“Non-blocking”, i.e., do not block, or stop, waiting for some event. The ‘event’ could be a timeout, the arrival of data from the network, freeing up space in an outgoing network buffer, the completion of a disk request—anything that might occur in the future.

The key idea, for implementing a ‘Non-blocking’ program is to use 1 thread, and use `select()` which blocks until one of many things happen, such as

- time elapses
- a socket has data to read
- a socket can be written without waiting
- etc.

Use this basic technique: Only block in `select`; create an object called a channel that corresponds to each communications channel. Then the select returns only when one of the channels can be used immediately.

For example, suppose we’re writing a TCP server that needs to process a large amount of arriving data, and send many big files that take a long time to transmit.

Make channels for the incoming server socket, and for each socket connected to a client. Then select on the channels for the events

- data ready for reading on a socket
- a socket is ready for writing
- a timeout fires because a connected socket has been inactive for too long

and do whatever can be done right away.
import java.io.*;
import java.net.*;
import java.nio.*;
import java.nio.channels.*;
import java.util.*;

class Nonblocking_IO_example {

    // set input parameters
    String host = "localhost"; // "slinky.cs.nyu.edu";
    int port = 10000;
    int datagramSize = 512;
    int datagramInterval = 3000;
    int numMsgs = 24;

    // array of bytes for a datagram to send
    byte[] sendData = new byte[ datagramSize ];
    ByteBuffer theSendDataByteBuffer = ByteBuffer.wrap( sendData );

    // array of bytes for receiving datagrams
    byte[] receiveData = new byte[ datagramSize ];
    ByteBuffer theReceiveDataByteBuffer = ByteBuffer.wrap( receiveData );

    Nonblocking_IO_example ()
    {
        try
        {
            Random theRandom = new Random();
            InetSocketAddress theInetSocketAddress = new InetSocketAddress( host, port);

            // make a DatagramChannel
            DatagramChannel theDatagramChannel = DatagramChannel.open();

            // A channel must first be placed in nonblocking mode
            // before it can be registered with a selector
            theDatagramChannel.configureBlocking( false );

            // instantiate a selector
            Selector theSelector = Selector.open();

            // register the selector on the channel to monitor reading
            // datagrams on the DatagramChannel
            theDatagramChannel.register( theSelector, SelectionKey.OP_READ );

            long millisecsUntilSendNextDatagram = 0;
            int i = 1;  int j = 1;

            // send and read concurrently, but do not block on read:
            while (true)
            {
                long start = System.currentTimeMillis();

                // which comes first, next send or a read?
                // in case millisecsUntilSendNextDatagram <= 0 go right to send
                if ( millisecsUntilSendNextDatagram <= 0 ||
                    theSelector.select( millisecsUntilSendNextDatagram ) == 0 )
                {
                    // just for fun, send between 0 and 4 datagrams
                    for( int k = 0; k < theRandom.nextInt( 5 ); k++ )
                    {
                        theDatagramChannel.send( theSendDataByteBuffer, theInetSocketAddress );
                        System.out.println("sent Datagram " + j++);
                    }
                    millisecsUntilSendNextDatagram = datagramInterval;
                } else
                {
                    // which comes first, next send or a read?
                    // in case millisecsUntilSendNextDatagram <= 0 go right to send
                    // otherwise read
                    theDatagramChannel.send( theSendDataByteBuffer, theInetSocketAddress );
                    System.out.println("sent Datagram " + j++);
                    theSelector.select( millisecsUntilSendNextDatagram ) == 0
                }
            }
        }
    }
}
else
{
  // read the datagram from the socket,
  theDatagramChannel.receive( theReceiveDataByteBuffer );
  System.out.println("theDatagramChannel.receive " + i++);
  // datagram would be processed here

  // Get an iterator over the set of selected keys
  Iterator it = theSelector.selectedKeys().iterator();
  // will be exactly one key in the set, but iterator is
  // only way to get at it
  while( it.hasNext() ){
    it.next();
    // Remove key from selected set; it's been handled
    it.remove();
  }
  // how much time used up
  millisecsUntilSendNextDatagram = System.currentTimeMillis() - start;
  if( j > numMsgs ) break;
}
theSelector.close();
}
catch (Exception e)
{
  e.printStackTrace();
  System.out.println("Exception " + e);
  return;
}
}
public static void main( String args[] )
{
  new Nonblocking_IO_example();
}