Az „új” be- és kimenet könyvtár

```
import java.io.IOException;
import java.io.FileInputStream;
import java.io.FileOutputStream;

import java.nio.channels.FileChannel;
import java.nio.ByteBuffer;

class Masol {
  static void masol( FileChannel in, FileChannel out ) throws IOException {
     ByteBuffer buffer = ByteBuffer.allocate(1024);
     buffer.clear();
     while( in.read(buffer)  != -1 ){
        buffer.flip();
        out.write(buffer);
        buffer.clear();
     }
  }
  public static void main( String[] args ) throws IOException {
     FileChannel in  = (new  FileInputStream(args[0])) .getChannel();
     FileChannel out = (new FileOutputStream(args[1])) .getChannel();
     masol(in,out);
     out.close();
     in.close();
  }
}
```

```
static void másol( ReadableByteChannel in, WritableByteChannel out ) throws IOException {
   ByteBuffer buffer = ByteBuffer.allocate(1024);
   while(  in.read( (ByteBuffer)buffer.clear() )  !=  -1  ){
      out.write( (ByteBuffer)buffer.flip() );
   }
}
```

```
public static void main( String[] args ) throws IOException {
   FileInputStream fin = new FileInputStream(args[0]);
   try {
       FileChannel in = fin.getChannel();
       FileOutputStream fout = new FileOutputStream(args[1]);
       try {
           FileChannel out = fout.getChannel();
           másol(in,out);
      } finally { fout.close(); }
   } finally { fin.close(); }
}
```

1. A bufferek működése

```
public static void main( String[] args ) throws IOException {
   ByteBuffer buf = ByteBuffer.allocate(100);
   buf.clear();
   buf.put((byte)3); buf.put((byte)5); buf.put((byte)7); buf.put((byte)11);
   buf.flip();
   FileOutputStream fout = new FileOutputStream(args[0]);
   try {
       FileChannel out = fout.getChannel();
       out.write(buf);
   } finally { fout.close(); }
}
```

```
buf.put((byte)3).put((byte)5).put((byte)7).put((byte)11);
```

```
buf.put( new byte[]{3,5,7,11} );
```

```
buf.put( primek, 1, 4 );
```

```
static boolean benneVan( byte minta, ReadableByteChannel in ) throws IOException {
  ByteBuffer buffer = ByteBuffer.allocate(1024);
  while( in.read((ByteBuffer)buffer.clear()) != -1 ){
      buffer.flip();
      while( buffer.position() < buffer.limit() ){
          if( buffer.get() == minta ){ return true; }
      }
  }
  return false;
}
```

```
while( buffer.hasRemaining() ){
   if( buffer.get() == minta ){ return true; }
}
```

```
public ByteBuffer get( byte[] dst )
public ByteBuffer get( byte[] dst, int offset, int length )
```

```
public ByteBuffer put( int index, byte b )
public byte get( int index )
```

```
public final int position()
public final int limit()
public final Buffer position(int newPosition)
public final Buffer limit(int newLimit)
```

2. Bufferek létrehozása

2.1. Tömb a buffer belsejében

```
static void kitolt( ByteBuffer buffer, byte ertek ){
   if( buffer.hasArray() ){
      byte[] tartalom = buffer.array();
      int kezdoPozicio = buffer.arrayOffset();
      int utolsoPozicioPluszEgy = kezdoPozicio + buffer.capacity();
      java.util.Arrays.fill(tartalom,kezdoPozicio,utolsoPozicioPluszEgy,ertek);
   } else {
      ...
   }
}
```

```
static void kiir( byte[] adatok, WritableByteChannel out ) throws IOException {
   ByteBuffer buffer = ByteBuffer.wrap(adatok);
   out.write( buffer );
}
```

2.2. Bufferek tartalmának megosztása

```
ByteBuffer masik = buffer.duplicate();
ByteBuffer harmadik = buffer.slice();
```

```
import java.nio.ByteBuffer;
class Szelet {
   public static void main( String[] args ){
      ByteBuffer buffer = ByteBuffer.allocate(1024);
      for( byte b=0; b<10; ++b ){
         buffer.put( b );
      }
      buffer.position(3).limit(7);
      ByteBuffer szelet = buffer.slice();
      for( int i=0; i<szelet.capacity(); ++i ){
         byte b = szelet.get(i);
         b *= 11;
         szelet.put( i, b );
      }
      buffer.position(0).limit(10);
      while( buffer.remaining() > 0 ){
         System.out.print( buffer.get() + " " );
      }
   }
}
```

```
0 1 2 33 44 55 66 7 8 9
```

2.3. Különböző primitív típusok támogatása

```
static double osszeg( ReadableByteChannel in ) throws IOException {
   double osszeg = 0.0;
   ByteBuffer buffer = ByteBuffer.allocate(1024);
   while( in.read((ByteBuffer)buffer.clear()) != -1 ){
      buffer.flip();
      FloatBuffer adatok = buffer.asFloatBuffer();
      while( adatok.hasRemaining() ){
         osszeg += adatok.get();
      }
   }
   return osszeg;
}
```

```
static double osszeg( ReadableByteChannel in ) throws IOException {
   double osszeg = 0.0;
   ByteBuffer buffer = ByteBuffer.allocate(1024);
   FloatBuffer adatok = buffer.asFloatBuffer();
   while( in.read((ByteBuffer)buffer.clear()) != -1 ){
      adatok.position(0).limit(buffer.remaining()/4);
      while( adatok.hasRemaining() ){
         osszeg += adatok.get();
      }
   }
   return osszeg;
}
```

```
static double osszeg( ReadableByteChannel in ) throws IOException {
   double osszeg = 0.0;
   ByteBuffer buffer = ByteBuffer.allocate(1024);
   while( in.read((ByteBuffer)buffer.clear()) != -1 ){
      buffer.flip();
      while( buffer.hasRemaining() ){
         osszeg += buffer.getFloat();
      }
   }
   return osszeg;
}
```

```
static void becsomagol( ByteBuffer buffer, Szamla szamla ){
   buffer.clear();
   buffer.putLong(szamla.azonosito());
   buffer.putInt(szamla.egyenleg());
   int[] tranzakciok = szamla.tranzakciok();
   buffer.putShort((short)tranzakciok.length);
   IntBuffer nezet = buffer.asIntBuffer();
   nezet.put(tranzakciok);
   buffer.position( buffer.position() + 4*tranzakciok.length );
   buffer.flip();
}
```

```
buffer.order(ByteOrder.LITTLE_ENDIAN);
```

2.4. Csak olvasható bufferek

```
static long adatokatOlvasEsValamitKiszamol( IntBuffer buffer ){...}
...
public static void main( String[] args ){
   IntBuffer buffer;
   ... // létrehozzuk a buffert és feltöltjük adatokkal
   long eredmeny = adatokatOlvasEsValamitKiszamol(buffer.asReadOnlyBuffer());
   ... // felhasználjuk a kiszámított eredményt
}
```

2.5. Közvetlen és leképzett bufferek

```
FileChannel channel = new RandomAccessFile("large.dat","rw").getChannel();
MappedByteBuffer buffer = channel.map(FileChannel.MapMode.READ_WRITE,
                                      0, channel.size());
channel.close();
buffer.load();
int limit = buffer.limit();
for( int i = buffer.position(); i < limit; ++i ){
   buffer.put( i, (byte) (buffer.get(i)+1) );
}
buffer.force();
```

3. Nem blokkoló IO

```
try {
   while( true ){
      out.writeDouble( (double) in.readShort() );
      out.flush();
   }
} catch( java.io.EOFException eof ){}
```

```
Selector selector = Selector.open();
try { /* használjuk */ } /* esetleg catch-ek */ finally { selector.close(); }
```

```
SelectableChannel channel = ...
channel.configureBlocking(false);
channel.register( selector, SelectionKey.OP_READ | SelectionKey.OP_WRITE );
```

```
s.select();
Set<SelectionKey> feldolgozhatók = s.selectedKeys();
for( SelectionKey feldolgozható: feldolgozhatók ){
   SelectableChannel channel = feldolgozható.channel();
   if( feldolgozható.isReadable() ){
      // olvassunk channel-ről
   }
   if( feldolgozható.isWritable() ){
      // írjuk ki channel-re, amit szeretnénk
   }
}
feldolgozhatók.clear();
```

```
SelectionKey feldolgozható = ...  // egy feldolgozható csatorna kulcsa
SelectableChannel channel = feldolgozható.channel();
ByteBuffer buffer = ...
...
int beolvasottBájtokSzáma = ((ReadableByteChannel)channel).read(buffer);
if( beolvasottBájtokSzáma == -1 ){
   // EOF jel, a partner bontotta a kapcsolatot
   feldolgozható.cancel();
   channel.close();
} else {
   ...
}
```

4. Egy bonyolultabb példa

4.1. Többszálú szerver

```
class EgyKliensKiszolgálása extends Thread {
   private final java.net.Socket socket;
   private final DataInputStream in;
   private final DataOutputStream out;
   EgyKliensKiszolgálása( java.net.Socket socket ) throws IOException {
      this.socket = socket;
      in = new DataInputStream(socket.getInputStream());
      out = new DataOutputStream(socket.getOutputStream());
   }
   public void run(){
      try {
         while( true ){
            out.writeDouble( válasz(in.readShort()) );
            out.flush();
         }
      } catch( EOFException eof ){
      } catch( Throwable t ){
         // nem várt kivétel log-olása
      } finally {
         try { socket.close(); }
         catch( Throwable t ){ /* nem várt kivétel log-olása */ }
      }
   }
   double válasz( short kérés ){
      return (double) kérés;
   }
}
```

```
import java.io.*;

class EgyKliensKiszolgálása extends Thread { ... }

class Szerver {
   public static void main( String[] args ) throws IOException {
      java.net.ServerSocket kapcsolódásiHely = new java.net.ServerSocket(9999);
      while( true ){
         java.net.Socket kapcsolat = null;
         try {
            kapcsolat = kapcsolódásiHely.accept();
            new EgyKliensKiszolgálása(kapcsolat).start();
         } catch( IOException e ){
            // kivétel naplózása
            if( kapcsolat != null ){
               try { kapcsolat.close(); }
               catch( IOException ioe ){ /* kivétel naplózása */ }
            }
         }
      }
      // kapcsolódásiHely.close(); }
   }
}
```

4.2. Több kliens kiszolgálása egy szálon

```
import java.io.*;
import java.nio.*;
import java.nio.channels.*;
import java.util.*;
import java.net.*;

class SzelektívVárakozás {

   public static void main( String[] args ) throws IOException {
      final Selector multiplexer = Selector.open();
      try {
         ServerSocketChannel kapcsolódásiCsatorna = null;
         try {
            kapcsolódásiCsatorna = kapcsolódásiCsatornátLétrehoz(9999);
            kapcsolódásiCsatorna.register( multiplexer, SelectionKey.OP_ACCEPT );
            leállítóSzálatElindít(multiplexer);
            System.out.println("Kliensek kiszolgálása. Leállítás: <Enter>");
            klienseketKiszolgál(multiplexer);
            System.out.println("Leállítás...");
         } finally {
            if( kapcsolódásiCsatorna != null ){ kapcsolódásiCsatorna.close(); }
         }
      } finally { multiplexer.close(); }
   }

   ...

}
```

```
private static ServerSocketChannel kapcsolódásiCsatornátLétrehoz( int port )
throws IOException {
   final ServerSocketChannel kapcsolódásiCsatorna = ServerSocketChannel.open();
   kapcsolódásiCsatorna.configureBlocking(false);
   final InetAddress localhost = InetAddress.getLocalHost();
   final InetSocketAddress address = new InetSocketAddress(localhost, port);
   kapcsolódásiCsatorna.socket().bind(address);
   return kapcsolódásiCsatorna;
}
```

```
private static volatile boolean legyenVége = false;

private static void leállítóSzálatElindít( final Selector multiplexer ){
   assert multiplexer != null;
   Thread leállítóSzál = new Thread(){
                                 public void run(){
                                    try { System.in.read(); }
                                    catch( IOException e ){}
                                    legyenVége = true;
                                    multiplexer.wakeup();
                                 }
                              };
   leállítóSzál.setDaemon(true);
   leállítóSzál.start();
}
```

```
private static class KliensKezelő {
   final ByteBuffer buffer = ByteBuffer.allocate(2);              // short
   private final ByteBuffer outBuffer = ByteBuffer.allocate(8);   // double
   private final SocketChannel csatorna;
   KliensKezelő( SocketChannel csatorna ){
      this.csatorna = csatorna;
   }
   void válaszol() throws IOException {
      if( buffer.remaining() == 0 ){
         buffer.flip();
         final short kérés = buffer.getShort();
         buffer.clear();
         final double válasz = válasz(kérés);
         outBuffer.clear();
         outBuffer.putDouble(válasz);
         outBuffer.flip();
         csatorna.write(outBuffer);
      }
   }
   static double válasz( short kérés ){
      return (double)kérés;
   }
}
```

```
private static void klienseketKiszolgál( Selector multiplexer ) throws IOException {
   assert multiplexer != null;
   Map<SocketChannel,KliensKezelő> kliensek = new HashMap<SocketChannel,KliensKezelő>();
   try {
      while( !legyenVége ){
         multiplexer.select();
         if( !legyenVége ){
            Set<SelectionKey> kulcsok = multiplexer.selectedKeys();
            for( SelectionKey kulcs: kulcsok ){
               SelectableChannel csatorna = kulcs.channel();
               if( csatorna instanceof ServerSocketChannel ){
                  kapcsolatotLétrehoz( (ServerSocketChannel)csatorna, multiplexer, kliensek );
               } else if ( csatorna instanceof SocketChannel ){
                  klienstKiszolgál( (SocketChannel)csatorna, kulcs, kliensek );
               } else { assert false; }
            }
            kulcsok.clear();
         }
      }
   } finally {
      klienseketLezár(multiplexer);
   }
}
```

```
private static void kapcsolatotLétrehoz(
   ServerSocketChannel kapcsolódás,
   Selector multiplexer,
   Map<SocketChannel,KliensKezelő> kliensek
){
   assert kapcsolódás.keyFor(multiplexer).isAcceptable();
   SocketChannel kapcsolat = null;
   try {
      kapcsolat = kapcsolódás.accept();
      assert kapcsolat != null;
      kapcsolat.configureBlocking(false);
      kapcsolat.register(multiplexer,SelectionKey.OP_READ);
      kliensek.put( kapcsolat, new KliensKezelő(kapcsolat) );
   } catch( IOException e ){
      hibaüzenet("Error when accepting a connection.",e);
      if( kapcsolat != null ){ csatornátLezár(kapcsolat, null); }
   }
}
```

```
private static void klienstKiszolgál(
   SocketChannel kapcsolat,
   SelectionKey kulcs,
   Map<SocketChannel,KliensKezelő> kliensek
){
   assert kapcsolat.equals(kulcs.channel()) && kulcs.isReadable()
          && kliensek.containsKey(kapcsolat);
   KliensKezelő kliensKezelő = kliensek.get(kapcsolat);
   try {
      if( kapcsolat.read(kliensKezelő.buffer) == -1 ){
         kliensek.remove(kapcsolat);
         csatornátLezár(kapcsolat,kulcs);
      } else {
         kliensKezelő.válaszol();
      }
   } catch( IOException e ){
      hibaüzenet("Error during communication with a client.",e);
      csatornátLezár(kapcsolat,kulcs);
   }
}
```

```
private static void klienseketLezár( Selector multiplexer ){
   assert multiplexer != null;
   for( SelectionKey kulcs: multiplexer.keys() ){
      SelectableChannel csatorna = kulcs.channel();
      if( csatorna instanceof SocketChannel ){
         csatornátLezár((SocketChannel)csatorna, kulcs);
      }
   }
}

private static void csatornátLezár( SocketChannel kapcsolat, SelectionKey kulcs ){
   assert (kapcsolat != null) && (kulcs == null || kulcs.channel().equals(kapcsolat));
   try {
      if( kulcs != null ) kulcs.cancel();
      kapcsolat.close();
   } catch( IOException e ){
      hibaüzenet("Error when closing a connection.",e);
   }
}
```

```
private static void hibaüzenet( String üzenet, Throwable kivétel ){
   if( üzenet != null ){ System.err.println(üzenet); }
   if( kivétel != null ){ kivétel.printStackTrace(); }
}
```