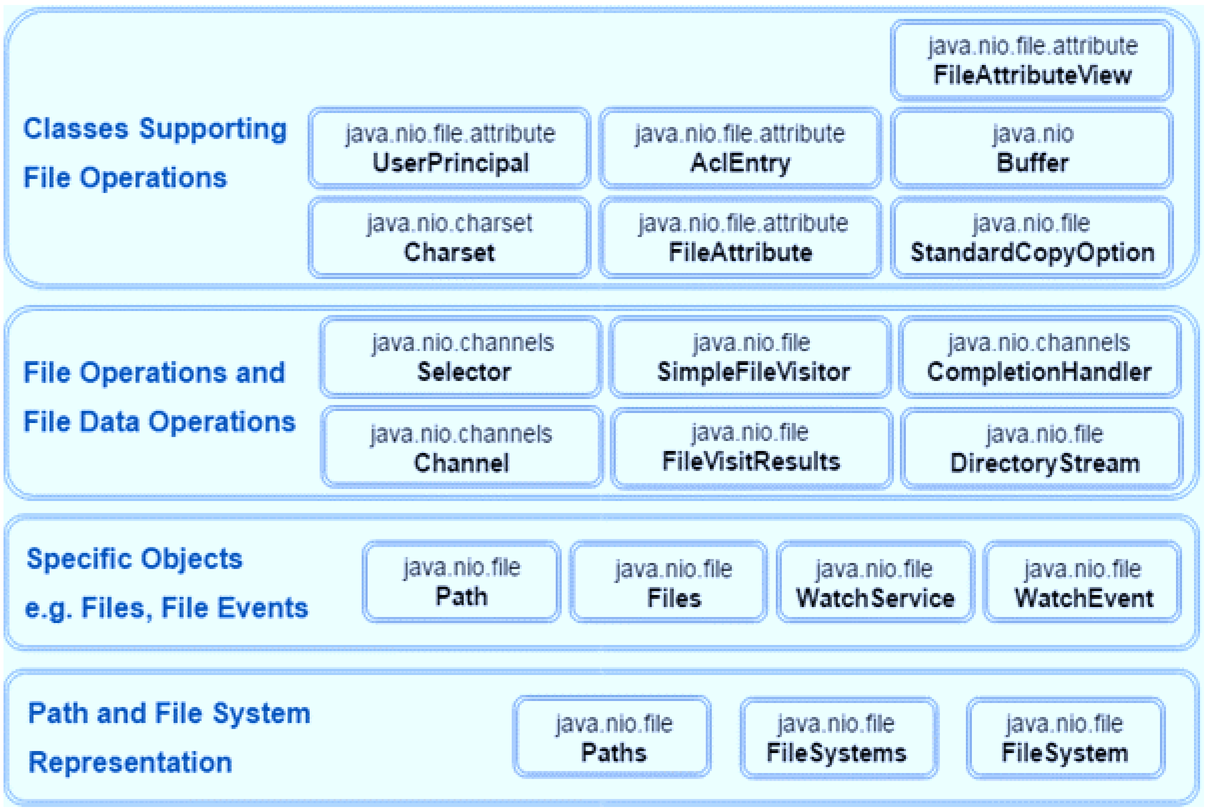
Java NIO Package

- Java NIO (New IO) was introduced from JDK 4 to implement high-speed IO operations. It is an alternative to the standard IO API's.
- Java NIO(New Input/Output) is high-performance networking and file handling API
- It is important to understand that the NIO subsystem, it does not replace the stream based I/O classes available in java.io package. The good working knowledge of java.io is helpful for understanding NIO.
- The important NIO classes grouped under different categories are shown below:


Classes Supporting File Operations
java.nio.file.attribute FileAttributeView
java.nio.file.attribute UserPrincipal
java.nio.file.attribute AclEntry
java.nio Buffer
java.nio.charset Charset
java.nio.file.attribute FileAttribute
java.nio.file StandardCopyOption
File Operations and File Data Operations
java.nio.channels Selector
java.nio.file SimpleFileVisitor
java.nio.channels CompletionHandler
java.nio.channels Channel
java.nio.file FileVisitResults
java.nio.file DirectoryStream
Specific Objects e.g. Files, File Events
java.nio.file Path
java.nio.file Files
java.nio.file WatchService
java.nio.file WatchEvent
Path and File System Representation
java.nio.file Paths
java.nio.file FileSystems
java.nio.file FileSystem

- The NIO classes are contained in the packages as given below:

Package	Purpose
java.nio	It is top-level package for NIO system. The various types of buffers are encapsulated by this NIO system.
java.nio.charset	It encapsulates the character sets and also supports encoders and decoders operation that convert characters to bytes and bytes to characters, respectively.
java.nio.charset.spi	It supports the service provider for character sets.
java.nio.channels	It support the channel, which are essentially open the I/O connections.
java.nio.channels.spi	It supports the service providers for channels.
java.nio.file	It provides the support for files.
java.nio.file.spi	It supports the service providers for file system.
java.nio.file.attribute	It provides the support for file attributes.

- Check the Oracle Document reference for more details: https://docs.oracle.com/javase/8/docs/api/java/nio/package-summary.html

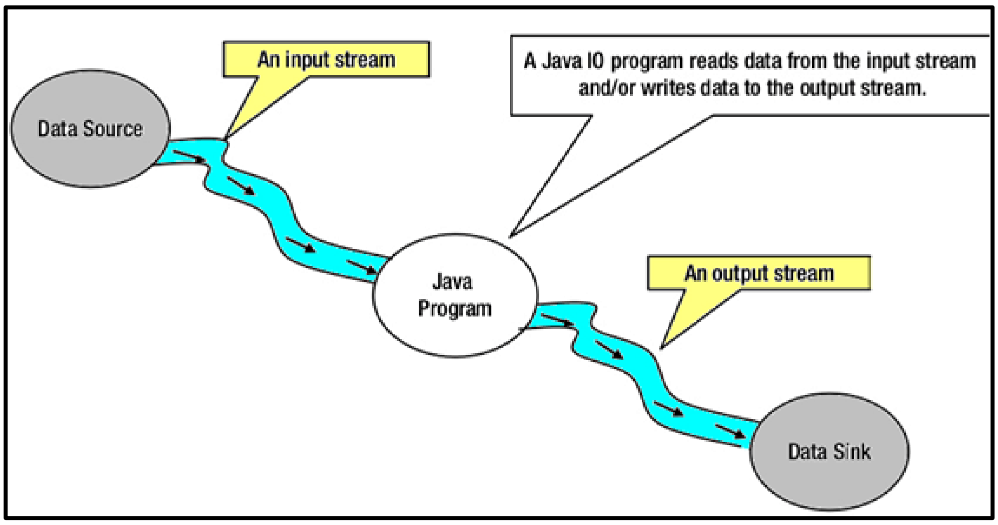
Java IO vs. NIO

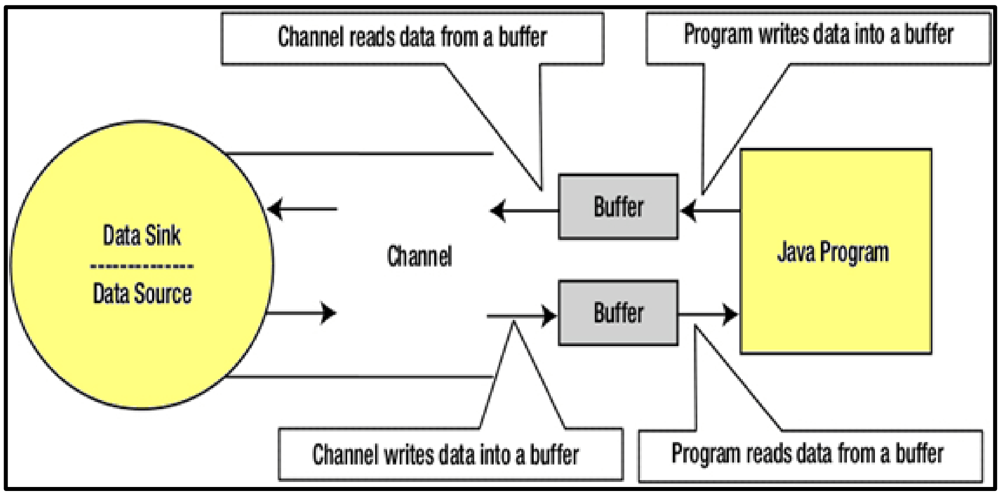
IO	NIO
It is based on the Blocking I/O operation	It is based on the Non-blocking I/O operation
It is Stream-oriented	It is Buffer-oriented
Channels are not available	Channels are available for Non-blocking I/O operation
Selectors are not available	Selectors are available for Non-blocking I/O operation

- Blocking I/O: Blocking IO wait for the data to be write or read before returning. Java IO's various streams are blocking. It means when the thread invoke a write() or read(), then the thread is blocked until there is some data available for read, or the data is fully written.
- Non blocking I/O: Non blocking IO does not wait for the data to be read or write before returning. Java NIO non- blocking mode allows the thread to request writing data to a channel, but not wait for it to be fully written. The thread is allowed to go on and do something else in a mean time.

Stream Oriented vs. Buffer Oriented

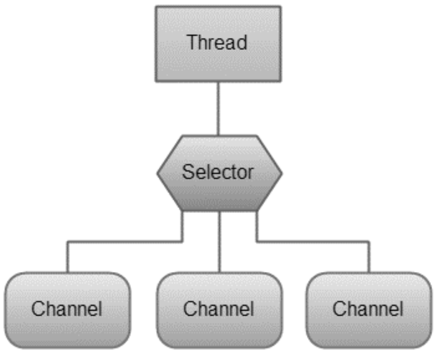
- Stream Oriented: Java IO is stream oriented I/O means we need to read one or more bytes at a time from a stream. It uses streams for transferring the data between a data source/sink and a java program. The I/O operation using this approach is slow.
- Buffer Oriented: Java NIO is buffer oriented I/O approach. Data is read into a buffer from which it is further processed using a channel. In NIO we deal with the channel and buffer for I/O operation.
- The major difference between a channel and a stream is: A stream can be used for one-way data transfer and a channel provides a two-way data transfer facility.

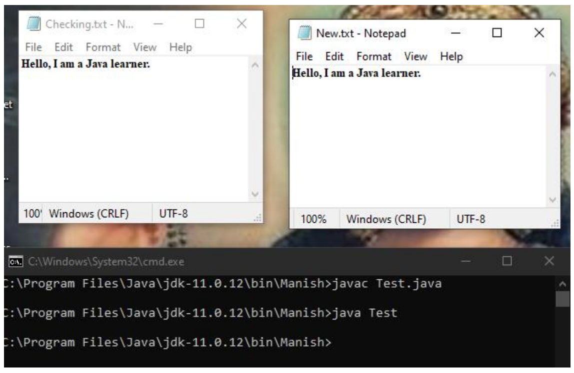

An input stream
A Java IO program reads data from the input stream and/or writes data to the output stream.
Data Source
Java Program
An output stream
Data Sink


Channel reads data from a buffer
Program writes data into a buffer
Data Sink
Data Source
Channel
Buffer
Buffer
Java Program
Channel writes data into a buffer
Program reads data from a buffer

Channels

- In Java NIO, the channel is a medium that transports the data efficiently between the entity and byte buffers. It reads the data from an entity and places it inside buffer blocks for consumption.
- Channels act as gateway provided by java NIO to access the I/O mechanism. Usually, channels have one-to-one relationship with operating system file descriptor for providing the platform independence operational feature.
- NIO Channel Basics: Channel implementation uses the native code to perform actual work. The channel interface allows us to gain access to low-level I/O services in a portable and controlled way.
- **Selectors:** In Java NIO the selector is a multiplexor of selectable channels, which is used as a special type of channel that can be put into non-blocking mode. It can examine one or more NIO Channel's and determines which channel is ready for communication i.e., reading or writing.
- The selector is used for handling the multiple channels using a single thread. Therefore, it require less threads to handle the channels. Switching between the threads is expensive for operating system. Therefore, for improving the system efficiency selector is use.




Checking.txt - N...
File Edit Format View Help
100' Windows (CRLF) UTF-8
New.txt - Notepad
File Edit Format View Help
100% Windows (CRLF) UTF-8
C:\Windows\System32\cmd.exe

Hello, I am a Java learner.

Hello, I am a Java learner.

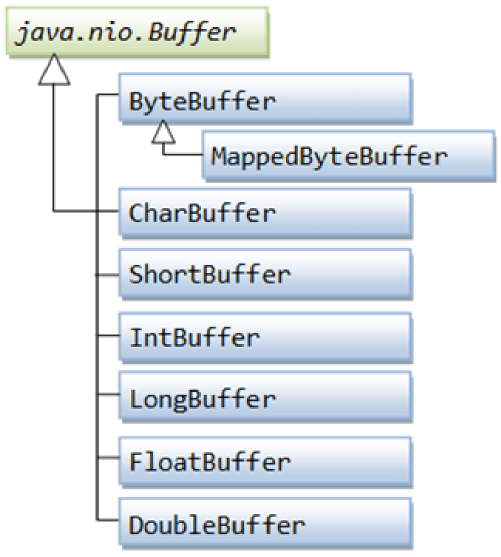
```
:\Program Files\Java\jdk-11.0.12\bin\Manish>javac Test.java

:\Program Files\Java\jdk-11.0.12\bin\Manish>java Test

:\Program Files\Java\jdk-11.0.12\bin\Manish>
```

- The example of copy the data from one channel to another channel or from one file to another file:

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.ReadableByteChannel;
import java.nio.channels.WritableByteChannel;
public class Test
{
    public static void main(String args[]) throws IOException
    {
            FileInputStream input = new FileInputStream ("Checking.txt");
                    ReadableByteChannel source = input.getChannel();
            FileOutputStream output = new FileOutputStream ("New.txt");
                    WritableByteChannel destination = output.getChannel();
            copyData(source, destination);
            source.close();
                    destination.close();
    }
    private static void copyData(ReadableByteChannel src, WritableByteChannel dest) throws IOException
    {
                    ByteBuffer buffer = ByteBuffer.allocateDirect(20 * 1024);
            while (src.read(buffer) != -1)
                    {
                    buffer.flip();
                            while (buffer.hasRemaining())
                            {
                                    dest.write(buffer);
                            }
                            buffer.clear();
             }
     }  }
```

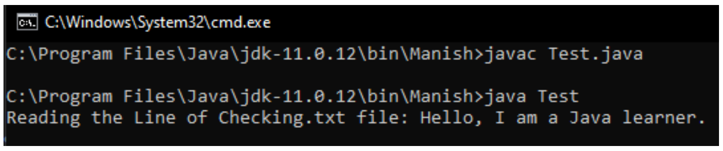
Buffers

- Buffers are defined inside java.nio package. It defines the core functionality which is common to all buffers: limit, capacity and current position.
- Java NIO buffers are used for interacting with NIO channels. It is the block of memory into which we can write data, which we can later be read again. The memory block is wrapped with a NIO buffer object, which provides easier methods to work with the memory block.
- Types of Buffer: For every primitive type there is a buffer type and all buffer classes can implement the buffer interface.



- The simple example of reading the line from Checking.txt file using the BufferedReader:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
public class Test
{
    public static void main(String[] args)
    {
                Path file = null;
                BufferedReader bufferedReader = null;
                try
            {
                    file = Paths.get("Checking.txt");
                    InputStream inputStream = Files.newInputStream(file);
                    bufferedReader = new BufferedReader(new InputStreamReader(inputStream));
                    System.out.println("Reading the Line of Checking.txt file: "+ bufferedReader.readLine());
                }
            catch (IOException e)
            {
                    e.printStackTrace();
                }
            finally
            {
                    try
                {
                        bufferedReader.close();
                    }
                catch (IOException ioe)
                {
                        ioe.printStackTrace();
                    }
                }
    }
 }
```



```
C:\Windows\System32\cmd.exe
C:\Program Files\Java\jdk-11.0.12\bin\Manish>javac Test.java

C:\Program Files\Java\jdk-11.0.12\bin\Manish>java Test
Reading the Line of Checking.txt file: Hello, I am a Java learner.
```