PT&SCHE

Introduction of part-time and short cycle studies in Serbia

Funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

UNIVERZITET METROPOLITAN BEOGRAD

KI105 - JAVA 3: PROGRAMIRANJE KORISNIČKOG INTERFEJSA

Ulazne i izlazne datoteke

Lekcija 06

PRIRUČNIK ZA STUDENTE

KI105 - JAVA 3: PROGRAMIRANJE KORISNIČKOG INTERFEJSA

Lekcija 06

ULAZNE I IZLAZNE DATOTEKE

UVOD

Java obezbeđuje mnogo klasa za realizaciju ulaza ili izlaza tekstualnih i binarnih podataka.

Ova lekcija obezbeđuje ispunjenje sledećih ciljeva:

- Otkriti svojstva fajlova i direktorijuma, brisanje i promena imena fajlova upotrebom klase File
- Upisivanje podataka na fajlu upotrebom klase PrintWriter
- Čitanje podataka iz fajla upotrebom Scanner klase i razumevanje tog procesa
- Otvaranje fajlova upotrebom GUI dijalog polja
- Čitanje podataka sa veba
- Obrada ulaza i i izlaza u Javi
- Razlikovanje između ulaza i izlaza tekstova i ulaza i izlaza binarnih podataka
- Upotreba klasa **FileInput Stream i FileOutputStream**
- Filtriranje podataka upotrebom osnovnih klasa **FilterInputStream i FilterOutputStream**
- Čitanje i šisanje primitivnih vrednosti i stringova upotrebom **DataInputStream i DataOutputStream**
- Poboljšanje performansi ulaza i izlaza upotrebom **BufferedInputStream i Buffered OutputStream**
- Pisanje programa za kopiranje datoteka
- Memorisanje objekata upotrebom **ObjectOutputStream i ObjectInputStream**
- Serijalizacija nizova
- Čitanje i pisanje datoteka upotrebom klase **RandomAccessFile**

Referenca: Y. Daniel Liang, INTRODUCTION TO JAVA PROGRAMMING (COMPREHENSIVE VERSION), Tenth Edition, Pearson, ISBN 10: 0-13-376131-2, ISBN 13: 978-0-13-376131-3

Ovo je osnovni udžbenik za ovaj predmet i preporučuje se studentima da ga koriste,

Poglavlje 1

Datoteka File

ŠTA SU DATOTEKE?

Apsolutni naziv datoteka je put do logičke adrese datoteke smeštene u nekom memorijskom medijumu. Relativni naziv datoteke se definiše u odnosu na direktorijum u kome je smeštena

Podaci koji se koriste u programima su memorisani u radnoj memoriji računara samo privremeno, dok program radi. Po završetku programa, ti podaci se gube. Da bi se trajno sačuvali, potrebno je da ih smestite u neku datoteku koju bi onda smestili (memorisali) na nekom drugom sredstvu za trajni smeštaj podataka. Tako uskladištena, tj. trajno zapisana na nekom memorijskom medijumu (npr. magnetnom disku) datoteka sa podacima se može transportovati i čitati u drugim programima.

Datoteka je skup podataka koji se smeštaju na jednoj logičkoj adresi u trajnoj memoriju računara (npr. diskovi).

Svaka **datoteka** (engl. file) se smešta u elektronsku **fasciklu**, tj. direktorijum (engl. directory) sistema datoteka koji obezbeđuje operativni sistem računara. *Apsolutno (puno) ime datoteke sadrži kompletan put do datoteke, polazeći od oznake memorijskog medijuma na kome se nalazi. Na primer:* ***c:\book\Welcome.java*** označava da je datoteka **Welcome.java** smeštena u medijumu koji je označen sa c u operativnom sistemu računara i u direktorijumu koji se naziva book.

.

Apsolutni naziv datoteka pokazuje put do logičke adrese datoteke smeštene u nekom memorijskom medijumu. Apsolutni nazivi datoteka zavise od računara i od operativnog sistema koji koriste. Na primer, ako radite na UNIX operativnom sistemu, apsolutna adresa datoteke **Welcome.java** može biti**: /home/liang/book/Welcome.java** , gde je **/home/liang/book** put do direktorijuma na kome je smeštena datoteka **Welcome.java**

Relativni naziv datoteke (ili fajla) se definiše u odnosu na direktorijum u kome je smeštena. Na primer, **Welcome.java** je relativan naziv datoteke, jer ne sadrži informaciju o putu direktorijuma u kome je datoteka smeštena.

Nemojte koristiti apsolutna imena datoteka u vašem programu, jer time ograničavate primenu vašeg programa samo na platformi koja primenjuje apsolutnu adresu koju koristite. Da bi to ograničenje izbegli, koristite relativna imena datoteka, jer su ona univerzalna i primenjuju se na svim platformama (računarima i operativnim sistemima).

KLASA FILE

Klasa File sadrži metode za dobijanje svojstava datoteka i direktorijuma, kao i za definisanje njihovih naziva i za njihovo brisanje.

Klasa***File*** obezbeđuje atribute i metode koji sadrži podatke koji su specifični za platformu na kojoj je datoteka smeštena. To omogućava da korisnik/programer ne mora da zna te detalje, ako se kreira objekt klase***File*** za platformu na kojoj radi. Slika 1 prikazuje svojstva (atribute i metode) klase***File***.

Klasa ***File*** je Javina datoteka definisana i paketi***java.io***. Sve datoteke (fajlovi) koje kreirate, su podklase ove datoteke, te automatski nasleđuju sva njena svojstva, tj. atribute i metode.

Slika 1.1 Atributi i metodi klase File

PRIMER 1 - KREIRANJE DATOTEKA

Dati primer kreira objekat klase File

Ovde je prikazan primer programa klase **TestFileClass** koji kreira ***File*** objekat i upotrebljava njegove metode (klase***File***) da bi se dobili njegova svojstva. Objekat ***File*** se smešta u datoteku **us.gif** koja se smešta u direktorijum **image.**

Metod **lastModified()** vraća datum i vreme poslednje promene datoteke, merenjem u milisekundama od 1.1.1970, po GMT. Klasa ***Date*** se upotrebljava da bi se podaci o datumu i vremenu prikazali, u linijama 14-15.

```
1 public class TestFileClass {
  2   public static void main(String[] args) {
  3  java.io.File file = new java.io.File("image/us.gif");
  4  System.out.println("Does it exist? "+ file.exists());
```

```
 5  System.out.println("The file has " + file.length() + " bytes");
 6  System.out.println("Can it be read? " + file.canRead());
 7  System.out.println("Can it be written? " + file.canWrite());
 8  System.out.println("Is it a directory? " + file.isDirectory());
 9  System.out.println("Is it a file? " + file.isFile());
10  System.out.println("Is it absolute? " + file.isAbsolute());
11  System.out.println("Is it hidden? " + file.isHidden());
12  System.out.println("Absolute path is " +
13     file.getAbsolutePath());
14  System.out.println("Last modified on " +
15     new java.util.Date(file.lastModified()));
16   }
17 }
```

KLASA FILE (VIDEO)

Video objašnjava primenu klase File koju Java nudi za rad sa datotekama koje sadrže tekstualne podatke.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ZADACI 1.1 - 1.3

Proverite vaše razumavanje klase File.

1. Šta je pogrešno u sledećem iskazu kreiranja File objekta?

new File("c:\book\test.dat");

2. Kako proveravate da neka datoteka već postoji? Kako brišete datoteku? Kako menjati naziv datoteke? Kako nalazite veličinu datoteke (u bajtovima) upotrebom klase File? Kako kreirate direktorijum?

3. Kako upotrebljavate klasu File za ulaz/izlaz? Da li se kreiranje File objekta kreira i datoteka na disku?

Poglavlje 2

Ulaz i izlaz tekstualnih datoteka

KLASA PRINTWRITER

PrintWriter klasa se koristi za upisivanje tekstualnih podataka u neku datoteku.

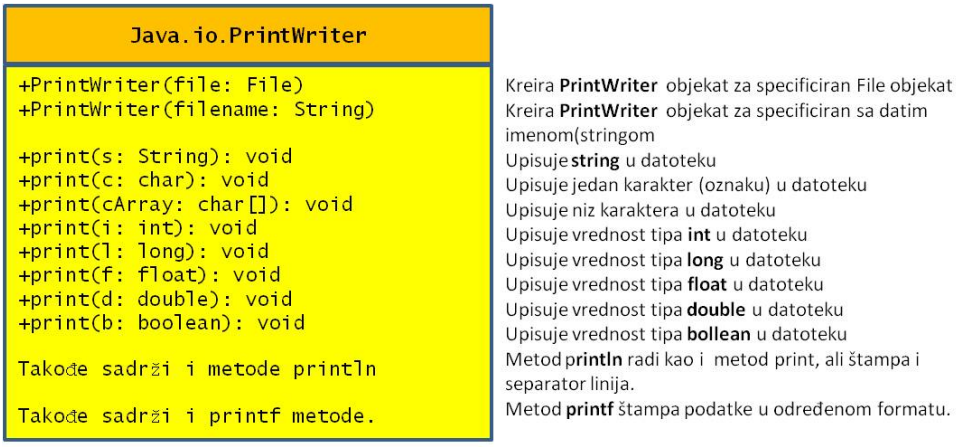
Klasa ***File*** (te i njeni objekti) sadrži svojstva datoteke (atribute i metode), ali ne sadrži metode za kreiranje datoteke i za čitanje ili pisanje tekstualnih podataka u datoteku, odn. iz datoteke. Postoje dve vrste datoteke:

1. Tekstualne datoteke – sadrže tekstualne stringove i numeričke vrednosti
2. Binarne datoteke – sadrže podatke u binarnom obliku

Ovde ćemo se najpre baviti tekstualnim datotekama, a kasnije, i binarnim.

Klasa ***java.io.PrintWriter*** se može koristiti za kreiranje datoteke i upisivanje podataka u tekstualnu datoteku. Sledećim iskazom se kreira tekstualna datoteka:

Posle kreiranja objekta klase ***PrintWriter***, mogu se pozivati njeni metodi: print, println i printf kojim se zapisuju podaci u tekstualnu datoteku. Na slici 3 prikazana su najčešće korišćeni metodi klase PrintWriter.



Slika 2.1 Atributi i metodi PrintWriter

PRIMER 2 - KORIŠĆENJE KLASE PRINTWRITER

Kreira se objekt klase PrintWriter za upisivanje dve linije sa imenima u datoteku score.txt

Na slici 4 prikazan je primer kreiranja objekta (instance) klase **PrintWriter** i upisivanje dve linije u datoteku **scores.txt**. Svaka linija sadrži ime (kao string), inicijal srednjeg imena (kao karakter), prezime (kao string) i ocenu (kao ceo broj).

Linije 4-7 proveravaju da li već postoji datoteka **scores.txt.** Ako postoji, program završava (linija 6). Pozivanjem konstruktora klase ***PrintWrite***r, kreira se nova datoteka ako datoteka ne postoji. Ako datoteka postoji, njen sadržaj se zamenjuje novim.

Poziv konstruktora***PrintWriter*** može da izbaci jedan ulazno/izlazni izuzetak. Radi jednostavnosti, ovde je u liniji 2 metoda ***main*** objavljen izuzetak sa throws IOException.

Koriste se sledeći metodi za prikazivanje teksta na standardnoj konzori računara: ***System.out.print, System.out.println***, i ***System.out.printf.***

System.out je standardni objekat Jave za prikazivanje izlaza na konzoli računara. Kada kreirate***PrintWriter*** objekte za upisivanje teksta u bilo koju datoteku primenom ***metoda print, println*** i ***printf*** (linije 13-16). Metod clo***se()*** se mora da koristi da bi se zatvorila datoteka. U suprotnom, podaci se mogu pogrešno memorisati.

```
1 public class WriteData {
  2   public static void main(String[] args) throws IoException {
  3  java.io.File file = new java.io.File("scores.txt");
  4  if (file.exists()) {
  5    System.out.println("File already exists");
  6    System.exit(1);
  7  }
  8
  9  // Kreiranje datoteke
 10  java.io.PrintWriter output = new java.io.PrintWriter(file);
 11
 12  // Upisivanje formatizovanog izlaza u datoteku
 13  output.print("John T Smith ");
 14  output.println(90);
 15  output.print("Eric K Jones ");
 16  output.println(85);
 17
 18  // Zatvsrsnje datoteke
 19  output.close();
 20   }
 21 }
```

UPISIVANJE PODATAKA U DATOTEKU (VIDEO)

Video objašnjava kako iz java programa možete zapisati tekstualne podatke u neku datoteku.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

PRIMER 3

Cilj ovog zadatka je da se pokažu dva metoda upisivanja u fajl preko Jave.

Napisati program koji upisuje u tekstualni fajl sledeći tekst: Ovo je jedna linija u fajlu. Napraviti program tako da jednom upiše ovaj tekst u fajl, a drugi put da na postojeći tekst doda isti tekst kao nova linija teksta ispod.

Objašnjenje:

PrintWriter je klasa koja nam omogućava upis u fajl. Ona kroz konstruktor prima naziv fajla. Korišćenjem metode println ispisujemo novu liniju u fajl. Metoda close() se uvek mora staviti na kraju da bi se ono što je upisano bilo i sačuvano.

FileWriter u ovom delu zadatka nam omogućava da dodajemo tekst na postojeći sadržaj u fajlu.
Moramo vršiti obradu izuzetaka FileNotFoundException i IOException korišćenjem try i catch bloka. Kada se dogodi greška u izvršavanju koda koji se nalazi pod try blokom program prelazi na izvršenje catch bloka.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package zadatak1;

import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

/**
 *
 * @author razvoj
 */
public class Zadatak1 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        new Zadatak1();
    }
```

```
    public Zadatak1() {

        //Upis linije u fajl sa brisanjem svega sto je u fajlu do sada
        PrintWriter upisUFajl;
        try {
            upisUFajl = new PrintWriter("novifajl.txt");
            upisUFajl.println("Ovo je jedna linija u fajlu");
            upisUFajl.close();
        } catch (FileNotFoundException e) {
            System.out.println("Došlo je do greške");
        }

        //Dodavanje linije na postojeci fajl
        try {
            upisUFajl = new PrintWriter(new FileWriter("novifajl.txt", true));
            upisUFajl.println("Ovo je jedna linija dodata na proslu liniju u
fajlu");
            upisUFajl.close();
        } catch (FileNotFoundException e) {
            System.out.println("Došlo je do greške");
        } catch (IOException ex) {
            System.out.println("Došlo je do greške");
        }
    }
}
```

KLASA SCANNER

Klasa Scanner se upotrebljava za čitanje tekstualnih podataka iz datoteke

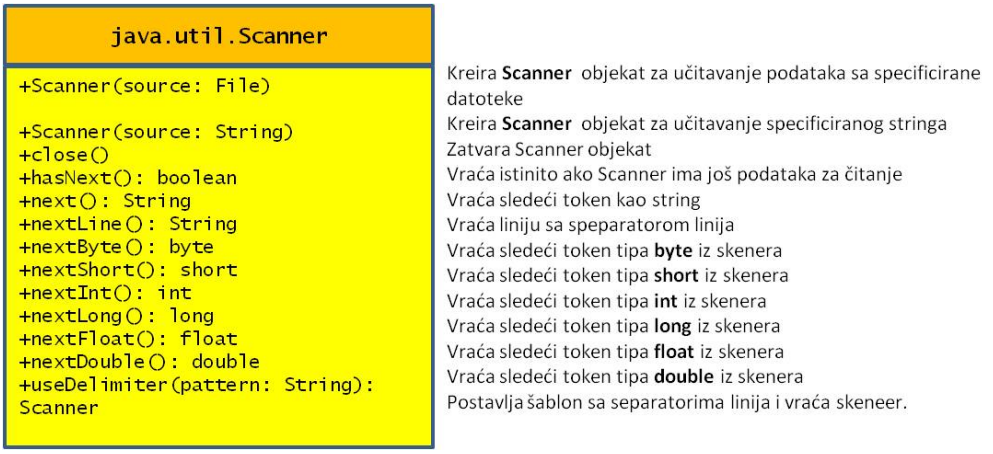
Klasa ***java.util.Scanner*** se upotrebljava za čitanje stringova i primitivnih vrednosti sa konzole. Klasa ***Scanner*** deli ulaz u tokene (engl. tokens) između kojih se nalazi po jedno blanko polje. Da bi učitavali podatke sa tastature treba da kreirate ***Scanner*** objekat za ***System*** objekat (koji opisuje vaš sistem):

```
Scanner input = new Scanner(System.in);
```

Ako želite da učitate tekstualne podatke sa neke datoteke, koristite konstruktor klase ***Scanner***, ali koji koristi datoteku:

```
Scanner input = new Scanner(new File(filename));
```

Na slici 2 su prikazani najčešće korišćeni metodi klase ***Scanner.***



Slika 2.2 Atributi i metodi klase Scanner

PRIMER 4 - UČITAVANJE TEKSTUALNIH PODATAKA

Klasa ReadData kreira objekat klase Scanner koji učitava tekstualne podatke iz datoteke scores.txt

Listing klase **ReadData** prikazuje primer kreiranja objekta klase ***Scanner*** i učitavanje podataka iz datoteke **scores.txt**.

```
1 import java.util.Scanner;
  2
  3 public class ReadData {
  4   public static void main(String[] args) throws Exception {
  5  // Kreiranje File objekta
  6  java.io.File file = new java.io.File("scores.txt");
  7
  8  // Kreiranje Scanner objekta za File objekat
  9  Scanner input = new Scanner(file);
 10
    // Učitavanje podataka sa datoteke
 12  while (input.hasNext()) {
 13    String firstName = input.next();
 14    String mi = input.next();
 15    String lastName = input.next();
 16    int score = input.nextInt();
 17    System.out.println(
 18      firstName + " " + mi + " " + lastName + " " + score);
 19  }
 20
 21  // Zatvaranje datoteke
 22  input.close();
 23   }
 24 }
```

```
```

Iskazom **new Scanner(String**) kreira se ***Scanner*** objekat za unos stringa sa tastature. Međutim iskazom **new Scanner(file)** kreira se ***Scanner*** objekat (linija 9) koji učitava tekstualne podatke iz prethodno kreiranog File objekta (linija 6).

Pozivanjem konstruktora sa **new Scanner(File)** može se izbaciti U/I izuzetak, jer je je metod ***main*** objavio ***trows Exception*** u liniji 4.

Svaka iteracija unutar ***while*** petlje čita ime, srednje slovo, prezime i ocenu iz tekstualne datoteke **scores.txt** (linije 12-19). Datoteka se zatvara u liniji 22. Zatvaranje datoteke nije obavezno, ali to predstavlja dobru praksu da bi se oslobodili zauzeti resursi tom datotekom.

UNOS TEKSTUALNIH PODATAKA PREKO DATOTEKE

Unos tekstualnih podataka sa datoteteke, npr. test.txt, se vrši kreiranjem objekta Scanner(new File("test.txt")) i korišćenjem njegovih metoda kao što su nextInt(), nextFloat() i dr.

Metodi***Scanner*** klase: ***nextByte(), nextShort(), nextLong(), next(Float), nextDouble() i next()*** su metodi za čitanje tokena, jer čitaju tokene ograničene separatorima.

Po pravilu, separatori su prazna polja. Ako želite da definišete svoje separatore, to možete uraditi sa metodom ***useDelimiter(String regex).*** Navedeni metodi za čitanje tokena prvo preskaću separatore i čitaju token do sledećeg separatora. *Pročitan token se automatski konvertuju u vrednost* odgovarajućeg tipa (***byte, short, int, long, float*** ili double). Mod primena metoda ***next()*** ne vrši se automatska konverzija učitanog tokena. Ako on ne odgovara očekivanom tipu, javlja se izuzetak izvršenja ***java.util.InputMismatchException.***

Metopdi ***next()*** i ***nextLine()*** čitaju string. Metod ***next()*** čita string odvojen separatorima, a ***nextLine()*** čita liniju teksta koja se završava separatorom linija.

Sepratori linja zavise od operativnog sistem. U Windows OS to su \5\n I \n kod UNIX OS. Da bi utvrdili koji se separator koristi, možete koristiti sledeći iskaz:

```
String lineSeparator = System.getProperty("line.separator");
```

Ako koristite tastaturu, separator linje se generiše kada pritisnete dugme ENTER.

Metodi za čitanje tokena ne čitaju separatore tokena. Separator linje se čita sa ***nextLine(),*** ali se ne prikazuje. Radi ilustracije, uzmimo primer datoteke test.txt koja sadrži liniju:

34 567

Posle izvršenja sledećeg dela programa (koda):

```
Scanner input = new Scanner(new File("test.txt"));
int intValue = input.nextInt();
String line = input.nextLine();
```

dobija vrednost za intValue je 34, a line sadrži karaktere ‘ ‘,5,6, i 7.

UNOS TEKSTUALNIH PODATAKA PREKO TASTATURE

Unos podataka preko tastature se vrši kreiranjem objekta Scanner(System.in) i korišćenje njegovih metoda kao što su: nextInt(), nextFloat(), nextLine() i dr.

Ako se tekstualni podaci unose preko tastature onda se posale svakog unosa podatka pritiska dugme ENTER. Na premer, unesete 34 i pritisnete dugme ENTER, onda otkucate 567 i pritisnite ENTER dugme da bi se izvršio sledeći deo programa:

```
Scanner input = new Scanner(System.in);
 int intValue = input.nextInt();
 String line = input.nextLine();
```

displays

The sum is 27

PRIMER 5 - ZAMENA TEKSTA

Primenom objekta klase Scanner učitavaju se novi tekstualni podataci sa izvorne datoteke, a sa objektom klase PrintWriter se ovi podaci upisuju u ciljnu datoteku, koja je imala neki stari tekst.

Pretpostavimo da treba da napišete program **ReplaceText** koji treba da zameni sva javljanja nekog stringa u tekstualnoj datoteci sa novim stringom. Ime datoteke, kao i nazivi stringova unose se kao argumenti komandne linije, u sledećoj formi:

```
java ReplaceText sourceFile targetFile oldString newString
```

Na primer, ako se unese sledeća komandna linija:

```
java ReplaceText FormatString.java t.txt StringBuilder StringBuffer
```

zamenjuje sve pojave stringa: StringBuilder sa stringom **StringBuffer** u datoteci **FormatString.java** i tako izmenjen sadržaj smešta u datoteku **t.txt.**

Sledeći listing klase **ReplaceText** sadrži program koji ovaj zadatak ostvaruje. Program proverava broj argumenata koji se predaje metodu ***main*** (linije 7-11), proverava da li postoje izvorna i ciljna datoteka (linije 14-25), kreira ***Scanner*** objekat sa izvornu datoteku (linija 28), kreira ***PrintWriter*** objekat za ciljnu datoteku, i čita, sa ponavljanjem, iz ciljne datoteke (linija

32), zamenjuje tekst (linija 33) i piše novu liniju na ciljnoj datoteci (linija 34). Na kraju, zatvara se izlazna datoteka (linija 38) da bi se obezbedilo da su podaci memorisani kako treba. U normalnim okolnostima, program se završava posle kopiranja datoteke. Program ne završava normalno ako argumenti na komandnoj liniji nisu uneti na odgovarajući način (linije 7-11), ako izvorna datoteka ne

postoji (linije 14-18) ili ako ciljna datoteka već postoji (linije 22.25). Izlazni status 1,2 i 3 se koristi da bi se ukazalo na ove slučajeve nenormalnog završetka programa (linije 10,17 i 24).

```
1 import java.io.*;
  2 import java.util.*;
  3
  4 public class ReplaceText {
  5   public static void main(String[] args) throws Exception {
  6  // Provera upotrebe argumenata komandne linije
  7  if (args.length != 4) {
  8     System.out.println(
  9     "Usage: java ReplaceText sourceFile targetFile oldStr newStr");
 10     System.exit(1);
 11  }
 12
 13  // Provera da li postoji izvorna datoteka
 14  File sourceFile = new File(args[0]);
 15  if(!sourceFile.exists() ) {
 16    System.out.println("Source file " + args[0] + " does not exist");
 17    System.exit(2);
 18  }
 19
 20  // Provera da postoji ciljna datoteka
 21  File targetFile = new File(args[1]);
 22  if( targetFile.exists()) {
 23    System.out.println("Target file " + args[1] + " already exists");
 24    System.exit(3);
 25  }
 26
 27  // Kreiranje Scanner objekta za ulaz u PrintWriter objekta za izlaz
 28    Scanner input = new Scanner(sourceFile);
 29    PrintWriter output = new PrintWriter(targetFile);
 30
 31  while(input.hasNext()) {
 32    String s1 = input.nextLine();
 33    String s2 = s1.replaceAll(args[2], args[3]);
 34    output.println(s2);
 35  }
 36
 37  input.close();
 38  output.close();
 39   }
 40 }
```

PRIMER 6

Cilj ovog zadatka je da se pokaže primer koji sadrži i čitanje i upis u fajl.

Napisati program koji upisuje u tekstualni fajl brojeve od 1 do 10 tako da svaki broj bude jedan ispod drugog. Nakon ovoga učitati brojeve iz fajla, smestiti ih u niz brojeva, a potom svaki broj pomnožiti sa 5 i ponovo smestiti u fajl.

Za čitanje tekstualnog fajla možemo koristiti BufferedReader koji smo pominjali do sada kada smo čitali podatke iz konzole. U konstruktor BufferedReader-a stavljamo FileReader i prosleđujemo mu ime fajla za učitavanje.
While petlja nam omogućava da čitamo svaku liniju fajla posebno i da je dodamo kao Integer u niz koji potom modifikujemo i pomoću PrintWriter-a upisujemo ponovo.

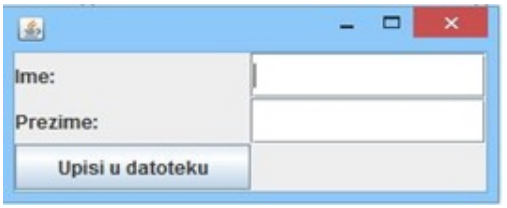
```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package zadatak2;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;

/**
 *
 * @author razvoj
 */
public class Zadatak2 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        new Zadatak2();
    }

    public void upisiUFajlBrojeve() {
        //Upisivanje brojeva u fajl
        PrintWriter upisUFajl;
        try {
            upisUFajl = new PrintWriter("brojevi.txt");
```

```
            for (int i = 1; i <= 10; i++) {
                upisUFajl.println(i);
            }
            upisUFajl.close();
        } catch (FileNotFoundException e) {
            System.out.println("Došlo je do greške");
        }
    }

    public Zadatak2() {
        upisiUFajlBrojeve();
        int[] array = ucitajBrojeveUNiz();
        upisiNizUFajl(array);

    }

    public void upisiNizUFajl(int[] array) {
        PrintWriter upisUFajl;
        try {
            upisUFajl = new PrintWriter("brojevi.txt");
            for (int i = 0; i < array.length; i++) {
                upisUFajl.println(array[i] * 5);
            }
            upisUFajl.close();
        } catch (FileNotFoundException e) {
            System.out.println("Došlo je do greške");
        }
    }

    public int[] ucitajBrojeveUNiz() {
        int[] array = new int[10];
        BufferedReader br;
        try {
            br = new BufferedReader(new FileReader("brojevi.txt"));
            String line;
            int i = 0;
            while ((line = br.readLine()) != null) {
                array[i] = Integer.parseInt(line);
                i++;
            }
            br.close();
        } catch (FileNotFoundException e) {
            System.out.println("Došlo je do greške");
        } catch (IOException ex) {
            System.out.println("Došlo je do greške");
        }
        return array;
    }
}
```

PRIMER 7

Cilj ovog zadatka je da se prikaže upis u fajl podataka koji se unesu preko grafičkog interfejsa.

Napraviti program koji sadrži dva polja za unos. U jedno polje treba da se upiše ime, dok u drugo prezime. Ime i prezime upisati u tekstualan fajl. Program treba da izgleda kao na sledećoj slici.



Slika 2.3 - Grafički interfejs trećeg zadatka

Klasa Osoba:

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package zadatak3;

/**
 *
 * @author razvoj
 */
public class Osoba {

    private String ime;
    private String prezime;

    public Osoba(String ime, String prezime) {
        this.ime = ime;
        this.prezime = prezime;
    }

    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;
    }

    @Override
    public String toString() {
        return ime + " " + prezime;
    }

}
```

Klasa Util:

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package zadatak3;

import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

/**
 *
 * @author razvoj
 */
public class Util {

    public static void upisiUFajl(Osoba o) {
        PrintWriter upisUFajl;
        try {
            upisUFajl = new PrintWriter(new FileWriter("osobe.txt", true));
            upisUFajl.println(o);
            upisUFajl.close();
        } catch (FileNotFoundException e) {
            System.out.println("Došlo je do greške");
        } catch (IOException ex) {
            System.out.println("Došlo je do greške");
        }
    }
}
```

Klasa Zadatak3:

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
```

```
 * and open the template in the editor.
 */
package zadatak3;

import java.awt.GridLayout;
import java.awt.event.ActionListener;
import javafx.event.ActionEvent;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;

/**
 *
 * @author razvoj
 */
public class Zadatak3 extends JFrame {

    JButton btnSubmit = new JButton("Upisi u datoteku");
    JLabel lblIme = new JLabel("Ime:");
    JTextField txtIme = new JTextField();
    JLabel lblPrezime = new JLabel("Prezime:");
    JTextField txtPrezime = new JTextField();

    public static void main(String[] args) {
        new Zadatak3();
    }

    public Zadatak3() {
        setSize(300, 130);
        setLayout(new GridLayout(3, 2));
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        setVisible(true);
        initComponents();
        initListeners();
    }

    public void initComponents() {
        add(lblIme);
        add(txtIme);
        add(lblPrezime);
        add(txtPrezime);
        add(btnSubmit);
    }

    public void initListeners() {
        btnSubmit.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(java.awt.event.ActionEvent e) {
                Util.upisiUFajl(new Osoba(txtIme.getText(), txtPrezime.getText()));
                JOptionPane.showMessageDialog(rootPane, "Uspešno ste upisali
```

```
osobu");
                txtIme.setText("");
                txtPrezime.setText("");
            }
        });
    }
}
```

PRIMER 8

Cilj ovog zadatka je da prikaže mogućnosti obrade teksta u fajlovima putem Jave.

Treba napraviti program koji učita fajl textzaobradu.txt u String, a potom zameni svuda reč tekst sa KI103. Dobijeni tekst ponovo smestiti u fajl textzaobradu.txt. Pre pokretanja programa napraviti textzaobradu.txt u folderu projekata i napisati tekst koji sadrži reč tekst. Proveriti da li je posle izvršenja programa tekst promenjen.

Objašnjenje:

Koristimo BufferedReader za čitanje teksta, dok za upis klasu PrintWriter. Zamena teksta se vrši metodom replace, String klase.
Na kraju svake učitane linije bitno je da u String dodamo novi red (\r\n).

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package zadatak7;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintWriter;
import sun.applet.Main;

/**
 *
 * @author razvoj
 */
public class Zadatak7 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        new Zadatak7();
```

```
    }

    public Zadatak7() {
        BufferedReader br;
        String fullText = "";
        try {
            br = new BufferedReader(new FileReader("tekstzaobradu.txt"));
            String line;
            int i = 0;
            while ((line = br.readLine()) != null) {
                fullText += line + "\r\n";
            }
            br.close();
            fullText = fullText.replace("tekst", "KI103");
        } catch (FileNotFoundException e) {
        } catch (IOException ex) {
        }

        PrintWriter upisUFajl;
        try {
            upisUFajl = new PrintWriter("tekstzaobradu.txt");
            upisUFajl.append(fullText);
            upisUFajl.close();
        } catch (FileNotFoundException e) {
            System.out.println("Došlo je do greške");
        }

    }
}
```

PRIMER 9

Ovaj zadatak služi za demonstriranje čitanja teksta iz datoteke kao i upis teksta u datoteku

Napraviti klasu FileUtil koja će služiti za rad sa datotekom tako što će datoteka sa kojom će se raditi biti atribut klase. Koristiti klase BufferedReader i BufferedWriter za rad sa datotekom. Pomenute klase se takođe mogu koristiti kao atributi klase FileUtil. U okviru klase potrebno je implementirati sledeće metode:

-metodu koja prima kao parametar string koji se treba upisati u datoteku

-metodu koja čita ceo sadržaj datoteke i vraća ga kao jedan string

-metodu koja čita prvih n karaktera iz datoteke

-setter za datoteku sa kojom se radi

FileUtil klasa:

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package zadatak8;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

/**
 *
 * @author razvoj
 */
public class FileUtil {

    private File file;
    private BufferedWriter bw;
    private BufferedReader br;

    // konstruktor koji prima kao parametar file nad kojim ce se vrsiti metode
    public FileUtil(File file) throws IOException {
        this.bw = new BufferedWriter(new FileWriter(file));
        this.br = new BufferedReader(new FileReader(file));
        this.file = file;
    }

    // prima kao parametar string koji treba upisati u datoteku
    public void writeInFile(String text) throws IOException {
        bw.write(text);
        bw.flush();
    }

    // cita sav sadrzaj iz datoteke i vraca ga kao string
    public String readFromFile() throws IOException {
        String line = null;
        String result = "";
        while ((line = br.readLine()) != null) {
            result += line + "\r\n";
        }
        resetReader();
        return result;
    }

    // cita prvih n karaktera iz datoteke
    public String readCharacters(int n) throws IOException {
        br.mark(n);
        int r = 0;
```

```
        char tmpChr = 0;
        String result = "";
        while (r < n) {
            tmpChr = (char) br.read();
            result += tmpChr;
            r++;
        }
        br.reset();
        return result;
    }

    public void resetReader() throws FileNotFoundException {
        br = new BufferedReader(new FileReader(file));
    }

    public void setFile(File file) {
        this.file = file;
    }
}
```

Main klasa:

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package zadatak8;

import java.io.File;
import java.io.IOException;

/**
 *
 * @author razvoj
 */
public class Zadatak8 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws IOException {
        // TODO code application logic here
        new Zadatak8();
    }

    public Zadatak8() throws IOException {
        File file = new File("text.txt");
        FileUtil fu = new FileUtil(file);
        fu.writeInFile("Hello world!");
        System.out.println(fu.readFromFile());
        System.out.println(fu.readCharacters(5));
    }
```

```
}
```

PRIMER 10

U ovom zadatku će se obnoviti obrada stringa i provežbati rad sa datotekom.

U prethodno kreiranu klasu FileUtil dodati metodu koja prebrojava koliko karaktera ima u datoteci kao i metodu koja prebacuje sva velika slova u mala i obrnuto i tako izmenjen tekst dodaje u datoteku. Za izradu ovog zadatka treba koristiti metode koje ste već napisali u prvom zadatku.

```
package zadatak9;

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
import zadatak8.*;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

/**
 *
 * @author razvoj
 */
public class FileUtil {

    private File file;
    private BufferedWriter bw;
    private BufferedReader br;

    // konstruktor koji prima kao parametar file nad kojim ce se vrsiti metode
    public FileUtil(File file) throws IOException {
        this.bw = new BufferedWriter(new FileWriter(file));
        this.br = new BufferedReader(new FileReader(file));
        this.file = file;
    }

    // prima kao parametar string koji treba upisati u datoteku
    public void writeInFile(String text) throws IOException {
        bw.write(text);
        bw.flush();
```

```
    }

    // cita sav sadrzaj iz datoteke i vraca ga kao string
    public String readFromFile() throws IOException {
        String line = null;
        String result = "";
        while ((line = br.readLine()) != null) {
            result += line + "\r\n";
        }
        resetReader();
        return result;
    }

    // cita prvih n karaktera iz datoteke
    public String readCharacters(int n) throws IOException {
        br.mark(n);
        int r = 0;
        char tmpChr = 0;
        String result = "";
        while (r < n) {
            tmpChr = (char) br.read();
            result += tmpChr;
            r++;
        }
        br.reset();
        return result;
    }

    public void resetReader() throws FileNotFoundException {
        br = new BufferedReader(new FileReader(file));
    }

    public void setFile(File file) {
        this.file = file;
    }

    // broji karaktere u datoteci
    public int countCharacters() throws IOException {
        int charCount = 0;
        while (br.read() > -1) {
            charCount++;
        }
        resetReader();
        return charCount;
    }

// zamenjuje mala slova velikim i obrnuto u datoteci
    public void replaceUpperLowerCase() throws IOException {
        String original = readFromFile();
        String result = "\r\n" + upperLowerStrReplace(original);
        bw.write(result);
        bw.flush();
        resetReader();
```

```
    }

    // zamenjuje velika i mala slova u string-u
    private String upperLowerStrReplace(String str) {
        for (int i = 0; i < str.length(); i++) {
            char tmpChar = str.charAt(i);
            if (Character.isUpperCase(tmpChar)) {
                tmpChar = Character.toLowerCase(tmpChar);
            } else if (Character.isLowerCase(tmpChar)) {
                tmpChar = Character.toUpperCase(tmpChar);
            }
            str = str.substring(0, i) + tmpChar + str.substring(i + 1);
        }
        return str;
    }
}
```

Main klasa:

```
package zadatak9;

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
import zadatak8.*;
import java.io.File;
import java.io.IOException;

/**
 *
 * @author razvoj
 */
public class Zadatak9 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws IOException {
        // TODO code application logic here
        new Zadatak9();
    }

    public Zadatak9() throws IOException {
        File file = new File("text.txt");
        FileUtil fu = new FileUtil(file);
        fu.writeInFile("Hello world!");
        fu.replaceUpperLowerCase();
        System.out.println(fu.readFromFile());
        System.out.println(fu.countCharacters());

    }
```

```
}
```

ZADACI 2.1 - 2.7

Proverite vaše razumevanje pređenog gradiva "Ulaz i izlaz tekstualnih datoteka".

1. Kako kreirate **PrinterWrite**r objekat radi zapisivanja podataka u datoteku? Koji je razlog objave ***throws Exception*** u metodu ***main*** u listingu klase **WriteData.java** ? Šta bi se desilo ako se ne bi pozvao metod ***close()*** u klasi **WriteData?**

2. Prikaži sadržaj datoteke **temp.txt** posle izvršenja sledećeg programa:

```
public class Test {
  public static void main(String[] args) throws Exception {
    java.io.PrintWriter output = new
    java.io.PrintWriter("temp.txt");
    output.printf("amount is %f %e\r\n", 32.32, 32.32);
    output.printf("amount is %5.4f %5.4e\r\n", 32.32, 32.32);
    output.printf("%6b\r\n", (1 > 2));
    output.printf("%6s\r\n", "Java");
    output.close();
  }
}
```

3. Kako kreirate***Scanner*** objekt za čitanje podataka iz datoteke? Koji je razlog da se definiše ***throws Exception*** u glavnom metodu klase **ReadData.jav**a ? Šta će se desiti ako se ne pozove metod ***close()*** u tom listingu?

4. Šta će se desiti ako pokušate da kreirate***Scanner*** objekat za datoteku koja ne postoji? Šta će se desiti ako pokušate da kreirate ***PrintWriter*** objekat za postojeću datoteku?

5. Da li je separator linije isti za sve platforme? Kako izgleda separator za Windows OS?

6. Pretpostavimo da želite da unesete 45 57.8 789 i da onda pritisnete ENTER dugme. Pokažite sadržaj promenljivih posle izvršenja sledećeg koda:

```
Scanner input = new Scanner(System.in);
 int intValue = input.nextInt();
 double doubleValue = input.nextDouble();
 String line = input.nextLine();
```

7. Pretpostavimo da unosite 45, pritiskate ENTER, 57.8, prisikate ENTER i 789 i pritiskate ENTER dugme. Pokažite sadržaj promenljivih posle izvršenja sledećeg koda:

```
Scanner input = new Scanner(System.in);
 int intValue = input.nextInt();
 double doubleValue = input.nextDouble();
 String line = input.nextLine();
```

ČITANJE TEKSTUALNIH PODATAKA IZ DATOTEKE (VIDEO)

Video pokazuje kako u program da ubacim aalvanumeričke podatke iz neke datoteke u našu Java aplikaciju

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

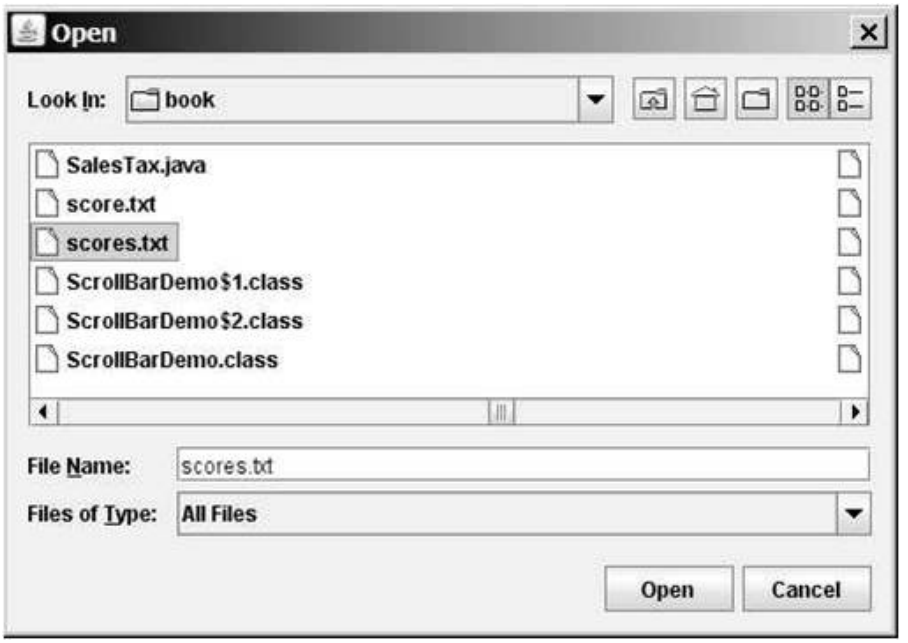
Poglavlje 3

Izbor raspoloživih datoteka

KLASA JFILECHOOSER

Klasa JFileChooser daje prikaz raspoloživih datoteka u računarskom sistemu.

Java obezbeđuja ***klasu javax.swing.JFileChooser*** za prikaz mogućih datoteka radi izbora, kao što je prikazano na slici 1. Iz ovog dijalog polja, korisnik može izabrati jednu od ponuđenih datoteka. U narednom izlaganju biće razvijen konkretan primer za demonstraciju rada sa klasom JFileChooser.



Slika 3.1 JFileChoose prikazuje raspoložive datoteke radi izbora i otvaranja datoteke

PRIMER 11 - PRIMENA KLASE JFILECHOOSER

JFileChoose prikazuje raspoložive datoteke radi izbora i otvaranja datoteke

Program kreira **JFileChosse**r objekat u liniji 6. Metod showOpenDialog(null) prikazuje dijalog boks, prikazan na slici 1. Metod vraća jednu int vrednost, ili APPROVE_OPTION ili CANCER_OPTION, koja ukazuje da li je izabrano dugme OPEN ili dugme CANCEL .Metod **getSelectedFile()** (linija 10) vraća izabranu datoteku iz dijalog boksa. Linija 13 kreira skener

objekat za datoteku. Program kontinualno čita linije iz datoteke i prikazuje ih na konzoli (linije 16-18).

```
1import java.util.Scanner;
  2import javax.swing.JFileChooser;
  3
  4 public class ReadFileUsingFileChooser {
  5   public static void main(String[] args) throws Exception {
  6  JFileChooser fileChooser = new JFileChooser();
  7  if (fileChooser.showOpenDialog(null
  8      == FileChooser.APPROVE_OPTION) {
  9    // Preuzmi izabranu datoteku
 10    java.io.File file = fileChooser.getSelectedFile();
 11
 12    // Kreiranje Scanner objekta za datoteku
 13    Scanner input = new Scanner(file);
 14
 15    // Čitanje teksta iz datoteke
 16    while (input.hasNext()) {
 17      System.out.println(input.nextLine());
 18    }
 19
 20    // Zatvaranje datoteke
 21    input.close();
 22  }
 23  else {
 24    System.out.println("No file selected");
 25  }
 26   }
 27 }
```

ZADATAK 3

Samostalan rad sa FileChooser klasom

Kako kreirate **FileOpen** dijalog boks?

Šta se vraća pozivom metoda **getSelectedFile()** na **JFileChooser** objektu?

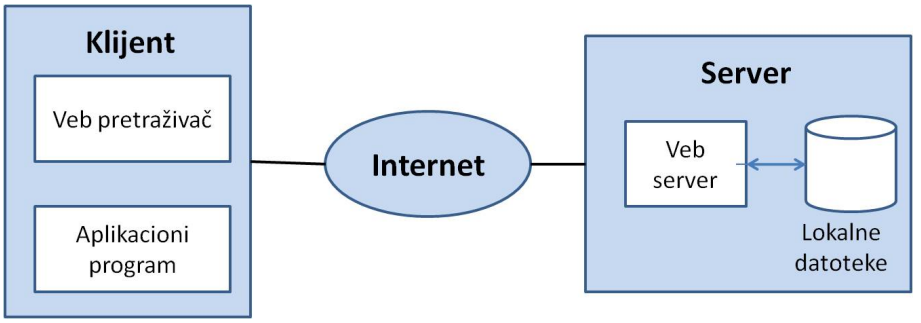
Poglavlje 4

Čitanje podataka sa veba

ČITANJE DATOTEKA SA VEBA

Kao što možete čitati podatke sa datoteke na vašem kompjuteru, možete čitati iz datoteke koja se nalazi na Vebu.

Kao što možete čitati podatke sa neke datoteke na vašem kompjuteru, tako možete čitati podatke i iz datoteke koja se nalazi negde na Internetu, tj. na vebu, ako vam je poznata adresa gde se datoteka nalazi, tj. ***URL (Uniform Resource Locator)***. To je jedinstvena adresa neke datoteke na Vebu. Kada unesete ***URL*** u vašem veb pretraživaču, Veb server šalje podatke u vaš pretraživač, što grafički izražava podatke (slika 1).



Slika 4.1 Klijent prima datoteke sa veb servera

Da bi ste učitali podatke URL, morate prvo da kreirate URL objekat upotrebom java.net.URL klase sa ovim konstruktorom.

```
public URL(String spec) throws MalformedURLException
```

Na primer, sledeći iskazi kreiraju URL objekat za http://www.google.com/index.html.

```
1 try {
2   URL url = new URL("http://www.google.com/index.html");
3 }
4 catch (MalformedURLException ex) {
5   ex.printStackTrace();
6 }
```

Izuzetak ***MalformedURLException*** je izbačen ako ***URL*** sintaksa ima grešku. Posle kreiranja ***URL*** objekta , možete da pozovete metod openStream() definisan u***URL*** klasi za otvaranje ulaznog toka i za upotrebu ovog toka za kreiranje ***Scanner*** objekta:

```
Scanner input = new Scanner(url.openStream());
```

PRIMER 12 - ČITANJE DATOTEKA SE URL ADRESE

Klasa Scanner omogućava i čitanje datoteka sa veba, tj. sa neke URL adrese.

Sada možete da učitate podatke sa ulaznog toka isto kao i sa lokalne datoteke. Primer listing klase **ReadFileFromURL** pita korisnika da unese URL i prikazuje veličinu datoteke.

```
1 import java.util.Scanner;
  2
  3 public class ReadFileFromURL {
  4   public static void main(String[] args) {
  5     System.out.print("Enter a URL: ");
  6     String URLString = new Scanner(System.in).next();
  7
  8     try {
  9     java.net.URL url = new java.net.URL(URLString);
 10     int count = 0;
 11     Scanner input = new Scanner(url.openStream());
 12     while (input.hasNext()) {
 13         String line = input.nextLine();
 14         count += line.length();
 15     }
 16
 17     System.out.println("The file size is " + count + " bytes");
 18     }
 19     catch (java.net.MalformedURLException ex) {
 20     System.out.println("Invalid URL");
 21     }
 22     catch (java.io.IOException ex) {
 23         System.out.println("I/O Errors: no such file");
 24     }
 25   }
 26 }
```

Program pita korisnika da unese ***URL*** string (linja 6) i kreira***URL*** objekat (linija 9). Konstruktor izbacuje izuzetak java.net.***MalformedURLException*** (linija 19) ako ***URL*** adresa nije dobro definisana.

Program kreira ***Scanner*** objekat iz ulaznog toka za URL (linija 11). Ako je URL unet tačno, ali on ne postoji, izbacuje se izuzetak IOException (linija 22).

Slika 2 prikazuje dobijeni rezultat na konzoli u dva slučaja.

Slika 4.2 Rezultat unosa dve URL adrese za ulazne datoteke

ZADATAK 4

Čitanje podataka sa veba - samostalna provera znanja.

1. Kako kreirate Scanner objekat za čitanje teksta sa neke URL adrese?
2. Kreirajte kod kojim se preuzimaju podaci iz ulaznog toka.

Poglavlje 5

Ulaz-izlaz binarnih podataka

ČITANJE I ZAPISIVANJE TEKSTUALNIH PODATAKA

Java obezbeđuje više klasa koje izvršavaju U/I operacije sa tekstualnim i binarnim podacima. Klasa Scanner učitava, a klasa PrintWriter zapisuje tektualne podatke u datoreku.

Datoteke mogu biti ili tekstualne ili binarne.**Tekstualna datoteka** se može obrađivati (čitati, kreirati ili menjati) upotrebom editora tekstova, kao što je MS Notepad. Sve ostale datoteke su **binarne datoteke**. Binarne datoteke ne čitamo tekstualnim editorima, već direktno iz programa. Na primer, izvorn kod u Java zapisan u nekoj datoteki, je primer tekstualne datoteke,a datoteka sa Java fajlovima sa extenyijom class je primer binarne datoteke.

Tekstualne datoteke možemo zamisliti kao datoteke koje sadrže nizove oznaka koji se šifrirani pomoću šeme kodiranja ASCII ili Unicode. Binarne datoteke sadrže niz bitova. Zato što se ne vrši šifriranje binarnih podataka, rad sa binarnim datotekama je znatno brži nego rad sa tekstualnim datotekama.

Java ima mnogo klasa koje obezbeđuju ulaz i izlaz datoteka. One se mogu klasifikovati kao:

- Tekstualne U/I datoteke (npr. Scanner, PrintWriter)
- Binarne U/I datoteke.

Da bi se tekstualni podaci zapisli u datoteku temp.txt potrebno je kreirati objekat klase***PrintWriter***:

```
PrintWriter output = new PrintWriter("temp.txt");
```

Da bi sada ovako kreiran objekat zapisali kao string u datoteku potrebno je koristiti metod ***print()*** klase ***PrintWriter***. Na primer, ako želite da zapišete tekst “Java101” u ovu datoteku, treba da napišete iskaz:

```
output.print("Java 101");
```

Na kraju, sledećim iskazom treba da zatovorie datoteku:

```
output.close();
```

Postoji puno klasa za rad sa podacima. Mogu se por+deliti na dve kategorije:

1. *Ulazne klase*: sadrže metode za učitavanje podataka (npr. ***Scanner*** klasa)
2. *Izlazne klase*: sadrže metode za zapisivanje podataka (npr. ***PrintWriter*** klasa)

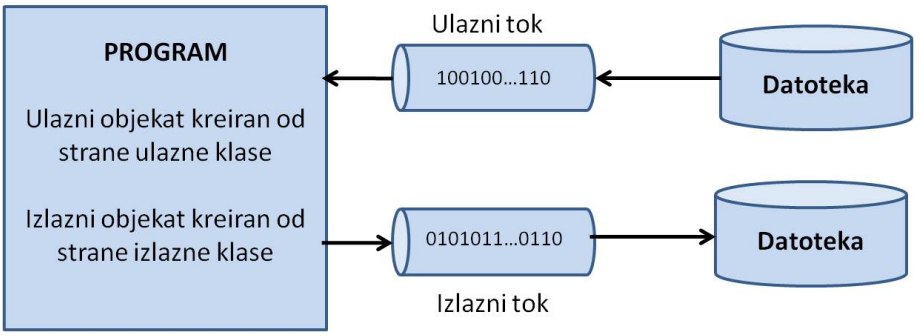
Sledeći kod kreira ulazni objekat ya datoteku temp.txt i čita podatke iz datoteke:

```
Scanner input = new Scanner(new File("temp.txt"));
System.out.println(input.nextLine());
```

UPOREĐENJE U/I TEKSTUALNIH I BINARNIH PODATAKA

Ulazni objekat učitava tok podataka sa binarne datoteke, a izlazni objekat piše tok podataka na binarnu datoteku.

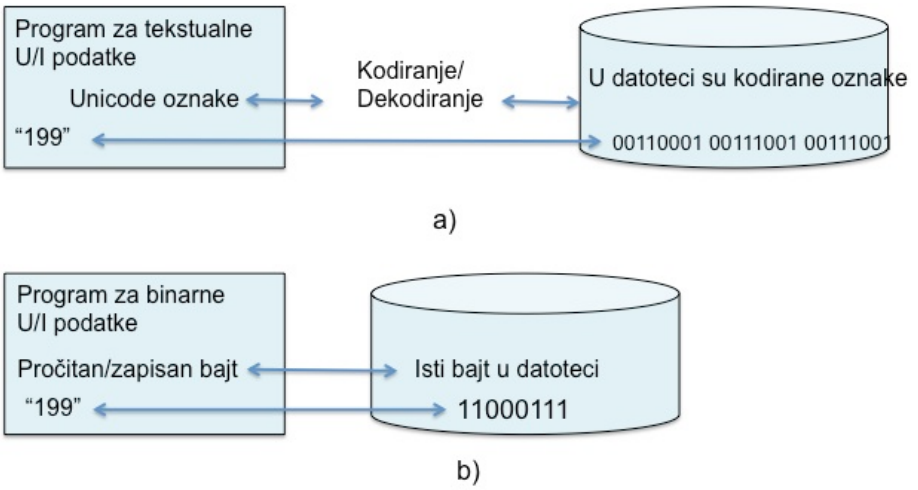
Na slici 1 prikazano je Java U/I programiranje. Ulazni objekat učitava tok podataka sa binarne datoteke, a izlazni objekat piše tok podataka na binarnu datoteku. Ulazni objekat se često naziva **ulaznim tokom,** a izlazni objekat **- izlaznim tokom.**



Slika 5.1 Program prima podatke preko ulaznog objekta, a šalje podatke preko izlaznog objekta

Sve datoteke sadrže samo binarne podatke. Tekstualni U/I podaci nastaju šfriranjem (encoding) binarnih U/I podataka (slika 2.a), a binarni U/I podaci nastaju dešifrivanjem (decoding) tekstualnog U/I podatka (slika 2.b). JVM convertuje Unicode tekst u određeni šifrirani binarni zapis, a kovertuje šifrirane binarne podatke u Unicode tekstualni zapis, tj. podatke.

Binarni U/I podaci se ne konvertuju. Ista vrednost se zapisuje i u glavnoj memoriji i na disku. Zato je mnogo brži prenos binarnih podataka u program i iz programa ka datoteci.



Slika 5.2 Tekstualni U/I podaci zahtevaju kodiranje i dekodiranje, a binarni to zahtevaju

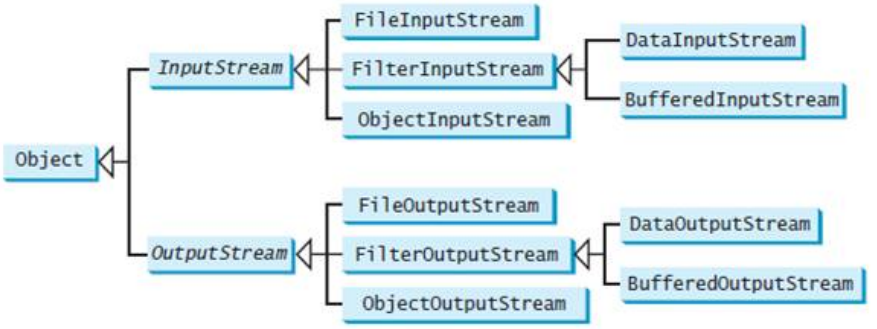
Binarne datoteke su nezavisne od šeme kodiranja računara, te su prenošljive. Java programi mogu čitati binarnu datoteku kreiranu na bilo kojoj mašini. Zato su **class** datoteke Jave binarne i mogu da se izvršavaju na JVM bilo koje mašine.

BINARNE U/I KLASE

Binarne U/I klase su Java klase koje obezbeđuju podršku binarnog ulaza i izlaza podataka, tj. ulaznog i izlaznog toka binarnih podataka.

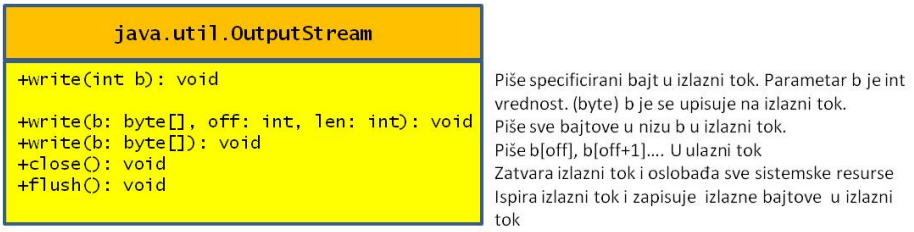
Tekstualne datoteke se koriste ako se koristi editor teksta ili neki program za generisanje teksta. Binarne datoteke se koriste za unos podataka proizvedenih od strane Java programom sa binarnim izlazom rezultata.

Slika 3 prikazuje neke od Java klasa koje obezbeđuju podršku binarnog ulaza i izlaza podataka. Klasa***InputStream (slika 4)*** je koren svih binarnih ulaznih klasa, klasa***OutputStream (slika 5)*** je koren sa sve binarne izlazne klase.

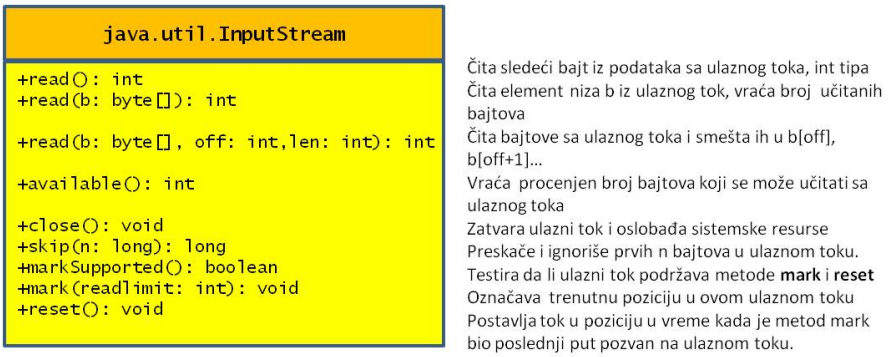


Slika 5.3 Struktura Java klasa koje podržavaju ulaz i izlaz binarnih podataka

Binarni podaci koji se učitavaju u program iz neke binarne datoteke čine **ulazni tok** podataka, a kada izlaze iz programa radi zapisa u datoteci čine **izlazni tok**.



Slika 5.4 Apstraktna klasa OutputStream

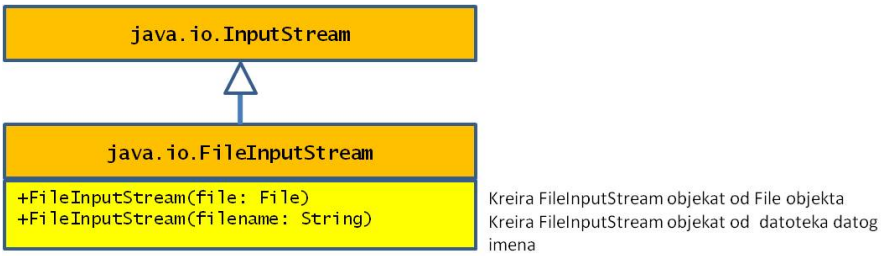


Slika 5.5 Apstraktna klasa InputtStream

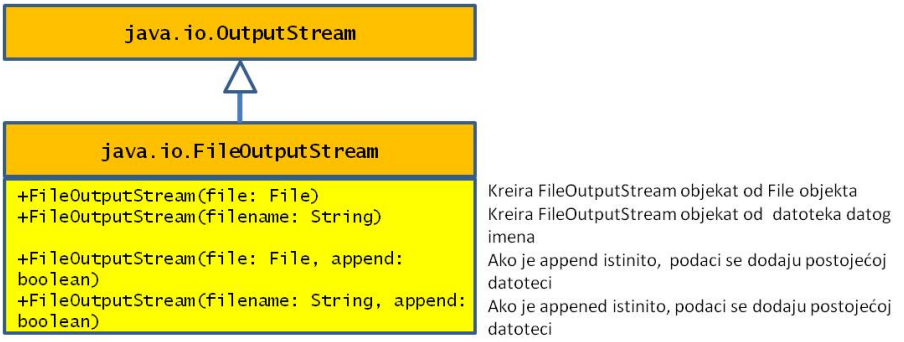
KLASE FILEINPUTSTREAM/FILEOUTPUTSTREAM

Klasa FileInputStream čita bajtove iz datoteka. Klasa FileOutputStream piše bajtove u datoteke

Klasa ***FileInputStream*** čita bajtove iz datoteka. Klasa ***FileOutputStream*** piše bajtove u datoteke. Sve metode ovih klasa su nasleđene od klasa ***InputStraem*** i ***OutputStream***. One nemaju svoje sopstvene metode. Kreiranje objekta klase ***FileInputStream*** se vrši konstruktorima prikazanim na slici 6, a objekta klase FileOutputStream – na slici 7.



Slika 5.6 Klasa FileInputStream unosi tok bajtova iz datoteke



Slika 5.7 Klasa FileOutputStream iznosi i tok bajtova u datoteku

Skoro svi metodi U/I klasa izbacuju izuzetak ***java.io.IOException***. Zato, treba da deklarišete ***java.io.IO Exception*** i da ga izbacite u metodu ili kodu u try-catch bloku, kao na slici 8.

Objava izuzetka u metodu

```
public static void main(String[] args)
    trows IPException {
   // Izvršava U/I operacije
}
```

Upotreba try-catch blok

```
public static void main(String[] args) {
  try {
    // Izvršava U/I operacije
  }
  catch (IOException ex) {
    ex.printStackTrace();
  }
}
```

Slika 5.8 Klasa FileInputStream čita bajtove iz datoteka. Klasa FileOutputStream piše bajtove u datoteke

PRIMER 13 - RADA SA TOKOVIMA BAJTOVA

Primer pokazuje upotrebu U/I binarnih operacija pisanja deset bajtova sa vrednostima od 1 do 10 u datoteku temp.dat i čita ih nazad iz datoteke.

Na slici 9 prikazan je listing u kome se upotrebljavaju U/I binarne operacije pisanja deset bajtova sa vrednostima od 1 do 10 u datoteku **temp.dat** i čita ih nazad iz datoteke.

```
import java.io.*;

  public class TestFileStream {
    public static void main(String[] args) {
    // Kreiranje izlaznog strima za datoteku
    FileOutputStream output = new FileOutputStream("temp.dat");

    // Izlazne vrednosti za datoteku
    for (int i = 1; i <= 10; i++)
        output.write(i);

    // Zatvara izlazni strim
    output.close();

    // Kreiranje ulaznog strima za datoteku
```

```
    FileInputStream input = new FileInputStream("temp.dat");

    // Čitaanje vrednosti iz datoteke
    int value;
    while ((value = input.read() ) != -1)
        System.out.print(value + " ");

    // Zatvaranje izlaznog strima
    input.close();
    }
}
```

Program posle izvršenja na monitoru prikazujesledeći rezultat: 1 2 3 4 5 6 7 8 9 10.

Objekat ***FileOutputStream*** je kreiran za datoteku **temp.dat** u liniji 6. Petlja for piše deset vrednosti u datoteku (linije 9-10). Pozivanjem **write(i)** ostvaruje se isti efekat kao i pozivanjem **write((byte)u**). Linija 13 zatvara izlazni tok. Linija 16 kreira objekat FileInputStream za datoteku **temp.dat.** Vrednosti se učitavaju iz datoteke i prikazuju na konzoli u linijama 19-21- Izraz ((value = input.read()) != -1) (linija 20) čita bajt iz input.read()., dodeljuje ga vrednosti i proverava da li je to -1. Ulazna vrednost -1 označava kraj datoteke.

Datoteka **temp.dat** kreirana u ovom primeru je binarna datoteka. Ona se može čitati korišćenjem nekog Java programa, ali ne i uz pomoć etitora teksta.

Kada tok nije više potrebam uvek ga zatcorite sa **close()** metodom. U suprotnom, može da dođe do mešanja podataka u izlaznoj datoteci ili javljanja drugi programerskih grešaka.

KLASE DATAINPUTSTREAM/DATAOUTPUTSTREAM

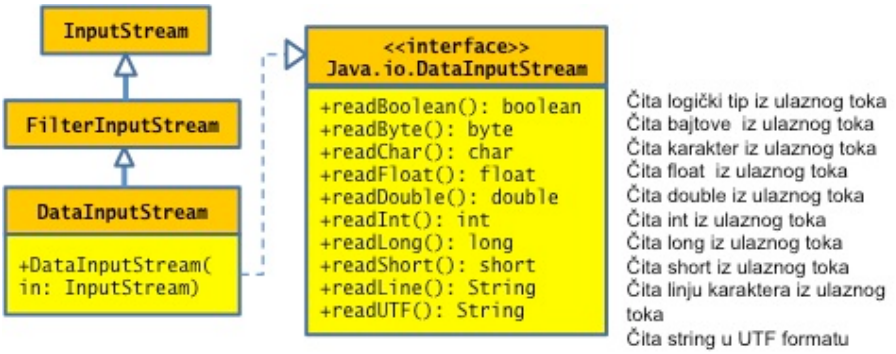
Kada treba da obrađujete primitivne brojčane tipove, upotrebite DataInputStream i DataOutputStream radi filtriranja bajtova.

Ponekad, poptrebno je filtrirate tokove podataka. Filter tokovi su tokovi koji fitriraju bajtove zbog nekog razloga. Osnovni ulazni tok za učitavanje bajtova je metod ***read()*** koji se koristi samo za čitanje bajtova. Ako želite da čitate samo cele brojeve (***int***), decimalne brojeve duple tačnosti (***double)*** i stringova umesto bajtova i karaktera potrebna vam je klasa koja bi filtrirala ulazni ili izlazni tok podataka.FilterInputStreami ***FilterOutputStream*** su osnovne klase za filtriranje podataka

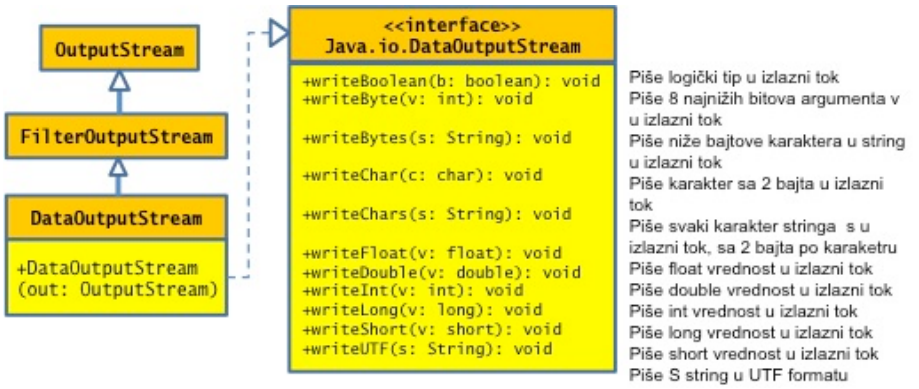
Kada treba da obrađujete primitivne brojčane tipove, upotrebite ***DataInputStream*** i ***DataOutputStream*** radi filtriranja bajtova. Klasa ***DataInputStream*** čita bajtove iz toka i kovertuje ih u odgovarajuće tipove vrednosti ili u stringove.

Klasa***DataOutputStream*** konvertuje primitivne tipove vrednosti ili stringove u bajtove i iznosi bajtove u tok.

Na slici 9 prikazana je struktura klasa za ulazne tokove i ***DataInput*** interfejs, a na slici 10, struktura klasa za izlazne tokove i ***DataOutput*** interfejs.



Slika 5.9 DataInputStream filtrira ulazni tok bajtova u primitivne tipove podataka o stringove



Slika 5.10 DataOutputStream omogućava pisanje primitivnih tipova vrednosti podataka u izlazni tok

METODI KLASA DATAINPUTSTREAM/ DATAOUTPUTSTREAM

Metodi klase DataInputStream i DataOutputStream pišu oznake ili bajtove u izlazni tok

Jedan Unicode karakter se sastoji od dva bajta- Metod ***writeChar(char c)*** piše Unicode karakter c u izlazni tok. Metod ***writeChars(String s)*** piše Unicode za svaki karakter (oznaku) stringa **S** u izlazni tok. Metod ***writeBytes(String s)*** niže bajtove Unicode za svaki karakter u stringu S u izlazni tok. Metod ***writeBytes()*** je odgovarajući za stringove koje čine ACII karakteri, jer je ASCII kod se memoriše samo sa nižim bajtovima Unicode-a. Ako string ima samo ne-ASCII karaktere, morate da koristite ***writeChars()*** metod za pisanje stringa.

Metod ***writeUTF(String s)*** piše dužinu informacije od dva bajta u izlazni tok, u skladu sa modifikovanim UTF-8 predstavljanjem svakog karaktera u stringu s. UTF-8 je šema kodiranja koja dozvoljava sistemima da rade i sa ASCII i sa Unicode. Najveći broj operativnih sistema upotrebljava ASCII. Java upotrebljava Unicode. Skup ASCII karaktera je pod-skup skupa Unicode karaktera. Kako najveći broj aplikacija ima potrebu samo za skupom ASCII karaktera, rasipanje resursa je da se 8-bitni ASCI karakter predstavi sa 16- bitnim Unicode karakterom. Modifikovana UTF-8 šema memoriše jedan karakter upotrebom jednog, dva ili tri bajta. Karakteri se kodiraju sa jednim bajtom ako je njihov kod manji od 0x7F, dva bajta ako je njihov kod veći od 0x7F a manji od 0x7FF, ili sa tri bajta ako je njihov kod veći od 0x7FF.

Početni bitovi UTF-8 karakter označavaju da li karakter se smešta u jednom bajtu, dva bajta ili tri bajta. Ako je prvi bit 0, onda je to karakter sa jednim karakterom. Ako su prvi bitovi 110, onda je to prvi bajt karaktera koji se predstavlja sa dva bajta. Ako su prvi bitovi 1110, onda su to prvi bitovi karaktera predstavljenog s atri bajta. Informacija koja označava broj karaktera u stringu je memorisan u prvom od dva bajta koji idu pre UTF-8 karaktera.

Metod ***writeUTF(String s)*** konvertuje string u niz bajtova u UTF-8 formatu i piše ih u izlazni tok. Metod ***readUTF()*** čita string koji je bio upisan upotrebom ***writeUTF()*** metoda**.**

UTF-8 format ima prednost u memorisanju bajtova za svaki ASCII karakter, jer jedan Unicode karakter zauzima dva bajta a jedan ASCII karakter sa UTF-8, zauzima samo jedan bajt. Ako najveći broj kataktera u jednom dugom stringu koriste ASCII karaktere, ond aje upotreba UTF-9 efikasniji (traži manje memorisjkog prostora,a time se povećava brzina obrade).

PRIMER 14 - ULAZ I IZLAZ STRIMOVA BAJATOVA

Primer prikazuje program koji piše imena studenata i njihove ocene u datoteku temp.dat i čita podatke nazad sa datoteke

Objekti klase ***DataInputStream*** i ***DataOutputStream*** se kreiraju upotrebom sledećih konstruktora

```
public DataInputStream(InputStream instream)
public DataOutputStream(OutputStream outstream)
```

Sledeći iskazi kreiraju tokove podataka. Prvi iskaz kreira ulazni tok za datoteku **in.dat**. Drugi iskaz kreira izlazni tok za datoteku **out.dat**:

```
DataInputStream input =
new DataInputStream(new FileInputStream("in.dat"));
DataOutputStream output =
new DataOutputStream(new FileOutputStream("out.dat"));
```

Sledeći listing prikazuje program koji piše imena studenata i njihove ocene u datoteku temd.dat i čita podatke nazad sa datoteke. Izvršenjem ovog programa dobija se sledeći rezultat na monitoru računara:

John 85.5

Susan 185.5

Kim 105.25

```
1 import java.io.*;
  2
  3 public class TestDataStream {
  4 public static void main(String[] args) throws IOException {
  5 // Kreiranje izlaznog toka za datoteku temp.dat
  6 DataOutputStream output =
  7   new DataOutputStream(new FileOutputStream("temp.dat"));
```

```
 8
 9  // Pisanje ocena studentskog testa u datoteku
10  output.writeUTF("John");
11  output.writeDouble(85.5);
12  output.writeUTF("Susan");
13  output.writeDouble(185.5);
14  output.writeUTF("Kim");
15  output.writeDouble(105.25);
16
17  // Zatvaranje izaznog toka
18  output.close();
19
20  // Kreiranje ulaznog toka za datoteku temp.dat
21  DataInputStream input =
22    new DataInputStream(new FileInputStream("temp.dat"));
23
24  // Čitanje ulaznog toka za datoteku temp.dat
25  System.out.println( + " " + );
26  System.out.println(input.readUTF() + " " + input.readDouble());
27  System.out.println(input.readUTF() + " " + input.readDouble());
28   }
29 }
```

PRIMER 15 - ZAUSTAVLJANJE ČITANJA DATOTEKE

Zaustavljanje čitanja ulazne datoteke se vrši izbacivanjem izuzetka EOFException kada se dođe do kraja datotetke.

U prethodnom primeru, Objekat ***DataOutputStream*** je kreiran za datoteku **temp.dat** u linijama 6-7. Imane studenta i ocene zapisuju se u datoteku u linijama 10-15. Linija 18 zatvara izlazni tok- Objekat ***DataInputStream*** za istu datoteku u linijama 21-22- Imena studenata i njihove ocene se učitacaju nazad iz datoteke i prikazan je na konzoli u linijama 25-27.

Objekti ***DataInputStream*** i ***DataOutputStream*** čitaju i pišu Java primitivne tipove vrednosti i stringove na način koji je nezavisan od računara, što vam omogućava da datoteku podataka formirate na jednom računaru, a koristite je na drugom računaru. Neka aplikacija može da upotrebljava izlazni tok podataka da bi upisala podatke koji se kasnije mogu čitati pomoću programa koji upotrebljava ulazni tok podataka. Važno je da se obrati pažnja da treba da učitavate podatke po istom redosledu i formatu u kome su i memorisani.

Ako čitate podatke sa neke datoteke, morate da regularno zaustavite izvršenje tog programa kada se dođe do kraja datoteke. Za to se koristi izuzetak koji ukazuje na kraj datoteke. Ovde se prikazuje listing ovakvog programa, koji daje sledeći rezultat:

4.5

43.25

3.2

All data were read

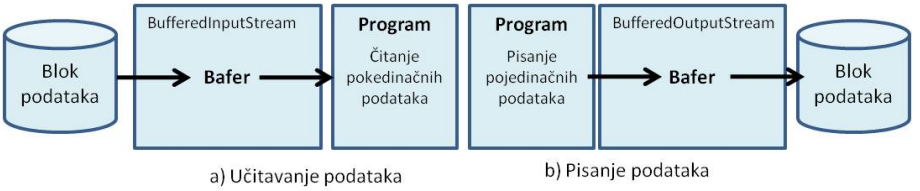
Program upisuje tri vrednosti tipa fouble u datoteku upotrebom DataOutputStream (linije 6-10) i čita podatke upotrebom DataInputStream (linije 13-14). Kada se pri učitavanju dođe do kraja datoteke, izbacuje se izuzetak tipa EOFException. Ovaj izuzetak se hvata u liniji 19.

```
1 import java.io.*;
  2
  3 public class DetectEndOfFile {
  4   public static void main(String[] args) {
  5  try {
  6    DataOutputStream output =
  7       new DataOutputStream(new FileOutputStream("test.dat"));
  8    output.writeDouble(4.5);
  9    output.writeDouble(43.25);
 10    output.writeDouble(3.2);
 11    output.close();
 12
 13    DataInputStream input =
 14       new DataInputStream(new FileInputStream("test.dat"));
 15    while (true) {
 16       System.out.println( );
 17    }
 18  }
 19  catch (EOFException ex) {
 20    System.out.println("All data were read");
 21  }
 22  catch (IOException ex) {
 23    ex.printStackTrace();
 24  }
 25   }
 26 }
```

KLASE BUFFEREDINPUTSTREAM/ BUFFEREDOUTPUTSTREAM

Klase BufferedInputStream/BufferedOutputStream obezbeđuju memorijski bafer pri unosu i izlazu tokova podataka što znatno ubrzava ove operacije.

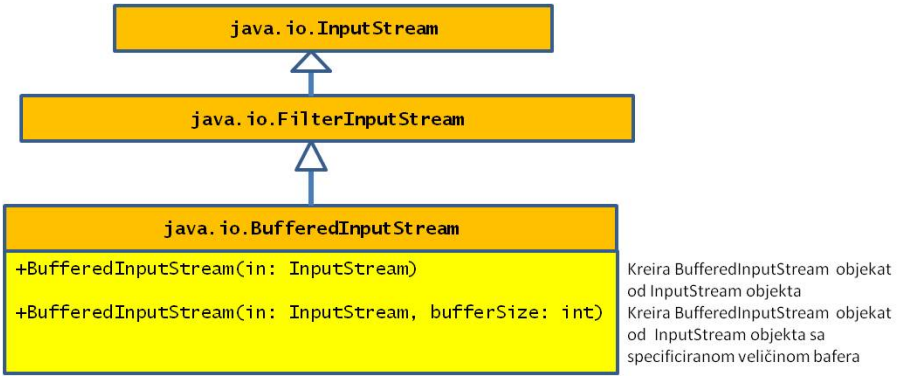
Objekti kasa ***BufferedInputStream/BufferedOutputStream*** se mogu koristiti da se ubrza ulaz i izlaz podataka na taj način što će se smanjiti borij čitanja sa diska ili pisanja na disku (jer je ta operacija relativno spora). Upotrebom objekta ***BufferedInputStream,*** ceo blok podataka sa diska se čita u privremeni memorijski prostor, tj. bafer (engl. buffer). Pojedinačni podaci se onda isporučuju programu iz ovog bafera, kao što je to prikazano na slici 11a. Upotrebom objekta ***BufferedOutputStream*** pojedinačni podaci se iz programa prvo smeštaju u bafer memoriju. Kada se on napuni, svi podaci iz bafer sa onda odjedanput upišu na disk, kao što je prikazano na slici 11b.



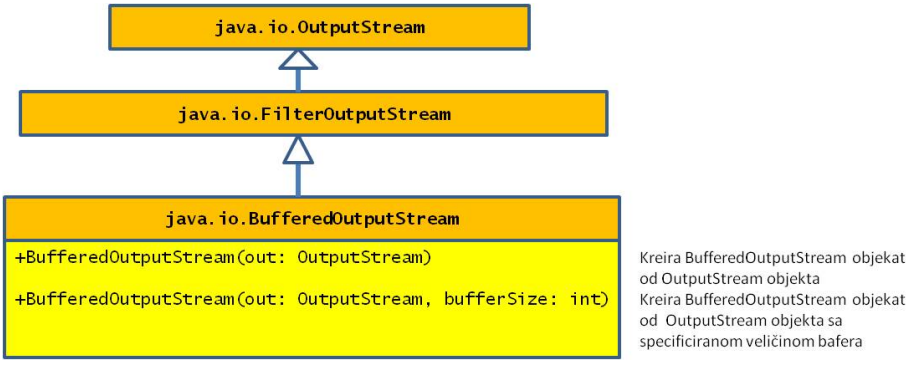
Slika 5.11 U/I baferi omogućavaju bržu obradu U/I podataka

Klase ***BufferedInputStream/BufferedOutputStream*** ne sadrže nove metode. Svi metoi ovih klasa su nasleđeni iz klasa ***InputStream/OutputStream***. Klase ***BufferedInputStream*** i ***BufferedOutputStream*** upravljaju korišćenjem bafera i obezbeđuju automatski čitanje i pisane podataka sa – i na disk.

Konstruktori klasa BufferedInputStream/BufferedOutputStream su prikazani na slikama12 i 13.



Slika 5.12 Baferi ulaznog toka upravljani klasom BufferedInputStream



Slika 5.13 Baferi izlaznog toka upravljani klasom BufferedOutputStream

KOJA VELIČINA BAFERA?

Baferi povećavaju brzinu unosa i izlaza tokova bajtova. Trebalo bi da uvek koristite bafere pri unosu i izlazu podataka u datoteke, naročito ako su veće od 100 MB.

Ako se ne specificira veličina bafera, onda se inicira unapred definisana vrednost od 512 bajta. Možete poboljšati performanse ranije prikayanog programa **TestDataStream** na dodavanjem bafera u tokove u linijama 6-7 i 21-22, na sledeći način:

```
DataOutputStream output = new DataOutputStream(
new BufferedOutputStream(new FileOutputStream("temp.dat")));
DataInputStream input = new DataInputStream(
new BufferedInputStream(new FileInputStream("temp.dat")));
```

Trebalo bi da uvek koristite bafere pri unosu i izlazu podataka u datoteke. Naročito pri radu sa većim datotekama, npr. od preko 100 MB, dobićete vidljivu povećanje efikasnosti u/I operacija.

PRIMER 16 - PROGRAM ZA KOPIRANJE DATOTEKE

Program treba da kopira izvornu datoteku u ciljnu datoteku i da prikaže broj bajtova u datoteci.

Ovim primerom želimo da pokažemo kako možete da napravite program koji će kopirati datoteke. Na komandnoj liniji računara, potrebno je da prvo obezbedite unos izvorne i ciljne datoteke:

java Copy izvornaDatoteka ciljnaDatoteka

Program treba da kopira izvornu datoteku u ciljnu datoteku i da prikaže broja bajtova u datoteci. Program bi trebalo da upozori korisnika ako izvorna datoteka ne postoji ili ako ciljna datoteka već postoji. Listing programa je ovde prikazan.

Program najpre provera da li je korisnik uneo dva zahtevana argumenta na komandnoj liniji (linije 10-14). Program koristi klasu ***File*** da proveri da li izvorna i ciljna datoteka postoje. Ako izvorna datoteka ne postoji (linije 18-22) ili ako ciljna datoteka već postoji (linije 15-30), program završava.

Ulazni tok se kreira upotrebom objekta ***BufferedInputStream*** ugnježden u klasi ***FileInputStream*** (linije 33-34) a izlazni tok je kreiran upotrebom objekta ***BufferOutputStream***, koji je ugnježden u ***FileOutputStream*** (linije 37-38).

Izraz: ((r = input.read()) != -1) (linija 42) čita bajt iz **input.read(),** dodeljuje ga promenljivoj r, i proverava da li je -1. Ova vrednost označava kraj datoteke. Program kontinualno čita bajtove iz ulaznog toka i šallje ih na izlazni tok, sve do svi bajtovi nisu proitani.

```
1 import java.io.*;
  2
  3 public class Copy {
  4   /** Main metod
  5   @param args[0] za izvornu datoteku
  6   @param args[1] za ciljnu datoteku
  7   */
  8   public static void main(String[] args) throws IOException {
  9  // Provera primene  parametara u komandnoj liniji
```

```
10  if (args.length != 2) {
11    System.out.println(
12      "Usage: java Copy sourceFile targetFile");
13    System.exit(1);
14  }
15
16  // Check whether source file exists
17  File sourceFile = new File(args[0]);
18  if (!sourceFile.exists()) {
19    System.out.println("Source file " + args[0]
20      + " does not exist");
21    System.exit(2);
22  }
23
24  // Provera da li ciljna datoteka već postoji
25  File targetFile = new File(args[1]);
26  if (targetFile.exists()) {
27     System.out.println("Target file " + args[1]
28       + " already exists");
29     System.exit(3);
30  }
31
32  // Kreiranje ulaznog toka podataka
33  BufferedInputStream input =
34    new BufferedInputStream(new FileInputStream(sourceFile));
35
36  // Kreiranje ulaznog toka
37   BufferedOutputStream output =
38      new BufferedOutputStream(new FileOutputStream(targetFile));
39
40  // Kontinualno čitanje bajtova sa ulaza i njegovo pisanje na izlazu
41  int r, numberOfBytesCopied = 0;
42    while ((r = input.read()) != -1) {
43      output.write((byte)r);
44      numberOfBytesCopied++;
45    }
46
47  // Zatvaranje tokova
48  input.close();
49  output.close();
50
51  // Prikaz veličine datoteke
52  System.out.println(numberOfBytesCopied + " bytes copied");
53   }
54 }
```

ZADACI 5.1- 5.10

Ulaz-izlaz binarnih podataka - proverite vaše razumevanje pređenog gradiva.

1. Da li morate da deklarišete izbacivanje IOException u metodu ili da upotrebite try-catch blok da bi obradili IOException za Java U/I programe?

2. Zašto bi trebalo uvek da zatvorite tokove podataka?

3. Metod read() u klasi InputStream čita bajtove. Zašto vraća podatak tipa int umesto byte? Nađite apstraktne metode u InputStream i OutputStream klasama.

4. Da li klase FileInputStream/FileOutputStream donose nove metode, sem metoda koje su nasledile od InputStream/OutputStream? Kako kreirate objekte klasa FileInputStream/ FileOutputStream?

5. Šta se dešava ako pokušate da kreirate ulazni tok koristeći nepostojeću datoteku? Šta će se desiti ako pokušate da kreirate izlazni tok na već postojeću datoteku? Da li možete da dodate podatke već postojećoj datoteci?

6. Kako dodajete podatke postojećoj tekstualnoj datoteci upotrebom java.io.PrinerWriter?

7. Pretpostavimo da datoteka sadrži nespecificiran broj vrednosti tipa double. Ove vrednosti su zapisane u datoteku upotrebom writeDouble metoda klase DataOutputStream. Kako bi napisali program da čita sve ove vrednosti? Kako otkrivate kraj datoteke?

8. Šta se zapisuje u datoteku kada upotrebite metod writeByte(91) primenom objekta FileOutputStream?

9. Kako proveravate kraj datoteke u ulaznom toku ((FileInputStream, DataInputStream)?

10. Šta je pogrešno u sledećem kodu?

```
import java.io.*;
 public class Test {
   public static void main(String[] args) {
    try (
      FileInputStream fis = new FileInputStream("test.dat"); ) {
      }
    catch (IOException ex) {
      ex.printStackTrace();
    }
    catch (FileNotFoundException ex) {
      ex.printStackTrace();
    }
   }
 }
```

ZADACI 5.11- 5.14

Ulaz-izlaz binarnih podataka - proverite vaše razumevanje pređenog gradiva. - zadaci 11-14

11. Pretpostavimo da izvršavate program na Windows OS upotrebom standardnog ASCII kodiranja. Posle završetka programa, koliko bajtova se nalazi u datoteci t.txt? Pokažite sadržaje za svaki bajt.

```
public class Test {
   public static void main(String[] args)
    throws java.io.IOException {
    try (java.io.PrintWriter output =
      new java.io.PrintWriter("t.txt"); ) {
      output.printf("%s", "1234");
      output.printf("%s", "5678");
      output.close();
   }
 }
```

12. Za svaki od sledećih iskaza u DataOutputStream, koliko je bajtova poslato na izlaz?

```
output.writeChar('A');
 output.writeChars("BC");
 output.writeUTF("DEF");
```

13. Posle završetka programa, koliko je bajtova sadržano u datoteci t.txt? Pokažite sadržaj svakog bajta?

```
import java.io.*;
 public class Test{
   public static void main(String[] args) throws IOException {
    try (DataOutputStream output = new DataOutputStream(
      new FileOutputStream("t.dat")); ) {
        output.writeInt(1234)  ;
        output.writeInt(5678);
        output.close();
    }
   }
 }
```

14. Koje su prednosti upotrebe bafera za tokove podataka? Da li su sledeći iskazi ispravno napisani?

```
BufferedInputStream input1 =
   new BufferedInputStream(new FileInputStream("t.dat"));
 DataInputStream input2 = new DataInputStream(
   new BufferedInputStream(new FileInputStream("t.dat")));
 DataOutputStream output = new DataOutputStream(
   new BufferedOutputStream(new FileOutputStream("t.dat")));
```

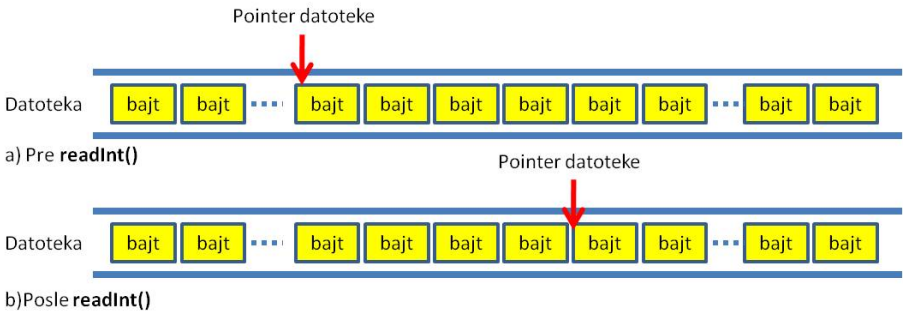
Poglavlje 6

Datoteke sa slučajnim pristupom

OPERACIJA ČITANJA I PISANJA SA SLUČAJNIM PRISTUPOM

Operacija čitanja ili pisanja se izvršava na lokaciji pointera datoteke. Kada čitate ili pišete podatke u datoteku, pointer datoteke se pomera unapred do sledećeg podatka.

Datoteka sa slučajnim pristupom (engl., Random-access file) sadrži niz bajtova. Specijalan marker koji se naziva **pointer** datoteke (engl., file pointer) pozicionira se na jedan od ovih bajtova. Operacija čitanja ili pisanja se izvršava na lokaciji pointera datoteke. Kada je datoteka otvorena, pointer datoteke se postavlja na početak datoteke. Kada čitate ili pišete podatke u datoteku, pointer datoteke se pomera unapred do sledećeg podatka. Na primer, ako čitate **int** vrednost upotrebom **readInt(),** JVM čita 4 bajta od mesta gde se nalazi pointer datoteka. Na primer, ako čitate int vrednost upotrebom readInt(), JVM čita 4 bajta od pozicije pointera, i onda se pointer pomera 4 bajta unapred u odnosu na prethodnu poziciju, kao što je pokazano na slici 2.



Slika 6.1 Posle čitanja int vrednost, pointer datoteka se pomera unapred za 4 bajta

Ako je ***raf*** objekat klase ***RandomAccessFile***, možete da upotrebite metod **raf.seek(position)** da se kreće pointer datoteke u specificiranu poziciju. Metod **raf.seek(0)** pomera pointer na početak datoteke, a **raf.seek(raf.length())** pomera pointer na kraj datoteke.

PRIMER 17 - PRIMENA KLASE RANDOMACCESSFILE

Metod raf.seek(position) stavlja pointer datoteke u specificiranu poziciju. Metod raf.seek(0) pomera pointer na početak datoteke, a raf.seek(raf.length()) pomera pointer na kraj datoteke.

Sledeći listing prikazuje primer primene klase RandomAccessFile.

```
1 import java.io.*;
  2
  3 public class TestRandomAccessFile {
  4   public static void main(String[] args) throws IOException {
  5  // Kreiranje datoteke sa slučajnim pristupom
  6  RandomAccessFile inout = new RandomAccessFile("inout.dat", "rw");
  7
  8  // Čišćenje datoteke radi uništenja starog sadržaja
  9  inout.setLength();
 10
 11  // Zapisivanje novih celih brojeva u datoteku
 12  for (int i = 0; i < 200; i++)
 13    inout.writeInt(i);
 14
 15  // Prikazivanje sadašnje dužine datoteke
 16  System.out.println("Current file length is "+ inout.length());
 17
 18  // Dobijanje prvog broja
 19  inout.seek(0); // Pomera pointer na početak datoteke
 20  System.out.println("The first number is "+ inout.readInt());
 21
 22  // Dobijanje drugog broja
 23  inout.seek(1 * 4); // Pomera pointer do drugog broja
 24  System.out.println("The second number is "+ inout.readInt());
 25
 26  // Dobijanje desetog broja
 27  inout.seek(9 * 4); // Pomeranje pointera na 10. broj
 28  System.out.println("The tenth number is "+ inout.readInt());
 29
 30  // Promena 11. broja
 31  inout.writeInt(555);
 32
 33  // Dodavanje novog broja
 34  inout.seek(inout.length()); // Pomeranje pointera na kraj datoteke
 35  inout.writeInt(999);
 36
 37  // Prikazivanje nove dužine
 38  System.out.println("The new length is "+inout.length() );
 39
 40  // Dobijanje novoh 11. broja
 41  inout.seek(10 * 4); // Pomeranje pointera do sledećeg broja
 42  System.out.println("The eleventh number is "+ inout.readInt());
```

```
43
44  inout.close();
45   }
46 }
```

Metod **inout.setLength(0)** postavlja dužinu na 0 u liniji 9. Ovim se uništava stari sadržaj datoteke. Petlja for piše 200 int vrednosti od 0 do 199 u datoteku u linijama 12-13. Kako svaka int vrednost uzima 4 bajta, ukupna dužina se vraća metodom **inout.length()** je sada 800 (linija 16), kao što se vidi i na prikazanom rezultatu.

Pozivom **inout.seek(0)** u liniji 19 postavlja pointer datoteke na početak datoteke **inout.** Metod **readInt()** čita prvu vrednost u liniji 20 i pomera pointer datoteke do sledećeg broja. Drugi broj se čita u liniji u liniji 24.

Metod **inout.seek(9 * 4)** u liniji 27, pomera pointer datoteke do 10. broja . Metod **inout.read()** 10. Broj i pomerate pointer datoteke na 11. Broj u liniji 28. Metod **inout.write(555)** piše novi 11. Broj na trenutnoj poziciji (linija 31). Prethodnih 11 brojeva su uništeni.

Metod **inout.seek(inout.length())** pomera pointer datoteke na kraj datoteke (linija 34), **inout.writeInt(999)** piše 999 na kraj datoteke. Sada je dužina datoteke je povećana za 4 bajta, tako da sada **inout.length()** vraća 804 (linija 38).

Metod **inout.seek(10*4)** pomera pointer datoteke na 11. Broj u liniji 41. Novi 11. Broj, 555, je prikazan u liniji 42

KLASA RANDOMACCESSFILE

Java obezbeđuje klasu RandomAccessFile koja dozvoljava da se datoteka čita ili da se u nju vrši zapis na slučano određenim lokacijama.

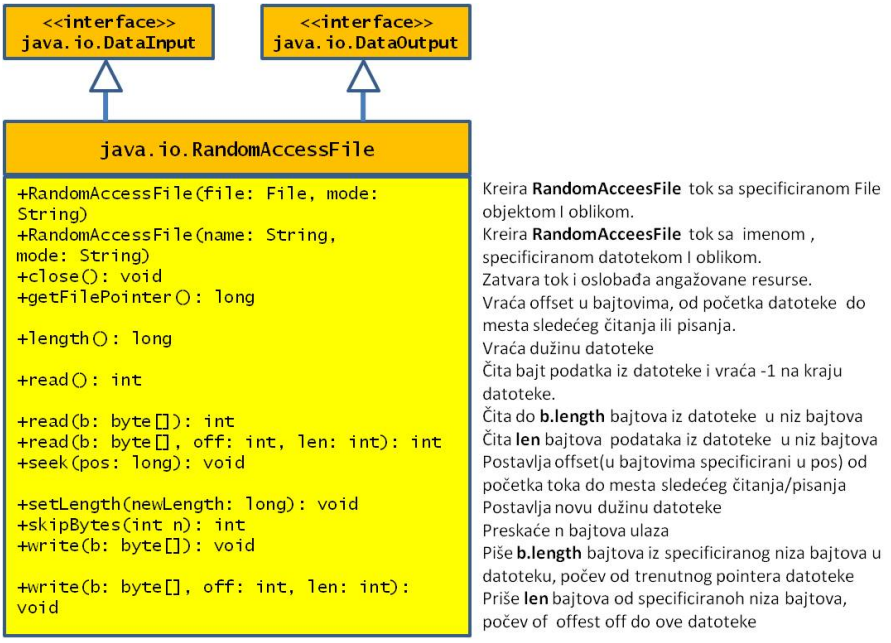
Kada se radi sa datotekama, podaci se zapisuju ili čitaju tokovima podataka, pri čemu se koriste sekvencijalne datoteke. To znači, svaka promena sadržaja datoteke zahteva formiranje nove datoteke. Java primenom klase RandomAccess File omogućava da se čitanje ili zapisivanje podataka u datoteci vrši na slučajno odabranim lokacijama.

Klasa RandomAccessFile primenjuje DataInput i DataOutput interfejse, kao što je to pokazano na slici 1. Interfejs DataInput definiše metode za čitanje primitivnih tipova podataka i stringova, a DataOutput interfejs definiše metode za zapisivanje primitivnih tipova podataka i stringove u datoteke.

Kada se kreira objekt klase RandomAccessFile, možete da specificirate jedan od dva oblika korišćenja: r ili rw. Oblik r znači da ke tok samo za čitanje, a oblik rw – da je i za čitanje i za zapisivanje. Na primer, sledeći iskaz kreira novi tok, raf, koji dozvoljava programu da ga čita i da piše u datoteci test.dat

```
RandomAccessFile raf = new RandomAccessFile("test.dat", "rw");
```

Ako datoteka **test.dat** već postoji, rad kreira se kreira da bi je koristila; ako test.dat ne postoji, kreira se nova datoteka sa nazivom test.dat i raf se kreira da bi pristupao novoj datoteci. Metod raf=length() vraća broj bajtova u test.dat u bilo koje vreme. Ako dodate nove podatke u datoteku, raf.length() se povećava.



Slika 6.2 Klasa RandomAccessFile primenjuje DataInput i DataOutput interfejse sa dodatnim metodima koji podržavaju slučajni pristup

ZADACI 6.1 - 6.3

Datoteke sa slučajnim pristupom - proverite vaše razumevanje pređenog gradiva.

1. Da li ***RandomAccessFile*** tok čita i piše datoteku sa podacima kreirane od strane***DataOutoutStream***? Da li RandomAccessFile tok može da čita i piše objekte?
2. Kreirajte jedan***RandomAccessFile*** tok za datoteku address.dat koji bi dozvolio promenu informaciju o studentu u datoteci. Kreirajte ***DataOutputStream*** da datoteku address.dat. Objasni razlike između ova dva iskaza.
3. Šta se dešava ako datoteka test.dat ne postoji kada pokušate da kompilirate i izvršite sledeći program?

```
import java.io.*;
 public class Test {
   public static void main(String[] args) {
    try ( RandomAccessFile raf =
          new RandomAccessFile("test.dat", "r"); ) {
      int i = raf.readInt();
    }
    catch (IOException ex) {
```

```
      System.out.println("IO exception");
    }
   }
}
```

Poglavlje 7

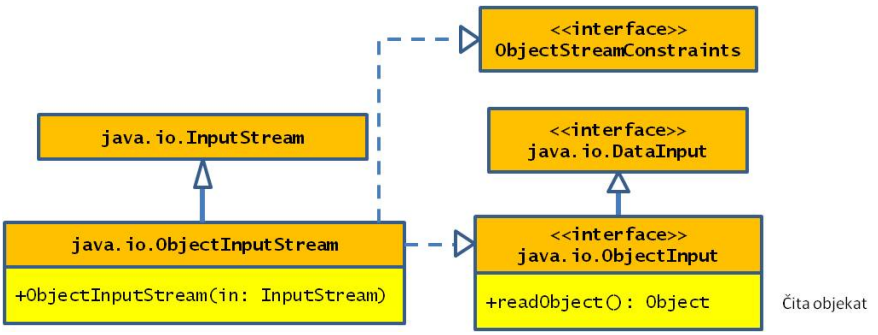
Ulaz i izlaz objekata

KLASE OBJECTINPUTSTREAM/ OBJECTOUTPUTSTREAM

Klase ObjectInputStream/ObjectOutputStream omogućavaju vam da realizujete U/I operacije nad objektima

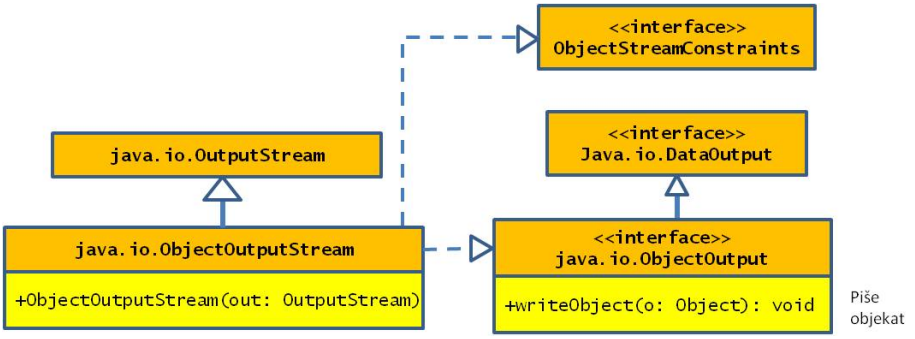
Klase ***DataInputStream/DataOutputStream*** omogućavaju vam da izvršite U/I operacije nad primitivni tipovima podataka i nad stringovima. Klase ***ObjectInputStream/ ObjectOutputStream*** omogućavaju vam da realizujete U/I operacije nad objektima, pored primitivnih tipova podatak i stringova. Kako klase ***ObjectInputStream/ ObjectOutputStream*** sadrže sve funkcije klasa ***DataInputStream/DataOutputStream,*** to one mogu da zamene kompletno ove klase.

ObjectInputStream proširuje***InputStrema*** i primenjuje ***ObjectInput*** i ***ObjectStreamConstraints,*** kao što je prikazano na slici 30. ***ObjectInput*** je pod-interfejs interfejsa ***DataInput.*** ***ObjectStreamConstraints*** sadrži ograničenja koje prate ***ObjectInputStream/ObjectOutputStream***



Slika 7.1 ObjectInputStream čita objekte, primitivne tipove podataka i stringove

Klasa ObjectOutputStream proširuje klasu OutputStream i primenjuje ObjectOutput i ObjectStreamConstraints, kao što je prikazano na slici 2. ObjectOutput je pod-interfejs interfejsa DataOutput.



Slika 7.2 ObjectOutputStream piše objekte, primitivne tipove podataka i stringove.

Možete smestiti ***ObjectInputStream/ObjectOutputStream*** u bilo koji ***InputStream/ OutputStream*** upotrebom sledećih konstruktora

```
// Kreiranje  ObjectInputStream
 public ObjectInputStream(InputStream in)
 // Kreiranje ObjectOutputStream
 public ObjectOutputStream(OutputStream out)
```

PRIMER 18 - PRIMENA KLASE OBJECTOUTPUTSTREAM

Program treba da zapisuje imena studenata, njihove ocene i trenutni datum u datoteku pod nazivom object.dat.

Ovde se prikazuje listing programa koji zapisuje imena studenata, njihove ocene i trenutni datum u datoteku pod nazivom **object.dat.**

```
1 import java.io.*;
  2
  3 public class TestObjectOutputStream {
  4   public static void main(String[] args) throws IOException {
  5   // Create an output stream for file object.dat
  6   ObjectOutputStream output =
  7  new ObjectOutputStream(new FileOutputStream("object.dat"));
  8
  9 // Write a string, double value, and object to the file
 10 output.writeUTF("John");
 11 output.writeDouble(85.5);
 12 output.writeObject(new java.util.Date());
 13
 14 // Close output stream
 15 output.close();
 16   }
 17 }
```

Objekat ***ObjectOutputStream*** piše podate u datoteku **object.dat** (linije 6-7). Jedan string, jedna vrednost tipa double i jedan objekat su zapisani u datoteku u linijama 10-12. Da bi se poboljšale performanse, dodat je bafer za tok podataka upotrebom sledećih iskaza, koje bi trebalo da zamene linije 6-7:

```
ObjectOutputStream output = new ObjectOutputStream(
new BufferedOutputStream(new FileOutputStream("object.dat")));
```

PRIMER 19 - ČITANJE PODATAKA IZ DATOTEKE

Više objekata ili primitiva se mogu upisati u tok padataka.

Više objekata ili primitiva se mogu upisati u tok padataka. Objekti moraju da se pročitaju nazad iz odgovarajućeg **ObjectInputStream** objekta sa istim tipovima i sa istim redosledom čitanja kao što je bio pri zapisivanju podataka u datoteku. Na slici 4 prikazan je listing programa kojim se čitaju podaci iz datoteke **object.dat.**

```
1 import java.io.*;
  2
  3 public class TestObjectOutputStream {
  4   public static void main(String[] args) throws IOException {
  5   // Create an output stream for file object.dat
  6   ObjectOutputStream output =
  7  new ObjectOutputStream(new FileOutputStream("object.dat"));
  8
  9 // Write a string, double value, and object to the file
 10 output.writeUTF("John");
 11 output.writeDouble(85.5);
 12 output.writeObject(new java.util.Date());
 13
 14 // Close output stream
 15 output.close();
 16   }
 17 }
```

Izvršenjem ovog programa, dobija se na konzoli sledeći rezultat:

John 85.5 Sun Dec 04 10:35:31 EST 2011

Metod ***readableObject*** može da izbaci ***java.lang.ClassNotFoundException,*** jer kada JVM obnovi objekat, prvo čita klasu objekta ako nije ranije učitana u memoriju računara. Kako izuzetak ***ClassNotFoundException*** spada u vrstu proverljivih izuzetaka, metod***main*** objavljuje izbacivanje u liniji 5. Objekat ***ObjectInputStream je*** kreiran radi čitanja ulaza iz datoteke objectt.dat u linijiama 7-8. Podaci se moraju čitatu u istom redosledu po kom su zapisivani u datoreku. Po jedan string, double vrednost i objekat su učitani iz datoteke u linijama 11-13. Kako***readObject*** vraća ***Object,*** on se konvertuje (***casting***) u Date objekat i pridodaje se ***Date*** promenljivoj u liniji 13.

INTERFEJS SERIALIZABLE

Primena ovog interfejsa omogućva relizaciju mehanizma Java serijalizacije koji automatizuje proces memorisanja objekata i nizova

Ne može svaki objekat da se upisuje u izlani tok. Objekti koji se mogu pisati se nazivauju serijalizovani objekti . Serijalizovani objekt je primerak interfejsa ***java.io.Serializable*** tako da klasa objekta mora da primeni klasu Serilizable.

Interfejs ***Serializable*** je interfejs marker – obeleživač. Kako on nema nikakvih metoda, ne morate da dodate dodatni kod u vašu klasu koja primenjuje ***Serializable*** interfejs. Primena ovog interfejsa omogućva relizaciju mehanizma Java serijalizacije koji automatizuje proces memorisanja objekata i nizova. Ova automatizacija znatno pojednostavlje memorisanje objekata, koji imajuu puno atributa (svojih i nasleđenih). To je proces serijalizacije, koji primenje ***ObjectOutputStream.***

Suprotan proces serijalizacije, je proces deserijalizacije. To je proces čitanja objekata sa datoteke, koji primenjuje ObjectInputStream. Mnoge klase u Java API primenjuju Serializable interfejs. Na primer, sve GUI komponente primenjuju Serializable interfejs, a i ***Date*** klasa. Ako pokušate da memorišete objekat koji ne podržava Serialiable interfejs, dobićete izuzetak ***NotSerializableException.***

Kada se serijalizovani objekat memoriše (smesti) u datoteku, klasa objekta se kodira, što znači da se navodi ime klase,

potpis (signatura) klase, vrednost promenljivih objekta (atributi), i zatvaranje bilo kog objekta koje poziva ovaj objekat. Vrednosti statističkih promenljivih objekta se ne memorišu.

Ako je objekt primenjuje ***Serializable*** interfejs, a sadrži i neserijalizovane primerke polja podataka, da li se mogu serijalizovati? Odgovor je ne. Da bi se ipak objekt serijalizovao, moraju se ova neserijalizovana polja obeležiti sa transient ključnu rel, d abi rekli JVM da ignoriše ova polja, kada se objekat upisuje u tok objekata. Pogledajmo ovu klasu:

```
public class C implements java.io.Serializable {
   private int v1;
   private static double v2;
   private transient A v3 = new A();
 }
 class A { } // A nije serijalizovana klasa
```

Kada se neki objekt klase **C** serijalizuje, samo promenljiva **v1** se serijalizuje. Promenljiva **v2** se ne serijalizuje jer je statička promenljiva, i promenljiva **v3** nije serijalizovana jer je obeležena sa ***transient.*** Ako se ***v3*** ne obeleži sa transient, pojaviće se izuzetak ***java.io.NoSerializableException***.

PRIMER 20 - SERIJALIZACIJA NIZOVA

Niz se može biti serijalizovan ako su svi njegovi elementi serijalizovani, pri čemu se koriste metodi: writeObject() i readObject().

Niz se može biti serijalizovan ako su svi njegovi elementi serijalizovani. U tom slučaju, ceo niz se može memorisati u datoteku upotrebom metoda ***writeObject*** , a kasnije se učitava objekat iz datoteke upotrebom ***readObject.*** Listing prikazuje slučaj memorisanja niza sa pet***int*** vrednosti i niz sa tri stringa i čita ih nazad na displej konzole.

Izvršenje prikazanog programa, dobija se sledeći rezultat na displeju konzole

1 2 3 4 5

John Susan Kim

Linije 14-15 pišu dva niza u datoteku array.dat. Linije 24-25 čita dva niza nazad, istim redosledom po kome su upisivani. Kako metod ***readObjects()*** vraća ***Object,*** koji se konvertuju ***u int[]*** i ***String[].***

```
1 import java.io.*;
  2
  3 public class TestObjectStreamForArray {
  4   public static void main(String[] args)
  5    throws ClassNotFoundException, IOException {
  6  int[] numbers = {1, 2, 3, 4, 5};
  7  String[] strings = {"John", "Susan", "Kim"};
  8
  9  // Kreiranje izlaznog toka za datoteku array.dat
    ObjectOutputStream output = new ObjectOutputStream(new
 11     FileOutputStream("array.dat", true));
 12
 13  // Pisanje niza u izlaz toka objekata
 14  output.writeObject(numbers);
 15  output.writeObject(strings);
 16
 17  // Zatvaranje toka
 18  output.close();
 19
 20  // Kreiranje ulaznog toka za datoteku array.dat
 21  ObjectInputStream input =
 22    new ObjectInputStream(new FileInputStream("array.dat"));
 23
 24  int[] newNumbers = (int[])(input.readObject());
 25  String[] newStrings = (String[])(input.readObject());
 26
 27  // Prikaz niza
 28  for (int i = 0; i < newNumbers.length; i++)
 29    System.out.print(newNumbers[i] + " ");
 30  System.out.println();
 31
```

```
 32  for (int i = 0; i < newStrings.length; i++)
 33    System.out.print(newStrings[i] + " ");
 34
 35  // Zatvaranje toka
 36  input.close();
 37   }
 38 }
```

PRIMER 21

Ovaj primer služi za demonstraciju serijalizacije objekata u datoteku.

Napraviti klasu osoba koja ima atribute ime, prezime, godine i broj telefona i poruka. Dodati prazan i preopterećen konstruktor, Za sve atribute ubaciti getter-e i setter-e i dodati toString metodu. Ovaj zadatak treba uraditi u paru sa kolegom tako što će jedan kreirati objekat sa inicijalnim vrednostima atributa, izvršiti njegovu serijalizaciju i sačuvati ga u datoteku. Datoteku će poslati drugom kolegi koji će izvrsiti deserijalizaciju i pozivajući metode toString() i getPoruka() videti sadrzaj poruke kao i pošiljaoca. Metode za serijalizaciju i deserijalizaciju možete napraviti u odvojenoj klasi SerializationUtil i postaviti ih da budu statičke.

Klasa Osoba:

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package zadatak11;

import java.io.Serializable;

/**
 *
 * @author razvoj
 */
public class Osoba implements Serializable {

    private String ime;
    private String prezime;
    private int godine;
    private String brojTelefona;
    private String poruka;

    /**
     * Prazan konstruktor
     */
    public Osoba() {
    }
```

```
    /**
     * Preopterecen konstruktor koji prima sve atribute kao parametre
     *
     * @param ime ime osobe
     * @param prezime prezime osobe
     * @param godine godine osobe
     * @param brojTelefona broj telefona osobe
     * @param poruka poruka koju osoba moze da salje
     */
    public Osoba(String ime, String prezime, int godine, String brojTelefona,
String poruka) {
        this.ime = ime;
        this.prezime = prezime;
        this.godine = godine;
        this.brojTelefona = brojTelefona;
        this.poruka = poruka;
    }

    /**
     * Vraca ime osobe
     *
     * @return ime osobe
     */
    public String getIme() {
        return ime;
    }

    /**
     * Vraca prezime osobe
     *
     * @return prezime osobe
     */
    public String getPrezime() {
        return prezime;
    }

    /**
     * Vraca godine osobe
     *
     * @return godine osobe
     */
    public int getGodine() {
        return godine;
    }

    /**
     * Vaca broj telefona osobe
     *
     * @return broj telefona osobe
     */
    public String getBrojTelefona() {
        return brojTelefona;
    }
```

```
    /**
     * Vaca sadrzaj poruke koju osoba moze da salje
     *
     * @return sadrzaj poruke
     */
    public String getPoruka() {
        return poruka;
    }

    /**
     * Postavlja ime osobe
     *
     * @param ime ime osobe
     */
    public void setIme(String ime) {
        this.ime = ime;
    }

    /**
     * Postavlja prezime osobe
     *
     * @param prezime prezime osobe
     */
    public void setPrezime(String prezime) {
        this.prezime = prezime;
    }

    /**
     * Postavlja broj godina osobe
     *
     * @param godine broj godina osobe
     */
    public void setGodine(int godine) {
        this.godine = godine;
    }

    /**
     * Postavlja broj telefona osobe
     *
     * @param brojTelefona broj telefona osobe
     */
    public void setBrojTelefona(String brojTelefona) {
        this.brojTelefona = brojTelefona;
    }

    /**
     * Postavlja poruku koju osoba moze da salje
     *
     * @param poruka sadrzaj poruke
     */
    public void setPoruka(String poruka) {
        this.poruka = poruka;
```

```
    }

    @Override
    public String toString() {
        return "Osoba - " + "ime:" + ime + ", prezime: " + prezime + ", godine: " +
godine + ", brojTelefona: " + brojTelefona;
    }

}
```

SerializationUtil klasa:

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package zadatak11;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

/**
 *
 * @author razvoj
 */
public class SerializationUtil {
    // deserijalizuje objekat klase Osoba koji se cita iz datoteke
    public static Osoba deserializeOsoba(File file) throws IOException,
ClassNotFoundException {
        FileInputStream fis = new FileInputStream(file);
        ObjectInputStream ois = new ObjectInputStream(fis);
        Osoba osoba = (Osoba) ois.readObject();
        ois.close();
        return osoba;
    }

    // serijalizuje objekat klase Osoba koji se upisuje u datoteku
    public static void serializeOsoba(Osoba osoba, File file) throws IOException {
        FileOutputStream fos = new FileOutputStream(file);
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(osoba);
        fos.close();
    }

}
```

Main klasa:

```
package zadatak11;

import java.io.File;
import java.io.IOException;

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
/**
 *
 * @author razvoj
 */
public class Zadatak11 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws IOException,
ClassNotFoundException {
        // TODO code application logic here
        Osoba o = new Osoba("Marko", "Markovic", 25, "123456", "Hello");
        SerializationUtil.serializeOsoba(o, new File("osoba.txt"));
        System.out.println(SerializationUtil.deserializeOsoba(new
File("osoba.txt")));
    }

}
```

ZADACI 7.1 - 7.6

Ulaz i izlaz objekata - proverite vaše razumevanje pređenog gradiva.

1. Koji tipovi objekata se mogu memorisati upotrebom ***ObjectOutputStream*****?** Koji je metod za zapisivanje objekata u datoteku? Koji je metod za učitavanje objekata iz datoteke? Koji je povratni tip metoda koji čita objekt iz ***ObjectInputStream*****?**
2. Ako serijalizujete dva objekta istog tipa, da li će oni zauzeti istu veličinu memorijskog prostora? Ako ne, dajte primer.
3. Da li je istina da nilo koji primerak java.io.Serialiyable može da bude uspešno serijalizovan? Da li se statičke promenljive objekta mogu da serijalizuju? Kako označavate promenljive objekta koje se ne serijalizuju?
4. Da li možete da zapišete niz u ObjectOutputStream?
5. Da li se DataInputStream/DataOutputStream može uvek da zameni sa ObjectInputStream/ObjectOutputStream?

6. Šta će se desiti kada pokušate da izvršite sledeći program?

```
import java.io.*;
 public class Test {
   public static void main(String[] args) throws IOException {
    try ( ObjectOutputStream output =
      new ObjectOutputStream(new FileOutputStream("object.dat")); ) {
      output.writeObject(new A());
    }
   }
 }
 class A implements Serializable {
   B b = new B();
 }
 class B {
 }
```

Poglavlje 8

Domaći zadaci

ZADACI 1- 3

Ovo su zadaci koje studenti treba da rade za vreme časova individualnih vežbi - zadaci 1-4

Nakon konslutacija sa asistentom - tutorom, svaki polaznik dobija 3 zadatka za domaći rad iz liste ponuđenih.

Zadatak 1

Memorisanje objekata i nizova u datoteku: Napišite program koji memoriše niz od pet int vrednosti: 1, 2, 3, 4 i 5, i objekat Date za trenutno vreme, i double vrednost za 5.5 u datoteku pod nazivom Vezba5.dat.

Zadatak 2

Pretpostavimo da pratite koliko puta program je imao svoje izvršenje. Možete veličinu tipa int da bi nrojali broj izvršenja programa. Povećajte broj izvršenja za 1 uvek kada se program izvršava. Neka se program zove Vezba8 i smestite ga u datoteku Vezba6.dat.

Zadatak 3

Podela datoteka: Pretpostavimo da želite da arhivirate veliku AVI datoteku od 10 GB. To možete postići njeom podelom na nekoliko manjih datoteka i njihovim nezavisnim arhiviranjem. Napišite korisnički program koji deli veliku datoteku u manje, upotrebom sledeće komande:

java Vezba9 SourceFile numberOfPieces

ZADACI 4-7

Ovo su zadaci koje studenti treba da rade za vreme časova individualnih vežbi - zadaci 5-7

Zadatak 4

Kombinovanje datoteka: Napišite korisnički program koji kombinuje datoteke i kreira jednu novu datoteku upotrebom komande:

java Vezba10 SourceFile1 . . . SourceFilen TargetFile

Zadatak 5

Šifriranje **datoteke**: Šifrirajte datoteku tako što ćete dodati 5 svakom bajtu u datoteci. Napišite program koj pita korisnika da unese naziv ulazne datoteke i naziv izlazne datoteke, a koji memoriše šifriranu verziju ulazne datoteke u obliku izlazne datoteke.

Zadatak 6

Dešifriranje datoteke: Pretpostavimo da je datoteka šifrirana upotrebom šeme u zadatku br. 11. Napišite program koji dešifrira šifriranu datoteku. Program treba da pita korisnika da unese naziv ulazne šifrirane datoteke i naziv izlazne nešifrirane datoteke.

Zadatak 7

Frekvencija karaktera: Napišite program koji će pitati korisnika da unese ime ASCII tekstualne datoteke i koji će prikazati frekvenciju (učestlost javljanja) karaktera u datoteci.

ZADACI 8-11

Ovo su zadaci koje studenti treba da rade za vreme časova individualnih vežbi - zadaci 8-11

Zadatak 8

Kreiranje tekstualne datoteke: Napišite program za kreiranje datoteke pod nazivom Vezba1.txt ako ne postoji. Dodajte novi podatak u datoteku, ako ona već postoji. Zapišite 100 celih brojeva slučajno kreiranih u datoteku upotrebom tekstualnog U/I. Celi brojevi su razdvojeni blanko poljem,

Zadatak 9

Kreiranje binarne datoteke podataka: Napišite program koji kreira datoteku po nazivu Vezba2.dat, ako ona ne postoji. Dodajte nove podatke u nju, ako ona već postoji. Zapište u datoteku 100 celih brojeva slučajno kreiranih primenom binarnih U/I operacija.

Zadatak 10

Zbir **svih celih brojeva smeštenih u binarnih datoteci podataka:** Pretpostavimo da već postoji binarna datoteka podataka sa nazivom Vezba2.dat. i da su njeni podaci zapisani sa metodom writeInt(int) klase DataOutputStream. Datoteka sadrži nespecificiran broj celih brojeva. Napišite program za obračun zbira celih brojeva.

Zadatak 11

Konverzija **tekstualne datoteke u UTF**: Napišite program koji čita linije sa karakterima iz tekstualne datoteka i koji zapisuje svaku liniju kao UTF-8 string u binarnu datoteku. Prikažite veličinu tekstualne datoteke i binarne datoteke. Upotrebite sledeću komandu za izvršenje programa:

Zaključak

GLAVNE POUKE LEKCIJE

ObjectInputStream/ObjectOutputStream se može upotrebiti sa čitanje/ pisanje objekata. Klasa RandomAccessFile omogućava čitanje i pisanje podataka u datoteku.

1. Klasa File se koristi za dobijanje svojstava datoteke i za manipulaciju sa datotekama. Ona ne sadrži metode za kreiranje datoteka ili za čitanje/zapisivanje podataka sa datoteke, odn. na datoteku.
2. Možete koristiti Scanner klasu da čitate stringove i vrednosti prostih promenljivih sa tekstualnih datoteka i da koristite PrintWriter da kreirate datoteku i da zapisujete podatke u tekstualnu datoteku.
3. Klasa JFileChooser se koristi za prikaz datoteka koje se mogu izabrati radi učitavanja.
4. Možete čitati datoteku sa veba, upotrebom klase URL.
5. Klase***InputStream*** i ***OutputStream*** su koreni za sve binarne U/I klase.***FileInputStream/FileOutputStream*** povezuju datoteku za ulaz/izlaz. ***BufferedInputStream/BufferedOutputStream*** se može koristiti za uvijanje svaki U/I binarni U/I toka da bi se poboljšale perfoemanse. ***DataInputStream/ DataOutputStream*** se koristi za čitanje/pisanje primitivnih vrednosti primitivnih vrednosti i stringovi.
6. ***ObjectInputStream/ObjectOutputStream*** se može upotrebiti sa čitanje/pisanje objekata, pored primitivnih vrednosti i stringova. Da bi se omogućila serijalizacija objekta, klasa ovog objekta mora da primeni java.io.Serializable marker interfejs.
7. Klasa ***RandomAccessFile*** omogućava čitanje i pisanje podataka u datoteku. Možete ovoriti datoteku sa formom r (za čitanje) i sa formom rw (za čitanje i pisanje). Kako ***RandomAccessFile*** klasa primenjuje ***DataInput*** i ***DataOutput*** interfejse, mnogi metodi klase ***RandomAccessFile*** su isti kao oni u ***DataInputStream*** i ***DataoutputStream.***

Referenca: Y. Daniel Liang, INTRODUCTION TO JAVA PROGRAMMING (COMPREHENSIVE VERSION), Tenth Edition, Pearson, ISBN 10: 0-13-376131-2, ISBN 13: 978-0-13-376131-3

Ovo je osnovni udžbenik za ovaj predmet i preporučuje se studentima da ga koriste,