Ulazno/izlazni tokovi u Javi i datoteke

Uvod

- Podaci se osim preko tipkovnice (konzole) u aplikaciju mogu unositi iz različitih izvora, kao što su npr. datoteke
- Osim toga se podaci nakon završetka rada programa također mogu zapisivati u datoteke kako bi bili pohranjeni za kasniju upotrebu
- Za komunikaciju s datotekama se u Javi koriste tokovi podataka
- Postoje dvije vrste datoteka:
  - Datoteke s binarnim sadržajem
  - Datoteke s tekstualnim sadržajem
- Program u Javi otvara datoteku tako da kreira jedan objekt koji je povezan s tokom *byte*-ova ili znakova

Tokovi u Javi

- Programi u Javi mogu se tijekom izvođenja programa spojiti s različitim uređajima sa sljedećim tokovima podataka (engl. *streams*):
  - **`System.in`** tok omogućava unos podataka s tipkovnice
  - **`System.out`** tok omogućava ispis podataka na ekranu
  - **`System.err`** tok omogućava ispis pogrešaka na ekran
- Tokovi su predstavljeni sekvencama (poredanih) *byte*-ova neodređene duljine

Tokovi za čitanje i pisanje sadržaja u datoteke (1/2)

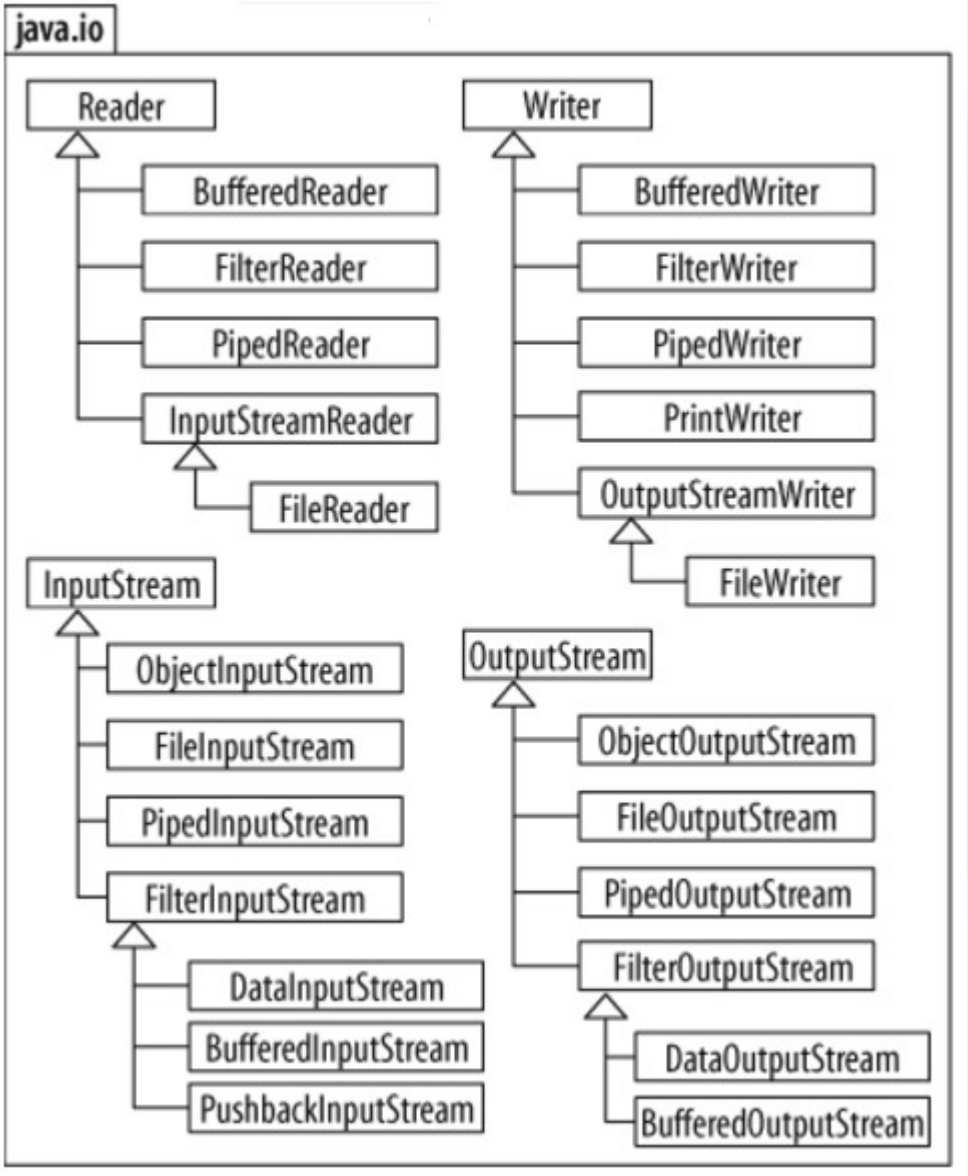
- U Javi postoji nekoliko klasa koje predstavljaju tokove za razmjenu podataka s datotekama, a dijele se na dvije osnovne skupine:
  - Ulazni tokovi podataka
  - Izlazni tokovi podataka
- Osnovne apstraktne klase iz kojih su izvedene ostale klase za čitanje i zapisivanje toka *byte*-ova (za binarne datoteke):
  - **`java.io.InputStream`** (šalje *byte*-ove iz vanjskog izvora u Java program)
  - **`java.io.OutputStream`** (šalje *byte*-ove iz Java programa u neko vanjsko odredište)
- Tokovi *byte*-ova (binarnih podataka) prenose jedinice podataka duljine 8 bitova

Tokovi za čitanje i pisanje sadržaja u datoteke (2/2)

- Osim binarnih tokova postoje i tokovi znakova koji se koriste za čitanje i zapisivanje znakova u tekstualne datoteke
- Osnovne apstraktne klase iz kojih su izvedene ostale klase za čitanje i zapisivanje znakovnih tokova:
  - **`java.io.Reader`** (prima znakove iz vanjskog izvora i šalje u Java program)
  - **`java.io.Writer`** (šalje znakove iz Java programa u vanjski izvor)
- Tokovi znakova prenose jedinice podataka u obliku 16-bitnih cjelina koje predstavljaju **char** tipove podataka

Klase koje predstavljaju tokove podataka u Javi

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



Ulazni tokovi *byte*-ova

- Osnovna metoda klase **`InputStream`** je:

```
public abstract int read() throws IOException
```

- Postoji niz implementacija metode **`read`** u klasama koje nasljeđuju klasu **`InputStream`** (**`FileInputStream`**, **`AudioInputStream`**...)
- Metoda **`read`** čita jedan *byte* iz ulaznog toka, a vraća cjelobrojnu vrijednost tog *byte*-a
- Metoda **`read`** blokira izvođenje ostatka programa (čeka) tako dugo dok se jedan *byte* ne pročita

Primjer čitanja datoteke s binarnim sadržajem (1/3)

- Sadržaj binarne datoteke "datumi.txt" moguće je pročitati na sljedeći način:

```
public static final String FILENAME = "datumi.txt";
public static final int DATE_FORMAT_LENGTH = "dd.MM.yyyy.".length();

public static void main(String[] args) {
    try {
        InputStream in = new FileInputStream(FILENAME);
        char[] data = new char[DATE_FORMAT_LENGTH];
        for (int i = 0; i < data.length; i++) {
            int datum = in.read();
            if (datum == -1)
                break;
            data[i] = (char) datum;
        }
        System.out.println("Pročitani datum : " + String.valueOf(data));
        in.close();
    } catch (IOException ex) {
        System.err.println(ex.getMessage());
    }
}
```

Primjer čitanja datoteke s binarnim sadržajem (2/3)

- Prije početka čitanja sadržaja potrebno je otvoriti datoteku:

```
InputStream in = new FileInputStream(FILENAME);
```

- Pomoću metode "read" čita se *byte* po *byte*, a vraća se cjelobrojna vrijednost tog *byte*a:

```
int datum = in.read();
```

- Ako metoda "read" vrati vrijednost "-1", to znači da nije pročitan nijedan *byte*, odnosno, da je pročitan cijeli sadržaj datoteke i "-1" predstavlja indikator kraja datoteke
- Nakon završetka faze čitanja datoteke obvezno je potrebno zatvoriti datoteku:

```
in.close();
```

- Ako datoteka ne postoji ili je nemoguće čitati njen sadržaj, baca se **označena** iznimka "IOException"

Primjer čitanja datoteke s binarnim sadržajem (3/3)

- Osim lokalnih datoteka moguće je dohvaćati i resurse s mrežnih stranica:

```
InputStream in = null;
try {
    URL u = new URL("http://www.hnb.hr/tecajn/f201110.dat");
    in = u.openStream();
     int znak;
     while((znak = in.read()) >= 0)
        System.out.print((char)znak);
    }
} catch (IOException ex) {System.err.println(ex);}
  finally {
    if (in != null) {
        try {
            in.close();
        } catch (IOException ex) {
            System.err.println(ex);
        }
    }
}
```

Izlazni tokovi *byte*-ova

- Osnovna metoda klase **`OutputStream`** je:

```
public abstract void write(int b)
                                throws IOException
```

- Postoji niz implementacija metode **`write`** u klasama koje nasljeđuju klasu **`OutputStream`** (**`FileOutputStream`**, **`ObjectOutputStream`**...)
- Metoda **`write`** šalje jedan *byte* podataka preko izlaznog toka do odredišta (npr. konzole) koja taj znak interpretira na određen način
- Npr. konzola numeričke vrijednosti tretira kao ASCII znakove

Primjer ispisa ASCII tablice u konzolu

```
public static final int ROW_LENGTH = 8;
public static final int ASCII_RANGE = 128;
public static final int FIRST_CHAR_INDEX = 32;

public static void main(String[] args) {
 for (int i = FIRST_CHAR_INDEX; i < ASCII_RANGE; i++) {
   System.out.write(i);
   if (i % ROW_LENGTH == ROW_LENGTH - 1) {
      System.out.write('\n');
   }
   else {
      System.out.write('\t');
   }
 }
}
```

Primjer zapisivanja podataka u datoteku s binarnim sadržajem (1/2)

```
public static void main(String[] args) {
    OutputStream out = null;
    try {
        out = new FileOutputStream("brojevi.dat");
        for (int i = 0; i < 10; i++) {
            out.write(i + '0');
        }
    }
    catch (IOException ex) {
        System.err.println(ex);
    }
    finally {
        if (out != null) {
            try {
                out.close();
            }
            catch (IOException ex) {
                System.err.println(ex);
            }
        }
    }
}
```

Primjer zapisivanja podataka u datoteku s binarnim sadržajem (2/2)

- Slično kao i kod čitanja datoteka, kod zapisivanja je također potrebno koristiti odgovarajući objekt za zapisivanje sadržaja u datoteku:

```
out = new FileOutputStream("brojevi.dat");
```

- Pomoću tog objekta moguće je zapisivati podatke *byte* po *byte*:

```
out.write(i + '0');
```

- Nakon završetka faze zapisivanja potrebno je zatvoriti datoteku:

```
out.close();
```

- Ako datoteka ne postoji ili je nemoguće u nju zapisivati novi sadržaj, baca se **označena** iznimka "IOException"

Datoteke u Javi

- U Javi postoji klasa pod nazivom **`File`** koja predstavlja datoteke
- Objekti **`java.io.File`** klase predstavljaju putanju do datoteke, ime datoteke i njezinu veličinu, ali ne i sam sadržaj datoteke
- Kao i u C-u, postoje binarne i tekstualne datoteke
- Na primjer, objekt klase **`java.io.File`** može označavati i direktorij (mapu)
- Objekt klase **`java.io.File`** moguće je kreirati pomoću nekoliko različitih konstruktora:

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

Metode klase java.io.File

- Postoji niz korisnih metoda u klasi **java.io.File**:
  - Provjera postoji li navedena datoteka ili direktorij:

    ```
    public boolean exists();
    ```
  - Provjera radi li se o datoteci:

    ```
    public boolean isFile();
    ```
  - Provjera radi li se o direktoriju:

    ```
    public boolean isDirectory();
    ```
  - Dohvaćanje imena datoteke (ne uključuje putanju):

    ```
    public String getName();
    ```
  - Dohvaćanje putanje datoteke:

    ```
    public String getPath();
    ```

Primjer čitanja tekstualnih datoteka (1/2)

- Primjer programa za čitanje podataka iz datoteke "input.txt" redak po redak i ispisivanje u konzolu:

```
public static final String FILE_NAME = "input.txt";

public static void main(String[] args) {
    try {
        BufferedReader in = new BufferedReader(
                            new FileReader(FILE_NAME));
        String line;
        while ((line = in.readLine()) != null) {
            System.out.println(line);
        }
        in.close();
    } catch (IOException e) {
        System.err.println(e);
    }
}
```

Primjer čitanja tekstualnih datoteka (2/2)

- Za čitanje tekstualnih datoteka također postoje namijenjene klase za tu svrhu:

```
BufferedReader in = new BufferedReader(
    new FileReader(FILE_NAME));
```

- Metodom "readLine" moguće je čitati liniju po liniju:

```
line = in.readLine();
```

- Ako metoda "readLine" vrati "null" kao rezultat čitanja, a ne String objekt koji predstavlja pročitanu liniju teksta, to označava da je pročitana cijela datoteka
- Kao i kod binarnih datoteka, tekstualne je također potrebno zatvoriti nakon operacije čitanja:

```
in.close();
```

- Ako datoteka ne postoji ili je nemoguće čitati njen sadržaj, baca se **označena** iznimka "IOException"

Primjer zapisivanja u tekstualne datoteke (1/2)

- Primjer programa koji zapisuje deset redaka teksta u tekstualnu datoteku:

```
public static void main(String[] args) {
    try {
        File f = new File(FILE_NAME);
        PrintWriter out = new PrintWriter(new FileWriter(f));
        int i = 0;
        do {
            out.println((i + 1) + ". redak");
            i++;
        }
        while (i < 10);
        out.close();
    } catch (IOException e) {
        System.err.println(e);
    }
}
```

Primjer zapisivanja u tekstualne datoteke (2/2)

- Kod zapisivanja sadržaja u tekstualne datoteke potrebno je koristiti klase koje završavaju s nazivom "Writer":

```
PrintWriter out = new PrintWriter(new
  FileWriter(f));
```

- Samo dodavanje redaka u datoteku omogućeno je korištenjem metode "println":

```
out.println((i + 1) + ". redak");
```

- Na kraju je uvijek potrebno zatvoriti datoteku:

```
out.close();
```

- Ako datoteka ne postoji ili je nemoguće dodavati novi sadržaj u nju, baca se **označena** iznimka "IOException"

"Try" blok s resursima i datoteke

- Problem sa zatvaranjem datoteke moguće je riješiti korištenjem "Try" bloka s resursima uvedenog u Javi 7
- Prošli program moguće je riješiti na puno kraći način:

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

- Unutar "Try" bloka s resursima potrebno je odraditi inicijalizaciju resursa kao što je tok podataka iz/u datoteku
- Korištenjem "Try" bloka s resursima nije potrebno pozivati metodu "close" jer se ona **automatski poziva**

Serijalizacija i deserijalizacija u Javi (1/3)

- Tijekom izvođenja programa objekti koji se koriste nalaze se u memoriji, a nakon završetka programa se oslobađa memorija koju su zauzimali
- Ako je potrebno "sačuvati" objekte kako bi se mogli koristiti i nakon završetka programa, potrebno ih je serijalizirati
- **Serijalizacija** se odnosi na zapisivanje stanja objekta (vrijednosti varijabli) u binarnom obliku u datoteku
- Obrnuti proces čitanja objekata u binarnom obliku i njihovo "aktiviranje" u trenutno aktivnom programu naziva se **deserijalizacija**
- Klase čiji objekti se žele serijalizirati moraju implementirati sučelje **Serializable**

Serijalizacija i deserijalizacija u Javi (2/3)

- Primjer koda za serijalizaciju objekta:

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

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

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

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

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

Serijalizacija i deserijalizacija u Javi (3/3)

- Primjer koda za deserijalizaciju objekata:

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

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

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

- Za deserijalizaciju objekata koristi se klasa "ObjectInputStream" i metoda "readObject"