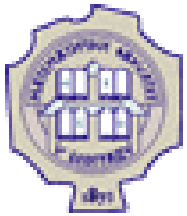
Објектно орјентисано програмирање

Владимир Филиповић
vladaf@matf.bg.ac.rs
Александар Картељ
kartelj@matf.bg.ac.rs

Улаз и излаз, серијализација

Владимир Филиповић
vladaf@matf.bg.ac.rs
Александар Картељ
kartelj@matf.bg.ac.rs

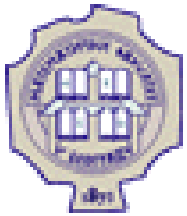
Математички факултет
vladaf@matf.bg.ac.rs



Токови, читачи и писачи

- Улаз и излаз у Јави су (исто као и мрежна и веб комуникација) реализоване преко токова података.
- Основу 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 throw IOException.

Математички факултет
vladaf@matf.bg.ac.rs

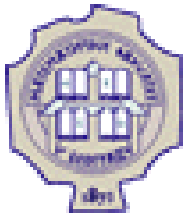


Токови, читачи и писачи (2)

- Поред токова података, за улаз и излаз се још користе читачи и писачи.
- Основу чине две апстрактне класе:
  1. Reader
  2. Writer
- Поступак је да се креира читач/писач, који ће приликом позива конструктора бити придружен датотеци, конзоли или мрежном порту.
- Улазно/излазне операције се реализују позивима одговарајућих метода над креираним читачем/писачем.

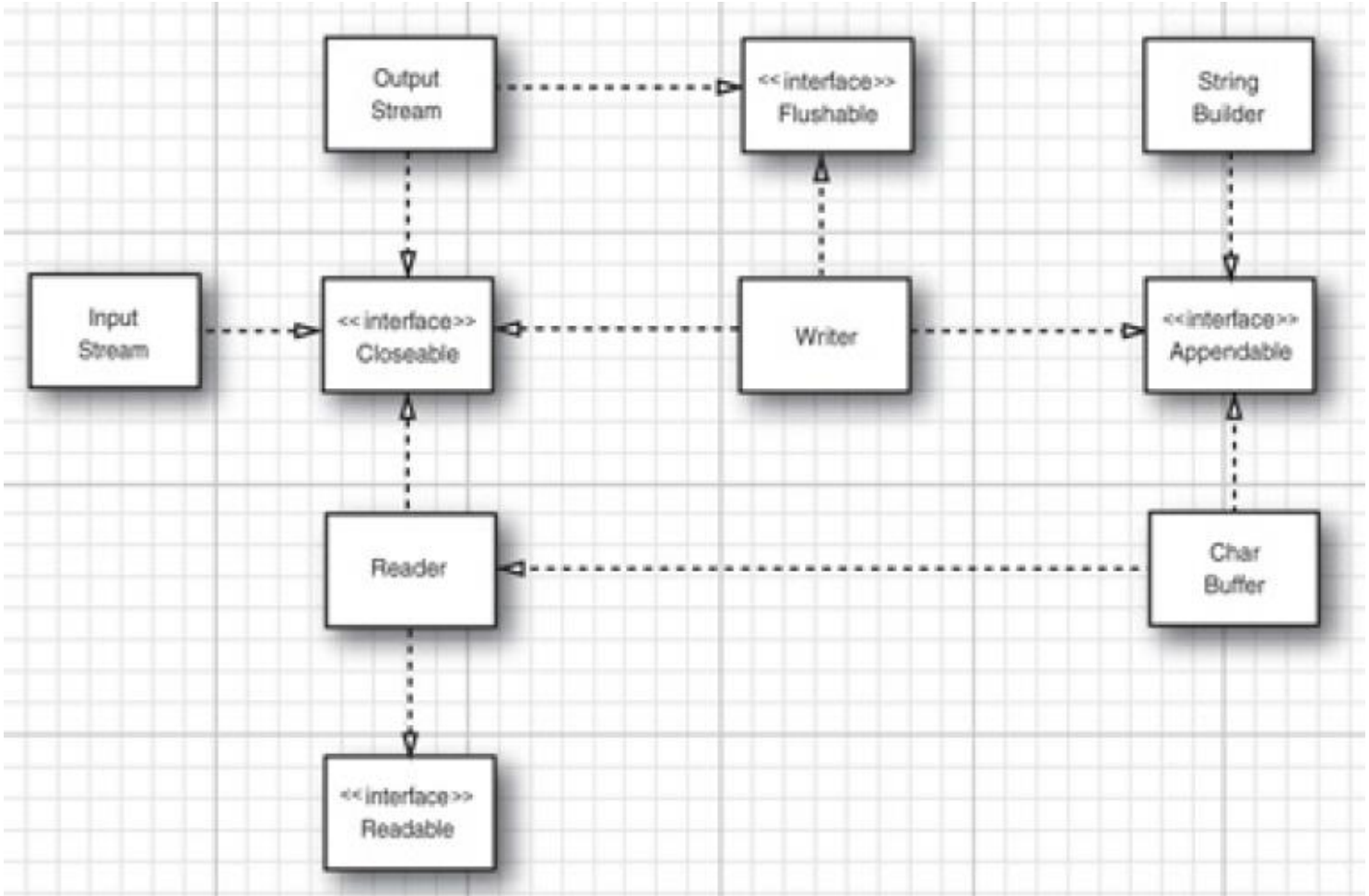
Математички факултет

vladaf@matf.bg.ac.rs



Токови, читачи и писачи (3)

- Дијаграм који следи приказује односе наслеђивања између токова података, читача и писача.



Математички факултет

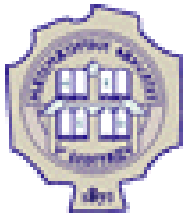
vladaf@matf.bg.ac.rs

Токови, читачи и писачи (4)

- Како су InputStream и OutputStream апстрактне класе, то се улаз/излаз обично реализује преко њихових потклаcа, као што су:
  - FileInputStream, FileOutputStream
  - DataInputStream и DataOutputStream.
- Наравно, често се користе и друге поткласе ових класа чиме се нпр. омогућава филтерисање и/или баферисан улаз/излаз.
- Како су Reader и Writer апстрактне класе, то се улаз/излаз обично реализује преко њихових потклаcа, као што су:
  - InputStreamReader, InputStreamWriter
  - FileReader, FileWriter.

Математички факултет

vladaf@matf.bg.ac.rs

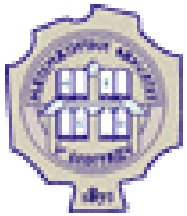


Токови, читачи и писачи (5)

- Токови који су већ коришћени у ранијим програмима су: System.in и System.out, примерци класа InputStream и PrintStream.

- „Стандардни“ излазни ток System.out је већ отворен и спреман за прихватање података.
  - Обично овај ток одговара излазу конзоле или другом одредишту за излаз који је одредило окружење домаћина.

- „Стандардни улазни ток System.in је већ отворен и спреман за прихват улазних података.
  - Обично овај ток одговара улазу са тастатуре или неком другом улазном извору који је одредило Јава окружење домаћина, односно корисник.



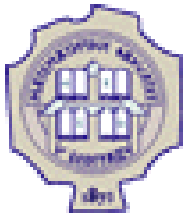
Улазни токови података

- Основни метод у класи InputStream је метод read.
- Тај метод чита један бајт (број 0-255).
  Ако се при читању препозна крај тока, метод враће -1.
  Потпис овог метода је:

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

- Улазне операције обично не реализују директним позивима метода read, већ се користе други објекти, нпр. објекти типа Scanner.
- Методи за читање низа бајтова (ни они се директно не позивају много често):

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

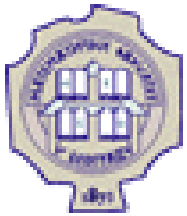


Улазни токови података (2)

- Улазне операције реализоване преко класе InputStream су операције тзв. ниског нивоа.
- Рад на том нивоу није атрактиван ни ефикасан (са тачке гледишта типичног програмера) па је развијен велики број потклаcа за организацију улаза „специјалних“ врста података.
- Дакле, улаз се обично организује преко потклаcа као што су: FileInputStream, FilterInputStream, ByteArrayInputStream, ObjectInputStream итд.

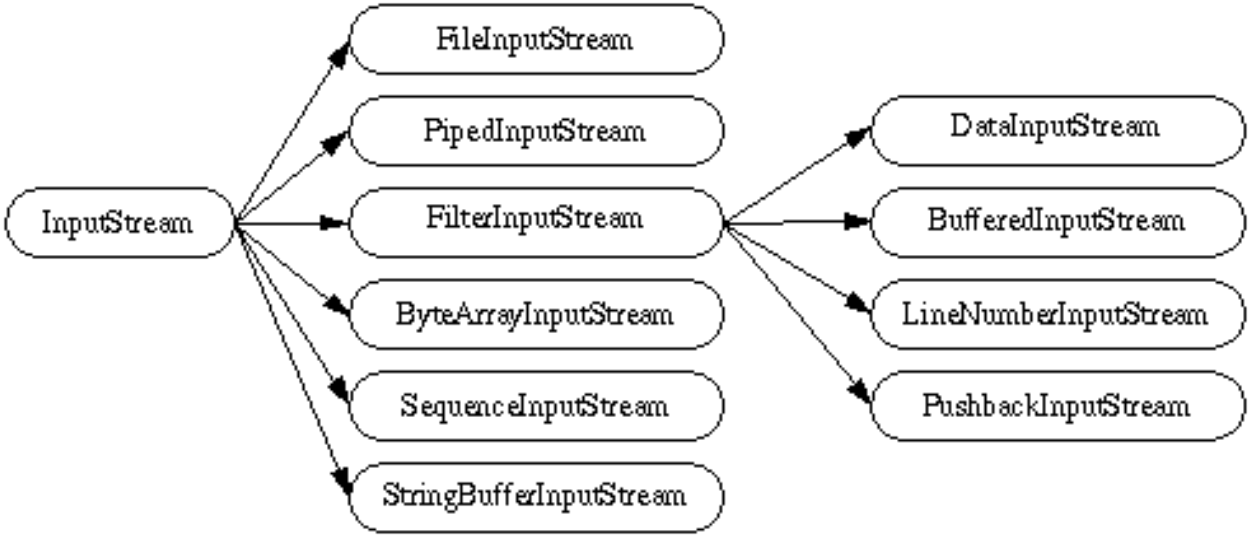
Математички факултет

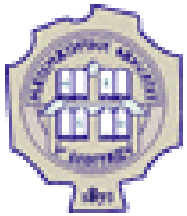
vladaf@matf.bg.ac.rs



Улазни токови података (3)

- Следећи дијаграм приказује потклассе класе InputStream:





Излазни токови података

- 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:

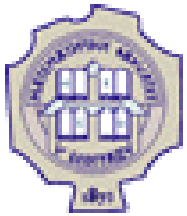
```
public void flush() throws IOException
```

- Метод close затвара излазни ток података, чиме се омогућује боље коришћење ресурса:

```
public void close() throws IOException
```

Математички факултет

vladaf@matf.bg.ac.rs



Излазни токови података (2)

Излазне операције реализоване преко класе OutputStream су операције тзв. ниског нивоа.

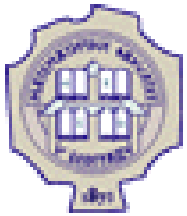
Рад на том нивоу није атрактиван ни ефикасан (са тачке гледишта типичног програмера) па је развијен велики број потклаcа за организацију излаза „специјалних“ врста података.

Дакле, излаз се обично организује преко потклаcа као што су: FileOutputStream, FilterOutputStream, ByteArrayOutputStream, ObjectOutputStream итд.

Наравно, у овим класама се по потреби редефинишу методи класе OutputStream, али се уводе и нови методи.

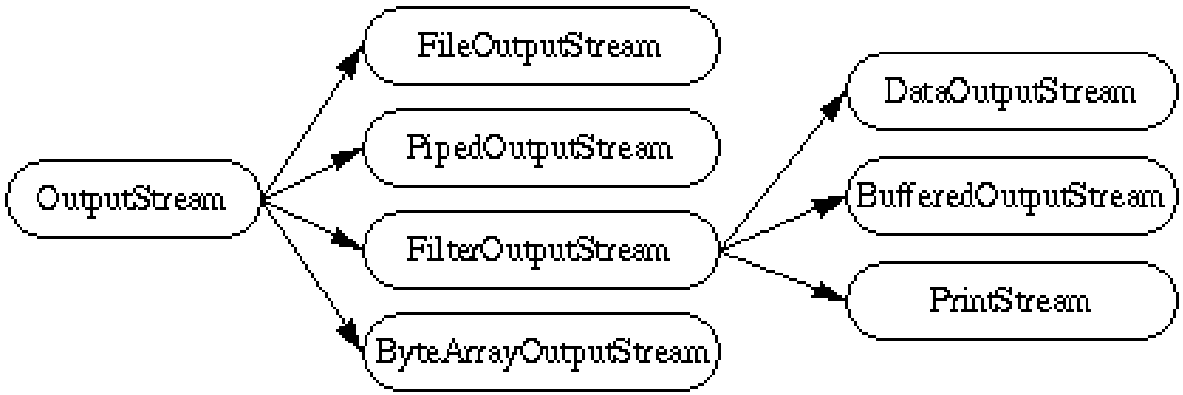
Математички факултет

vladaf@matf.bg.ac.rs

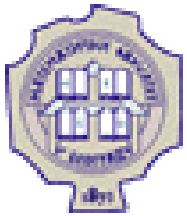


Излазни токови података (3)

- Следећи дијаграм приказује потклассе класе OutputStream:



vladaf@matf.bg.ac.rs

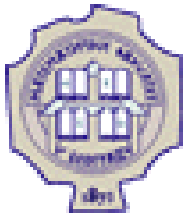


Читачи

- 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 враће -1. Потпис овог метода је:

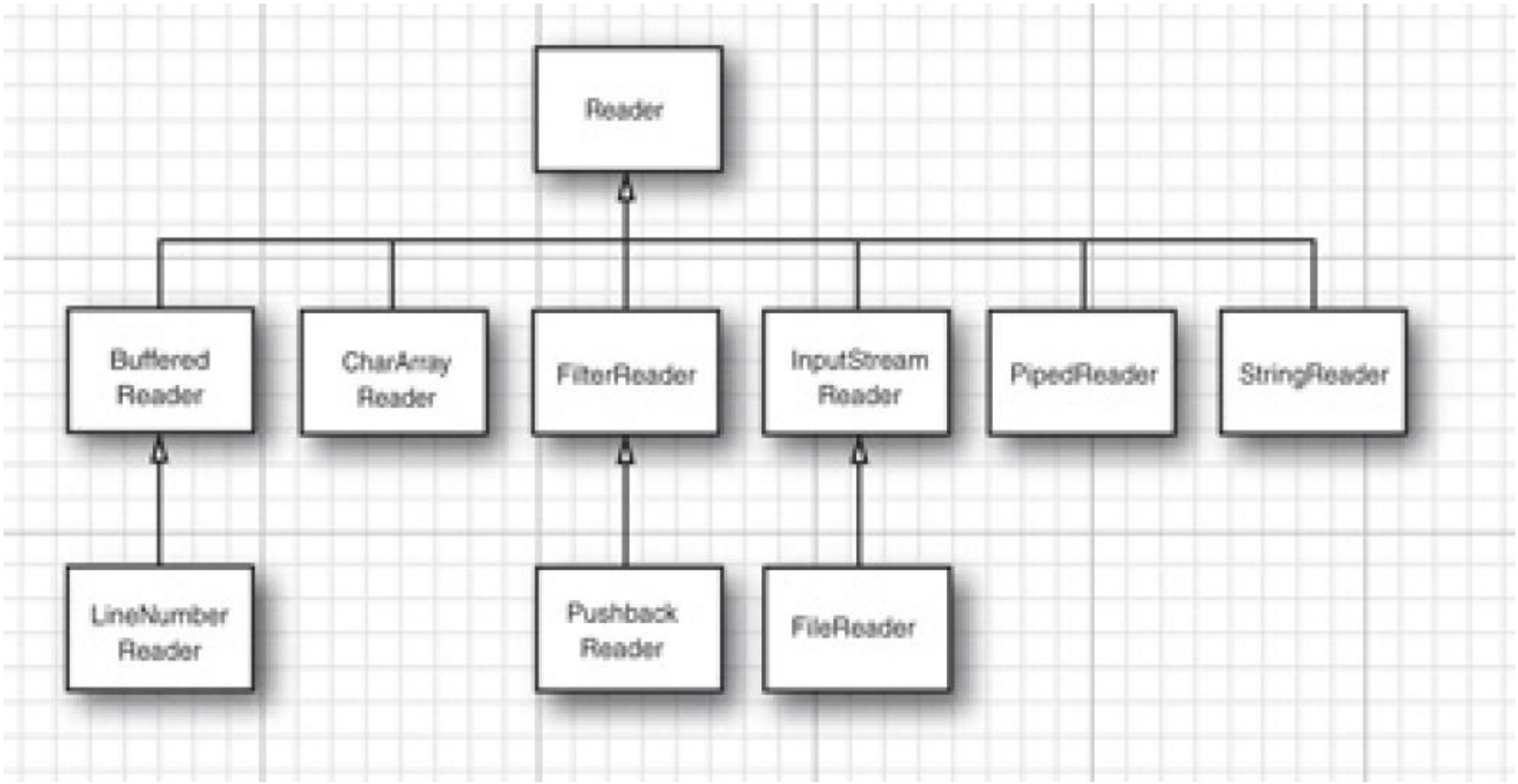
```
public abstract int read() throws IOExeption
```

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



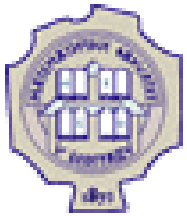
Читачи (2)

- Следећи дијаграм приказује потклассе класе Reader:



Математички факултет

vladaf@matf.bg.ac.rs

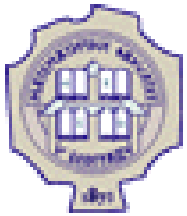


Писачи

- Метод write у класи Writer је такође преоптерећен:

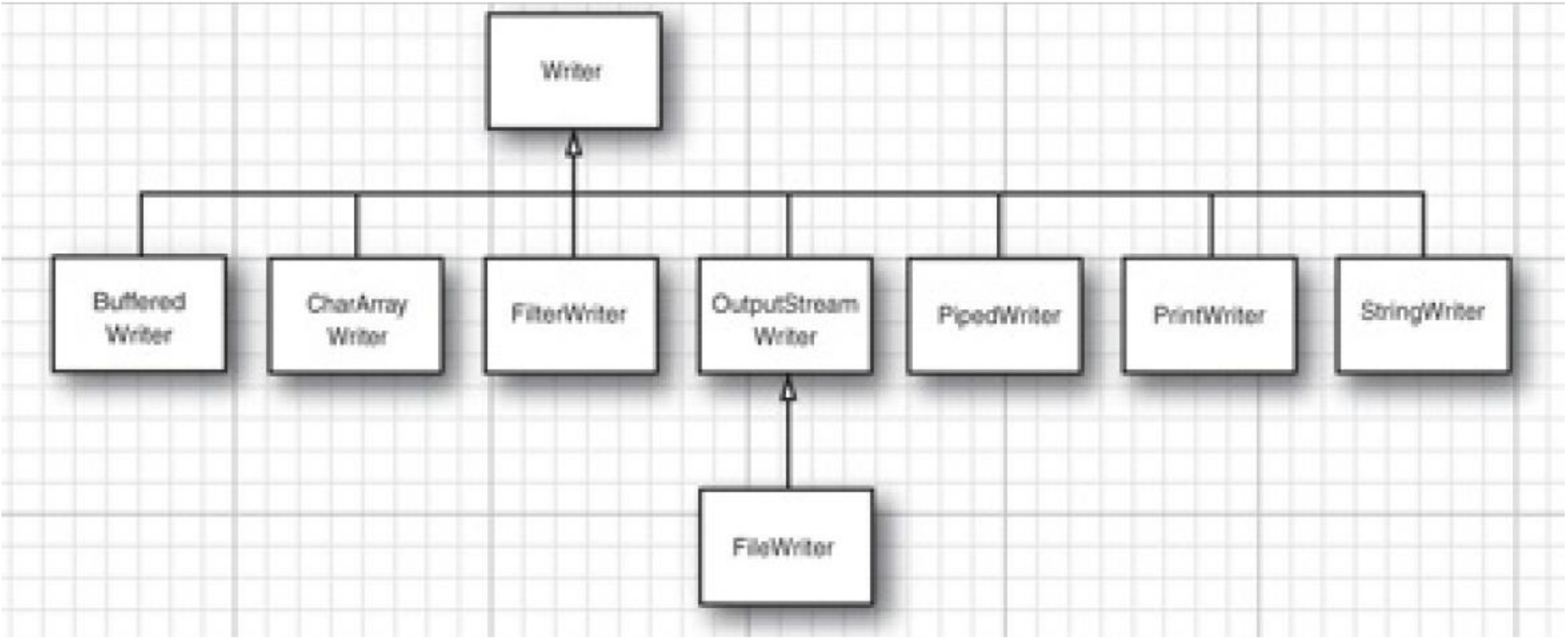
```
public abstract void write(byte b) throws IOException
public void write(char cbuf[]) throws IOException
abstract public void write(char cbuf[], int off, int len) throws IOException
public void write(String str) throws IOException
public void write(String str, int off, int len) throws IOException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

- Ту су још следећи методи:
  - Метод append служи за додавање знака/знакова у писач;
  - Метод flush служи за пражњење излазног бафера писача;
  - Метод close затвара писач, чиме се омогућује боље коришћење ресурса.



Писачи (2)

- Следећи дијаграм приказује потклассе класе Writer:

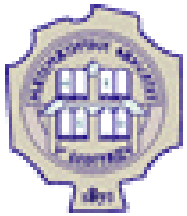


Математички факултет
vladaf@matf.bg.ac.rs

Уланчавање токова

- У Јави се користи паметан механизам за раздвајање две врсте одговорности.
- Наиме, неки токови као што су FileInputStream могу да прибаве бајтове из датотека или из других „егзотичних“ локација.
- Други токови као што су DataInputStream и PrintWriter могу од бајтова да склопе корисније типове података.
- Јава програмер само треба да комбинује ове два функционалности.

vladaf@matf.bg.ac.rs

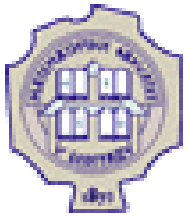


Уланчавање токова (2)

- На пример, да би се могао прочитати реалан број из датотеке, прво треба креирати примерак класе FileInputStream и потом га проследити конструктору класе DataInputStream.

```
FileInputStream fin = new FileInputStream("employee.dat");
DataInputStream din = new DataInputStream(fin);
double s = din.readDouble();
```

Математички факултет
vladaf@matf.bg.ac.rs



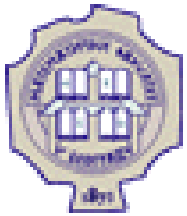
Уланчавање токова (3)

- Програмер може, додатним уланчавањем, тј. угњежђавањем филтера, додати више способности.
- На пример, токови нису баферисани. То значи да читају бајт по бајт.
- Ефикасније је да се захтева блок података и да се они сместе у бафер:

```
DataInputStream din = new DataInputStream(
                          new BufferedInputStream(
                          new FileInputStream("employee.dat")));
```

Математички факултет

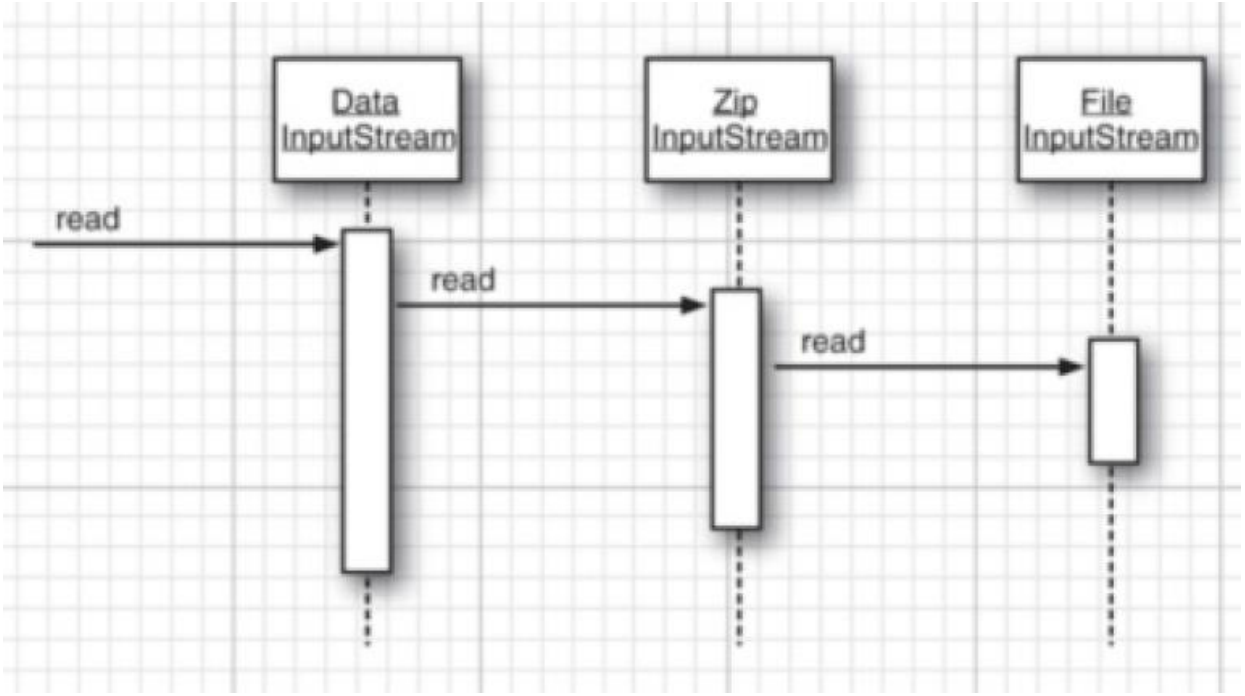
vladaf@matf.bg.ac.rs



Уланчавање токова (4)

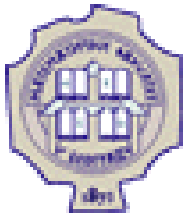
- Читање бројева из компресоване ZIP датотеке може да се постигне на следећи начин:

```
ZipInputStream zin = new ZipInputStream(new
                         FileInputStream("employee.zip"));
DataInputStream din = new DataInputStream(zin);
```



Математички факултет

vladaf@matf.bg.ac.rs

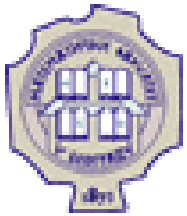


Датотеке и директоријуми

- 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 (каталози) служе за груписање датотека.
- У програмском језику Јава, за рад са датотекама и директоријумима се користи класа File.

Математички факултет

vladaf@matf.bg.ac.rs



Рад са File објектима

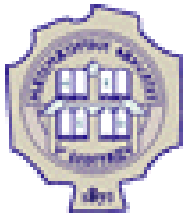
- Примерак класе File заправо не представља датотеку, већ енкапсулира путању до нечега што може, а не мора бити датотека или директоријум.

- File објекат са путањом до неке датотеке или директоријума не значи да сама та датотека или директоријум постоји.

- Често се дефинише File објекат који енкапсулира путању до нове датотеке или новог директоријума који ће тек касније да буде креиран.

Математички факултет

vladaf@matf.bg.ac.rs



Креирање File објеката

- У класи File постоје четири конструктора.
- Први очекује као аргумент стринг са путањом фајла или директоријума:

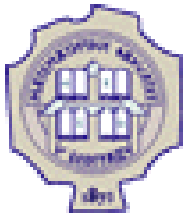
```
File myDir = new File("C:/jdk1.5.0/src/java/io");
File myDir = new File("C:\\jdk1.5.0\\src\\java\\io");
```

- Следећа два конструктора омогућују да се одвојено задају родитељски директоријум и име датотеке:

```
File myDir = new File("C:/jdk1.5.0/src/java/io");
File myFile = new File(myDir, "File.java");
File myFile = new File("C:/jdk1.5.0/src/java/io", "File2.java");
```

Математички факултет

vladaf@matf.bg.ac.rs

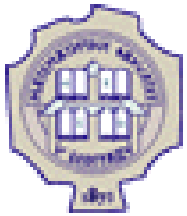


Креирање File објеката (2)

- Последњи, четврти конструктор, омогућује креирање File објекта од објекта типа URI (енг. uniform resource identifier), који енкапсулира униформни идентификатор ресурса на вебу.

```
File remoteFile = new File(
        new URI("http://www.wrox.com/misc-pages/booklist.shtml"));
```

Математички факултет
vladaf@matf.bg.ac.rs

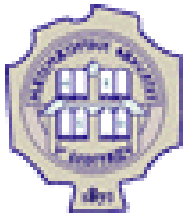


Апсолутне и релативне путање

- Приликом рада са File објектима, могу се користити и апсолутне и релативне путање.
- Пример коришћења релативе путање:

```
File myFile = new File("output.txt");
```

- Путања је релативна у односу на текући директоријум па се датотека "output.txt" налази у директоријуму где је и програм.

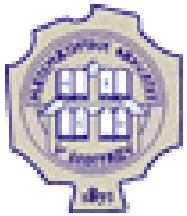


Тестирање File објеката

- Класа File садржи преко тридесет метода.
- Информације о самом File објекту могу се добити следећим методима:

1) Испитивање објеката

- **getName( )**
  враћа име датотеке, не укључујући путању.
  Ако објекат представља директоријум, враћа само име директоријума
- **getPath( )**
  враћа путању, укључујући име датотеке или директоријума
- **getAbsolutePath( )**
  враћа апсолутну путању датотеке, односно директоријума реферисаног текућим File објектом.
- **isAbsolute( )**
  враћа true ако је путања апсолутна, false иначе
- **getParent( )**
  враћа име родитељског директоријума (за датотеку или директоријум).



Тестирање File објеката (2)

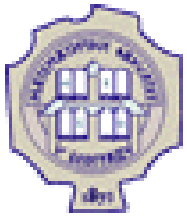
- **getParentFile( )**
  враћа родитељски директоријум као File објекат или null ако не постоји родитељ.
- **toString( )**
  враћа исту ниску исти као getPath( )
- **equals( )**
  метод се користи за поређење два File објекта. Пореде се путање.

2) Испитивање датотека и директоријума

- **exists( )**
- **isDirectory( )**
- **isFile( )**
- **isHidden( )**
- **canRead( )**
- **canWrite( )**
  за овај и претходне методе јасно је из имена када враћају true, односно false.

Математички факултет

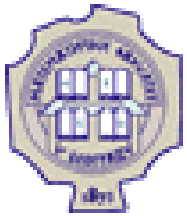
vladaf@matf.bg.ac.rs



Тестирање File објеката (3)

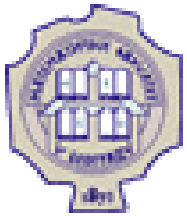
3) Добијање додатних информација о датотеци или директоријуму

- **list()**
  ако текући File објекат представља директоријум, метод враћа низ ниски са именима чланова директоријума иначе враћа null ако је објекат датотека.
- **listFiles()**
  Ако је текући објекат директоријум, враћа низ File објеката који одговарају датотекама и директоријумима у том директоријуму.
- **length()**
  враћа вредност типа long, која је дужина, у бајтовима, датотеке на коју реферише текући File објекат. Ако се ради о датотеци који не постоји, враћена дужина биће 0. Ако објекат реферише на директоријум, није дефинисано шта је повратна вредност метода.



Тестирање File објеката (4)

- **lastModified()**
  враћа вредност типа long, која представља време када је датотека или директоријум реферисан текућим објектом последњи пут измењен. Време је изражено бројем милисекунди протеклих од поноћи 1. јануара 1970. по Гриничу.
- **listRoots()**
  Статички метод класе File враћа низ File објеката, при чему сваки елемент у низу одговара кореном директоријуму текућег система датотека.
  Путања сваке од датотека у систему почиње неким од корених директоријума.
  На Unix систему враћени низ имаће само један елемент који одговара једином кореном директоријуму /.
  Под Windows-ом, низ садржи по елемент за сваки логички уређај који имамо, укључујући floppy, CD, DVD.



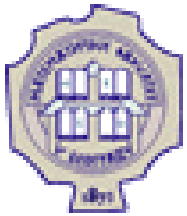
Тестирање File објеката (5)

4) Филтрирање листе

- Постоје и верзије метода list и listFiles класе File које очекују аргумент за филтрирање листе.

Пример.

- Траже се све датотеке/директоријуми чија имена почињу словом T.
- Аргумент који се прослеђује методу list мора бити типа FilenameFilter, док метод listFiles прихвата аргумент типа FilenameFilter или FileFilter.



Тестирање File објеката (6)

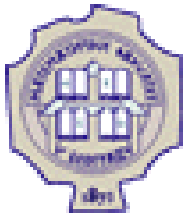
- И FilenameFilter и FileFilter су интерфејси који садрже апстрактни метод accept.

```
public interface FilenameFilter {
        public abstract boolean accept(File directory, String filename);
}

public interface FileFilter {
        public abstract boolean accept(File pathname);
}
```

- Филтрирање листе коју враћају list и listFiles врши се тако што се за сваки елемент листе позива метод accept објекта прослеђеног као аргумент метода list, односно listFiles.
- Ако accept врати true, елемент остаје у листи, а иначе се искључује из ње.

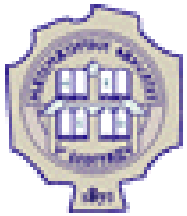
Математички факултет
vladaf@matf.bg.ac.rs



Тестирање File објеката (7)

5) Креирање и модификовање датотека и директоријума

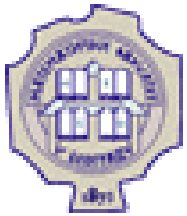
- **renameTo(File path)**
  датотека-директоријум реферисан текућим објектом бива преименован у складу са аргументом.
  Датотека/директоријум на који реферише текући објекат, након овога више не постоји, јер сада има ново име, а можда је и у другом директоријуму.
- **setReadOnly()**
  враћа true ако је операција успела



Тестирање File објеката (8)

- **mkdir()**
  креира директоријум са путањом одређеном текућим објектом. Метод не успева ако родитељски директоријум не постоји. Враћа true ако је операција успела.
- **mkdirs()**
  за разлику од претходног, креира неопходне родитељске директоријуме. Враћа true ако је операција успела. Чак и ако операција не успе, могуће је да су креирани неки од родитељских директоријума.
- **createNewFile()**
  креира нову празну датотеку са путањом задатом текућим објектом, ако такав не постоји. Метод креира датотеку само у постојећем директоријуму (не креира директоријуме одређене путањом). Враћа true ако је успео.
- **createTempFile()**
  статички метод који креира привремени фајл у задатом директоријуму (трећи аргумент), са именом које одређују прва два аргумента метода. При томе, први аргумент одређује почетни део имена фајла, а други његову екстензију.

vladaf@matf.bg.ac.rs



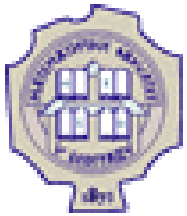
Тестирање File објеката (9)

- **delete()**
  брише датотеку/директоријум представљен текућим објектом и враћа true ако је брисање успело. Не брише директоријуме који нису празни. Да би се обрисао директоријум, најпре се мора обрисати сав његов садржај.
- **deleteOnExit()**
  датотека/директоријум представљен текућим објектом биће избрисан по завршетку рада програма. Метод нема повратну вредност. Брисање ће бити покушано само ако се JVM заврши нормално. Једном када се овај метод позове за File објекат, операција је неповратна.

- Датотеке који се креирају коришћењем метода createTempFile нису нужно привремени јер се не бришу аутоматски.
- Стога се мора користити delete или deleteOnExit како би се уклониле датотеке које више нису потребне.

Математички факултет

vladaf@matf.bg.ac.rs



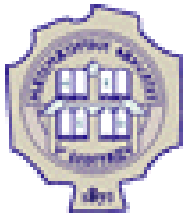
Класа Scanner

- Користи се за парсирање примитивних типова и стрингова.
- Раставља свој улаз на делове (токене) при чему за подразумеване знаке раздвајања сматра белине (' ', '\n', '\t', итд.).
- Примерак класе Scanner може читати из ма ког објекта који имплементира интерфејс Readable.
- Резултујући токени се могу конвертовати у вредности различитих типова користећи разне варијанте next методе.
- **Пример.** Читање наредног целог броја из тока System.in:

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

Математички факултет

vladaf@matf.bg.ac.rs



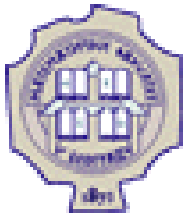
Класа Scanner (2)

- **Пример.**Читање целих бројева типа long из дате датотеке, све док их има:

```
Scanner sc = new Scanner(new File("myNumbers"));
while( sc.hasNextLong() )
        long aLong=sc.nextLong();
```

- **Пример.** Читање из ниске:

```
String input = "1 2 java san";
Scanner s = new Scanner(input);
System.out.println(s.nextInt()); // 1
System.out.println(s.nextInt()); // 2
System.out.println(s.next()); // java
System.out.println(s.next()); // san
```



Класа Scanner (3)

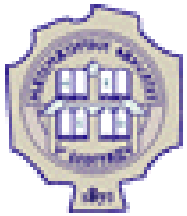
java.util.Scanner

- **useDelimiter()**
  поставља друге знаке као знаке раздвајања при парсирању регуларних израза (уместо белина).
- **next()**
- **nextInt()**
- **next<u>XXX()</u>** (где је <u>XXX</u> име примитивног типа Boolean, Double, Float, ...)
  најпре прескачу знакове раздвајања, а затим парсирају наредни токен и покушавају да врате вредност циљног типа која одговара парсираном токену.
- **hasNext()**
- **hasNextInt()**
- **hasNext<u>XXX()</u>** (где је <u>XXX</u> име примитивног типа Boolean, Double, Float, ...)
  најпре прескачу знакове раздвајања, а затим парсирају наредни токен и покушавају да враћају true уколико парсирани токен представља вредност циљног типа.
  У супротном враће false.

Класа Scanner (4)

java.util.Scanner

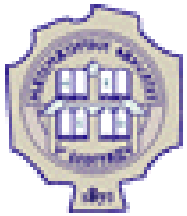
- **public Scanner(InputStream source)**
- **public Scanner(File source)**
- **public Scanner(String source)**
  креира објекат типа Scanner на основу задатог аргумента.
- **public boolean hasNext();**
  враћа true ако постоји наредни токен на улазу.
- **public String next();**
  проналази и враће цео наредни токен. Цео токен је окружен знацима раздвајања (и испред токена и иза њега су знаци за раздвајање).
- **public boolean hasNext(String pattern);**
  враће true ако наредни токен одговара обрасцу pattern.
- **public String next(String pattern);**
  враће наредни токен ако он одговара обрасцу pattern.
- **public boolean hasNextLine();**
  враће true ако постоји следећа линија на улазу.
- **public String nextLine();**
  враће остатак текуће линије искључујући ознаку за крај реда са њеног краја.



Серијализација

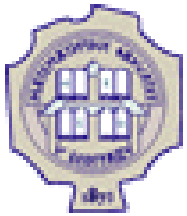
- Серијализација је процес писања објеката у датотеку и читања објеката из датотеке.
- Поступак је веома једноставан. Неопходно је да класа објеката које треба писати у датотеку или читати из датотеке имплементира интерфејс Serializable.
- У већини случајева довољно је само декларисати да класа имплементира овај интерфејс и није неопходан никакав додатни код.

Математички факултет
vladaf@matf.bg.ac.rs



Серијализација (2)

- Уколико постоје чланови тј. поља објекта који су референце на објекте неких других класа, те друге класе такође морају имплементирати интерфејс Serializable и онда ће се њихова серијализација вршити аутоматски.
- Писање објеката у датотеку обавља се позивом метода writeObject за објекат типа ObjectOutputStream.
- При том треба предузети мере да се ухвате изузеци који могу бити избачени.



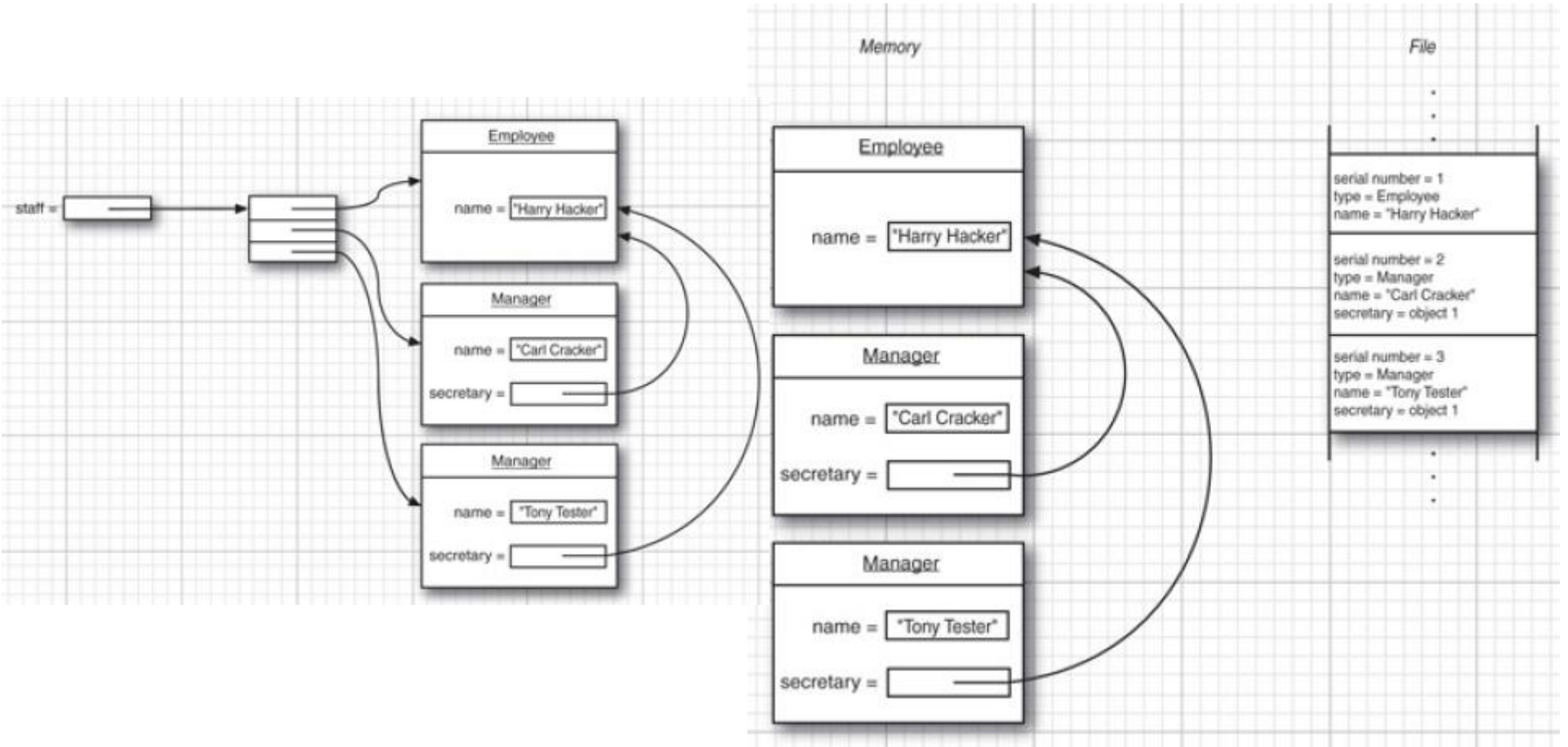
Серијализација (3)

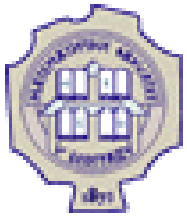
- **Пример.** Фрагмент кода који креира ObjectOutputStream објекат и пише објекат imenik типа HashMap<Osoba,Unos> у датотеку "C:\temp\Imenik.bin":

```
try {
        ObjectOutputStream out = new ObjectOutputStream(
                                 new FileOutputStream(
                                 new File("C:/temp/Imenik.bin")));
        out.writeObject(imenik); out.close();
} catch (IOException e) {
        e.printStackTrace();
        System.exit(1);
}
```

Серијализација (4)

- Следећи дијаграм показује објектни граф (лево) и начин на који ови објекти бивају серијализовани (десно).



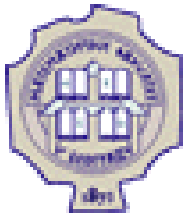


Серијализација (5)

- Читање објеката из датотеке (десеријализација) је подједнако једноставно као и писање у датотеку.
- Креира се ObjectInputStream објекат за читање из жељене датотеке, а потом се објекти из те датотеке читају позивима метода readObject.
- Метод readObject враћа референцу на прочитани објекат као вредност типа Object, па је неопходно извршити експлицитну конверзију (кастовање) у одговарајући тип објекта.
- Низови у Јави такође представљају објекте, па се правило о кастовању односи и на њих.

Математички факултет

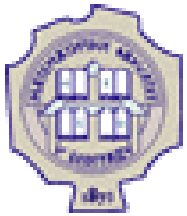
vladaf@matf.bg.ac.rs



Серијализација (6)

Пример. Фрагмент кода који следи креира примерак класе ObjectInputStream и чита објекат типа HashMap<Osoba,Unos> из датотеке "C:\temp\Imenik.bin":

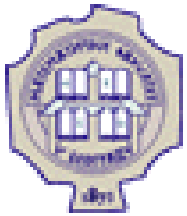
```
try {
        ObjectInputStream in = new ObjectInputStream(
                                new FileInputStream(
                                new File("C:/temp/Imenik.bin")));
        imenik = (HashMap<Osoba, Unos>) in.readObject();
        in.close();
} catch (ClassNotFoundException e) {
        e.printStackTrace();
        System.exit(1);
} catch (IOException e) {
        e.printStackTrace();
        System.exit(1);
}
```



Напредна серијализација

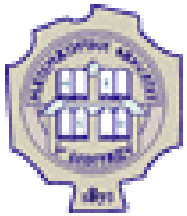
Следећи услови који морају бити испуњени како би се серијализација објеката класе одвијала аутоматски:

1. класа мора бити public;
2. мора имплементирати интерфејс Serializable;
3. ако има атрибуте који су класних типова, ти типови такође морају имплементирати Serializable интерфејс;
4. наткласе морају имплементирати Serializable интерфејс;
5. класе чланова морају имплементирати Serializable интерфејс;
6. у случају да постоји наткласа која не имплементира Serializable интерфејс, она мора имати public подразумевани конструктор, а класа која се серијализује се мора побринути о прослеђивању чланица те наткласе у излазни ток.



Проблеми код серијализације

- За већину класа и примена, серијализација тече праволинијски.
- Међутим, конфузија може настати приликом покушаја писања у датотеку неколико верзија истог објекта.
- Наиме, процес серијализације прати који објекти су уписани у ток и сваки покушај поновног писања објекта не резултује стварним писањем дупликата објекта, већ се у ток уписује само референца која указује на прву појаву објекта у току.



Проблеми код серијализације (2)

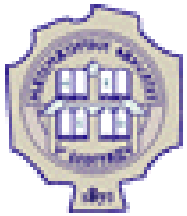
- Компликације настају када се дефиниција класе на неки начин промени између писања и читања објекта.

- Приликом писања објекта у датотеку, уписује се и информација која идентификује класу, тзв. идентификатор верзије, односно вредност статичког поља serialVersionUID.

- Ова информација служи како би се проверило да је дефиниција класе која се користи приликом читања објекта из датотеке компатибилна са оном која је коришћена приликом његовог уписа.

Математички факултет

vladaf@matf.bg.ac.rs



Проблеми код серијализације (3)

- Промене између писања и читања, могу променити идентификатор верзије, због чега операција читања не успева, па долази до избацивања изузетка типа InvalidClassException.

- Промене које најчешће доводе до некомпатибилности су:

1. брисање поља
2. померање класе навише или наниже кроз хијерархију
3. промена поља из нестатичког у статичко
4. промена типа поља

- Компатибилне промене су:

1. додавање поља
2. промена приступних атрибута поља (public, private...)
3. промена поља из статичког у нестатичко

Математички факултет

vladaf@matf.bg.ac.rs

Захвалница

Велики део материјала који је укључен у ову презентацију је преузет из презентације коју је раније (у време када је он држао курс Објектно орјентисано програмирање) направио проф. др Душан Тошић.

Хвала проф. Тошићу што се сагласио са укључивањем тог материјала у садашњу презентацију, као и на помоћи коју ми је пружио током конципцирања и реализације курса.

Надаље, један део материјала је преузет од колегинице Марије Милановић.

Хвала Марији Милановић на помоћи у реализацији ове презентације.