

# Vhod in izhod (delo z datotekami)

Programiranje 2, Tomaž Dobravec



# Razred `java.io.File`

---

- Razred `File` omogoča sistemsko neodvisen dostop do datotečnega sistema.
- Objekt razreda `File` uporabljamo za delo z datotekami in NE za delo z vsebino datotek!
- Objekt razreda `File` lahko predstavlja
  - datoteko ali
  - direktorij.





# Kako ustvarim objekt razreda `File`

Primeri:

ime datoteke v sistemu Windows

```
File f1 = new File("c:\\delo\\datoteka.txt");
```

pozor: znak `\` je kontrolni znak; `\n` pomeni 'nova vrstica', `\\` pa znak `'\'`

ime datoteke v sistemu Linux

```
File f2 = new File("/home/tomaz/delo/datoteka.txt");
```

► Dva ekvivalentna načina:

```
File pd = new File("/home/tomaz/delo/datoteka.txt");
```

```
File p = new File("/home/tomaz/delo");
```

```
File pd = new File(p, "datoteka.txt");
```



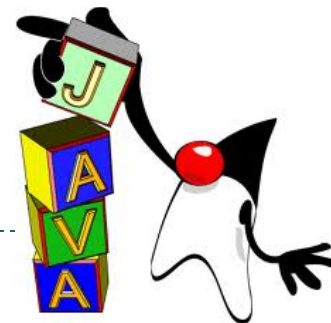


# Nekaterere metode objektov razreda `File`

---

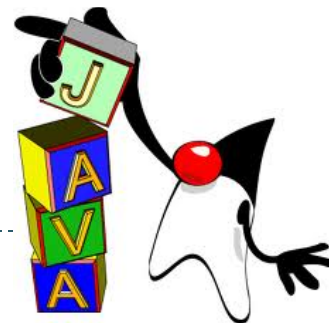
- `public boolean canRead()`
- `public boolean canWrite()`
- `public boolean exists()`
- `public boolean isDirectory()`
- `public long lastModified()`
- `public long length()`
- `public boolean delete()`
- `public boolean mkdir()`
- `public boolean makedirs()`
- `public boolean renameTo(File dest)`
- `public String[] list()`





- ▶ Napiši program, ki izpiše velikost datoteke, ki je podana kot prvi argument ob klicu programa.

```
File f = new File(args[0]);  
if (!f.exists())  
    System.out.printf("Datoteka '%s' ne obstaja!\n", args[0]);  
else  
    System.out.printf("Velikost datoteke '%s' je %d bajtov\n",  
        args[0], f.length());
```



TreeA.java, TreeB.java

Napiši program, ki izpiše drevo datotek, kot ga izpiše program `tree` v Linux lupini.

Primer izpisa:

### Verzija A

```
build
build/classes
build/classes/Dn1.class
build/classes/Dn2.class
build/classes/Dn3.class
build/classes/Dn4.class
build/classes/Dn5.class
build/classes/KvadratneEnacbe.class
build/classes/p2
build/classes/p2/testi
build/classes/p2/testi/Test.class
```

### Verzija B

```
|__ build
| |__ classes
| | |__ Dn1.class
| | |__ Dn2.class
| | |__ Dn3.class
| | |__ Dn4.class
| | |__ Dn5.class
| | |__ KvadratneEnacbe.class
| | |__ p2
| | | |__ testi
| | | | |__ Test.class
```

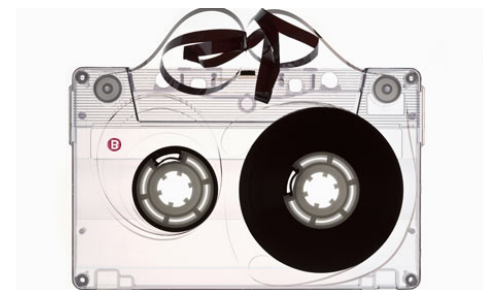




# Podatkovni tokovi



- Za delo z datotekami Java uporablja t.i. tokove (*angl. stream*).
- Java pozna vhodne (za branje) in izhodne (za pisanje) tokove.
- Vsak vhodni tok ima svoj izvor (datoteka, od koder podatki izvirajo), vsak izhodni tok svoj ponor (datoteka, kamor se podatki pišejo).
- Razlika med tabelo in tokom podatkov?
  - ▶ tabela: v vsakem hipu enako hitro dobimo katerikoli podatek;
  - ▶ tok: v nekem hipu lahko beremo le “trenutni” podatek.
- Primerjava toka in avdio kasete
  - ▶ poslušam samo skladbo, ki je trenutno na vrsti;
  - ▶ nekateri kasetofoni ne dovolijo previjanja;
  - ▶ če “previjanje” obstaja, je praviloma počasno.





# Podatkovni tokovi



## Osnovni način uporabe tokov

```
File f = new File(args[0]);  
  
FileInputStream fis = new FileInputStream(f);  
  
while (fis.available() > 0) {  
    int z = fis.read();  
    // ... "obdelamo" podatek z  
}  
  
fis.close();
```

izberem primeren tok in  
ga povežemo z virom

“pretočim” vse podatke

tok zaprem

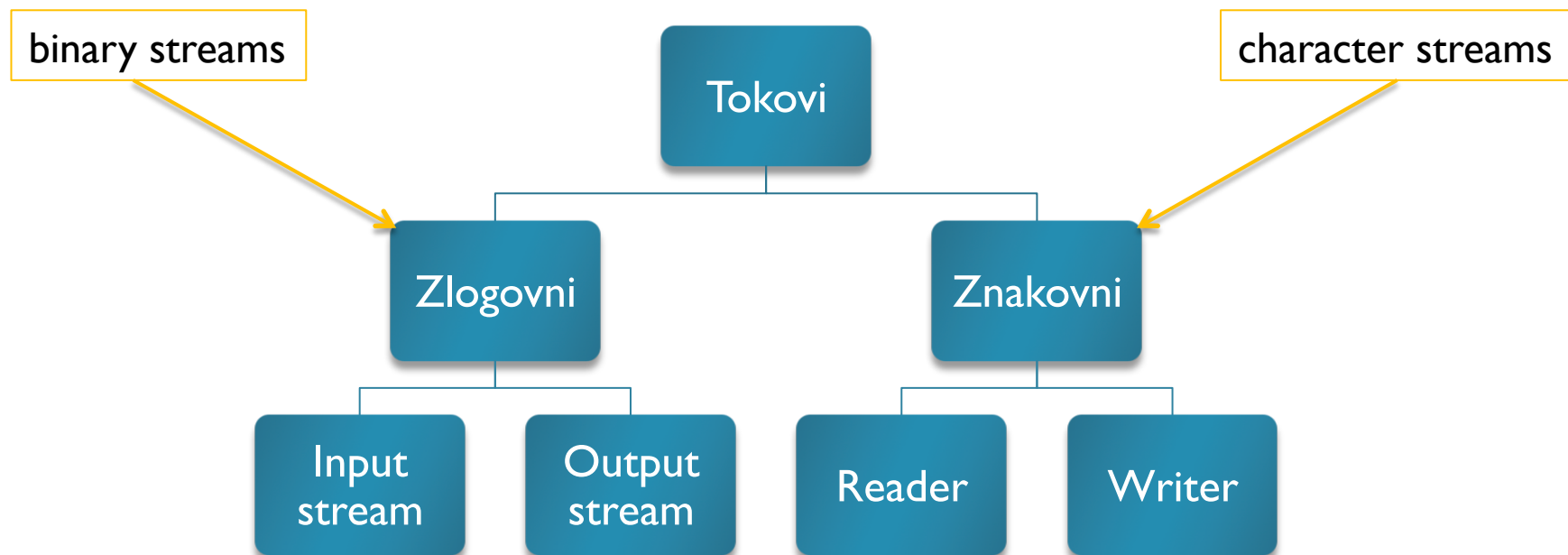






# Tokovi v Javi

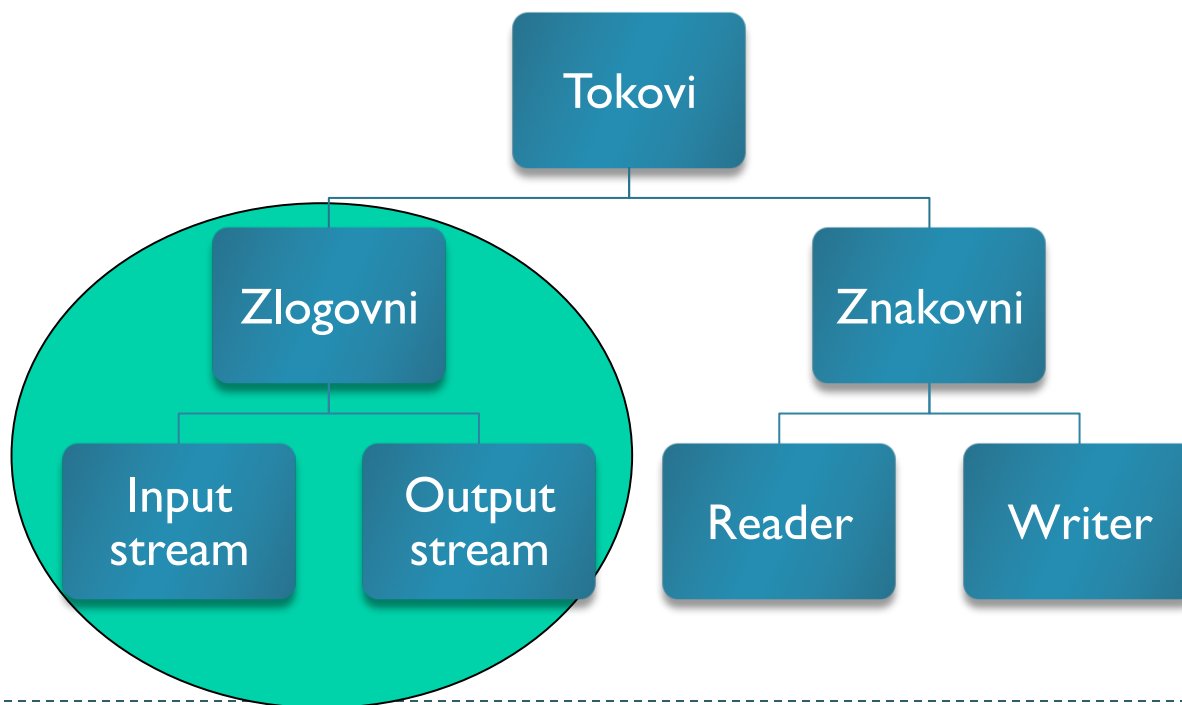
- Java loči med znakovnimi in zlogovnimi tokovi.





# Zlogovni tokovi

- ▶ Zlogovni tokovi se uporabljajo za branje/pisanje podatkov, ki so shranjeni v binarni obliki, na primer: slike (jpg, png, gif), filmi (mov, avi, mp4), dokumenti (doc, docx), arhivske datoteke (zip, rar), programi (exe, class), ...
- ▶ Osnovni podatek v zlogovnih tokovih je velik en zlog (byte).





# Vhodni in izhodni zlogovni tokovi

---

- Vhodni zlogovni tok se imenuje *input stream*

- Osnovne metode razreda `InputStream`

```
public abstract int read()  
public int read(byte[] b)  
public int read(byte[] b, int off, int len)
```

- Izhodni zlogovni tok se imenuje *output stream*

- Osnovne metode razreda `OutputStream`

```
public abstract void write(int b)  
public void write(byte[] b)  
public void write(byte[] b, int off, int len)
```

- Vse omenjene metode ob napaki vržejo izjemo `IOException`





# Nekatere implementacije razredov InputStream in OutputStream

---

## Vhodni podatkovni tokovi

```
java.io.InputStream
|
+--java.io.FileInputStream
|
+--java.io.FilterInputStream
|   |
|   +--java.io.DataInputStream
|   |
|   +--java.io.BufferedReader
|
+--java.io.ObjectInputStream
```

## Izhodni podatkovni tokovi

```
java.io.OutputStream
|
+--java.io.FileOutputStream
|
+--java.io.FilterOutputStream
|   |
|   +--java.io.DataOutputStream
|   |
|   +--java.io.BufferedOutputStream
|
+--java.io.ObjectOutputStream
```





# FileInputStream in FileOutputStream

---

- Osnovna razreda za branje/pisanje zlogovnih datotek
- Nekateri konstruktorji

```
public FileInputStream(File file)  
public FileInputStream(String name)
```

```
public FileOutputStream(File file)  
public FileOutputStream(String name)  
public FileOutputStream(String name, boolean append)
```

Vsi omenjeni konstruktorji ob napaki vržejo izjemo `FileNotFoundException`



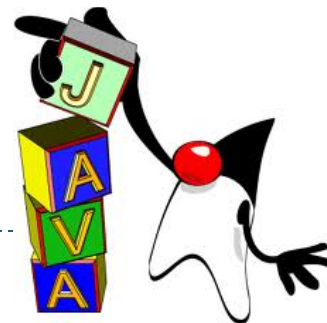


# FileInputStream in FileOutputStream

---

- `FileInputStream`
  - če datoteka, s katero želimo povezati tok, ne obstaja, konstruktor vrže izjemo `FileNotFoundException`.
- `FileOutputStream`
  - če datoteka že obstaja, se bo ob klicu konstruktorja resetirala (vsebina se pobriše);
  - pri nekaterih konstruktorjih lahko izberemo opcijo *append* (dodajanje podatkov v datoteko);
  - če želimo pisati v datoteko, pa za to nimamo pravic, konstruktor vrže izjemo `SecurityException`





Napiši program za izpis zlogov (šestnajstiške kode) datoteke.

```

C:\WINDOWS\system32\cmd.exe
D:\>java io.Hexdump studenti.txt
36 33 30 30 30 30 30 31 3A 4D 69 63 6B 61 3A 4B 63000001:Micka:K
6F 76 61 63 65 76 61 3A 39 3A 31 38 33 2E 35 3A ovaceva:9:183.5:
52 4A 41 56 41 0D 0A 36 33 30 30 30 30 30 32 3A RJAVA..63000002:
4A 61 6E 65 7A 3A 4E 6F 76 61 6B 3A 38 3A 31 37 Janez:Novak:8:17
36 3A 52 44 45 43 41 0D 0A 36 33 30 30 30 30 30 6:RDECA..6300000
33 3A 4D 69 68 61 3A 44 6F 6C 7A 61 6E 3A 31 30 3:Miha:Dolzan:10
3A 31 38 30 2E 35 3A 43 52 4E 41 0D 0A 36 33 30 :180.5:CRNA..630
30 30 30 30 34 3A 4D 65 74 6B 61 3A 4D 61 7A 65 000004:Metka:Maze
3A 31 30 3A 31 37 38 2E 35 3A 52 4A 41 56 41 0D :10:178.5:RJAVA.
0A 36 33 30 30 30 30 30 35 3A 54 61 64 65 6A 3A .63000005:Tadej:
48 6F 63 65 76 61 72 3A 39 3A 31 36 35 2E 32 3A Hocevar:9:165.2:
43 52 4E 41 0D 0A 36 33 30 30 30 30 30 36 3A 53 CRNA..63000006:S
74 65 66 6B 61 3A 44 72 6F 62 74 69 6E 61 3A 36 tefka:Drobtina:6
3A 31 37 38 3A 42 4C 4F 4E 44 :178:BLOND
D:\>

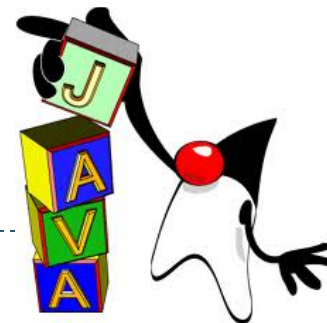
```



# Naloga

## Prepis ene datoteke v drugo (copy)

io/Copy.java



Napiši program za prepis vsebine ene datoteke v drugo datoteko.

Nalogo reši na dva načina:

- a) vhodno datoteko preberi bajt-po-bajtu;
- b) vhodno datoteko beri po blokih velikosti 2048 bajtov.

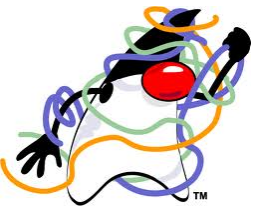
Izmeri in med seboj primerjaj čas izvajanja obeh rešitev pri prepisu velike datoteke (100MB ali več).

Opomba: za prepis datotek lahko sicer uporabljaš tudi metodo `copyFile` iz paketa `FileUtils` zbirke Apache Commons.



Vhod in izhod (delo z datotekami)





# DataInputStream in DataOutputStream

---

Tokove tipa “*data*” uporabljamo za branje in pisanje javanskih osnovnih podatkovnih tipov (`boolean`, `char`, `byte`, `short`, `int`, `long`, `float`, `double`, `String`) v zlogovni obliki

## Pisanje:

- `boolean` se zapiše z enim zlogom kot 0 ali 1
- `char` se zapiše z dvema zlogoma
- `int` se zapiše s štirimi zlogi
- ...

## Branje:

- ukaz za branje prebere primerno število zlogov (odvisno od tipa – za vsak tip imamo svoj ukaz) in jih pretvori v podatek ustreznega tipa;
- Primer: z ukazom `readByte()` preberemo 1 zlog, z ukazom `readInt()` pa štiri.





# DataInputStream in DataOutputStream

---

- DataInputStream in DataOutputStream sta ovojna tokova (filtra) – vedno jih uporabimo v kombinaciji z nekim drugim tokom

- Konstruktorji:

```
public DataStream(InputStream in)
```

```
public DataStream(OutputStream out)
```





# DataOutputStream - uporaba

1. Ustvarimo objekt tipa `FileOutputStream`

```
FileOutputStream fos = new FileOutputStream("podatki.dat");
```

2. `FileOutputStream` ovijemo v `DataOutputStream`

```
DataOutputStream dos = new DataOutputStream(fos);
```

3. Podatke zapisujemo s pomočjo toka tipa "data" takole:

```
dos.writeBoolean(true);  
dos.writeByte(32);  
dos.writeChar('A');  
dos.writeChar('\u01D7');  
dos.writeChars("abcd");  
dos.writeUTF("abcd");  
dos.writeInt(32);
```

4. Zapremo tok

```
dos.close();
```

vsebina datoteke  
(bajti datoteke so  
prikazani v  
šestnajstiški obliki)

01	20	00	41	01	D7
00	61	00	62	00	63
00	64	00	04	61	62
63	64	00	00	00	20





# DataInputStream - uporaba

1. Ustvarimo objekt tipa `FileInputStream`

```
FileInputStream fis = new FileInputStream("podatki.dat");
```

2. `FileInputStream` ovijemo z `DataInputStream`

```
DataInputStream dis = new DataInputStream(fis);
```

3. Preberemo podatke z uporabo metod "data" toka:

```
boolean b = dis.readBoolean();  
byte z = dis.readByte();  
char c1 = dis.readChar();  
char c2 = dis.readChar();  
dis.skipBytes(8); // preskočim 4 znake  
String s2 = dis.readUTF();  
int i = dis.readInt();
```

Poznati moramo  
natančno vsebino  
datoteke, da lahko  
beremo v pravem  
vrstnem redu!



01	20	00	41	01	D7
00	61	00	62	00	63
00	64	00	04	61	62
63	64	00	00	00	20

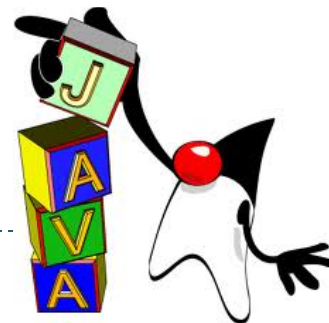
4. Zapremo tok

```
dis.close();
```



# Naloga

## Branje vsebina class datoteke



io/ClassDump.java

Napiši program, ki v datoteki tipa `class` (prevedena javanska koda) prebere podatke o verziji jave, s katero je bila datoteka ustvarjena.

Podrobneje: Začetek datoteke tipa `class` vsebuje naslednje podatke:

```
batji 1, 2,3,4:      ca fe ba be      ... podatek, ki ga preverimo (da
                                gre res za class datoteko)

bajti 5,6:          java minor version

bajti 7,8:          java major version ... podatek, ki ga potrebujemo

bajti 9, 10, ...:   ostali podatki
```

```
kepica:src tomaz$ javac Veckratniki.java
kepica:src tomaz$ hexdump -C Veckratniki.class
00000000  ca fe ba be 00 00 00 34 00 4b 0a 00 0f 00 22 07 |.....4.K....".|
00000010  00 23 09 00 24 00 25 0a 00 02 00 26 09 00 24 00 |. #..$.%...&..$.|
00000020  27 08 00 28 0a 00 29 00 2a 0a 00 2b 08 00 2c 00 |' ( ) * . . . .|
```



Vhod in izhod (delo z datotekami)



# BufferedInputStream in BufferedOutputStream

---

- BufferedInputStream in BufferedOutputStream sta ovojna tokova, ki jih uporabljamo za **povečanje hitrosti** branja/pisanja.
- Ker sta to ovojna tokova, ju vedno jih uporabimo v kombinaciji z nekim drugim tokom.

## Primer uporabe za BufferedInputStream

1. Ustvarimo objekt tipa FileInputStream

```
FileInputStream fis = new FileInputStream("podatki.dat");
```

2. Ustvarimo objekt tipa BufferedInputStream

```
BufferedInputStream bif = new BufferedInputStream(fis);
```

3. Ustvarimo objekt tipa DataInputStream

```
DataInputStream dis = new DataInputStream(bif);
```

Tok `dis` uporabimo na enak način, kot bi ga uporabili, če bi z njim namesto `BufferedInputStream` ovili `FileInputStream` (če bi izpustili korak 2), le da je sedaj branje veliko hitrejše!

Enako (simetrično) velja za `BufferedOutputStream`.



Vhod in izhod (delo z datotekami)



# Serializacija in deserializacija objektov

- Serializacija objektov je postopek, ki omogoča pretvorbo objekta v zaporedje zlogov.
- Deserializacija je postopek za generiranje objektov iz (pravilnega) zaporedja zlogov
- Objekt, ki ga želimo serializirati, mora implementirati vmesnik *Serializable*.

Primer:

```
public class Oseba implements Serializable {  
    private String ime;  
  
    public Oseba(String ime) {  
        this.ime = ime;  
    }  
}
```

ker Oseba implementira vmesnik *Serializable*, bo java objekte tega razreda znala avtomatsko serializirati/deserializirati





# ObjectInputStream in ObjectOutputStream

- Tok `ObjectOutputStream` uporabljamo za serializacijo objektov in zapis v zlogovno datoteko.
- Tok `ObjectInputStream` uporabljamo za deserializacijo objektov, zapisanih v zlogovnih datotekah.

```
Oseba o = new Oseba("Janez");
```

dostop do datoteke "imenik.dat" predo tokov tipa "data"

```
ObjectOutputStream oos = new ObjectOutputStream(...);  
ObjectInputStream ois = new ObjectInputStream (...);
```

ime: "Janez"

o ... objekt tipa Oseba

`oos.writeObject(o);`

`Oseba no = ois.readObject();`

ime: "Janez"

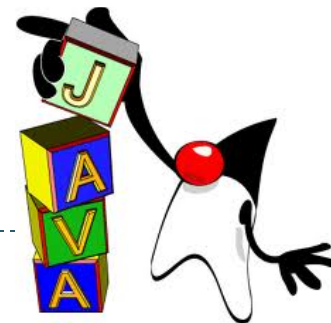
no ... objekt tipa Oseba

```
kepica:code tomaz$ hexdump -C imenik.dat
```

```
00000000  ac ed 00 05 73 72 00 0e  64 61 74 6f 74 65 6b 65  |....sr..datotekel|  
00000010  2e 4f 73 65 62 61 df e1  da 79 67 8b 5b d6 02 00  |.Oseba...yg.[...|  
00000020  01 4c 00 03 69 6d 65 74  00 12 4c 6a 61 76 61 2f  |.L..imet..Ljava/|  
00000030  6c 61 6e 67 2f 53 74 72  69 6e 67 3b 78 70 74 00  |lang/String;xpt.l|  
00000040  05 4a 61 6e 65 7a                |.Janez|
```







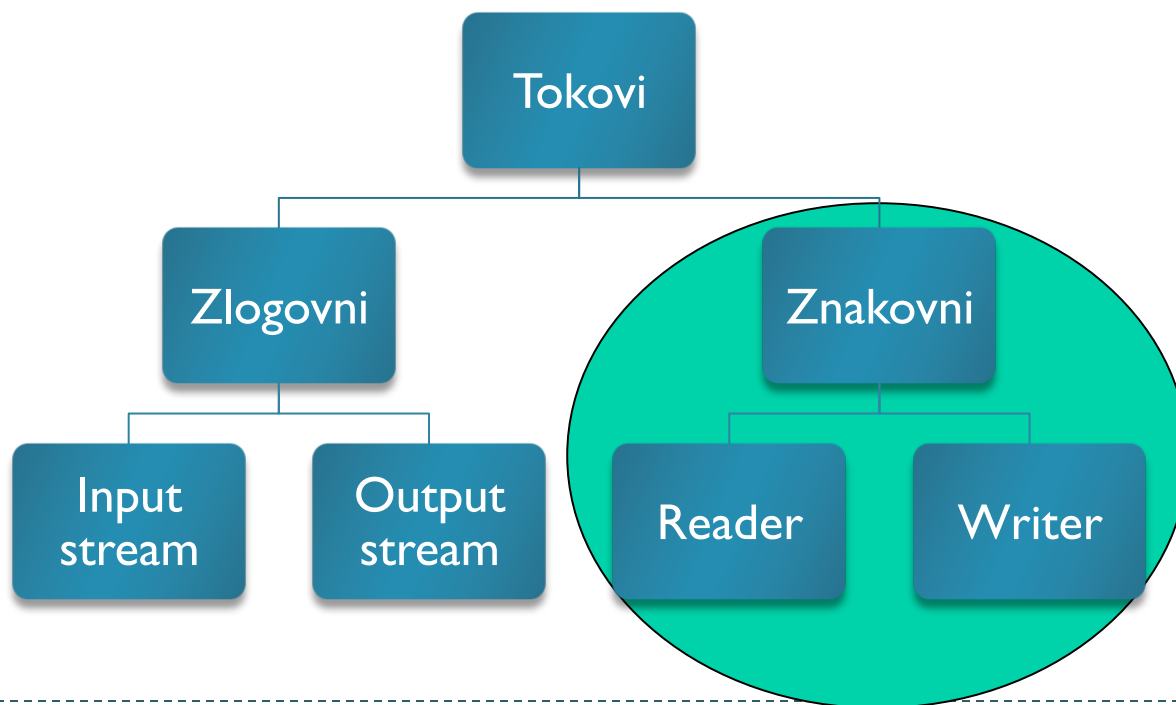
- ▶ Napiši razred `Oseba`, v katerem hraniš podatke (ime, priimek in starost) o neki osebi. Napiši razred `Imenik`, ki omogoča pisanje in branje objektov tipa `Oseba`.





# Zlogovni tokovi

- ▶ Znakovni tokovi se uporabljajo za branje/pisanje podatkov, ki so shranjeni v znakovni obliki, na primer v besedilnih datotekah s končnici `txt`.
- ▶ Velikost podatka v znakovni datoteki je odvisen od načina kodiranja.



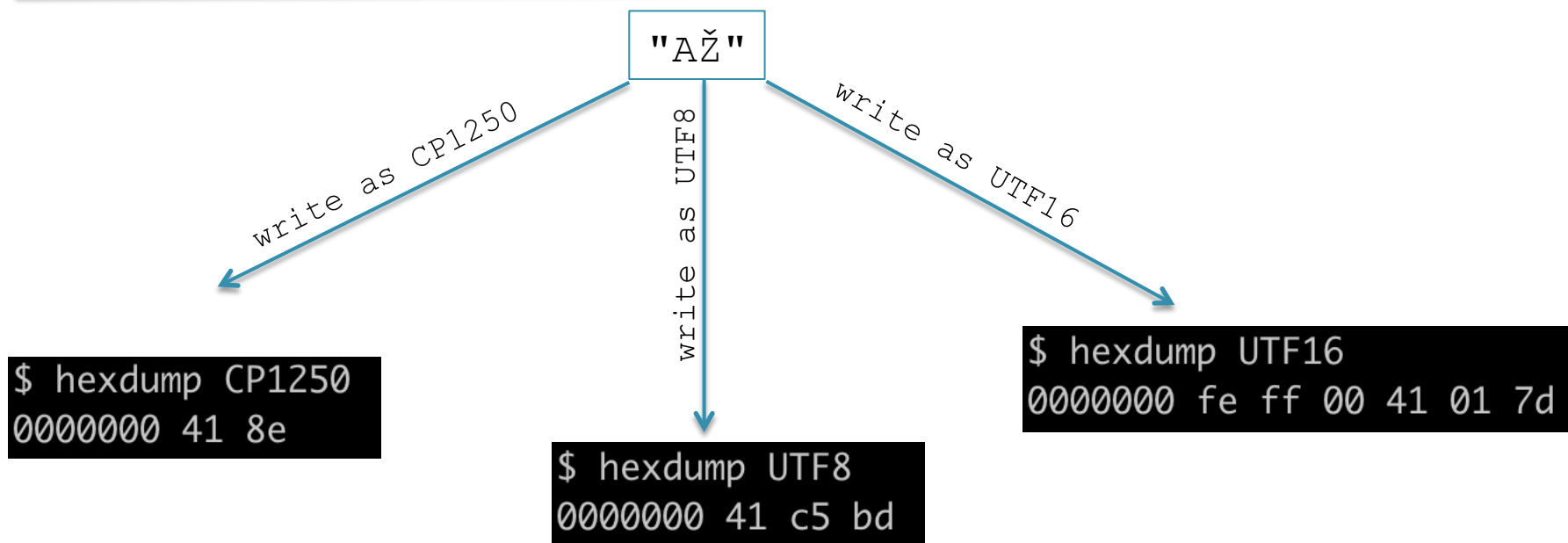


# Načini kodiranja

Poznamo več načinov kodiranja tekstovnih datotek, na primer:

- **ASCII**: 128 standardnih znakov; znak je v datoteki zapisan z enim bajtom,
- **CP1250** (Windows Central European): 256 različnih znakov (poleg standardnih ASCII še posebni znaki, na primer: čšžČŠŽŇŘó...); znak je v datoteki zapisan z enim bajtom,
- **UTF8**: vsi Unicode znaki (65536 znakov); znak je v datoteki zapisan z enim ali dvema bajtoma
- **UTF16**: vsi Unicode znaki (65536 znakov); znak je v datoteki zapisan z dvema bajtoma

Pozor: podatek o izbranem načinu kodiranja (praviloma) NI zapisan v datoteki!





# Znakovni tokovi

---

## Osnovne smernice pri uporabi znakovnih tokov:

- ▶ Za branje in pisanje znak po znaku uporabim

`InputStreamReader` in `OutputStreamWriter`

- ▶ Za formatirano branje in pisanje uporabim

`Scanner` in `PrintWriter`

- ▶ Ne glede na izbrani način branja in pisanja, vedno imam možnost nastavitve privzetega načina kodiranja znakov.



## InputStreamReader in OutputStreamWriter

---

- Tokova `InputStreamReader` in `OutputStreamWriter` uporabljamo za branje/pisanje znakov datoteke.
- Za kodiranje poskrbi java (na primer, ko preberemo znak, java sama ugotovi, ali mora prebrati 1 ali 2 bajta).
- Če uporabnik kodiranja ne določi eksplicitno, se uporabi sistemsko privzet način.





# InputStreamReader in OutputStreamWriter

---

- Nekateri konstruktorji:

```
InputStreamReader(InputStream in)
```

```
InputStreamReader(InputStream in, String charsetName)
```

```
OutputStreamWriter(OutputStream out)
```

```
OutputStreamWriter(OutputStream out, String charsetName)
```

- Nekatere metode

```
String getEncoding()
```

```
int read()
```

```
int read(char[] cbuf, int offset, int length)
```

```
boolean ready()
```

```
void close()
```

```
String getEncoding()
```

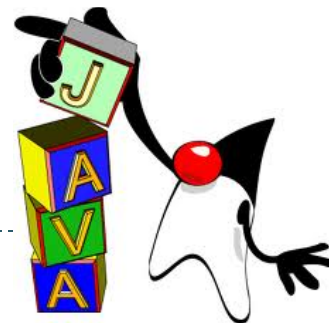
```
void write(int c)
```

```
void write(char[] cbuf, int off, int len)
```

```
void close()
```



Vhod in izhod (delo z datotekami)



Napiši program, ki omogoča pretvorbo iz enega v drug kodirni sistem. Ime vhodne in izhodne datoteke ter imena obeh kodirnih sistemov so podani kot argumenti ob klicu programa.





# Scanner

---

- Razred `Scanner` se uporablja za branje besedila in za razbijanje prebranega besedila na posamezne dele.
- Pri razbijanju besedila se uporablja ločilo (delimiter).
- Privzeta ločila: tabulator, presledek, nova vrsta.
- Posamezni deli besedila se lahko avtomatsko pretvorijo v javanske primitivne tipe.

Primer branja celega števila iz standardnega vhoda:

```
Scanner sc = new Scanner(System.in);  
int i = sc.nextInt();
```







# Scanner

---

- Nekateri konstruktorji:


```
public Scanner(File source)
public Scanner(File source, String charsetName)
public Scanner(InputStream source)
public Scanner(String source)
```

- Nekatere metode:

```
public boolean hasNextInt()
public int nextInt()
public boolean hasNextBoolean()
public boolean nextBoolean()
```

## Primer uporabe:

```
Scanner sc = new Scanner(new File("podatki.txt"));
boolean jeRes    = sc.nextBoolean();
int koliko      = sc.nextInt();
String letniCas = sc.nextLine();
sc.close();
```



**true**  
**32**  
**Pomlad**





# PrintWriter

---

- `PrintWriter` je ovojni tok, ki se uporablja za formatiran izpis besedila
- `PrintWriter` izpisuje primitivne javanske tipe v tekstovni obliki
- Za kodiranje podatkov poskrbi oviti tok.
- Nekateri konstruktorji:

```
public PrintWriter (OutputStream out)  
public PrintWriter (Writer out)
```





# PrintWriter

---

- Nekatere metode


```
public void print(char c)
```

```
public void print(int i)
```

```
public void print(double d)
```

## Primer uporabe:

```
PrintWriter pw = new PrintWriter("podatki.txt");  
pw.print(true);  
pw.println(32);  
pw.println("Pomlad");  
pw.close();
```



<b>true</b>
<b>32</b>
<b>Pomlad</b>





# Branje iz drugih virov

---

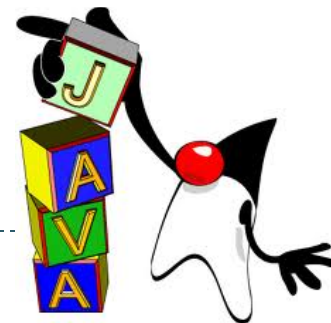
- Za branje podatkov lahko poleg tipkovnice in datoteke uporabljamo tudi druge vire (niz, spomin, internet, ...)
- Primer branja podatkov iz internetne strani:

```
URL yahoo = new URL("http://www.yahoo.com");
Scanner sc = new Scanner(yahoo.openStream());
while (sc.hasNextLine()) {
    System.out.println(sc.nextLine());
}
sc.close();
```



# Naloga

## Branje iz drugih virov



io/Bitcoin.java

Napiši program, ki izpiše trenutno vrednost izbrane kriptovalute.

Podatke o vrednosti kriptovalut dobiš na strani

<https://min-api.cryptocompare.com/data/price>

Parametra:

izvorna valuta: **fsym** (primer: BTC, ETH, ...)

ciljna valuta: **tsyms** (primer: EUR, USD, ...)

Primer izpisa programa:

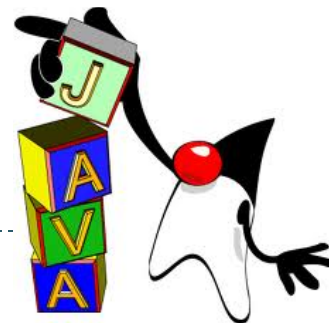
**1 BTC = 41887.25 EUR**



Vhod in izhod (delo z datotekami)

# Naloga

## Branje iz drugih virov



io/Temperatura.java

Napiši program, ki izpiše trenutno temperaturo v Ljubljani. Podatke o temperaturi preberi na strani <https://meteo.arso.gov.si/met/sl/service/>

[https://meteo.arso.gov.si/uploads/probase/www/observ/surface/text/sl/observation\\_si\\_latest.html](https://meteo.arso.gov.si/uploads/probase/www/observ/surface/text/sl/observation_si_latest.html)

<b>TOREK, 21.05.2019 22:00 CEST</b>	<b>Oblačnost</b>	<b>Pojavi</b>	<b>Temperatura [°C]</b>	<b>Veter</b>	<b>Hitrost vetra [km/h]</b>	<b>Sunki vetra [km/h]</b>	<b>Tlak [hPa]</b>	<b>Tendenca tlaka</b>	<b>Padavine v zadnjih 24h [mm]</b>
<b>Bilje pri Novi Gorici</b>		oblačno	14				1013		
<b>Celje</b>			11	↑	4		1011		
<b>Črnomelj</b>			16	↘	7		1012		
<b>Kredarica</b>		v oblakih	0	↘↗	36	58	* 746		
<b>Letališče Cerklje ob Krki</b>		zmerno oblačno	13	↗	7		1012		
<b>Letališče Edvarda Rusjana Maribor</b>		pretežno oblačno	14	↗	11		1010		
<b>Letališče Jožeta Pučnika Ljubljana</b>		pretežno oblačno	12	→	0		1012		
<b>Letališče Lesce</b>			11	↓	0		1012		
<b>Letališče Portorož</b>		oblačno	15	↑	0		1013		

```
<td class="meteoSI-th">Ljubljana</td><td class="nn_icon_wwsyn_icon"></td><td class="wwsyn_longText">14</td><td class="ddff_icon"></td><td class="ff_val">11</td><td class="ffmax_val">&nbsp;</td><td class="pa_shortText">&nbsp;</td><td class="rr24h_val">&nbsp;</td>
```



Vhod in izhod (delo z datotekami)