

The background of the slide is a deep blue gradient. On the left side, there is a stylized, semi-transparent globe showing latitude and longitude lines. Several white, wavy, ribbon-like lines swirl around the globe. A bright, horizontal beam of light emanates from the right side of the globe, stretching across the upper half of the slide. The overall aesthetic is high-tech and digital.

# Ulazno-izlazni podsistem

Programski jezici II

# I/O

- Java posjeduje mnoštvo klasa u paketima `java.io` i `java.nio` za rad sa ulazno-izlaznim podsistemom
- tokovi su opšti mehanizam koji Java koristi za rad sa ulazno-izlaznim podsistemom
- tokovi implementiraju sekvencijalni pristup podacima
- postoje dvije vrste tokova: bajt tokovi i karakter tokovi, tj. binarni i tekstualni tokovi, respektivno
- postoje ulazni i izlazni tokovi:
  - ulazni tokovi se posmatraju kao izvorište podataka, odnosno ulazni tokovi služe za čitanje podataka
  - izlazni tokovi se mogu posmatrati kao odredište podataka, odnosno oni služe za upis podataka
- kao ulazni i izlazni tokovi mogu se posmatrati sljedeći entiteti: nizovi bajtova ili karaktera, datoteke, cijevi (eng. *pipe*), konzola ili mrežne konekcije...
- tokovi se mogu ulančavati sa filterima radi obezbjeđivanja dodatnih funkcionalnosti
- mnoštvo klasa u paketima `java.io` i `java.nio` (od verzije 1.4)
- `java.nio`:
  - proširenje
  - brzina

# RandomAccessFile klasa

- služi za osnovnu manipulaciju datotekama
- nije dio InputStream i OutputStream hijerarhija klasa – zasebna klasa
- metode za:
  - čitanje/pisanje (readXxx/writeXxx) primitivnih tipova podataka, uključujući nizove bajtova i stringove
  - readBoolean(), readByte(), readChar(), readDouble(), readInt(), readFloat()...
  - pozicioniranje (seek) – klase iz InputStream i OutputStream hijerarhije nemaju ovakvu funkcionalnost
  - veličina datoteke (length)
  - čitanje sljedeće linije: readLine()

# File klasa

- objekat klase File predstavlja ime datoteke ili imena skupa datoteka u direktorijumu, tj. predstavlja putanju do datoteke ili direktorijuma
- klasa File nije namijenjena za rukovanje sadržajem datoteke, već za rukovanje datotekama i direktorijumima, i to:
  - kreiranje datoteka i direktorijuma,
  - brisanje datoteka i direktorijuma,
  - pristup atributima datoteka i direktorijuma,
  - modifikaciju naziva i atributa datoteka i direktorijuma

# File klasa

- File klasa – od JDK 1.0
- File klasa – bolje ime bi bilo FilePath
- apstraktna putanja sastoji se iz:
  - opcioni, sistem-zavisni prefiks (string), npr. oznaka disk-drive-a, "/" za Unix/Linux root direktorijum ili "\" za Microsoft Windows UNC (*Universal Naming Convention*) putanju
  - niz od 0 ili više naziva (string-ova)
    - prvi naziv u apstraktnoj putanji označava naziv direktorijuma ili hostname (u slučaju Windows UNC), svaki sljedeći naziv označava direktorijum, dok posljednji označava direktorijum ili datoteku
  - primjer – generička forma UNC sintakse za Windows:
    - \\ComputerName\SharedFolder\Resource
- putanja – apsolutna i relativna

# File klasa

- klasa File definiše platformski nezavisne konstante koje se koriste pri definisanju putanja do datoteke ili direktorijuma

```
public static final char separatorChar  
public static final String separator  
public static final char pathSeparatorChar  
public static final String pathSeparator
```

- statički atributi separatorChar i separator definišu karakter ili string koji predstavlja separator direktorijuma i datoteka u putanjama – riječ je o separatorima „/“, „\“ i „:“ za Unix, Windows i Macintosh operative sisteme, respektivno
- statički atributi pathSeparatorChar i pathSeparator definišu karakter ili string koji predstavlja separator naziva datoteka ili direktorijuma u listi putanja – riječ je o separatorima „:“ ili „;“, za Unix i Windows, respektivno



# File klasa

- datoteke i direktorijumi mogu biti referencirani korištenjem apsolutnih ili relativih putanja, pri čemu je obavezno poštovanje konvencija platforme na kojoj se softver izvršava
  - na Unix platformama putanja je apsolutna ako je prvi karakter separator karakter
  - na Windows platformama putanja je apsolutna ako je prvi karakter ASCII znak „\“, ili ako putanja počinje oznakom diska (na primjer, C: ili D:)
  - na Macintosh platformi putanja je apsolutna ako ona počinje nazivom iza kojeg dolazi dvotačka
- preporuka je da se konvencije specifične za platformu izbjegavaju pri razvoju Java programa, a da se koriste pomenuti statički atributi
- u slučaju neodgovarajućih prava i privilegija korisničkog naloga pod kojim se program izvršava, pojedine metode ove klase neće biti uspješno izvršene

# File klasa – kreiranje objekata

- kreiranje objekta klase File ne znači i kreiranje datoteke ili direktorijuma na bazi specificirane putanje
- kreirani objekat predstavlja apstraktnu putanju, na kojoj se može, ali ne mora, nalaziti datoteka ili direktorijum
- ako na ovoj putanji ne postoji datoteka ili direktorijum, moguće ih je kreirati, pozivima odgovarajućih metoda nad kreiranim objektom
- ako na ovoj putanji postoji datoteka ili direktorijum, onda će se njima moći manipulirati pozivima odgovarajućih metoda nad kreiranim objektom
- putanja ne može biti promijenjena nakon što se objekat instancira, jer su instance klase File nepromjenljive



# File klasa – kreiranje objekata

- konstruktori klase File

```
File(String pathname)
```

```
File(String directoryPathname, String fileName)
```

```
File(File directory, String fileName)
```

# File klasa – korisne metode

- metode za dobijanje informacija o putanjama

```
String getName()  
String getPath()  
String getAbsolutePath()  
String getCanonicalPath() throws IOException  
String getParent()  
boolean isAbsolute()  
long lastModified()  
long length()
```

```
public class KlasaFile {  
    public static void main(String[] args) throws IOException {  
        File datoteka = new File("d:\\test\\datoteka.txt");  
        // File datoteka = new File("datoteka.txt");  
        System.out.println(datoteka.getName());  
        System.out.println(datoteka.getPath());  
        System.out.println(datoteka.getAbsolutePath());  
        System.out.println(datoteka.getCanonicalPath());  
        System.out.println(datoteka.getParent());  
        System.out.println(datoteka.isAbsolute());  
        System.out.println(datoteka.lastModified());  
        System.out.println(datoteka.length());  
    }  
}
```

# File klasa – korisne metode

- metode za rad sa privilegijama

```
boolean canWrite()  
boolean canRead()  
boolean canExecute()  
boolean setReadable(boolean readable)  
boolean setReadable(boolean readable, boolean owner)  
boolean setWritable(boolean writable)  
boolean setWritable(boolean writable, boolean owner)  
boolean setExecutable(boolean executable)  
boolean setExecutable(boolean executable, boolean owner)
```

- metode za provjeru postojanja direktorijuma i datoteka

```
boolean exists()  
boolean isFile()  
boolean isDirectory()  
boolean isHidden()
```

# File klasa – korisne metode

- metode za rad sa sadržajem direktorijuma

```
String[] list()
String[] list(FilenameFilter filter)
File[] listFiles()
File[] listFiles(FilenameFilter filter)
File[] listFiles(FileFilter filter)
```

- metode za kreiranje direktorijuma i datoteka

```
boolean createNewFile() throws IOException
boolean mkdir()
boolean mkdirs()
```

- metode za promjenu naziva i brisanje direktorijuma i datoteka

```
boolean renameTo(File dest)
boolean delete()
```

# Tokovi

- Java posjeduje tokove kao opšti mehanizam za rad sa ulazno-izlaznim podsistemom
- ulazno-izlazni podsistem baziran je na četiri apstraktne klase: InputStream, OutputStream, Reader i Writer – ove apstraktne klase definišu osnovne zajedničke funkcionalnosti svih klasa tokova
- klase InputStream i OutputStream su dizajnirane za bajt tokove, dok su klase Reader i Writer dizajnirane za karakter tokove
- ove dvije vrste klasa kreiraju različite hijerarhije
- u opštem slučaju, karakter tokovi se koriste kada je potrebno raditi sa karakterima ili stringovima, dok se bajt tokovi koriste kada se radi sa bajtovima, tj. binarnim podacima

# Bajt tokovi

- apstraktne klase `InputStream` i `OutputStream` su korijenske klase hijerarhije klasa koje vrše čitanje i pisanje bajtova
- njihove klase nasljednice implementiraju različite vrste ulaznih i izlaznih bajt tokova, redefinišući metode klasa `InputStream` i `OutputStream`
  - `BufferedInputStream`, `BufferedOutputStream`
  - `ByteArrayInputStream`, `ByteArrayOutputStream`
  - `DataInputStream`, `DataOutputStream`
  - `FileInputStream`, `FileOutputStream`
  - `FilterInputStream`, `FilterOutputStream`
  - `ObjectInputStream`, `ObjectOutputStream`
  - `PipedInputStream`, `PipedOutputStream`
  - `PrintStream`
  - `PushbackInputStream`
  - `SequenceInputStream`
  - `StringBufferInputStream`
- Postoje i druge klase, poput `javax.sound.sampled.AudioInputStream`



# Klasa InputStream

- `public abstract int read() throws IOException`
- `public int read(byte[] b) throws IOException`
- `public int read(byte[] b, int off, int len) throws IOException`
- `public byte[] readAllBytes() throws IOException` (9)
- `public int readNBytes(byte[] b, int off, int len) throws IOException` (9)
- `public long skip(long n) throws IOException`
- `public int available() throws IOException`
- `public void close() throws IOException`
- `public void mark(int readlimit)`
- `public void reset() throws IOException`
- `public boolean markSupported()`
- `public long transferTo(OutputStream out) throws IOException` (9)
- napomena: metode za čitanje iz tokova su sinhronizovane

# Klasa OutputStream

```
abstract void write(int b) throws IOException // 1
void write(byte[] b) throws IOException // 2
void write(byte[] b, int off, int len) throws IOException // 3
void close() throws IOException // 4
void flush() throws IOException // 5
```

- napomena: metode za upis u tokove su sinhronizovane

# Klasa FileInputStream

- klasa FileInputStream kreira ulazni tok za čitanje bajtova iz datoteke
- primjer ovakve datoteke je slika
- ova klasa se ne koristi za čitanje karaktera iz datoteke

```
FileInputStream(String name) throws FileNotFoundException // 1  
FileInputStream(File file) throws FileNotFoundException // 2  
FileInputStream(FileDescriptor fdObj) // 3
```

# Klasa FileOutputStream

- klasa FileOutputStream kreira izlazni tok za upis bajtova u datoteku
- primjer ovakve datoteke je slika
- ova klasa se ne koristi za upisivanje karaktera u datoteku

```
FileOutputStream(String name) throws FileNotFoundException // 1
FileOutputStream(String name, boolean append) throws
FileNotFoundException // 2
FileOutputStream(File file) throws FileNotFoundException // 3
FileOutputStream(File file, boolean append) throws
FileNotFoundException // 4
FileOutputStream(FileDescriptor fdObj) // 5
```

# Klasa ByteArrayInputStream

- klasa ByteArrayInputStream kreira ulazni tok koji kao izvoriste koristi niz bajtova
- pročitani bajtovi se smještaju u interni bafer
- interni brojač vodi računa o poziciji sljedećeg bajta koji će biti pročitán pozivom read metode

```
ByteArrayInputStream(byte buf[])
```

```
ByteArrayInputStream(byte buf[], int offset, int length)
```

# Klasa ByteArrayOutputStream

- klasa ByteArrayOutputStream kreira izlazni tok koji kao odredište koristi niz bajtova
- ovaj niz bajtova (bafer) automatski raste kako podaci u njega bivaju upisani
- podaci iz bafera mogu biti dobijeni pozivom metoda toByteArray (koja vraća niz bajta) ili toString

```
ByteArrayOutputStream()  
ByteArrayOutputStream(int size)
```



# Filter bajt tokovi

- filter bajt tokovi su tokovi višeg nivoa koji obezbjeđuju dodatne funkcionalnosti tokovima koji su na njih vezani
- filter tok na određeni način manipuliše podacima toka koji je na njega vezan

Klase nasljednice <code>FilterInputStream</code>	Klase nasljednice <code>FilterOutputStream</code>
<code>BufferedInputStream</code>	<code>BufferedOutputStream</code>
<code>CheckedInputStream</code>	<code>CheckedOutputStream</code>
<code>CipherInputStream</code>	<code>CipherOutputStream</code>
<code>DataInputStream</code>	<code>DataOutputStream</code>
<code>DeflaterInputStream</code>	<code>DeflaterOutputStream</code>
<code>DigestInputStream</code>	<code>DigestOutputStream</code>
<code>InflaterInputStream</code>	<code>InflaterOutputStream</code>
<code>LineNumberInputStream</code>	
<code>ProgressMonitorInputStream</code>	
<code>PushbackInputStream</code>	
	<code>PrintStream</code>

- `FilterInputStream` i `FilterOutputStream`, zajedno sa njihovim klasama nasljednicama definišu ulazne i izlazne filter tokove

# Klasa BufferedInputStream

- I/O baferovanje je čest način optimizacije u cilju poboljšanja performansi
- klasa BufferedInputStream omogućava okruživanje bilo kojeg InputStream objekta i na taj način ga pretvara u baferizovani tok

```
BufferedInputStream(InputStream in)
```

```
BufferedInputStream(InputStream in, int size)
```

# Klasa BufferedOutputStream

- klasa BufferedOutputStream slična je bilo kojoj klasi nasljednici OutputStream klase sa izuzetkom dodatne flush metode
- ova metoda se koristi kako bi se podaci koji se nalaze u baferu stvarno upisali u odredište

```
BufferedOutputStream(OutputStream out)  
BufferedOutputStream(OutputStream out, int size)
```

- svrha korištenja ove klase jeste poboljšanje performansi na taj način što će se smanjiti broj stvarnih upisa u odredište

# Klase DataInputStream i DataOutputStream

- klasa DataInputStream implementira interfejs DataInput i na taj način omogućava aplikaciji da čita primitivne Java tipove podataka iz ulaznog toka, na mašinski nezavisan način

```
DataInputStream(InputStream in)  
DataOutputStream(OutputStream out)
```

- klasa DataOutputStream implementira interfejs DataOutput i na taj način omogućava aplikaciji da upisuje primitivne Java tipove podataka (kao binarne reprezentacije) u izlazni tok, nakon čega će ovi podaci moći biti pročitani korištenjem objekta klase DataInputStream

# Kompresija/dekompresija

- podržan rad sa GZIP i ZIP formatima arhiva

Klasa	Namjena
<b>DeflaterInputStream</b>	Output stream implementacija za kompresiju podataka. Koristi DEFLATE algoritam (RFC 1951), kombinacija LZ77 i Huffman-ovog algoritma
<b>CheckedInputStream</b>	<b>GetChecksum( )</b> kreira checksum za bilo koji <b>InputStream</b> . Checksum se može koristiti za provjeru integriteta ulaznih podataka
<b>CheckedOutputStream</b>	<b>GetChecksum( )</b> kreira checksum za bilo koji <b>OutputStream</b> . Checksum se može koristiti za provjeru integriteta izlaznih podataka
<b>ZipOutputStream</b>	<b>DeflaterOutputStream</b> koji kompresuje podatke u ZIP format
<b>GZIPOutputStream</b>	<b>DeflaterOutputStream</b> koji kompresuje podatke u GZIP format
<b>InflaterInputStream</b>	Input stream implementacija za dekompresiju podataka.
<b>ZipInputStream</b>	<b>InflaterInputStream</b> koji dekompresuje podatke iz ZIP formata
<b>GZIPInputStream</b>	<b>InflaterInputStream</b> koji dekompresuje podatke iz GZIP formata

# Karakter tokovi

- kodovanje karaktera predstavlja šemu za reprezentaciju karaktera
- Java programski jezik interno koduje vrijednosti tipa char kao 16-bitne Unicode karaktere, dok platforma na kojoj se aplikacija izvršava može koristiti i drugačiju šemu kodovanja karaktera
- iako klase bajt tokova posjeduju funkcionalnosti neophodne za rad sa bilo kojim tipom ulazno-izlaznih operacija, one ne mogu raditi direktno sa Unicode karakterima
- apstraktne klase Reader i Writer su korijenske klase hijerarhije klasa karakter tokova
- njihove klase nasljednice implementiraju različite vrste ulaznih i izlaznih karakter tokova, redefinišući metode klasa Reader i Writer – ove klase drugačije se nazivaju čitači i pisači
  - BufferedReader, BufferedWriter
  - CharArrayReader, CharArrayWriter
  - FileReader, FileWriter
  - FilterReader, FilterWriter
  - InputStreamReader, OutputStreamWriter
  - LineNumberReader
  - PipedReader, PipedWriter
  - PrintWriter
  - PushbackReader
  - StringReader, StringWriter



# Karakter tokovi

- čitači i pisači se pojavljuju u verziji 1.1
- ispravljaju problem sa tokovima – slabu podršku Unicode rasporedu:
  - tokovi ne prenose dobro Unicode stringove
  - poseban problem predstavljaju različite hardverske platforme (little-endian, big-endian)
- internacionalizacija – 16-bitni Unicode karakteri – tokovi prenose bajtove (8-bit)
- čitači/pisači ne zamjenjuju tokove – oni ih dopunjavaju
- čitači/pisači se koriste kada je potrebno prenijeti Unicode stringove ili karaktere – u ostalim situacijama koriste se tokovi

# Karakter tokovi

- gotovo sve klase tokova imaju odgovarajuće klase čitača/pisača
- čitače/pisače potrebno je koristiti kad god je to moguće – postoje situacije kada je potrebno koristiti tokove

Tokovi	Čitači / pisači
<b>InputStream</b>	<b>Reader</b> adapter: <b>InputStreamReader</b>
<b>OutputStream</b>	<b>Writer</b> adapter: <b>OutputStreamWriter</b>
<b>FileInputStream</b>	<b>FileReader</b>
<b>FileOutputStream</b>	<b>FileWriter</b>

# Karakter tokovi

- gotovo sve klase tokova imaju odgovarajuće klase čitača/pisača

Tokovi	Čitači / pisači
<b>StringBufferInputStream</b> (deprecated)	<b>StringReader</b>
(nema odgovarajuće klase)	<b>StringWriter</b>
<b>ByteArrayInputStream</b>	<b>CharArrayReader</b>
<b>ByteArrayOutputStream</b>	<b>CharArrayWriter</b>
<b>PipedInputStream</b>	<b>PipedReader</b>
<b>PipedOutputStream</b>	<b>PipedWriter</b>

# Karakter tokovi

Filteri – Java 1.0	Java 1.1
<b>FilterInputStream</b>	<b>FilterReader</b>
<b>FilterOutputStream</b>	<b>FilterWriter</b> (apstraktna klasa bez nasljednica)
<b>BufferedInputStream</b>	<b>BufferedReader</b> (posjeduje readLine())
<b>BufferedOutputStream</b>	<b>BufferedWriter</b>
<b>DataInputStream</b>	<b>koristiti DataInputStream</b> (osim kad je potreban readLine() – u tom slučaju koristiti <b>BufferedReader</b> )
<b>PrintStream</b>	<b>PrintWriter</b>
<b>LineNumberInputStream</b> (deprecated)	<b>LineNumberReader</b>
<b>StreamTokenizer</b>	<b>StreamTokenizer</b> (koristiti konstruktor koji kao argument uzima objekat klase <b>Reader</b> )
<b>PushbackInputStream</b>	<b>PushbackReader</b>

# Klasa Reader

- klasa Reader je apstraktna klasa za čitanje karakter tokova

```
int read(CharBuffer target) throws IOException // 1
int read() throws IOException // 2
int read(char[] cbuf) throws IOException // 3
abstract int read(char[] cbuf, int off, int len) throws
IOException // 4
long skip(long n) throws IOException // 5
boolean ready() throws IOException // 6
boolean markSupported() // 7
void mark(int readAheadLimit) throws IOException // 8
void reset() throws IOException // 9
abstract void close() throws IOException // 10
```

# Klasa Writer

- klasa Writer je apstraktna klasa za upis karaktera u tokove

```
public void write(int c) throws IOException // 1
public void write(char[] cbuf) IOException // 2
public abstract void write(char[] cbuf, int off, int
len)IOException // 3
public void write(String str) throws IOException // 4
public void write(String str, int off, int len) throws
IOException // 5
public Writer append(CharSequence csq) throws IOException // 6
public Writer append(CharSequence csq, int start, int end)
throws IOException // 7
public Writer append(char c) throws IOException // 8
public abstract void flush() throws IOException // 9
public abstract void close() throws IOException // 10
```



# Klasa CharArrayReader

- klasa CharArrayReader je implementacija ulaznog toka koji kao izvorište koristi niz karaktera

```
CharArrayReader(char[] buf)
```

```
CharArrayReader(char[] buf, int offset, int length)
```

# Klasa CharArrayWriter

- klasa CharArrayWriter je implementacija izlaznog toka koji kao odredište koristi niz karaktera

```
CharArrayWriter()
```

```
CharArrayWriter(int initialSize)
```

# Klasa BufferedReader

- BufferedReader je čitač koji baferuju karaktere iz čitača kojeg okružuje

```
BufferedReader(Reader in)
```

```
BufferedReader(Reader in, int sz)
```

```
String readLine() throws IOException
```

# Klasa BufferedWriter

- BufferedWriter je pisač koji baferuju karaktere koji se upisuju
- BufferedWriter posjeduje metodu flush koja se koristi za stvarno upisivanje podataka u odredište

```
BufferedWriter(Writer out)  
BufferedWriter(Writer out, int sz)
```

- korištenje BufferedWriter klase poboljšava performanse aplikacije tako što redukuje broj stvarnih upisa podataka u odredište

# Klasa PrintWriter

- klasa PrintWriter je karakter-orientisana verzija klase PrintStream
- ova klasa, u odnosu na Writer klasu, uvodi preklopljene metode print i println
- ova klasa namijenjena je za upis formatirane tekstualne reprezentacije Java primitivnih tipova i objekata u karakter izlazni tok

```
PrintWriter(File file)
PrintWriter(File file, String csn)
PrintWriter(OutputStream out)
PrintWriter(OutputStream out, boolean autoFlush)
PrintWriter(String fileName)
PrintWriter(String fileName, String csn)
PrintWriter(Writer out)
PrintWriter(Writer out, boolean autoFlush)
```

- s ciljem njihovog upisa PrintWriter treba biti vezan na pisače, bajt izlazni tok, objekat klase File ili String

# Klasa PrintWriter

- varijante print i println metode

print metode	println metode
	println()
print(boolean b)	println(boolean b)
print(char c)	println(char c)
print(int i)	println(int i)
print(long l)	println(long l)
print(float f)	println(float f)
print(double d)	println(double d)
print(char[] s)	println(char[] ca)
print(String s)	println(String s)
print(Object obj)	println(Object obj)



# Klasa InputStreamReader

- klasa `InputStreamReader` predstavlja vezu između bajt tokova i karakter tokova – konvertor
- ova klasa koristi se za čitanje bajtova i njihovo dekodovanje u karaktere koristeći specificirani karakter set
- karakter set može biti specificiran navođenjem imena ili kodne oznake, a može se koristiti i podrazumijevani karakter set platforme na kojoj se aplikacija izvršava

```
InputStreamReader(InputStream in)
InputStreamReader(InputStream in, Charset cs)
InputStreamReader(InputStream in, CharsetDecoder dec)
InputStreamReader(InputStream in, String charsetName) throws
UnsupportedEncodingException
```

- radi efikasnije konverzije bajtova u karaktere, obično se unaprijed čita više karaktera iz toka na koji je objekat ove klase vezan
- u cilju postizanja bolje efikasnosti preporuka je da se objekat ove klase okruži objektom klase `BufferedReader`

# Klasa OutputStreamWriter

- klasa OutputStreamWriter predstavlja vezu između karakter tokova i bajt tokova – konvertor
- karakteri koji se upisuju u ovaj tok enkoduju se u bajtove koristeći specificirani karakter set
- karakter set može biti specificiran navođenjem imena ili kodne oznake, a može se koristiti i podrazumijevani karakter set platforme na kojoj se aplikacija izvršava

```
OutputStreamWriter(OutputStream out)
OutputStreamWriter(OutputStream out, Charset cs)
OutputStreamWriter(OutputStream out, CharsetEncoder enc)
OutputStreamWriter(OutputStream out, String charsetName)
throws UnsupportedOperationException
```

- svako upisivanje (poziv write metode) prouzrokuje konverziju nad datim karakterom – bajtovi nastali konverzijom smještaju se u bafer, prije nego što se upišu u vezani izlazni tok – veličina ovog bafera može biti eksplicitno specificirana, ali podrazumijevana vrijednost zadovoljava potrebe u većini slučajeva
- u cilju postizanja bolje efikasnosti preporuka je da se objekat ove klase okruži objektom klase BufferedWriter, kako bi se izbjegle suviše česte konverzije

# Klasa FileReader

- FileReader je čitač koji se koristi za čitanje sadržaja datoteke koja sadrži karaktere
- konstruktori klase FileReader koriste podrazumijevano kodovanje karaktera i podrazumijevanu veličinu bafera

```
FileReader(File file) throws FileNotFoundException  
FileReader(String fileName) throws FileNotFoundException  
FileReader(FileDescriptor fd)
```

- ako je neophodno eksplicitno specificirati ove vrijednosti, potrebno je kreirati objekat klase InputStreamReader ili FileInputStream

# Klasa FileWriter

- FileWriter je pisač koji se koristi za upis karaktera u datoteku
- konstruktori klase FileWriter koriste podrazumijevano kodovanje karaktera i podrazumijevanu veličinu bafera

```
FileWriter(File file) throws IOException // 1
FileWriter(File file, boolean append) throws IOException // 2
FileWriter(FileDescriptor fd) // 3
FileWriter(String fileName) throws IOException // 4
FileWriter(String fileName, boolean append) throws
IOException // 5
```

- ako je neophodno eksplicitno specificirati ove vrijednosti, potrebno je kreirati objekat klase OutputStreamWriter ili FileOutputStream

# Klasa Console

- konzola je jedinstven karakter-bazirani uređaj povezan sa JVM
- da li JVM posjeduje konzolu zavisi od platforme na kojoj se izvršava, kao i od načina na koji je JVM pozvana
  - ako se JVM poziva iz komandne linije, i ako standardni ulaz i izlaz nisu redirektovani, konzolu će predstavljati tastatura i prozor iz kojeg je JVM pozvana
  - ako je JVM pozvana automatski, od npr. pozadinskog procesa, onda ona obično neće imati konzolu
- konzoli se može pristupiti korištenjem instance klase Console – instanca klase Console dobija se pozivom statičke metode console klase System – ako ne postoji konzola koja je vezana za JVM, biće vraćena null vrijednost
- operacije čitanja i pisanja su sinhronizovane tako da se garantuje njihovo atomsko izvršavanje
- metode za čitanje sa konzole vraćaju null kada je dostignut kraj ulaznog toka konzole, npr. u slučajevima pritiska na tastere CTRL-D na Unix ili CTRL-Z na Windows OS

# Klasa Console

- primjer

```
public class ConsoleTest {  
    public static void main(String[] args) {  
        String username, password;  
        Console console = System.console(); // 1  
        if (console == null) {  
            System.err.println("Konzola nije dostupna...");  
            return;  
        }  
        username = console.readLine("Korisnicko ime:"); // 2  
        char pass[] = console.readPassword("Lozinka:"); // 3  
        password = new String(pass);  
        System.out.println("\nUNESI PODACI");  
        System.out.println("Korisnicko ime: " + username);  
        System.out.println("Lozinka: " + password);  
    }  
}
```



# Predefinisani tokovi

- klasa `java.lang.System`
  - sadrži 3 predefinisani stream variable: `in`, `out`, `err` (`public`, `static`, `final`)
- `System.out` – standardni izlazni stream
  - konzola
  - `PrintStream`
- `System.in` – standardni ulazni stream
  - tastatura
  - `InputStream`
- `System.err` – standardni error stream
  - konzola
  - `PrintStream`

# Klase InputStreamReader i OutputStreamWriter

- konvertori - klase za spregu tokova i čitača/pisača:
  - InputStreamReader – adapter sa InputStream-a na Reader
  - OutputStreamWriter – adapter sa OutputStream-a na Writer

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

```
Writer out = new BufferedWriter(new  
    OutputStreamWriter(System.out));
```

# Zatvaranje datoteka

- Poziv `close()` metode
  - tradicionalni pristup
  - prije Java 7
- `try-with-resources`
  - od verzije Java 7
  - automatsko zatvaranje datoteka kada više nisu potrebne (pri izlasku iz try bloka)
  - automatsko zatvaranje resursa se vrši bez obzira na to kako je try blok okončan

```
try (<resursi>) {  
    <naredbe>  
}
```

- kao resursi se mogu pojaviti svi objekti čije klase implementiraju `java.lang.AutoCloseable` interfejs, što uključuje i sve objekte čije klase implementiraju `java.io.Closeable` interfejs (nasljeđuje `AutoCloseable`)
- stream-ovi implementiraju `java.io.Closeable` interfejs

# Serijalizacija

- životni vijek objekta:
  - od kreiranja
  - do uništenja
  - nikako poslije završetka programa
- postoje situacije u kojima je potrebno sačuvati objekat i nakon završetka programa – radi njegovog korištenja pri ponovnom pokretanju programa
- serijalizacija je:
  - proces transformacije objekta u sekvencu bajtova na osnovu kojih se kasnije može izvršiti rekonstrukcija originalnog objekta
  - prevođenje objekta u niz bajtova i njegova rekonstrukcija iz niza bajtova u “živ” objekat
- deserijalizacija – proces kreiranja originalnog objekta na osnovu uskladištene sekvence bajtova
  - deserijalizovani objekat ima stanje koje je objekat imao u trenutku serijalizacije, osim članova koji nisu serijalizovani
- serijalizovan niz bajtova se može snimiti u datoteku ili poslati preko mreže – i jedno i drugo upotrebom tokova

# Serijalizacija

- slanje preko mreže – npr. prevazilaženje razlika u operativnim sistemima (primjer, Linux i Windows)
- procesi serijalizacije i deserijalizacije su dizajnirani tako da rade korektno i u situacijama kada objekat koji se serijalizuje referencira drugi objekat, i obrnuto
- moguće su i kompleksnije strukture gdje postoji čitavo stablo međusobno referenciranih objekata
- u ovakvim situacijama, pokušaj serijalizacije objekta na vrhu stabla dovešće do lociranja drugih objekata i njihove serijalizacije
- u procesu deserijalizacije svi serijalizovani objekti i njihove reference će biti korektno kreirane
- drugi nazivi
  - serijalizacija: *deflating* ili *marshalling*
  - deserijalizacija: drugi nazivi: *inflating* ili *unmarshalling*

# Serijalizacija

- da bi se neki objekat serijalizovao:
  - potrebno je da implementira `java.io.Serializable` interfejs
  - da su atributi i parametri metoda takođe serijalizabilni
- interfejs `java.io.Serializable` nema metode – markerski interfejs
- promjenljive koje se deklarišu kao tranzijentne promjenljive neće biti upisane u tok pri procesu serijalizacije, kao ni statičke promjenljive



# Serijalizacija

- serijalizaciju Java obezbjeđuje putem `ObjectInput` i `ObjectOutput` interfejsa čije implementacije omogućavaju čitanje iz tokova i upis u tokove
- `ObjectInput` i `ObjectOutput` interfejsi nasljeđuju `DataInput` i `DataOutput` interfejse, respektivno
- klase `ObjectInputStream` i `ObjectOutputStream` implementiraju `ObjectInput` i `ObjectOutput` interfejse, respektivno, obezbjeđujući metode za čitanje i upis binarnih reprezentacija objekata, kao i vrijednosti Java primitivnih tipova
- metode za čitanje i upis u ovim klasama mogu baciti izuzetak `IOException`, dok metode za čitanje mogu baciti i izuzetak `EOFException`, u slučaju pokušaja čitanja kada je dostignut kraj toka

# Serijalizacija

- serijalizacija objekta – kreiranje nekog `OutputStream` objekta i okružiti ga `ObjectOutputStream` objektom
- `writeObject()` – serijalizovan i poslat `OutputStream-u`
- deserijalizacija - kreiranje nekog `InputStream` objekta i okružiti ga `ObjectInputStream` objektom
- `readObject()` – referenca na `Object` – potrebno je uraditi downcast u odgovarajući tip
- serijalizacija – pored čuvanja objekata čuva i objekte koje referenciraju, kao i objekte koje referenciraju referencirani objekti – mreža objekata (*web of objects*)
- kontrolisanje serijalizacije

# Klasa ObjectOutputStream

- klasa ObjectOutputStream se koristi za upis objekata u bilo koji tok koji nasljeđuje klasu OutputStream – npr. upis u datoteku ili mrežnu konekciju

```
ObjectOutputStream(OutputStream out) throws IOException // 1  
ObjectOutputStream() throws IOException, SecurityException
```

```
public class OOS {  
    public static void main(String[] args) throws Exception {  
        Kalkulator kalkulator = new Kalkulator(1, 2);  
        FileOutputStream fis = new  
            FileOutputStream("D:\\test\\k.ser");  
        ObjectOutputStream out = new ObjectOutputStream(fis);  
        out.writeObject(kalkulator);           // 1  
        out.close();  
    }  
}
```

# Klasa ObjectInputStream

- klasa ObjectInputStream se koristi za čitanje objekata iz bilo kojeg toka koji nasljeđuje klasu InputStream

```
ObjectInputStream(InputStream in) throws IOException // 1  
ObjectInputStream() throws IOException, SecurityException
```

```
public class OIS {  
    public static void main(String[] args) throws Exception{  
        FileInputStream fis = new  
FileInputStream("D:\\test\\k.ser");  
        ObjectInputStream ois = new ObjectInputStream(fis);  
        Kalkulator kalkulator = (Kalkulator)  
ois.readObject(); //1  
        ois.close();  
        System.out.println(kalkulator.zbir()); // 2  
    }  
}
```

- na primjer, objekat ove klase može da pročita i rekonstruiše objekte (deserijalizacija) koji su prethodno upisani u datoteku ili mrežnu konekciju

# Serijalizacija

- da li je moguće utvrditi tip serijalizovanog objekta na osnovu podataka u datoteci

```
import java.io.*;
public class Player implements Serializable {}
-----
import java.io.*;

public class FreezePlayer {
    public static void main(String[] args) throws Exception {
        ObjectOutput out = new ObjectOutputStream(new
        FileOutputStream("X.file"));
        Player pancev = new Player();
        out.writeObject(pancev);
        out.close();
    }
}
```

# Serijalizacija

- da li je moguće utvrditi tip serijalizovanog objekta na osnovu podataka u datoteci

```
import java.io.*;

public class ThawPlayer {
    public static void main(String[] args) throws Exception {
        ObjectInputStream in = new ObjectInputStream(
            new FileInputStream(new File(".", "X.file")));
        Object mystery = in.readObject();
        System.out.println(mystery.getClass());
        in.close();
    }
}
```

```
-----
> java ThawPlayer
class Player
```



# Serijalizacija

- ključna riječ **transient**

```
import java.io.*;
import java.util.*;

class Logon implements Serializable {
    private Date date = new Date();
    private String username;
    private transient String password;
    Logon(String name, String pwd) {
        username = name;
        password = pwd;
    }
    public String toString() {
        String pwd =
            (password == null) ? "(n/a)" : password;
        return "logon info: \n    " +
            "username: " + username +
            "\n    date: " + date +
            "\n    password: " + pwd;
    }...
}
```

# Serijalizacija

- Ključna riječ **transient**

```
public static void main(String[] args)
throws IOException, ClassNotFoundException {
    Logon a = new Logon("Marko", "mark0passw0rd");
    System.out.println( "logon a = " + a);
    ObjectOutputStream o =
        new ObjectOutputStream(
            new FileOutputStream("Logon.out"));
    o.writeObject(a);
    o.close();
    int seconds = 5;
    long t = System.currentTimeMillis()
        + seconds * 1000;
    while(System.currentTimeMillis() < t)
        ;
    ObjectInputStream in =
        new ObjectInputStream(
            new FileInputStream("Logon.out"));
    System.out.println(
        "Recovering object at " + new Date());
    a = (Logon)in.readObject();
    System.out.println( "logon a = " + a);
}
}
```

# Serijalizacija

- kontrolisanje serijalizacije
- Externalizable interfejs:
  - writeExternal( )
  - readExternal( )
  - automatski se pozivaju nad objektom za vrijeme serijalizacije i deserijalizacije
  - kod deserijalizacije poziva se default-ni konstruktor, a nakon njega readExternal( )

# Serijalizacija

```
import java.io.*;
class Player1 implements Externalizable {
    public Player1() {
        System.out.println("Player1 Constructor");
    }
    public void writeExternal(ObjectOutput out)
        throws IOException {
        System.out.println("Player1.writeExternal");
    }
    public void readExternal(ObjectInput in)
        throws IOException, ClassNotFoundException {
        System.out.println("Player1.readExternal");
    }
}
```

# Serijalizacija

```
import java.io.*;
class Player2 implements Externalizable {
    Player2() {
        System.out.println("Player2 Constructor");
    }
    public void writeExternal(ObjectOutput out)
        throws IOException {
        System.out.println("Player2.writeExternal");
    }
    public void readExternal(ObjectInput in)
        throws IOException, ClassNotFoundException {
        System.out.println("Player2.readExternal");
    }
}
```

# Serijalizacija

```
import java.io.*;
public class Players {
    public static void main(String[] args)
        throws IOException, ClassNotFoundException {
        System.out.println("Constructing objects:");
        Player1 b1 = new Player1();
        Player2 b2 = new Player2();
        ObjectOutputStream o = new ObjectOutputStream( new
            FileOutputStream("Players.out"));
        System.out.println("Saving objects:");
        o.writeObject(b1);
        o.writeObject(b2);
        o.close();
        ObjectInputStream in = new ObjectInputStream( new
            FileInputStream("Players.out"));
        System.out.println("Recovering b1:");
        b1 = (Player1)in.readObject();
        // OOPS! izuzetak!!!
        //! System.out.println("Recovering b2:");
        //! b2 = (Player2)in.readObject();
    }
}
```

java.io.InvalidClassException:  
Player2; no valid constructor



# Serijalizacija

- alternativa za Externalizable:
  - implementacija Serializable interfejsa i
  - dodavanje writeObject( ) i readObject( ) metoda koje će se, ako postoje, koristiti umjesto default-ne serijalizacije

```
private void writeObject(  
    ObjectOutputStream stream) throws  
    IOException;
```

```
private void readObject(  
    ObjectInputStream stream) throws  
    IOException,  
    ClassNotFoundException
```

# Serijalizacija

- ove metode će koristiti objekti klase `ObjectOutputStream` i `ObjectInputStream`, umjesto svojih `writeObject()` i `readObject()` metoda
- ako serijalizabilni objekat posjeduje navedene metode, `ObjectOutputStream` i `ObjectInputStream` će ih pozvati
- iz vlastito implementirane metode `writeObject()` moguće je izvršiti default-nu `writeObject()` akciju pozivom metode `defaultWriteObject()`
- pozivom `defaultReadObject()` metode iz vlastito implementirane `readObject()` metode moguće je izvršiti default-nu `readObject` akciju