JAVA NIO

Topics to be covered

- Java NIO (NIO 2) Overview
- NIO classes: Fundamentals, Path Interfaces, Manage metadata of a file or directory,
- Byte Buffers & Channels
- UDP, TCP and IP
- Communication with TCP/IP Protocol

NIO 2 - Overview

- To support high performance and intensive I/O operations
- To complement the existing Standard I/O (in java.io package), not as a replacement
- non-blocking I/O
- more efficient buffer-based I/O

IO vs NIO

IO	NIO
It is based on the Blocking I/O operation	It is based on the Non-blocking I/O operation
It is Stream-oriented	It is Buffer-oriented
Channels are not available	Channels are available for Non-blocking I/O operation
Selectors are not available	Selectors are available for Non-blocking I/O operation

NIO Fundamentals

- **Channels and Buffers:**
  a. Standard I/O API=> **character streams and byte streams**
  b. NIO => **channels and buffers**. Data is always written from a buffer to a channel and read from a channel to a buffer.
- **Selectors:**
  a. It is an object that can be used for monitoring the multiple channels for events like data arrived, connection opened etc.
- **Non-blocking I/O:**
  a. The application returns immediately whatever the data available and application should have pooling mechanism to find out when more data is ready.

Path Class - import java.nio.file.Path;import java.nio.file.Paths

A Path instance contains the information used to specify the location of a file or directory.

- Path p1 = Paths.get("/tmp/foo");
- Path p2 = Paths.get(args[0]);
- Path p3 = Paths.get(URI.create("file:///Users/sakthis/FileTest.java"));

The `Paths.get` method is shorthand for the following code:

- Path p4 = FileSystems.getDefault().getPath("/users/sakthis");

The following example creates `/u/joe/logs/foo.log` assuming your home directory is `/u/joe`, or `C:\joe\logs\foo.log` if you are on Windows.

- Path p5 = Paths.get(System.getProperty("user.home"),"logs", "foo.log");

File Metadata

Methods	Comment
`size(Path)`	Returns the size of the specified file in bytes.
`isDirectory(Path, LinkOption)`	Returns true if the specified `Path` locates a file that is a directory.
`isRegularFile(Path, LinkOption...)`	Returns true if the specified `Path` locates a file that is a regular file.
`isSymbolicLink(Path)`	Returns true if the specified `Path` locates a file that is a symbolic link.
`isHidden(Path)`	Returns true if the specified `Path` locates a file that is considered hidden by the file system.
`getLastModifiedTime(Path, LinkOption...)` `setLastModifiedTime(Path, FileTime)`	Returns or sets the specified file's last modified time.
`getOwner(Path, LinkOption...)` `setOwner(Path, UserPrincipal)`	Returns or sets the owner of the file.
`getPosixFilePermissions(Path, LinkOption...)` `setPosixFilePermissions(Path,Set<PosixFilePerm)`	Returns or sets a file's POSIX file permissions.
`getAttribute(Path, String, LinkOption...)` `setAttribute(Path, String, Object, LinkOption)`	Returns or sets the value of a file attribute.

Channels

- A channel represents a connection to a physical I/O device (such as file, network socket, or even another program)
- It is similar to Standard I/O's stream, but a more platform-dependent version of stream. Because channels have a closer ties to the underlying platform, they can achieve better I/O throughput
    - **FileChannel**
    - **SocketChannel: support non-blocking connection for TCP socket**
    - **DatagramChannel: UDP Datagram-oriented socket**

Channel -> Object

- A Channel object can be obtained by calling the **getChannel()** method
- java.io.FileInputStream, java.io.FileOutputStream, java.io.RandomAccessFile, java.net.Socket, java.net.ServerSocket, java.net.DatagramSocket, and java.net.MulticastSocket

```
FileInputStream fis = new FileInputStream("in.dat");

FileChannel fc = fis.getChannel();

// or

 FileChannel fc = new FileInputStream("in.dat").getChannel();
```

- channel I/O reads/write a buffer at a time
- For FileChannel, data is transferred via a ByteBuffer object in read()/write() methods

```
public abstract int read(ByteBuffer dest)

public abstract int write(ByteBuffer source)
```

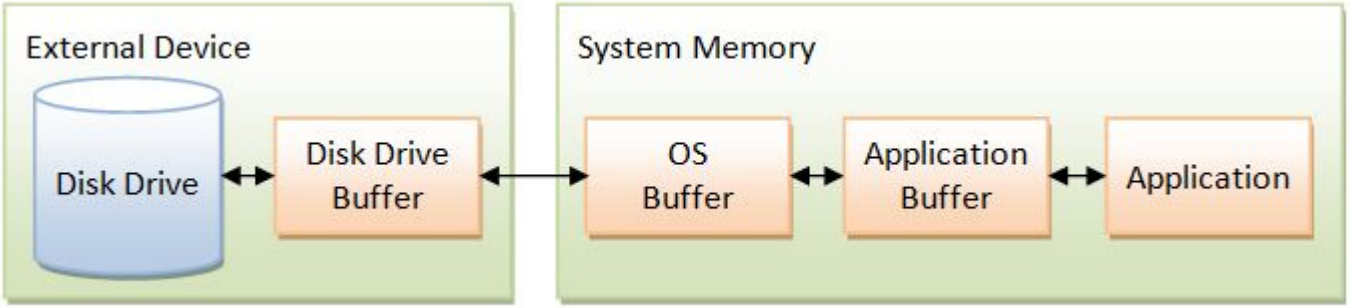
You can transfer data between an input channel and an output channel directly via:

```
public abstract long transferFrom(ReadableByteChannel source, long position, long count)

public abstract long transferTo(long position, long count, WritableByteChannel target)
```

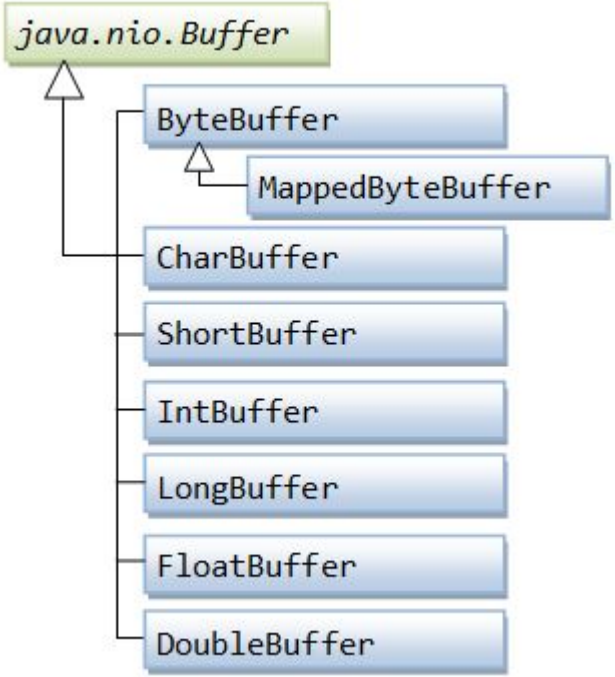
NIO Buffers

- Physical I/O operation is thousands times slower than memory access
- Hence, a chunk of data is often cache or buffer to improve the throughput
- NIO data transfer is through the so-called buffers implemented in java.nio.Buffer class



Buffer

- A Buffer is similar to an array, except that it is implemented much more efficiently by closely coupled with the underlying operating system
- A Buffer is a contiguous, linear storage
- Similar to an array, a Buffer has a fixed capacity
- A Buffer has a capacity, position, limit, and an optional mark



Buffer classes

ByteBuffer, CharBuffer, IntBuffer,ShortBuffer, LongBuffer, FloatBuffer and DoubleBuffer

Data Transfer (Get/Put): getXxx()/putXxx() methods to parse raw bytes into other primitive types

- clear(): sets the position to 0, limit to the capacity, and discards mark. It prepares the buffer for input.
- flip(): sets the limit to the current position, position to 0, and discard mark. Buffer populated and ready for output.
- rewind(): set the position to 0, and discard mark. It prepares the buffer for re-read.

Continued…

Allocating a Buffer:

```
ByteBuffer buf = ByteBuffer.allocate(28);

CharBuffer buf = CharBuffer.allocate(2048);
```

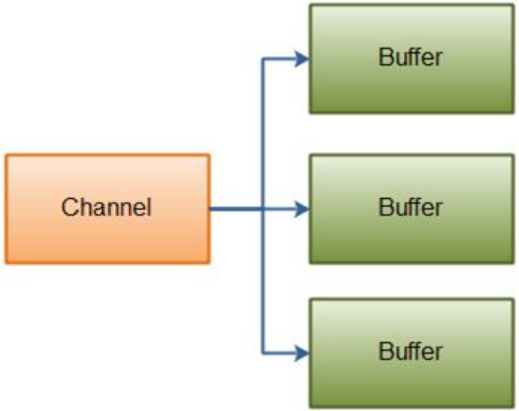
Reading Data from a Buffer

```
byte aByte = buf.get();

int bytesWritten = inChannel.write(buf);
```

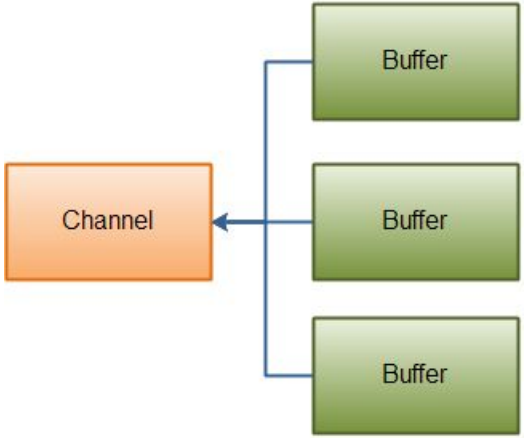
Scatter/Gather

Java NIO has inbuilt scatter/gather support. It can be used in reading from channels and writing to channels



The 'scattering read' is used for reading the data from a single channel into multiple

```
public interface ScatteringByteChannel extends ReadableByteChannel
{
    public long read (ByteBuffer [] argv) throws IOException;
    public long read (ByteBuffer [] argv, int length, int offset) throws IOException;
}
```



The 'gathering write' is used for writing the data from a multiple buffers into a single channel

```
public interface GatheringByteChannel extends WritableByteChannel
{
    public long write(ByteBuffer[] argv) throws IOException;
    public long write(ByteBuffer[] argv, int length, int offset) throws IOException;
}
```

Example

```
ByteBuffer buffer1 = ByteBuffer.allocate(8);
ByteBuffer buffer2= ByteBuffer.allocate(400);

buffer1.asIntBuffer().put(1024);
buffer2.asCharBuffer().put(data);

GatheringByteChannel gatherer = new
FileOutputStream("testout.txt").getChannel();
            gatherer.write(new ByteBuffer[] {
buffer1, buffer2 });
```

```
ScatteringByteChannel scatter = new
FileInputStream("testout.txt").getChannel();
            scatter.read(new ByteBuffer[] {
buffer1, buffer2 });
int bufferOne = buffer1.asIntBuffer().get();
        String bufferTwo =
buffer2.asCharBuffer().toString()
```

UDP, TCP and IP

- A Java NIO DatagramChannel is a channel that can send and receive UDP packets Since UDP is a connectionless network protocol
- Open
  - DatagramChannel channel = DatagramChannel.open();
  - channel.socket().bind(new InetSocketAddress(9999));
- Receive
  - ByteBuffer buf = ByteBuffer.allocate(48);
  - buf.clear();
  - channel.receive(buf);
- Send
  - String newData = "Hello”;
  - ByteBuffer buf = ByteBuffer.allocate(48);
  - buf.clear();
  - buf.put(newData.getBytes());
  - buf.flip();
  - int bytesSent = channel.send(buf, new InetSocketAddress("localhost", 80));

Java NIO SocketChannel

- NIO SocketChannel is used for connecting a channel with a TCP (Transmission Control Protocol) network socket
- Opening a SocketChannel
  - `SocketChannel sc = SocketChannel.open();`
  - `sc.connect(new InetSocketAddress("localhost", 70));`
- Reading from a SocketChannel
  - `ByteBuffer bb = ByteBuffer.allocate(84);`
  - `int bytesRead = SocketChannel.read(bb);`
- Writing to a SocketChannel
  - `String newData = "The new String is writing in a file ..." + System.currentTimeMillis();`
  - `ByteBuffer bb= ByteBuffer.allocate(48);`
  - `bb.clear();`
  - `bb.put(newData.getBytes());`
  - `bb.flip();`
  - `while(bb.hasRemaining()) {  SocketChannel.write(bb);  }`

Java NIO ServerSocketChannel

- Opening a ServerSocketChannel
  - ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();
- Listening for Incoming Connections
  - while(true){ SocketChannel socketChannel = serverSocketChannel.accept(); }
  - Non-blocking Mode
  - serverSocketChannel.socket().bind(new InetSocketAddress(9999));
  - **serverSocketChannel.configureBlocking(false);**
- Reading from a ServerSocketChannel
  - ByteBuffer bb = ByteBuffer.allocate(84);
  - int bytesRead = ServerSocketChannel.read(bb);
- Writing to a ServerSocketChannel
  - String newData = "The new String is writing in a file ..." + System.currentTimeMillis();
  - ByteBuffer bb= ByteBuffer.allocate(48);
  - bb.clear();
  - bb.put(newData.getBytes());
  - bb.flip();
  - while(bb.hasRemaining()) { ServerSocketChannel.write(bb); }