Visoka škola elektrotehnike i računarstva strukovnih studija

Objektno orijentisano projektovanje

Ulazno izlazne operacije

2017

Sadržaj

Uvod

Java jezik za ulaz i izlaz podataka preko konzole pruža ograničenu podršku jer se u realnom okruženju za rad i komunikaciju sa korisnicima uglavnom koriste grafička okruženja. Java podržava takav pristup pa operacije koje se odvijaju preko konzole ne predstavljaju nešto čemu se posvetila pažnja prilikom projektovanja klasa. Jedine metode za izlazne operacije za prikaz podataka preko konzole koje su se do sada koristile su `print()` i `println()` a za ulaz se koristila klasa `Scanner`.

Bez obzira kako se komunicira sa korisnicima ili kako se prikazuju podaci (grafički ili preko konzole), čuvanje podataka na spoljašnjoj memoriji je nešto bez čega se ne može. Java nudi jednostavnu realizaciju operacija za rad sa datotekama i podacima na mreži.

Za ulazno-izlazne operacije Java koristi tokove ili strimove (*stream*). Tok je apstrakcija koja ili proizvodi ili prihvata podatke i u Javi je sa fizičkim uređajem povezan preko sistema ulazno-izlaznih klasa i metoda. Bez obzira o kom spoljašnjem uređaju je reč, svi tokovi se ponašaju isto i zbog toga se uvek koriste iste klase i metode za organizaciju ulaza i izlaza podataka. Za prihvat i obradu nebitno je da li podaci stižu sa tastature, sa diska ili mrežnog priključka.

Java prepoznaje dve vrste tokova: binarne i znakovne. Binarni služe za slanje i prihvatanje binarnih podataka a znakovni za primanje i slanje znakova. Suštinski, na najnižem nivou i jedan i drugi tok je u stvari binarni, samo što znakovni tokovi pružaju neke olakšice za rad sa znakovima koji su često prisutan tip podatka.

Za rad sa tokovima u Javi su zadužene klase u okviru paketa `java.io`.

Binarni tokovi

Rad sa binarnim tokovima se realizuje preko dve hijerarhije klasa. Na vrhu se nalaze dve apstraktne klase: `InputStream` i `OutputStream`. Svaka od ovih klasa ima više potklasa a u tabeli 1 navedene su neke od njih.

Tabela 1: Klase koje se odnose na rad sa binarnim tokovima

Klasa	**Svrha**
BufferedInputStream	Baferovani binarni ulazni tok
BufferedOutputStream	Baferovani binarni izlazni tok
ByteArrayInputStream	Ulazni tok koji podatke čita iz niza bajtova
ByteArrayOutputStream	Ulazni tok koji podatke upisuje u niz bajtova
DataInputStream	Ulazni tok koji sadrži metode za čitanje Javinih standardnih tipova podataka
DataOutputStream	Izlazni tok koji sadrži metode za upisivanje Javinih standardnih tipova podataka

Klasa	Svrha
FileInputStream	Ulazni tok koji čita podatke iz datoteke
FileOutputStream	Izlazni tok koji upisuje podatke u datoteku
InputStream	Apstraktna klasa koja opisuje ulazni tok
ObjectInputStream	Ulazni tok za objekte
ObjectOutputStream	Izlazni tok za objekte
OutputStream	Apstraktnaklasa koja opisuje izlazni tok
PrintStream	Izlazni tok koji sadrži metode print() i println()
RandomAccessFile	Slučajan pristup datotekama za čitanje i upisivanje

Apstraktne klase `InputStream` i `OutputStream` definišu nekoliko ključnih metoda koje realizuju potklase tokova. Najvažnije među njima su `read()` i `write()` koje su apstraktene i preko kojih se učitavaju, odnosno upisuju podaci.

Znakovni tokovi

Rad sa znakovnim tokovima se realizuje preko dve hijerarhije klasa na čijem vrhu se nalaze dve apstraktne klase: `Reader` i `Writer`. Ove klase su osposobljene da rade sa `Unicode` znakovima. Svaka od ovih klasa ima više potklasa a u tabeli 2 navedene su neke od njih.

Tabela 2: Klase koje se odnose na rad sa znakovnim tokovima

Klasa	Svrha
BufferedRead	Baferovani znakovni ulazni tok
BufferedWriter	Baferovani znakovni izlazni tok
CharArrayReader	Ulazni tok koji podatke čita iz niza znakova
CharArrayWriter	Ulazni tok koji podatke upisuje u niz znakova
FileReader	Ulazni tok koji podatke čita iz datoteke
FileWriter	Izlazni tok koji podatke upisuje u datoteku
InputStreamReader	Ulazni tok koji podatke tipa bajt prevodi u podatke tipa char
LineNumberReader	Ulazni tok koji broji redove
OutputStreamWriter	Izlazni tok koji podatke tipa char prevodi u podatke tipa bajt
PrintWriter	Izlazni tok koji sadrži metode print() i println()
Reader	Apstraktna klasa koja opisuje ulazni znakovni tok
StringReader	Ulazni tok koji podatke čita iz znakovnog niza
StringWriter	Izlazni tok koji podatke upisuje u znakovni niz
Writer	Apstraktnaklasa koja opisuje izlazni tok

Dve najznačajnije apstraktne metode koje potklase moraju da redefinišu su `read()` i `write()` koje učitavaju odnoso upisuju znakovne podatke. Nalaze se u apstraktim klasama `Reader` odnosno `Writer`.

Unapred definisani tokovi

Kao što je već rečeno, svaki Java program automatski uvozi paket java.lang koji definiše klasu `System`. Klasa `System` sadrži tri unapred definisane `public static` promenljive koje se odnose na tokove: `in`, `out` i `err`. Objekat `System.out` se odnosi na standardni izlazni tok što je podrazumevano konzola, objekat `System.in` se odnosi na standardni ulazni tok što podrazumevano predstavlja tastaturu i objekat `System.err` predstavlja standardni tok za greške što je podrazumevano konzola. Međutim, ovi tokovi mogu da se preusmere na bilo koji odgovarajući uređaj.

`System.in` je objekat tipa `InputStream` a `System.out` i `System.err` su objekti tipa `PrintStream` što znači da su to binarni tokovi ali uobičajeno se koriste za upisivanje i čitanje znakova preko konzole. To znači da se ovim tokovima mogu obuhvatiti i znakovni tokovi.

Unos podataka preko konzole

Osnovni način unosa podataka preko konzole je binarni tok. Vremenom se pokazalo da je pogodnije da se za unos podataka preko konzole koristi znakovni tok jer to omogućava višejezičnu podršku, lakšu obradu podataka i lakše održavanje.

Podaci koji stižu preko konzole unose se preko objekta `System.in`. Da bi se ulazni tok povezao sa konzolom potrebno je prvo da se napravi, što se postiže omotavanjem objekta `System.in` objektom klase `BufferedReader`. Klasa `BufferedReader`, kao što i samo njeno ime kaže, predstavlja baferovani ulazni tok. Konstruktor koji se najčešće koristi je:

BufferedReader (Reader ulazniTok)

gde je `ulazniTok` tok koji se povezuje sa primerkom objekta `BufferedReader`. Već je rečeno, da je `Reader` apstraktna klasa i jedna od njenih konkretnih potklasa je `InputStreamReader` koja binarne podatke pretvara u znakove. Da bi se dobio objekat klase `InputStreamReader` koji je povezan sa objektom `System.in` koristi se sledeći konstruktor:

InputStreamReader (InputStream inputStream)

Kada se slože navedena dva konstruktora dobija se sledeći programski red kojim se stvara objekat klase `BufferedReader`, povezan sa tastaturom:

BufferedReader ulaz=new BufferedReader(new InputStreamReader (System.in));

Ako preko konzole treba da se učita jedan znak onda se koristi metoda `read()` koja učitava jedan znak sa ulaznog toka i vraća ga kao celobrojnu vrednost. Kada se tok završi metoda vraća -1. Ova metoda može da generiše izuzetak `IOException` i zahteva da se on obradi.

```
1:  package ucitavanjeznakova;
2:  import java.io.*;
3:  public class UcitavanjeZnakova {
4:  public static void main(String[] args) {
5:  BufferedReader tok=new BufferedReader(new InputStreamReader(System.in));
6:  System.out.println("Unosite znakove, za kraj unesite broj");
7:  char znak;
8:  try{
9:  while(true)
10: {znak=(char) tok.read();
11:  if(znak<'0' || znak>'9') System.out.print(znak);
12:  else break;
13:  }
14:  System.out.println();
15:      } catch (IOException e){System.out.println ("Greška");}
16:                                                  }
17:                                              }
```

```
run:
Unosite znakove, za kraj unesite broj
ObjektnoOrijentisanoPr0jektovanje
ObjektnoOrijentisanoPr
BUILD SUCCESSFUL (total time: 15 seconds)
```

Slika 1: Unos znakova preko konzole

Na slici 1 dat je primer programa koji koristi metodu read da bi se učitao niz znakova sa tastature. Program završava rad kada se na ulazu pojavi znak cifre. Treba naglasiti da `System.in` standardno ima linijski bafer što znači da se ulazni tok ne obrađuje sve dok se ne pritisne `ENTER`. Zbog toga ova metoda nije najpogodnija za interaktivan rad preko konzole.

Da bi se preko tastature uneo znakovni niz (string) koristi se metoda `readLine()` klase `BufferedReader`. Njen opšti oblik je:

String readLine() throws IOException

Vidi se na osnovu deklaracije da metoda `readLine()` vraća objekat tipa `String`. Na slici 2 prikazan je program koji pravi niz objekata tipa `String` u koji se smeštaju redovi treksta koji stižu preko konzole. Program učitava najviše 100 redova ili dok se ne učita reč `STOP`. Metoda `readLine()` takođe može da generiše izuzetak i mora da se obradi. U primeru na slici 2 izuzetak se ne obrađuje preko `try-catch` bloka već „bacanjem“ izuzetka Javinom sistemu. Zbog toga u 4: redu redu stoji napomena da u `main()` metodi može da dođe do te situacije (... `throws IOException`).

```
1:  package ucitavanjepodatakaprekokonzole;
2:  import java.io.*;
3:  public class UcitavanjePodatakaPrekoKonzole {
4:  public static void main(String[] args) throws IOException {
5:  BufferedReader tok=new BufferedReader(new InputStreamReader (System.in));
6:  String redovi[]=new String[100];
7:  int n=0;
8:  System.out.println("Unesite redove teksta, za kraj unesite reč STOP");
9:  for(int i=0;i<100;i++)
10: {redovi[i]=tok.readLine();
11:   if(redovi[i].contentEquals("STOP")) {n=i;break;}}
12:   if(n==0)System.out.println("Niste uneli ni jedan red teksta");
13:   else {System.out.println("\nUneli ste sledeći tekst\n");
14:         for(int i=0;i<n;i++)
15:         System.out.println(redovi[i]);
16:         }
17:                                                    }
18:                                              }
```

```
run:
Unesite redove teksta, za kraj unesite reč STOP
Jedan red teksta.
Sada je mesec maj.
Danas radimo OOP.
Uskoro stizu tresnje.
STOP

Uneli ste sledeći tekst

Jedan red teksta.
Sada je mesec maj.
Danas radimo OOP.
Uskoro stizu tresnje.
BUILD SUCCESSFUL (total time: 33 seconds)
```

Slika 2: Obrada teksta

Ispisivanje podataka na konzoli

Do sada korišćene metode za ispis podataka na konzoli su `print()` i `println()`. One su definisane klasom `PrintStream` koja predstavlja tip objekta na koji referencira objekat `System.out`. `PrintStream` predstavlja izlazni tok izveden iz klase `OutputStream` to znači da nasleđuje i metodu `write()` koja na taj način može da se iskoristi za ispisivanje podataka na konzoli. Međutim, ona se ne koristi često u tu svrhu jer je mnogo lakše da se radi sa `print()` i `println()`.

Za ispisivanje podataka na konzoli do sada u svim primerima se koristio objekat System.out (a i na dalje će). To je sasvim uredu u jednostavnim primerima ali za ispisivanje na konzoli u realnim programima preporuka je da se koristi tok `PrintWriter`. Klasa `PrintWriter` ima nekoliko konstruktora a jedan od njih je:

PrintWritter (OutputStream izlazniTok, boolean noviRed)

gde `izlazniTok` predstavlja objekat tipa `OutputStream` a `noviRed` određuje da li Java svaki put kada se pozove metoda ispisuje podatke iz izlaznog toka. Ako promenljiva `noviRed` ima vrednost `true` ispisivanje toka je automatsko u suprotnom nije.

Metode `print()`i `println()` toka `PrintWriter` podržavaju rad sa svim tipovima uključujući i tip `Object` tako da se normalno koriste kao i u objektu `System.out`.

U primeru na slici 3 prikazana je upotreba klase `PrintWriter` i metoda `write()`, `println()` i `print()`. Navedeni primer je dobijen modifikacijom mprimera prikazanog na slici 2. Obrada izuzetka koji može da generiše metoda `redaLine()` je u okviru `try-catch` bloka koji okružuje samo poziv te metode (red 11:). U redu 7: definisan je objekat `pw` koji povezuje izlazni tok i konzolu. Parametar `true` je neophodan da bi došlo do ispisivanja. U redovima 15: i 16: korišćena je metoda `write()` dok je u redu 18: korišćena metoda `println()` jer upotreba `write()` ne bi dovela do željenog ispisa (zašto?).

```
1:  package ispispodatakanakonzoli;
2:  import java.io.*;
3:  public class IspisPodatakaNaKonzoli {
4:  public static void main(String[] args) {
5:  String redovi[]=new String[100];
6:  BufferedReader tok=new BufferedReader(new InputStreamReader (System.in));
7:  PrintWriter pw=new PrintWriter(System.out, true);
8:  int n=0;
9:  System.out.println("Unesite redove teksta, za kraj unesite reč STOP");
10: for(int i=0;i<100;i++)
11: {try { redovi[i]=tok.readLine();} catch (IOException e){System.out.println("Greška");return;}
12:         if(redovi[i].contentEquals("STOP")) {n=i;break;}
13: }
14: if(n==0)System.out.println("Niste uneli ni jedan red teksta");
15: else { pw.write("\nBroj redova teksta koji ste uneli je "+n);
16:          pw.write("\nUneli ste sledeći tekst:\n");
17:          for(int i=0;i<n;i++)
18:          pw.println(redovi[i]);}
19:                                                  }
20:                                                  }
```

```
run:
Unesite redove teksta, za kraj unesite reč STOP
Objektno orijentisano programiranje.
Olovka je na stolu.
Volim da kupujem knjige.
STOP

Broj redova teksta koji ste uneli je 3
Uneli ste sledeći tekst:
Objektno orijentisano programiranje.
Olovka je na stolu.
Volim da kupujem knjige.
BUILD SUCCESSFUL (total time: 1 minute 25 seconds)
```

Slika 3: Upotreba klase `PrintWriter`

Rad sa datotekama

Java obezbeđuje više klasa i metoda koje omogućavaju da se podaci upisuju i čitaju iz datoteke i sve rade sa binarnim podacima. Dve najčešće korišćene klase tokova su `FileInputStream` i `FileOutputStream` koje prave binarne tokove povezane sa datotekama. Ove klase imaju nekoliko konstruktora ali se najčešće koritre:

FileInputStream (String imeDatoteke) throws FileNotFoundException
FileOutputException (String imeDatoteke) throws FileNotFoundException

gde je `imeDatoteke` naziv datoteke kojoj se obraćamo za upis odnosno čitanje podataka. Obe metode generišu izuzetak ukoliko tokovi ne mogu da se povežu sa naznačenom datotekom i koji mora da se obradi. U slučaju čitanja problem je ukoliko naznačena datoteka ne može da se pronađe (voditi računa o putanji) a u slučaju izlaza problem je ukoliko ciljana datoteka ne može da se napravi. Ukoliko datoteka postoji i program joj se obrati za upis, prethodni sadržaj će biti uništen.

Kada se završi rad sa datotekom ona mora da se zatvori i to se radi pomoću sledeće metode:

void close() throws IOException

Čitanje podataka bajt po bajt se realizuje pomoću metode `read()` a upisivanje u datoteku bajt po bajt se realizuje pomoću metode `write()`:

int read () throws IOException
void write (int vrednost)throws IOException

Na slici 4 dat je primer rada sa datotekama. Prva datoteka iz koje se čitaju podaci je datoteka sa izvornim Java kodom. Ime te datoteke je smešteno u string `datoteka.java`. Datoteka u koju se prepisuje kod je datoteka sa ekstenzijom `txt` ali sa istim imenom kao i Java datoteka, smešta se u isti folder gde se nalazi i izvorni kod a kompletno ime se čuva u stringu `datotekaTxt` (redovi 7: i 11:). U redu 4: naglašava se da klasa može da generiše izuzetak i to se odnosi na izuzetak koji može da se generiše kod zatvaranja datoteka (redovi 26: i 34:). U prvom try-catch bloku (redovi od 15: do 18:) se pravi objekat koji povezuje tok i datoteku iz koje će se čitati podaci. Slično, u redovima 0d 18: do 20: se povezuje izlazni tok sa datotekom u koju će biti upisani podaci. U try-catch bloku koji počinje u redu 21: a završava se u bloku 25: vrši se čitanje podatka bajt po bajt iz prve datoteke i upisivanje u odredišnu datoteku takođe bajt po bajt. Treba primetiti da metoda read() vraća celobrojnu vrednost i ona će biti -1 kada se stigne do kraja datoteke. Zbog toga i stoji takav uslov u petlji while (red 23:). Zatim se oba toka zatvaraju. Nakon toga se ponovo otvara tekstualna datoteka i njen sadržaj se prpisuje u konzolu. Na taj način u konzoli se ispisuje kompletan izvorni Java kod (redovi 27: do 34:).

```
1:  package pravljenjekopiranjeprikazdatoteke;
2:  import java.io.*;
3:  public class PravljenjeKopiranjePrikazDatoteke {
4:  public static void main(String[] args) throws IOException{
5:     FileInputStream fin=null;
6:     FileOutputStream fout=null;
7:     String datotekaJava="C:\\Users\\motranet\\Documents\\NetBeansProjects\\
8:                          PravljenjeKopiranjePrikazDatoteke\\src\\
9:                          pravljenjekopiranjeprikazdatoteke\\
10:                         PravljenjeKopiranjePrikazDatoteke.java";
11:    String datotekaTxt="C:\\Users\\motranet\\Documents\\NetBeansProjects\\
12:                        PravljenjeKopiranjePrikazDatoteke\\src\\
13:                        pravljenjekopiranjeprikazdatoteke\\
14:                        PravljenjeKopiranjePrikazDatoteke.txt";
15:    try
16:       {fin=new FileInputStream(datotekaJava);}
17:    catch (FileNotFoundException e){System.out.println("Datoteka nije pronađena");}
18:    try
19:       {fout=new FileOutputStream(datotekaTxt);}
20:    catch (FileNotFoundException e){System.out.println("Datoteka nije pronađena");}
21:    try
22:       {int c;
23:        while((c=fin.read())!=-1)
24:        fout.write(c);}
25:    catch (IOException e){System.out.println("Greška u radu sa datotekama");}
26:    fin.close(); fout.close();
27:    try
28:       {fin=new FileInputStream(datotekaTxt);}
29:    catch (FileNotFoundException e){System.out.println("Datoteka nije pronađena");}
30:    try
31:       {int c;
32:        while((c=fin.read())!=-1)System.out.write(c);}
33:    catch (IOException e){System.out.println("Greška u radu sa datotekama");}
34:    fin.close();
35:                                    }
36:                                 }
```

```
run:
package pravljenjekopiranjeprikazdatoteke;
import java.io.*;
public class PravljenjeKopiranjePrikazDatoteke {
public static void main(String[] args) throws IOException{
   FileInputStream fin=null;
   FileOutputStream fout=null;
   String datotekaJava="C:\\Users\\motranet\\Documents\\NetBeansProjects\\
                        PravljenjeKopiranjePrikazDatoteke\\src\\
                        pravljenjekopiranjeprikazdatoteke\\
                        PravljenjeKopiranjePrikazDatoteke.java";
   String datotekaTxt="C:\\Users\\motranet\\Documents\\NetBeansProjects\\
                       PravljenjeKopiranjePrikazDatoteke\\src\\
                       pravljenjekopiranjeprikazdatoteke\\
                       PravljenjeKopiranjePrikazDatoteke.txt";
   try
      {fin=new FileInputStream(datotekaJava);}
   catch (FileNotFoundException e){System.out.println("Datoteka nije pronađena");}
   try
```

```
            {fout=new FileOutputStream(datotekaTxt);}
          catch (FileNotFoundException e){System.out.println("Datoteka nije pronađena");}
          try
            {int c;
             while((c=fin.read())!=-1)
             fout.write(c);}
          catch (IOException e){System.out.println("Greška u radu sa datotekama");}
          fin.close(); fout.close();
          try
            {fin=new FileInputStream(datotekaTxt);}
          catch (FileNotFoundException e){System.out.println("Datoteka nije pronađena");}
          try
            {int c;
             while((c=fin.read())!=-1)System.out.write(c);}
          catch (IOException e){System.out.println("Greška u radu sa datotekama");}
          fin.close();
                                          }
      BUILD SUCCESSFUL (total time: 0 seconds)
```

Slika 4: Upisivanje i čitanje iz datoteka

Klasa Scanner i ulaz/izlaz

Klasa `Scanner` se do sada koristila za učitavanje podataka koji stižu sa konzole i to različitih tipova. Međutim, ona može da se koristi za čitanje podataka iz datoteke, znakovnog niza, ili nekog drugog izvora koji realizuje interfejs `Readable` ili `ReadableByteChannel`.

Konstruktori definisani u klasi `Scanner` dati su u tabeli 3.

Tabela 3: Konstruktori klase Scanner

Klasa	**Svrha**
Scanner (File dat) throws FileNotFoundException	Pravi objekat tipa Scanner koji koristi datoteku dat kao izvor podatka
Scanner (File dat, String font) throws FileNotFoundException	Pravi objekat tipa Scanner koji koristi datoteku dat kao izvor podatka napisanim tipom slova font
Scanner (InputStream tok)	Pravi objekat tipa Scanner koji koristi tok kao izvor ulaznih podataka
Scanner (InputStream tok, String font)	Pravi objekat tipa Scanner koji koristi tok kao izvor ulaznih podataka napisanim tipom slova font
Scanner (Readable dat)	Pravi objekat tipa Scanner koji objekat dat sa interfejsom Readable koristi kao izvor ulaznih podataka
Scanner (ReadableByteChanneldat)	Pravi objekat tipa Scanner koji objekat dat sa interfejsom Readable koristi kao izvor ulaznih podataka

Klasa	Svrha
Scanner (ReadableByteChanneldat, String font)	Pravi objekat tipa Scanner koji objekat dat sa interfejsom Readable koristi kao izvor ulaznih podataka napisanih tipom slova font
Scanner (String str)	Peravi objekat tipa Scanner koji koristi znakovni niz str kao izvor ulaznih podataka

Na slici 5 dat je primer upotrebe klase `Scanner` za ištitavanje podataka iz tekstualne datoteke. Konkretno, podaci se čitaju red po red sve dok redovi postoje a najviše 5 redova.

```
1:  package pravljenjekopiranjeprikazdatoteke;
2:  import java.io.*;
3:  import java.util.*;
4:  public class PravljenjeKopiranjePrikazDatoteke {
5:  public static void main(String[] args) throws IOException{
6:  FileReader fin=null;
7:  String datotekaTxt="C:\\Users\\motranet\\Documents\\
8:                          NetBeansProjects\\PravljenjeKopiranjePrikazDatoteke\\src\\
9:                          pravljenjekopiranjeprikazdatoteke\\
10:                         PravljenjeKopiranjePrikazDatoteke.txt";
11:  fin=new FileReader(datotekaTxt);
12:  Scanner izvor=new Scanner (fin);
13:  int n=0;String red;
14:  while (izvor.hasNext() && n<5)
15:      {red=izvor.nextLine();n++;
16:       System.out.println(red);}
17:  fin.close();
18:                                                  }
19:                                              }
```

```
run:
package pravljenjekopiranjeprikazdatoteke;
import java.io.*;
public class PravljenjeKopiranjePrikazDatoteke {
public static void main(String[] args) throws IOException{
   FileInputStream fin=null;
BUILD SUCCESSFUL (total time: 1 second)
```

Slika 5: Čitanje podataka iz tekstualne datoteke upotrebom klase Scanner

U redovima 11: i 12: pravi se objekat tipa `Scanner` koji čita tekstualnu datoteku sa imenom koje je smešteno u string `datotekaTxt` (tekstualna datoteka iz prethodnog primera). Ovo je moguće zato što `FileReader` realizuje interfejs `Redable` pa se poziv konstruktora razrešava u `Scanner(Readable)` konstruktoru. Treba primetiti da nigde ne postoji `try-catch`. Ako nije spcifikovano da metoda generiše izuzetak to ne znači da se to neće desiti. Kao i uvek i ovde neprolaženje naznačene datoteke će proutrokovati pojavu izuzetka i o tome treba misliti prilikom pisanja programa.

Paket java.nio

Java nudi i čitav niz ulazno izlaznih metoda smeštenih u nekoliko paketa koji se jednim imenom nazivaju `NIO` (`NIO - new input output`). NIO paketi ne predstavljaju zamenu za standardni java.io paket već ga nadopunjuje.

Sistem NIO predstavlja drugačiji pristup ulazno izlaznim operacijama koji se bazira na dvema osnovnim strukturama: baferima ikanalima. Kanali predstavljaju otvorenu vezu sa ulazno izlaznim uređajem (koji može biti bilo koji). Podaci se čitaju sa uređaja ili upisuju na uređaj preko bafera. Dakle, za svaki ulazno izlazni uređaj se definiše kanal i bafer u koji će se smeštati podaci koji se razmenjuju sa uređajem.

Rad sa NIO paketima pruža čitav niz mogućnosti a u osnovi rad kanalima i baferima je prilično jednostavan. Na slici 6 dat je primer pristupanja tekstualnoj datoteci i čitanja podataka iz nje preko klasa iz NIO sistema. U samom kodu se nalaze objašnjenja za svaki red. Tekstualna datoteka koja se do sada koristila u primerima je na istoj lokaciji ali joj je promenjen sadržaj pre upotrebe.

```
1:  package pravljenjekopiranjeprikazdatoteke;
2:  import java.io.*;
3:  import java.nio.*;
4:  import java.nio.channels.*;
5:  public class PravljenjeKopiranjePrikazDatoteke {
6:  public static void main(String[] args) throws IOException{
7:     String datotekaTxt="C:\\Users\\motranet\\Documents\\
8:                         NetBeansProjects\\PravljenjeKopiranjePrikazDatoteke\\
9:                         src\\pravljenjekopiranjeprikazdatoteke\\
10:                        PravljenjeKopiranjePrikazDatoteke.txt";
11:    FileInputStream dat;
12:    FileChannel kanal;
13:    long velicinaDatoteke;
14:    ByteBuffer bafer;
15:
16:    try
17:    {//Otvara se datoteka za ulaz
18:     dat=new FileInputStream(datotekaTxt);
19:
20:     //Pribavlja se kanal za datoteku
21:     kanal=dat.getChannel();
22:
23:     //Pribavlja se veličina datoteke
24:     velicinaDatoteke=kanal.size();
25:
26:    //Pravi se bafer odgovarajuće veličine
27:     bafer=ByteBuffer.allocate((int)velicinaDatoteke);
28:
29:    //Učitavanje datoteke u bafer preko kanala u bafer
30:     kanal.read(bafer);
31:
32:    //Vraćanje na početak bafera da bi podaci mogli da se čitaju
```

```
33:             bafer.rewind();
34:
35:            //Čitanje i i prikaz na konzoli bajtova iz bafera
36:             for(int i=0;i<velicinaDatoteke;i++)   System.out.print((char)bafer.get());
37:             System.out.println();
38:
39:             //Zatvaranje kanala
40:             kanal.close();
41:
42:             //Zatvaranje datoteke
43:             dat.close();
44:            }
45:            catch(IOException e){System.out.println("Greška");}
46:                                                 }
47:                                           }
```

```
run:
□╕ □
   try
   {//Otvara se datoteka za ulaz
    dat=new FileInputStream(datotekaTxt);
    //Pribavlja se kanal za datoteku
    kanal=dat.getChannel();
    //Pribavlja se velicina datoteke
    velicinaDatoteke=kanal.size();
    //Pravi se bafer odgovarajuce velicine
    bafer=ByteBuffer.allocate((int)velicinaDatoteke);
    //Ucitavanje datoteke u bafer preko kanala u bafer
    kanal.read(bafer);
    //Vracanje na pocetak bafera da bi podaci mogli da se citaju
    bafer.rewind();
    //Citanje i i prikaz na konzoli bajtova iz bafera
    for(int i=0;i<velicinaDatoteke;i++)
       System.out.print((char)bafer.get());
    System.out.println();
    //Zatvaranje kanala
    kanal.close();
    //Zatvaranje datoteke
    dat.close();
   }
   catch(IOException e){System.out.println("Greska");}

BUILD SUCCESSFUL (total time: 0 seconds)
```

Napomena: U poslednjem primeru (a i u nekoliko prethodnih) u deklaraciji glavne metode stoji da ona može da generiše izuzetak (ali izuzetak se ne baca već se javlja Javin automatski sistem za obradu grešaka) ali je istovremeno i sve stavljeno u try-catch blok koji hvata teaj isti tip izuzetka. Bespotrebno dupliranje je samo u funkciji da se naznači važnost obrade izuzetaka i da je neophodno da se u svakom trenutku prilikom pisanja programa vodi računa o situacijama koje generišu izuzetke.