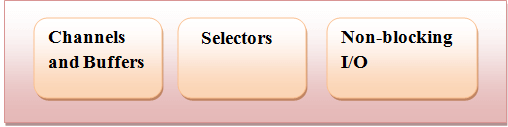
# Java NIO Tutorial

Java has provided a second I/O system called NIO (New I/O). Java NIO provides the different way of working with I/O than the standard I/O API's. It is an alternate I/O API for Java (from Java 1.4).

It supports a buffer-oriented, channel based approach for I/O operations. With the introduction of JDK 7, the NIO system is expanded, providing the enhanced support for file system features and file-handling. Due to the capabilities supported by the NIO file classes, NIO is widely used in file handling.

NIO was developed to allow Java programmers to implement high-speed I/O without using the custom native code. NIO moves the time-taking I/O activities like filling, namely and draining buffers, etc back into the operating system, thus allows for great increase in operational speed.

**Java NIO fundamental components are given below:**



ADVERTISEMENT

* **Channels and Buffers:** In standard I/O API the character streams and byte streams are used. In NIO we work with channels and buffers. Data is always written from a buffer to a channel and read from a channel to a buffer.
* **Selectors:** Java NIO provides the concept of "selectors". It is an object that can be used for monitoring the multiple channels for events like data arrived, connection opened etc. Therefore single thread can monitor the multiple channels for data.
* **Non-blocking I/O:** Java NIO provides the feature of Non-blocking I/O. Here the application returns immediately whatever the data available and application should have pooling mechanism to find out when more data is ready.Let's see the Application and Operating system interface for Non-blocking I/O:

# Java NIO Components

In Java NIO reading and writing are the fundamental process of I/O. Reading from channel: We can create a buffer and then ask a channel to read the data. Writing from channel: We can create a buffer, fill it with data and ask a channel to write the data.

The core components used in the reading and writing operation are:

* Channels
* Buffers
* Selectors

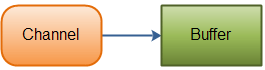
Java NIO has more components and classes than these, but the Channel, Buffer and Selector uses as the core of the API.

## Channels and Buffers

In standard I/O API the character streams and byte streams are used. In NIO we work with channels and buffers. All the I/O in NIO is started with a channel. Data is always written from a buffer to a channel and read from a channel to a buffer.

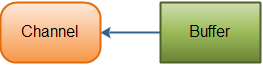
**Data reading operation:**

Let's see the channels read data into buffers illustration shown below:



**Data writing operation:**

Let's see the buffers write data into channels illustration shown below:



## Channels List

In Java NIO the primary Channels used are given below:

* DatagramChannel
* SocketChannel
* FileChannel
* ServerSocketChannel

The above channels cover the UDP (User Datagram Protocol)+TCP(Transmission Control Protocol) network I/O, and file I/O.

## Buffers List

In Java NIO the core Buffer used are given below:

* CharBuffer
* DoubleBuffer
* IntBuffer
* LongBuffer
* ByteBuffer
* ShortBuffer
* FloatBuffer

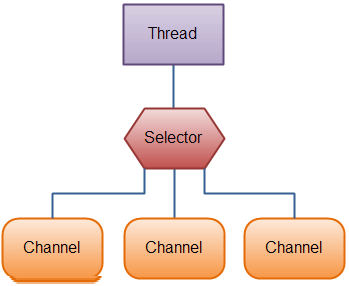
The above buffer's cover the basic data types that we can send via I/O: characters, double, int, long, byte, short and float.

## Selectors

Java NIO provides the concept of "selectors". It is an object that can be used for monitoring the multiple channels for events like data arrived, connection opened etc. Therefore single thread can monitor the multiple channels for data.

It is used if the application has many Channels (connections) open, but has low traffic on each connection. For example: In a chat server.

Let's see the thread using a Selector to handle the 3 Channel's illustration shown below:



# **Java NIO - Channels**

[Previous](https://www.tutorialspoint.com/java_nio/java_nio_vs_io.htm)

[Next](https://www.tutorialspoint.com/java_nio/java_nio_file_channel.htm)

## Description

As name suggests channel is used as mean of data flow from one end to other.Here in java NIO channel act same between buffer and an entity at other end in other words channel are use to read data to buffer and also write data from buffer.

Unlike from streams which are used in conventional Java IO channels are two way i.e can read as well as write.Java NIO channel supports asynchronous flow of data both in blocking and non blocking mode.

## Implementations of Channel

Java NIO channel is implemented primarily in following classes −

* **FileChannel** − In order to read data from file we uses file channel. Object of file channel can be created only by calling the getChannel() method on file object as we can't create file object directly.
* **DatagramChannel** − The datagram channel can read and write the data over the network via UDP (User Datagram Protocol).Object of DataGramchannel can be created using factory methods.
* **SocketChannel** − The SocketChannel channel can read and write the data over the network via TCP (Transmission Control Protocol). It also uses the factory methods for creating the new object.
* **ServerSocketChannel** − The ServerSocketChannel read and write the data over TCP connections, same as a web server. For every incoming connection a SocketChannel is created.

## Example

Following example reads from a text file from **C:/Test/temp.txt** and prints the content to the console.

### temp.txt

Hello World!

### ChannelDemo.java

import java.io.IOException;

import java.io.RandomAccessFile;

import java.nio.ByteBuffer;

import java.nio.channels.FileChannel;

public class ChannelDemo {

public static void main(String args[]) throws IOException {

RandomAccessFile file = new RandomAccessFile("C:/Test/temp.txt", "r");

FileChannel fileChannel = file.getChannel();

ByteBuffer byteBuffer = ByteBuffer.allocate(512);

while (fileChannel.read(byteBuffer) > 0) {

// flip the buffer to prepare for get operation

byteBuffer.flip();

while (byteBuffer.hasRemaining()) {

System.out.print((char) byteBuffer.get());

}

}

file.close();

}

}

## Output

Hello World!

# **Java NIO - Socket Channel**

[Previous](https://www.tutorialspoint.com/java_nio/java_nio_datagram_channel.htm)

[Next](https://www.tutorialspoint.com/java_nio/java_nio_server_socket_channel.htm)

Java NIO socket channel is a selectable type channel which means it can be multiplexed using selector, used for stream oriented data flow connecting sockets.Socket channel can be created by invoking its static **open()** method,providing any pre-existing socket is not already present.Socket channel is created by invoking open method but not yet connected.In order to connect socket channel **connect()** method is to be called.One point to be mentioned here is if channel is not connected and any I/O operation is tried to be attempted then NotYetConnectedException is thrown by this channel.So one must be ensure that channel is connected before performing any IO operation.Once channel is get connected,it remains connected until it is closed.The state of socket channel may be determined by invoking its **isConnected** method.

The connection of socket channel could be finished by invoking its **finishConnect()** method.Whether or not a connection operation is in progress may be determined by invoking the isConnectionPending method.By default socket channel supports non-blocking connection.Also it support asynchronous shutdown, which is similar to the asynchronous close operation specified in the Channel class.

Socket channels are safe for use by multiple concurrent threads. They support concurrent reading and writing, though at most one thread may be reading and at most one thread may be writing at any given time. The connect and finishConnect methods are mutually synchronized against each other, and an attempt to initiate a read or write operation while an invocation of one of these methods is in progress will block until that invocation is complete.

## Important methods of Socket channel

* **bind(SocketAddress local)** − This method is used to bind the socket channel to the local address which is provided as the parameter to this method.
* **connect(SocketAddress remote)** − This method is used to connect the socket to the remote address.
* **finishConnect()** − This method is used to finishes the process of connecting a socket channel.
* **getRemoteAddress()** − This method return the address of remote location to which the channel's socket is connected.
* **isConnected()** − As already mentioned this method returns the status of connection of socket channel i.e whether it is connected or not.
* **open() and open((SocketAddress remote)** − Open method is used open a socket channel for no specified address while parameterized open method open channel for specified remote address and also connects to it.This convenience method works as if by invoking the open() method, invoking the connect method upon the resulting socket channel, passing it remote, and then returning that channel.
* **read(ByteBuffer dst)** − This method is used to read data from the given buffer through socket channel.
* **isConnectionPending()** − This method tells whether or not a connection operation is in progress on this channel.

# **Java NIO - ServerSocket Channel**

[Previous](https://www.tutorialspoint.com/java_nio/java_nio_socket_channel.htm)

[Next](https://www.tutorialspoint.com/java_nio/java_nio_scatter.htm)

Java NIO server socket channel is again a selectable type channel used for stream oriented data flow connecting sockets.Server Socket channel can be created by invoking its static **open()** method,providing any pre-existing socket is not already present.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.

One point to be mentioned here is if channel is not bound and any I/O operation is tried to be attempted then NotYetBoundException is thrown by this channel.So one must be ensure that channel is bounded before performing any IO operation.

Incoming connections for the server socket channel are listen by calling the ServerSocketChannel.accept() method. When the accept() method returns, it returns a SocketChannel with an incoming connection. Thus, the accept() method blocks until an incoming connection arrives.If the channel is in non-blocking mode then accept method will immediately return null if there are no pending connections. Otherwise it will block indefinitely until a new connection is available or an I/O error occurs.

The new channel's socket is initially unbound; it must be bound to a specific address via one of its socket's bind methods before connections can be accepted.Also the new channel is created by invoking the openServerSocketChannel method of the system-wide default SelectorProvider object.

Like socket channel server socket channel could read data using **read()** method.Firstly the buffer is allocated. The data read from a ServerSocketChannel is stored into the buffer.Secondly we call the ServerSocketChannel.read() method and it reads the data from a ServerSocketChannel into a buffer. The integer value of the read() method returns how many bytes were written into the buffer

Similarly data could be written to server socket channel using **write()** method using buffer as a parameter.Commonly uses write method in a while loop as need to repeat the write() method until the Buffer has no further bytes available to write.

## Important methods of Socket channel

* **bind(SocketAddress local)** − This method is used to bind the socket channel to the local address which is provided as the parameter to this method.
* **accept()** − This method is used to accepts a connection made to this channel's socket.
* **connect(SocketAddress remote)** − This method is used to connect the socket to the remote address.
* **finishConnect()** − This method is used to finishes the process of connecting a socket channel.
* **getRemoteAddress()** − This method return the address of remote location to which the channel's socket is connected.
* **isConnected()** − As already mentioned this method returns the status of connection of socket channel i.e whether it is connected or not.
* **open()** − Open method is used open a socket channel for no specified address.This convenience method works as if by invoking the open() method, invoking the connect method upon the resulting server socket channel, passing it remote, and then returning that channel.
* **read(ByteBuffer dst)** − This method is used to read data from the given buffer through socket channel.
* **setOption(SocketOption<T> name, T value)** − This method sets the value of a socket option.
* **socket()** − This method retrieves a server socket associated with this channel.
* **validOps()** − This method returns an operation set identifying this channel's supported operations.Server-socket channels only support the accepting of new connections, so this method returns SelectionKey.OP\_ACCEPT.

## Example

The following example shows the how to send data from Java NIO ServerSocketChannel.

### C:/Test/temp.txt

Hello World!

### Client: SocketChannelClient.java

import java.io.IOException;

import java.net.InetSocketAddress;

import java.nio.ByteBuffer;

import java.nio.channels.FileChannel;

import java.nio.channels.ServerSocketChannel;

import java.nio.channels.SocketChannel;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.nio.file.StandardOpenOption;

import java.util.EnumSet;

public class SocketChannelClient {

public static void main(String[] args) throws IOException {

ServerSocketChannel serverSocket = null;

SocketChannel client = null;

serverSocket = ServerSocketChannel.open();

serverSocket.socket().bind(new InetSocketAddress(9000));

client = serverSocket.accept();

System.out.println("Connection Set: " + client.getRemoteAddress());

Path path = Paths.get("C:/Test/temp1.txt");

FileChannel fileChannel = FileChannel.open(path,

EnumSet.of(StandardOpenOption.CREATE,

StandardOpenOption.TRUNCATE\_EXISTING,

StandardOpenOption.WRITE)

);

ByteBuffer buffer = ByteBuffer.allocate(1024);

while(client.read(buffer) > 0) {

buffer.flip();

fileChannel.write(buffer);

buffer.clear();

}

fileChannel.close();

System.out.println("File Received");

client.close();

}

}

### Output

Running the client will not print anything until server starts.

### Server: SocketChannelServer.java

import java.io.IOException;

import java.net.InetSocketAddress;

import java.net.SocketAddress;

import java.nio.ByteBuffer;

import java.nio.channels.FileChannel;

import java.nio.channels.SocketChannel;

import java.nio.file.Path;

import java.nio.file.Paths;

public class SocketChannelServer {

public static void main(String[] args) throws IOException {

SocketChannel server = SocketChannel.open();

SocketAddress socketAddr = new InetSocketAddress("localhost", 9000);

server.connect(socketAddr);

Path path = Paths.get("C:/Test/temp.txt");

FileChannel fileChannel = FileChannel.open(path);

ByteBuffer buffer = ByteBuffer.allocate(1024);

while(fileChannel.read(buffer) > 0) {

buffer.flip();

server.write(buffer);

buffer.clear();

}

fileChannel.close();

System.out.println("File Sent");

server.close();

}

}

### Output

Running the server will print the following.

Connection Set: /127.0.0.1:49558

File Received