UNIT 9
NON BLOCKING I/O

LH - 3HRS

PRESENTED BY: **ER. SHARAT MAHARJAN**

NETWORK PROGRAMMING

PRIME COLLEGE, NAYABAZAAR

CONTENT (LH - 3HRS)

9.1 An example client and server

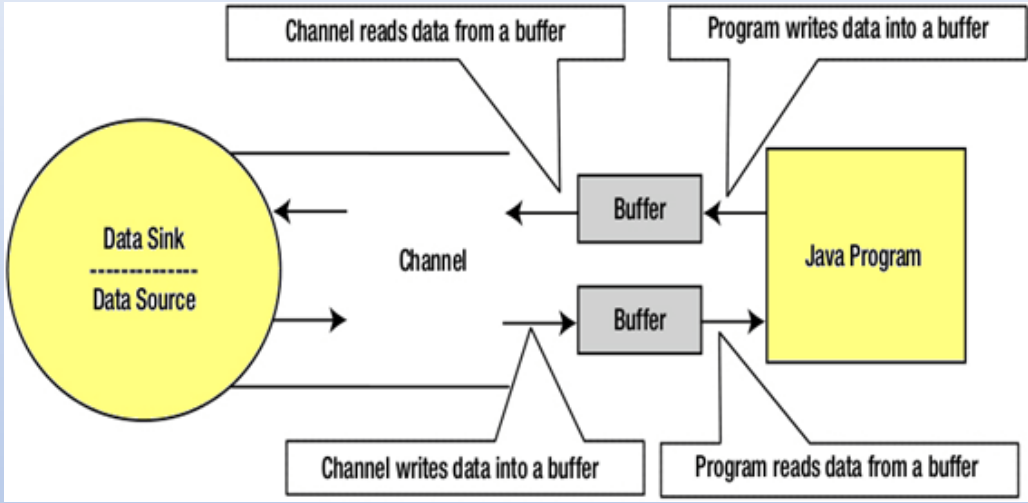
9.2 Buffers: Creating Buffers, filling and draining, bulk methods, data conversion, view buffers, compacting buffers, Duplicating buffers, Slicing Buffers, Marking and Resetting, Object Methods.

9.3 Channels: Sockets Channel, ServerSocketChannel, The Channels Class, Asynchronous Channels, Socket Options

9.4 Readiness Selection: The Selectors Class, The Selection Key Class

9.1 An example client and server

- Java NIO stands for Java New Input Output
- Java NIO operates inside java.nio package
- Java NIO is Buffer oriented
- Non-blocking IO operation
- Channels are available - buffer in and out
- Contains the concept of Selectors - handling multiple channels using a single thread
- The data is written into a buffer from which it is further processed using a channel.



9.2 Buffers

- Instead of writing data onto output streams and reading data from input streams, you read and write data from buffers.
- Before bytes can be read from or written to a channel, the bytes have to be stored in a buffer.
- Primary parameters that defines Java NIO buffer could be defined as:

a. **Capacity**: Maximum amount of data that can be stored in the Buffer.

b. **Limit**: In write mode, Limit is equal to the capacity that maximum data could be written in buffer. In read mode, Limit means the limit of how much data can be read from the Buffer.

c. **Position**: Points to the current location of cursor in buffer.

d. **Mark**: When mark() method is called the current position is recorded.

Creating Buffers

- The basic allocate( ) method simply returns a new, empty buffer with a specified fixed capacity.
- For example, the line create byte buffer with a size of 100:

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

- If using array, we call wrap() method as:

```
char [] myArray = new char[100];
CharBuffer charbuffer = CharBuffer.wrap(myArray);//capacity 100
```

Filling and Draining

- Buffer is filled using put() method as:

```
char [] myArray = new char[100];
CharBuffer buffer = CharBuffer.wrap(myArray);
buffer.put('H').put('e') .put('l') .put('l') .put('o');
```

- Before we can read the data out again, we need to flip the buffer:

```
buffer.flip( );
```

- This repositions the cursor at the start of the buffer.

- Buffer is drained using get() method and two methods hasRemaining() and remaining() are used to know buffer's limit as:

```
for(int i=0; buffer.hasRemaining(),  i++){
myArray[i] = buffer.get();
}

for(int i=0; buffer.remaining(),  i++){
myArray[i] = buffer.get();
}
```

Bulk methods

- Even with buffers it's often faster to work with blocks of data rather than filling and draining one element at a time.
- For example, CharBuffer has put( ) and get( ) methods that fill and drain a CharBuffer from a pre-existing char array:

public CharBuffer get(char [] dst)

public CharBuffer put(char [] src)

- The put methods throw a BufferOverflowException if the buffer does not have sufficient space for the array. The get methods throw a BufferUnderflowException if the buffer does not have enough data remaining to fill the array.

Compacting buffers:

- Occasionally, you may wish to drain some, but not all, of the data from a buffer, then resume filling it.
- To do this, the unread data need to be shifted down so that the first element is at index zero.
- While this could be inefficient if done repeatedly, it's occasionally necessary and the API provides a method compact():

buffer.compact();

- You can use a buffer in this way as a First In First Out (FIFO) queue.

Marking and Resetting

- Attribute ‘mark’ allows a buffer to remember a position and return to it later.
- When mark() method is called, the mark is set to the current position.

public final Buffer mark( )

public final Buffer reset( )

- The reset( ) method makes position back to marked position and throws an InvalidMarkException , a runtime exception, if the mark is not set.

Duplicating buffers

- It's often desirable to make a copy of a buffer to deliver the same information to two or more channels.
- The duplicate() method :

public abstract ByteBuffer duplicate( )

- The return values are not clones.
- The duplicated buffers share the same data, including the same backing array.
- Changes to the data in one buffer are reflected in the other buffer.

9.3 Channels

- Channels move blocks of data into and out of buffers to and from various I/O sources such as files, sockets, datagrams, and so forth.
- Implementations of Channel

a. FileChannel
b. DatagramChannel
c. SocketChannel
d. ServerSocketChannel

Sockets Channel:

- The SocketChannel class does not have any public constructors.
- Instead, you create a new SocketChannel object using static open( ) method :

public static SocketChannel open(SocketAddress remote) throws IOException

- For example:

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

ServerSocketChannel:

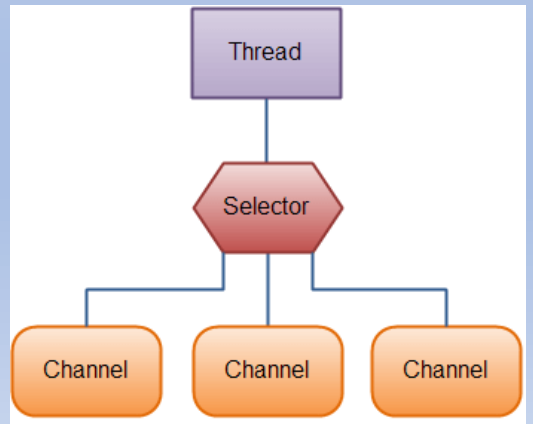
- The ServerSocketChannel class has one purpose: to accept incoming connections using accept() method.
- The static factory method ServerSocketChannel.open() creates a new ServerSocketChannel object.

ServerSocketChannel server= ServerSocketChannel.open( );

- Server Socket Channel is created by invoking open method but not yet bound.
- In order to bound socket channel bind() method is to be called on ServerSocket object.

Asynchronous Channels:

- Java NIO supports concurrency and multi-threading which allows us to deal with different channels concurrently at same time.
- The API which is responsible for this in Java NIO package is AsynchronousFileChannel which is defined under NIO channels package.
- AsynchronousFileChannel enables file operations to execute asynchronously unlike of synchronous I/O operation in which a thread enters into an action and waits until the request is completed.
- In asynchronous the request is passed by thread to the OS's kernel to get it done while thread continues to process another job.



9.4 Readiness Selection

- The only constructor in Selector is protected. Normally, a new selector is created by invoking the static factory method Selector. open ( ) :

public static Selector open( ) throws IOException

- The next step is to register channel with selector instance which returns an instance of SelectionKey.
- SelectionKey helps in knowing the state of channel.

- The major operations or state of channel represented by selection key are:

SelectionKey.OP_CONNECT - Channel which is ready to connect to server.

SelectionKey.OP_ACCEPT - Channel which is ready to accept incoming connections.

SelectionKey.OP_READ - Channel which is ready to data read.

SelectionKey.OP_WRITE - Channel which is ready to data write.

THANK YOU FOR YOUR ATTENTION