**NAPOMENA:** Vežbe podrazumevaju da je student ovladao teorijom iz dela "Programski jezik Java - Ulazno izlazni podsistem" (IO.ppt).

# Ulazno izlazni podsistem (IO, java.io)

Predstavlja standardnu biblioteku za ulazno/izlazne operacije. Omogućava operacije nad različitim izvorištima tj odredištima poput memorije, fajl sistema i mrežne konekcije. Oslanja se na tokove (engl. *streams*) i čitače/pisače (engl. *reader/writer*).

*File* klasa (*java.io.File*) omogućuje sistemski nezavisan apstraktni pogled ka datotekama i direktorijumima. Ona omogućuje manipulaciju (kreiranje, izmena, brisanje) datoteka, direktorijuma i njihovih atributa. Jedina razlika korišćenja klase *File* u odnosu na sistem predstavlja separator direktorijuma u putanji do nekog direktorijuma ili datoteke.

Primer 1. Izlistavanje sistemskih podešavanja i podešavanje apsolutne i relativne putanje.

```
System.out.println("Izlistavanje svih podešavanja OS-a");
System.getProperties().list(System.out);
//preuzimanje određenog podešavanja - naziva OS-a
System.out.println(System.getProperty("os.name"));
//preuzimanje separatora putanje
String separatorPutanje = System.getProperty("file.separator");
//apsolutna putanja u windows os:  d:\\userName
String winAbsPath = "d:"+separatorPutanje+System.getProperty("user.name");
//apsolutna putanja u unix os:  /home/userName
String unixAbsPath = "/home"+separatorPutanje+System.getProperty("user.name");
//ispis apsolutinih putanja
System.out.println("Apsolutne putanje:\n" + winAbsPath+"\n"+unixAbsPath);
//ispis relativnih putanja
String relPath = "." + separatorPutanje + "Direktorijum";
System.out.println("Relativna putanja:\n" + relPath);
```

Primer 2. Napisati program koji omogućuje čitanje sadržaja direktorijuma, kreiranje direktorijuma i datoteka i provera njihovog postojanja.

```
// kreiranje objekta
File dir = new File("." + System.getProperty("file.separator") + "src1");
// provera da li postoji
if (dir.exists()) {
    System.out.println("Putanja apsolutna " + dir.getAbsolutePath());
    System.out.println("Putanja relativna " + dir.getPath());
    // provera da li je objekat File direktorijum
    if (dir.isDirectory()){
        System.out.println(dir.getName() + " je direkorijum");
        System.out.println(dir.listFiles().length);
    }else{ if (dir.isFile())
        System.out.println(dir.getName() + " je datoteka");
    }dir.delete();
} else {
    dir.mkdir();
    // ili
    dir.createNewFile();
}
```

Tokovi su bazirani na bajtovima tj prenos niza bajtova - bajt po bajt. Omogućuju mehanizme čitanja/pisanja nezavisno od izvorišta / odredišta. Dakle, omogućuju rad sa datotekama (*FileInputStream*, *FileOutputStream*), niza bajtova (*ByteArrayInputStream*, *ByteArrayOutputStream*), sekvence drugih tokova (*SequenceInputStream*) itd.

Primer 3. Napisati program koji će vršiti upis i čitanje u datoteku korišćenjem *FileInputStream* i *FileOutputStream*.

```
String sp = System.getProperty("file.separator");
FileOutputStream os = new FileOutputStream(new File("."+sp+"tmp.txt"),true);
//FileOutputStream os = new FileOutputStream(".\\tmp.txt");
os.write(new byte[] {82, 83, 84});
os.close();

FileInputStream is = new FileInputStream("."+sp+"tmp.txt");
int b = 0;
System.out.println(is.available());
while ((b = is.read()) != -1) {
    System.out.print(b+" ");
}
is.close();
```

Koncept filtera donose dodatnu funkcionalnost tokovima. Neki od filter klasa su: *BufferedInputStream*, *BufferedOutputStream*, *DataInputStream*, *DataOutputStream*, *ObjectInputStream* i *ObjectOutputStream*.

Primer 4. Napisati program koji će vršiti upis i čitanje u datoteku korišćenjem *DataInputStream* i *DataOutputStream*.

```
String sp = System.getProperty("file.separator");
DataOutputStream dataOutputStream =
        new DataOutputStream(
                new FileOutputStream("."+sp+"tmp.bin"));

dataOutputStream.writeInt(12);
dataOutputStream.writeFloat(12.34F);
dataOutputStream.writeDouble(3.1456778);

dataOutputStream.close();


DataInputStream dataInputStream =
        new DataInputStream(
                new FileInputStream("."+sp+"tmp.bin"));

System.out.println("int: " + dataInputStream.readInt());
System.out.println("float: " + dataInputStream.readFloat());
System.out.println("double: " + dataInputStream.readLong()); // <-

dataInputStream.close();
```

Čitači/pisači omogućavaju dodatne mehanizme čitanja/pisanja i rešavaju problem tokova koji je vezan za rad sa stringovima sa *Unicode* rasporedom. Tačnije, problem predstavljaju različite hardverske platforme (engl. *little-endian*, *big-endian*) zbog kojih tokovi nisu pouzdani. Iz tih razloga, čitači/pisači se koriste kada je potrebno preneti *Unicode* stringove ili karaktere, dok se u ostalim situacijama koriste tokovi. Dakle, omogućuju rad sa datotekama (*FileReader*, *FileWriter*), nizovima karaktera (*CharArrayReader*, *CharArrayWriter*), stringovima (*StringReader*, *StringWriter*) itd. Spregu tokova i čitača/pisača se obezbeđuje klasama *InputStreamReader* i *OutputStreamWriter*. Dodatno se koriste i klase *BufferedReader* i *PrintWriter*, gde *BufferedReader* poseduje metodu *readLine* a klasa *PrintWriter* metodu *println*.

Primer 5. Napisati program koji će vršiti upis i čitanje u datoteku korišćenjem *BufferedReader* i *PrintWriter*.

```
String sp = System.getProperty("file.separator");
PrintWriter out = new PrintWriter(new FileWriter("."+sp+"tmp.txt",true));
out.printf("Rezultat je: %.4f \n", 3.123456);
out.close();
String sCurrentLine;

BufferedReader br = new BufferedReader(new FileReader("."+sp+"tmp.txt"));

while ((sCurrentLine = br.readLine()) != null) {
    System.out.println(sCurrentLine);
}
```

Serijalizacija objekta omogućuje prevođenje objekta u niz bajtova i njegova rekonstrukcuja iz niza u aktivan objekat u memoriji. Prilikom serijalizacije, serijalizuju se osim samog objekta i njegovi atributi – stablo serijalizovanih objekata. Niz bajtova koji predstavlja serijalizovani objekat može se upotrebom odgovarajućih tokova upisati u fajl ili poslati preko mreže. Čitanjem iz fajla moguće je izvršiti rekonstrukcija sačuvanih objekta.

Da bi se neki objekat serijalizovao potrebno je da dati objekat, kao i njegovi atributi i parametri metoda, implementiraju *java.io.Serializable* interfejs. Većina bibliotečkih klasa i primitivni tipovi su serijalizabilni. Ključna reč *transient* se stavlja uz atribut ako se želi naznačiti da vrednost atributa ne bude preneta postupkom serijalizacije. To znači da će nakon rekonstrukcije objekta atribut imati podrazimevanu inicijalnu vrednost.

Primer 6. Napisati program koji će vršiti upis i čitanje objekta u datoteku korišćenjem *ObjectInputStream* i *ObjectOutputStream*.

```
String sp = System.getProperty("file.separator");
ArrayList<Student> stud = new ArrayList<Student>();

ObjectOutputStream out = new ObjectOutputStream(
        new FileOutputStream("."+sp+"stud.dat"));
out.writeObject(stud);
out.writeBoolean(true);
out.writeInt(new Integer(123456));
out.flush();
out.close();

ObjectInputStream in = new ObjectInputStream(
        new FileInputStream("."+sp+"stud.dat"));
stud = (ArrayList<Student>) in.readObject();
System.out.println(in.readBoolean());
System.out.println(in.readInt());
in.close();
```

Klase GZipInputStream, GzipOutputStream, ZipInputStream, ZipOutputStream omogućavaju rad sa *GZip* i *Zip* formatima arhiva. Klasa *ZipFile* se koristi za pojednostavljeno čitanje i ekstraktovanje zip arhiva. Klasa *ZipEntry* se koristi za reprezentaciju kompresovane datoteke u arhivi.

Primer 7. Napisati program koji će kreirati arhivu *Zip* formata.

```
String sp = System.getProperty("file.separator");
ZipOutputStream zip = new ZipOutputStream(
        new FileOutputStream("."+sp+"arhiva.zip"));
ZipEntry file = new ZipEntry("direktorijum"+sp+"datoteka.txt");
zip.putNextEntry(file);

byte[] buffer = new byte[1024];
FileInputStream in = new FileInputStream("."+sp+"tmp.txt");

int len;
while ((len = in.read(buffer)) > 0) {
    zip.write(buffer, 0, len);
}
in.close();
zip.close();
```

Standardni izlaz na ekran i ulaz sa tastature je rađen na prethodnim časovima.

Zaključna razmatranja za ulazno izlazni podsistem:

- podaci se u čitaju iz ulaznih tokova, a pišu u izlazne tokove
- iz programa se retko radi direktno sa bajtovima nego se tokovi ugrađuju u *Filter* klase koje imaju odgovarajuće metode za čitanje/pisanje objekata, primitivnih tipova itd.
- za rad sa karakterima/stringovima, koristimo čitače i pisače
- za rad sa tastaturom i ekranom postoje posebne klase
- klasa *java.io.File* omogućava osnovne operacije nad fajl sistemom (kreiranje fajla, provera da li postoji fajl, itd.)

# Obrada izuzetaka

Tokom izvršavanja aplikacije mogu se javiti greške različitog nivoa prioriteta. Da bi aplikacija nesmetano funkcionisala, neophodno je obezbediti obradu mogućih grešaka. Postoje mehanizmi obrade izuzetaka odnosno prijave grešaka određenog tipa. Dakle, izuzeci se izazivaju kada se otkrije određena situacija za koju ne postoji odgovarajuća obrada.

U Javi izuzeci su realizovani kao objekti koji su izvedeni direktno od klase *Throwable* ili neke njene podklase. Hijerarhija izuzetaka je data u nastavku:

- *Throwable* – roditeljska klasa
    - Error – ozbiljne sistemske graške
    - *Exception* – bazna klasa za sve standardne izuzetke
        - *unchecked*: *RuntimeException* i njene naslednice koje ne moraju da se obuhvate *try/catch* blokom (npr. *NullPointerException*, IndexOutOfBoundsException itd)
        - *checked*: Ostale klase koje nasleđuju *Exception* klasu i koje moraju da se obuhvate *try/catch* blokom (npr. *EOFException*, *IOException* itd)

Obrada (hvatanje) izuzetaka se vrši uz pomoć *try/catch* bloka.

Primer 8. Definisati *try/catch* blok koji hvata različite tipove izuzetaka.

```
try {
  // kod koji može da izazove
  // izuzetak
}
catch (java.io.EOFException ex) {
  System.out.println("Kraj datoteke pre vremena!");
}
catch (IndexOutOfBoundsException ex) {
  System.out.println("Pristup van granica niza");
}
catch (Exception ex) {
  System.out.println("Svi ostali izuzeci");
}
finally {
  // kod koji se izvršava u svakom slučaju
}
```

Izuzeci se mogu programski izazvati korišćenjem ključne reči *throw*. U tom slučaju metoda u kojoj se izaziva određeni izuzetak mora imati deklaraciju datog tipa izuzetka koji izaziva.

Primer 9. Definisati funkciju koja izaziva izuzetak opšteg standardnog tipa *Exception*.

```
void func(boolean dat) throws Exception{
    if (dat != true) {
        throw new Exception("dat je false");
    }
}
```

Prilikom poziva metode koja izaziva određeni izuzetak, zahteva da se obuhvati *try/catch* blokom. Ipak to nije neophodno ako se dati izuzetak propagira na viši nivo tj. ako se odgovornost hvatanja izuzetka prebaci na pozivajuću metodu.

Primer 10. Pozvati metodu koja izaziva opšti tip izuzetka *Exception* i proslediti odgovornost hvatanja istog na viši nivo.

```
    void metoda() throws Exception{
        boolean dat = true;
//      try {
            func(dat);
//      } catch (Exception e) {
//          e.printStackTrace();
//      }
    }
```

Ukoliko ne postoji određeni tip izuzetka može se kreirati nov tip izuzetka koji omogućava da se određeni tip izuzetka uhvati i obradi na odgovarajući način. Pri tome se moraju naslediti odgovarajuća klasa izuzetka.

Primer 11. Definisati nov tip izuzetka i testirati na primeru.

```
class MyException extends Exception{
    private String details;
    MyException(String details) {
       this.details = details;
    }
    public String toString(){
       return ("Output = "+details) ;
    }
}
```

# Zadaci

Zadatak 1. Napisati program koji za uneti naziv (apsolutna ili relativna putanja) ispituje da li je datoteka ili direktorijum. Zatim ispisuje sadržaj datoteke/direktorijuma.

Zadatak 2. Napisati program koji proširuje prethodni zadatak (zadatak 1) i omogućuje ispis sadržaja svih poddirektorijuma koliko ih ima.

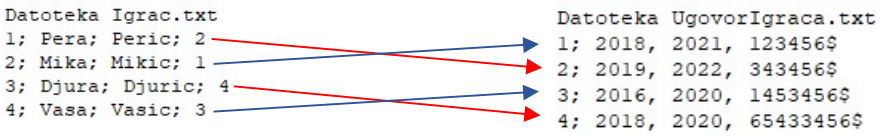
Zadatak 3. Napisati program koji na osnovu sadržaja datoteke *hijerarhija.txt* pravi hijerarhiju datoteka i direktorijuma. Primer sadržaja datoteke:

```
root
    usr
    home
        student
            Documents
                Student.java
                Zadatak.txt
            Desktop
                Zadatak.txt
    etc
```

Zadatak 4. Napisati program koji proširuje zadatak iz prethodnih vežbi (VEZBE_06) i omogućuje rad sa ulazno-izlaznim podsistemom. Za svaku klasu podatke čuvati u zasebnoj tekstulanoj datoteci a veze između njih definisati na sledeći način:

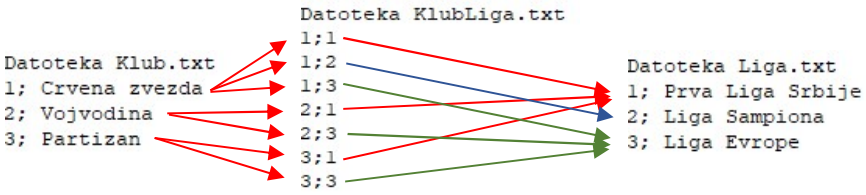
Veze klasa:

- 1 na 1




- * na 1 (1 na *)


- * na *




Napomene:

- Pri pokretanju programa se učitavaju podaci iz datoteka u odgovarajuće kolekcije a pri završavanju rada programa snimiti sve izmene u datoteke
  - Za projekat se izmene čuvaju nakon svake izmene objek(a)ta
- Prvo se učitavaju podaci iz datoteka glavnih entiteta a nakon toga podaci pomoćnih datoteka (npr. KlubLiga.txt)
- Prvo se kreiraju objekti entiteta a nakon toga njihove veze (asocijacije, agregacije)
  - Npr. U slučaju veze *na*, prvo se kreiraju objekti klase Klub i Liga, a nakon toga se povezuju
- Datoteke se uvek (na ovom kursu) prepisuju novim sadržajem tj. ne ispravljaju se pojedinačni redovi u datoteci
- Datoteke se čuvaju u zasebnom direktorijumu projekta (van *src* direktorijuma)
- Za lociranje datoteka se koriste relativne putanje