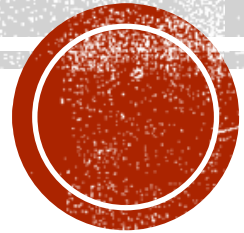
ОБЈЕКТНО ОРИЈЕНТИСАНО ПРОГРАМИРАЊЕ

ПРОГРАМСКИ ЈЕЗИК JAVA – 1

Улаз и излаз



УЛАЗ И ИЗЛАЗ

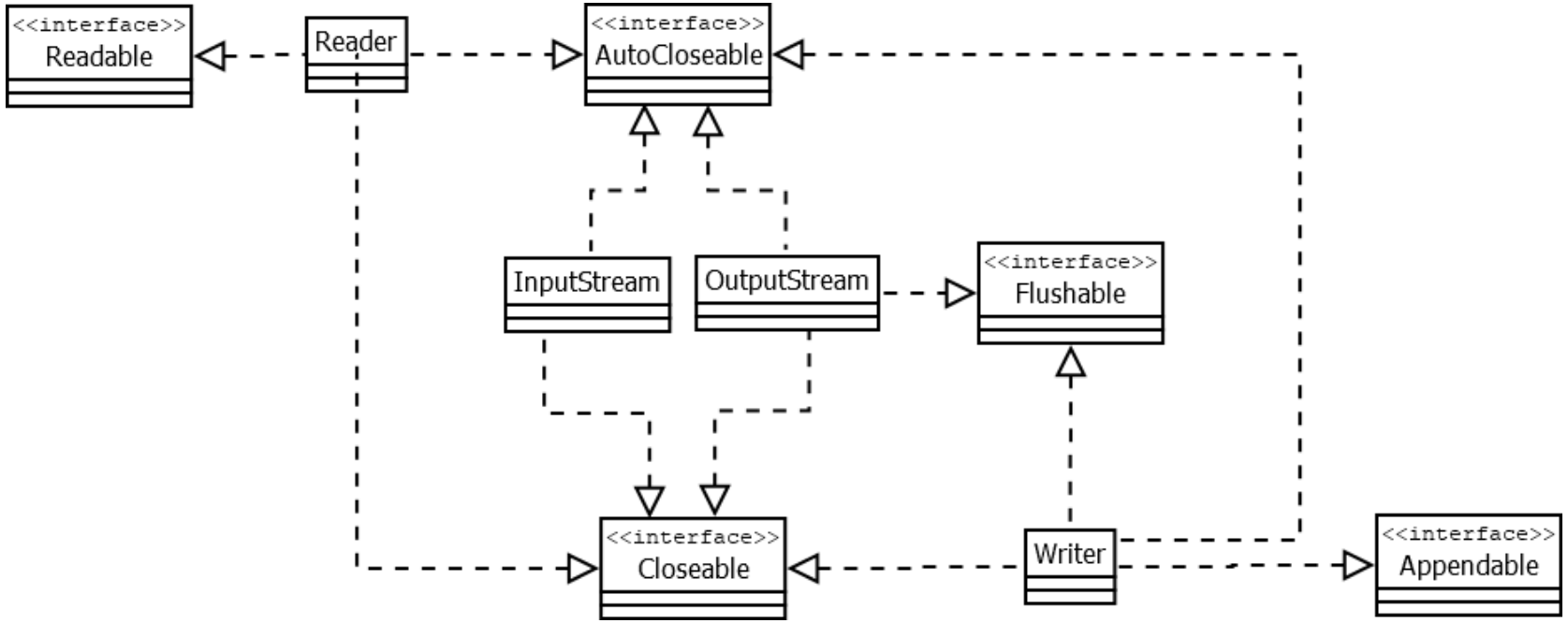
- Један од фундаменталних аспеката програмског језика је учитавање улазних података у програм и приказивање или складиштење излазних података.
- Објектно оријентисани програмски језици теже ка омогућавању униформне, вишеструко искористиве и прошириве могућности рада са улазом и излазом.
- У програмском језику C методи за форматирани улаз са конзоле, датотеке и ниске се реализују засебно: **scanf()**, **fscanf()** и **sscanf()**.
  - Са друге стране, у Јави се полази од интерфејса и апстрактних класа које прописују методе за читање са апстрактног улаза, а након тога се у конкретним изведеним класама реализују методи типове улаза попут: конзоле, датотеке, ниске, мрежног порта и слично.
- У Јави постоје две библиотеке за рад са улазом и излазом.
  - Прва је Јава IO библиотека која постоји још од самог почетка Јаве (користи блокирање).
  - Друга је новијег датума и зове се Јава NIO (енг. non-blocking IO, не користи блокирање).

БЛОКИРАЈУЋИ УЛАЗ И ИЗЛАЗ – JAVA.IO

- Јава IO реализује своје основне функционалности кроз употребу блокирајућих метода за читање и писање (**read**() и **write**()).
- Блокираност подразумева да нит, која приступа подацима помоћу ових метода, остаје неактивна све док подаци нису доступни.
  - На пример, ако оперативни систем не може одмах да достави садржај датотеке.
- Улаз и излаз у Јава IO су реализовани преко токова података.
- Основу токова чине две апстрактне класе: **InputStream** и **OutputStream**.
  - Улаз и излаз се, у овом случају, организују преко тока бајтова.
  - Поступак је да се креира ток, који ће приликом позива конструктора бити придружен датотеци, конзоли или мрежном порту, а улазно/излазне операције се реализују позивима одговарајућих метода над тако креираним током.
- Скоро сви улазно-излазни методи могу генерисати изузетке, па они обично у декларацији садрже **throws IOException**.

БЛОКИРАЈУЋИ УЛАЗ И ИЗЛАЗ – JAVA.IO (2)

- Поред токова података, за улаз и излаз се још користе читачи и писачи.
  - Они су прилагођени читању карактера, док су **InputStream** и **OutputStream** намењени читању бинарних садржаја.
  - Основу чине две апстрактне класе: **Reader** и **Writer.**



УЛАЗНИ И ИЗЛАЗНИ ТОКОВИ

- Како су **InputStream** и **OutputStream** апстрактне класе, то се улаз/излаз обично реализује преко њихових потокласа, као што су:
  - **FileInputStream** и **FileOutputStream**;
  - **DataInputStream** и **DataOutputStream**
- Раније коришћени токови података **System.in** и **System.out** представљају примерке потокласа **InputStream** и **OutputStream.**
- Приметимо да су ови токови дефинисани као статичка поља у оквиру класе **System** — они представљају унапред припремљене токове за двосмерну комуникацију.
- „Стандардни“ излазни ток **System.out** је већ отворен и спреман за прихватање података. Обично овај ток одговара излазу конзоле.
- „Стандардни” улазни ток **System.in** је већ отворен и спреман за прихват улазних података. Обично овај ток одговара улазу са тастатуре.

УЛАЗНИ ТОК INPUTSTREAM

- Основни метод у класи **InputStream** је метод **read()**.
- Тај метод чита један бајт (број 0-255). Ако се препозна крај тока, метод враћа **-1**.

```
public abstract int read() throws IOExeption;
```

- Методи за читање низа бајтова се реализују вишеструким позивима **read()**.

```
public int read(byte b[]) throws IOException
public int read(byte b[], int pocetak, int duzina) throws IOException
```

- Улазне операције реализоване преко класе **InputStream** су операције тзв. ниског нивоа.
- Рад на том нивоу није атрактиван, нити ефикасан па је развијен велики број потклаcа за организацију улаза „специјалних“ врста података, нпр. ниски, бројева...
- Неке од ових потклаcа су: **FileInputStream**, **FilterInputStream**, **ByteArrayInputStream**, **ObjectInputStream** итд.

ПРИМЕР 1

- Демонстрирати употребу **FileInputStream** за читање података из датотеке чија је путања задата као аргумент командне линије.
- Прочитане бајтове исписати у формату карактера на конзоли.
- Тестирати читање из бинарне датотеке, на пример, слике у **PNG** формату, као и из текстуалне **TXT** датотеке у **ASCII** и **UTF-8** кодним странама.

ПРИМЕР 1 (2)

```
String putanja = args[0];
FileInputStream fin = null;
try {
      fin = new FileInputStream(putanja);
      int i = 0;
      while ((i = fin.read()) != -1)
          System.out.print((char) i);
} catch (FileNotFoundException e) {
      System.err.println(e.getMessage());
} finally {

      fin.close();
}
```

```
java ProcitajDatoteku
"ostalo/tekstASCII.txt"
Ovo je primer ASCII teksta.

java ProcitajDatoteku
"ostalo/tekstUTF8.txt"
ÐÐ²Ð¾ ÑÐµ Ð¿ÑÐ¸Ð¼ÐµÑUTF8 ÑÐµÐºÑÑÐ°.

java ProcitajDatoteku
"ostalo/cmd.png"
PNG
...
óáÿw¼□□ IE
```

ИЗЛАЗНИ ТОК OUTPUTSTREAM

- У класи **OutputStream** најчешће се користи метод **write()**.
- Слично, као и у претходном случају, излазне операције се обично не реализују директним позивима метода **write()**.

```
public abstract void write(int b) throws IOException;
public void write(byte b[]) throws IOException
public void write(byte b[], int offset, int len) throws IOException
```

- Метод **flush()** служи за пражњење излазног бафера:

```
public void flush() throws IOException
```

- Дакле, излаз се обично организује преко потокласа класе **OutputStream** као што су: **FileOutputStream**, **FilterOutputStream**, **ByteArrayOutputStream**, **ObjectOutputStream** итд.

ПРИМЕР 2

- Демонстрирати употребу **DataOutputStream** за писање различитих примитивних типова на стандардни излаз, тј. конзолу.
- Такође, у одвојеној класи, демонстрирати употребу **DataOutputStream** и **FileOutputStream** за писање различитих примитивних типова у бинарну датотеку.
- Потом ту датотеку отворити и прочитати употребом **FileInputStream** и **DataInputStream** (инверзни кораци) .

ПРИМЕР 2 (2)

```
DataOutputStream tok = new
        DataOutputStream(System.out);
try {

      // да форсирамо писање и пре пуњења бафера
      tok.write(65);
      tok.flush();
      tok.writeChars(System.lineSeparator());
      // бинарна репрезентација реалног броја
      // неће имати текстуално смислен запис
      tok.writeDouble(3432.3);
      tok.writeChars(System.lineSeparator());
      tok.writeUTF("Ovo je UTF8 текст са...");
      tok.writeChars(System.lineSeparator());
} catch (IOException e) {
      System.err.println(e.getMessage());
} finally {
      tok.close();
}
```

```
A

@�Й����

Ovo je UTF8 текст са različitim
писмима.
```

ПРИМЕР 2 (3)

- Други део решења, везан за упис и читање у датотеку, погледати у књизи.

ЧИТАЧИ И ПИСАЧИ

- **Reader** и **Writer** класе су доста сличне токовима,
  с тим што уместо са бајтовима, раде са карактерима.
- Како су **Reader** и **Writer** апстрактне класе, то се улаз/излаз реализује преко њихових потокласа, као што су: **InputStreamReader, InputStreamWriter, FileReader, FileWriter.**

ЧИТАЧИ

- Основни метод у класи **Reader** је метод **read()**.
- Тај метод чита један цео број који представља код **Unicode** знака. Ако се при читању препозна крај улаза, метод враћа **-1**.

```
public abstract int read() throws IOExeption;
```

- Поред метода **read()**, у овој класи су дефинисани и методи: **skip()**, **ready()**, **mark()**, **reset()** и **close()**.

ПРИМЕР 3

- За дату ниску, применом **StringReader** класе, приказати све карактере ниске за које важи да се у N наредних карактера не налази ознака за празан простор.
- Поред метода **read()** користити и методе **mark()** и **reset()**.

ПРИМЕР 3 (2)

```
String niska = "Пример неког кратког текста.";
int N = 3;
Reader citac = new StringReader(niska);
try {
    while (true) {
        int c = citac.read();
        if (c == -1)
            break;
        // проверавамо да ли је неки од n наредних празан простор
        citac.mark(N);
        boolean prazan = false;
        for (int i = 0; i < N; i++) {
            char a = (char) citac.read();
            if (a == -1)
                break;
            if (a == ' ') {
                prazan = true;
                break;
            }
        }
        // враћамо се назад на активну позицију
        citac.reset();
        if (!prazan)
            System.out.print((char)c);
    …
```

```
При не крат текста.
```

ПИСАЧИ

- Сви писачи су изведени из апстрактне класе **Writer.**
- Метод **write()**, декларисан у класи **Writer**, преоптерећен је на следећи начин:

```
public abstract void write(byte b) throws IOException;
public void write(char cbuf[]) throws IOException
abstract public void write(char cbuf[], int off, int len) throws IOException
public void write(String str) throws IOException
public void write(String str, int off, int len) throws IOException
```

- Неке од изведених класа су: **BufferedWriter, FilterWriter, OutputStreamWriter, PrintWriter…**

ПРИМЕР 4

- Реализовати програм који задати низ ниски уписује у датотеку чија је путања дата као аргумент командне линије.
- За испис у датотеку користити класу **PrintWriter**.

ПРИМЕР 4 (2)

```
if (args.length != 1) {
    System.err.println("Аргумент командне линије мора садржати путању до датотеке.");
    System.exit(1);
}
String putanja = args[0];

String[] niske = new String[] {
        "Низ од", "неколико ниски", "свака", "записана у", "засебном реду",
        "излазне", "датотеке."
};

PrintWriter pisac = null;
try {
    pisac = new PrintWriter(putanja);
    for(String niska: niske)
        pisac.println(niska);
} catch (FileNotFoundException e) {
    System.err.println(e.getMessage());
}finally {
    if(pisac!=null)
        pisac.close();
}
```

УЛАНЧАВАЊЕ ТОКОВА

- У неким од ранијих примера са токовима података демонстрирана је могућност комбиновања различитих токова.
- У примеру 2 је току **DataOutputStream** прослеђен ток **FileOutputStream** како би се примитивни типови записали у датотеку.
- Овај концепт је заступљен и код читача/писача, и назива се уланчавање токова.
  - Уланчавање функционише тако што се основни ток проследи као аргумент конструктора другом току.
  - Други ток (надток) даље користи услуге основног тока у реализацији својих метода.
  - На овај начин се врши надограђивање функционалности једне класе, без употребе наслеђивања — наслеђивање није ни могуће, јер сви токови већ наслеђују **InputStream**.

ПРИМЕР 5

- Прочитати карактере из задате текстуалне датотеке у **UTF-8** формату.
- У реализацији користити уланчавање класа **FileInputStream** (за приступ току бајтова датотеке), **InputStreamReader** (за конверзију бајтова у карактере) и **BufferedReader** (за побољшање перформанси).

ПРИМЕР 5 (2)

```
 String putanja = args[0];

// ток података који испоручује бајтове (не ради са карактерима)
InputStream fTok = null;
Reader citac = null;
Reader bCitac = null;
try {
    fTok = new FileInputStream(putanja);
    // користи ток бајтова основног тока и производи карактере
    citac = new InputStreamReader(fTok);
    // унапређено читање применом бафера
    bCitac = new BufferedReader(citac);
    int c;
    do {
        c = bCitac.read();
        System.out.print((char) c);
    } while (c != -1);
    System.out.println();
} catch (Exception e) {
    System.err.println(e.getMessage());
} finally {
   …
}
```

РАД СА ДАТОТЕКАМА - FILE

- У претходним секцијама било је речи о начелном раду са токовима бајтова и карактера, без осврта на извор у којем су ти бајтови/карактери похрањени.
- Подаци у спољашњој меморији рачунарског система су обично организовани у виду датотека и директоријума.
- Датотека представља колекцију података који чине једну логичку целину, а директоријуми (фолдери) служе за груписање датотека.
- У програмском језику Јава, у оквиру библиотеке Java IO, за рад са датотекама и директоријумима се користи класа **File**.
- Примерак класе **File** не представља датотеку, већ енкапсулира путању до нечега што може, а не мора бити датотека или директоријум.
  - **File** објекат са путањом до неке датотеке или директоријума не значи да сама та датотека или директоријум постоји.

РАД СА ДАТОТЕКАМА – FILE (2)

- У класи **File** постоји неколико конструктора. Неки од њих имају следећу форму:

```
File dir = new File("C:/Program Files/Java");
File dat = new File(dir, "Primer.java");
File dat = new File("C:/Program Files/Java", "Primer.java");
```

- Приликом рада са **File**, могу се користити и апсолутне и релативне путање.
- Класа **File** садржи преко тридесет метода (погледати у књизи):
  - **getPath()**
  - **getParent()**
  - **getName()**
  - **exists()**
  - **isDirectory()**
  - **listFiles()**
  - **…**

ПРИМЕР 6

- Корисник уноси путању датотеке или директоријума као аргумент командне линије.
- Програм треба да изврши рекурзивни обилазак система датотека у дубину, почев од те путање.
- Притом је потребно исписивати пуну апсолутну путању свих датотека (не и директоријума) на које метод наиђе.

ПРИМЕР 6 (2)

```
private static void ObidjiUDubinu(String putanja) {
      File fAktivni = new File(putanja);
      // ако датотека не постоји, нема смисла ићи даље
      if (!fAktivni.exists())
          return;
      // за датотеке само исписујемо путању и враћамо се
      // пошто нема сигурно даљих потомака
      if (fAktivni.isFile()) {
          System.out.println(fAktivni.getAbsolutePath());
          return;
      }
      // иначе, ако је директоријум
      // онда рекурзивно обилазимо по свим потомцима
      File[] fPotomci = fAktivni.listFiles();
      if (fPotomci != null)
          for (File fp : fPotomci)
              ObidjiUDubinu(fp.getAbsolutePath());
}
```

ПАРСИРАЊЕ ПРИЛИКОМ ЧИТАЊА - SCANNER

- Ова класа не припада хијерархији токова нити читача, већ је директно изведена из класе **Object**.
- Међутим, она имплементира неке од интерфејса карактеристичних за улазне токове и читаче (**Closeable, AutoCloseable**) и додатно интерфејс **Iterator<String>**.
- Такође, приликом креирања, објекти класе **Scanner** кроз аргумент конструктора могу прихватити било који тип који имплементира интерфејс **Readable**, па самим тим и било који читач.
- Поред читача могуће је проследити и објекат класе **File, String, InputStream** итд.
- Због овога се класа **Scanner** врло често користи као алтернатива читачима и улазним токовима или чак као примарно решење у обради улазних података.

ПРИМЕР 7

- Из датотеке “ostalo/studenti.txt” која има следећи садржај:

1009987567890 Марко Петровић 1987 23 9.33

2001967567890 Ана Ковачевић 1967 13 8.43

1009997567890 Марија Мирковић 1997 111 9.36

- учитати студенте у листу објеката класе **Student**, који се описују редом ЈМБГ-ом, именом, презименом, годином рођења, бројем индекса и просечном оценом.
- Потом исписати елементе листе.
- За учитавање података из датотеке користити класу **Scanner** са подешеном кодном страном UTF-8.
- За испис на конзолу користити **PrintWriter** са подешеном кодном страном UTF-8.

ПРИМЕР 7 (2)

```
Scanner skener = null;
try {
    List<Student> studenti = new ArrayList<>();
    skener = new Scanner(new File("ostalo/studenti.txt"), "UTF-8");
    while (skener.hasNext()) {
        String JMBG = skener.next();
        String ime = skener.next();
        String prezime = skener.next();
        int godinaRodjenja = skener.nextInt();
        String indeks = skener.next();
        double prosecnaOcena = skener.nextDouble();
        Student student = new Student(JMBG, ime, prezime,
                godinaRodjenja, indeks, prosecnaOcena);
        studenti.add(student);
    }
    for (Student student : studenti)
        System.out.println(student);

     …
```

ПИТАЊА И ЗАДАЦИ

- Упоредити рад са улазним и излазним подацима Јава IO библиотеке и Јава NIO библиотеке.
- Које су класе изведене из апстрактне класе **InputStream**?
- Примером илустровати употребу неких од њих.
- Које су класе изведене из апстрактне класе **OutputStream**? Које од тих класа имају своје парњаке у хијерархији класа изведених из **InputStream**? Илустровати примером употребу неких од парова.
- Навести и примером илустровати употребу метода класе **Reader**.

ПИТАЊА И ЗАДАЦИ (2)

- Навести и примером илустровати употребу метода класе **Writer.**
- Шта је уланчавање токова, како функционише и зашто се користи? Илустровати примером.
- Написати Јава програм који за путању датотеке или директоријума коју корисник уноси са тастатуре, исписује све информације до којих се може доћи употребом метода класе **File.**
- Објаснити како се класа **Scanner** може користи као алтернатива читачима и улазним токовима. Илустровати примером.

ПИТАЊА И ЗАДАЦИ (3)

- Податке из улазне датотеке треба преписати у излазну датотеку. Појединачна линија улазне датотеке садржи информацију о податку и његовом приоритету. Након учитавања 10 линија улазне датотеке у излазну датотеку се исписује 5 података који имају највиши приоритет. Након тога се учитава следећих 10 линија улазне датотеке, а у излазну се поново преписује 5 података који имају највиши приоритет. Поступак се понавља док се не дође до краја улазне датотеке, након чега се остатак података преписује у излазну датотеку по приоритету.
- У улазној датотеци се налазе информације о студентима. Појединачна линија улазне датотеке садржи информације о једном студенту: број индекса (облика број/година уписа), име, презиме, година студија, начин финансирања и просечна оцена. Написати Јава програм који у излазну датотеку исписује информације о студентима који су уписани на факултет 2020. године, финансирају се из буџета и имају просечну оцену већу од 8.00.