Java NIO

Shristi Technology Labs

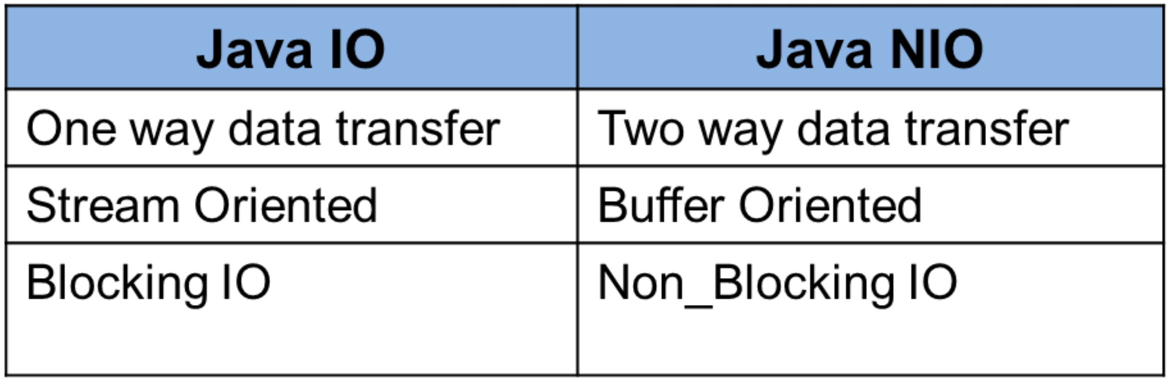
Contents

- Overview of NIO
- Channels
- Buffers
- Selectors
- Channel ToChannel
- Asynchronous FileChannel

Overview

- The new input output package is used for dealing with input and output operations
- This supports both unidirectional and bidirectional data transfer
- This is buffer oriented

NIO Vs IO



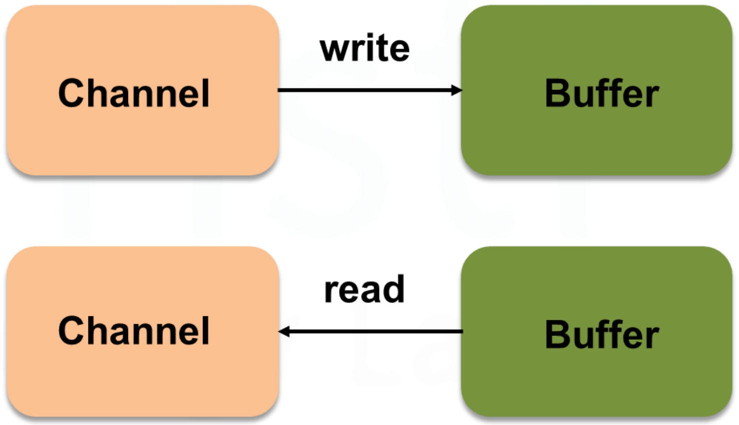
Java IO	Java NIO
One way data transfer	Two way data transfer
Stream Oriented	Buffer Oriented
Blocking IO	Non_Blocking IO

Components of NIO

- **Channel**
  - A Channel is a like a stream.
  - Data can read from a channel into a Buffer.
  - Data can also be written from a Buffer into a Channel
- **Buffer**
  - A container for data of a specific primitive type.
- **Selector**
  - Selector is used to check the readiness of Channels for reading or writing.

Channel

- It is like a stream that transports data from a file to a buffer or vice versa



- With Channels
  - Read data (from file) into a buffer
  - Write data (into a file) from the buffer

- Channels can be unidirectional(read or write) or bidirectional(read and write). Streams are one-way (read or write).

Channel Implementations

The Channel implementations in Java NIO

FileChannel

- This channel can read/write data from/to files.

DatagramChannel

- This channel can read and write data over the network via UDP(User Datagram Protocol)

SocketChannel

- This channel can read and write data over the network via TCP.

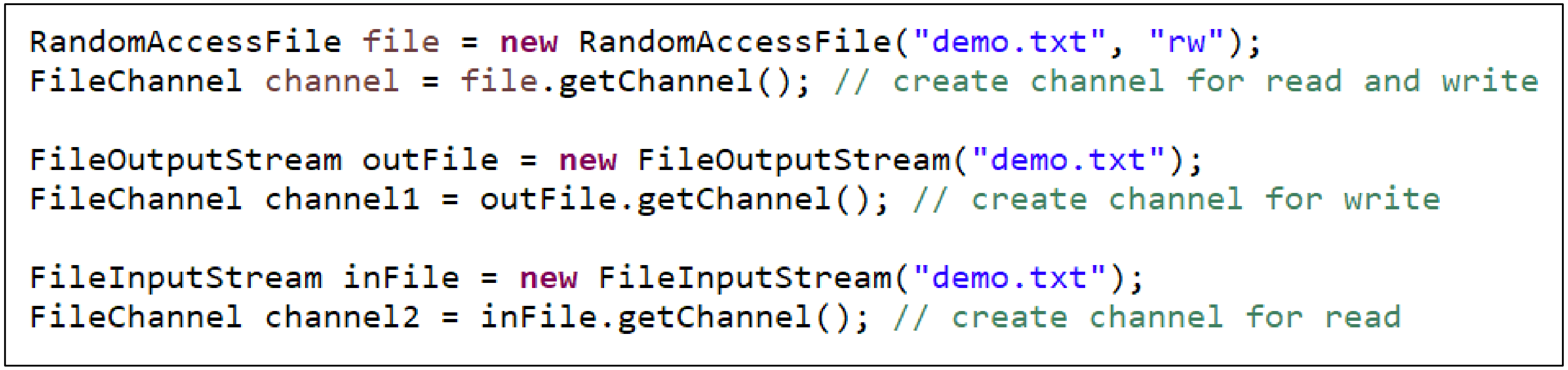
ServerSocketChannel

- The ServerSocketChannel allows you to listen for incoming TCP connections, like a web server .
- A SocketChannel is created for each incoming connection.

FileChannel

- Helps to read data from a file, and write data to a file
- Alternate to reading and writing file using IO
- Get a *FileChannel* using an *InputStream*, *OutputStream*, or a *RandomAccessFile*

Example



```
RandomAccessFile file = new RandomAccessFile("demo.txt", "rw");
FileChannel channel = file.getChannel(); // create channel for read and write

FileOutputStream outFile = new FileOutputStream("demo.txt");
FileChannel channel1 = outFile.getChannel(); // create channel for write

FileInputStream inFile = new FileInputStream("demo.txt");
FileChannel channel2 = inFile.getChannel(); // create channel for read
```

Buffer

- Is a linear, finite sequence of elements(container) of a specific primitive type.
- It has a capacity, limit and position

Capacity

- the number of elements it can hold
- If the buffer is full, clear it before writing data into it

Limit

- the index of the first element that should not be read or written

Position

- the index of the next element that should be read or written
- The position can be max capacity-1

Buffer Types

The Buffer types represent different data types

- ByteBuffer
- MappedByteBuffer
- CharBuffer
- DoubleBuffer
- FloatBuffer
- IntBuffer
- LongBuffer
- ShortBuffer

Methods – allocate, read, get

Get Buffer Object

```
// create buffer with capacity of 100 bytes
ByteBuffer buffer = ByteBuffer.allocate(100);
```

Write data from a channel into a Buffer

```
// write data into a buffer
// read from channel into a buffer
int bytes = channel.read(buffer);
```

Read data from buffer into a channel

```
char data = (char) buffer.get();
System.out.print(data);
```

Example – read data from channel into buffer

```
RandomAccessFile file = new RandomAccessFile("demo.txt", "rw");
System.out.println(file.length());
FileChannel channel = file.getChannel(); // channel to read and write
ByteBuffer buffer = ByteBuffer.allocate(100);// create buffer of 100 bytes
// read from channel into a buffer(write into buffer)
int bytes = channel.read(buffer);
System.out.println("read " + bytes);
buffer.flip();// flip the buffer to read
while (buffer.hasRemaining()) {
    char data = (char) buffer.get();
    System.out.print(data);
}
channel.close();
file.close();
```

Example – write data from buffer into channel

```
RandomAccessFile file = new RandomAccessFile("demo.txt", "rw");
System.out.println(file.length());
FileChannel channel = file.getChannel(); // channel to read and write
ByteBuffer buffer = ByteBuffer.allocate(100);// create buffer of 100 bytes
//add content to buffer - read mode
String message = "Have a good day";
buffer.put(message.getBytes());

buffer.flip(); //change to write mode
//write data from buffer into channel
while (buffer.hasRemaining()) {
    channel.write(buffer);
}
channel.close();
file.close();
```

Buffer methods – flip, clear, rewind

flip()

- switches a Buffer from writing mode to reading mode or vice versa
- Sets the position to 0

rewind()

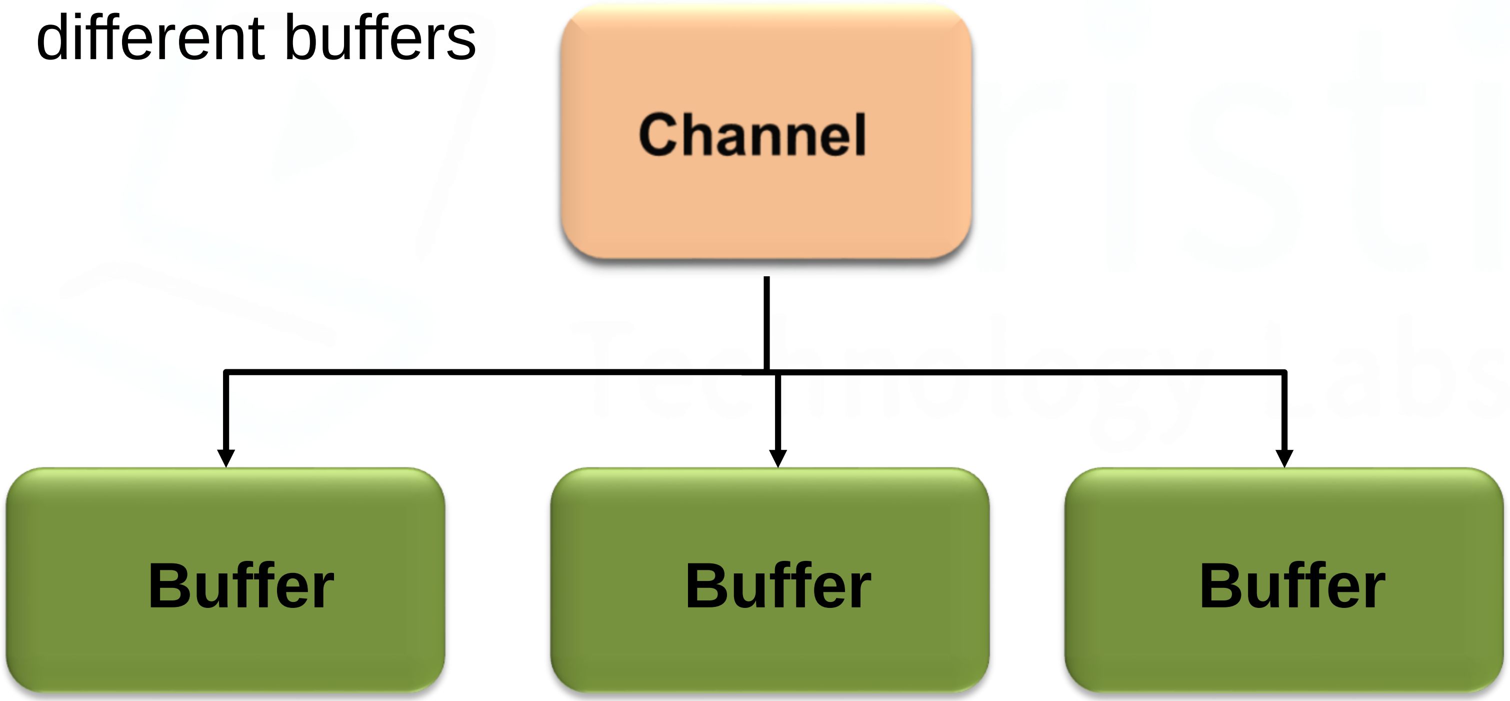
- Sets position of buffer back to 0.
- Limit doesn’t change and there is no data loss

clear()

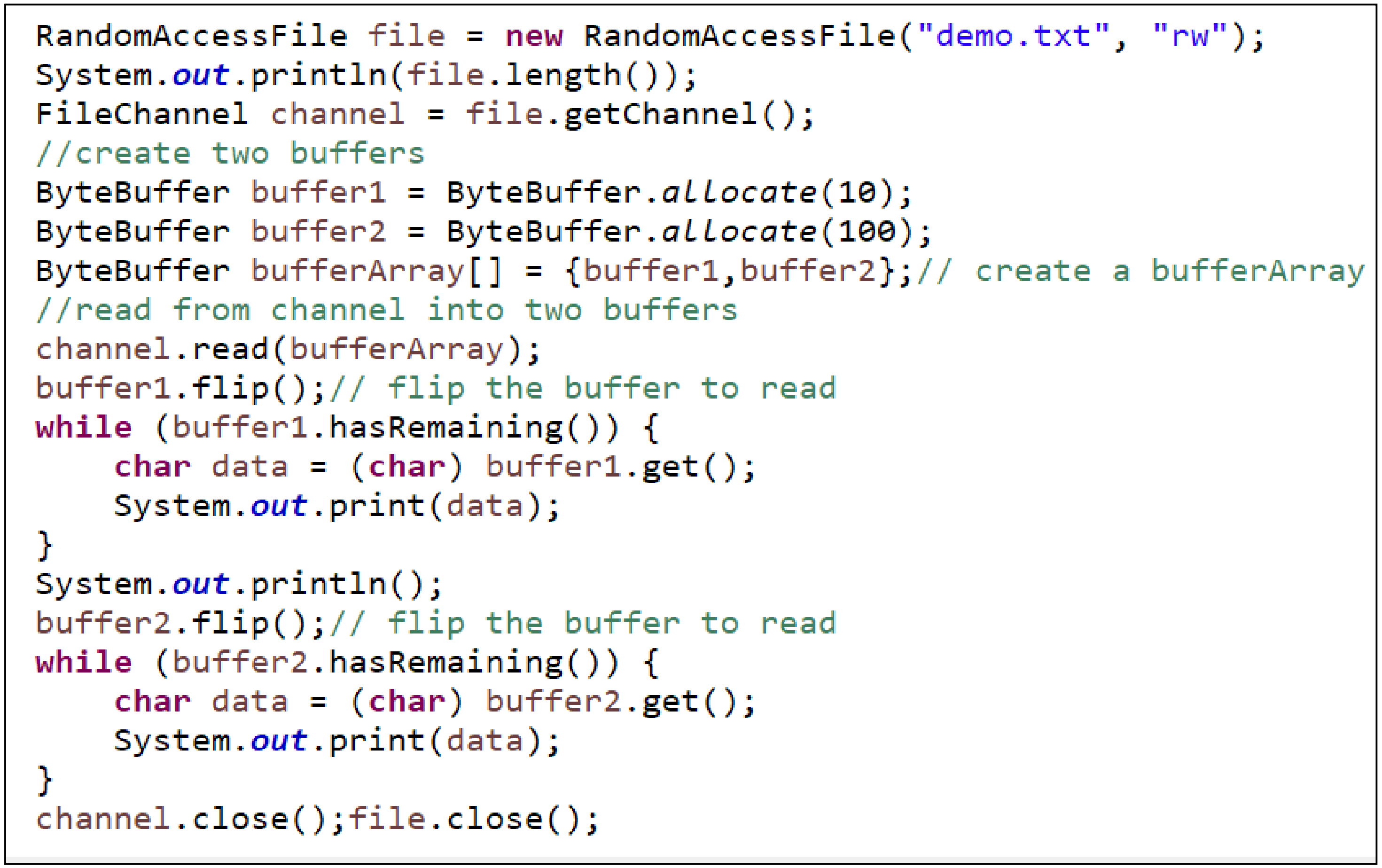
- Set position of buffer back to 0.
- Limit to capacity.
- Overwrites unread data.

Scattered Reads

- Read data from single channel to multiple buffers
- A message with header and body can be stored in two different buffers



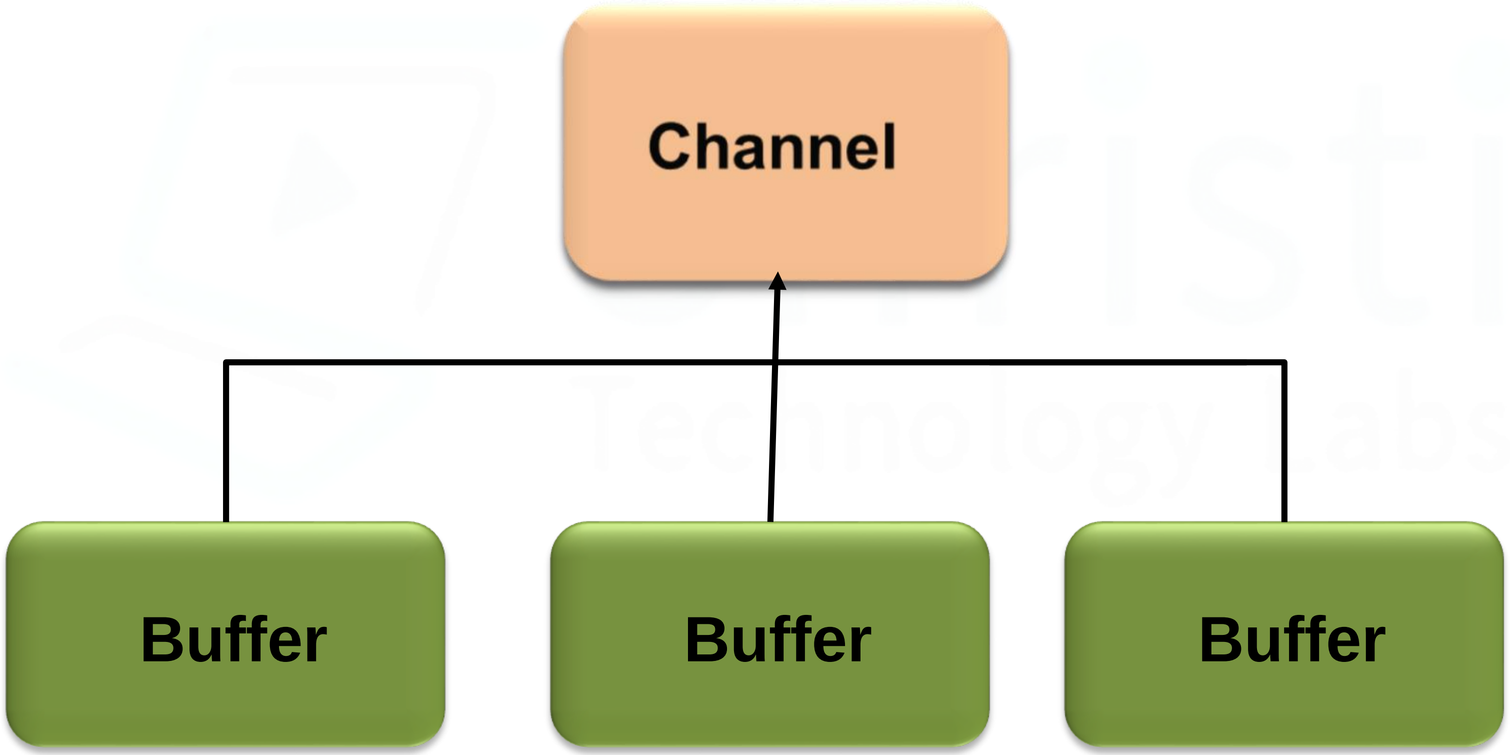
Example



```java
RandomAccessFile file = new RandomAccessFile("demo.txt", "rw");
System.out.println(file.length());
FileChannel channel = file.getChannel();
//create two buffers
ByteBuffer buffer1 = ByteBuffer.allocate(10);
ByteBuffer buffer2 = ByteBuffer.allocate(100);
ByteBuffer bufferArray[] = {buffer1,buffer2};// create a bufferArray
//read from channel into two buffers
channel.read(bufferArray);
buffer1.flip();// flip the buffer to read
while (buffer1.hasRemaining()) {
    char data = (char) buffer1.get();
    System.out.print(data);
}
System.out.println();
buffer2.flip();// flip the buffer to read
while (buffer2.hasRemaining()) {
    char data = (char) buffer2.get();
    System.out.print(data);
}
channel.close();file.close();
```

Gathering Writes

- Writes data to a single channel from multiple buffers.



Example

```
RandomAccessFile file = new RandomAccessFile("demo.txt", "rw");
System.out.println(file.length());
FileChannel channel = file.getChannel();
// create two buffers and add content to it
ByteBuffer buffer1 = ByteBuffer.allocate(100);
String msg1 = "This is in the header";
buffer1.put(msg1.getBytes());

ByteBuffer buffer2 = ByteBuffer.allocate(100);
String msg2 = "This is in the body";
buffer1.put(msg2.getBytes());

// create a bufferArray
ByteBuffer bufferArray[] = { buffer1, buffer2 };
buffer1.flip();// flip the buffer to read
//write date into channel from the bufferArray
channel.write(bufferArray);
channel.close();
file.close();
```

Channel to Channel transfer

- Data can be transferred from one channel to another channel without an intermediate buffer
- Used with file channels and is very fast

`transferFrom(ReadableByteChannel src, long position, long count) throws IOException`

- – Transfers bytes into this channel's file from the given readable byte channel.

`transferTo(long position, long count,WritableByteChannel) throws IOException`

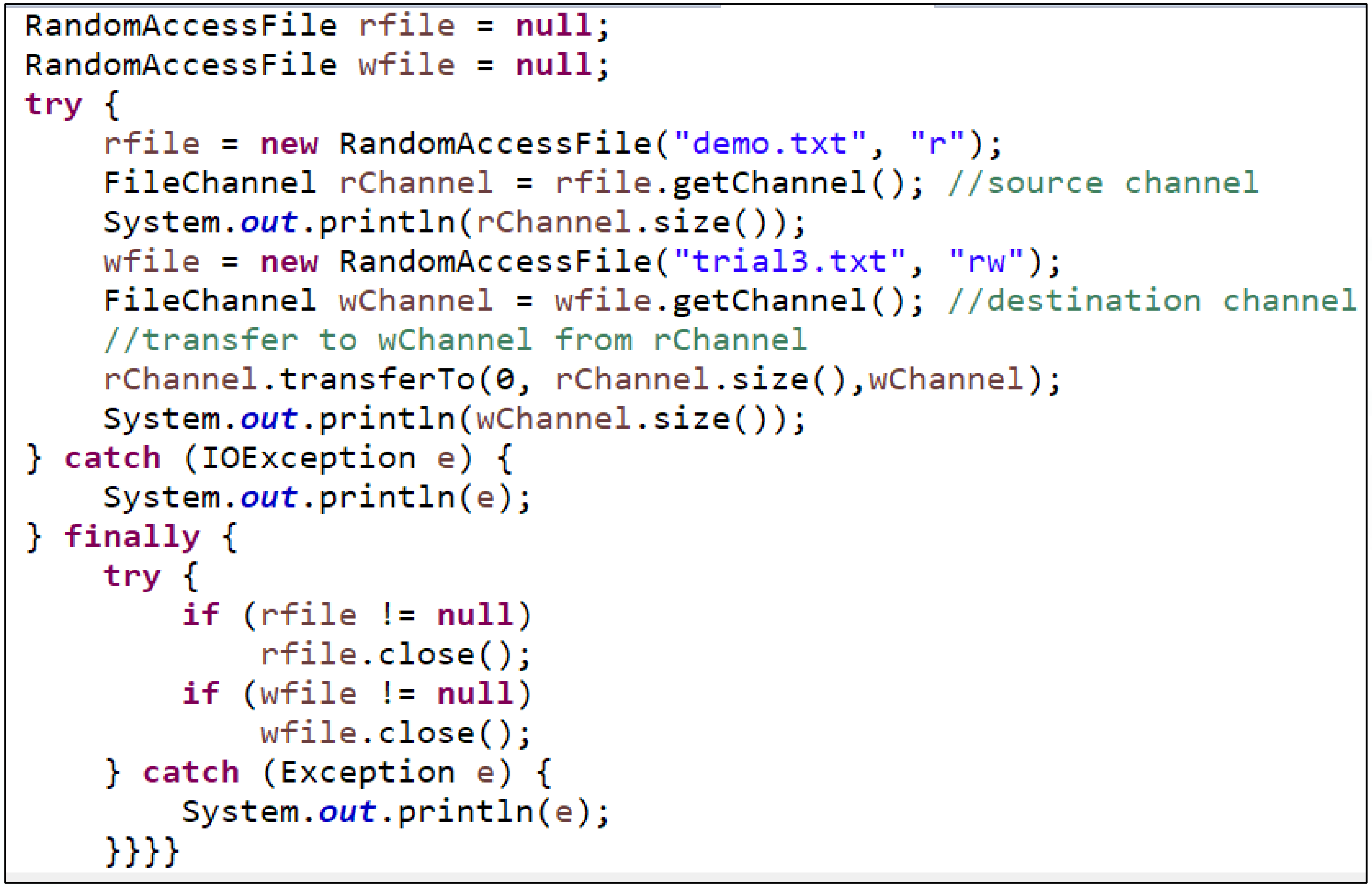
- – Transfers bytes from this channel's file to the given writable byte channel.

Shristi Technology Labs

Example - transferFrom

```
RandomAccessFile rfile = null;
RandomAccessFile wfile = null;
try {
    rfile = new RandomAccessFile("demo.txt", "r");
    FileChannel rChannel = rfile.getChannel(); //source channel
    System.out.println(rChannel.size());
    wfile = new RandomAccessFile("trial.txt", "rw");
    FileChannel wChannel = wfile.getChannel(); //destination channel
    //transfer from rChannel to wChannel
    wChannel.transferFrom(rChannel, 0, rChannel.size());
    System.out.println(wChannel.size());
} catch (IOException e) {
    System.out.println(e);
} finally {
    try {
        if (rfile != null)
            rfile.close();
        if (wfile != null)
            wfile.close();
    } catch (Exception e) { System.out.println(e);
    }}}}
```

Example - transferTo



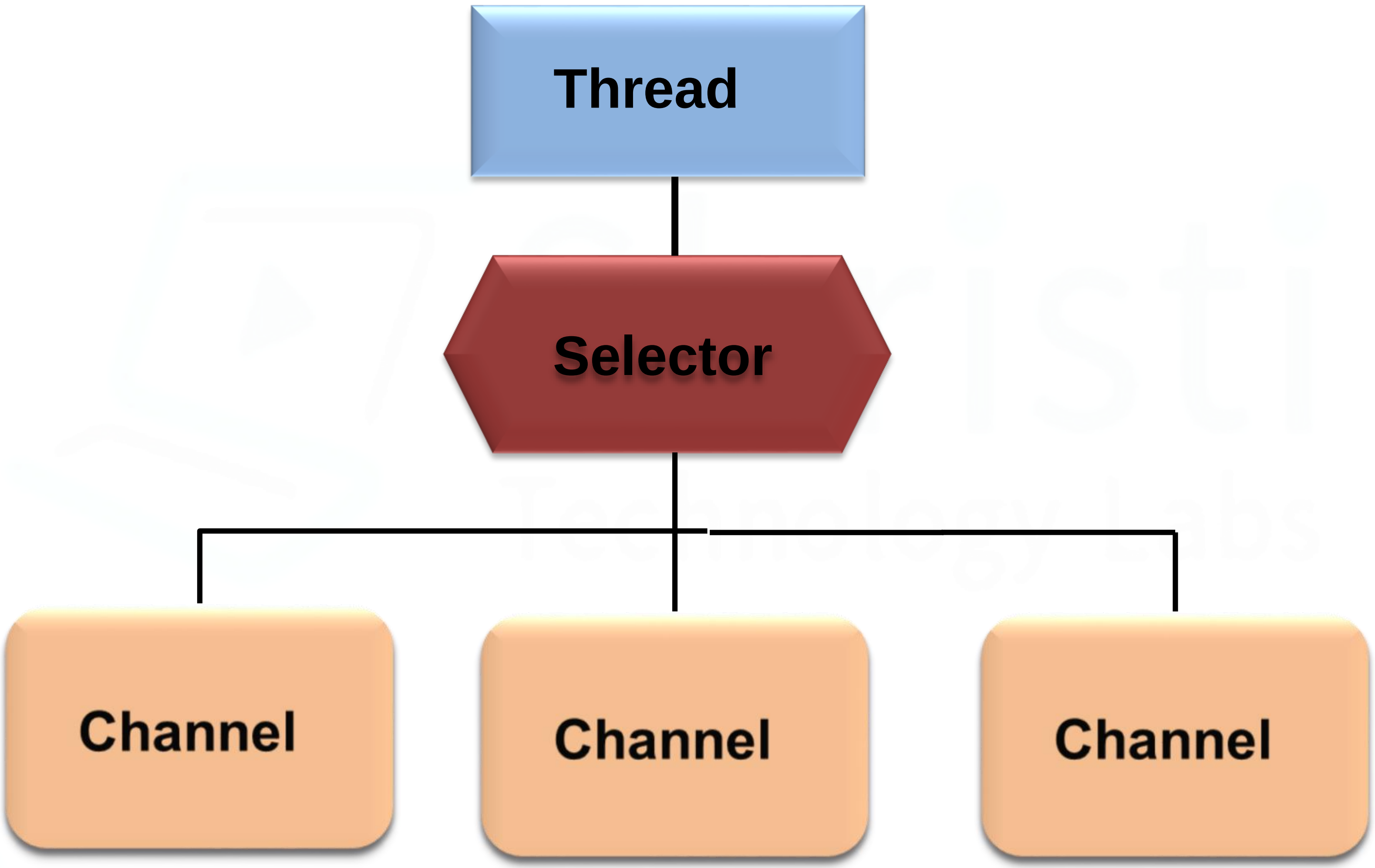
```java
RandomAccessFile rfile = null;
RandomAccessFile wfile = null;
try {
    rfile = new RandomAccessFile("demo.txt", "r");
    FileChannel rChannel = rfile.getChannel(); //source channel
    System.out.println(rChannel.size());
    wfile = new RandomAccessFile("trial3.txt", "rw");
    FileChannel wChannel = wfile.getChannel(); //destination channel
    //transfer to wChannel from rChannel
    rChannel.transferTo(0, rChannel.size(),wChannel);
    System.out.println(wChannel.size());
} catch (IOException e) {
    System.out.println(e);
} finally {
    try {
        if (rfile != null)
            rfile.close();
        if (wfile != null)
            wfile.close();
    } catch (Exception e) {
        System.out.println(e);
    }}}}
```

Selector

- Selector is used to check the readiness of Channels for reading or writing.

- Using Selector, a single thread can handle multiple channels.

Selector



Steps to work with Selector

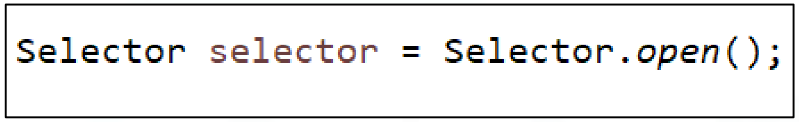
- Create a Selector
- Open the ServerSocketChannel – listens for incoming TCP connections
- Register the channel with the selector
- Check the channels for the readiness
- Perform the operation – read/write

- Create a client using SocketChannel – channel connected to TCP network socket

Steps

Create a selector

- Create the selector using open method

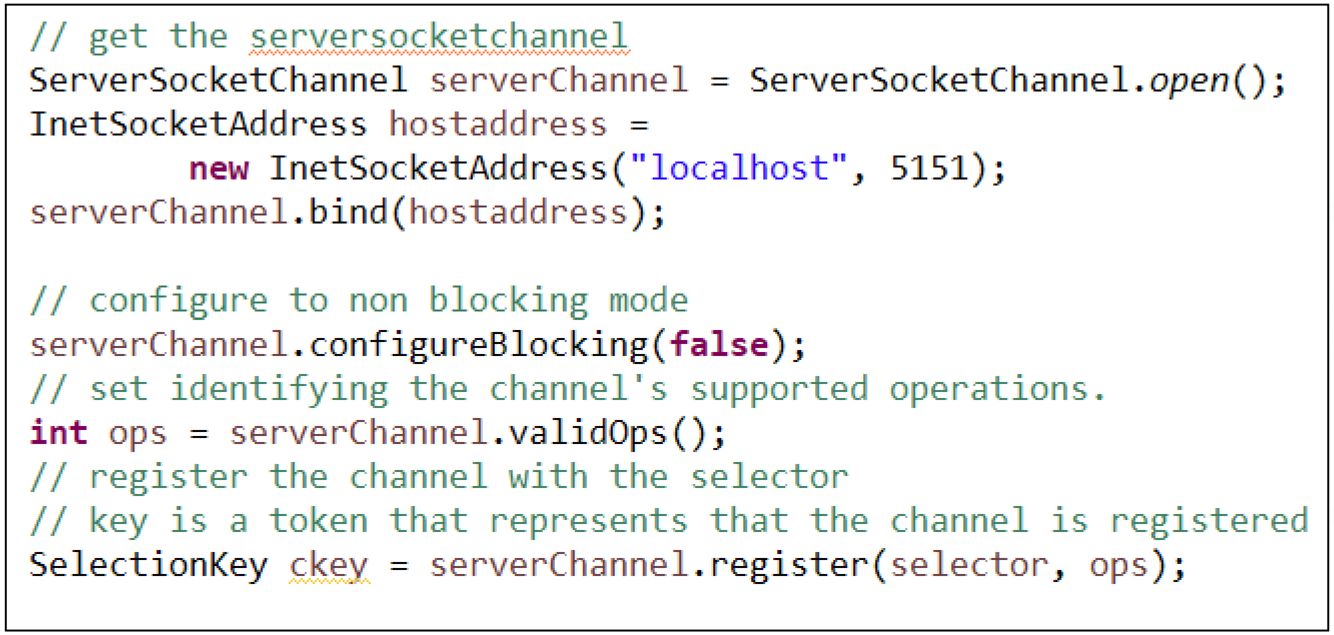


```
Selector selector = Selector.open();
```

Open and register the channel with the selector

- Open the channel and bind to the address
- The channel must be in non blocking mode to work with the selector
- The channel can be registered using register() method
- A selection key is created each time a channel is registered with a selector.
- The channel creates some events and the Selector listens to it

Open and Register



```java
// get the serversocketchannel
ServerSocketChannel serverChannel = ServerSocketChannel.open();
InetSocketAddress hostaddress =
        new InetSocketAddress("localhost", 5151);
serverChannel.bind(hostaddress);

// configure to non blocking mode
serverChannel.configureBlocking(false);
// set identifying the channel's supported operations.
int ops = serverChannel.validOps();
// register the channel with the selector
// key is a token that represents that the channel is registered
SelectionKey ckey = serverChannel.register(selector, ops);
```

Steps

Identify the events in the Channel, the Selector wants to listen

- There are four different events the Selector can listen for
  - **Connect** (A channel that connects to another server)
  - **Accept** (A channel accepts an incoming connection)
  - **Read** (A channel that has data ready to be read)
  - **Write** (A channel that is ready to accept data in it)
- The four events are represented by the four SelectionKey constants:
  - **SelectionKey.OP_CONNECT**
  - **SelectionKey.OP_ACCEPT**
  - **SelectionKey.OP_READ**
  - **SelectionKey.OP_WRITE**
- To use more than one event use the constants together

Steps

Select the Channel

- To get the channels that are ready for events, use **`select()`**

```
int readyChannels = selector.select();
```

- To access the channels that are ready use **`selectedKeys()`**.

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

Close the selector

- Use the close() method to close the selector
- Invalidates all SelectionKey instances registered with this Selector

Asynchronous FileChannel

- It is possible to read data from, and write data to files asynchronously using AsynchronousFileChannel.

- Create an AsynchronousFileChannel using open() method
- Read data from an AsynchronousFileChannel via
  - Future-
    - represents the result of an asynchronous computation.
  - Completion Handler
- Write data to an AsynchronousFileChannel via
  - Future
  - Completion Handler

Example – Read via Future

```
Path path = Paths.get("trial.txt");
AsynchronousFileChannel channel =
  AsynchronousFileChannel.open(path, StandardOpenOption.READ);
ByteBuffer buffer = ByteBuffer.allocate(100);

// read method returns even if operation not completed
Future<Integer> operation = channel.read(buffer, 0);

// check if read operation is completed
while (!operation.isDone()) {
    System.out.println("continue other work. Still reading ");
}
buffer.flip();
while (buffer.hasRemaining()) {
    char data = (char) buffer.get();
    System.out.print(data);
}
```

Example – Write via Future

```
Path path = Paths.get("trial.txt");
AsynchronousFileChannel channel =
        AsynchronousFileChannel.open(path, StandardOpenOption.WRITE);
ByteBuffer buffer = ByteBuffer.allocate(100);
String msg = "This is asynchronous";
buffer.put(msg.getBytes());
buffer.flip();

Future<Integer> operation = channel.write(buffer, 0);
// check if write operation is completed
while (!operation.isDone());
System.out.println("done");
```

CompletionHandler

- Is a handler for consuming the result of an asynchronous I/O operation.
- Is an interface from java.nio.channels package.

CompletionHandler<V,A>

  - V – The result type of the I/O operation
  - A – The type of the object attached to the I/O operation

Methods

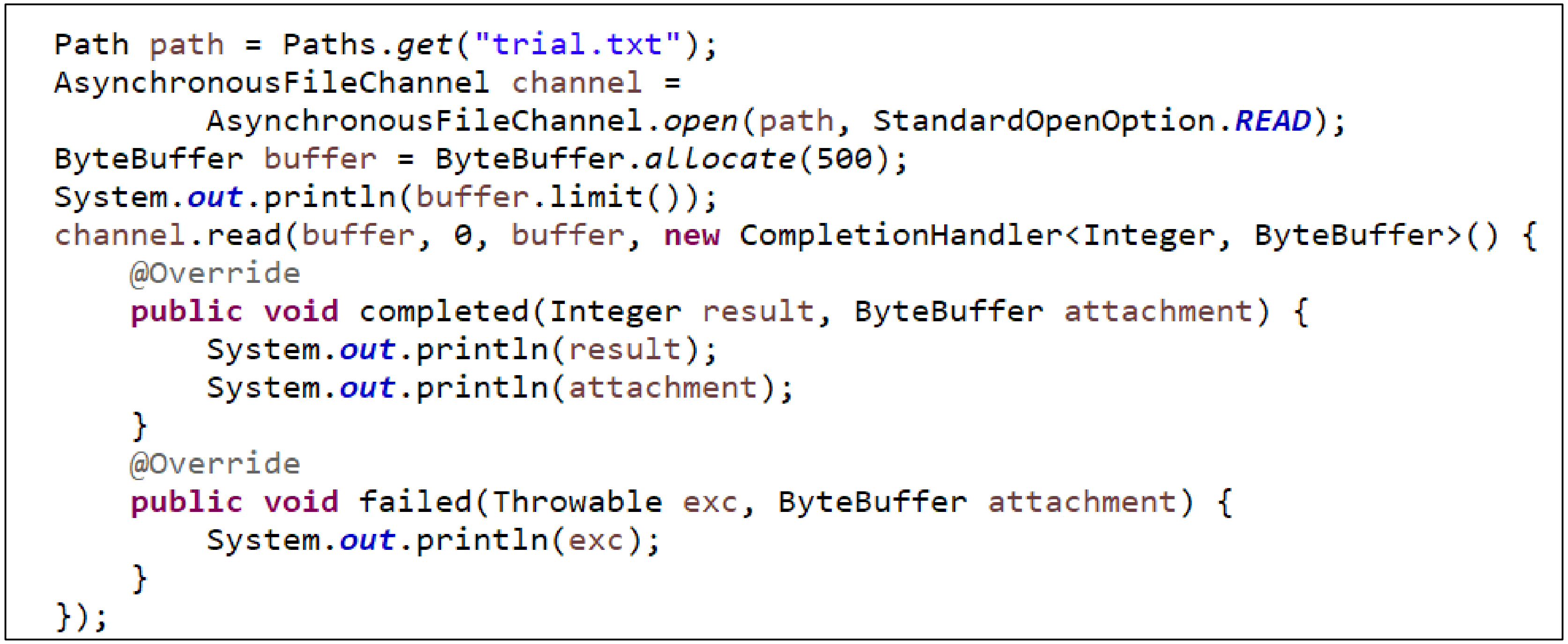
```
public void completed(V result, A attachement)
```

  - is invoked when the I/O operation completes successfully.

```
public void failed(Throwable ex, A attachement)
```

  - invoked if the I/O operations fails.

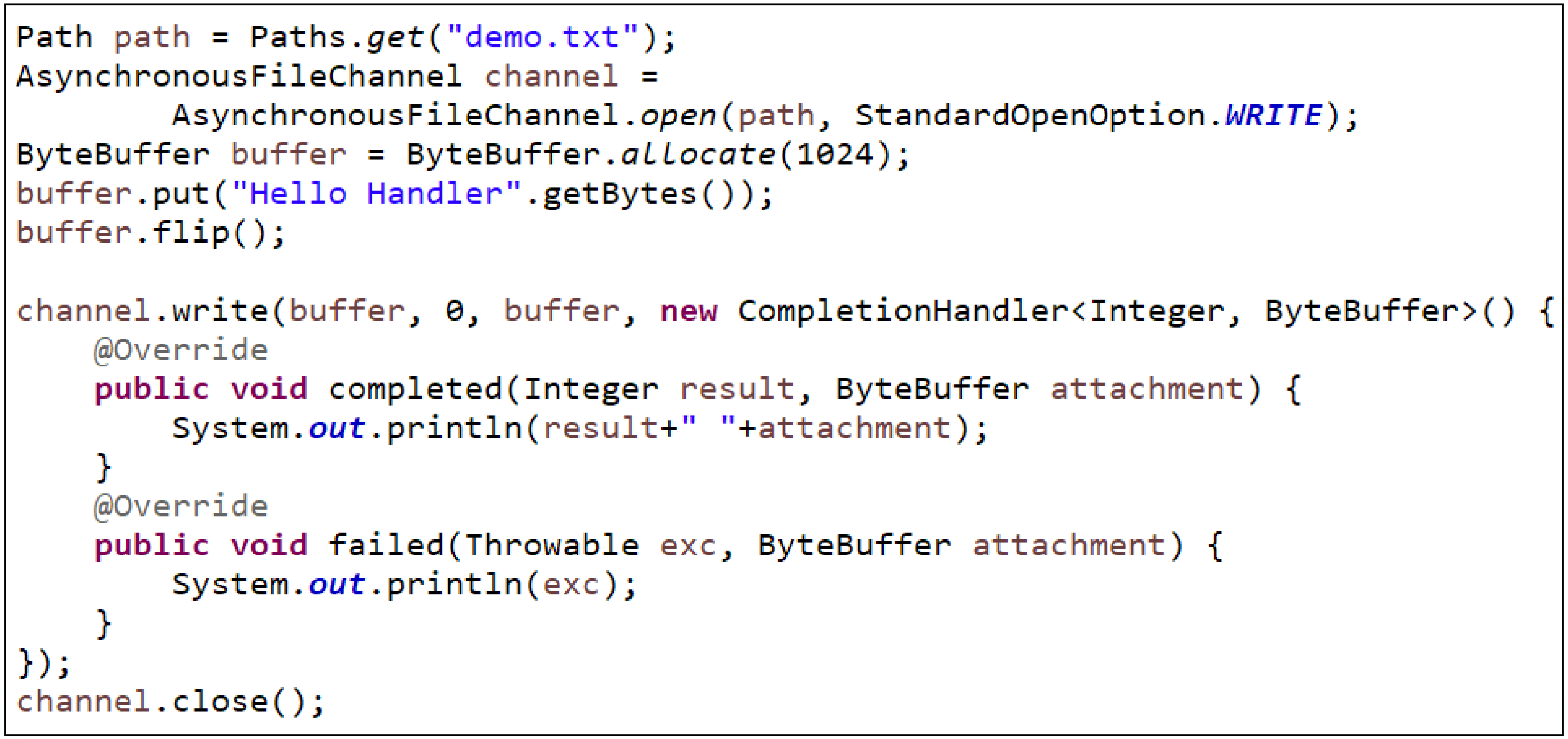
Example – Read via CompletionHandler



```java
Path path = Paths.get("trial.txt");
AsynchronousFileChannel channel =
        AsynchronousFileChannel.open(path, StandardOpenOption.READ);
ByteBuffer buffer = ByteBuffer.allocate(500);
System.out.println(buffer.limit());
channel.read(buffer, 0, buffer, new CompletionHandler<Integer, ByteBuffer>() {
    @Override
    public void completed(Integer result, ByteBuffer attachment) {
        System.out.println(result);
        System.out.println(attachment);
    }
    @Override
    public void failed(Throwable exc, ByteBuffer attachment) {
        System.out.println(exc);
    }
});
```

Shristi Technology Labs

Example – Write via CompletionHandler



```
Path path = Paths.get("demo.txt");
AsynchronousFileChannel channel =
        AsynchronousFileChannel.open(path, StandardOpenOption.WRITE);
ByteBuffer buffer = ByteBuffer.allocate(1024);
buffer.put("Hello Handler".getBytes());
buffer.flip();

channel.write(buffer, 0, buffer, new CompletionHandler<Integer, ByteBuffer>() {
    @Override
    public void completed(Integer result, ByteBuffer attachment) {
        System.out.println(result+" "+attachment);
    }
    @Override
    public void failed(Throwable exc, ByteBuffer attachment) {
        System.out.println(exc);
    }
});
channel.close();
```

Summary

- Overview of NIO
- Channels
- Buffers
- Selectors
- Channel ToChannel
- Asynchronous FileChannel

Thank you