Balíček java.io

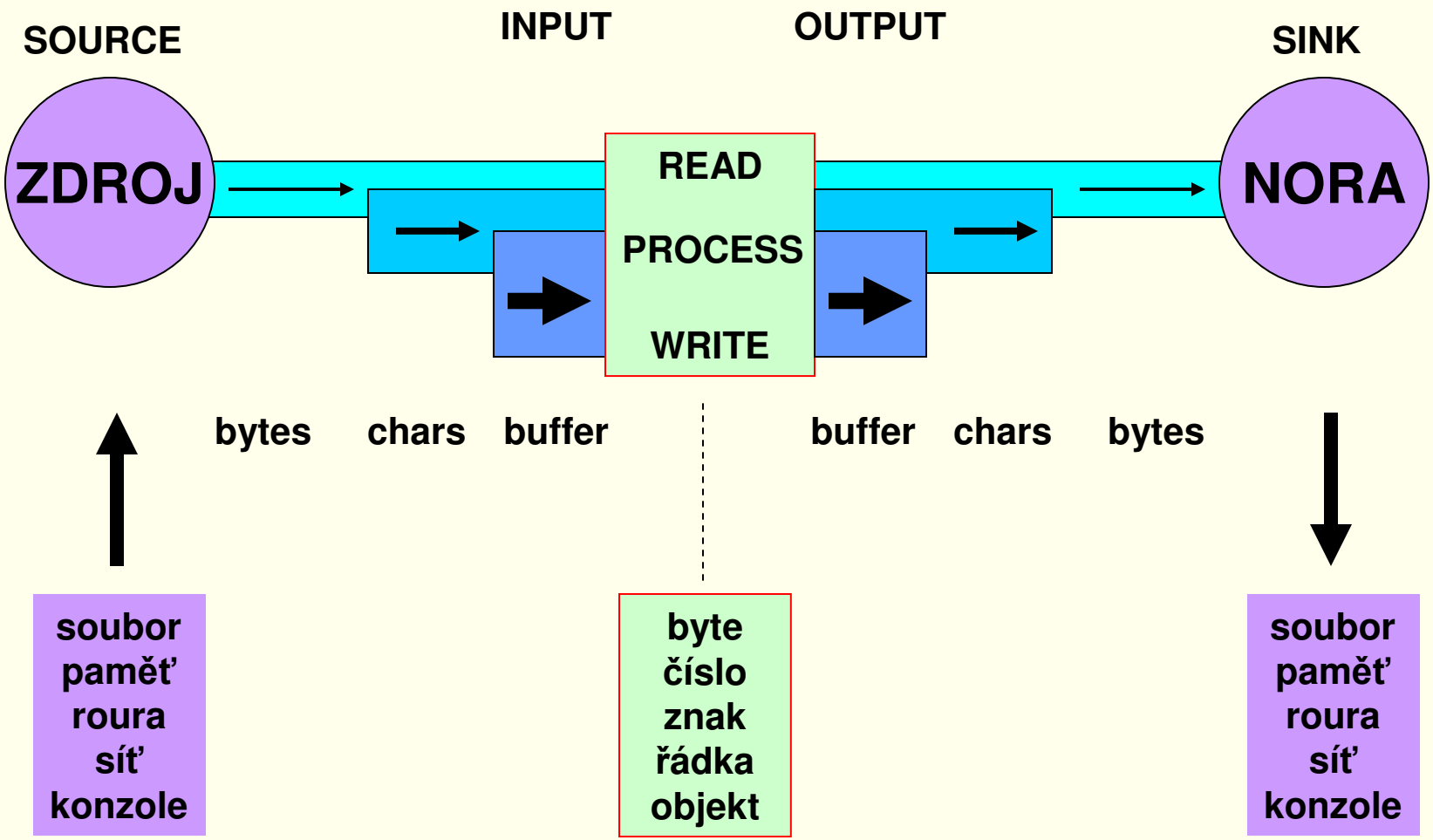
je rozsáhlý, obecně koncipovaný, slouží ke vstupu, výstupu a přenosu dat.

Jeho hlavní součástí jsou třídy realizující tzv. proudy bytů resp. znaků (byte resp. character streams). Tyto třídy jsou potomky čtyř abstraktních tříd: InputStream, OutputStream resp. Reader, Writer.
Proudy se otvírají automaticky – neexistují metody "open".
Proudy se však automaticky neuzavírají – je třeba volat metodu close( ).

V návaznosti s dalšími balíčky java.io podporuje:

- **java.net - přenos po síti**
- **java.util.zip - standardní komprese dat**
- **java.util.jar - standardní komprese dat - obvykle s manifestem**
- **java.nio - inovované I/O umožňuje jemněji pracovat na úrovni kanálů.**

Proudění dat



Třída File

Objekt typu File představuje soubor resp. adresář – nikoli vlastní data – na lokálním disku. Umožňuje přístup, informace, manipulace a cestu absolutní či relativní k pracovnímu adresáři (začíná-li lomítkem či nikoli). Konstruktory jen určují jen jméno, cestu případně i formou URI.

Některé důležitější metody:

- **boolean createNewFile( ) - vytvoří nový skutečný soubor pokud neexistuje**
- **boolean mkdir( ) - vytvoří skutečný adresář**
- **String getAbsolutePath( ) - zjistí absolutní cestu**
- **boolean isDirectory( ), boolean isFile( ) - zjistí typ, neexistuje-li pak false**
- **String[ ] list( ) - vytvoří seznam souborů a podadresářů v adresáři**
- **boolean exists( ) - zjistí existenci**
- **boolean canRead( ), boolean canWrite( ) - zjistí možnosti práce**
- **void delete( ) - zruší soubor nebo adresář**
- **long length( ) - zjistí rozsah souboru**
- **boolean renameTo (File name) - přejmenuje soubor resp. adresář**
- **long lastModified( ) - zjistí čas poslední modifikace**

Třída File

Vytvoření adresáře a souboru, zápis dat a přejmenování :

```
File dir1 = new File( "C:\\K\\L" );
dir1.mkdirs( );
File file1 = new File(  dir, "X.txt" );
file1.createNewFile( );

PrintWriter pw = new PrintWriter( file1 );
pw.println( "blablabla" );
pw.close( );

File dir2 = new File( "C:\\K\\M" );
dir1.renameTo( dir2 );
File file2 = new File(  dir2, "X.txt" );
File file3 = new File(  dir2, "Y.txt" );
file2.renameTo( file3 );
```

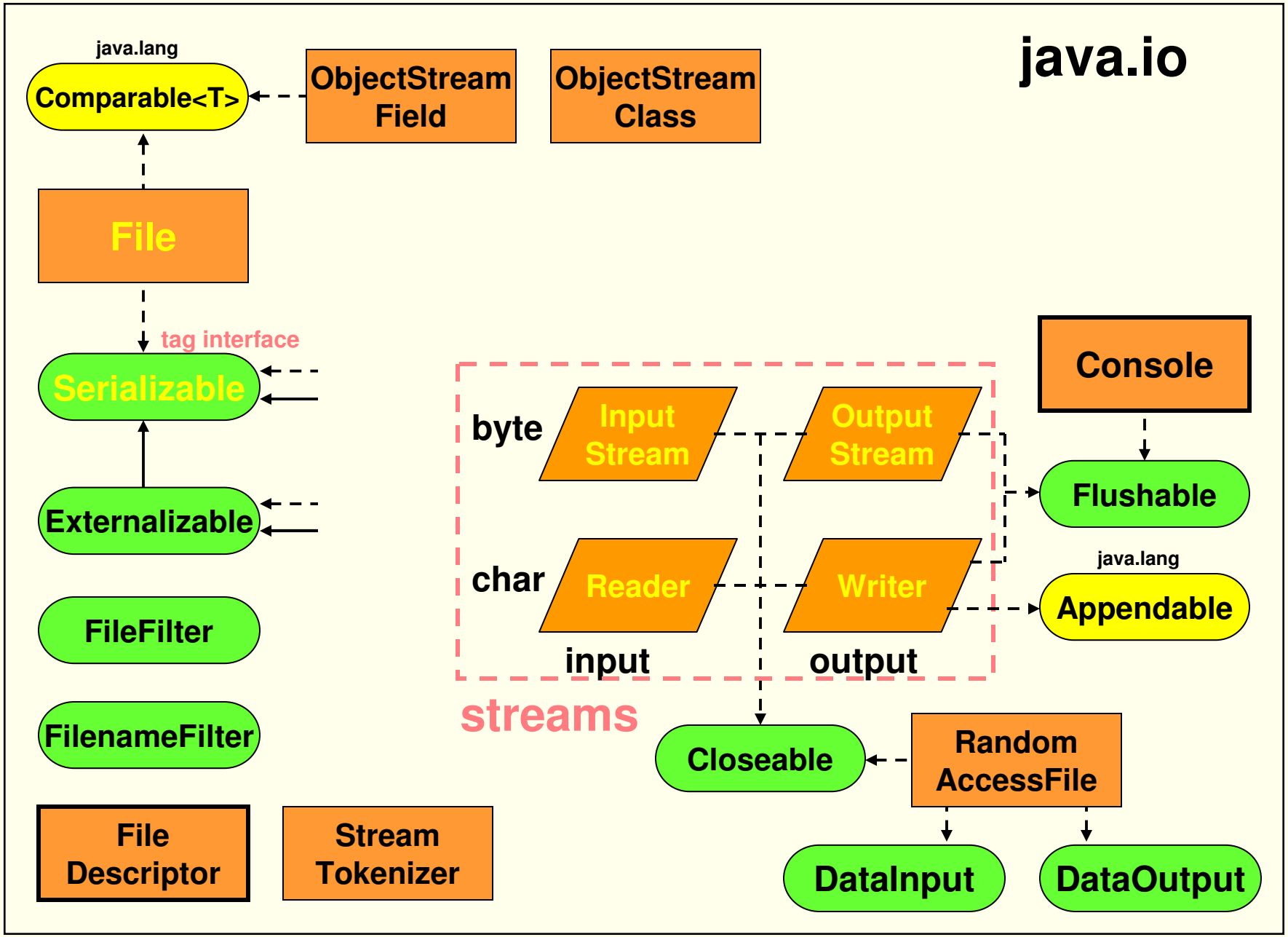
Rekurzivní výpis adresáře

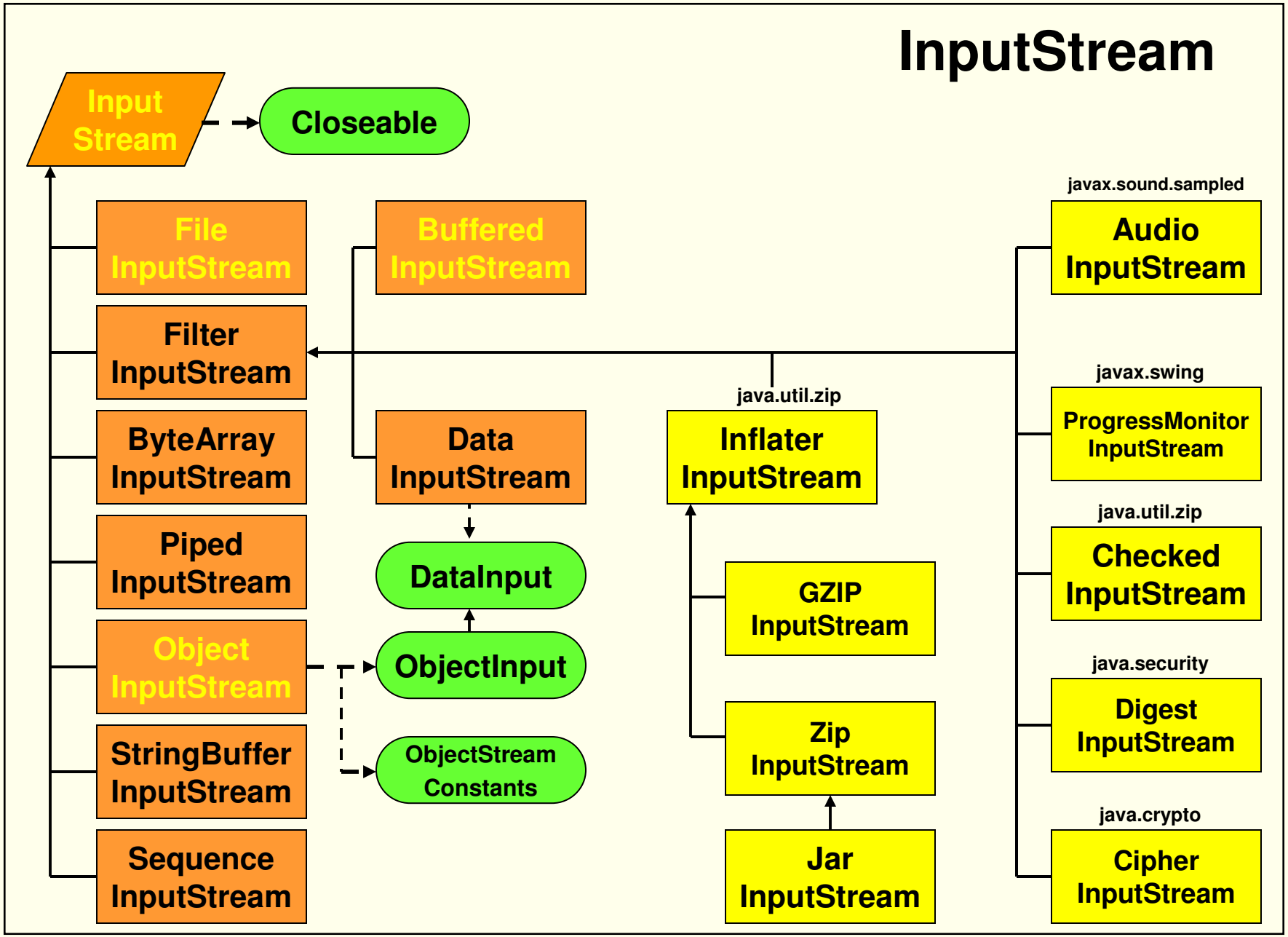
```
static final String spc= "                                                  ";

public static void main( String[ ] args ) {
    dir( null, "C:\\MyDir" , 0 );
 }

static void dir( File dir, String name, int level ) {
   File f =  ( dir == null ) ?  new File( name )  :  new File( dir, name );
   String inset = spc.substring( 0, level );
   System.out.println( inset +  name );
   if ( f.isFile( ) )  return;

   String[ ] sl = f.list( ) ;
   for ( int i = 0; i < sl.length; i++ )  dir( f, sl[ i ], level+1 );
}
```


java.io
java.lang
Comparable<T>
ObjectStream Field
ObjectStream Class
File
tag interface
Serializable
Externalizable
FileFilter
FilenameFilter
File Descriptor
Stream Tokenizer
byte
Input Stream
Output Stream
char
Reader
Writer
input
output
streams
Console
Flushable
java.lang
Appendable
Closeable
Random AccessFile
DataInput
DataOutput


InputStream
Input Stream
Closeable
File InputStream
Buffered InputStream
Filter InputStream
ByteArray InputStream
Data InputStream
Piped InputStream
DataInput
Object InputStream
ObjectInput
StringBuffer InputStream
ObjectStream Constants
Sequence InputStream
java.util.zip
Inflater InputStream
GZIP InputStream
Zip InputStream
Jar InputStream
javax.sound.sampled
Audio InputStream
javax.swing
ProgressMonitor InputStream
java.util.zip
Checked InputStream
java.security
Digest InputStream
java.crypto
Cipher InputStream

Abstraktní třída InputStream

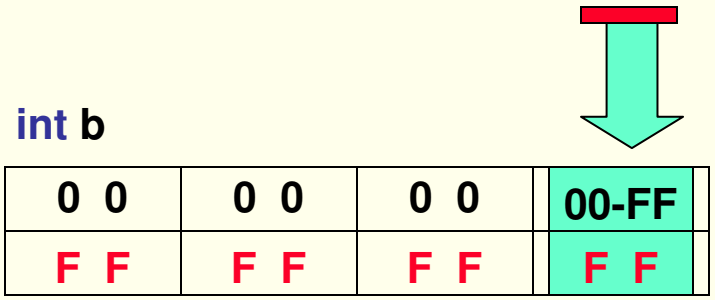
Následující metody jsou public a vyhazují výjimku IOException:

- int available( ) – vrací počet bytů, které lze číst nebo přeskočit bez blokování volajícího vlákna.
- void close( ) – uzavře proud a uvolní příslušné zdroje.
- abstract int read( ) – přečte další byte, vrátí číslo 0 – 255 či -1 (po konci).
- int read(byte[ ] b) – přečte sekvenci bytů do pole b.
- int read(byte[ ] b, int off, int len) – obdoba předchozí metody.
- void reset( ) – nastaví proud na pozici udanou metodou mark( ).
- long skip(long n) – přeskočí n bytů.

Následující metody jsou public:

- void mark(int readlimit) – poznamená pozici v proudu.
- boolean markSupported( ) – zjišťuje zda repozice je podporována.

Čtení a zápis byte streamu

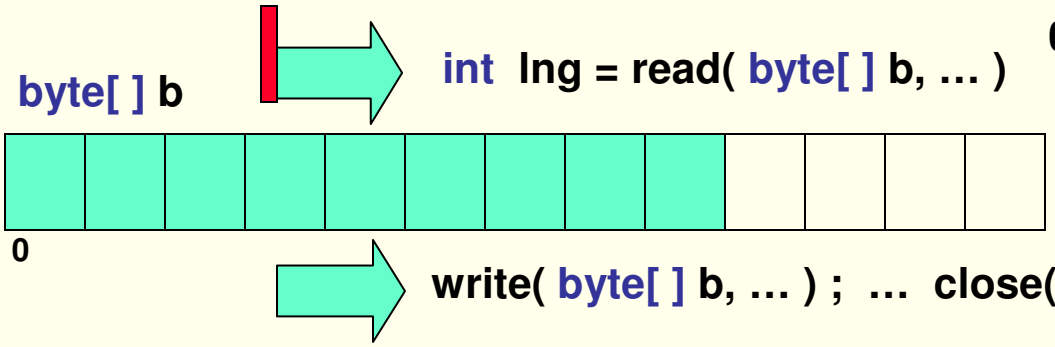


int b = read( )

0 <= b <= 255

b == -1 EOF

write(int b); … close( );



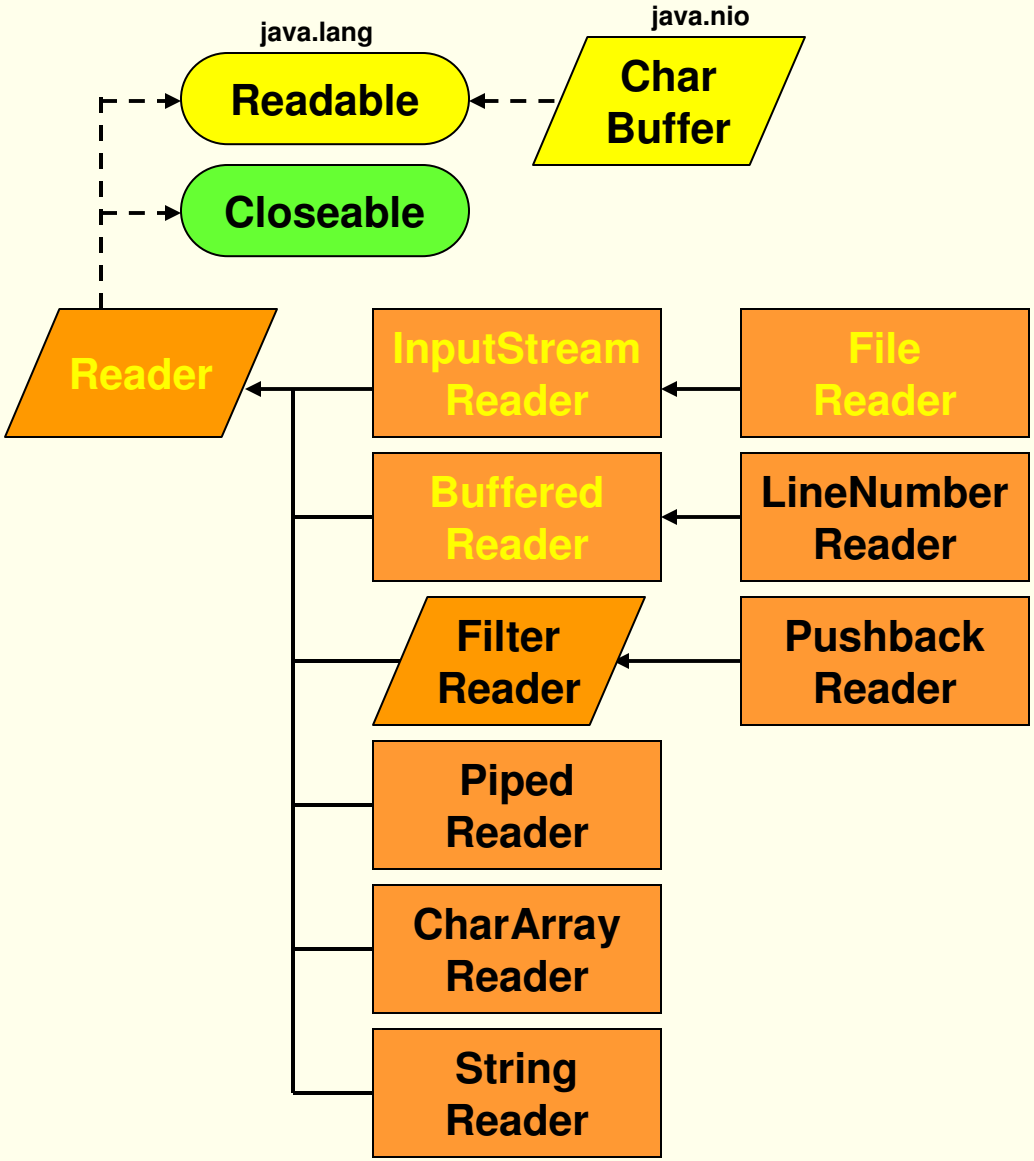
int lng = read(byte[ ] b, …)

0 <= lng < b.length

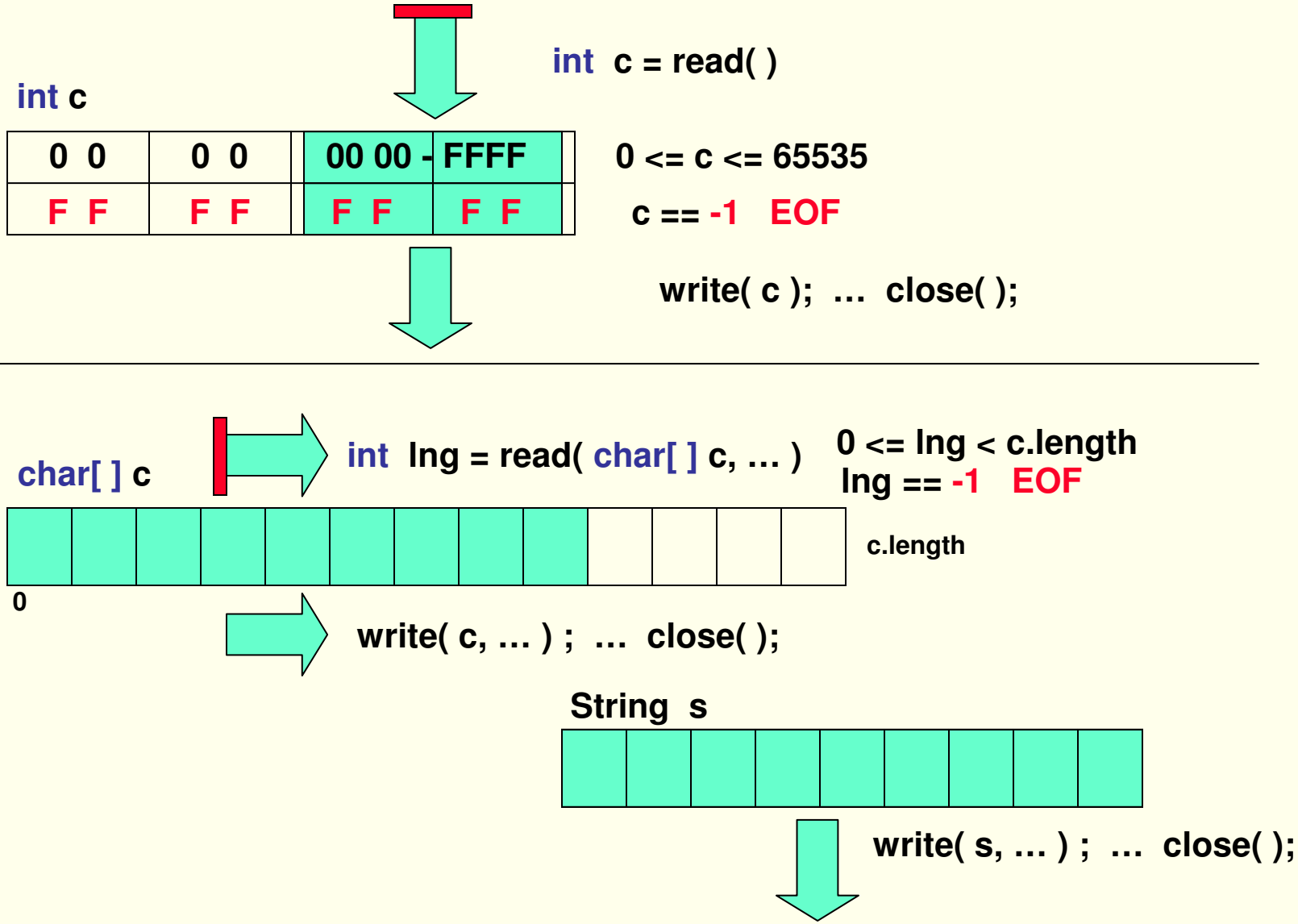
lng == -1 EOF

write(byte[ ] b, …) ; … close( );

Reader


java.lang
java.nio
Readable
Char Buffer
Closeable
Reader
InputStream Reader
File Reader
Buffered Reader
LineNumber Reader
Filter Reader
Pushback Reader
Piped Reader
CharArray Reader
String Reader

Čtení a zápis character streamu



Abstraktní třída Reader

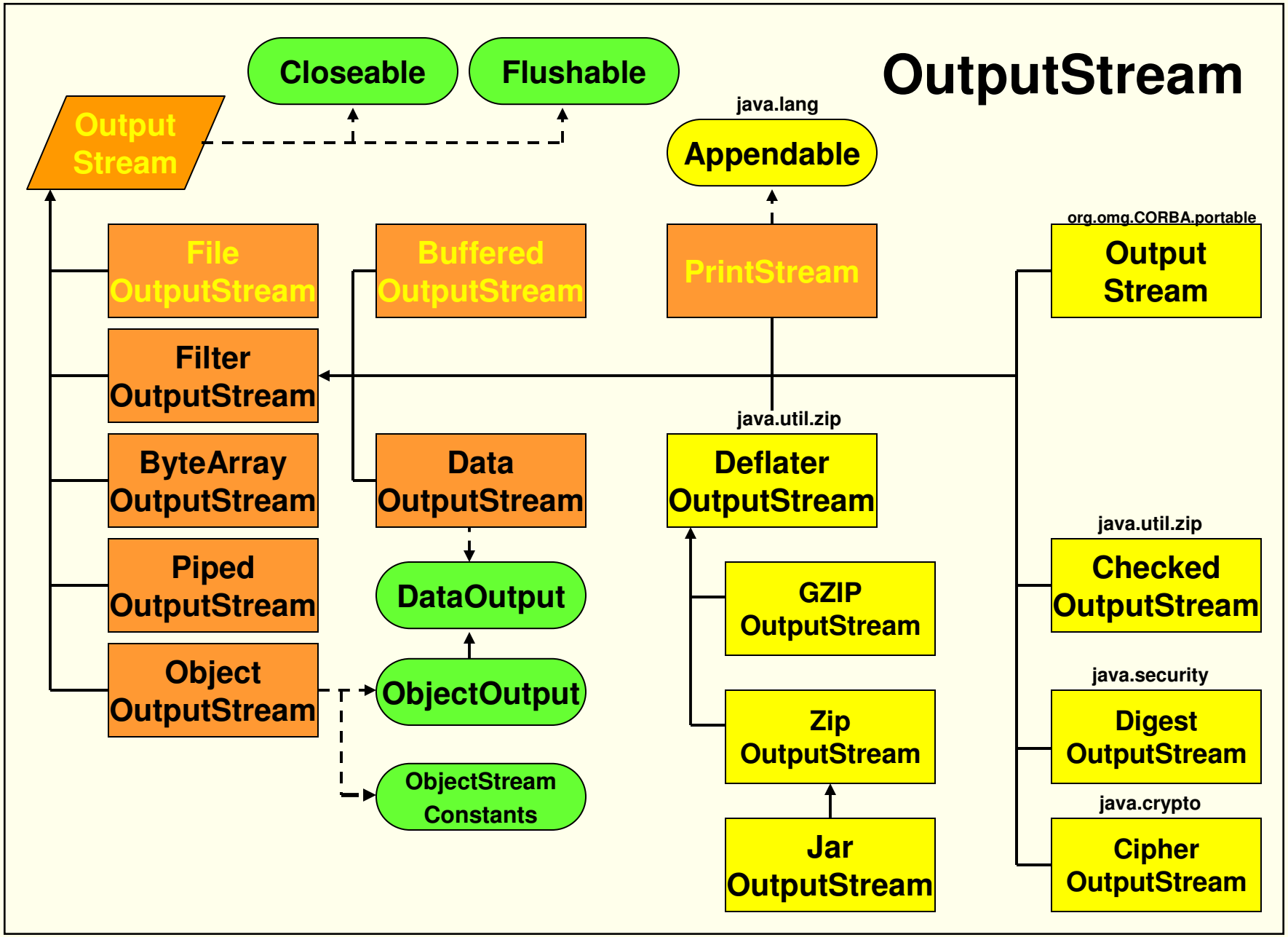
Následující metody jsou public a vyhazují IOException:

- **abstract void close( ) – uzavře proud a uvolní příslušné zdroje.**
- **int read( ) – přečte další znak, vrátí 0 – 65535 anebo -1 (při konci).**
- **int read(char[ ] c) – přečte sekvenci znaků do pole b.**
- **abstract int read(char[ ] c, int off, int len) – obdoba předchozí metody.**
- **boolean ready( ) – testuje připravenost ke čtení.**
- **void reset( ) – nastaví proud na pozici udanou metodou mark( ).**
- **long skip(long *n*) – přeskočí n znaků.**

Následující metody jsou public

- **void mark(int readAheadLimit) – poznamená pozici v proudu.**
- **boolean markSupported( ) – zjišťuje zda repozice je podporována.**

K dispozici je navíc konstruktor protected Reader(Object lock) pro synchronizaci kritických sekcí daného objektu.

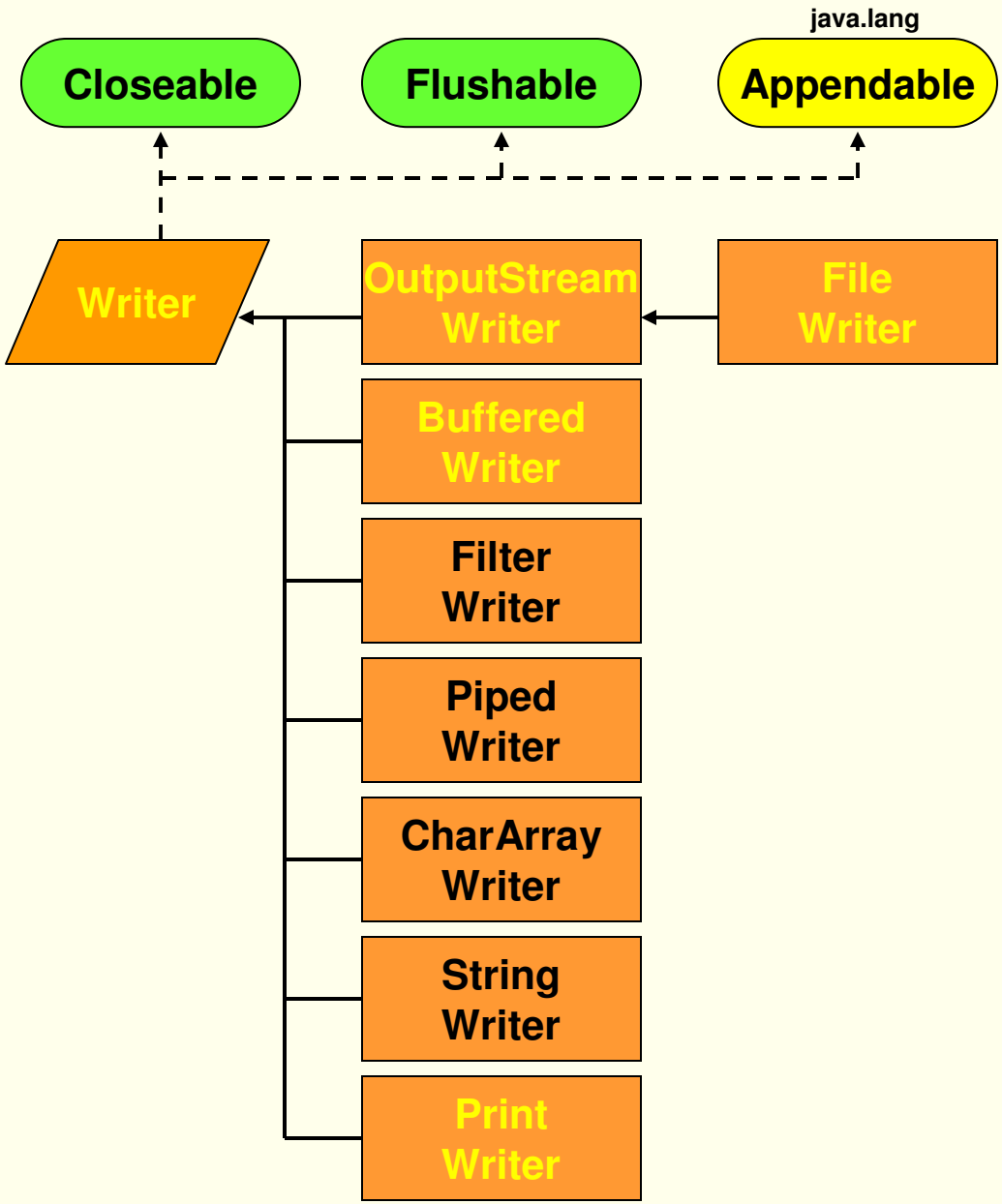

OutputStream
Closeable
Flushable
Output Stream
java.lang
Appendable
File OutputStream
Buffered OutputStream
PrintStream
org.omg.CORBA.portable
Output Stream
Filter OutputStream
java.util.zip
Deflater OutputStream
ByteArray OutputStream
Data OutputStream
java.util.zip
Checked OutputStream
Piped OutputStream
DataOutput
GZIP OutputStream
Object OutputStream
ObjectOutput
java.security
Digest OutputStream
Zip OutputStream
ObjectStream Constants
java.crypto
Jar OutputStream
Cipher OutputStream

Abstraktní třída OutputStream

Následující metody jsou public a vyhazují výjimku IOException*:*

- **void close( ) – uzavře proud a uvolní příslušné zdroje.**
- **void flush( ) – vypudí data do nory.**
- **abstract void write(int *b*) – zapíše obsah nejpravějšího byte do proudu.**
- **void write(byte[ ] *b*) – zapíše obsah pole do proudu dle jeho délky.**
- **void write(byte[ ], int offset, int len) – zapíše jen část obsahu.**

Writer


java.lang
Closeable
Flushable
Appendable
Writer
OutputStream Writer
File Writer
Buffered Writer
Filter Writer
Piped Writer
CharArray Writer
String Writer
Print Writer

Abstraktní třída Writer

Následující metody jsou public a vyhazují výjimku IOException*:*

- **abstract void close( ) – uzavře proud a uvolní příslušné zdroje.**
- **abstract void flush( ) – vypudí data do nory.**
- **abstract void write(int c) – zapíše obsah nejpravějšího znaku do proudu.**
- **void write(char[ ] c) – zapíše obsah pole do proudu dle jeho délky.**
- **void write(char[ ], int offset, int len) – zapíše jen část obsahu.**
- **void write(String s) – zapíše řetěz do pole.**
- **void write(String s, int offset, int len) – zapíše jen část řetězu.**
- **Writer append(char c) – připíše znak.**
- **Writer append(CharSequence csq) – připíše sekvenci znaků.**
- **Writer append(CharSequence csq, int start, int end) – připíše subsekv.**

K dispozici je navíc konstruktor protected Writer(Object lock) pro synchronizaci kritických sekcí daného objektu.

Překlad bytů na znaky

InputStreamReader a OutputStreamWriter slouží jako můstky mezi bytovými a znakovými proudy a dále pro volitelný překlad.

Konstruktory:

- **InputStreamReade*r*(InputStream in)**
- **InputStreamReader(InputStream in, String enc) – s překladem**
- **OutputStreamWriter(OutputStream out)**
- **OutputStreamWriter(OutputStream out, String enc) – s překladem**

Parametr enc – udává kódování – např. UTF8

Pro zvýšení efektivity radno používat buffery. Např.:

- **Reader in = new BufferedReader(new InputStreamReader(System.in));**
- **Writer out = new BufferedWriter(new OutputStreamWriter(System.out));**

Interfejsy DataInput a DataOutput

definují (abstraktní) metody pro čtení / zápis všech primitivů z / do proudu:

- boolean readBoolean( ), byte readByte( ), char readChar( ),
  double readDouble( ), ... atd. a ještě:
  - void readFully(byte[ ] b)
  - int readUnsignedByte( )
  - int readUnsignedShort( )
  - String readUTF( )

- void writeBoolean(boolean v), void writeByte(byte v),
  void writeChar(char v), void writeDouble(double v) ... atd. a ještě:
  - void write(byte[ ] b)
  - void writeUTF(String s)

Třídy DataInputStream, DataOutputStream a RandomAccessFile tyto interfejsy implementují, čímž umějí pracovat se všemi primitivy.

Kopírování souboru po bytech

Přetížené konstruktory FileInputStream a FileOutputStream určují soubor pomocí File nebo FileDescriptor anebo přímo jménem. FileOutputStream navíc umožňuje připsat data ke konci existujícího souboru, což se určí parametrem append v konstruktoru.

Příklad:

```
try {
        File ifile = new File("C:\\in");
        File ofile = new File("C:\\out");
        FileInputStream  fis = new FileInputStream( ifile );
        FileOutputStream fos = new FileOutputStream( ofile );
        int c;
        while ( ( c = fis.read( ) ) != -1 ) {  fos.write(c);  }
        fis.close( ); fos.close( );
} catch ( IOException ex ) { System.err.println( ex ); }
```

Kopírování textového souboru po řádcích

Přetížené konstruktory BufferedReader a BufferedWriter umožňují nastavit velikost bufru. Metoda readLine( ) nevrací znaky přechodu na novou řádku. Writer umožňuje připisovat ke konci existujícího souboru metodou append.

```
BufferedReader  br = null;
BufferedWriter   bw = null;
try {
        br = new BufferedReader(
                    new InputStreamReader(
                          new FileInputStream( "C:\\in.txt" )));
        bw = new BufferedWriter(
                     new OutputStreamWriter(
                           new FileOutputStream( "C:\\out.txt" )));
        String s = null;
        while ( (s = br.readLine( ) ) != null )  { bw.write( s );   bw.newLine( ); }
} catch ( IOException ex ) { System.err.println( ex ); }
finally  { if ( br != null ) br.close( );   if ( bw != null ) bw.close( );  }
```

Filtrování textového souboru po řádcích

```
class  TextFilter extends LineNumberReader {
    String s, stop ;
    public TextFilter( BufferedReader in, String stop ) {
        super( in );
        this.stop=stop;
    }
    public String readLine( ) {
      try {
         while ( (s=super.readLine( ) ) != null ) {
            if ( s.length( ) == 0 ) continue;          // odstraní prázdné řádky
            if ( s.startsWith( stop ) ) return null;   // podmínka ukončení
            return  s;
          }
      }
      catch ( Exception ex ) { }
      return null;
   }
}
```

Překlad souboru

Přetížené konstruktory FileReader a FileWriter určují soubor pomocí File nebo FileDescriptor anebo přímo jménem.

Kopírování souboru po znacích s překladem defaultního kódu do UTF8:

```
public static void main( String[ ] args ) throws Exception {
    FileReader fr = new FileReader( "Test.txt" );
    OutputStreamWriter osw =
        new OutputStreamWriter(
            new FileOutputStream( "Test.UTF8" ), "UTF8" );
    int c;
    while ( (c = fr.read( ) ) != -1 )  osw.write( c );
    fr.close( ); osw.close( );
}
```

Proudění rourou

Konstruktory (nepřipojené se připojí později metodou connect( )):

- **PipedInputStream(PipedOutputStream src)**
- **PipedInputStream( ) – ještě nepřipojený**
- **PipedOutputStream(PipedInputStream snk)**
- **PipedOutputStream( ) – ještě nepřipojený**

Příklad toku náhodných bytů rourou od producenta ke konzumentovi:

```
class Producer extends Thread {
    OutputStream os;
    Producer( OutputStream os ) { this.os = os; }

    public void run( ) {
        for ( int i = 0; i < 10; i++ )
            try { os.write( (int) ( Math.random( ) * 256) ); }
            catch ( IOException ex ) { }
    }
}
```

Proudění rourou

```
class Consumer extends Thread {
    InputStream is;
    Consumer( InputStream is ) { this.is = is; }

    public void run( ) {
        int i;
        try {  while ( ( i = is.read( ) ) != -1 )  System.out.println( i );  }
        catch ( IOException ex ) {  }
    }
}

class TestPipe {
   public static void main( String[ ] args ) throws Exception {
        PipedInputStream pi = new PipedInputStream( );
        PipedOutputStream po = new PipedOutputStream( pi );
        new Producer( po ).start( );
        new Consumer( pi ).start( );
    }
}
```

Serializace výstupu

Objekty implementující Serializable lze přenést objektovým proudem tzv. serializací, jež rozloží hodnoty jeho (nestatických) atributů na byty, přičemž modifikátory přístupu nemají vliv. Všechna pole jsou serializovatelná. Neserializují se atributy označené static či transient ani třídy ani metody. Rekurzivní probírkou referencí se serializují i všechny referované objekty – může to být rozsáhlý graf. Všechny jeho objekty musí být Serializable.

```
FileOutputStream      out = new FileOutputStream( "theTime" );
ObjectOutputStream  oos = new ObjectOutputStream( out );

oos.writeObject( "Today" );
oos.writeObject( new Date( ) );
oos.flush( );
```

Každý další stav téhož objektu lze zapsat jen zavoláním oos.reset( ) před vlastním zápisem.

Deserializace vstupu

je obnovení objektu (tj. i celého grafu). Přičemž musejí být k dispozici odpovídající třídy - kompatibilní dle serialVersionUID.
Deserializace konstruktory Serializable tříd nevolá, neboť by nastavily iniciální hodnoty. Volá však konstruktory nadtříd které nejsou Serializable.
Nepřenesené atributy nastaví na jejich defaultní hodnoty dle jejich typu.

```
FileInputStream in       = new FileInputStream( "theTime" );
ObjectInputStream ois = new ObjectInputStream( in );
String today = ( String ) ois.readObject( );
Date date     = ( Date )   ois.readObject( );
```

Jelikož metoda ois.readObject( ) vrací typ Object, je vhodné přetypovat.
Zjistit typ lze metodou getClass( ) anebo operátorem instanceof.

Objektové proudy implementují DataInput resp. DataOutput a tudíž lze proudem přenášet i primitivní hodnoty, řetězy a UTF.

serialVersionUID

Serializovatelné třídy mají identifikaci verze pomocí přiřazené konstanty serialVersionUID stanovené výpočtem či přiřazením.

Hodnotu serialVersionUID lze zjistit programem

jdk1.6\bin\serialver.exe –classpath .... class, ...

což je hešová hodnota odvozená z některých charakteristik třídy.

Při serializaci se přikládá do objektového proudu – viz třída java.io.ObjectStreamConstants. Při deserializaci se porovná s hodnotou třídy na vstupní straně. Při neshodě se vyhodí výjimka.

Programátor může explicitně stanovit, že třídy jsou kompatibilní tím, že třídě vnutí určitou hodnotu atributu např:

```
private static final long serialVersionUID=333333333333333333333333L;
```

Ovládání serializace a deserializace

Průběh a pořadí lze ovládat připsáním metod do serializovatelné třídy

```
class A  implements Serializable {
    static int i = 3;
    transient int j = 5;
    double d = 1.41;
    private void writeObject( ObjectOutputStream oos ) throws Exception {
       oos.writeInt( i );
       oos.defaultWriteObject( );          // serializuje normálně
    }
    private void readObject( ObjectInputStream ois ) throws Exception {
       i = ois.readInt( );
       ois.defaultReadObject( );          // deserializuje normálně
       j = -1;                             //  bez  přenosu – jen nastavení
    }
}
```

Externalizable

Pro realizaci vlastních představ lze implementovat třídu interfejsem java.io.Externalizable, což znamená realizovat explicitní konstruktor bez parametrů a tyto dvě metody:

```
public void writeExternal( ObjectOutput stream ) throws IOException {
    stream.writeInt( i );
    stream.writeObject( "Well done." );
 }

public void readExternal( ObjectInput stream )
    throws IOException, ClassNotFoundException {
    i = stream.readInt( );
    String msg = ( String ) ( stream.readObject( ) );
    System.out.println( msg );
 }
```

Externalizovaný vstup používá konstruktor, serializovaný nikoli.

Komprese

Lze vytvořit standardní soubor .zip, .gzip nebo .jar, který je čitelný standardními programy - např. WinZip. Kompresi provádí Deflater.

```
OutputStream  os = new FileOutputStream( "C:\\data.zip" );
ZipOutputStream  zos = new ZipOutputStream( os );

ZipEntry ze1 =  new ZipEntry( "dir1 \\ dir2 \\ YY" );
zos.putNextEntry( ze1 );
     for ( int i = 0; i < 10000; i++ ) {  zos.write( i );  }

ZipEntry z2 =  new ZipEntry( "dir1 \\ ZZ" );
zos.putNextEntry( ze2 );
    for ( int i = 0; i < 10000; i++ ) {  zos.write( i );  }
 ...
 zos.close();
```

Dekomprese

Takto lze dekomprimovat standardní .zip, .gzip a .jar.

Dekompresi provádí Inflater.

```
InputStream  is = new FileInputStream( "C:\\data.zip" );
ZipInputStream  zis = new ZipInputStream( is );

ZipEntry ze;
int i;
while ( ( ze = zis.getNextEntry( ) ) != null  ) {
    while ( ( i = zis.read( ) ) != -1 ) {
        System.out.print( i + "  "  );
    }
}
zis.close( );
```

ZipFile a ZipEntry

Uložení entry a dat je patrně různé při kompresi ZipOutputStreamem a některými oblíbenými programy – např. WinZip 7.0.
Pro čtení a dekompresi je pak třeba použít ZipFile.

```
ZipFile zf = new ZipFile ( "C:\\windata.zip" );

Enumeration en = zf.entries( );
while ( en.hasMoreElements( ) ) {  System.out.print(  en.nextElement( ) ;  }

ZipEntry ze = zf.getEntry( "dir1 / ZZ" );
InputStream is = zf.get InputStream( ze ) ;
int i;
while ( ( i = zis.read( ) ) != -1 )  {  System.out.print( i + "  "  ) ;   }
zis.close( ) ;
zf.close( ) ;
```

Třída RandomAccessFile

Umožňuje libovolný přístup, čtení, zápis i připisování jakoby bytového pole realizovaného na periferním zařízení. Konstruktory mají mody: r, rw – pro čtení i zápis, rws, rwd – pro synchonizovanou aktualizaci bratra.

Instanční metody:

- **void close( ) – uzavře soubor**
- **long getFilePointer ( ) - vrátí pointer**
- **void seek (long pos) - nastaví pointer**
- **read (…) , readXXX(…), readLine – různé způsoby čtení**
- **write (…), writeXXX(…) - různé způsoby psaní**
- **long length ( ) - zjistí rozsah souboru**
- **void setLength (long newLength) - zvětší-zmenší rozsah souboru**
- **int skipBytes (int n) - přeskočí**
- **FileChannel getChannel ( ) – vrátí kanál bratra**
- **FileDecriptor getFD ( ) – umožňuje připojení a synchronizaci bratra**