

# Java

Ulazno-izlazni podsistem, klasa String

# Ulazno izlazni podsistem

- java.io – Standardna biblioteka za ulazno/izlazne operacije
- Izvorišta/odredišta:
  - memorija
  - fajl sistem
  - mrežne konekcije
- Oslanja se na tokove (streams) i čitače/pisače (reader/writer)
- Nezavisno od tokova/čitača postoji i RandomAccessFile klasa i File klasa

# File klasa

- ▶ za manipulaciju datotekama i direktorijumima:
  - kreiranje datoteka i direktorijuma
  - brisanje datoteka i direktorijuma
  - pristup atributima datoteka i direktorijuma
  - modifikacija naziva i atributa datoteka i direktorijuma

# File klasa

## ▶ Primeri upotrebe:

- `File f = new File(".");`
- `File f = new File("C:\\Windows");`
- `File f = new File("C:\\Windows\\proba.txt");`
  
- `if(f.exists()) ...`
- `if(f.isDirectory()) ...`
- `f.createNewFile();`

# Tokovi (streams) 1 / 4

- ▶ Bazirani na bajtovima
  - prenos jednog bajta
  - prenos niza bajtova
- ▶ Omogućuju prenos podataka:
  - datoteke (FileInputStream, FileOutputStream)
  - niza bajtova (ByteArrayInputStream, ByteArrayOutputStream)
  - sekvence drugih tokova (SequenceInputStream)
  - itd.

# Tokovi (streams) <sub>2/4</sub>

- ▶ Osmišljeni kao mehanizam koji omogućuje unificiran pristup podacima
- ▶ Isti kod se koristi za čitanje/pisanje iz, na primer, datoteke ili mrežne konekcije
- ▶ Metode u tokovima:
  - `read()` – čita jedan bajt iz toka
  - `read(byte[])` – čita niz bajtova
  - `skip(long n)` – preskače zadati broj bajtova
  - `available()` – vraća broj raspoloživih bajtova iz toka koji se mogu pročitati pre blokiranja sledećeg čitanja
  - `close()` – zatvara tok

# Tokovi (streams) <sub>3/4</sub>

- ▶ Primer upotrebe – kopiranje sadržaja datoteke:

```
byte[] buffer = new byte[BUFFER_LENGTH];  
while((read=in.read(buffer, 0,BUFFER_LENGTH)) != -1) {  
    // obrada učitanoog niza bajtova  
    out.write(buffer, 0 , read);  
}
```

# Tokovi (streams) 4/4

- ▶ Koncept filtera – donose dodatnu funkcionalnost tokovima:
  - prenos primitivnih tipova na mašinski nezavisan način (DataInputStream, DataOutputStream)
  - baferizovan prenos podataka (BufferedInputStream, BufferedOutputStream)
  - prenos objekata (ObjectInputStream, ObjectOutputStream)
- ▶ Primer kreiranja stream-a uz upotrebu filtera

```
DataOutputStream out =  
    new DataOutputStream(  
        new BufferedOutputStream(  
            new FileOutputStream("testStream.dat")));
```



# Čitači/pisači (readers/writers) <sub>1/3</sub>

- ▶ Ispravljaju problem sa tokovima – slabu podršku Unicode rasporedu:
  - tokovi ne prenose dobro Unicode stringove
  - poseban problem predstavljaju različite hardverske platforme (*little-endian, big-endian*)
- ▶ Čitači/pisači ne zamenjuju tokove – oni ih dopunjuju
- ▶ Čitači/pisači se koriste kada je potrebno preneti Unicode stringove ili karaktere – u ostalim situacijama koriste se tokovi

# Čitači/pisači (readers/writers) <sub>2/3</sub>

- ▶ Omogućuju prenos karaktera iz/u:
  - datoteke (FileReader/FileWriter)
  - druge nizove karaktera (CharArrayReader/CharArrayWriter)
  - stringove (StringReader/StringWriter)
- ▶ Klase za spregu tokova i čitača/pisača – InputStreamReader, OutputStreamWriter:  
`BufferedReader in = new BufferedReader(  
 new InputStreamReader(System.in));`

# Čitači / pisači (readers / writers)

3 / 3

- ▶ Metode u čitačima:
  - read() – čita jedan karakter iz toka
  - read(char[]) – čita niz karaktera
  - skip(long n) – preskače zadati broj karaktera
  - close() – zatvara čitač

# Čitanje/pisanje stringova

- ▶ Koriste se klase `BufferedReader` i `PrintWriter` oko `FileReader`-a i `FileWriter`-a
- ▶ `BufferedReader` ima metodu `readLine`
- ▶ `PrintWriter` ima metodu `println`
- ▶ Primer:

```
BufferedReader in = new BufferedReader(new
    FileReader("testReaderWriter.dat"));
String s2;
while((s2 = in.readLine()) != null) {
    System.out.println(s2);
}
```

# Sprežne klase

- ▶ `InputStreamReader` i `OutputStreamWriter` služe za ručno sprezanje tokova i čitača/pisača

- ▶ Primer:

```
BufferedReader in = new BufferedReader(  
    new InputStreamReader(new FileInputStream(  
        "rezultati.csv"), "UTF8"));  
String s2;  
while((s2 = in.readLine()) != null) {  
    System.out.println(s2);  
}
```

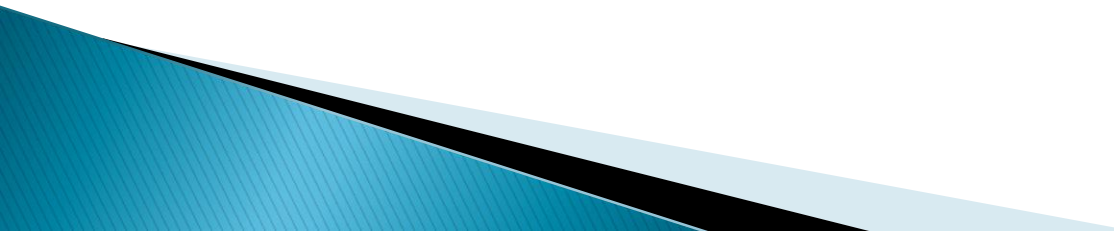
# Serijalizacija objekata

- ▶ Serijalizacija objekta – prevođenje objekta u niz bajtova i njegova rekonstrukcija iz niza u “živ” objekat
- ▶ Serijalizovan niz bajtova se može snimiti u datoteku ili poslati preko mreže – i jedno i drugo upotrebom tokova
- ▶ Da bi se neki objekat serijalizovao:
  - potrebno je da implementira `java.io.Serializable` interfejs
  - da su atributi i parametri metoda takođe serijalizabilni
- ▶ Primitivni tipovi su serijalizabilni
- ▶ Većina bibliotečkih klasa je serijalizabilna

# Serijalizacija objekata

- ▶ U Javi postoji ključna reč *transient* koja se može staviti uz atribut, a ona označava da vrednost atributa ne bude preneta postupkom serijalizacije.
- ▶ Ako ovu ključnu reč stavimo uz atribut koji je primitivni tip, po rekonstrukciji objekta, u njemu će biti podrazumevana vrednost za taj tip (nula za int, na primer), a *null* literal za reference

# Rad sa arhivama

- ▶ podržan rad sa GZip i Zip formatima arhiva
  - ▶ klase koje podržavaju rad sa arhivama:
    - GZipInputStream, GZipOutputStream
    - ZipInputStream, ZipOutputStream
    - ZipFile – za pojednostavljeno čitanje i ekstrakciju zip arhiva
    - ZipEntry – reprezentuje kompresovanu datoteku u arhivi
- 



# Štampanje na ekran

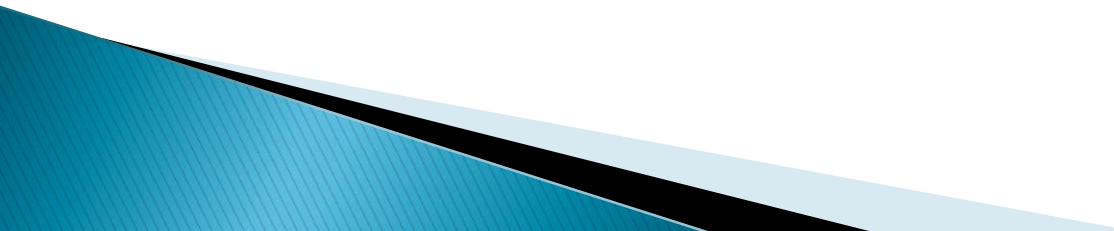
- ▶ Već smo čuli da je `System.out` izlazni tok:

```
System.out.print("Poruka");
```

```
System.out.println("Poruka");
```

- ▶ Ispis se može i formatirati:

```
System.out.printf("format", argumenti);
```



# Unos sa tastature

- ▶ System.in je ulazni tok:

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

```
String s = in.readLine();
```

- ▶ Unos nečega što nije string – koristi se wrapper klasa i njena metoda parseInt():

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

```
String s = in.readLine();
```

```
int i = Integer.parseInt(s);
```



# Wrapper klase za primitivne tipove

- ▶ Za sve primitivne tipove postoje odgovarajuće klase:
  - `int` → `Integer`
  - `long` → `Long`
  - `boolean` → `Boolean`
- ▶ Imaju statičku metodu `Xxxx.parseXxxx()`
  - `int i = Integer.parseInt("10")`
  - `long l = Long.parseLong("10")`
- ▶ Ove wrapper klase rade automatski boxing i unboxing, odnosno automatsku konverziju primitivnih tipova u objekte i obrnuto kada je to potrebno

# Unos sa tastature – klasa Scanner

- ▶ Alternativa je klasa Scanner koja ne učitava samo stringove, odnosno klasa Scanner služi za unos stringova i primitivnih tipova iz tekstualnih ulaza, radi kao jednostavan parser teksta koji je u stanju da iz tekstualnog ulaza izdvoji stringove po nekom obrascu, nakon izdvajanja stringa, u stanju je da konvertuje taj string u traženi primitivni tip:

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

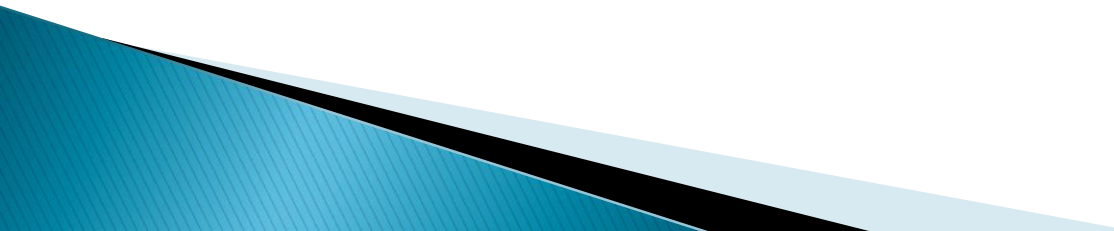
```
String s = sc.nextLine();
```

```
int i = sc.nextInt();
```

```
float f = sc.nextFloat();
```



# Klasa Scanner

- ▶ Klasa Scanner služi za unos stringova i primitivnih tipova iz tekstualnih ulaza.
  - ▶ Ona radi kao jednostavan parser teksta koji je u stanju da iz tekstualnog ulaza izdvoji stringove po nekom obrascu.
  - ▶ Nakon izdvajanja stringa, u stanju je da konvertuje taj string u traženi primitivni tip.
- 

# Klasa Scanner

## ▶ Primer:

```
Scanner sc = new Scanner(System.in);  
System.out.print("Unesite string:");  
String s = sc.nextLine();  
System.out.print("Unesite int:");  
int i = sc.nextInt();  
// kada citamo primitivne tipove,  
// ne uklanja se ENTER  
sc.nextLine();  
System.out.println(s + ", " + i);  
sc.close();
```

# Zaključna razmatranja

- ▶ Podaci se u čitaju iz ulaznih tokova, a pišu u izlazne tokove
- ▶ Iz programa se retko radi direktno sa bajtovima
  - zato se tokovi ugrađuju u Filter klase koje imaju odgovarajuće metode za čitanje/pisanje
  - zato imamo tokove objekata, tokove primitivnih tipova itd.
- ▶ Ako radimo sa karakterima/stringovima, koristimo čitače i pisače
- ▶ Postoje posebne klase za rad sa tastaturom i ekranom
- ▶ Nezavisno od ovog postoji i klasa `java.io.File` za koja nam omogućava osnovne operacije nad fajl sistemom (kreiranje fajla, proveru da li postoji fajl, itd.)

# Klasa String

- ▶ Niz karaktera je podržan klasom String. String **nije** samo niz karaktera – on je klasa!
- ▶ Objekti klase String se ne mogu menjati (*immutable*)!
- ▶ Izmena stringa konkatencijom ili dodelom novog string literala kreira se novi objekat na heap-u – alternativa StringBuffer ili StringBuilder klasa.
- ▶ Za cast-ovanje String-a u neki primitivni tip koristi se wrapper klasa i njena metoda parseXxx():

```
int i = Integer.parseInt(s);
```

- ▶ Klasa Object – metoda toString, više o ovome kasnije tokom kursa.



# Klasa String - metoda

## ► Reprezentativne metode:

- `str.length()`
- `str.charAt(i)`
- `str.indexOf(s)`
- `str.substring(a,b)`, `str.substring(a)`
- `str.equals(s)`, `str.equalsIgnoreCase(s)` – ne koristiti ==
- `str.startsWith(s)`
- `str.toLowerCase()`

# Ispis na ekran

```
String s1 = "Ovo je";  
String s2 = "je string";  
System.out.println(s1.substring(2)); // o je  
System.out.println(s2.charAt(3)); // s  
System.out.println(s1.equals(s2)); //false  
System.out.println(s1.indexOf("je")); // 4 , ako nema podstringa vratice -1  
System.out.println(s2.length()); //9  
System.out.println(s1.trim()); //Ovo je , inace skida whitespaces sa pocetka i kraja  
System.out.println(s2.startsWith("je")); //true
```

# Redefinisan + operator sa stringovima

- ▶ Ako je jedan od operandata klase String, ceo izraz je string!

```
String a = "Vrednost i je: " + i;
```

- ▶ Drugi operand se konvertuje u string (pravi se njegova string reprezentacija):

```
int i = 5;
```

```
String a = "Vrednost i je: " + i;
```

```
5 → "5"
```



# Metoda `split()` klase `String`

- ▶ "cepa" osnovni string na niz stringova po zadanom šablonu
  - originalni string se ne menja
  - parametar je regularni izraz
  - rezultat je niz stringova na koje je „pocepan“ originalni string
- ▶ Poziv: `String[] rez = s.split("regex") ;`
- ▶ Alternativa ovomo je upotreba klase `StringTokenizer`

# Metoda split() klase String

```
class SplitTest {  
    public static void main(String args[]) {  
        String text = "Ovo je probni tekst";  
        String[] tokens = text.split(" ");  
        for (int i = 0; i < tokens.length; i++)  
            System.out.println(tokens[i]);  
    }  
}
```