

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

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

| Klase nasljednice <code>InputStream</code> | Klase nasljednice <code>OutputStream</code> |
|--|---|
| <code>AudioInputStream</code> | |
| <code>ByteArrayInputStream</code> | <code>ByteArrayOutputStream</code> |
| <code>FileInputStream</code> | <code>FileOutputStream</code> |
| <code>FilterInputStream</code> | <code>FilterOutputStream</code> |
| <code>InputStream</code> | <code>OutputStream</code> |
| <code>ObjectInputStream</code> | <code>ObjectOutputStream</code> |
| <code>PipedInputStream</code> | <code>PipedOutputStream</code> |
| <code>SequenceInputStream</code> | |
| <code>StringBufferInputStream</code> | |

- njihove klase nasljednice implementiraju različite vrste ulaznih i izlaznih bajt tokova, redefinišući metode klase `InputStream` i `OutputStream`

Klasa InputStream

```
abstract int read() throws IOException  
int read(byte[] b) throws IOException  
int read(byte[] b, int off, int len) throws IOException  
int available() throws IOException  
void close() throws IOException  
void mark(int numBytes)  
boolean markSupported()  
void reset() throws IOException  
long skip(long numBytes) throws IOException
```

- 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      // 2
FileNotFoundException                                     // 3
FileOutputStream(File file) throws FileNotFoundException // 4
FileOutputStream(File file, boolean append) throws        // 5
FileNotFoundException                                     // 6
FileOutputStream(FileDescriptor fdObj)                     // 7
```

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

| Klase nasljednice Reader klase | Klase nasljednice Writer klase |
|--------------------------------|--------------------------------|
| BufferedReader | BufferedWriter |
| CharArrayReader | CharArrayWriter |
| FilterReader | FilterWriter |
| InputStreamReader | OutputStreamWriter |
| PipedReader | PipedWriter |
| | PrintWriter |
| StringReader | StringWriter |

- 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

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 |
|-------------------------|---|
| InputStream | Reader adapter: InputStreamReader |
| OutputStream | Writer adapter: OutputStreamWriter |
| FileInputStream | FileReader |
| FileOutputStream | FileWriter |

Karakter tokovi

- gotovo sve klase tokova imaju odgovarajuće klase čitača/pisača

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

Karakter tokovi

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

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

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));
```

Primjer

- korištenje klase `OutputStreamWriter`, zajedno sa klasama `FileOutputStream` i `PrintWriter` u cilju upisa UTF-8 karaktera u datoteku

```
public class UTFWriter {  
    public static void main(String[] args) throws IOException{  
        PrintWriter output = new PrintWriter(  
            new OutputStreamWriter(  
                new FileOutputStream("file.txt"), "UTF-8"));  
        output.println("šđčćžшђчђхšĐČĆŽШЂЧђХ");  
        output.close();  
    }  
}
```

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("\nUNESENI PODACI");  
        System.out.println("Korisnicko ime: " + username);  
        System.out.println("Lozinka: " + password);  
    }  
}
```

Klase iz Java 1.0

- Klase iz Java 1.0 koje nemaju odgovarajuće klase u verziji 1.1, tj. klase koje nisu mijenjane:
 - `DataOutputStream`
 - `File`
 - `RandomAccessFile`
 - `SequenceInputStream`

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()

Tipično korišćenje tokova

```
import java.io.*;

public class IOStreamDemo {
    public static void main(String[] args) throws
        IOException {
        // Čitanje liniju po liniju
        BufferedReader in = new BufferedReader(
            new FileReader("IOStreamDemo.java"));
        String s, s2 = new String();
        while((s = in.readLine()) != null)
            s2 += s + "\n";
        in.close();
        // Čitanje sa standardnog ulaza
        BufferedReader stdin = new BufferedReader(
            new InputStreamReader(System.in));
        System.out.print("Enter a line:");
        System.out.println(stdin.readLine());
    }
}
```

Tipično korišćenje tokova

```
// Čitanje iz memorije
```

```
StringReader in2 = new StringReader(s2);  
int c;  
while((c = in2.read()) != -1)  
    System.out.print((char)c);
```

```
// Čitanje iz memorije - formatirani ulaz
```

```
try {  
    DataInputStream in3 = new DataInputStream(  
        new ByteArrayInputStream(s2.getBytes()));  
    while(true)  
        System.out.print((char)in3.readByte());  
} catch(EOFException e) {  
    System.err.println("End of stream");  
}
```

Tipično korišćenje tokova

```
// 4. Pisanje u datoteku (human-readable)
```

```
in4 = new BufferedReader(  
    new StringReader(s2));  
PrintWriter out1 = new PrintWriter(  
    new BufferedWriter(new  
        FileWriter("IODemo.out")));  
int lineCount = 1;  
while((s = in4.readLine()) != null )  
    out1.println(lineCount++ + ": " + s);  
out1.close();  
} catch(EOFException e) {  
    System.err.println("End of stream");  
}
```


Tipično korištenje tokova

// 5. Smještane i recovery podataka (od strane drugog sistema, nije human-readable)

```
try {
    DataOutputStream out2 = new DataOutputStream(
        new BufferedOutputStream(
            new FileOutputStream("Data.txt")));
    out2.writeDouble(3.14159);
    out2.writeUTF("That was pi");
    out2.writeDouble(1.41413);
    out2.writeUTF("Square root of 2");
    out2.close();
    DataInputStream in5 = new DataInputStream(
        new BufferedInputStream(
            new FileInputStream("Data.txt")));
    // Must use DataInputStream for data:
    System.out.println(in5.readDouble());
    // Only readUTF() will recover the
    // Java-UTF String properly:
    System.out.println(in5.readUTF());
    // Read the following double and String:
    System.out.println(in5.readDouble());
    System.out.println(in5.readUTF());
} catch (EOFException e) {
    throw new RuntimeException(e);
}
```

Tipično korišćenje tokova

```
// 6. Čitanje i pisanje u RAF
```

```
RandomAccessFile rf =  
    new RandomAccessFile("rtest.dat", "rw");  
for(int i = 0; i < 10; i++)  
    rf.writeDouble(i*1.414);  
rf.close();  
rf = new RandomAccessFile("rtest.dat", "rw");  
rf.seek(5*8);  
rf.writeDouble(47.0001);  
rf.close();  
rf = new RandomAccessFile("rtest.dat", "r");  
for(int i = 0; i < 10; i++)  
    System.out.println("Value " + i + ": " +  
        rf.readDouble());  
rf.close();  
}  
}
```

Tipično korišćenje tokova

```
// čitanje/pisanje sa/na konzolu
```

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

```
public class StandardIOReadingWriting {
```

```
    public static void main(String[] args)
```

```
        throws IOException {
```

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

```
            String s;
```

```
            while((s = in.readLine()) != null &&  
                s.length() != 0)
```

```
                System.out.println(s);
```

```
        }
```

```
    }
```

Tipično korišćenje tokova

```
// redirekcija standardnog ulaza i izlaza
import java.io.*;

public class Redirecting {
    public static void main(String[] args) throws IOException {
        PrintStream console = System.out;
        BufferedInputStream in = new BufferedInputStream(
            new FileInputStream("Redirecting.java"));
        PrintStream out = new PrintStream(new BufferedOutputStream(
            new FileOutputStream("test.out")));
        System.setIn(in);
        System.setOut(out);
        System.setErr(out);
        BufferedReader br = new BufferedReader(
            new InputStreamReader(System.in));
        String s;
        while((s = br.readLine()) != null)
            System.out.println(s);
        out.close();
        System.setOut(console);
    }
}
```

Kompresija/dekompresija

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

public class ZipTest {
    public ZipTest() {
        byte[] buffer = new byte [BUFFER_LENGTH];
        try {
            // Kreiramo arhivu "test.zip"
            ZipOutputStream out =
                new ZipOutputStream(
                    new FileOutputStream("test.zip"));
            // Pripremimo se za citanje datoteke koju cemo zapakovati.
            BufferedInputStream fin =
                new BufferedInputStream(
                    new FileInputStream("ZipTest.java"));
            // Ubacivanje datoteke u arhivu pocinje metodom putNextEntry().
            out.putNextEntry(new ZipEntry("ZipTest.java"));
            // Poslije toga moramo da datoteku procitamo i smjestimo u
            arhivu.
            int read;
            while ((read = fin.read(buffer, 0, BUFFER_LENGTH)) != -1) {
                out.write(buffer, 0, read);
            }
            out.close();
        }
    }
}
```

Kompresija/dekompresija

```
// Sada cemo da otvorimo arhivu i procitamo iz nje datoteku "ZipTest.java"
ZipInputStream in = new ZipInputStream(new FileInputStream("test.zip"));
ZipEntry zipEntry;
// Prolazimo kroz sve datoteke u arhivi.
while ((zipEntry = in.getNextEntry()) != null) {
    System.out.println("Extracting file: " + zipEntry.getName());
    int total = 0;
    byte[] accumulator = new byte[MAX_FILE_LENGTH];
    while ((read = in.read(buffer, 0, BUFFER_LENGTH)) != -1) {
        for (int i = 0; i < read; i++)
            accumulator[total+i] = buffer[i];
        total += read;
    }
    // U nizu bajtova "accumulator" nalazi se raspakovana tekuca datoteka.
    String fileText = new String(accumulator, 0, total);
    System.out.println(fileText);
}
in.close();
} catch (Exception ex) {
    ex.printStackTrace();
}

public static void main(String[] args) {
    ZipTest zp = new ZipTest();
}

private static final int BUFFER_LENGTH = 1024;
private static final int MAX_FILE_LENGTH = 65536;
}
```


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 {
        ObjectOutputStream 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

Serijalizacija

- važno ograničenje serijalizacije objekata – isključivo Java rješenje – samo Java programi mogu deserijalizovati takve objekte
- interoperabilnije rješenje – konverzija podataka u XML format
- javax.xml – XML biblioteka u sastavu JDK-a
- XOM biblioteka – open-source – www.xom.nu
- Primjer:
 - Player objekat koji sadrži ime i prezime
 - serijalizacija u XML
 - Person klasa ima getXML() metodu – konvertuje podatke u XML Element objekat
 - Person klasa ima konstruktor koji kao argument ima XML Element – inicijalizacija

Serijalizacija

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

public class Person {
    private String first, last;
    public Person(String first, String last) {
        this.first = first;
        this.last = last;
    }
    public Element getXML() {
        Element person = new Element("person");
        Element firstName = new Element("first");
        firstName.appendChild(first);
        Element lastName = new Element("last");
        lastName.appendChild(last);
        person.appendChild(firstName);
        person.appendChild(lastName) ;
        return person;
    }
}
```

Serijalizacija

```
public Person(Element person) {
    first = person.getFirstChildElement("first").getValue();
    last = person.getFirstChildElement("last").getValue();
}
public String toString() { return first + " " + last; }

public static void format(OutputStream os, Document doc) throws Exception {
    Serializer serializer = new Serializer(os, "ISO-8859-1");
    serializer.setIndent(4);
    serializer.setMaxLength(60);
    serializer.write(doc);
    serializer.flush();
}

public static void main(String [] args) throws Exception {
    List<Person> people = Arrays.asList(
        new Person("Marko", "Markovic"),
        new Person("Petar", "Petrovic"),
        new Person("Mihajlo", "Mihajlovic"));

    System.out.println(people);
    Element root = new Element("people");
    for(Person p : people)
        root.appendChild(p.getXML());
    Document doc = new Document(root);
    format(System.out, doc);
    format(new BufferedOutputStream(new FileOutputStream("People.xml")), doc);
}
}
```


Preferences API

- *Preferences* API – JDK 1.4
- automatski pohranjuje i čita informacije – bliži je perzistenciji nego serijalizaciji objekata
- upotreba je ograničena na:
 - primitivne tipove i
 - String-ove, gdje dužina String-a ne smije biti veća od 8K
- dizajniran da pohrani i čita korisnička podešavanja i konfiguraciona podešavanja samog programa
- *preferences* – skupovi ključ-vrijednost (kao Maps) sačuvani kao hijerarhija čvorova
- iako je moguće kreirati komplikovane strukture, obično se koristi za kreiranje jednog čvora imenovanog imenom klase
- gdje se informacije čuvaju?
 - negdje na file sistemu, zavisno od operativnog sistema
 - za Windows OS - registry

Preferences API

```
import java.util.prefs.*;
import java.util.*;

public class PreferencesDemo {
    public static void main(String[] args) throws Exception {
        Preferences prefs = Preferences
            .userNodeForPackage(PreferencesDemo.class);
        prefs.put("Lokacija", "Banja Luka");
        prefs.put("Institucija", "ETF");
        prefs.putInt("Broj studenata", 2000);
        int usageCount = prefs.getInt("Koristen puta", 0);
        usageCount++;
        prefs.putInt("Koristen puta", usageCount);
        Iterator it = Arrays.asList(prefs.keys()).iterator();
        while(it.hasNext()) {
            String key = it.next().toString();
            System.out.println(key + ": " + prefs.get(key, null));
        }
    }
}
```



File I/O

- `java.nio.file` paket
- `Files`
- `FileStore`
- `FileSystem`
- `FileSystems`
- `LinkPermission`
- `Paths`
- `SimpleFileVisitor`
- `StandardWatchEventKinds`

File I/O

- `java.nio.file` paket
- `Path` klasa
 - predstavlja putanju na fajl sistemu
 - zavisna od platforme
 - na putanji predstavljenoj objektom klase `Path` ne mora postojati fajl ili direktorijum

Path klasa

- `int` `compareTo(Path other)`
- `boolean` `endsWith(Path other)`
- `boolean` `endsWith(String other)`
- `boolean` `equals(Object other)`
- `Path` `getFileName()`
- `FileSystem` `getFileSystem()`
- `Path` `getName(int index)`
- `int` `getNameCount()`
- `Path` `getParent()`
- `Path` `getRoot()`
- `int` `hashCode()`
- `boolean` `isAbsolute()`
- `Iterator<Path>` `iterator()`
- `Path` `normalize()`
- `WatchKey` `register(WatchService watcher, WatchEvent.Kind<?>... events)`
- `WatchKey` `register(WatchService watcher, WatchEvent.Kind<?>[] events, WatchEvent.Modifier... modifiers)`
- `Path` `relativize(Path other)`
- `Path` `resolve(Path other)`
- `Path` `resolve(String other)`
- `Path` `resolveSibling(Path other)`
- `Path` `resolveSibling(String other)`
- `boolean` `startsWith(Path other)`
- `boolean` `startsWith(String other)`
- `Path` `subpath(int beginIndex, int endIndex)`
- `Path` `toAbsolutePath()`
- `File` `toFile()`
- `Path` `toRealPath(LinkOption... options)`
- `String` `toString()`
- `URI` `toUri()`

Files klasa

- Files klasa
 - skup statičkih metoda za
 - čitanje,
 - pisanje i
 - manipulaciju fajlovima i direktorijumima

Files klasa

- static long copy(InputStream in, Path target, CopyOption... options)
- static long copy(Path source, OutputStream out)
- static Path copy(Path source, Path target, CopyOption... options)
- static Path createDirectories(Path dir, FileAttribute<?>... attrs)
- static Path createDirectory(Path dir, FileAttribute<?>... attrs)
- static Path createFile(Path path, FileAttribute<?>... attrs)
- static Path createLink(Path link, Path existing)
- static Path createSymbolicLink(Path link, Path target, FileAttribute<?>... attrs)
- static Path createTempDirectory(Path dir, String prefix, FileAttribute<?>... attrs)
- static Path createTempDirectory(String prefix, FileAttribute<?>... attrs)
- static Path createTempFile(Path dir, String prefix, String suffix, FileAttribute<?>... attrs)
- static Path createTempFile(String prefix, String suffix, FileAttribute<?>... attrs)
- static void delete(Path path)
- static boolean deleteIfExists(Path path)
- static boolean exists(Path path, LinkOption... options)

Files klasa

- static Stream<Path> find(Path start, int maxDepth, BiPredicate<Path, BasicFileAttributes> matcher, FileVisitOption... options)
- static Object getAttribute(Path path, String attribute, LinkOption... options)
- static <V extends FileAttributeView> V getFileAttributeView(Path path, Class<V> type, LinkOption... options)
- static FileStore getFileStore(Path path)
- static FileTime getLastModifiedTime(Path path, LinkOption... options)
- static UserPrincipal getOwner(Path path, LinkOption... options)
- static Set<PosixFilePermission> getPosixFilePermissions(Path path, LinkOption... options)
- static boolean isDirectory(Path path, LinkOption... options)
- static boolean isExecutable(Path path)
- static boolean isHidden(Path path)
- static boolean isReadable(Path path)
- static boolean isRegularFile(Path path, LinkOption... options)
- static boolean isSameFile(Path path, Path path2)
- static boolean isSymbolicLink(Path path)
- static boolean isWritable(Path path)
- static Stream<String> lines(Path path)
- static Stream<String> lines(Path path, Charset cs)
- static Stream<Path> list(Path dir)
- static Path move(Path source, Path target, CopyOption... options)

Files klasa

- static Path move(Path source, Path target, CopyOption... options)
- static BufferedReader newBufferedReader(Path path)
- static BufferedReader newBufferedReader(Path path, Charset cs)
- static BufferedWriter newBufferedWriter(Path path, Charset cs, OpenOption... options)
- static BufferedWriter newBufferedWriter(Path path, OpenOption... options)
- static SeekableByteChannel newByteChannel(Path path, OpenOption... options)
- static SeekableByteChannel newByteChannel(Path path, Set<? extends OpenOption> options, FileAttribute<?>... attrs)
- static DirectoryStream<Path> newDirectoryStream(Path dir)
- static DirectoryStream<Path> newDirectoryStream(Path dir, DirectoryStream.Filter<? super Path> filter)
- static DirectoryStream<Path> newDirectoryStream(Path dir, String glob)
- static InputStream newInputStream(Path path, OpenOption... options)
- static OutputStream newOutputStream(Path path, OpenOption... options)
- static boolean notExists(Path path, LinkOption... options)
- static String probeContentType(Path path)
- static byte[] readAllBytes(Path path)
- static List<String> readAllLines(Path path)
- static List<String> readAllLines(Path path, Charset cs)
- static <A extends BasicFileAttributes> A readAttributes(Path path, Class<A> type, LinkOption... options)
- static Map<String, Object> readAttributes(Path path, String attributes, LinkOption... options)
- static Path readSymbolicLink(Path link)
- static Path setAttribute(Path path, String attribute, Object value, LinkOption... options)
- static Path setLastModifiedTime(Path path, FileTime time)
- static Path setOwner(Path path, UserPrincipal owner)
- static Path setPosixFilePermissions(Path path, Set<PosixFilePermission> perms)
- static long size(Path path)
- static Stream<Path> walk(Path start, FileVisitOption... options)
- static Stream<Path> walk(Path start, int maxDepth, FileVisitOption... options)
- static Path walkFileTree(Path start, FileVisitor<? super Path> visitor)
- static Path walkFileTree(Path start, Set<FileVisitOption> options, int maxDepth, FileVisitor<? super Path> visitor)
- static Path write(Path path, byte[] bytes, OpenOption... options)
- static Path write(Path path, Iterable<? extends CharSequence> lines, Charset cs, OpenOption... options)
- static Path write(Path path, Iterable<? extends CharSequence> lines, OpenOption... options)

NIO

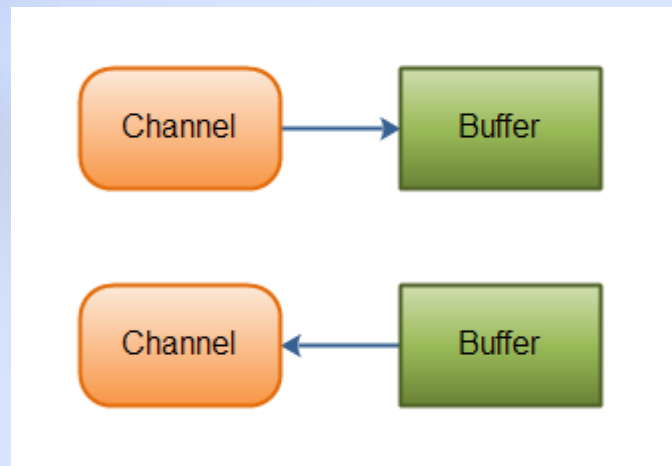
- IO API radi sa bajt i karakter tokovima, a NIO radi sa kanalima (*channels*) i baferima (*buffers*).
 - podaci se uvijek čitaju iz kanala u bafer, a upisuju iz bafera u kanal
- Java NIO omogućava neblokirajuće ulazno-izlazne operacije
 - npr. nit može tražiti od kanala da učitava podatke u bafer
 - dok kanal čita podatke u bafer, nit može raditi nešto drugo – kad su podaci učitani u bafer, nit može nastaviti da ih procesira – slično je i kod upisa podataka u kanal
- Java NIO uključuje i koncept selektora (*selectors*). Selektor je objekat koji može da prati događaje na više kanala (npr. otvaranje konekcije, pristizanje podataka,...)

NIO

- Osnovne Channel implementacije:
 - FileChannel
 - DatagramChannel
 - SocketChannel
 - ServerSocketChannel
 - ove implementacije pokrivaju UDP + TCP network IO, kao i file IO.
- Osnovne Buffer implementacije:
 - ByteBuffer
 - CharBuffer
 - DoubleBuffer
 - FloatBuffer
 - IntBuffer
 - LongBuffer
 - ShortBuffer
 - ove implementacije odnose se na slanje primirivnih tipova podataka putem IO: byte, short, int, long, float, double i char.

NIO

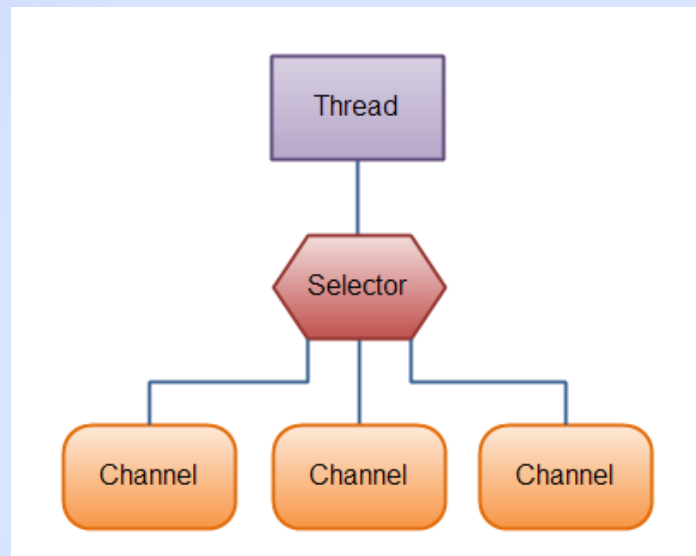
- Java NIO se sastoji od sljedećih osnovnih komponenti (klasa):
 - *Channels*
 - *Buffers*
 - *Selectors*
- Ostale komponente Java NIO (npr. Pipe i FileLock) su uglavnom utility klase koje se koriste zajedno sa osnovne tri klase.



- Tipično, sav ulaz-izlaz u NIO počinje sa kanalom. Iz kanala se podaci čitaju u bafer, dok se iz bafera upisuju u kanal

NIO

- Selector – omogućava da jedna nit upravlja sa više Channel-a
 - pogodno u slučaju kada aplikacija ima više otvorenih konekcija (Channel-a), pri čemu ima malu količinu saobraćaja na svakom kanalu.



NIO Channels

- Java NIO Channel – slični su stream-ovima uz sljedeće razlike:
 - moguće je čitati iz i pisati u Channel, dok su stream-ovi tipično jednosmjerni (ili čitanje ili upis)
 - čitanje i upis u Channel-e moguće je vršiti asinhrono
 - Channel-i uvijek čitaju iz i upisuju u Buffer.
- FileChannel – proizvode ga FileInputStream, FileOutputStream i RandomAccessFile
- Reader i Writer karakter-mod klase ne proizvode kanale, ali klasa `java.nio.channels.Channels` ima metode za proizvodnju Reader-a i Writer-a iz kanala

NIO Buffer

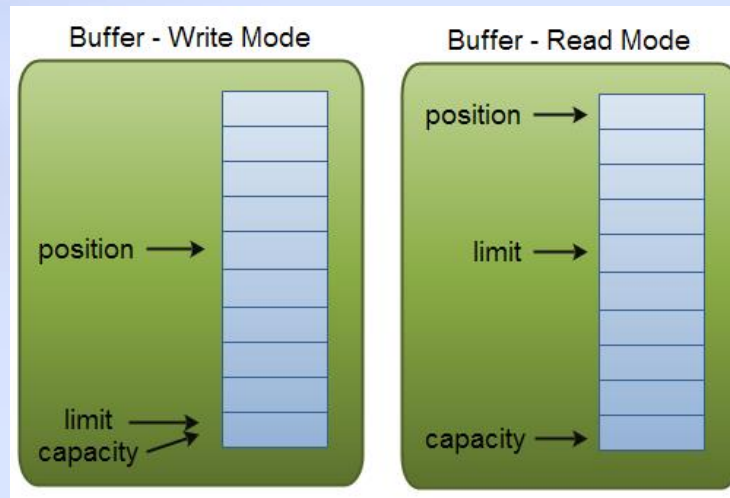
- Java NIO Buffer-i koriste se za interakciju sa NIO Channel-ima (podaci se iz Channel-a čitaju u Buffer-e, i iz Buffer-a se upisuju u Channel-e)
- buffer – memorijski blok u koji se podaci mogu upisivati i iz kojeg se mogu čitati – ovaj memorijski blok je “okružen” NIO Buffer objektom, koji obezbeđuje skup metoda koje omogućavaju lakši rad sa memorijskim blokom
- ByteBuffer – *buffer* koji sadrži *raw* bajtove
- put i get metode za rad sa *raw* bajtovima ili primitivnim tipovima podataka
- ne postoje put i get metode za rad sa objektima, čak ni za rad sa stringovima

NIO Buffer

- rad s Buffer-ima obično obuhvata sljedeće korake:
 - zapisivanje podataka u Buffer
 - poziv `buffer.flip()`
 - prebacuje Buffer iz moda za upis u mod za čitanje, limit se postavlja na tekuću poziciju, a onda se pozicija postavlja na nultu
 - čitanje podataka iz Buffer-a
 - poziv `buffer.clear()` ili `buffer.compact()`
 - pražnjenje Buffer-a, kako bi se učinio spremnim za ponovni upis podataka
 - `clear()` – prazni sve podatke iz buffer-a
 - `compact()` – prazni samo pročitane podatke. Nepročitani podaci se pomjeraju na početak buffer-a, a nakon njih se smještaju novoupisani podaci

NIO Buffer

- Buffer ima 3 važne osobine:
 - capacity
 - veličina/kapacitet buffer-a. U buffer je moguće upisati samo “capacity” podataka tipa byte, long, char, itd. Kad je buffer pun, potrebno ga je isprazniti (pročitati podatke,...), da bi se u njega mogli upisati novi podaci
 - position
 - pozicija kojom se referencira ćelija u buffer-u. Inicijalna pozicija kod upisa podataka je 0. Najveća pozicija je capacity-1
 - flip metoda prebacuje Buffer iz moda za upis u mod za čitanje – pozicija se resetuje na 0.



- limit
 - u modu za upis, limit Buffer-a označava količinu podataka koja može biti upisane u Buffer. U ovom modu limit je jednak capacity-ju Buffer-a.
 - u modu za čitanje, limit označava koliko podataka može biti pročitano iz Buffer-a. Prilikom poziva flip metode, limit se postavlja na mjesto gdje se nalazio position u write modu (a position se postavlja na nultu poziciju)

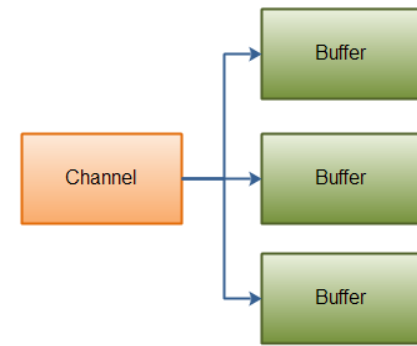
NIO Buffer

- apstraktna klasa Buffer
 - abstract Object array()
 - abstract int arrayOffset()
 - int capacity()
 - Buffer clear()
 - Buffer flip()
 - abstract boolean hasArray()
 - boolean hasRemaining()
 - abstract boolean isDirect()
 - abstract boolean isReadOnly()
 - int limit()
 - Buffer limit(int newLimit)
 - Buffer mark()
 - int position()
 - Buffer position(int newPosition)
 - int remaining()
 - Buffer reset()
 - Buffer rewind()

NIO Buffer

- svaka klasa nasljednica apstraktne klase Buffer ima allocate metodu:
 - primjer – ByteBuffer i IntBuffer
 - static ByteBuffer allocate(int capacity)
 - static IntBuffer allocate(int capacity)

NIO Buffer

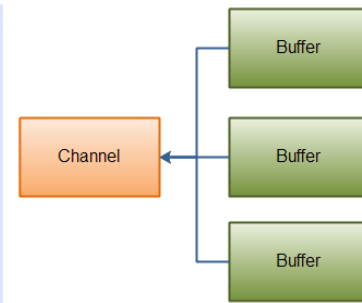


- Scattering Reads
- čitanje podataka iz jednog kanala u više buffer-a

```
ByteBuffer header = ByteBuffer.allocate(128);  
ByteBuffer body = ByteBuffer.allocate(1024);  
ByteBuffer[] bufferArray = { header, body };  
channel.read(bufferArray);
```

- prvo se popunjava jedan buffer – kad se on napuni, podaci se upisuju u sljedeći
- nedostatak: rad sa podacima promjenljive veličine – npr. header + body

NIO Buffer



- Gathering Writes
- upisivanje podataka iz više buffer-a u jedan kanal

```
ByteBuffer header = ByteBuffer.allocate(128);
ByteBuffer body = ByteBuffer.allocate(1024);
ByteBuffer[] bufferArray = { header, body };
channel.write(bufferArray);
```
- podaci se u Channel upisuju iz buffer-a, onim redoslijedom kako se oni i pojavljuju u nizu. Samo podaci između position i limit buffer-a se upisuju. Ako buffer ima kapacitet od 128 bajta, a sadrži samo 58 bajta, onda će u channel biti upisano samo 58 bajta.
- nedostatak: rad sa podacima promjenljive veličine

NIO Channel to Channel transfer

- Java NIO omogućava transfer podataka direktno iz jednog channel-a u drugi, ako je jedan od channel-a `FileChannel`.
- `FileChannel` klasa ima sljedeće metode:
 - `transferTo()`
 - transfer podataka iz `FileChannel`-a u odredišni channel-a
 - `transferFrom()`
 - transfer podataka iz izvorišnog channel-a u `FileChannel`

NIO Channel to Channel transfer

```
RandomAccessFile fromFile = new  
    RandomAccessFile("ulaz.txt", "rw");  
FileChannel    fromChannel = fromFile.getChannel();
```

```
RandomAccessFile toFile = new  
    RandomAccessFile("izlaz.txt", "rw");  
FileChannel    toChannel = toFile.getChannel();
```

```
long position = 0;  
long count    = fromChannel.size();
```

```
toChannel.transferFrom(fromChannel, position, count);
```

NIO Selector

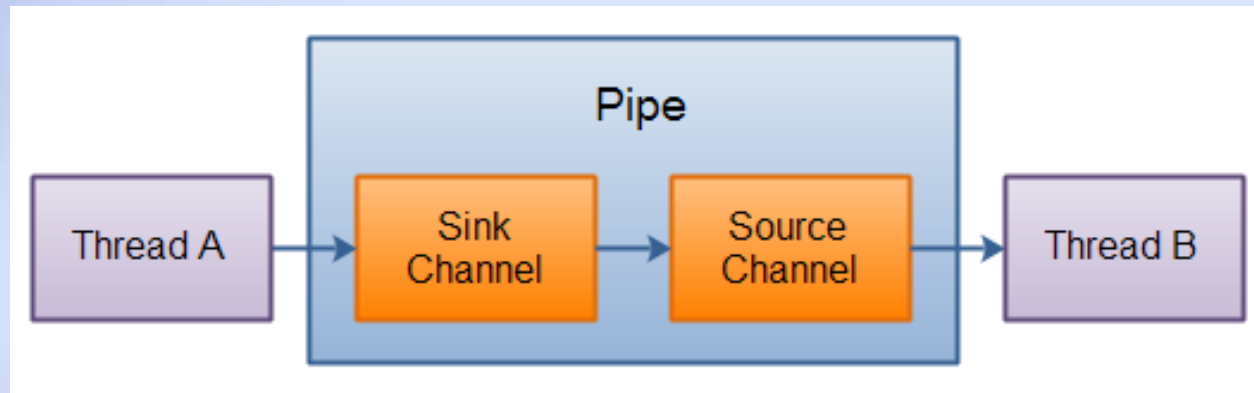
- Selector je Java NIO komponenta koja može ispitati jedan ili više NIO kanala i utvrditi koji kanali su spremni za čitanje ili pisanje – na ovaj način jedna nit može upravljati sa više kanala
- Prednost:
 - korištenje samo jedne niti za upravljanje sa više kanala
 - switch-ing između niti je skupa operacija (novi procesori su sve “bolji” u multitasking-u, tako da ova operacija postaje sve jeftinija)
 - svaka nit zauzima određene memorijske resurse

NIO

- `SocketChannel`
- `ServerSocketChannel`
- `DatagramChannel`

NIO Pipe

- Java NIO Pipe je jednosmjerna konekcija između dvije niti
- Pipe ima SourceChannel i SinkChannel
- podaci se upisuju u SinkChannel
- ovi podaci se onda mogu čitati iz SourceChannel-u



NIO Pipe

- kreiranje Pipe-a
`Pipe pipe = Pipe.open();`
- da bi se moglo upisivati u Pipe, potrebno je imati pristup SinkChannel-u
`Pipe.SinkChannel sinkChannel = pipe.sink();`
- Upis u SinkChannel – metoda `write`
`int write(ByteBuffer src) throws IOException` – metoda interfejsa `WritableByteChannel`
- Čitanje iz Pipe-a – potreban je pristup SourceChannel-u
`Pipe.SourceChannel sourceChannel = pipe.source();`
- Čitanje iz SourceChannel-a – `read` metoda
`ByteBuffer buf = ByteBuffer.allocate(48);`
`int bytesRead = sourceChannel.read(buf);`

NIO

```
public class GetChannel {
    private static final int BSIZE = 1024;
    public static void main(String[] args) throws Exception {
        // Upis u datoteku
        FileChannel fc = new
            FileOutputStream("data.txt").getChannel();
        fc.write(ByteBuffer.wrap("Some text ".getBytes()));
        fc.close();
        // Upis na kraj datoteke
        fc = new RandomAccessFile("data.txt", "rw").getChannel();
        fc.position(fc.size()); // Move to the end
        fc.write(ByteBuffer.wrap("Some more".getBytes()));
        fc.close();
        // Citanje
        fc = new FileInputStream("data.txt").getChannel();
        ByteBuffer buff = ByteBuffer.allocate(BSIZE);
        fc.read(buff);
        buff.flip();
        while (buff.hasRemaining())
            System.out.print((char)buff.get());
    }
}
```

NIO

```
public class ChannelCopy {
    private static final int BSIZE = 1024;
    public static void main(String[] args) throws Exception {
        String from, to;
        if(args.length != 2) {
            from = "data.txt";
            to = "dataCopy.txt";
        }else{
            from = args[0];
            to = args[1];
        }
        FileChannel
            in = new FileInputStream(from).getChannel(),
            out = new FileOutputStream(to).getChannel();
        ByteBuffer buffer = ByteBuffer.allocate(BSIZE);
        while(in.read(buffer) != -1) {
            buffer.flip(); // pripremi za pisanje
            out.write(buffer);
            buffer.clear(); // pripremi za citanje
        }
    }
}
```


java.io vs java.nio

- **IO**
 - Stream oriented
 - Java IO stream oriented – čitanje većeg broja bajta istovremeno iz stream-a, oni se ne keširaju, nema kretanja unaprijed, pa unazad kroz stream (osim ako se podaci ne keširaju u buffer)
 - Java NIO buffer oriented – podaci se čitaju u buffer iz kojeg se kasnije procesiraju, moguće je kretanje unaprijed/unazad. Treba voditi računa o tome da su svi potrebni podaci u buffer-u, kao i da neki podaci nisu “pregaženi” prilikom upisa.
- **NIO**
 - Buffer oriented
- Blocking IO
 - različiti IO stream-ovi su blokirajući – kada nit pozove read ili write metodu, onda je ona blokirana dok podaci ne budu pročitani/upisani.
 - Java NIO neblokirajući mod omogućava niti da zahtjeva čitanje podataka iz kanala i da dobije ono što je trenutno raspoloživo (ili ništa). Nit ne ostaje blokirana čekajući da podaci postanu dostupni i može da “radi” nešto drugo. Slično je i za upis podataka. Nit može zahtijevati da podaci budu upisani u kanal, ali ne mora čekati da oni i budu upisani u potpunosti. Idle time može biti iskorišten od strane niti da izvrši IO operacije na drugim kanalima, tj. jedna nit može upravljati višestrukim kanalima.
- Non blocking IO