita [nula ili jedinica]). Postoji mnogo podtipova ovih klasa specifičnih za izvore i odredišta podataka kojima su namenjene. Programi koje smo do sada koristitli sve podatke su evidentirali u privremenoj RAM memoriji. Fajlovi nam omogućuju dugotrajno skladištenje podataka. Pošto ćemo mi raditi sa fajlovima, koristićemo klase *FileInputStream* i *FileOutputStream* za bajtovski prenos podataka. Programski kod klase koja vrši kopiranje iz fajla u fajl koristeći tokove bajtova prikazan je listingom ispod.

```
public class KopiranjeFajla {

       public static void main(String[] args) throws IOException  {
         FileInputStream in = null;
        FileOutputStream out = null;

        try {
            in = new FileInputStream("src/vp/fajlovi/turing.txt");
            out = new FileOutputStream("src/vp/fajlovi/outturing.txt");
            int c;

            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
       }
}
```

Listing - Kopiranje fajla pomoću tokova

Otvoren je ulazni tok (`in=new FileInputStream("src/vp/fajlovi/turing.txt");`) kojim će se čitati iz ulaznog fajla `src/vp/fajlovi/turing.txt` i izlazni tok (`out = new FileOutputStream("src/vp/fajlovi/outturing.txt");`) kojim će se snimati u izlazni fajl `src/vp/fajlovi/outturing.txt`. Zatim u while petlji učitavamo blok po blok (vrednost tipa `int`, 4 bajta odnosno 32 bita) iz ulaznog toka (`c = in.read()`) i učitane vrednosti šaljemo u izlazni tok (`out.write(c)`). Kada završimo čitanje poslednjeg bloka (kada stignemo do kraja fajla), umesto podatka ćemo dobiti vrednost -1, kao indikator da je čitanje završeno. Veoma je važno da ulazne i izlazne tokove zatvorimo kada nam više nisu potrebni, kao što smo uradili u `finally` bloku.

Ovakva reprezentacija ulaznih i izlaznih podataka je pogodna ukoliko želimo da podatke iz ulaznog toka samo pošaljemo na izlazni tok. Međutim, najčešće nam treba da podatke koje učitavamo iz ulaznog toka transformišemo pre nego što ćemo ih poslati na izlazni tok. Tada trebamo da interpretiramo šta *bajtovi u fajlu znače* kako bismo mogli da ih koristimo. Ako radimo sa tekstom, prevođenje teksta u binarni oblik podrazumeva da će svakom karakteru u tekstu biti dodeljena jednoznačna vrednost. U *ASCII standardu* svaki karakter enkodovan je jednoznačnom vrednošću koja zauzima 1 bajt. Tako je, na primer, za predstavljanje karaktera „`a`" namenjena vrednost `01100001`, a za predstavljanje cifre „`1`" vrednost `00110001`. Tekst „`hello world!`" bio bi zapisan kao `01101000 01100101 01101100 01101100 01101111 00100000 01110111 01101111 01110010 01101100 01100100 00100001`. Čitavu ASCII tabelu možete pogledati na linku http://www.ascii-code.com/.

Upis i čitanje tekstualnih podataka pri čemu se karakteri teksta automatski prevode u binarnu reprezentaciju i obrnuto uz oslonac na ASCII tabelu realizovano je pomoću *tokova karaktera*. U programskom jeziku Java tokovi karaktera implementirani su *reader* i *writer* klasama. Na primer za

čitanje i pisanje tekstualnih fajlova imamo `FileReader` i `FileWriter`. Ove dve klase iskorišćene su u pravljenju klasa `BufferedReader` i `PrintWriter` koje ćemo koristiti za rad sa tekstualnim fajlovima.

Primer – korisnička imena – rad sa tekstualnim fajlovima

Dat je fajl sa imenima i prezimenima korisnika sistema korisnici.txt. Svaki red u fajlu predstavlja podatke o jednom korisniku u sledećem formatu.

```
<ime>,<prezime>
```

Prikazaćemo program koji za korisnike kreira korisnička imena takva da je korisničko ime sačinjeno od prvog slova imena spojenog sa prezimenom korisnika. Korisnička imena snimiti u fajl.

Implementacija klase KorisnickaImena data je listingom ispod.

```
public class KorisnickaImena {
       public static void main(String[] args) throws IOException {

         BufferedReader inputStream = null;
        PrintWriter outputStream = null;

        try {
            inputStream = new BufferedReader(new
                 FileReader("src/vp/fajlovi/korisnici.txt"));
            outputStream = new PrintWriter(new
                 FileWriter("src/vp/fajlovi/korisnickaImena.txt"));

            String l;
            String[] korisnik;
            while ((l = inputStream.readLine()) != null) {
                 korisnik = l.split(",");
                 outputStream.println(korisnik[0].substring(0,1)+korisnik[1]);
            }
        } finally {
            if (inputStream != null) {
                inputStream.close();
            }
            if (outputStream != null) {
                outputStream.close();
            }
        }
       }
}
```

Listing – Generisanje korisničkih imena

Za pristup čitanje iz tekstualnog fajla korišćenje objekat klase `BufferedReader` čiji konstruktor prima tok karaktera `FileReader`. Za upis je korišćen objekat klase `PrintWriter` konstruisan na osnovu izlaznog toka karaktera `FileWriter`. U while petlji se učitava jedan po jedan red iz fajla pomoću metode `readLine` (setimo se da jedan red predstavlja jednog korisnika). Red se podeli po karakteru zarezu, a zatim se novi string nastao od prvog karaktera prvog elementa (imena) i čitavog drugog elementa (prezimena) snimi u izlazni tok metodom `println`.

Primer 2 – studentska služba – perzistiranje podataka

Rad sa fajlovima ćemo iskoristiti da obezbedimo perzistiranje podataka o studentima iz primera rada sa listama. Celokupan rad sa fajlovima izdvojićemo u zasebnu klasu, prikazanu listingom ispod.

```
public class StudentiIO {

	private static String putanja = "src/vp/studentskasluzba/studenti.txt";

	private static Student string2Student(String red) {
		String[] s = red.split(",");
		Student retVal = new Student(s[0], s[1], s[2],
Integer.parseInt(s[3]));
		return retVal;
	}

	private static String student2String(Student student) {
		return student.brojIndeksa + "," + student.ime + "," +
student.prezime
				+ "," + student.godinaStudija;
	}

	public static ArrayList<Student> ucitajStudente() throws IOException {
		BufferedReader inputStream = null;
		ArrayList<Student> studenti = null;
		try {
			inputStream = new BufferedReader(new FileReader(putanja));
			studenti = new ArrayList<Student>();
		      String l;
			while ((l = inputStream.readLine()) != null) {
				studenti.add(string2Student(l));
			}
			return studenti;
		} finally {
			if (inputStream != null) {
				inputStream.close();
			}
		}
	}

	public static void snimiStudente(ArrayList<Student> studenti) throws
IOException{
		PrintWriter outputStream = null;
		try {
			outputStream = new PrintWriter(new FileWriter(putanja));
			for (int i = 0; i < studenti.size(); i++) {
				outputStream.println(student2String(studenti.get(i)));
			}
		} finally {
			if (outputStream != null) {
				outputStream.close();
			}
		}

	}
}
```

Listing – ucitavanje i cuvanje podataka o studentima

Konverziju podataka o studentu iz String u Java objekat i obrnuto, iz Java objekta u String realizovali smo metodama `string2Student` i `student2String`.

Čitanje svih studenata iz fajla realizovano je metodom `ucitajStudente`. Nakon što je kreiran objekat klase `BufferedReader`, metodom `readLine` se učitava jedan po jedan red iz fajla sa podacima o studentima. Metodom `string2Student` svaki red se konvertuje u novi objekat klase `Student` i taj objekat se dodaje u listu koja će biti povratna vrednost ove funkcije. U `finally` bloku se zatvara tok kroz koji se čitaju podaci iz fajla.

Upis podataka o studentima u fajl realizuje se metodom `snimiStudente`. U ovoj metodi se kreira objekat klase `PrintWriter`. U while petlji se svaki objekat klase `Student` iz prosleđene liste konvertuje u string pozivom metode `student2String`, i dobijeni string se snima u fajl pozivom metode `println`.

Klasa `StudentIO` skriva detalje realizacije perzistencije podataka o studentima i omogućuje korišćenje koje se svodi na pozive metoda `ucitajStudente` i `snimiStudente`. Obzirom da su ove metode **statičke** koristićemo ih u klasi `StudentskaSluzba` bez instanciranja objekta klase `StudentIO`. Učitavanje i snimanje podataka ostudentima prikazano je listingom ispod.

```
        public StudentskaSluzba(){
//              this.studenti = new ArrayList<Student>();
                try {
                this.studenti = StudentiIO.ucitajStudente();
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }

        private void sacuvaj(){
                try {
                        StudentiIO.snimiStudente(studenti);
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }

...
        public boolean upisiStudenta(Student student){
                if(pronadjiStudenta(student.brojIndeksa)==null){
                        studenti.add(student);
                        sacuvaj();
                        return true;
                }
                else{
                        return false;
                }
        }
```

Listing – korišćenje klase `StudentIO`

Prilikom kreiranja objekta klase StuentskaSluzba listi studenata dodelićemo listu studenata učitanu iz fajla dobijenu pozivom `StudentiIO.ucitajStudente();` . Metoda sacuvaj poziva metodu `SnimiStudente` klase `StudentiIO` prosledivši joj kao parametar listu `studenti` iz instance klase `StudentskaSluzba`. U operacijama u kojima se menja lista studenata (na primer prilikom

upisa novog studenta) nakon dodavanje vrednosti u listu studenti, pozvaćemo metodu metodu `sacuvaj`.