Practical Exercises in Computer Networks

Ethernet MAC-related programming in Java (WIP) (Part 1, 2014)

© 2013 and 2014, José María Foces Morán

This LAB assumes that the students already have taken up Ethernet like in the initial paragraphs of textbook section 2.6 (Ethernet and Multiple Access Networks), particularly the different types of MAC addressing.

Introduction

As we have reviewed in textbook section 2.6, MAC addresses are 48-bit long, not hierarchical and are of three types: Unicast, Broadcast and Multicast. Unicast addresses represent the MAC address of a single NIC (Network Interface Adaptor), the Broadcast address somehow represents all of the computers connected to a single LAN segment and we use this address when we want to explicitly send some information to all of them with an indication that it is relevant to all of them. Now, it is necessary to set this definition in context because, in previous lab sessions we mentioned that some ethernet media were of the *broadcast type*, specifically bus ethernets and star ethernets based on hubs. In these technologies, every ethernet frame, whichever computer it is addressed to, is delivered to each and every computer belonging to the considered LAN, the difference is that only the computer to which it is addressed to will convey the frame to the layer-3 protocol. The last MAC address type mentioned, the *multicast* type is used for communicating with a subset of all the nodes belonging to a LAN, *i.e.,* unicast represents one node's NIC, broadcast represents all of them and multicast represents any arbitrary subset thereof.

As an illustration of ethernet addressing, we are going to write a Java program that sends a broadcast frame. The purpose of this example frame consists of waking up a computer that is in sleep mode, the frame is known as *The Magic Packet,* we will learn how to build it and send it in Java.

The Magic Packet

This packet, when received by a computer's NIC causes it to send the power-up command to the computer system's power supply in which it is installed and properly configured (Boot and bios configuration). The standard Magic Packet frame must contain the following fields, always in *network byte order* (Recall there are two byte-orderings used in computers: Little-endian and big-endian, network-byte order is big-endian). In order for us to better grasp our task here, let's represent the byte sequence that is to be received by the target computer's NIC with an array of bytes, which is exactly the computational representation called for by the data encapsulation in Java DatagramPacket's objects:

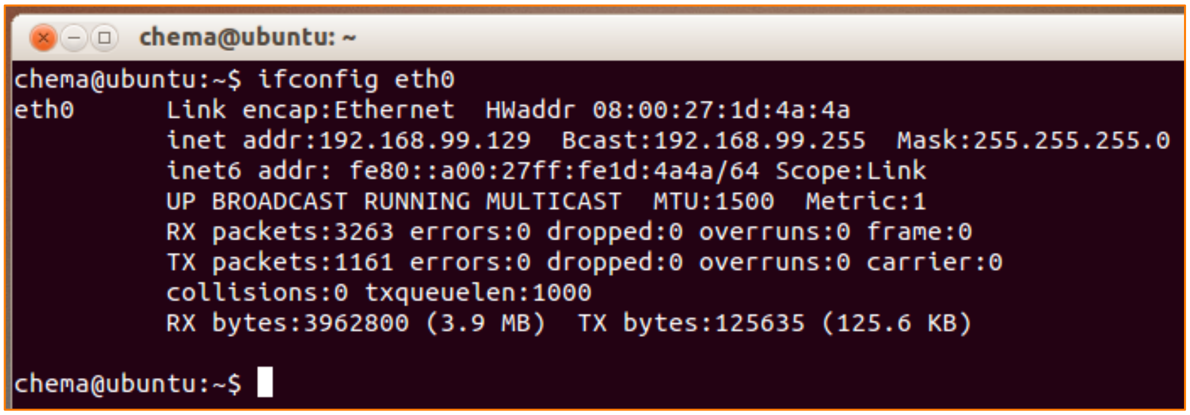
- 6 all-ones bytes, *i.e.*, bytes [0] through [5] filled up with 1's (The hex constant 0xff contains all ones)
- 16 times the MAC address of the target NIC in network-byte order (A total of 16 x 6 bytes)

1. **Exercise.** Appendix 1 of this lab exercise contains the source code of a Java program that sends the Magic Packet (WakeOnLan.java), download this program from http://paloalto.unileon.es/cn/WakeOnLan.java. The program builds the Magic Packet for a concrete,

example NIC instance whose MAC address is (In hex): **00:1D:09:07:8c:e9**, therefore you will have to **edit** the source code so that it fills the array up with your target MAC address. Let's review the most relevant code sections hereon:

```
//Fill the first 6 bytes with all ones
final int SIZE = 6 + 6 * 16;
byte data[] = new byte[SIZE];

        for (int i = 0; i < 6; i++) {
            data[i] = (byte) 0xff;
        }
```

Obviously, in order for you to successfully test the program, you will have to learn your LAN's broadcast IP address, this is determined by issuing a simple ifconfig command as in the following example, you will have to copy/paste the ifconfig's field identified by the string *BCast*, in this case 192.168.99.255. Technically, this IP broadcast address is known as *The Broadcast-directed address* which we will take up in chapter 3 section devoted to IP addressing. Fort the time being, we will want to stress the idea that there exist layer-3 addresses or IP addresses and layer-2 addresses which in the case of Ethernet are known as MAC addresses and that a full fledged protocol is devoted to mapping layer-3 addresses to layer-2 addresses and conversely. The IP protocol implementation of any mainstream operating system contains three interdependent software modules: The IP forwarding module, the ARP module and the ICMP module. We will delve into these topics in chapter 3.



```
chema@ubuntu:~$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 08:00:27:1d:4a:4a
          inet addr:192.168.99.129  Bcast:192.168.99.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe1d:4a4a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3263 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1161 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3962800 (3.9 MB)  TX bytes:125635 (125.6 KB)

chema@ubuntu:~$ 
```

Fig. 1. Ifconfig provides us the *broadcast-oriented* IP address of our LAN (Field Bcast)

Now, you will have to collaborate with one of your lab mates, you will need your mate's NIC's MAC address which can be found by ussuing the same ifconfig command mentioned before, however, in this case, you will have to copy/paste the HWaddr field and substitute the example MAC used in the program for the real one. Now, we can proceed to explain the rest of the program.

The following for loop will iterate a total of 16 times, each time it will fill a new block of subsequent 6 array positions with a new copy of the target MAC:

```
for (int i = 1; i <= 16; i++) {

/*
 * Example MAC address:
 *    00:1D:09:07:8c:e9
 *
 * Make sure you honor the byte order as you transcribe
 * the byte constants into the successive data array slots,
 * please, contrast the following assignments with the
 * preceding MAC address:
 */

            data[6 * i + 0] = (byte) 0x00;
            data[6 * i + 1] = (byte) 0x1d;
            data[6 * i + 2] = (byte) 0x09;
            data[6 * i + 3] = (byte) 0x07;
            data[6 * i + 4] = (byte) 0x8c;
            data[6 * i + 5] = (byte) 0xe9;

}
```

Once the array containing the Magic Packet's payload has been built, the program proceeds to encapsulate it into a Java DatagramPacket object which is subsequently sent to the ethernet LAN's broadcast address by using a DatagramSocket object.

You will have to make sure that the computer system hat is to receive the Magic Packet is properly configured so that it conveys the Magic Packet to the chipset block that issues the power-up command, to that end, you will have to reboot the system and pay attention to the boot-up message that informs about how to boot the BIOS configuration program (Press a softkey or a control combination)and browse its different menus until you find the relevant Magic Packet or Wake-On-Lan configuration entries.

2. **Exercise**. Make the program accept the target NIC's MAC address in hex base from the keyboard, so that there will be no need to continually recompile the program with each new target's MAC we want to test with. Test your program with at least one other target computer NIC.

Study the Magic Packet by using WireShark

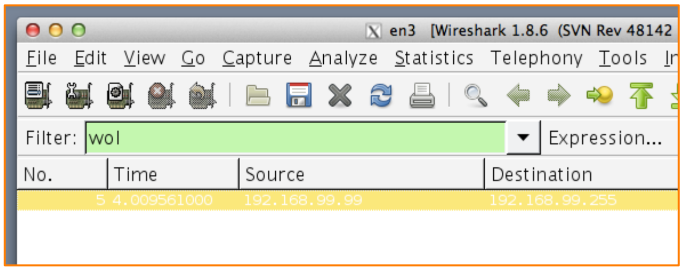
We can use a protocol analyzer connected to the same LAN where the Magic Packet (MP) sender and receiver reside –note that in principle, we can only send the Magic Packet if we generate it within the LAN where the target NIC is physically connected, if we take ad-hoc technical measures we will be able to generate the MP in a network other tan the target's, but we are not interested in that level of detail so far. We shall, therefore, continue with that simple context of a single LAN that contains the MP sender and the MP target. Now we want to watch the MP on the wire by using a protocol analyzer like Wireshark. Please, follow the next step list to be able to watch the MP son Wireshark packet display, pay special attention to the accompanying explanations and screen dumps of a Wireshark running on Mac and solving this exercise:

a) Make sure you have a version of the WakeOnLan program, properly compiled and ready to run

b) Start Wireshark

c) Enable the eth? (Linux) or the en? (Mac) network interface so that Wireshark can capture frames from it

d) In the Wireshark capture filter textbox (Filter: ____), enter "Wol" as the protocol whose frames you want to capture


en3 [Wireshark 1.8.6 (SVN Rev 48142
File Edit View Go Capture Analyze Statistics Telephony Tools
Expression...

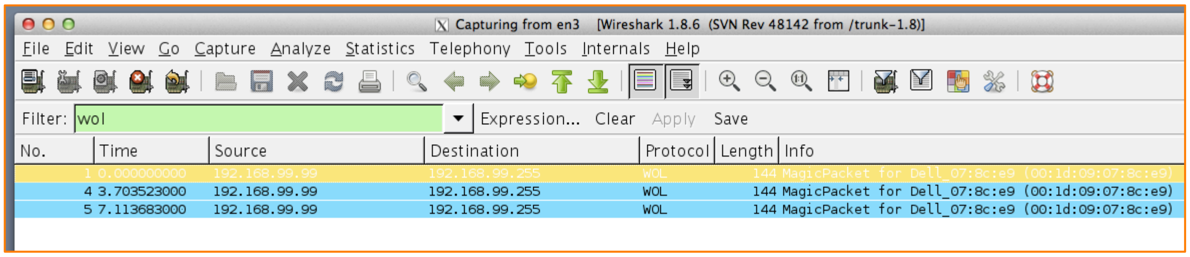
Filter: wol

```
No.     Time           Source           Destination
      5 4.009561000    192.168.99.99    192.168.99.255
```

Fig.2. A capture filter "wol" is specified before starting a new trace

e) Start a new capture in Wireshark

f) Send the MP by running the WakeOnLan program


Capturing from en3 [Wireshark 1.8.6 (SVN Rev 48142 from /trunk-1.8)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save

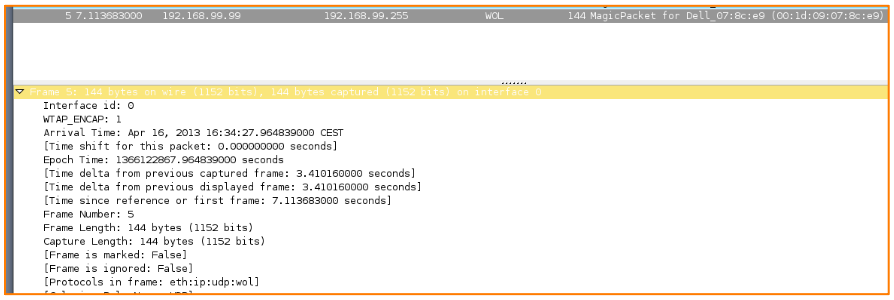
Filter: wol

```
No.     Time           Source          Destination       Protocol Length Info
      1 0.000000000    192.168.99.99   192.168.99.255    WOL         144 MagicPacket for Dell_07:8c:e9 (00:1d:09:07:8c:e9)
      4 3.703523000    192.168.99.99   192.168.99.255    WOL         144 MagicPacket for Dell_07:8c:e9 (00:1d:09:07:8c:e9)
      5 7.113683000    192.168.99.99   192.168.99.255    WOL         144 MagicPacket for Dell_07:8c:e9 (00:1d:09:07:8c:e9)
```

Fig. 3. The MP was sent three times and the three resulting frames were recorded by Wireshark

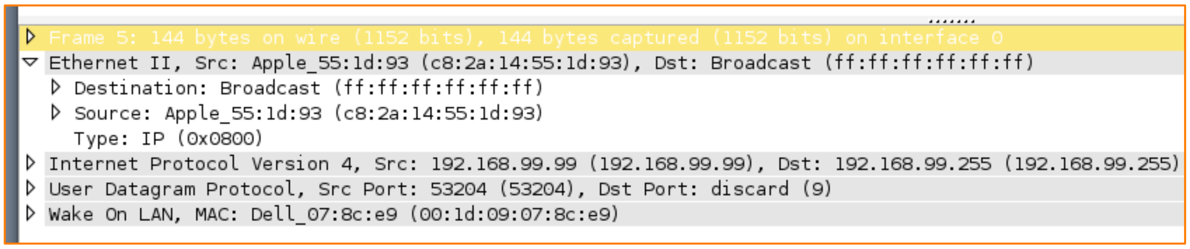
g) You will see one more line on the Wireshark packet display pane whose protocol is WOL (See the figure above), one more line for each time you execute WakeOnLan

h) Now, study the MP at levels 1, 2, 3 and 4 (Consult the included screen dumps)



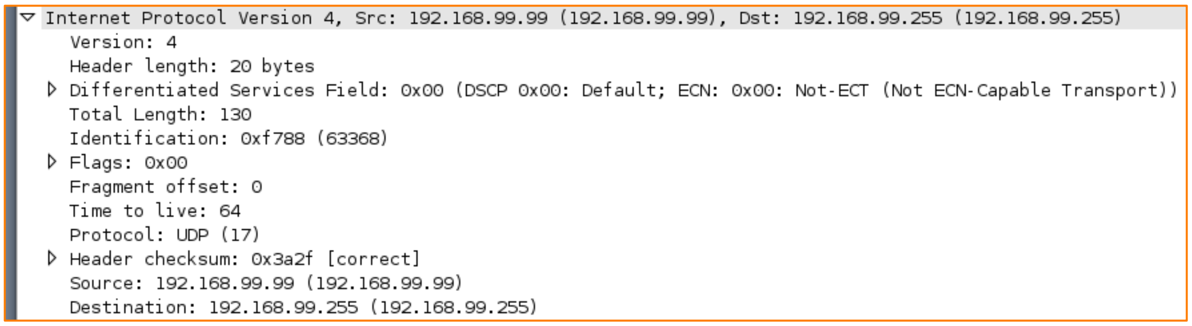
```
5 7.113683000    192.168.99.99          192.168.99.255          WOL        144 MagicPacket for Dell_07:8c:e9 (00:1d:09:07:8c:e9)

▽ Frame 5: 144 bytes on wire (1152 bits), 144 bytes captured (1152 bits) on interface 0
    Interface id: 0
    WTAP_ENCAP: 1
    Arrival Time: Apr 16, 2013 16:34:27.964839000 CEST
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1366122867.964839000 seconds
    [Time delta from previous captured frame: 3.410160000 seconds]
    [Time delta from previous displayed frame: 3.410160000 seconds]
    [Time since reference or first frame: 7.113683000 seconds]
    Frame Number: 5
    Frame Length: 144 bytes (1152 bits)
    Capture Length: 144 bytes (1152 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ip:udp:wol]
```

Fig. 4. Display of the MP at level 1 (Physical). Note that you can see the whole encapsulation/demultiplexing of protocol data units (PDU) starting with the Ethernet frame (See the line titled ‘Protocols in frame: eth:ip:udp:wol’)



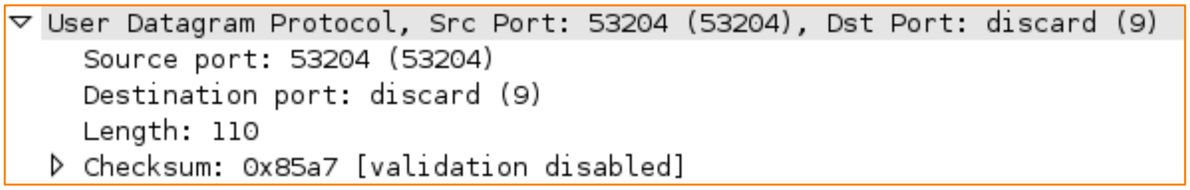
```
▷ Frame 5: 144 bytes on wire (1152 bits), 144 bytes captured (1152 bits) on interface 0
▽ Ethernet II, Src: Apple_55:1d:93 (c8:2a:14:55:1d:93), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ▷ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  ▷ Source: Apple_55:1d:93 (c8:2a:14:55:1d:93)
    Type: IP (0x0800)
▷ Internet Protocol Version 4, Src: 192.168.99.99 (192.168.99.99), Dst: 192.168.99.255 (192.168.99.255)
▷ User Datagram Protocol, Src Port: 53204 (53204), Dst Port: discard (9)
▷ Wake On LAN, MAC: Dell_07:8c:e9 (00:1d:09:07:8c:e9)
```

Fig. 5. Display of the MP at level 2 (Datalink). Note the demultiplexing key titled ‘type’, whose value is 0x0800, meaning that this frame is carrying an IP packet payload)



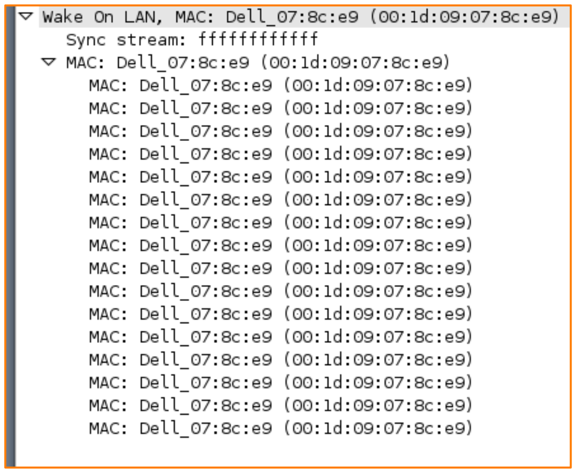
```
▽ Internet Protocol Version 4, Src: 192.168.99.99 (192.168.99.99), Dst: 192.168.99.255 (192.168.99.255)
    Version: 4
    Header length: 20 bytes
  ▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 130
    Identification: 0xf788 (63368)
  ▷ Flags: 0x00
    Fragment offset: 0
    Time to live: 64
    Protocol: UDP (17)
  ▷ Header checksum: 0x3a2f [correct]
    Source: 192.168.99.99 (192.168.99.99)
    Destination: 192.168.99.255 (192.168.99.255)
```

Fig. 6. Display of the MP at level 3 (Network). Note the demultiplexing key in this case is 9 meaning that the payload is to be delivered to layer-4 protocol number 17 (User Datagram Protocol) which is consistent with the Java DatagramSocket used.



```
▽ User Datagram Protocol, Src Port: 53204 (53204), Dst Port: discard (9)
    Source port: 53204 (53204)
    Destination port: discard (9)
    Length: 110
  ▷ Checksum: 0x85a7 [validation disabled]
```

Fig. 7. Display of the MP at level 4 (Transport, UDP).



```
▽ Wake On LAN, MAC: Dell_07:8c:e9 (00:1d:09:07:8c:e9)
    Sync stream: ffffffffffff
  ▽ MAC: Dell_07:8c:e9 (00:1d:09:07:8c:e9)
      MAC: Dell_07:8c:e9 (00:1d:09:07:8c:e9)
      MAC: Dell_07:8c:e9 (00:1d:09:07:8c:e9)
      MAC: Dell_07:8c:e9 (00:1d:09:07:8c:e9)
      MAC: Dell_07:8c:e9 (00:1d:09:07:8c:e9)
      MAC: Dell_07:8c:e9 (00:1d:09:07:8c:e9)
      MAC: Dell_07:8c:e9 (00:1d:09:07:8c:e9)
      MAC: Dell_07:8c:e9 (00:1d:09:07:8c:e9)
      MAC: Dell_07:8c:e9 (00:1d:09:07:8c:e9)
      MAC: Dell_07:8c:e9 (00:1d:09:07:8c:e9)
      MAC: Dell_07:8c:e9 (00:1d:09:07:8c:e9)
      MAC: Dell_07:8c:e9 (00:1d:09:07:8c:e9)
      MAC: Dell_07:8c:e9 (00:1d:09:07:8c:e9)
      MAC: Dell_07:8c:e9 (00:1d:09:07:8c:e9)
      MAC: Dell_07:8c:e9 (00:1d:09:07:8c:e9)
      MAC: Dell_07:8c:e9 (00:1d:09:07:8c:e9)
      MAC: Dell_07:8c:e9 (00:1d:09:07:8c:e9)
```

Fig 8. Display of the MP at application level

Additional Exercises (Part 1)

3. The Magic Packet is encapsulated into a Java DatagramPacket object that is built by invoking the constructor DatagramPacket(data, SIZE, address, 9), can you explain the significance of the constant 9 as fourth argument? If we change it for any other integer value of your choosing, would the Wol program continue functioning ok? Check this.

4. Investigate the ifconfig error-reporting capabilities, *i.e.,* the Ethernet errors that can be reported by the command.

Appendix 1: Source Code

```
/**
 * ***************************************************************************
 * Universidad de León, EIII Grado en Ingeniería Informática
 * Course on Computer Networks (C) 2013 José María Foces Morán, lecturer
 *
 * CN LAB on Ethernet addressing (Wake-On-Lan)
 * WakeOnLan.java
 *
 *  1. Java UDP sockets
 *  2. Broadcast-directed net address use
 *  3. Wake-On-Lan Magic Packet exercise

 * Builds a Magic Packet capable of waking up a node which MAC address is known
 * Magic Packet's structure is as follows:
 * 1. 6 all-ones bytes
 * 2. 16 copies of the node's MAC address in Network Byte Order
 * Each MAC address instance needs 6 bytes of storage for a total of:
 * 6 (all-ones) bytes + 16 times the node's MAC address, 6 bytes each =
 * (6 + 16 * 6) bytes
 *
 * ***************************************************************************
 */

package wol;

import java.net.*;
import java.util.logging.Level;
import java.util.logging.Logger;

public class WakeOnLan {

    private DatagramPacket buildWOLPacket(InetAddress address) {

        final int SIZE = 6 + 6 * 16; // See above for this magic number

        byte data[] = new byte[SIZE];

        for (int i = 0; i < 6; i++) {
            data[i] = (byte) 0xff;
        }

        for (int i = 1; i <= 16; i++) {

            /*
             * Example MAC address:
             *    00:1D:09:07:8c:e9
             *
             * Make sure you honor the byte order as you transcribe
             * the byte constants into the successive data array slots,
             * please, contrast the following assignments with the preceding
             * MAC address:
             */

            data[6 * i + 0] = (byte) 0x00;
            data[6 * i + 1] = (byte) 0x1d;
            data[6 * i + 2] = (byte) 0x09;
            data[6 * i + 3] = (byte) 0x07;
            data[6 * i + 4] = (byte) 0x8c;
            data[6 * i + 5] = (byte) 0xe9;

        }

        //A simplistic printout check of the WOL wolPacket:
        for (int i = 0; i < SIZE; i++) {
            //System.out.println(Integer.toHexString(data[i]));
            //System.out.println(data[i]);
            System.out.printf("%x\n", data[i]);
        }


        /*
         * Effectively return the instance of DatagramPacket containing
         * the aforementioned 17 6-byte WOL wolPacket segments:
         *
         */
        return new DatagramPacket(data, SIZE, address, 9);

    }

    public void start(InetAddress address) {

        /*
         * We start by building a WOL packet, this we delegate to method
         * buildWOLPacket() to which we pass an instnace of our inet address
         *
         */
        DatagramPacket wolPacket = buildWOLPacket(address);

        /*
         * Now we are ready for the transmission of the wol packet, we create
         * a DatagramSocket and subsequently invoke its send method with the
         * wol packet as the only argument, then, if no exception occurs, we
         * are done.
         */
        try {
            DatagramSocket socket = new DatagramSocket();
            socket.send(wolPacket);
        } catch (SocketException ex) {
            Logger.getLogger(WakeOnLan.class.getName()).log(Level.SEVERE, null, ex);
        } catch (java.io.IOException ex) {
            Logger.getLogger(WakeOnLan.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    /*
     * The following main method constitutes the entry point to the program,
     * upon entry, the virtual machine executes in a non-object-oriented mode
     * which is known as "static" (non-object, which by contrast is considered
     * "dynamic")
     *
     */
    public static void main(String[] args) {

        /* Capture the broadcast-directed IP address of the LAN
         * to which this computer belongs from the list of arguments
         * received by this process
         */

        if (args.length != 1) {
            System.err.println("Syntax: java wol.WakeOnLan <Broadcast-directed address
(DDN)>");
            System.exit(-1);
        }


        /* Now we try to convert the address argument into an InetAddress which
         * we'll pass to the start() method afterwards
         */
        InetAddress address = null;
        try {
            address = InetAddress.getByName(args[0]);
        } catch (UnknownHostException ex) {
            Logger.getLogger(WakeOnLan.class.getName()).log(Level.SEVERE, null, ex);
            System.exit(-2);
        }


        /* In this simple, introductory program we simply instantiate a
         * WakeOnLan class and assign the resulting object's reference to
         * a variable named w
         */
        WakeOnLan w = new WakeOnLan();

        /* By this time we can send messages to the object w, that is, invoke
         * its public methods thereby, effectively setting our program "into
         * motion".
         */
        w.start(address);

        System.out.println("WakeOnLan has finished.");

        /* Silently finish the program by implicitly calling " System.exit(0); "
         * which amounts to signalling the virtual machine that it should
         * finish. The System class is imported by default by all Java programs.
         */
    }
}
```