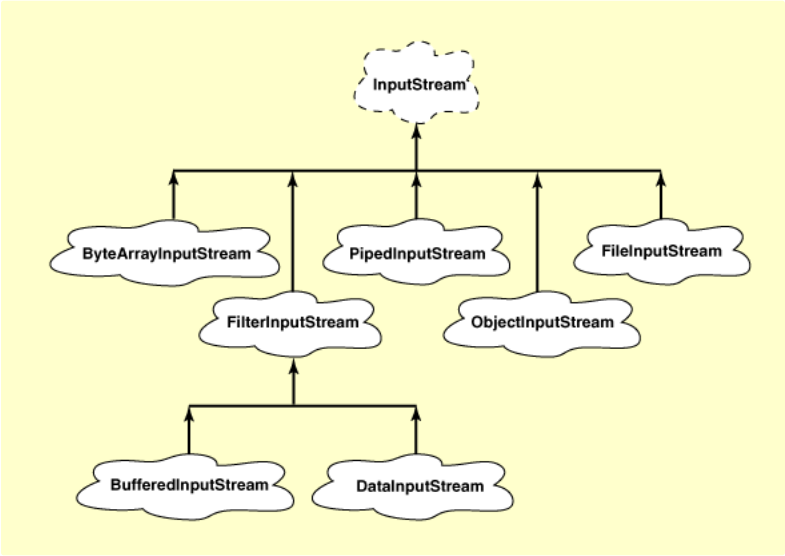
Dist 1.1

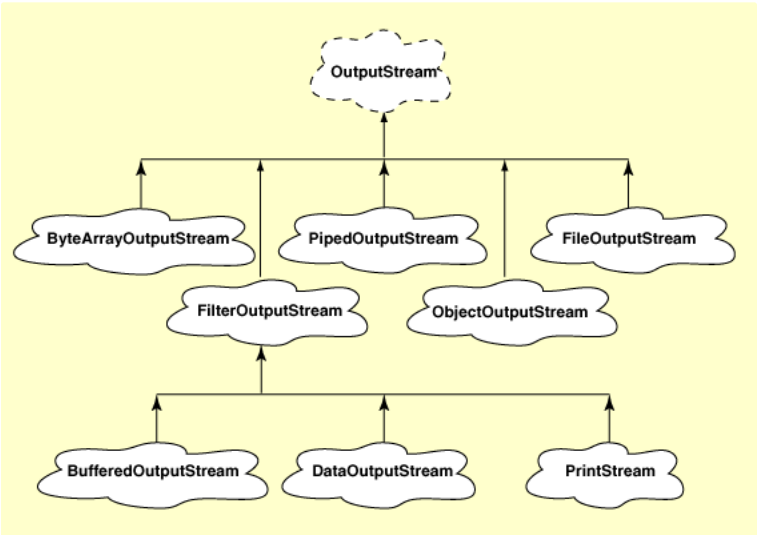
Normal

- #fleeting

- Stream moze da se zamisli kao proces prenosa podataka, neko prebacivanje deo po deo

Byte stream h1

- 8 bita
- Postoje 2 osnovna tipa uz kojih se izvode svi ostali
  - InputStream
    - 

      - Izvor: https://javadoc6dummies.blogspot.com/2013/02/javaio-class-hierarchy-diagram.html
  - OutputStream
    - 

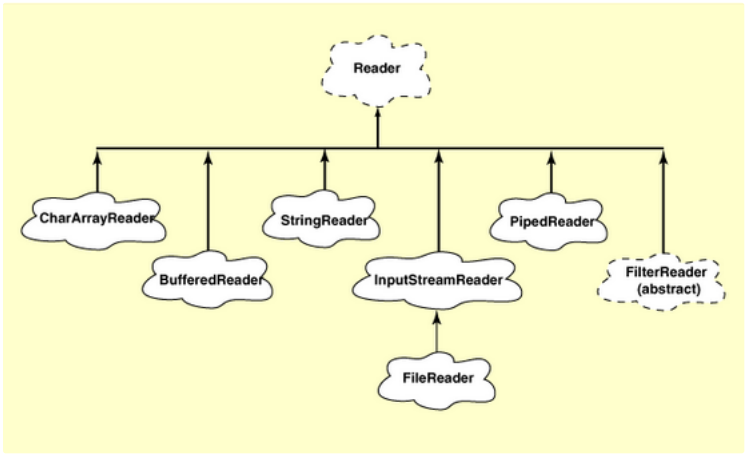
      - Izvor: https://javadoc6dummies.blogspot.com/2013/02/javaio-class-hierarchy-diagram.html
    - java

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

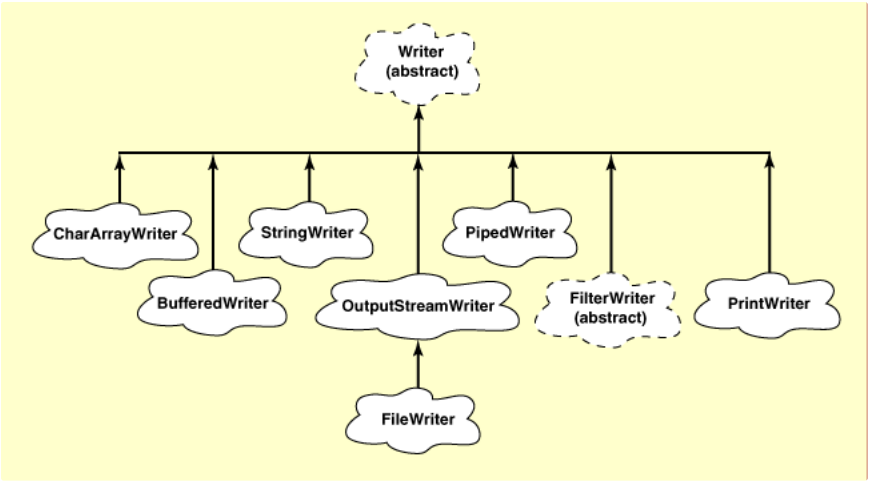
```
    try {
      in = new FileInputStream("i.txt");
      out = new FileOutputStream("o.txt");

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

- `InputStream` klasa i njene izvedene klase imaju metod `read` koji vraća `int` vrednost između `0` i `255`, a ukoliko nema više bajtova vraća se `-1`, osim ako ne dođe do exception-a.
- `OutputStream` klasa i njene izvedene klase koriste poslednjih `8` bitova ukoliko se šalje `int` ili niz bajtova ukoliko se prosledi niz bajtova

Character stream

- Čitaju se karakteri, nevezano za veličinu / encoding
- Dođe oko `16` bita pošto Java podrazumevano koristi Unicode
- Bitno je samo da se zna encoding pri čitanju
- Postoje 2 osnovna tipa iz kojih su drugi izvedeni
  - `Reader`
    - 

      - Izvor: http://javadoc4dummies.blogspot.com/2013/02/javaio-class-hierarchy-diagram.html
  - `Writer`
    -



- Izvor: http://javadoc4dummies.blogspot.com/2013/02/javaio-class-hierarchy-diagram.html

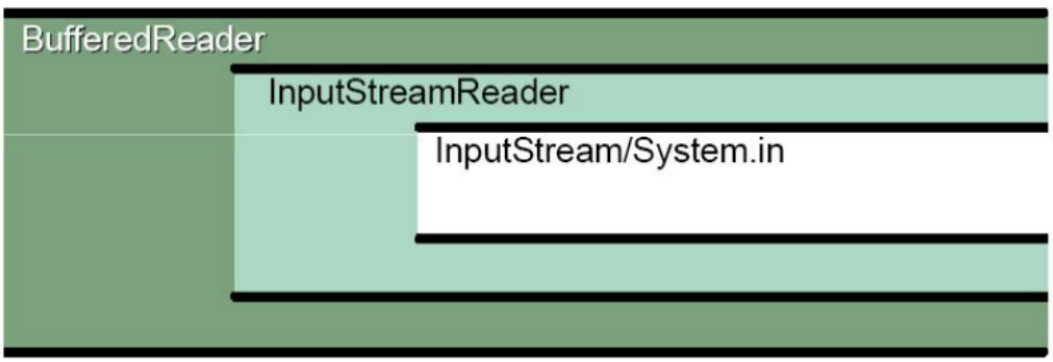
- Kod je isti kao za byte stream, s tim što se koriste klase `FileReader` i `FileWriter`

Konverzija između byte i character toka h1

- Konverzija byte stream-a u character stream se vrsi pomocu `InputStreamReader`
  - java
    ```
    // `System.in` je Byte Stream
    InputStreamReader isr = new InputStreamReader(System.in);
    ```
  - 

- Konverzija character stream-a u byte stream se vrsi pomocu `OutputStreamWriter`
  - java
    ```
    // `System.out` je Byte Stream
    OutputStreamWriter osw = new OutputStreamWriter(System.out);
    ```
  - 

Omotač tokova h1

- 

  - svaki "sloj" sadrži instancu "sloja" ispod sebe
  - `InputStream` je zadužen za rad sa bajtovima,

`InputStreamReader` je zadužen za rad sa karakterima, a `BufferedReader` za rad sa rečima

Klasa `BufferedReader` h1

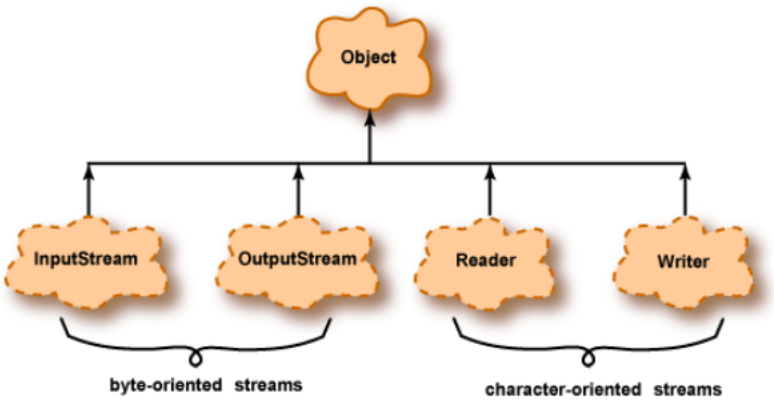
- Ima metod `.readLine()` kojim se ucitava cela linija bez prekidnog karaktera. To može da bude: `\n` , `\t` ili `EOF`
  - ovaj metod vraća `null` kada završi sa čitanjem (kada više nema šta da se pročita)
-

> Savetuje se da se ova klasa koristi za wrap-ovanje `Reader` -a čije operacija `read()` je veoma spora. `BufferedReader` koristi baferisanje što omogućava efikasniji rad

- Njegovim oslobađanjem se oslobađaju i resursi sa kojima upravlja (npr. instance "slojeva" ispod)
- Čitanje se vrši u `try/catch` bloku, a u `finally` bloku se takođe koristi `try/catch` blok kako bi se obuhvatio `.close()` koji može uzrokovati exception

Po čemu se razlikuje `FileInputStream` od `FileReader` -a? h1

- Te klase se pre svega razlikuju po hijerarhiji nasleđivanja. Prva klasa je naslednik klase `InputStream` , a druga naslednik klase `Reader`
  -



  - Izvor: https://chortle.ccsu.edu/java5/Notes/chap82/ch82_7.html
- Imaju različite svrhe: prva se koristi za čitanje **bajtova**, a druga za čitanje **teksta**
- Prva može da učitava jedan bajt ili niz bajtova, a druga jedan karakter ili više njih
- java

```java
// FileInputStream
new FileInputStream("fajl.txt");

// FileReader
new FileReader("fajl.txt");
```

-

> `FileReader` je do Jave 11 koristio podrazumevani encoding računara na kom se program izvršava što može da dovede do greške u čitanju ukoliko sadržaj nije encoding-a. U tom slučaju se preporučuje korišćenje `FileInputStream` koji se wrap-uje sa `InputStreamReader`

Razlika između `FileWriter` i `PrintWriter` ? h1

- `PrintWriter` ima dodatne metode za formatiranje

- `PrintWriter` ima opciju da se pri kreiranju predefiniše flushing tako da se radi automatski
  - i kod jednog i kod drugog (slučaju kada `PrintWriter` nije podešen da radi automatski flush) treba na kraju izvršiti flush (`x.flush()`), za slučaj da je neki sadržaj još uvek ostao u baferu
- `FileWriter` baca `IOExeption` , dok `PrintWriter` setuje interno neki `bool` koji se može proveriti metodom `checkError()`

- # ~~Lakše pamćenje~~ h1

-