Nonblocking I/O

Hyang-Won Lee
Department of Computer Science and Engineering
Konkuk University
leehw@konkuk.ac.kr
https://sites.google.com/view/leehwko/

Revisiting Multithreaded Program

- Limitations of naive multithreaded servers
  - one thread for each connection or request ⟹ too many threads for server with high volume of requests
  - overhead (thread is not free)
    - spawning threads
    - switching between threads
    - memory
- Solutions
  - thread pool
  - nonblocking I/O

Blocking I/O vs Nonblocking I/O

- ServerSocket.wait(), InputStream.read(),…
  - block I/O operations (CPU idled)
- Another thread can grab CPU
  - many threads can help but again threads require resources for management
- Nonblocking I/O (in a single thread)
  - when I/O operation cannot be performed, switch to other operations
  - e.g., server reading from and writing to client
    - when client socket is not ready to be written, try to read from another client

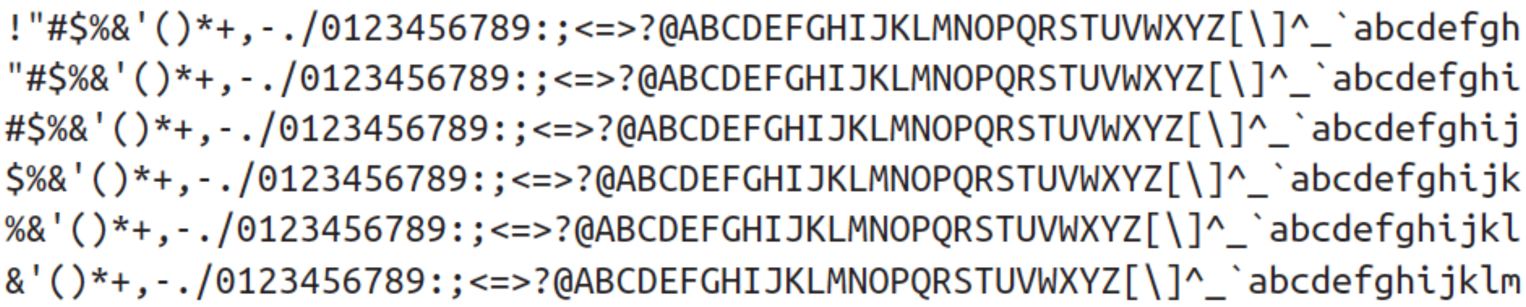
When is Nonblocking I/O useful?

- Example scenario
  - a server needs to serve a huge number of long-lived, simultaneous connections, say +10,000
  - each client doesn't send much data
- One thread for each connection
  - too many active threads
- Thread pool
  - can be a bad choice because it takes long until a thread is released to pool (other connections may have to wait for long)
- Nonblocking I/O can be a winner

Example Client

Chargen Client

- RFC 864
  - when client connects to server, server keeps sending ASCII characters



```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefgh
"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghi
#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghij
$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijk
%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijkl
&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklm
```

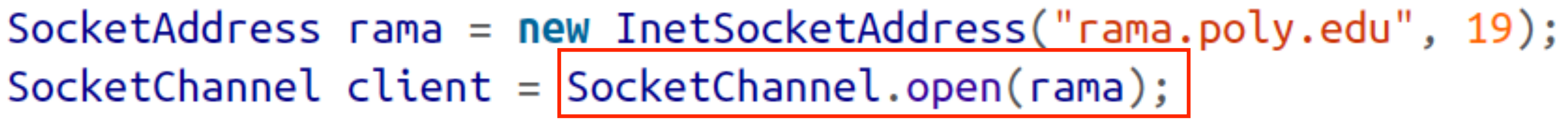
\r\n added

⋮

72 characters rotating out of 95 ASCII printable characters

SocketChannel

- java.nio.channels.SocketChannel
- Opening a channel



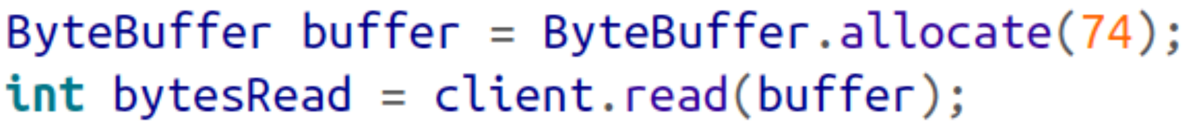
```
SocketAddress rama = new InetSocketAddress("rama.poly.edu", 19);
SocketChannel client = SocketChannel.open(rama);
```

opened in blocking mode
(next line executed only after conn established)

  - IOException is thrown if connection can’t be established

Reading/Writing

- Reading
  - input streams are not needed
  - read directly from channel

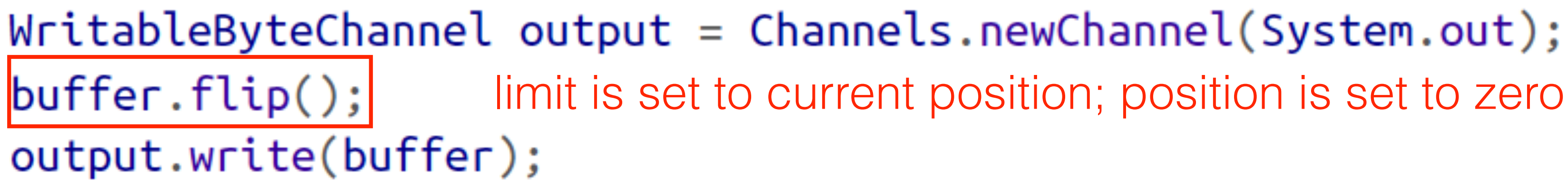


```
ByteBuffer buffer = ByteBuffer.allocate(74);
int bytesRead = client.read(buffer);
```

why 74 bytes?

read a sequence of bytes from client into buffer
read at least one byte or return -1 to indicate the end of data

- Writing
  - output streams are not needed
  - write directly to channel



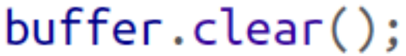
```
WritableByteChannel output = Channels.newChannel(System.out);
buffer.flip();
output.write(buffer);
```

limit is set to current position; position is set to zero

don’t need to tell output channel how many bytes to write; buffer keeps track of it

Reusing Buffer

- Creating buffer for every read/write operation can kill performance (slow down)
- Reuse existing buffer
  - need to reset position to zero



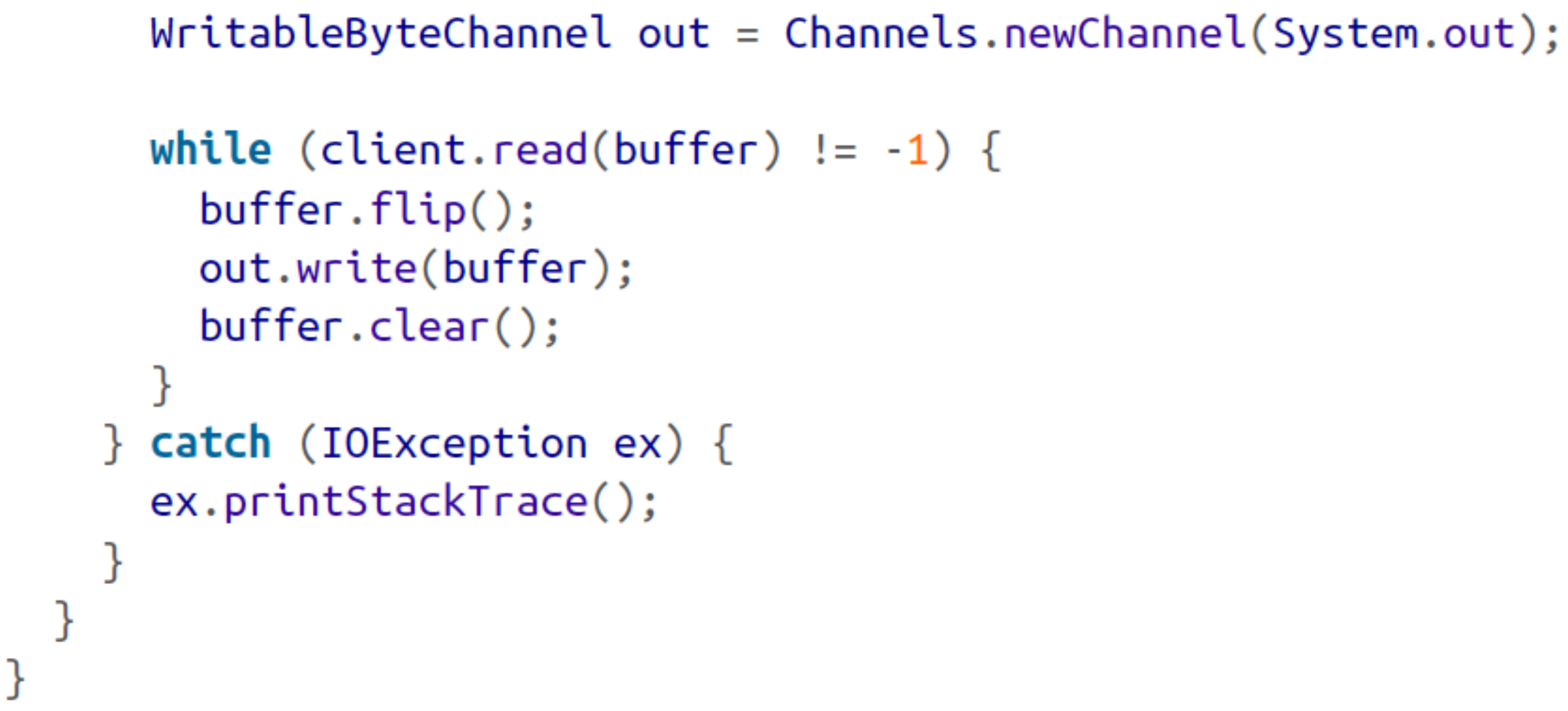
```
buffer.clear();
```

doesn’t delete data, but data will be overwritten

- flip() vs. clear()
  - flipping prepares for writing (write buffer data to channel)
  - clearing prepares for reading (write data read from channel to buffer)

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

public class ChargenClient {

  public static int DEFAULT_PORT = 19;

  public static void main(String[] args) {

    if (args.length == 0) {
      System.out.println("Usage: java ChargenClient host [port]");
      return;
    }

    int port;
    try {
      port = Integer.parseInt(args[1]);
    } catch (RuntimeException ex) {
      port = DEFAULT_PORT;
    }

    try {
      SocketAddress address = new InetSocketAddress(args[0], port);
      SocketChannel client = SocketChannel.open(address);

      ByteBuffer buffer = ByteBuffer.allocate(74);
```

Example (contd.)

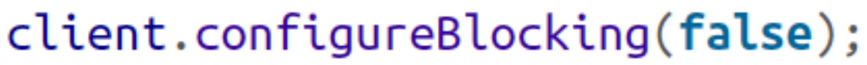


```
      WritableByteChannel out = Channels.newChannel(System.out);

      while (client.read(buffer) != -1) {
        buffer.flip();
        out.write(buffer);
        buffer.clear();
      }
    } catch (IOException ex) {
      ex.printStackTrace();
    }
  }
}
```

- Exercise
  - save the buffer content to a file

Activating Nonblocking Mode

- The example code works in blocking mode
- Activating nonblocking mode



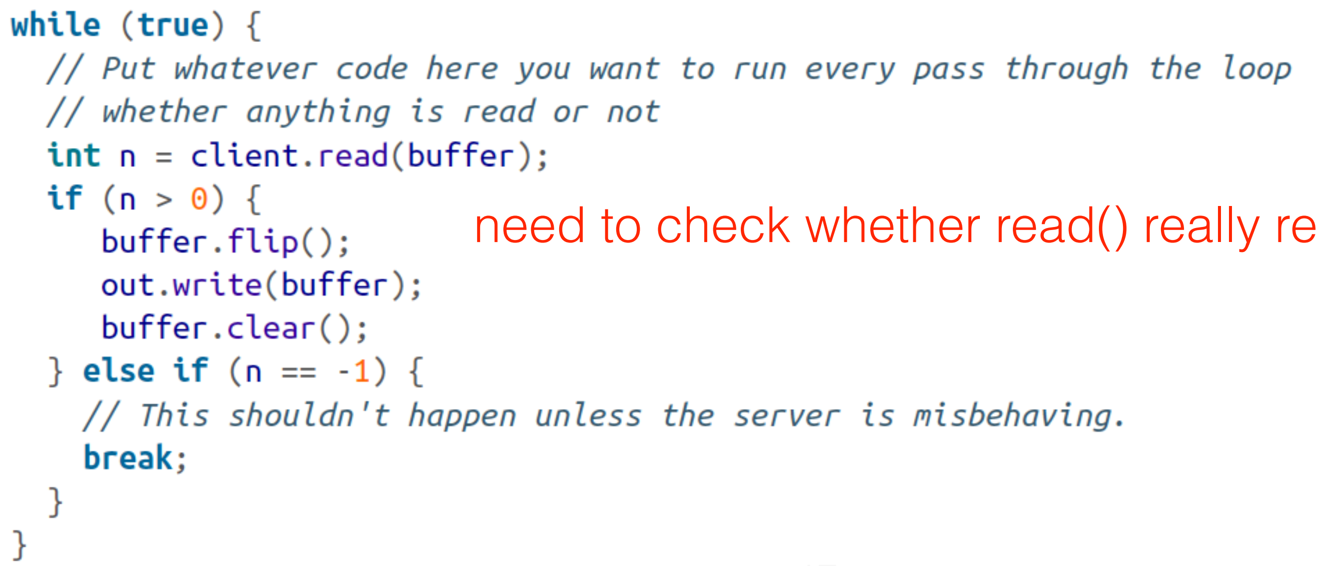
```
client.configureBlocking(false);
```

don’t block

true means block

  - read() returns immediately even if there is no data available to read

- Modified code



```
while (true) {
  // Put whatever code here you want to run every pass through the loop
  // whether anything is read or not
  int n = client.read(buffer);
  if (n > 0) {
     buffer.flip();
     out.write(buffer);
     buffer.clear();
  } else if (n == -1) {
    // This shouldn't happen unless the server is misbehaving.
    break;
  }
}
```

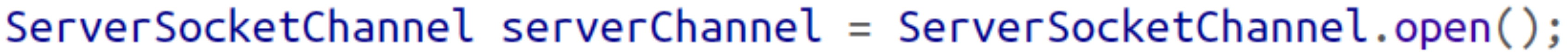
need to check whether read() really read data or not

Example Server

Chargen Server

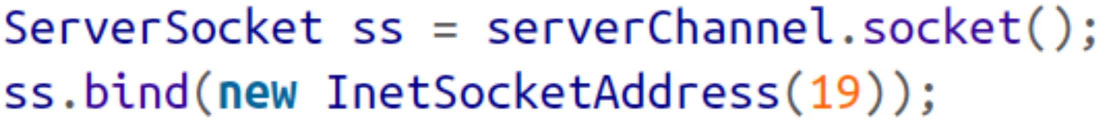
ServerSocketChannel

- Opening channel



```
ServerSocketChannel serverChannel = ServerSocketChannel.open();
```

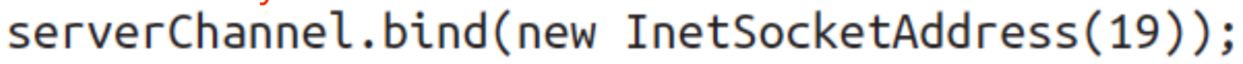
this doesn't listen to any port (until it's bound to a port)



```
ServerSocket ss = serverChannel.socket();
ss.bind(new InetSocketAddress(19));
```

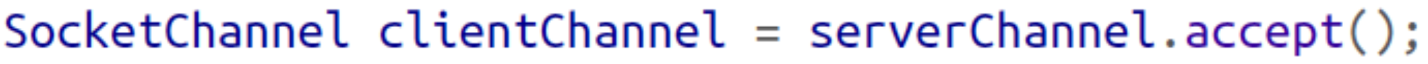
retrieve server socket and bind it to a port

alternatively



```
serverChannel.bind(new InetSocketAddress(19));
```

- Accepting request

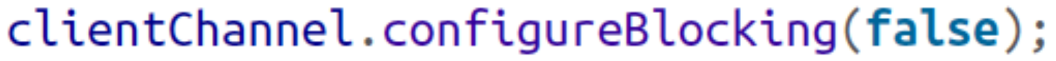


```
SocketChannel clientChannel = serverChannel.accept();
```

this line will be placed differently from the case of blocking mode

Activating Nonblocking Mode

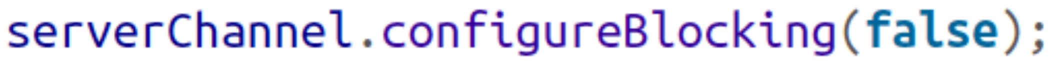
- Client channel



```
clientChannel.configureBlocking(false);
```

  - for writing data to client
  - write() returns when client channel is not ready to be written

- Server channel

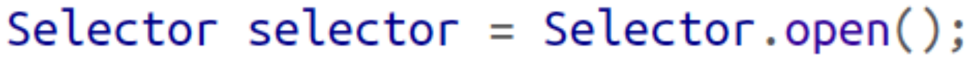


```
serverChannel.configureBlocking(false);
```

  - for accepting request from client
  - accept() returns null when there is no request
  - better to check if returned socket channel is null or not

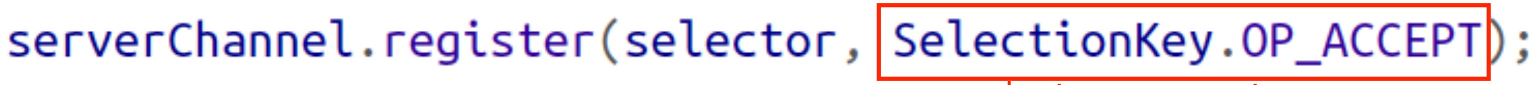
Selector

- Enable program to iterate over all connections that are ready to be processed



```
Selector selector = Selector.open();
```

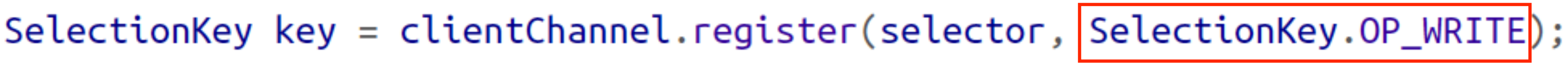
- Register each channel with selector (that monitors it)



```
serverChannel.register(selector, SelectionKey.OP_ACCEPT);
```

ready to accept new connection?

operation you are interested in



```
SelectionKey key = clientChannel.register(selector, SelectionKey.OP_WRITE);
```

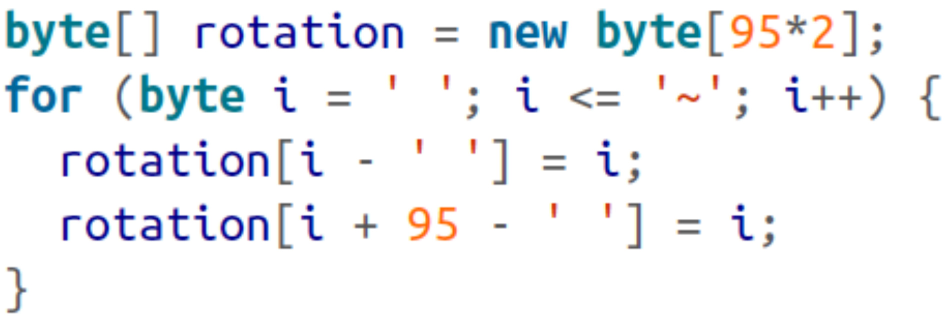
ready to write?

each SelectionKey has an attachment of arbitrary Object type

-

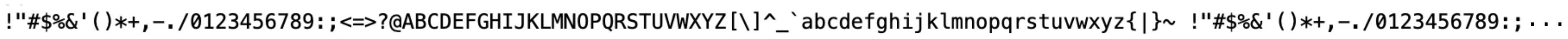
Constructing Buffers

- In this example, we will attach buffer that channel writes onto network
- Once buffer is fully drained, we will refill it (reused)
- Buffer containing two sequence copies of data



```
byte[] rotation = new byte[95*2];
for (byte i = ' '; i <= '~'; i++) {
  rotation[i - ' '] = i;
  rotation[i + 95 - ' '] = i;
}
```

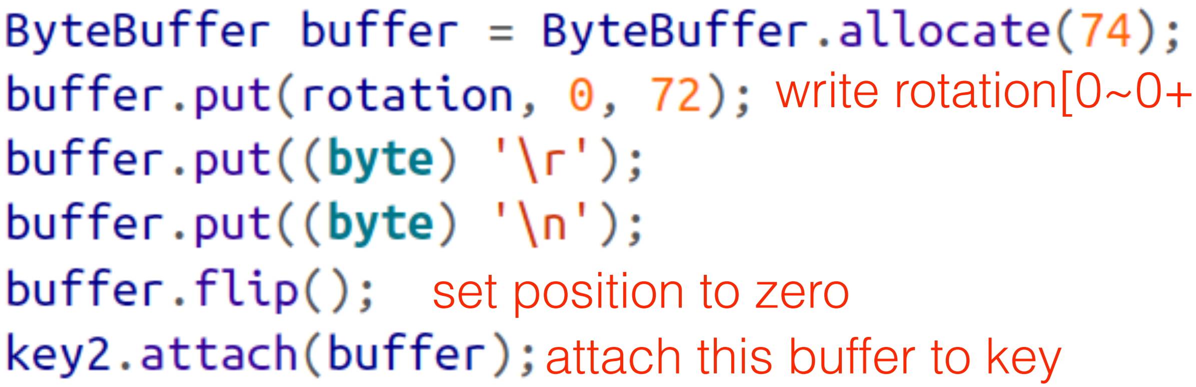
  - this will be used to construct buffer containing 72 characters
  - this form is convenient because what?



```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ !"#$%&'()*+,-./0123456789:;...
```

Constructing Buffers (contd.)

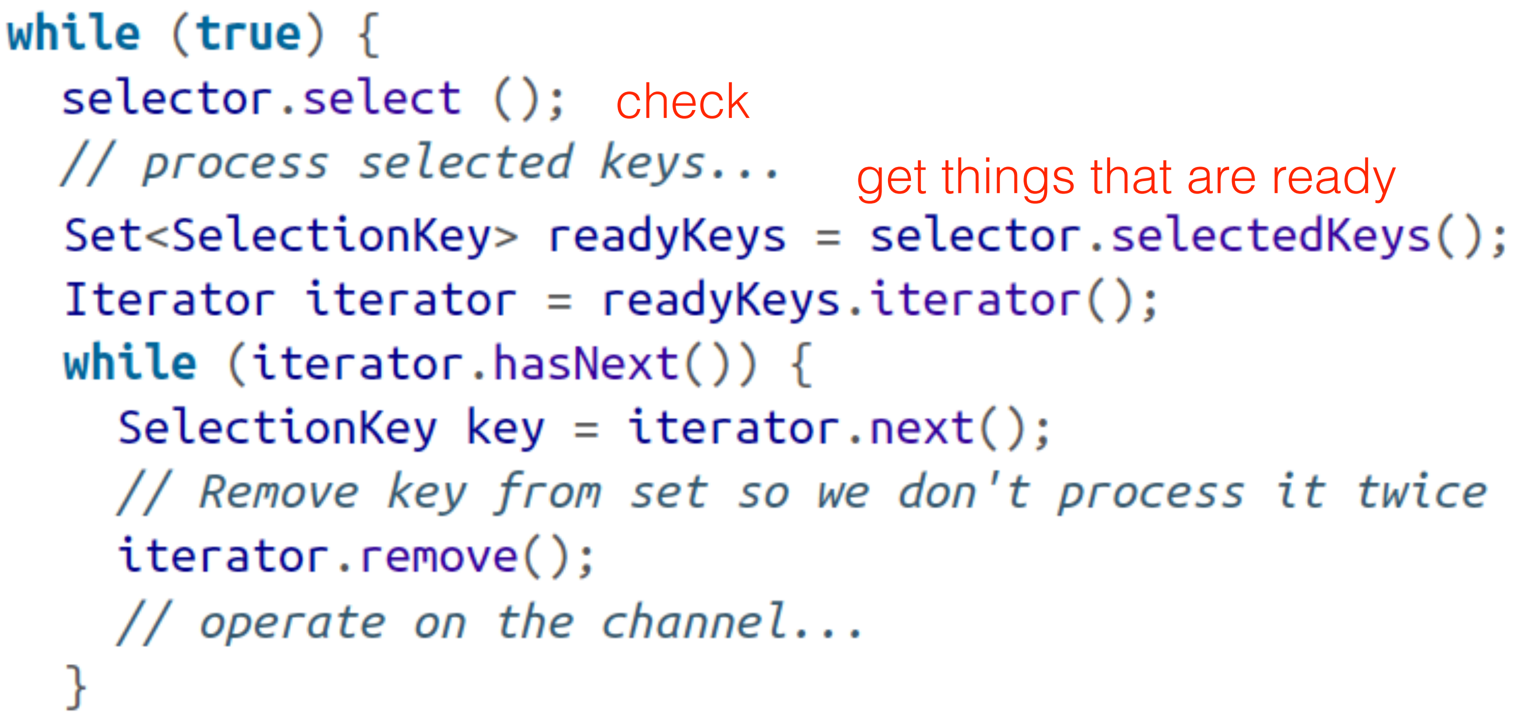
- Fill the buffer with 72 characters from rotation array and add \r\n



```
ByteBuffer buffer = ByteBuffer.allocate(74);
buffer.put(rotation, 0, 72); write rotation[0~0+71] into buffer
buffer.put((byte) '\r');
buffer.put((byte) '\n');
buffer.flip();    set position to zero
key2.attach(buffer);attach this buffer to key
```

Checking Readiness

- Check whether anything is ready to be acted on



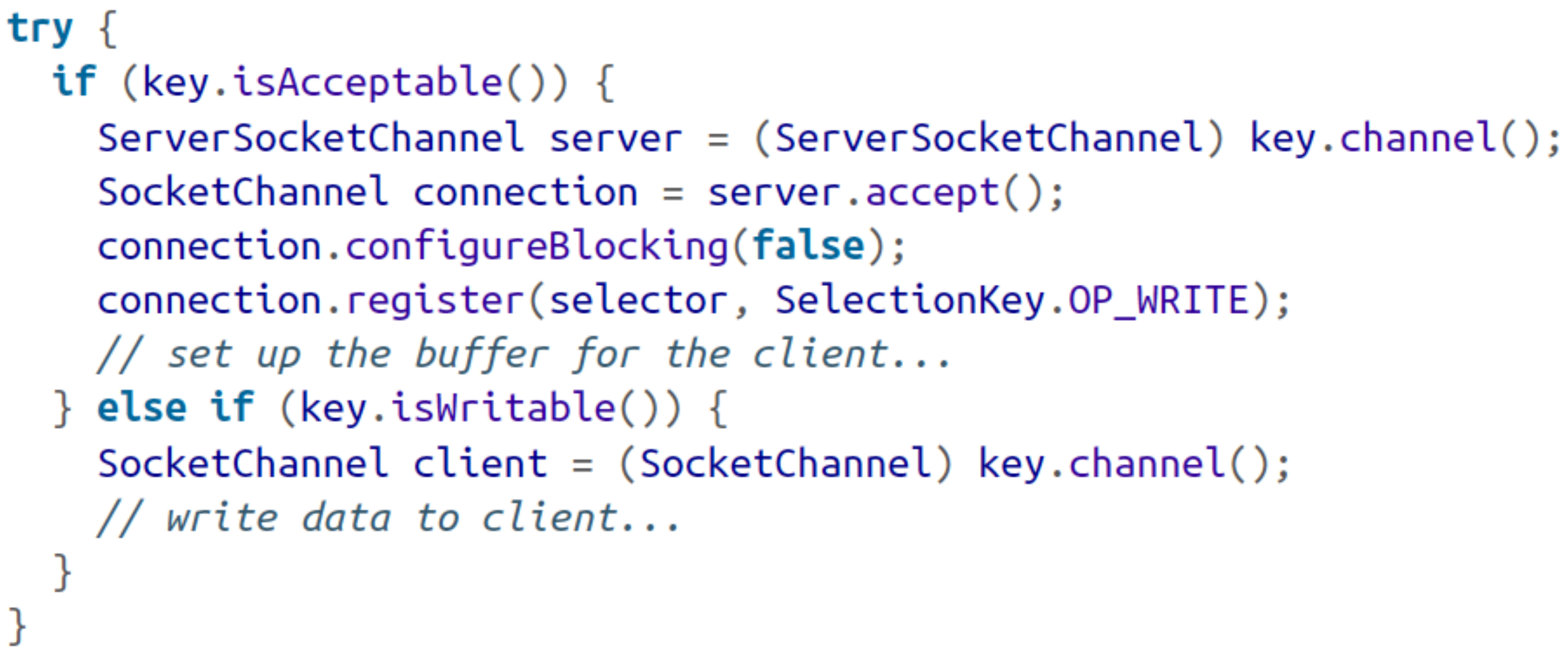
```
while (true) {
  selector.select ();
  // process selected keys...
  Set<SelectionKey> readyKeys = selector.selectedKeys();
  Iterator iterator = readyKeys.iterator();
  while (iterator.hasNext()) {
    SelectionKey key = iterator.next();
    // Remove key from set so we don't process it twice
    iterator.remove();
    // operate on the channel...
  }
```

check

get things that are ready

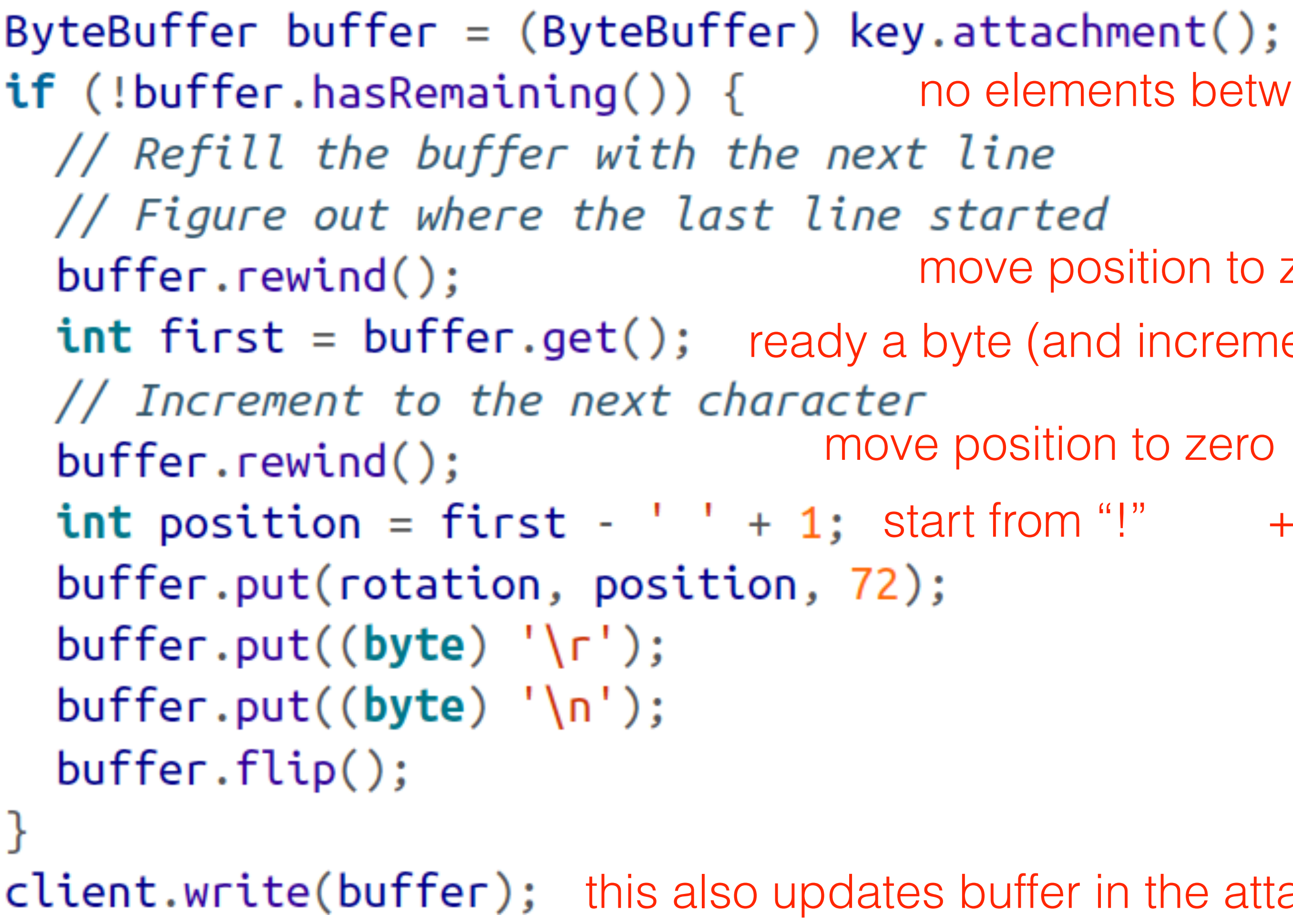
- If ready channel is server channel, program accepts new socket channel and add it to selector
- If ready channel is socket channel, program writes as much of the buffer as it can onto the channel
- If no channels are ready, selector waits for one
- One thread(main thread) processes multiple simultaneous connections

Checking Operation Types



```java
try {
  if (key.isAcceptable()) {
    ServerSocketChannel server = (ServerSocketChannel) key.channel();
    SocketChannel connection = server.accept();
    connection.configureBlocking(false);
    connection.register(selector, SelectionKey.OP_WRITE);
    // set up the buffer for the client...
  } else if (key.isWritable()) {
    SocketChannel client = (SocketChannel) key.channel();
    // write data to client...
  }
}
```

Writing Data onto Channel



```java
ByteBuffer buffer = (ByteBuffer) key.attachment();
if (!buffer.hasRemaining()) {
  // Refill the buffer with the next line
  // Figure out where the last line started
  buffer.rewind();
  int first = buffer.get();
  // Increment to the next character
  buffer.rewind();
  int position = first - ' ' + 1;
  buffer.put(rotation, position, 72);
  buffer.put((byte) '\r');
  buffer.put((byte) '\n');
  buffer.flip();
}
client.write(buffer);
```

get attachment

no elements between position and limit

move position to zero

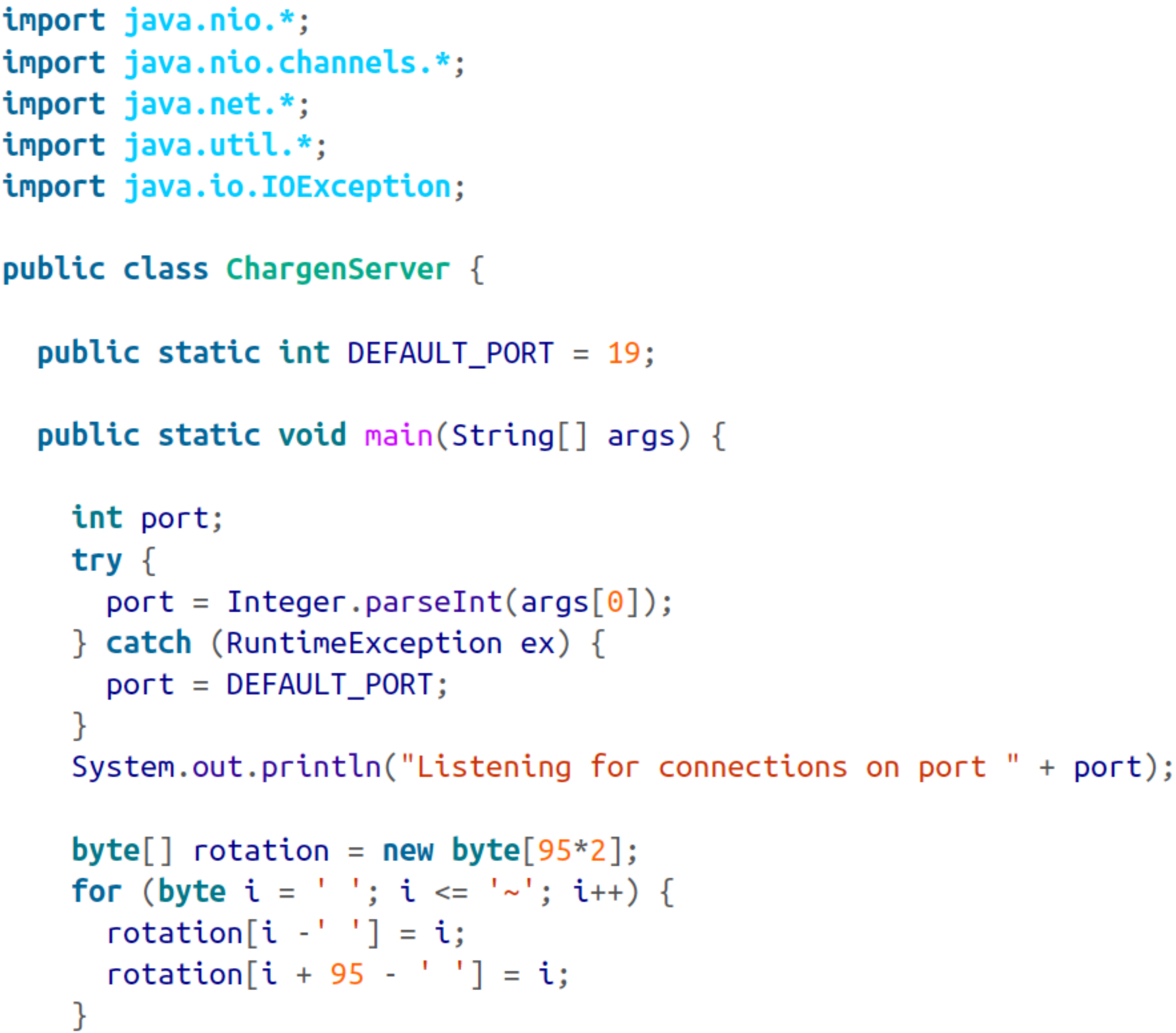
ready a byte (and increment position)

move position to zero

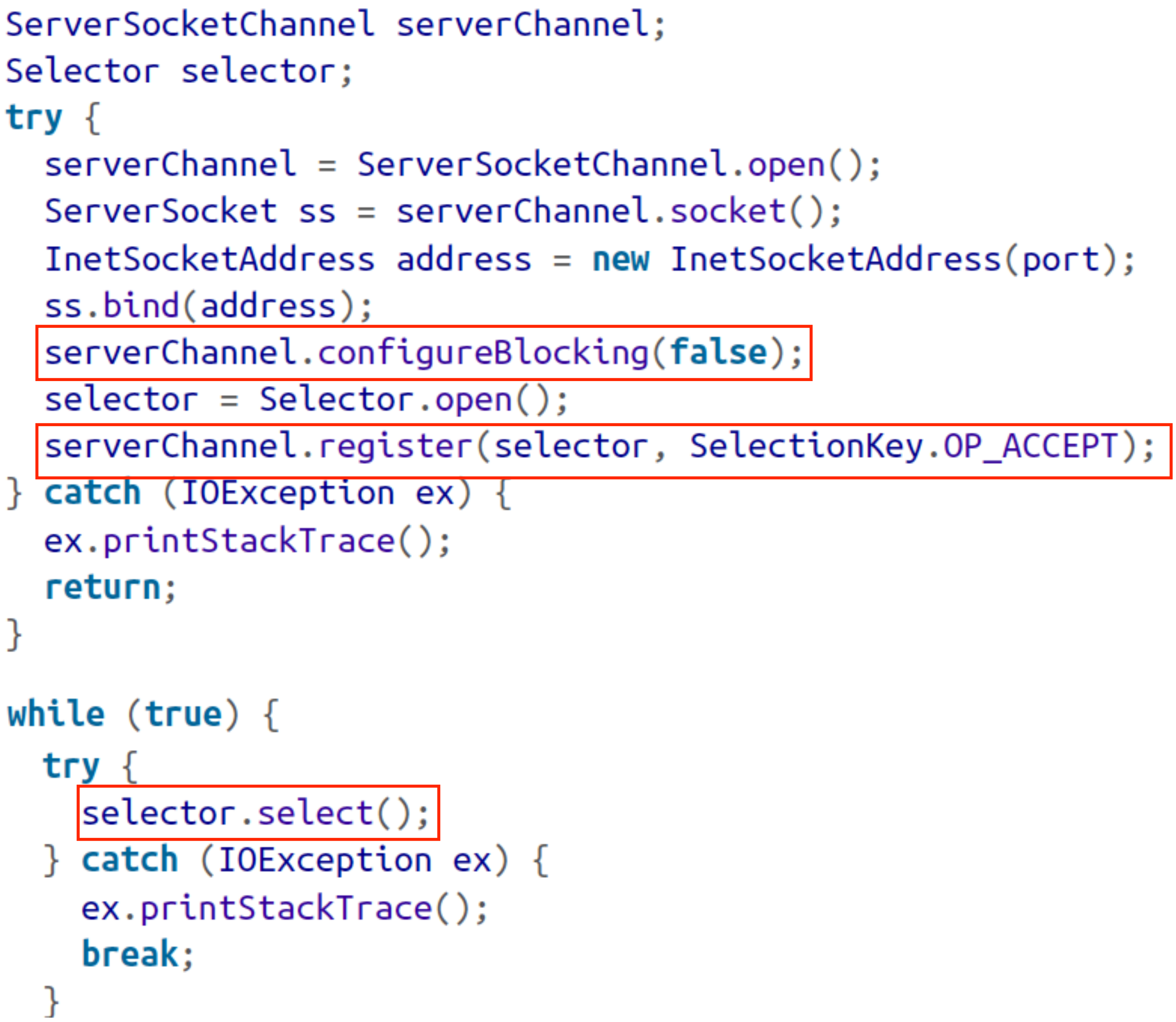
start from “!”

+1 makes it rotating

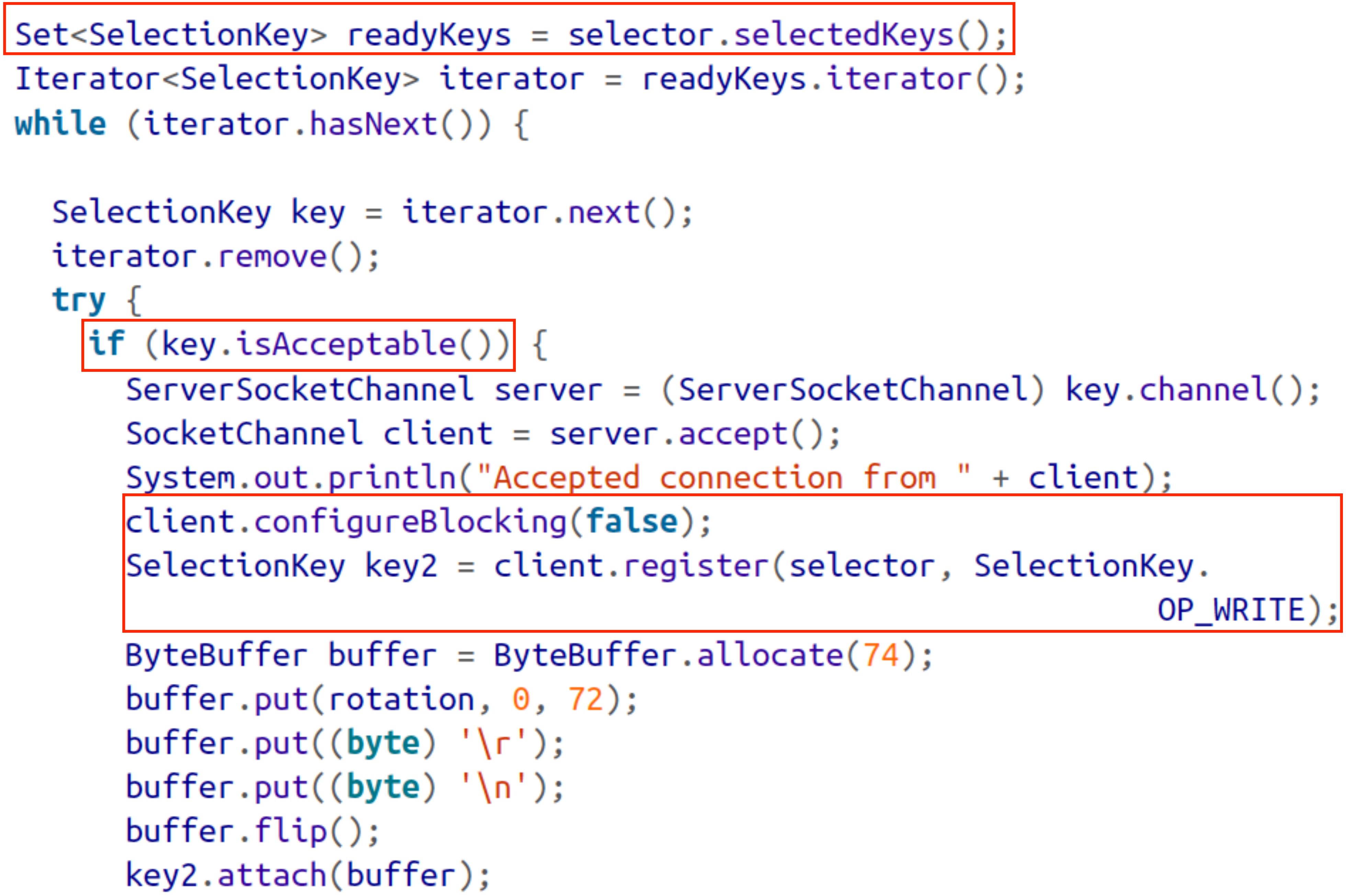
this also updates buffer in the attachment of key



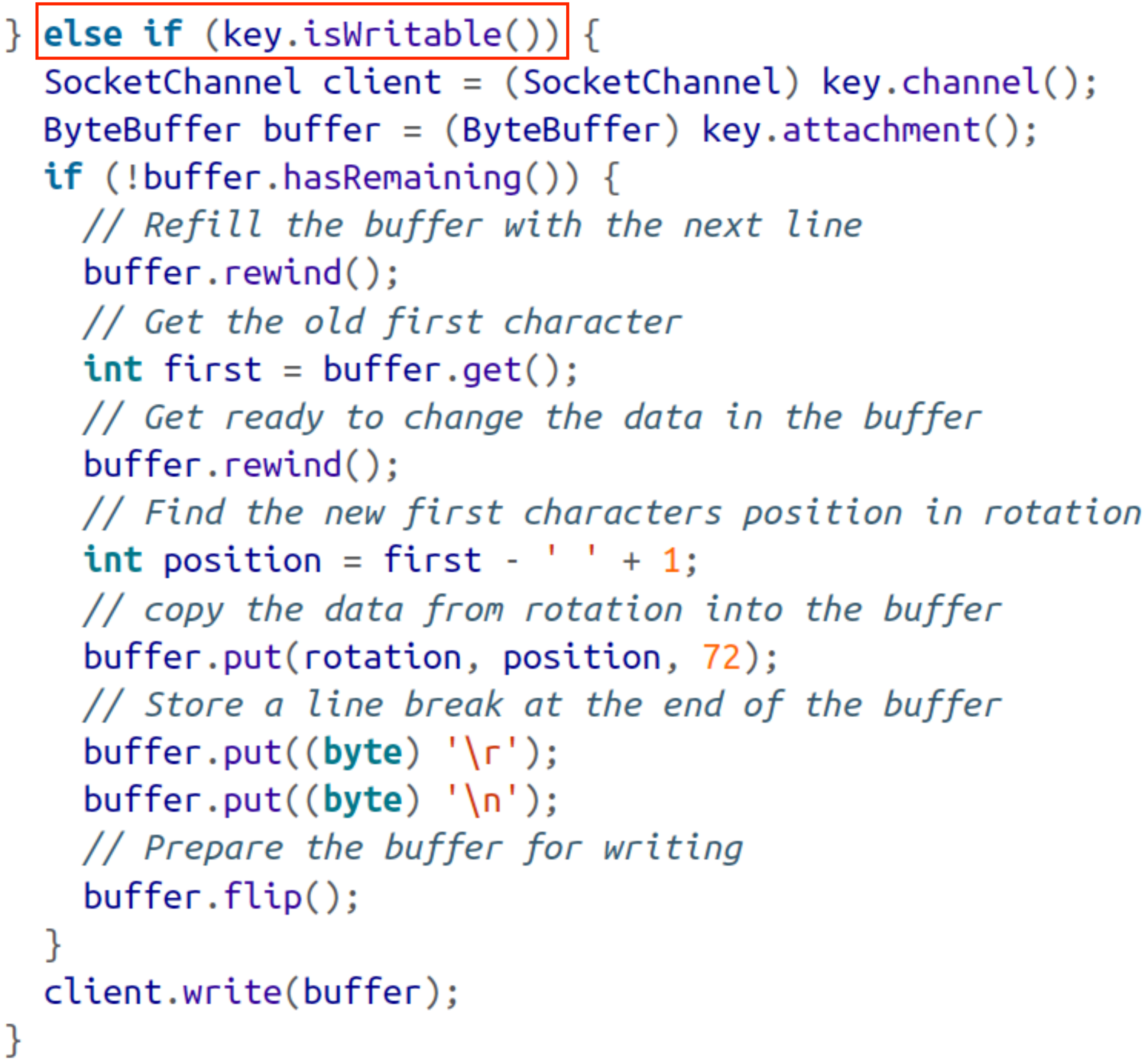
```java
import java.nio.*;
import java.nio.channels.*;
import java.net.*;
import java.util.*;
import java.io.IOException;

public class ChargenServer {

  public static int DEFAULT_PORT = 19;

  public static void main(String[] args) {

    int port;
    try {
      port = Integer.parseInt(args[0]);
    } catch (RuntimeException ex) {
      port = DEFAULT_PORT;
    }
    System.out.println("Listening for connections on port " + port);

    byte[] rotation = new byte[95*2];
    for (byte i = ' '; i <= '~'; i++) {
      rotation[i -' '] = i;
      rotation[i + 95 - ' '] = i;
    }
```



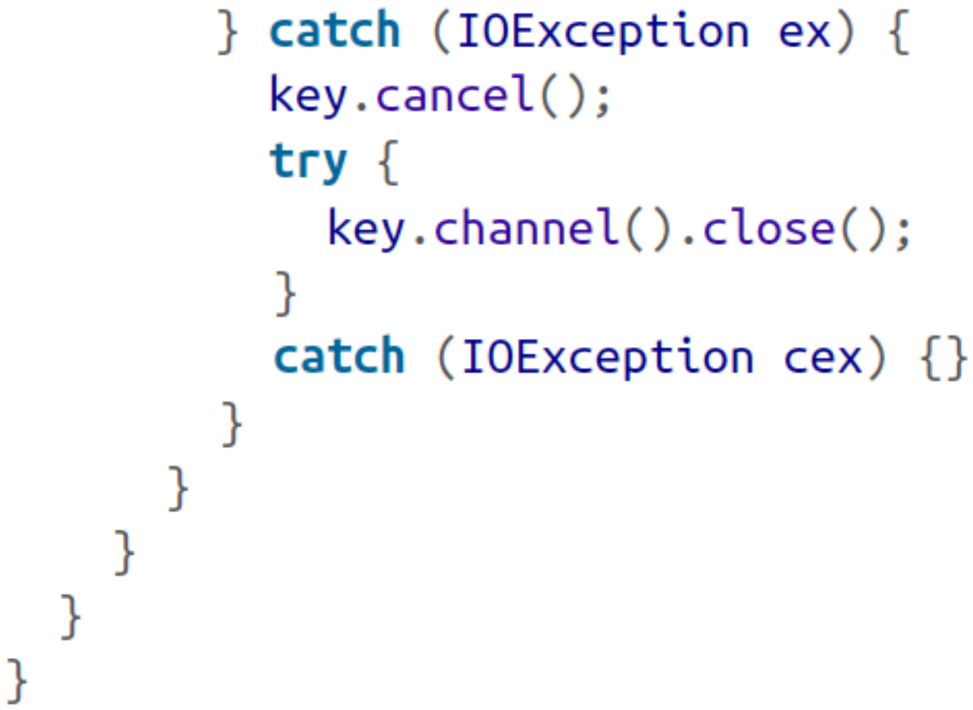
```
ServerSocketChannel serverChannel;
Selector selector;
try {
  serverChannel = ServerSocketChannel.open();
  ServerSocket ss = serverChannel.socket();
  InetSocketAddress address = new InetSocketAddress(port);
  ss.bind(address);
  serverChannel.configureBlocking(false);
  selector = Selector.open();
  serverChannel.register(selector, SelectionKey.OP_ACCEPT);
} catch (IOException ex) {
  ex.printStackTrace();
  return;
}

while (true) {
  try {
    selector.select();
  } catch (IOException ex) {
    ex.printStackTrace();
    break;
  }
```



```
Set<SelectionKey> readyKeys = selector.selectedKeys();
Iterator<SelectionKey> iterator = readyKeys.iterator();
while (iterator.hasNext()) {

  SelectionKey key = iterator.next();
  iterator.remove();
  try {
    if (key.isAcceptable()) {
      ServerSocketChannel server = (ServerSocketChannel) key.channel();
      SocketChannel client = server.accept();
      System.out.println("Accepted connection from " + client);
      client.configureBlocking(false);
      SelectionKey key2 = client.register(selector, SelectionKey.
                                                              OP_WRITE);
      ByteBuffer buffer = ByteBuffer.allocate(74);
      buffer.put(rotation, 0, 72);
      buffer.put((byte) '\r');
      buffer.put((byte) '\n');
      buffer.flip();
      key2.attach(buffer);
```



```
} else if (key.isWritable()) {
  SocketChannel client = (SocketChannel) key.channel();
  ByteBuffer buffer = (ByteBuffer) key.attachment();
  if (!buffer.hasRemaining()) {
    // Refill the buffer with the next line
    buffer.rewind();
    // Get the old first character
    int first = buffer.get();
    // Get ready to change the data in the buffer
    buffer.rewind();
    // Find the new first characters position in rotation
    int position = first - ' ' + 1;
    // copy the data from rotation into the buffer
    buffer.put(rotation, position, 72);
    // Store a line break at the end of the buffer
    buffer.put((byte) '\r');
    buffer.put((byte) '\n');
    // Prepare the buffer for writing
    buffer.flip();
  }
  client.write(buffer);
}
```



```
        } catch (IOException ex) {
          key.cancel();
          try {
            key.channel().close();
          }
          catch (IOException cex) {}
        }
      }
    }
  }
}
```

Notes

- Can extend this to the case of multiple threads
  - multiple CPUs can be exploited
- Can also extend to the case of thread pool
- select() ensures you are never wasting any time on connections that are not ready to receive data

Buffers

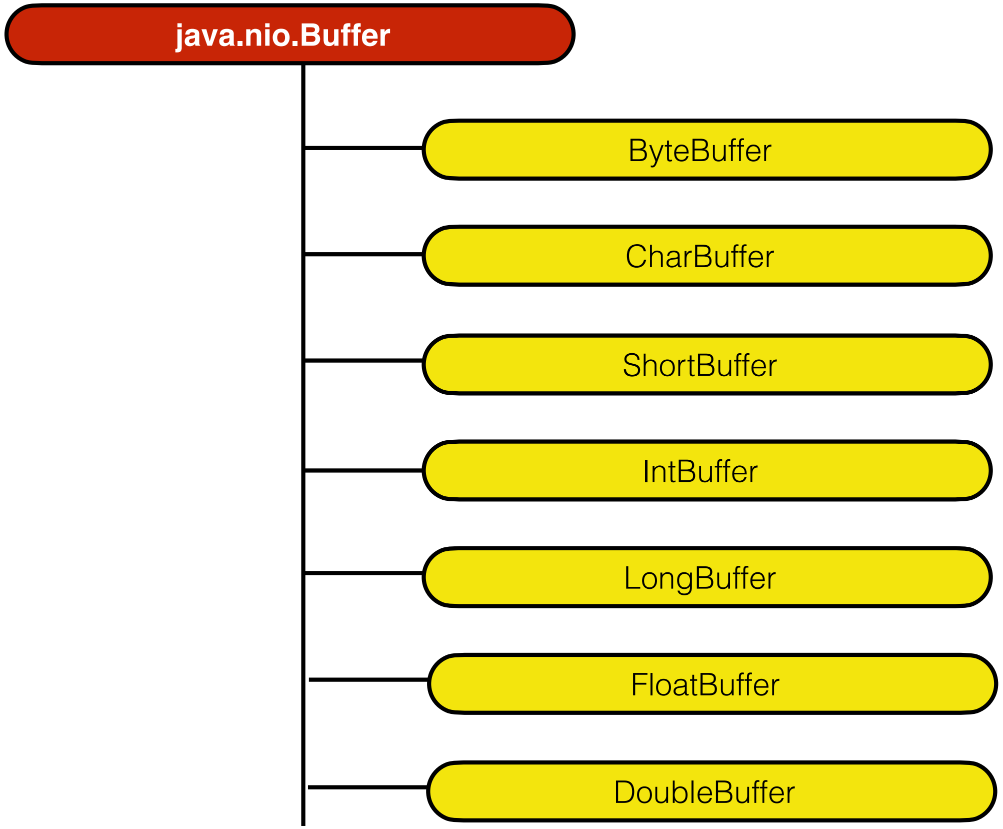
Streams vs. Channels

- Streams are byte-based
  - data is written or read in bytes
- Channels are block-based
  - data is written or read one buffer at a time

- Streams
  - read or write only
- Channels
  - can both read and write

Class Hierarchy

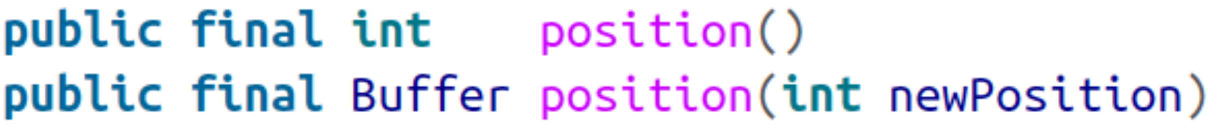
- Container for data of specific type



Network programs use ByteBuffer almost exclusively

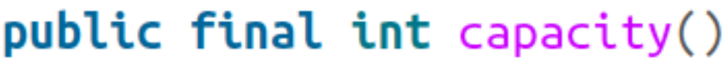
Four Properties

- position
  - next location in buffer that will be read from or written to
  - start counting at 0, and max value=size of buffer
  - methods



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

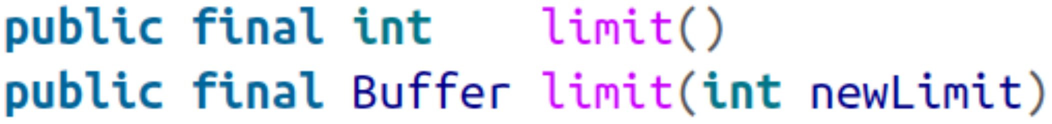
- capacity
  - max number of elements buffer can hold
  - set when buffer is created
  - method



```
public final int capacity()
```

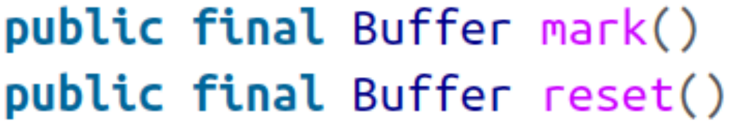
Four Properties (contd.)

- limit
  - end of accessible data
  - cannot write or read at or past this point



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

- mark
  - client-specified index



```
public final Buffer mark()
public final Buffer reset()
```

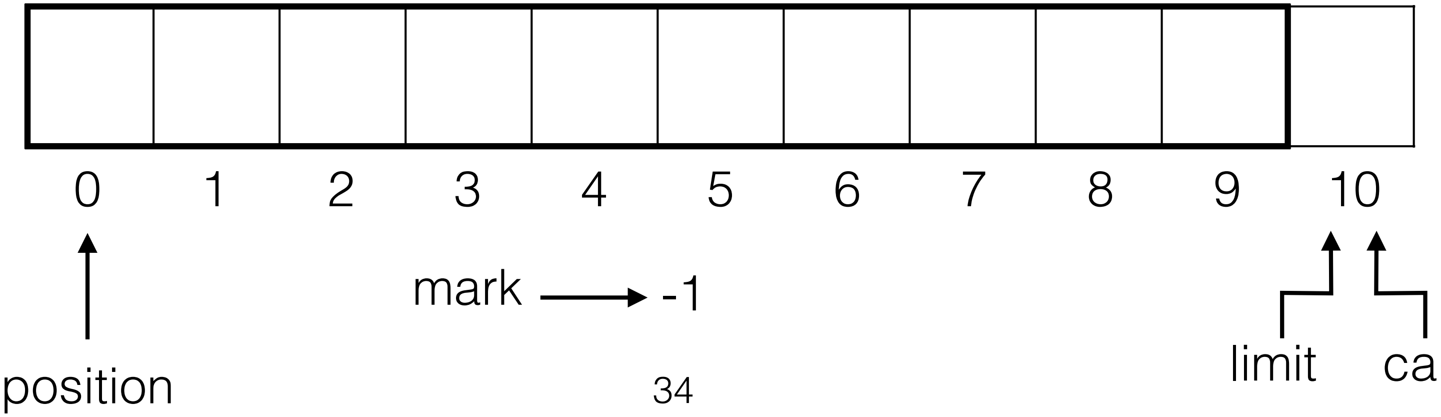
mark is set at current position

current position is set to marked position

  - if position is set below an existing mark, mark is discarded

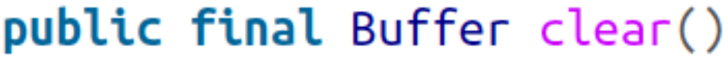
Initialization

- Position is set to zero and mark is undefined
- Each element of a newly-created buffer is initialized to zero
- Invariants
  -
- Exercise
  - create a ByteBuffer object with capacity 10, and check its initial properties



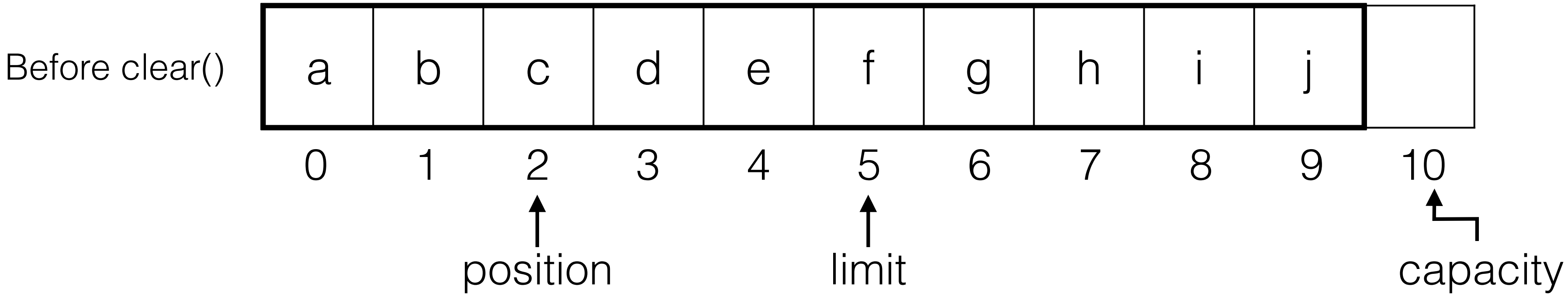
Clearing

- clear()



```
public final Buffer clear()
```

```
ByteBuffer buf = ByteBuffer.allocate(10);
...
buf.clear();
```



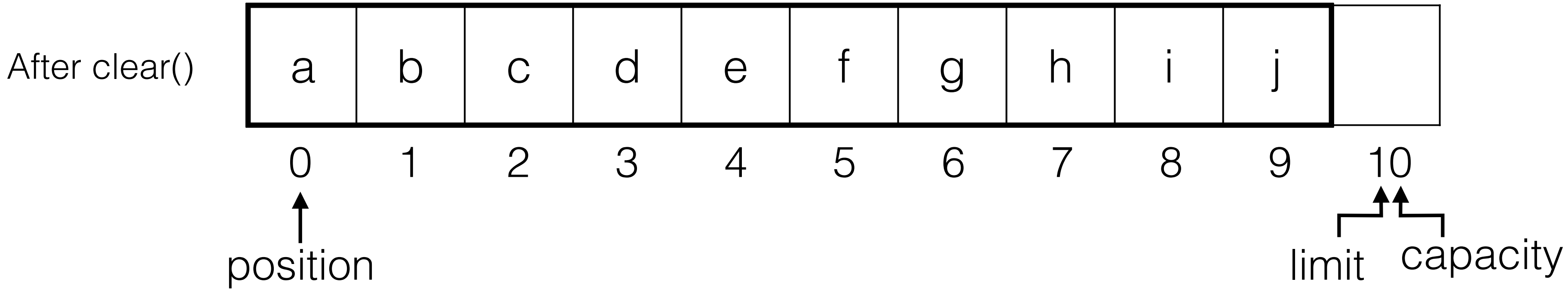
Before clear()

a	b	c	d	e	f	g	h	i	j	
0	1	2	3	4	5	6	7	8	9	10

position

limit

capacity



After clear()

a	b	c	d	e	f	g	h	i	j	
0	1	2	3	4	5	6	7	8	9	10

position

limit

capacity

- get ready for being written

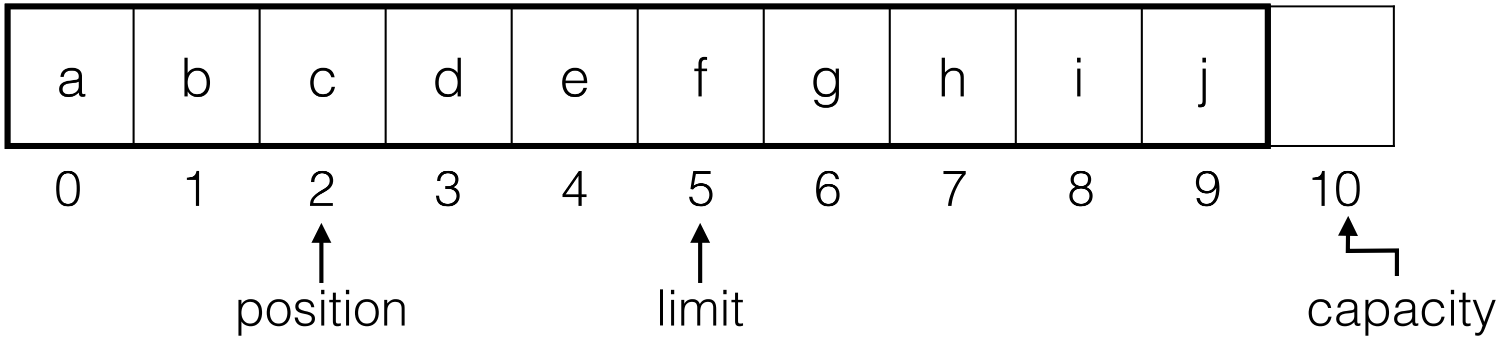
Rewinding

- rewind()

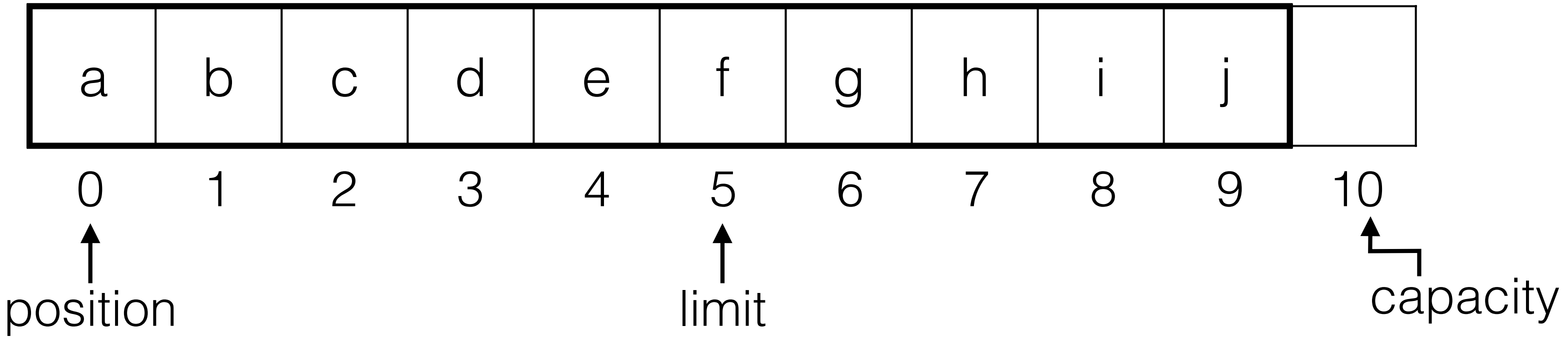
```
public final Buffer rewind()
```

```
ByteBuffer buf = ByteBuffer.allocate(10);
                         ...
buf.rewind();
```

Before rewind()



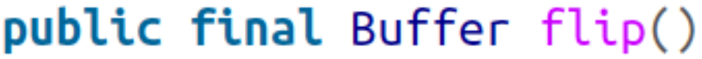
After rewind()



- allow the buffer to be reread

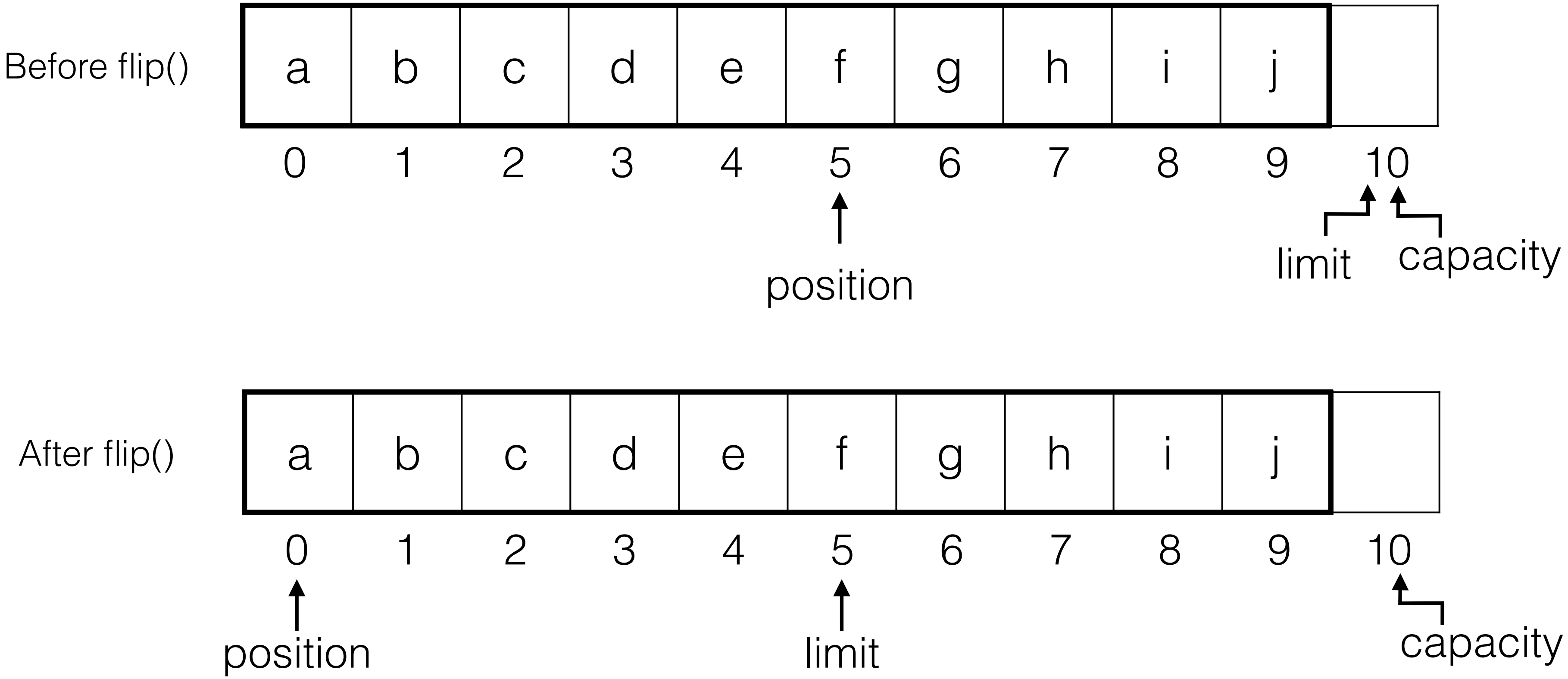
Flipping

- flip()



```
public final Buffer flip()
```

```
ByteBuffer buf = ByteBuffer.allocate(10);
                    ...
buf.flip();
```



Before flip()

a	b	c	d	e	f	g	h	i	j	
0	1	2	3	4	5	6	7	8	9	10

position

limit

capacity

After flip()

a	b	c	d	e	f	g	h	i	j	
0	1	2	3	4	5	6	7	8	9	10

position

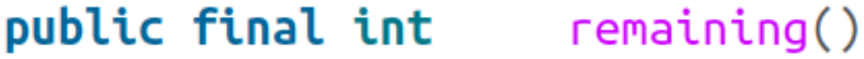
limit

capacity

- get ready for reading from the buffer just filled

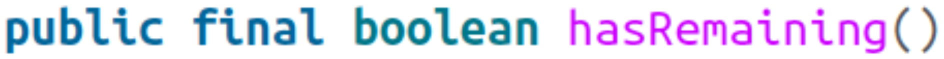
Info about Buffer

- Remaining elements



```
public final int      remaining()
```

number of elements between position and limit

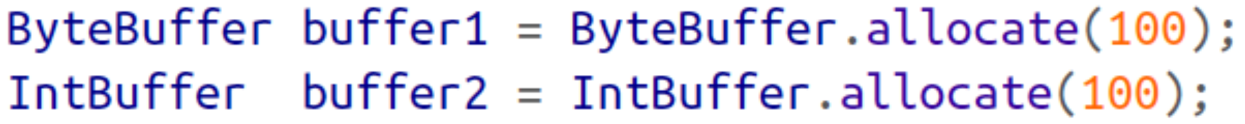


```
public final boolean hasRemaining()
```

true if remaining()>0

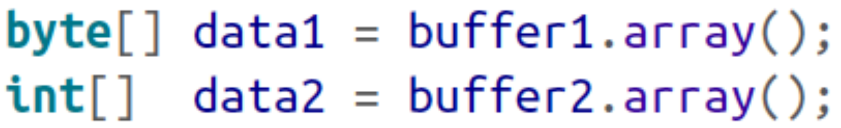
Creating Buffers: Allocation

- Allocation
  - allocate() returns a new, empty buffer with fixed specified capacity



```
ByteBuffer buffer1 = ByteBuffer.allocate(100);
IntBuffer  buffer2 = IntBuffer.allocate(100);
```

  - implemented on top of Java array and can be accessed by array() and arrayOffset() — return offset in the backing array



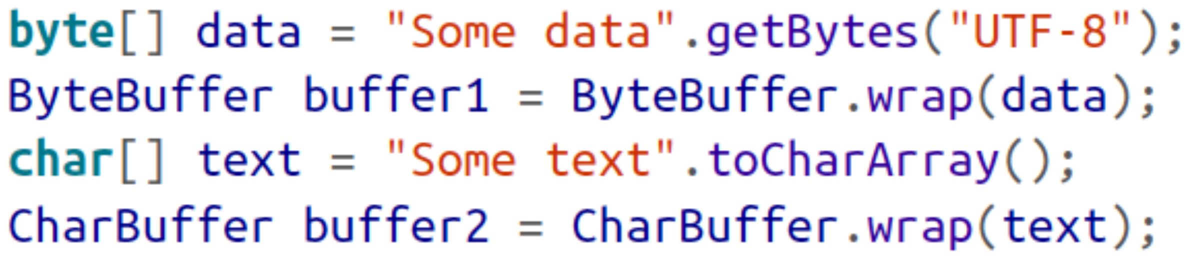
```
byte[] data1 = buffer1.array();
int[]  data2 = buffer2.array();
```

    - changes in data1,data2 are reflected in buffer1 and buffer2

- Exercise
  - check whether changes in int array can be seen in IntBuffer

Creating Buffer: Wrapping

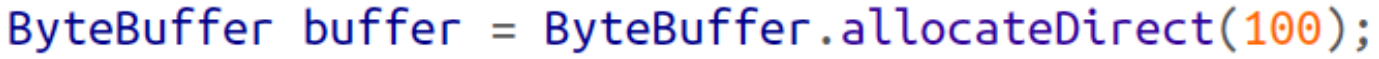
- In case you already have an array of data, you just need to wrap it to create a buffer



```java
byte[] data = "Some data".getBytes("UTF-8");
ByteBuffer buffer1 = ByteBuffer.wrap(data);
char[] text = "Some text".toCharArray();
CharBuffer buffer2 = CharBuffer.wrap(text);
```

Creating Buffers: Direct Allocation

- ByteBuffer class has an additional method allocateDirect()
    - don’t create a backing array



```
ByteBuffer buffer = ByteBuffer.allocateDirect(100);
```

    - direct memory access to the buffer on an Ethernet card, kernel memory, or something else

- Exercise
    - try to call array() and arrayOffset()

Filling and Draining

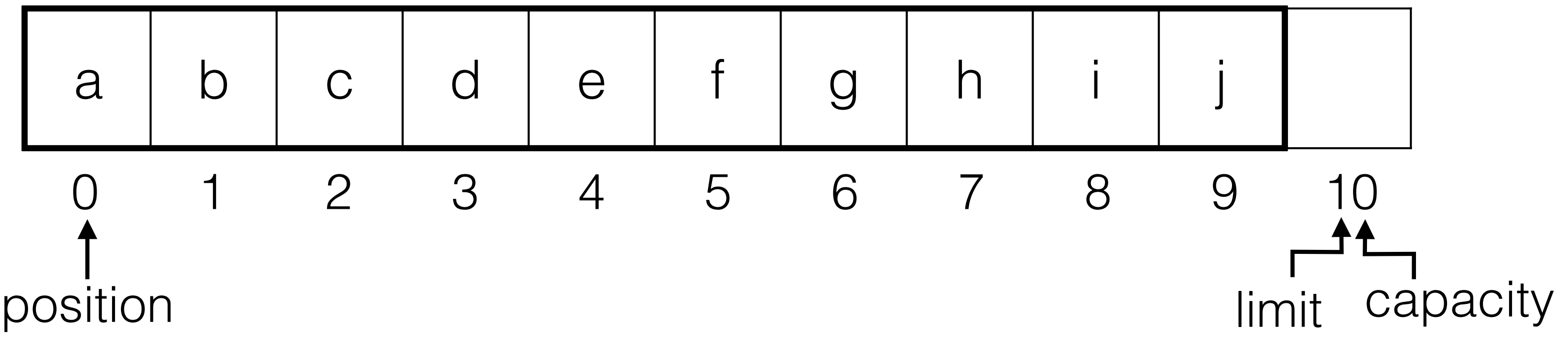
- based on current position
  - public abstract byte get()
  - public abstract ByteBuffer put(byte b)
- based on specified position
  - public abstract byte get(int index)
  - public abstract ByteBuffer put(int index, byte b)
- using array
  - public ByteBuffer get(byte[] dst)
  - public ByteBuffer get(byte[] dst, int offset, int length)
  - public final ByteBuffer put(byte[] src)
  - public ByteBuffer put (byte[] src, int offset, int length)
- buffer as an input argument
  - public ByteBuffer put (ByteBuffer src)

get() method

- read 1 byte and increment position

```
ByteBuffer buf = ByteBuffer.allocate(10);
...
byte b = buf.get();
```

Before get()



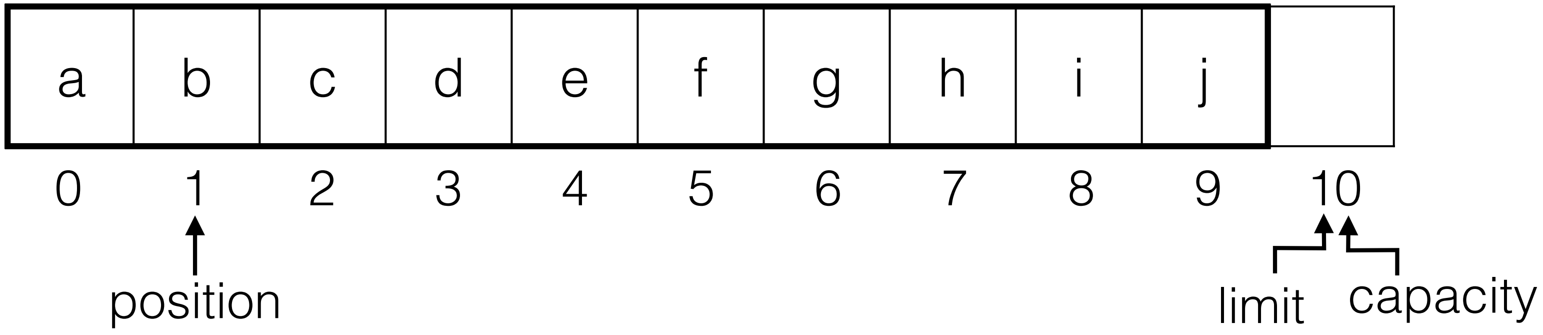
a b c d e f g h i j

0 1 2 3 4 5 6 7 8 9 10

position

limit capacity

After get()



a b c d e f g h i j

0 1 2 3 4 5 6 7 8 9 10

position

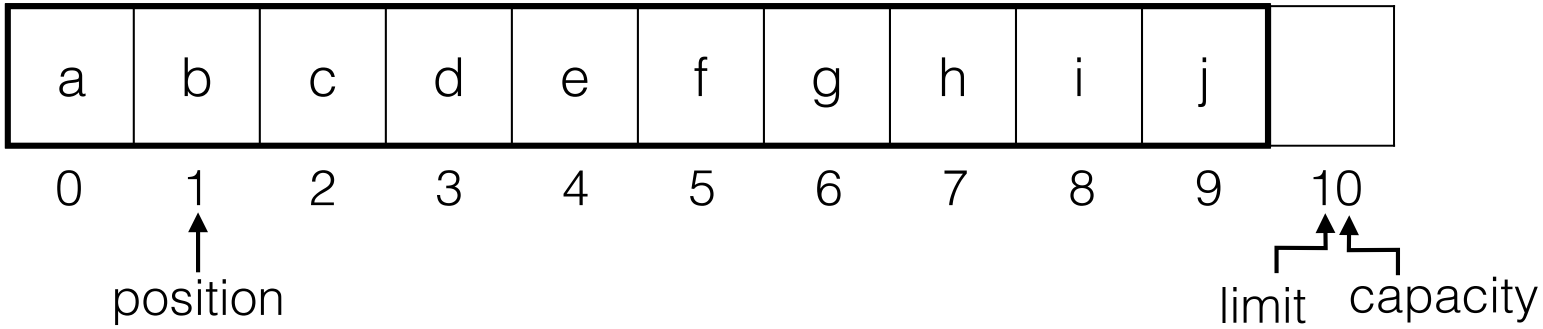
limit capacity

put() method

- write 1 byte and increment position

```
ByteBuffer buf = ByteBuffer.allocate(10);
...
byte b = buf.put((byte) z);
```

Before put()



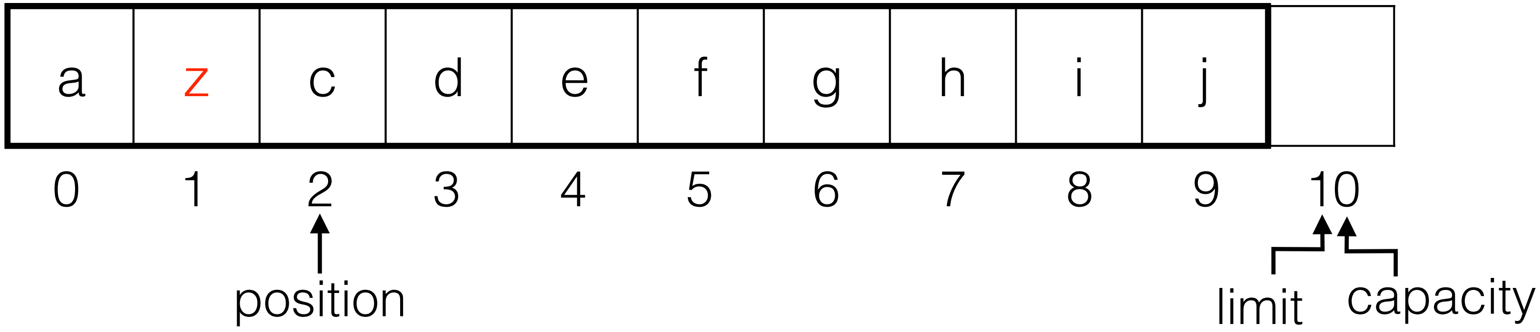
a b c d e f g h i j

0 1 2 3 4 5 6 7 8 9 10

position

limit capacity

After put()



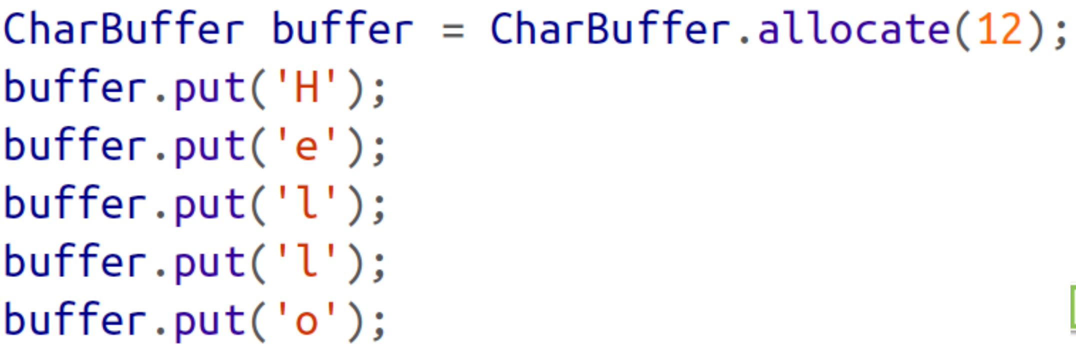
a z c d e f g h i j

0 1 2 3 4 5 6 7 8 9 10

position

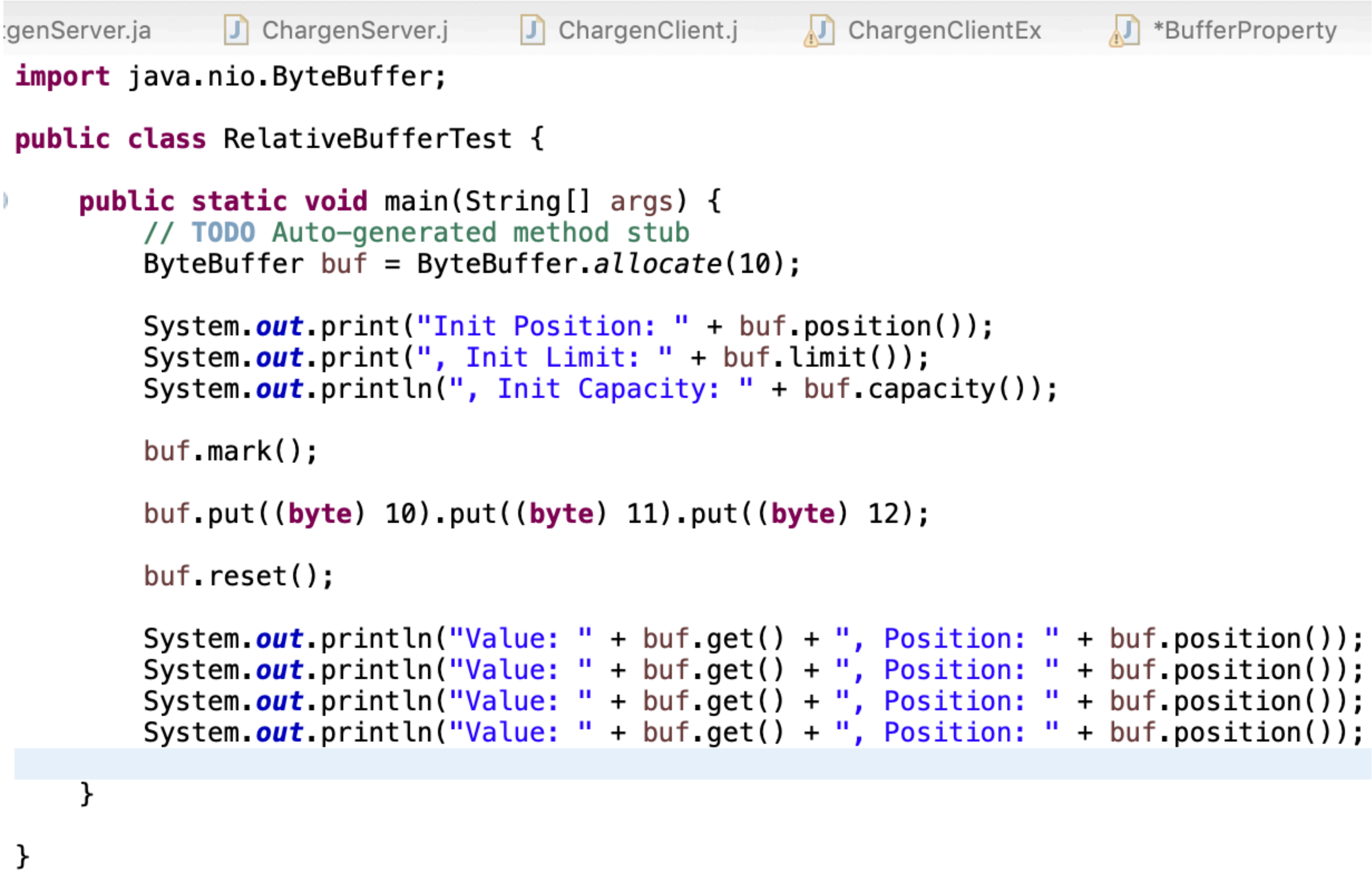
limit capacity

- Exercise
  - what's the position?



```
CharBuffer buffer = CharBuffer.allocate(12);
buffer.put('H');
buffer.put('e');
buffer.put('l');
buffer.put('l');
buffer.put('o');
```

Example

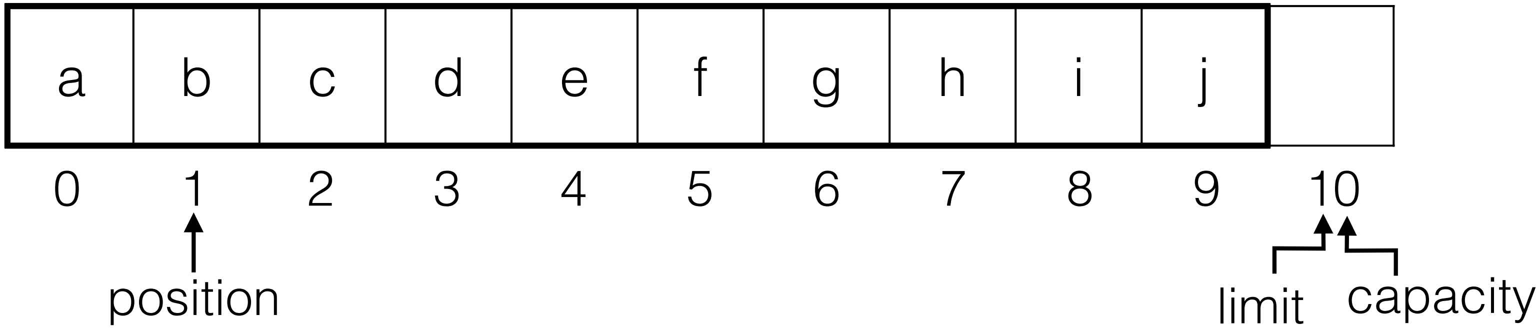


```java
import java.nio.ByteBuffer;

public class RelativeBufferTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ByteBuffer buf = ByteBuffer.allocate(10);

        System.out.print("Init Position: " + buf.position());
        System.out.print(", Init Limit: " + buf.limit());
        System.out.println(", Init Capacity: " + buf.capacity());

        buf.mark();

        buf.put((byte) 10).put((byte) 11).put((byte) 12);

        buf.reset();

        System.out.println("Value: " + buf.get() + ", Position: " + buf.position());
        System.out.println("Value: " + buf.get() + ", Position: " + buf.position());
        System.out.println("Value: " + buf.get() + ", Position: " + buf.position());
        System.out.println("Value: " + buf.get() + ", Position: " + buf.position());

    }

}
```

get(int index)

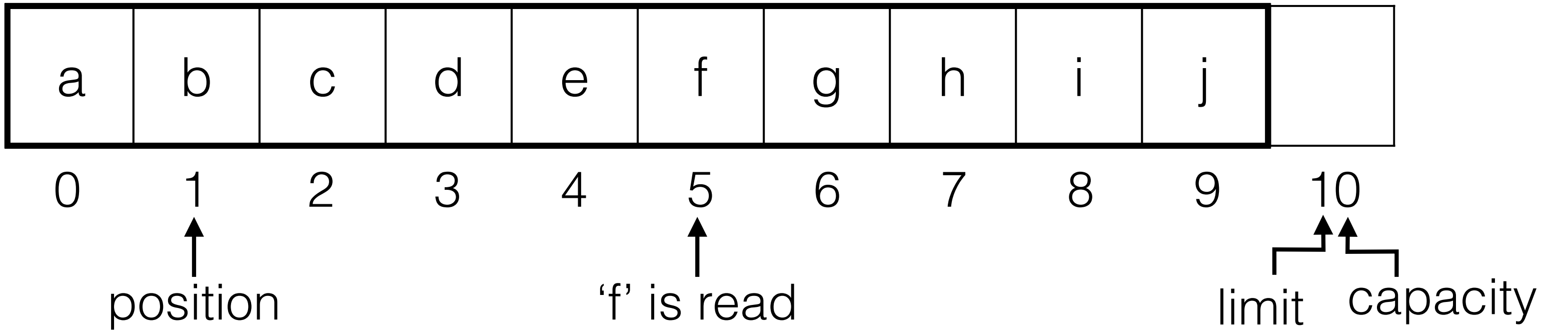
```
ByteBuffer buf = ByteBuffer.allocate(10);
                       ...
byte b = buf.get(5);
```

Before get()



a	b	c	d	e	f	g	h	i	j	
0	1	2	3	4	5	6	7	8	9	10

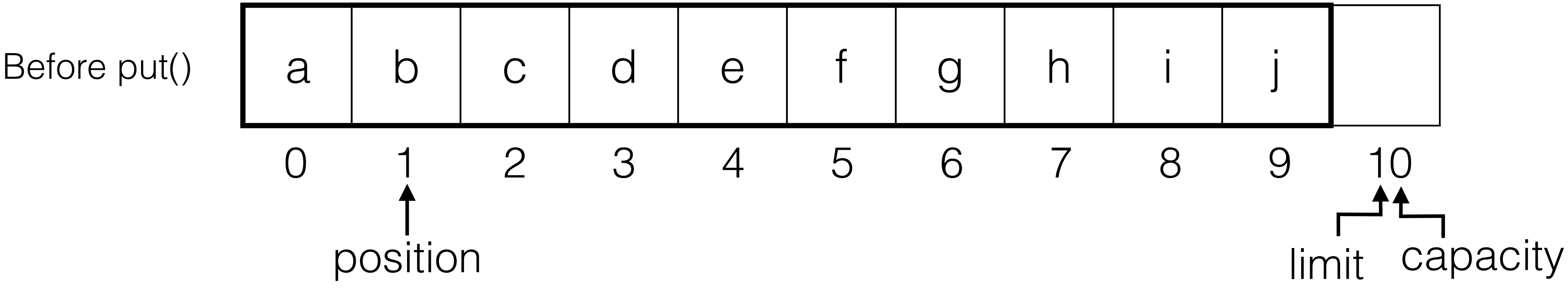
After get()

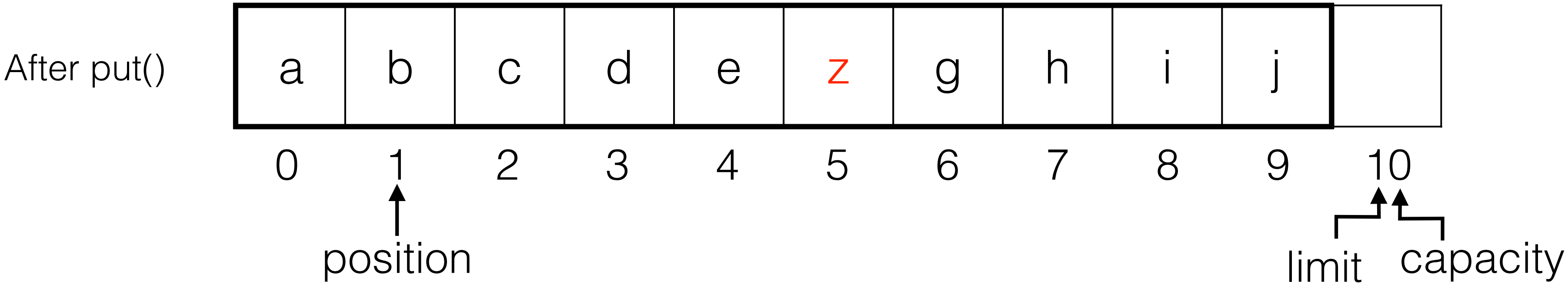


a	b	c	d	e	f	g	h	i	j	
0	1	2	3	4	5	6	7	8	9	10

put(int index, byte b)

```
ByteBuffer buf = ByteBuffer.allocate(10);
                    ...
byte b = buf.put(5, (byte) z);
```

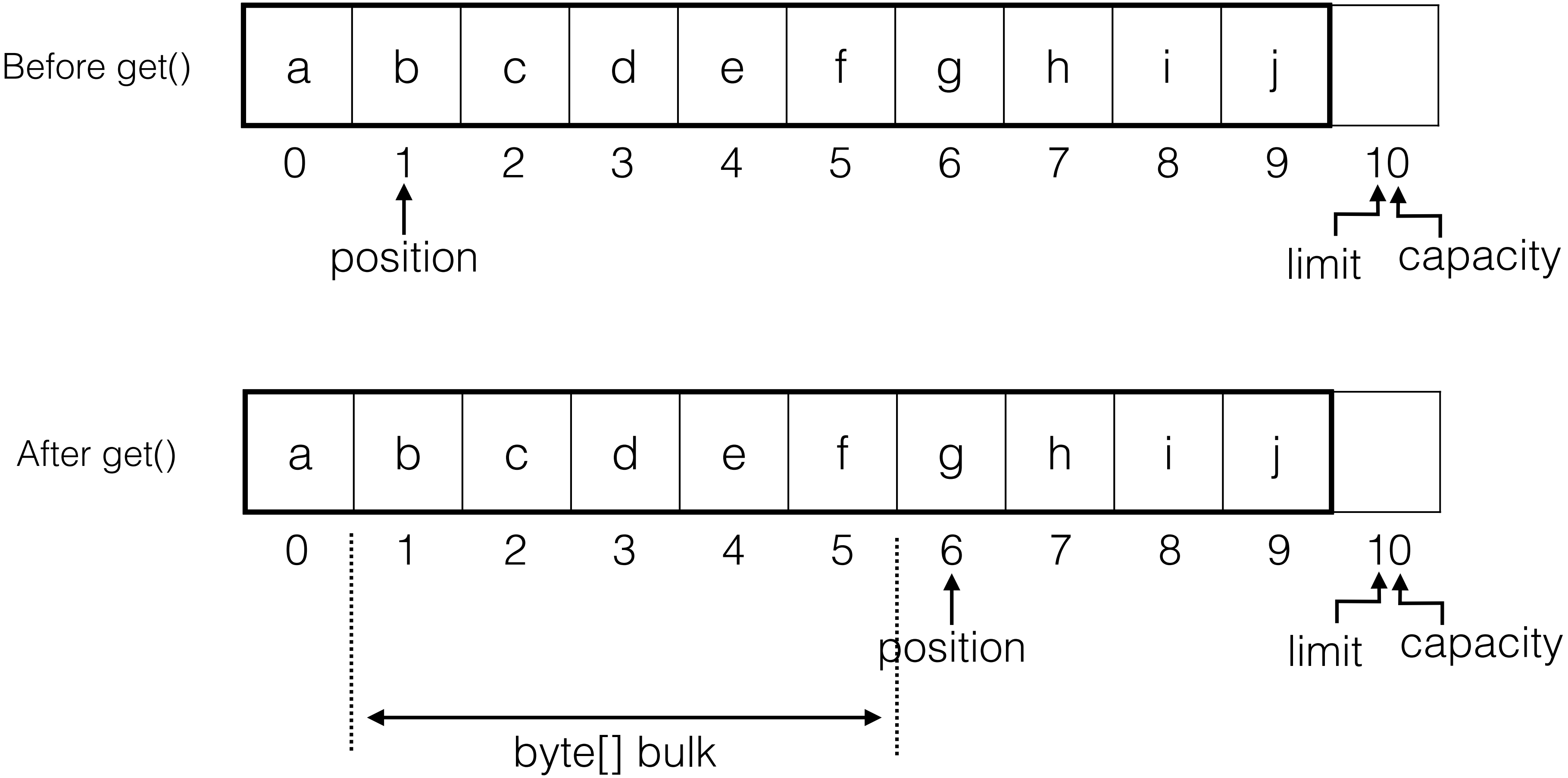




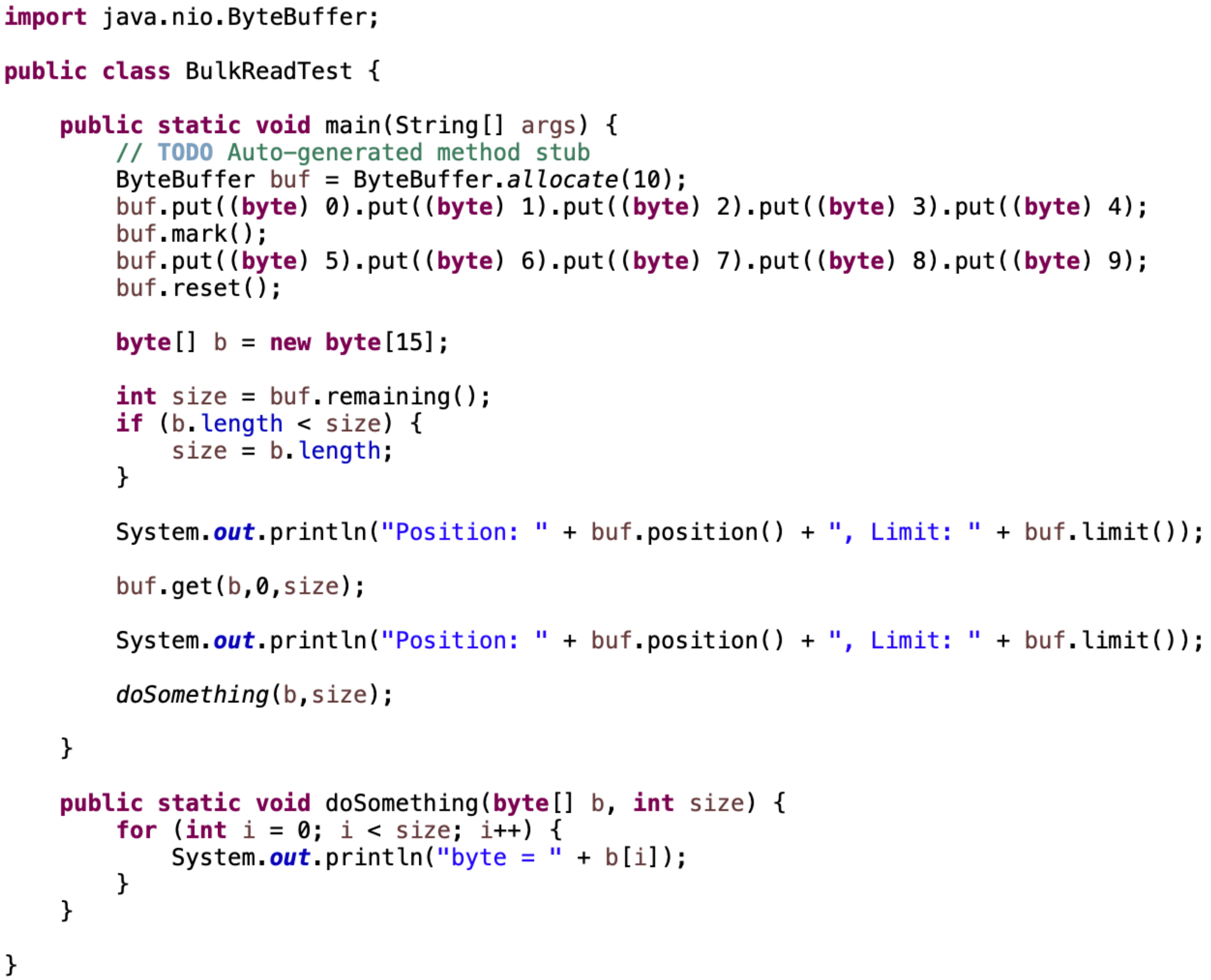
Bulk Methods

```
ByteBuffer buf = ByteBuffer.allocate(10);
byte[] bulk = new byte[5];
                    ...
byte b = buf.get(bulk);
```

get(bulk)==get(bulk, 0, bulk.length)



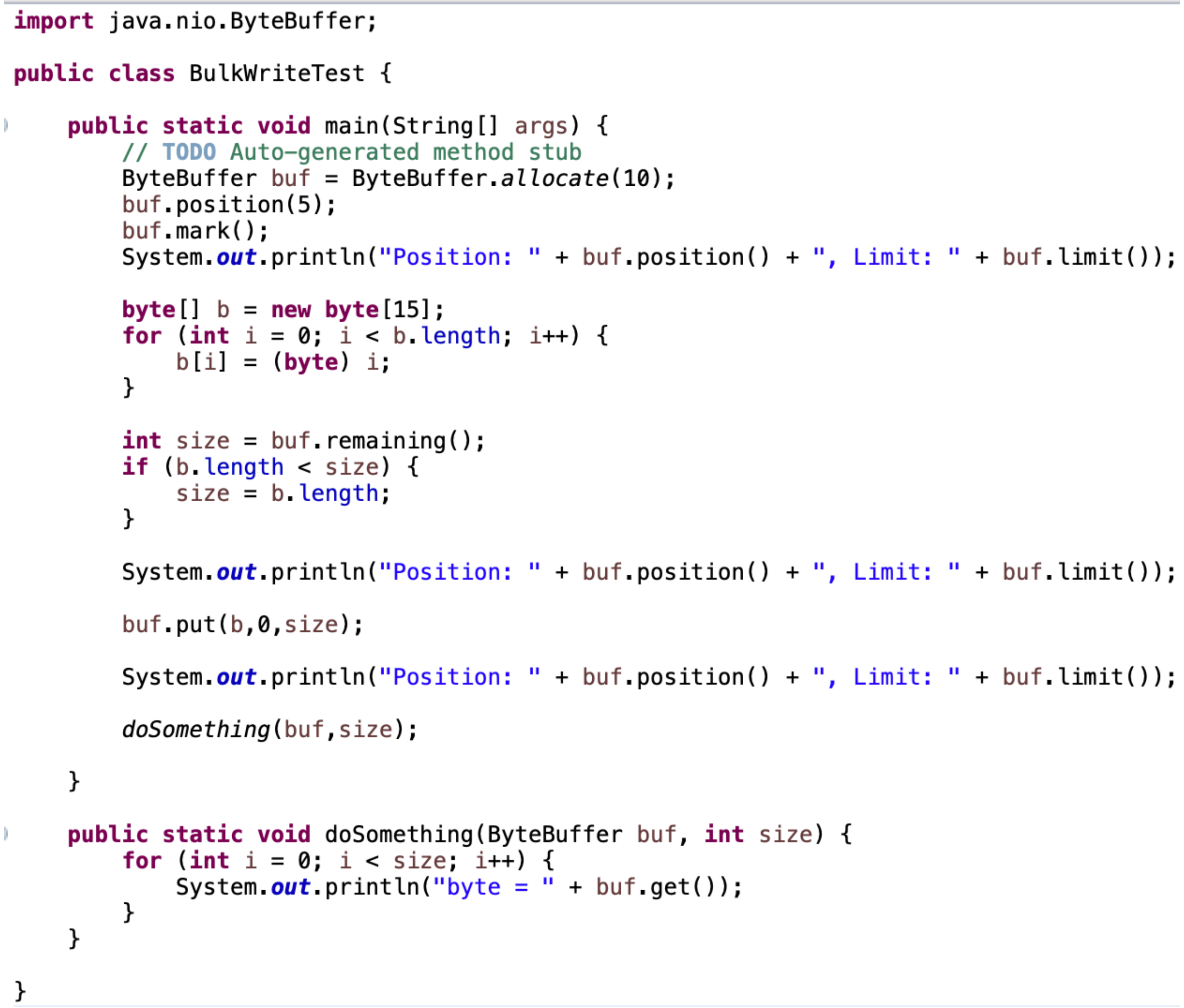
Example



```java
import java.nio.ByteBuffer;

public class BulkReadTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ByteBuffer buf = ByteBuffer.allocate(10);
        buf.put((byte) 0).put((byte) 1).put((byte) 2).put((byte) 3).put((byte) 4);
        buf.mark();
        buf.put((byte) 5).put((byte) 6).put((byte) 7).put((byte) 8).put((byte) 9);
        buf.reset();

        byte[] b = new byte[15];

        int size = buf.remaining();
        if (b.length < size) {
            size = b.length;
        }

        System.out.println("Position: " + buf.position() + ", Limit: " + buf.limit());

        buf.get(b,0,size);

        System.out.println("Position: " + buf.position() + ", Limit: " + buf.limit());

        doSomething(b,size);

    }

    public static void doSomething(byte[] b, int size) {
        for (int i = 0; i < size; i++) {
            System.out.println("byte = " + b[i]);
        }
    }

}
```

- Exercise
  - try to read more bytes than remaining bytes in buf

Example

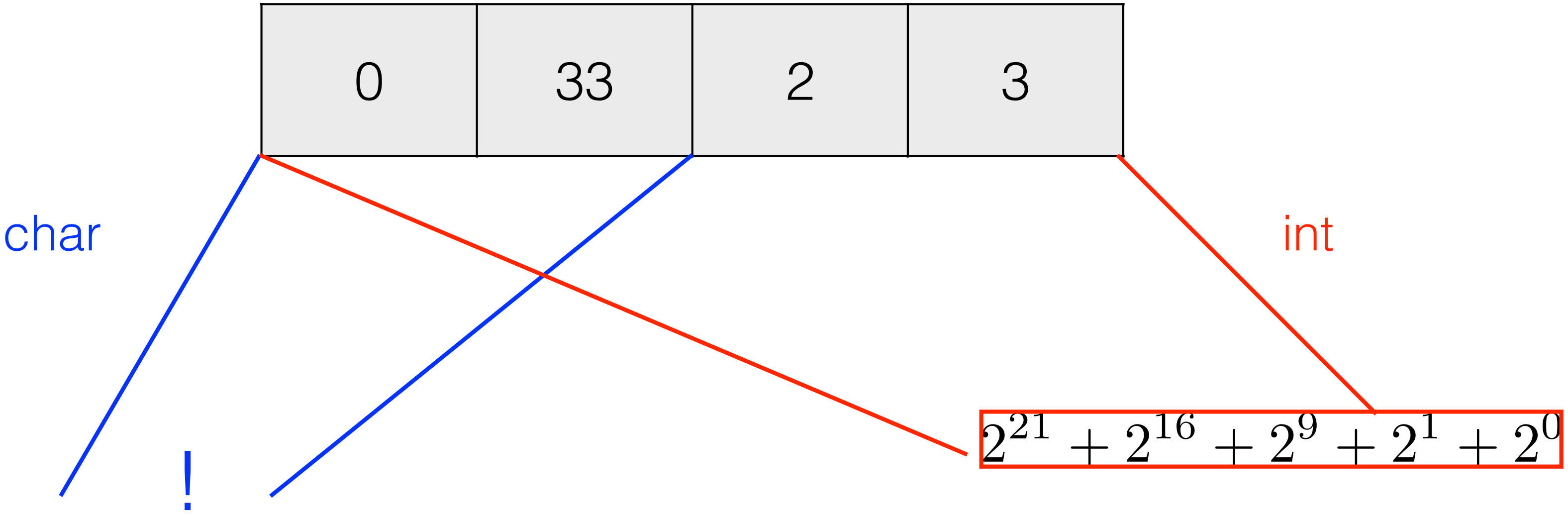


```
import java.nio.ByteBuffer;

public class BulkWriteTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ByteBuffer buf = ByteBuffer.allocate(10);
        buf.position(5);
        buf.mark();
        System.out.println("Position: " + buf.position() + ", Limit: " + buf.limit());

        byte[] b = new byte[15];
        for (int i = 0; i < b.length; i++) {
            b[i] = (byte) i;
        }

        int size = buf.remaining();
        if (b.length < size) {
            size = b.length;
        }

        System.out.println("Position: " + buf.position() + ", Limit: " + buf.limit());

        buf.put(b,0,size);

        System.out.println("Position: " + buf.position() + ", Limit: " + buf.limit());

        doSomething(buf,size);

    }

    public static void doSomething(ByteBuffer buf, int size) {
        for (int i = 0; i < size; i++) {
            System.out.println("byte = " + buf.get());
        }
    }

}
```

- Exercise
  - fix the BufferUnderflowException

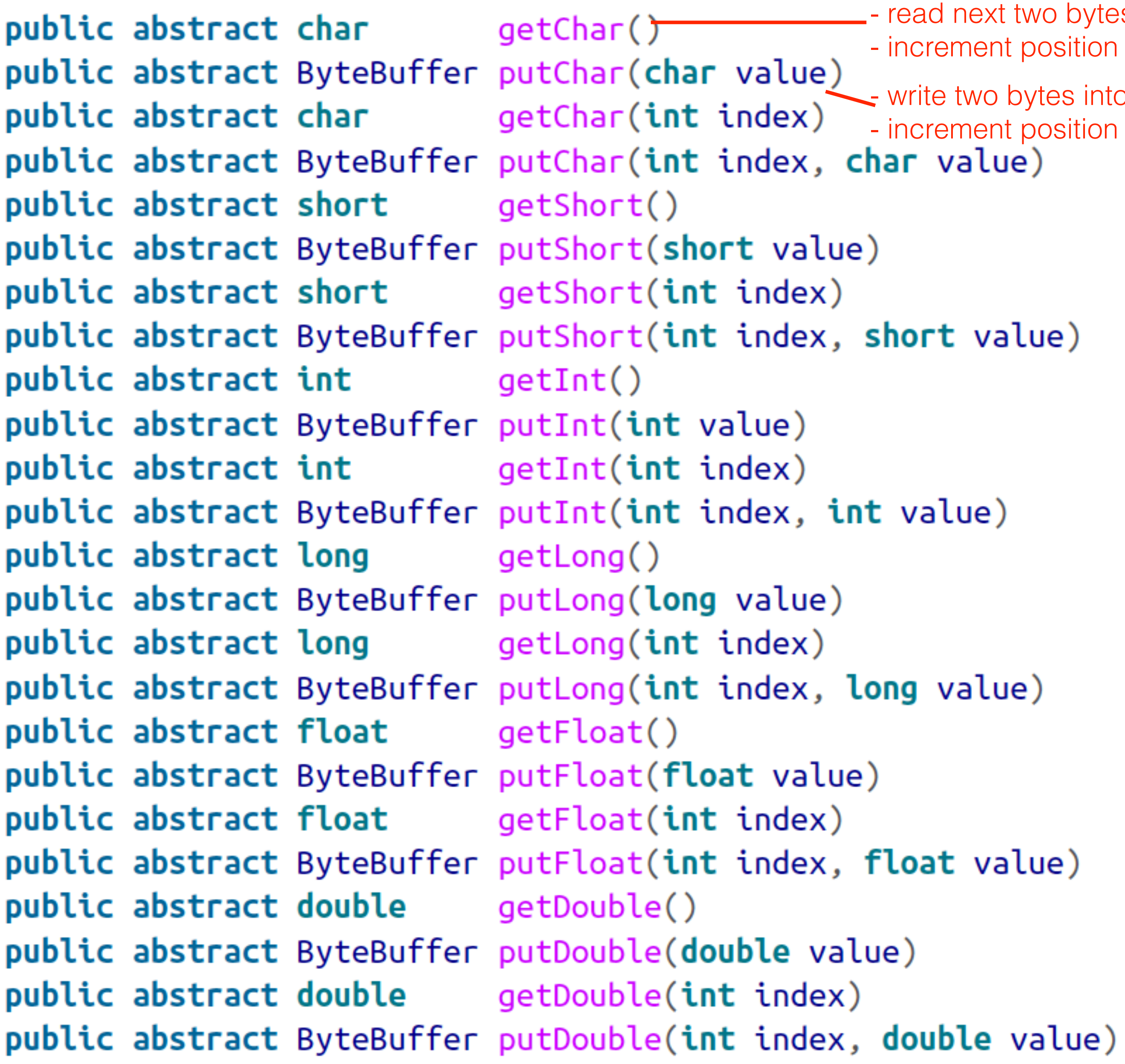
Data Conversion

- All data in Java ultimately resolves to bytes
  - any primitive data type (like int, double, float,…) can be written as bytes
- Any sequence of bytes of right length can be interpreted as a primitive datum



$2^{21} + 2^{16} + 2^9 + 2^1 + 2^0$

Methods

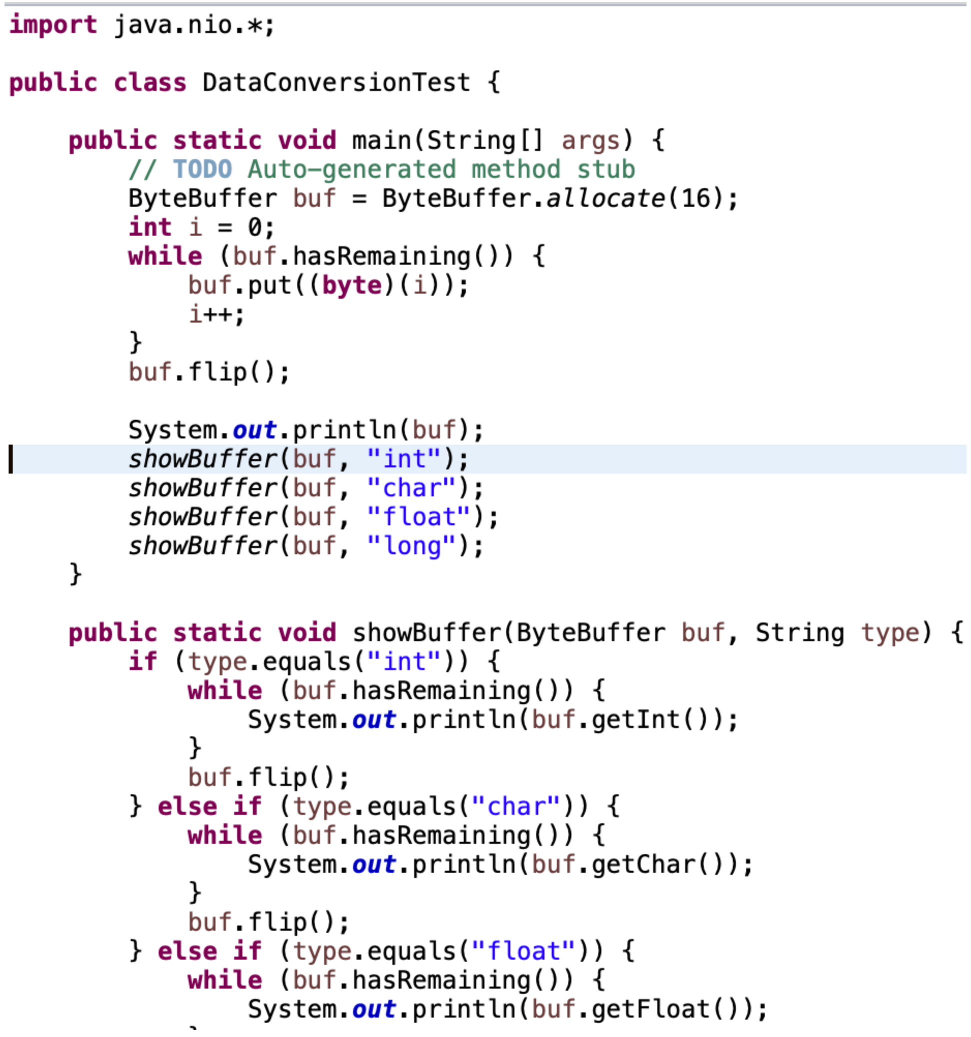


```
public abstract char       getChar()
public abstract ByteBuffer putChar(char value)
public abstract char       getChar(int index)
public abstract ByteBuffer putChar(int index, char value)
public abstract short      getShort()
public abstract ByteBuffer putShort(short value)
public abstract short      getShort(int index)
public abstract ByteBuffer putShort(int index, short value)
public abstract int        getInt()
public abstract ByteBuffer putInt(int value)
public abstract int        getInt(int index)
public abstract ByteBuffer putInt(int index, int value)
public abstract long       getLong()
public abstract ByteBuffer putLong(long value)
public abstract long       getLong(int index)
public abstract ByteBuffer putLong(int index, long value)
public abstract float      getFloat()
public abstract ByteBuffer putFloat(float value)
public abstract float      getFloat(int index)
public abstract ByteBuffer putFloat(int index, float value)
public abstract double     getDouble()
public abstract ByteBuffer putDouble(double value)
public abstract double     getDouble(int index)
public abstract ByteBuffer putDouble(int index, double value)
```

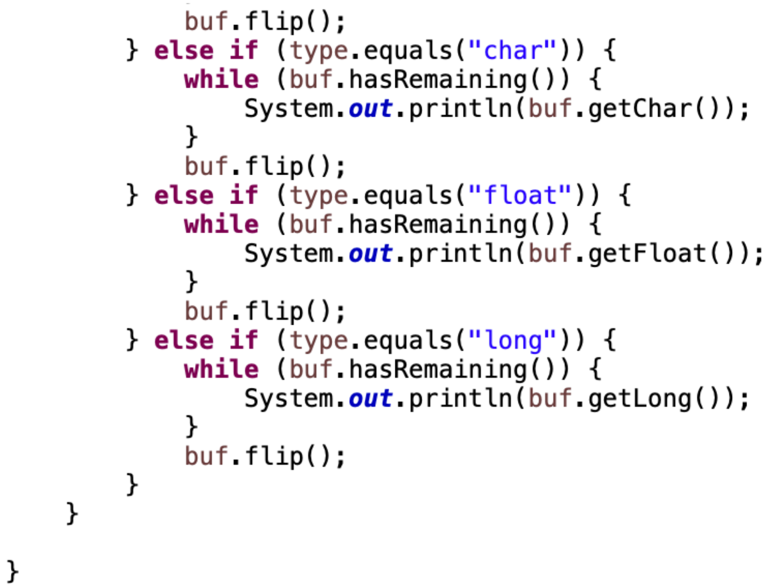
- read next two bytes and compose them into a character
- increment position by two

- write two bytes into current position
- increment position by two

Example

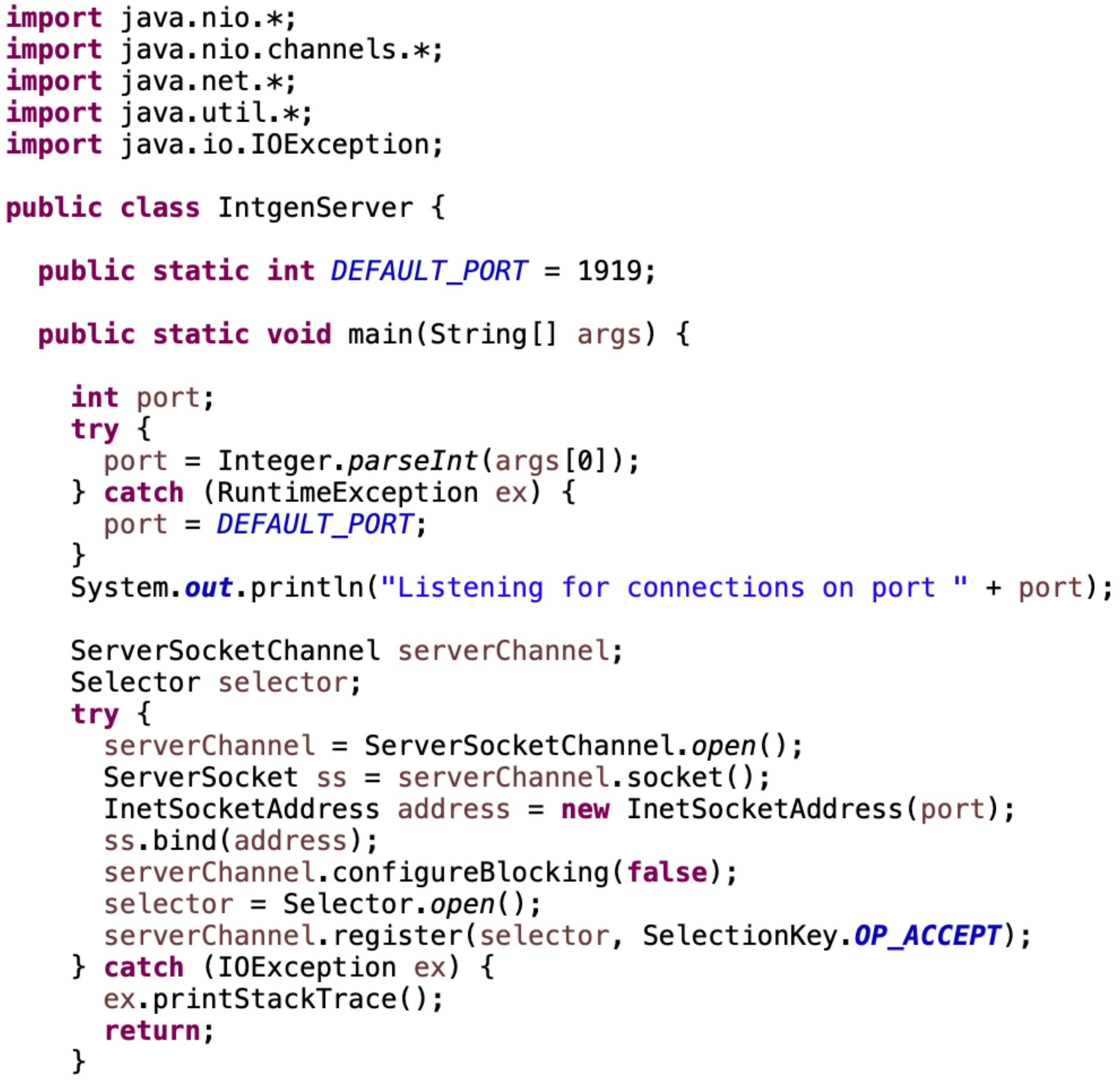


```
import java.nio.*;

public class DataConversionTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ByteBuffer buf = ByteBuffer.allocate(16);
        int i = 0;
        while (buf.hasRemaining()) {
            buf.put((byte)(i));
            i++;
        }
        buf.flip();

        System.out.println(buf);
        showBuffer(buf, "int");
        showBuffer(buf, "char");
        showBuffer(buf, "float");
        showBuffer(buf, "long");
    }

    public static void showBuffer(ByteBuffer buf, String type) {
        if (type.equals("int")) {
            while (buf.hasRemaining()) {
                System.out.println(buf.getInt());
            }
            buf.flip();
        } else if (type.equals("char")) {
            while (buf.hasRemaining()) {
                System.out.println(buf.getChar());
            }
            buf.flip();
        } else if (type.equals("float")) {
            while (buf.hasRemaining()) {
                System.out.println(buf.getFloat());
```

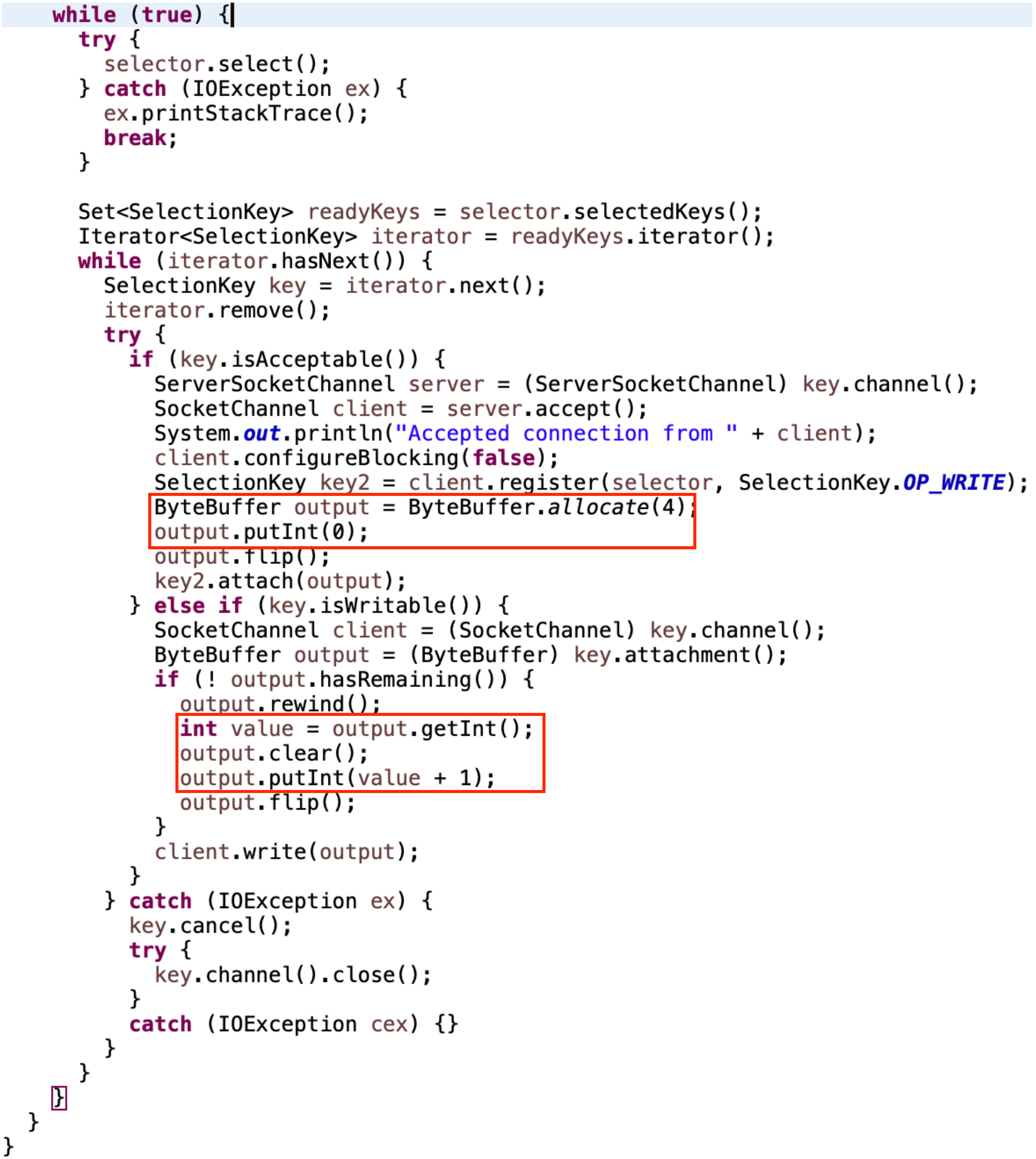


```
            buf.flip();
        } else if (type.equals("char")) {
            while (buf.hasRemaining()) {
                System.out.println(buf.getChar());
            }
            buf.flip();
        } else if (type.equals("float")) {
            while (buf.hasRemaining()) {
                System.out.println(buf.getFloat());
            }
            buf.flip();
        } else if (type.equals("long")) {
            while (buf.hasRemaining()) {
                System.out.println(buf.getLong());
            }
            buf.flip();
        }
    }
}
```

Example: IntgenServer



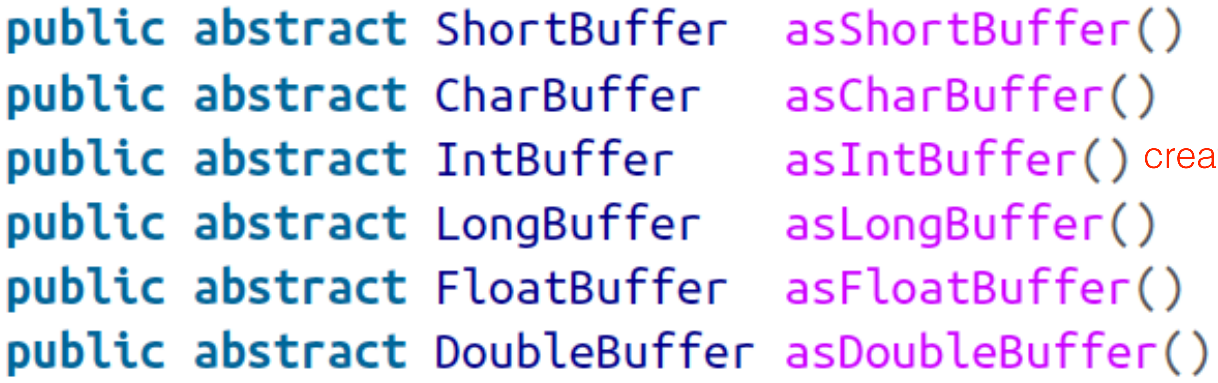
```
import java.nio.*;
import java.nio.channels.*;
import java.net.*;
import java.util.*;
import java.io.IOException;

public class IntgenServer {

  public static int DEFAULT_PORT = 1919;

  public static void main(String[] args) {

    int port;
    try {
      port = Integer.parseInt(args[0]);
    } catch (RuntimeException ex) {
      port = DEFAULT_PORT;
    }
    System.out.println("Listening for connections on port " + port);

    ServerSocketChannel serverChannel;
    Selector selector;
    try {
      serverChannel = ServerSocketChannel.open();
      ServerSocket ss = serverChannel.socket();
      InetSocketAddress address = new InetSocketAddress(port);
      ss.bind(address);
      serverChannel.configureBlocking(false);
      selector = Selector.open();
      serverChannel.register(selector, SelectionKey.OP_ACCEPT);
    } catch (IOException ex) {
      ex.printStackTrace();
      return;
    }
```



```
    while (true) {
      try {
        selector.select();
      } catch (IOException ex) {
        ex.printStackTrace();
        break;
      }

      Set<SelectionKey> readyKeys = selector.selectedKeys();
      Iterator<SelectionKey> iterator = readyKeys.iterator();
      while (iterator.hasNext()) {
        SelectionKey key = iterator.next();
        iterator.remove();
        try {
          if (key.isAcceptable()) {
            ServerSocketChannel server = (ServerSocketChannel) key.channel();
            SocketChannel client = server.accept();
            System.out.println("Accepted connection from " + client);
            client.configureBlocking(false);
            SelectionKey key2 = client.register(selector, SelectionKey.OP_WRITE);
            ByteBuffer output = ByteBuffer.allocate(4);
            output.putInt(0);
            output.flip();
            key2.attach(output);
          } else if (key.isWritable()) {
            SocketChannel client = (SocketChannel) key.channel();
            ByteBuffer output = (ByteBuffer) key.attachment();
            if (! output.hasRemaining()) {
              output.rewind();
              int value = output.getInt();
              output.clear();
              output.putInt(value + 1);
              output.flip();
            }
            client.write(output);
          }
        } catch (IOException ex) {
          key.cancel();
          try {
            key.channel().close();
          }
          catch (IOException cex) {}
        }
      }
    }
  }
}
```

View Buffers

- Interface for viewing buffer in a particular data type
- Methods

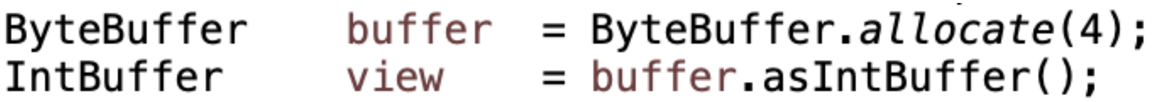


```
public abstract ShortBuffer  asShortBuffer()
public abstract CharBuffer   asCharBuffer()
public abstract IntBuffer    asIntBuffer()
public abstract LongBuffer   asLongBuffer()
public abstract FloatBuffer  asFloatBuffer()
public abstract DoubleBuffer asDoubleBuffer()
```

create a view of ByteBuffer as an IntBuffer

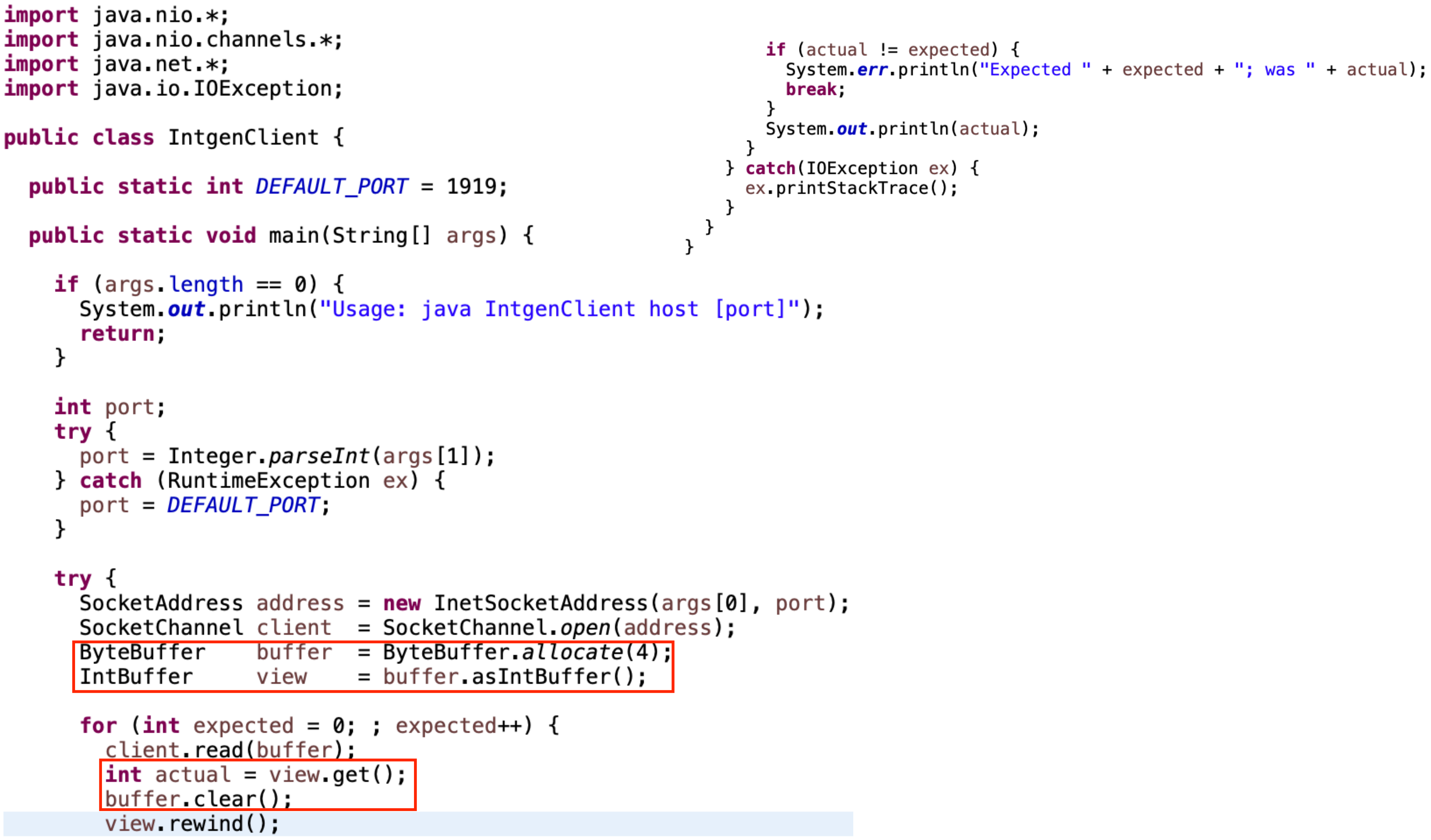
  - each buffer has its own limit, capacity, mark and position
  - changes in view are reflected in underlying buffer and vice versa

- Example



```
ByteBuffer    buffer  = ByteBuffer.allocate(4);
IntBuffer     view    = buffer.asIntBuffer();
```

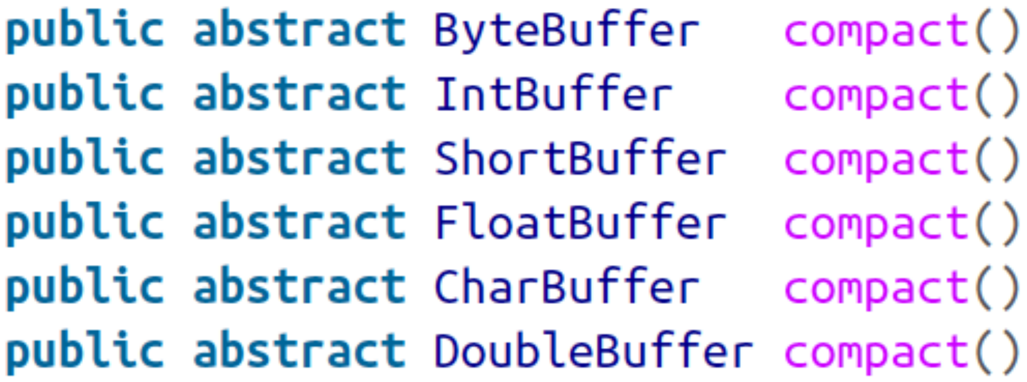
Example: IntgenClient



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

public class IntgenClient {

  public static int DEFAULT_PORT = 1919;

  public static void main(String[] args) {

    if (args.length == 0) {
      System.out.println("Usage: java IntgenClient host [port]");
      return;
    }

    int port;
    try {
      port = Integer.parseInt(args[1]);
    } catch (RuntimeException ex) {
      port = DEFAULT_PORT;
    }

    try {
      SocketAddress address = new InetSocketAddress(args[0], port);
      SocketChannel client  = SocketChannel.open(address);
      ByteBuffer    buffer  = ByteBuffer.allocate(4);
      IntBuffer     view    = buffer.asIntBuffer();

      for (int expected = 0; ; expected++) {
        client.read(buffer);
        int actual = view.get();
        buffer.clear();
        view.rewind();
        if (actual != expected) {
          System.err.println("Expected " + expected + "; was " + actual);
          break;
        }
        System.out.println(actual);
      }
    } catch(IOException ex) {
      ex.printStackTrace();
    }
  }
}
```

Compacting Buffers

- Methods

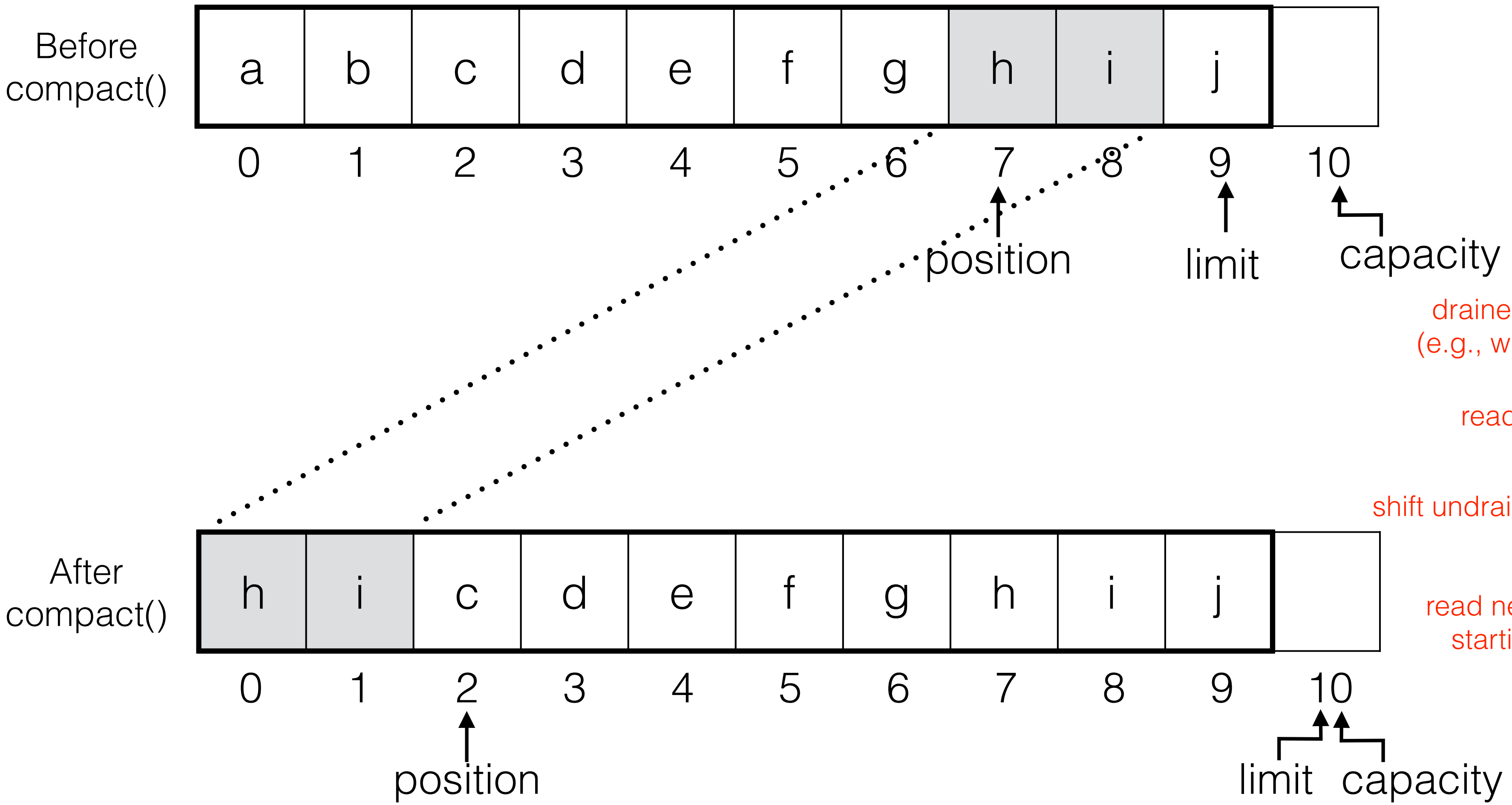


```
public abstract ByteBuffer   compact()
public abstract IntBuffer    compact()
public abstract ShortBuffer  compact()
public abstract FloatBuffer  compact()
public abstract CharBuffer   compact()
public abstract DoubleBuffer compact()
```

useful when write is not finished
and read must begin

Compacting Buffers

```
ByteBuffer buf = ByteBuffer.allocate(10);
                    ...
buf.compact();
```



drained up to position
(e.g., writing to channel)

↓

reading must begin

↓

shift undrained data block to zero

↓

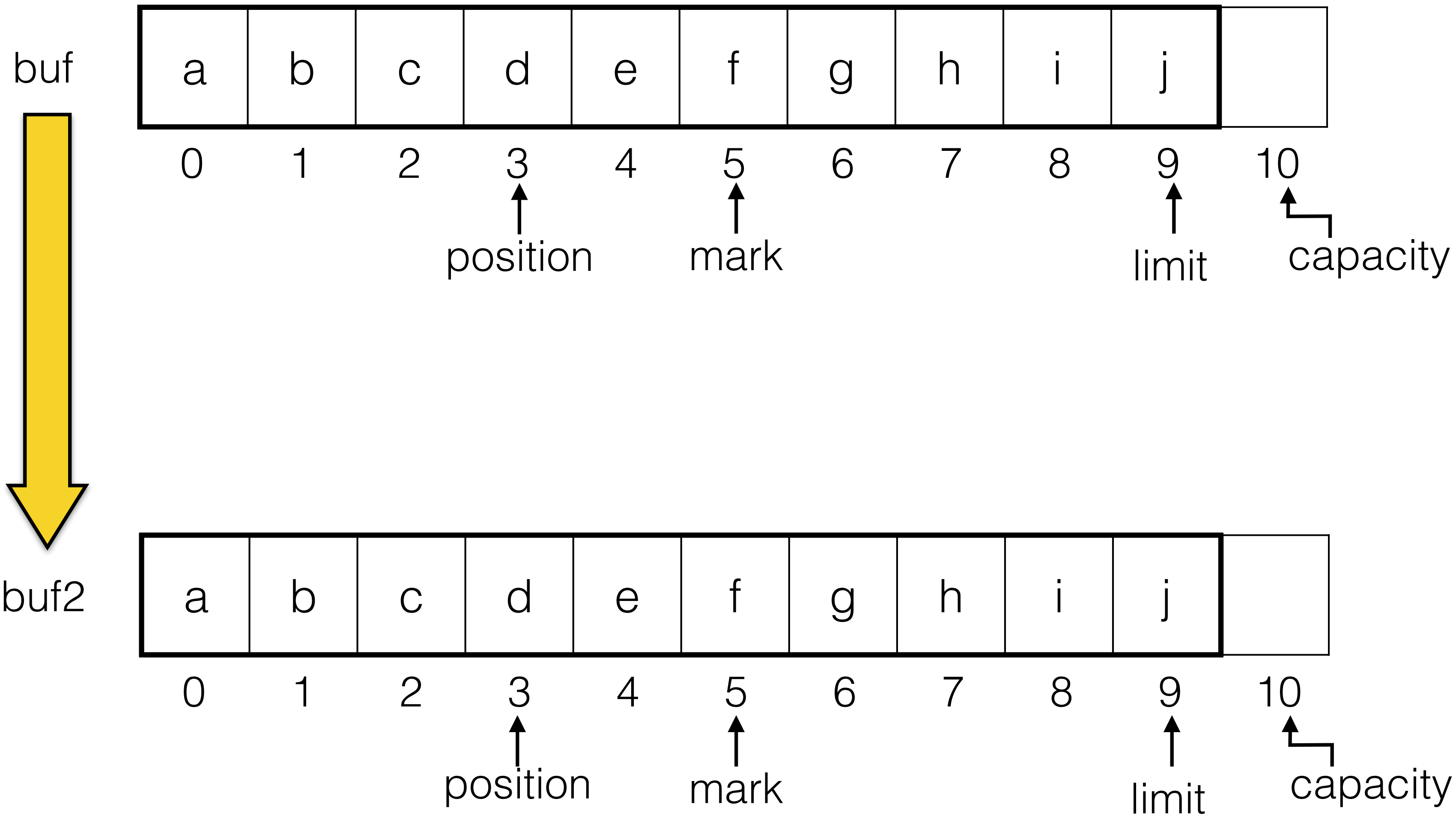
read new data into buffer
starting from position

Duplicating Buffers

- Create a copy of a buffer to deliver the same information
    - same backing array
    - change in one buffer is reflected to all other copies and backing array (so mainly used for reading)
- Useful when transmitting same data over multiple channels

Duplicating Buffers

```
ByteBuffer buf = ByteBuffer.allocate(10);
                    ...
ByteBuffer buf2 = buf.duplicate();
```



buf

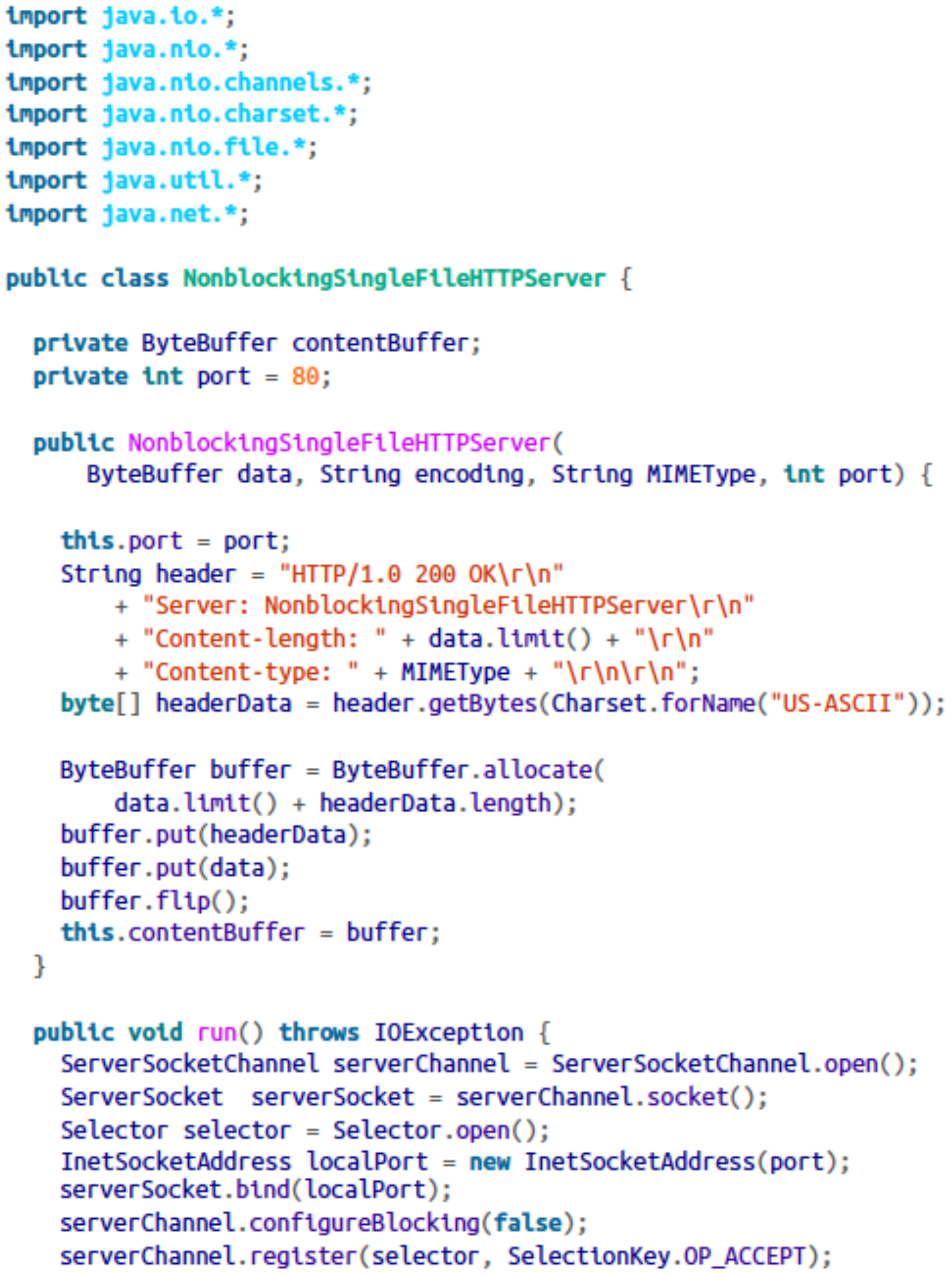
a	b	c	d	e	f	g	h	i	j	
0	1	2	3	4	5	6	7	8	9	10
			position		mark				limit	capacity

buf2

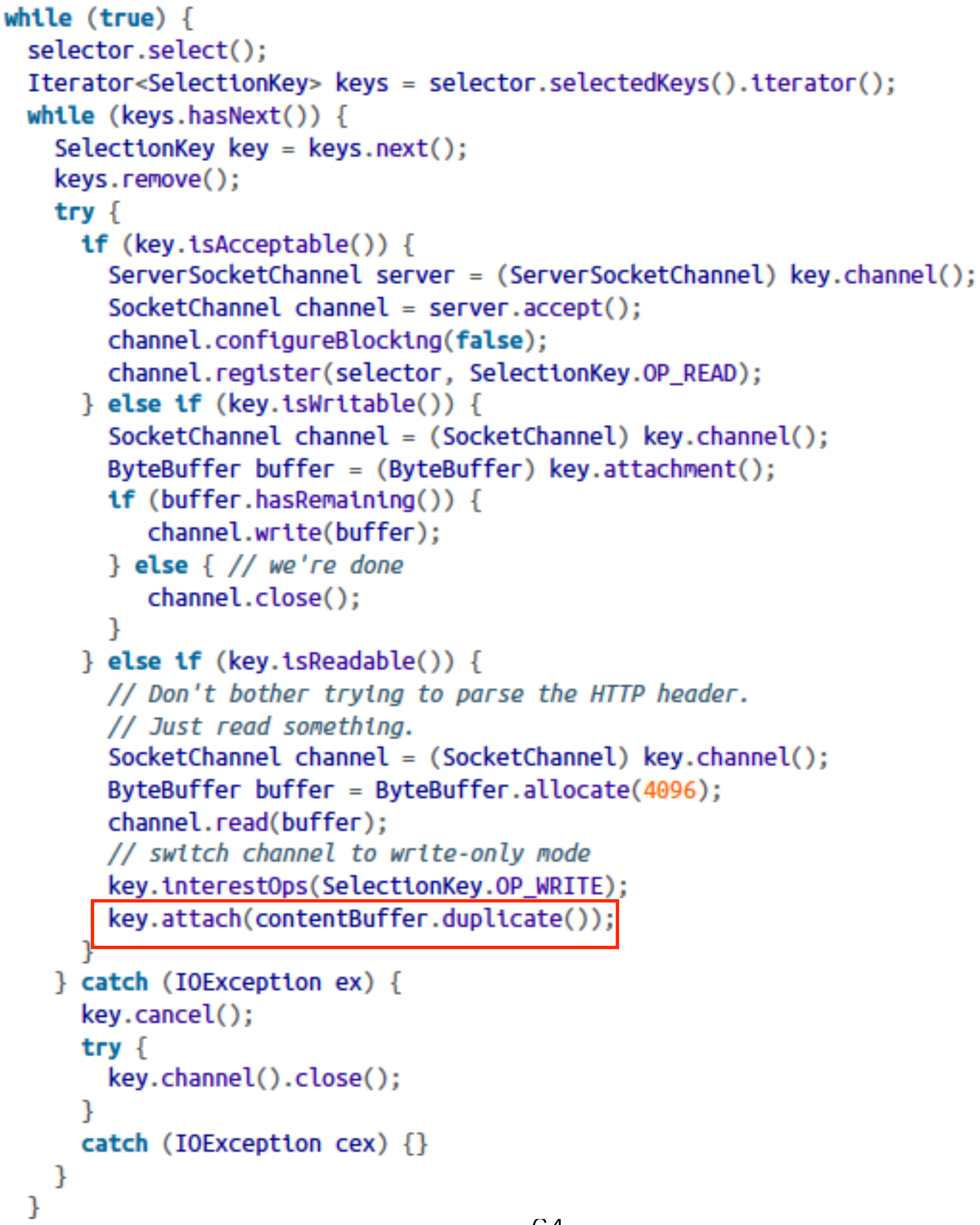
a	b	c	d	e	f	g	h	i	j	
0	1	2	3	4	5	6	7	8	9	10
			position		mark				limit	capacity

Example

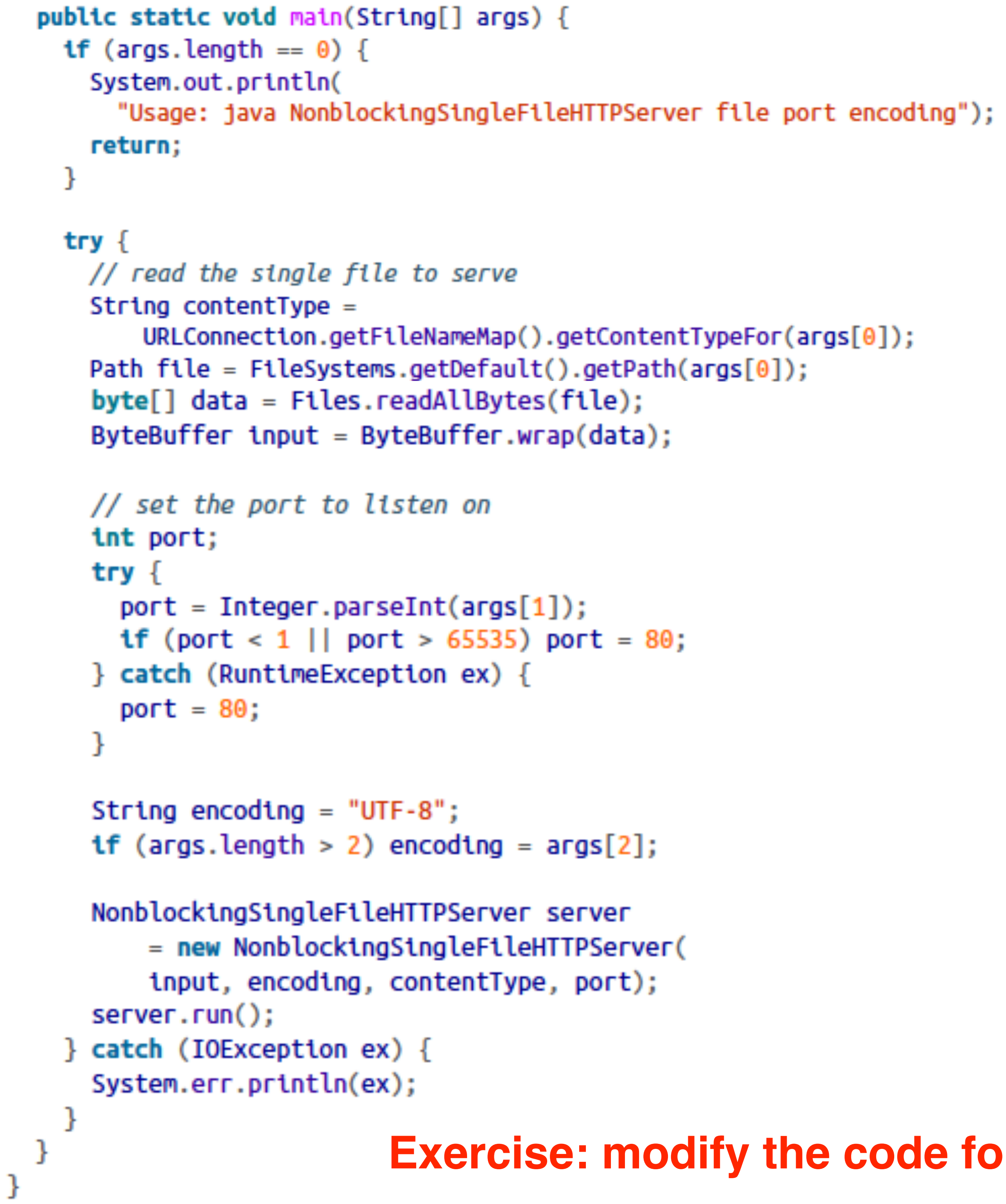
- Nonblocking version SingleFileHTTPServer
- Single file is stored in one constant, read-only buffer
- Every time a client connects, the program makes a duplicate of this buffer just for that channel (which is stored as channel's attachment)
  - without duplicates, one client has to wait until the other finishes so the original buffer can be rewound
  - duplicates enable simultaneous buffer reuse



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

public class NonblockingSingleFileHTTPServer {

  private ByteBuffer contentBuffer;
  private int port = 80;

  public NonblockingSingleFileHTTPServer(
      ByteBuffer data, String encoding, String MIMEType, int port) {

    this.port = port;
    String header = "HTTP/1.0 200 OK\r\n"
        + "Server: NonblockingSingleFileHTTPServer\r\n"
        + "Content-length: " + data.limit() + "\r\n"
        + "Content-type: " + MIMEType + "\r\n\r\n";
    byte[] headerData = header.getBytes(Charset.forName("US-ASCII"));

    ByteBuffer buffer = ByteBuffer.allocate(
        data.limit() + headerData.length);
    buffer.put(headerData);
    buffer.put(data);
    buffer.flip();
    this.contentBuffer = buffer;
  }

  public void run() throws IOException {
    ServerSocketChannel serverChannel = ServerSocketChannel.open();
    ServerSocket  serverSocket = serverChannel.socket();
    Selector selector = Selector.open();
    InetSocketAddress localPort = new InetSocketAddress(port);
    serverSocket.bind(localPort);
    serverChannel.configureBlocking(false);
    serverChannel.register(selector, SelectionKey.OP_ACCEPT);
```



```java
while (true) {
  selector.select();
  Iterator<SelectionKey> keys = selector.selectedKeys().iterator();
  while (keys.hasNext()) {
    SelectionKey key = keys.next();
    keys.remove();
    try {
      if (key.isAcceptable()) {
        ServerSocketChannel server = (ServerSocketChannel) key.channel();
        SocketChannel channel = server.accept();
        channel.configureBlocking(false);
        channel.register(selector, SelectionKey.OP_READ);
      } else if (key.isWritable()) {
        SocketChannel channel = (SocketChannel) key.channel();
        ByteBuffer buffer = (ByteBuffer) key.attachment();
        if (buffer.hasRemaining()) {
           channel.write(buffer);
        } else { // we're done
           channel.close();
        }
      } else if (key.isReadable()) {
        // Don't bother trying to parse the HTTP header.
        // Just read something.
        SocketChannel channel = (SocketChannel) key.channel();
        ByteBuffer buffer = ByteBuffer.allocate(4096);
        channel.read(buffer);
        // switch channel to write-only mode
        key.interestOps(SelectionKey.OP_WRITE);
        key.attach(contentBuffer.duplicate());
      }
    } catch (IOException ex) {
      key.cancel();
      try {
        key.channel().close();
      }
      catch (IOException cex) {}
    }
  }
```

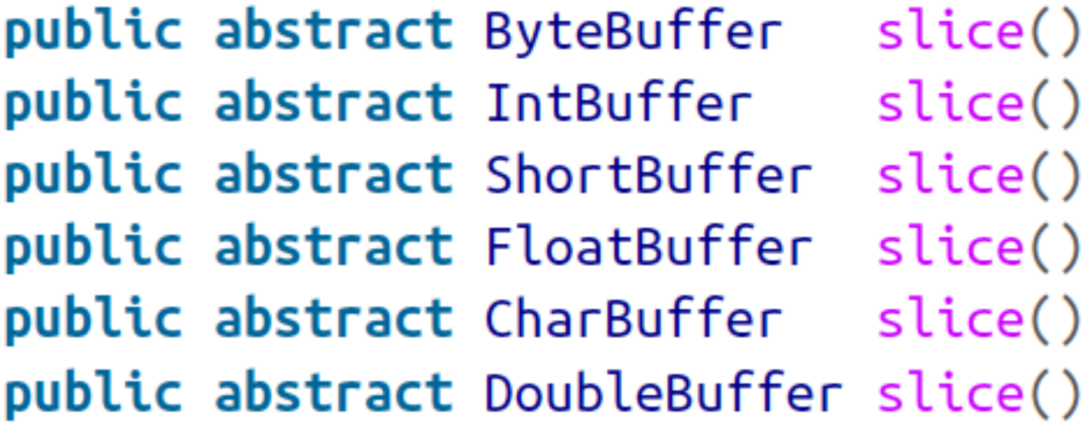


```
  public static void main(String[] args) {
    if (args.length == 0) {
      System.out.println(
        "Usage: java NonblockingSingleFileHTTPServer file port encoding");
      return;
    }

    try {
      // read the single file to serve
      String contentType =
          URLConnection.getFileNameMap().getContentTypeFor(args[0]);
      Path file = FileSystems.getDefault().getPath(args[0]);
      byte[] data = Files.readAllBytes(file);
      ByteBuffer input = ByteBuffer.wrap(data);

      // set the port to listen on
      int port;
      try {
        port = Integer.parseInt(args[1]);
        if (port < 1 || port > 65535) port = 80;
      } catch (RuntimeException ex) {
        port = 80;
      }

      String encoding = "UTF-8";
      if (args.length > 2) encoding = args[2];

      NonblockingSingleFileHTTPServer server
          = new NonblockingSingleFileHTTPServer(
          input, encoding, contentType, port);
      server.run();
    } catch (IOException ex) {
      System.err.println(ex);
    }
  }
}
```

Exercise: modify the code for korean text

Read-only Copy

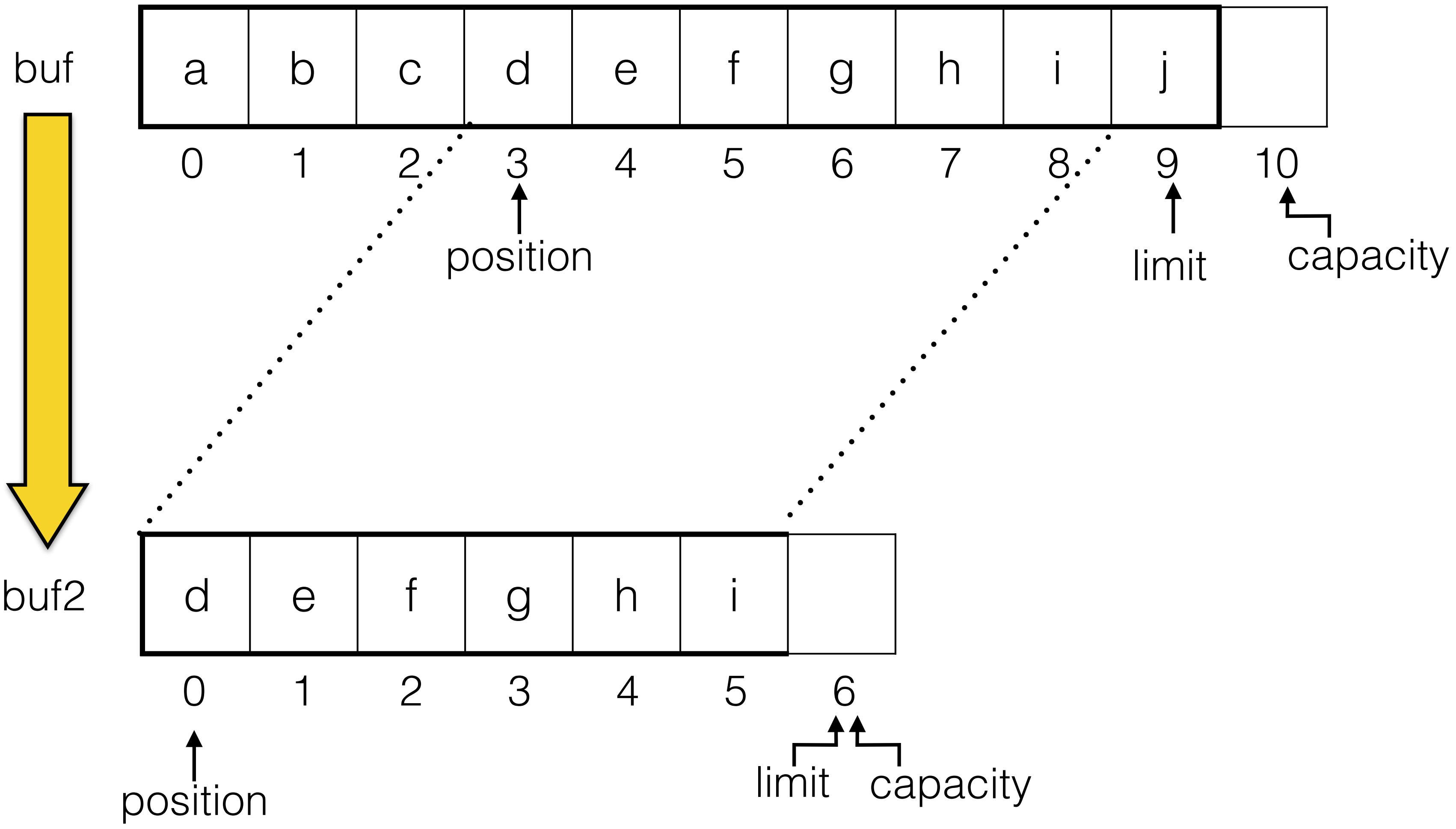
- asReadOnlyBuffer()
  - same as duplicate() except that copy created by asReadOnlyBuffer() only allows reading
- slice()
  - copy a subsequence of original buffer from position to limit
  - have independent 4 properties



```
public abstract ByteBuffer   slice()
public abstract IntBuffer    slice()
public abstract ShortBuffer  slice()
public abstract FloatBuffer  slice()
public abstract CharBuffer   slice()
public abstract DoubleBuffer slice()
```

Read-only Copy

```
ByteBuffer buf = ByteBuffer.allocate(10);
...
ByteBuffer buf2 = buf.slice();
```



Channels

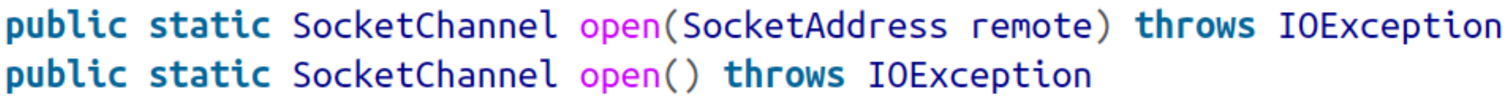
Channels

- Channels move blocks of data into and out of buffers to and from various I/O sources like files, sockets, datagrams,..
- Three main channel classes
  - SocketChannel: TCP
  - ServerSocketChannel: TCP
  - DatagramChannel: UDP

SocketChannel

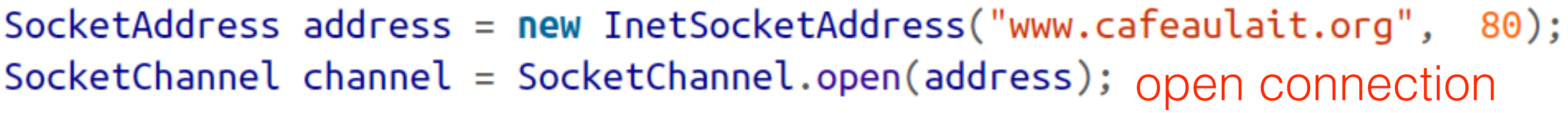
SocketChannel Class

- Read from and write to TCP sockets
- Data must be encoded in ByteBuffer
- Connecting



```
public static SocketChannel open(SocketAddress remote) throws IOException
public static SocketChannel open() throws IOException
```

  - blocking mode example

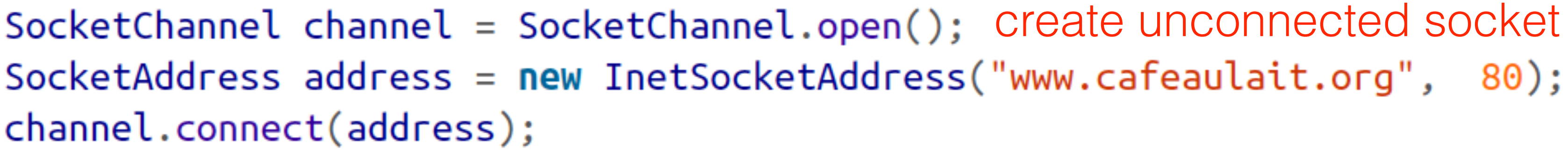


```
SocketAddress address = new InetSocketAddress("www.cafeaulait.org",  80);
SocketChannel channel = SocketChannel.open(address);
```

open connection

    - open(address) blocks

  - another example

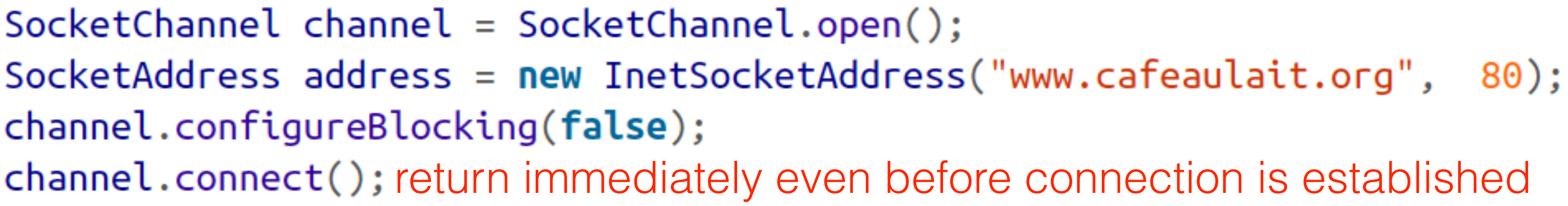


```
SocketChannel channel = SocketChannel.open();
SocketAddress address = new InetSocketAddress("www.cafeaulait.org",  80);
channel.connect(address);
```

create unconnected socket

SocketChannel Class (contd.)

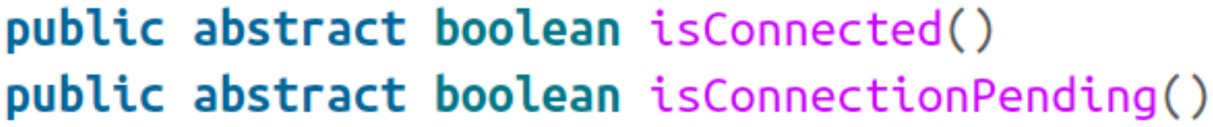
- Nonblocking mode example



```
SocketChannel channel = SocketChannel.open();
SocketAddress address = new InetSocketAddress("www.cafeaulait.org",  80);
channel.configureBlocking(false);
channel.connect();
```

return immediately even before connection is established

- Other methods

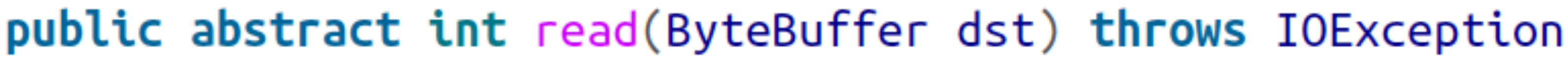


```
public abstract boolean isConnected()
public abstract boolean isConnectionPending()
```

  - isConnected(): true if connection established (socket is open and connected)
  - isConnectionPending(): true if connection is still being set up, but not yet open

Reading

- Method



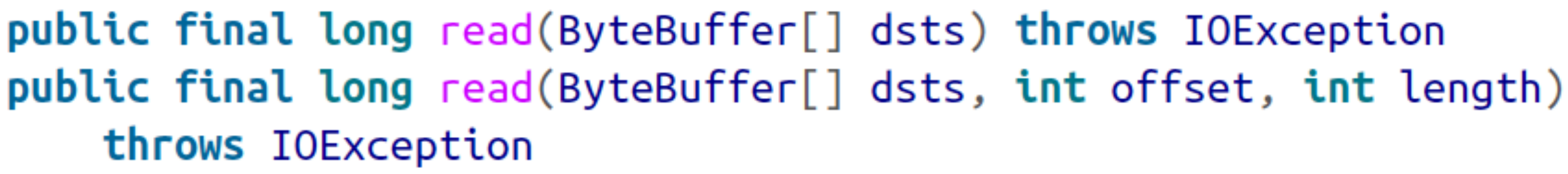
```
public abstract int read(ByteBuffer dst) throws IOException
```

  - channel fills buffer with as much data as it can
  - return # bytes put in the buffer
  - when it encounters end of stream while reading, it returns -1 on the next call to read()
  - blocking
    - read at least one byte, return -1, or throw an exception
  - nonblocking
    - can return 0

Reading (contd.)

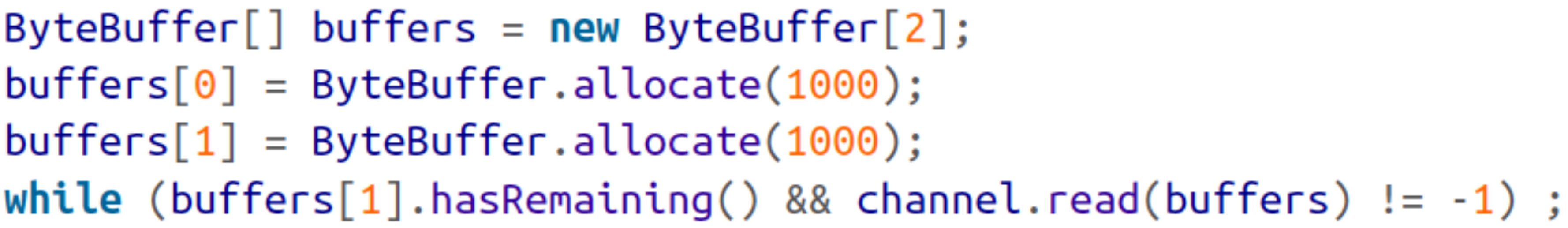
- How to read until buffer is filled or end of stream is detected?

- Scattering



```
public final long read(ByteBuffer[] dsts) throws IOException
public final long read(ByteBuffer[] dsts, int offset, int length)
    throws IOException
```

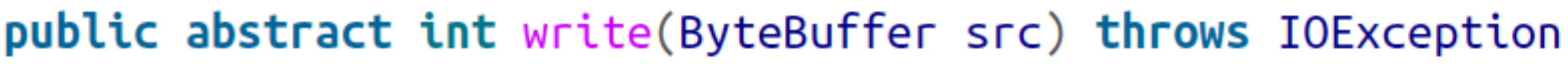
  - fill several buffers from one source
  - example



```
ByteBuffer[] buffers = new ByteBuffer[2];
buffers[0] = ByteBuffer.allocate(1000);
buffers[1] = ByteBuffer.allocate(1000);
while (buffers[1].hasRemaining() && channel.read(buffers) != -1) ;
```

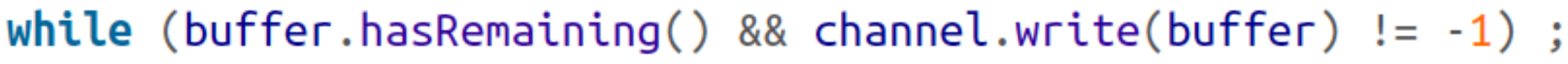
Writing

- Method



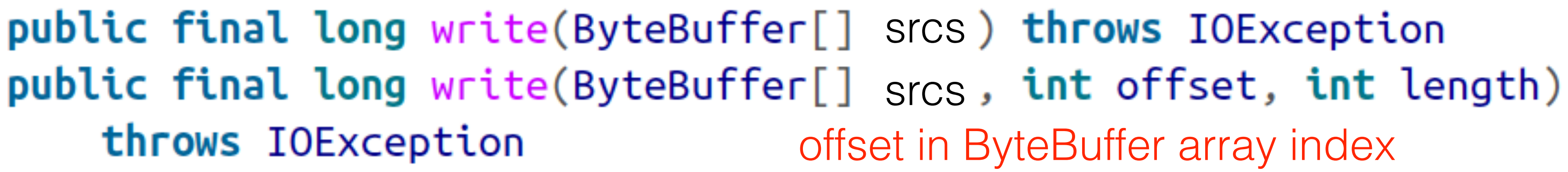
```
public abstract int write(ByteBuffer src) throws IOException
```

  - doesn’t guarantee to write complete content of buffer in nonblocking mode
  - useful clause



```
while (buffer.hasRemaining() && channel.write(buffer) != -1) ;
```

- Buffer array



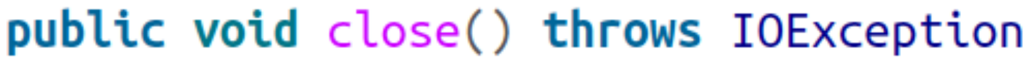
```
public final long write(ByteBuffer[] srcs ) throws IOException
public final long write(ByteBuffer[] srcs , int offset, int length)
    throws IOException
```

offset in ByteBuffer array index

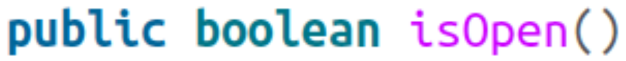
  - for example, HTTP head and body in separate buffers

Closing

- Methods



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



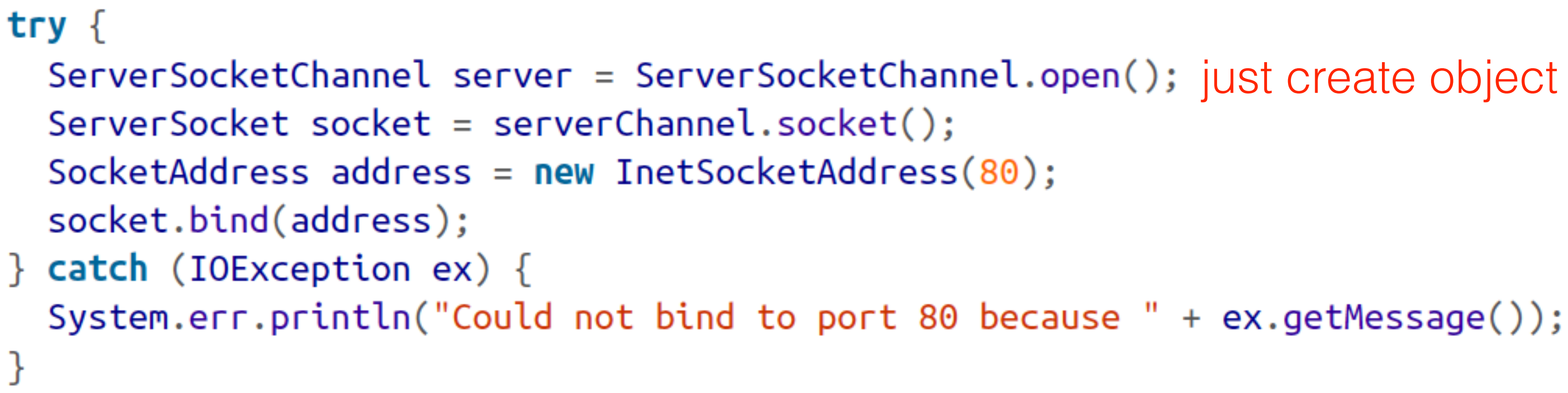
```
public boolean isOpen()
```

  - true if channel is open

ServerSocketChannel Class

ServerSocketChannel

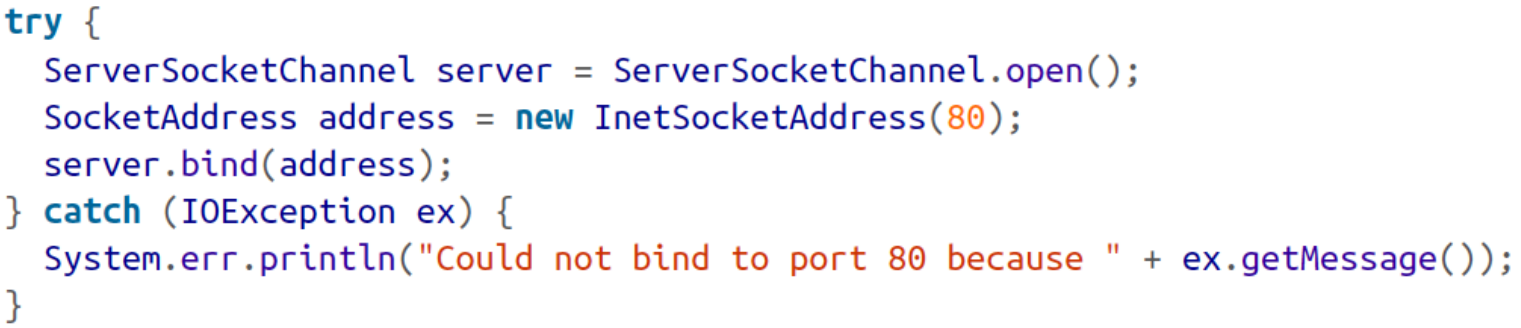
- One purpose
  - accept incoming connections
- This class declares only four methods including accept()
- Creating



```
try {
  ServerSocketChannel server = ServerSocketChannel.open();
  ServerSocket socket = serverChannel.socket();
  SocketAddress address = new InetSocketAddress(80);
  socket.bind(address);
} catch (IOException ex) {
  System.err.println("Could not bind to port 80 because " + ex.getMessage());
}
```

just create object

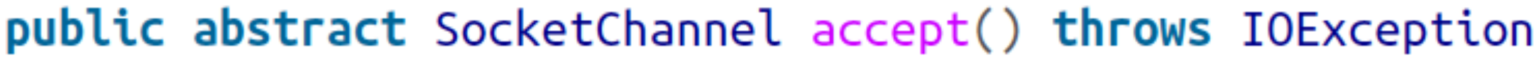
or



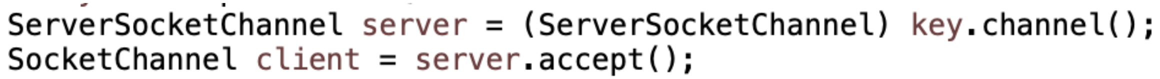
```
try {
  ServerSocketChannel server = ServerSocketChannel.open();
  SocketAddress address = new InetSocketAddress(80);
  server.bind(address);
} catch (IOException ex) {
  System.err.println("Could not bind to port 80 because " + ex.getMessage());
}
```

Accepting

- Method



```
public abstract SocketChannel accept() throws IOException
```



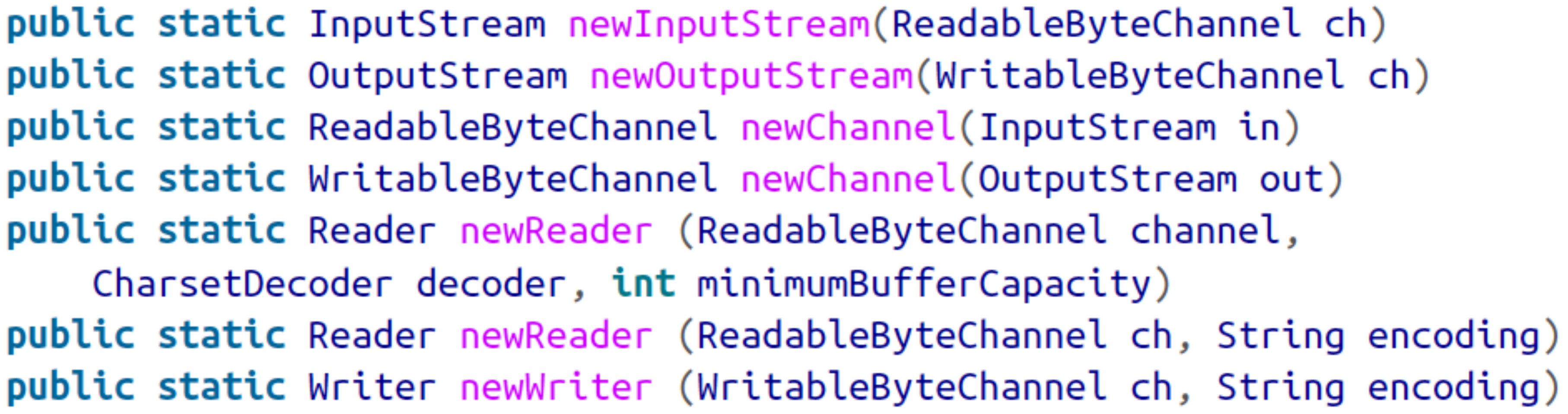
```
ServerSocketChannel server = (ServerSocketChannel) key.channel();
SocketChannel client = server.accept();
```

- Blocking mode (default)
  - wait for incoming connection, and
  - return SocketChannel object connected to client
- Nonblocking mode
  - return null if no incoming connection
  - appropriate for servers doing a lot of work for each connection and thus may want to process multiple requests in parallel
  - need to call configureBlocking(false)

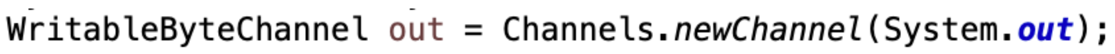
Channels Class

Channels Class

- Simple utility class for wrapping channels around traditional I/O-based streams, readers, writers,…
- Useful when you want to use the new I/O model in one part, but still incorporate legacy APIs that expect streams
- Methods



```
public static InputStream newInputStream(ReadableByteChannel ch)
public static OutputStream newOutputStream(WritableByteChannel ch)
public static ReadableByteChannel newChannel(InputStream in)
public static WritableByteChannel newChannel(OutputStream out)
public static Reader newReader (ReadableByteChannel channel,
    CharsetDecoder decoder, int minimumBufferCapacity)
public static Reader newReader (ReadableByteChannel ch, String encoding)
public static Writer newWriter (WritableByteChannel ch, String encoding)
```



```
WritableByteChannel out = Channels.newChannel(System.out);
```

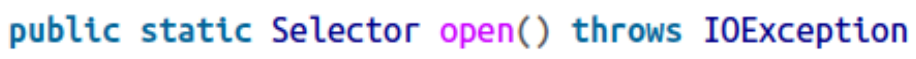
Readiness Selection

Motivation

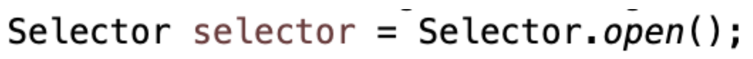
- It's important to choose a socket that will not block when read or written
  - primarily of interest to servers
- In order to perform readiness selection, channels are registered with a Selector object
  - each channel is assigned a SelectionKey
- Program can ask the Selector object for the set of keys to the channels that are ready to perform the operation you want to perform without blocking

Selector Class

- Creating a new selector, by calling Selector.open()

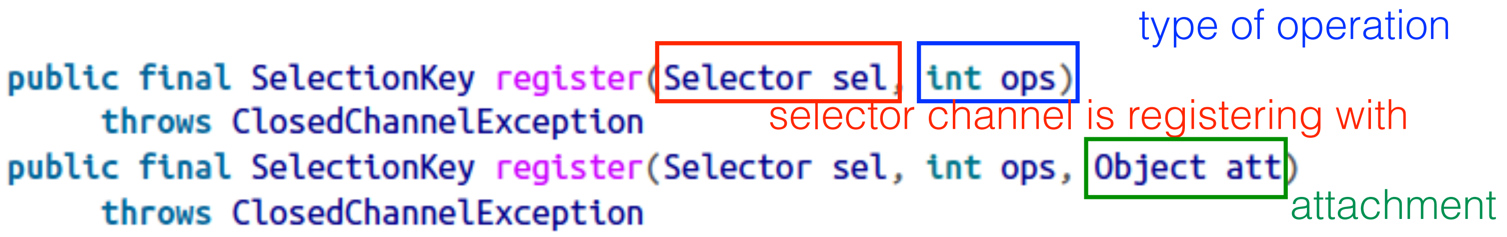


```
public static Selector open() throws IOException
```



```
Selector selector = Selector.open();
```

- Adding a channel to the selector

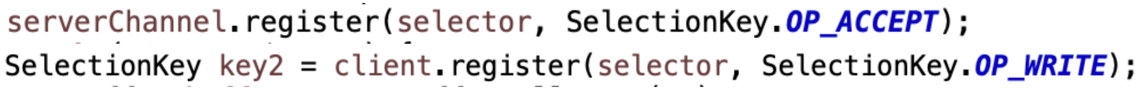


type of operation

```
public final SelectionKey register(Selector sel, int ops)
      throws ClosedChannelException
public final SelectionKey register(Selector sel, int ops, Object att)
      throws ClosedChannelException
```

selector channel is registering with

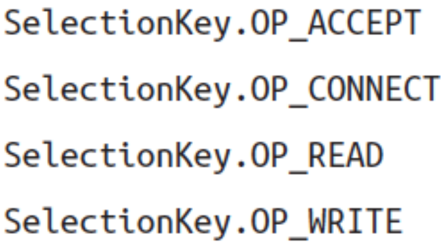
attachment



```
serverChannel.register(selector, SelectionKey.OP_ACCEPT);
SelectionKey key2 = client.register(selector, SelectionKey.OP_WRITE);
```

  - defined in SelectableChannel class
    - not all channels are selectable(e.g., FileChannel) but all network channels are

- Types of operation (defined in SelectionKey)

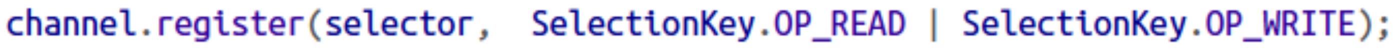
  - 

```
SelectionKey.OP_ACCEPT
SelectionKey.OP_CONNECT
SelectionKey.OP_READ
SelectionKey.OP_WRITE
```

bit-flag int constants (1, 2, 4,…)

exercise: find out the number of each type

- Example

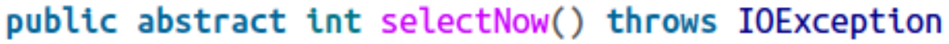


```
channel.register(selector,  SelectionKey.OP_READ | SelectionKey.OP_WRITE);
```

  - | bitwise OR operator
  - register for both reading and writing on a socket

Selecting Ready Channels

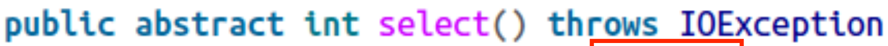
- selectNow()



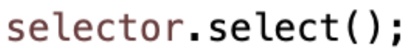
```
public abstract int selectNow() throws IOException
```

  - perform a nonblocking select, i.e., immediately return if no connections are ready to be processed

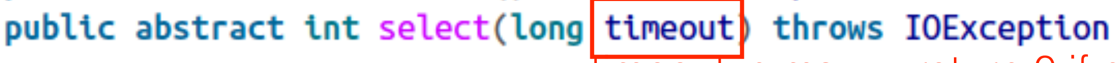
- select()



```
public abstract int select() throws IOException
```



```
selector.select();
```

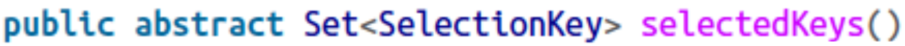


```
public abstract int select(long timeout) throws IOException
```

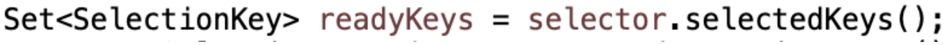
timeout in ms => return 0 if no channels are ready in timeout

  - wait until at least one registered channel is ready to be processed

- Retrieving ready channels



```
public abstract Set<SelectionKey> selectedKeys()
```

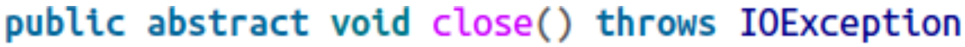


```
Set<SelectionKey> readyKeys = selector.selectedKeys();
```

  - you need to iterate through the returned set

Closing

- Method

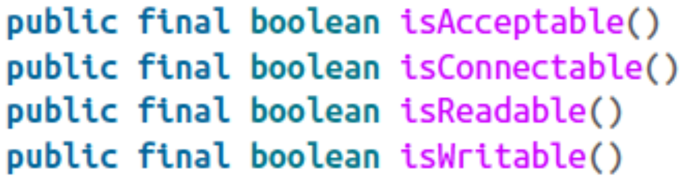


```
public abstract void close() throws IOException
```

  - release any resources associated with selector
  - cancel all keys registered
  - interrupt up any threads blocked by one of this selector’s select methods

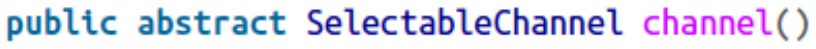
SelectionKey Class

- Serve as pointers to channels
- Hold an object attachment
- SelectionKey objects are returned by register() method
  - single channel can be registered with multiple selectors
- Testing what a key is ready to do

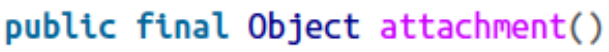


```
public final boolean isAcceptable()
public final boolean isConnectable()
public final boolean isReadable()
public final boolean isWritable()
```

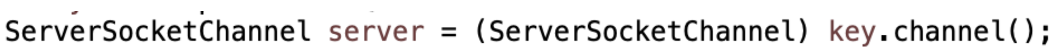
- Retrieving a channel and attachment



```
public abstract SelectableChannel channel()
```



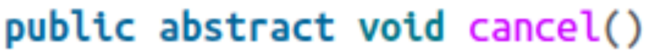
```
public final Object attachment()
```



```
ServerSocketChannel server = (ServerSocketChannel) key.channel();
```

Canceling

- Method



```
public abstract void cancel()
```

  - when you are finished with a connection, deregister its SelectionKey object so the selector doesn't waste any resources querying it for readiness
  - not absolutely necessary

- Note
  - closing a channel automatically deregisters all keys for that channel in all selectors
    - * a single channel can be registered with multiple selectors