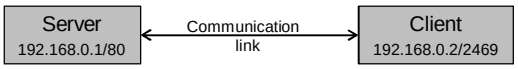
Introduction to Java Network Programming

Getting Started

- Java Development Platform
  - Java Virtual Machine (JVM) is available on many different operating systems such as Microsoft Windows, Solaris OS, Linux, or Mac OS.
  - Sun JDK 5 or 6 which includes Java Runtime Environment (JRE) and command-line development tools.
- Development Tools
  - Main tools (`javac` compiler and `java` launcher)
  - Using IDE (Eclipse, Netbeans, JCreator ….)
  - Automate compiling (Apache Ant) and testing (JUnit)

Socket

- What is Socket?
  - Internet socket or Network socket is one endpoint of two-way communication link between two network programs.



- Types of Socket
  - Stream sockets – connection-oriented sockets which use TCP
  - Datagram sockets – connection-less sockets which use UDP
  - Raw sockets – allow direct transmission of network packets from applications (bypassing transport layer)
- Modes of Operation
  - Blocking or Synchronous – application will block until the network operation completes
  - Nob-blocking or Asynchronous – event-driven technique to handle network operations without blocking.

Resolving Hostname

- java.net.InetAddress
  - The *InetAddress* class provides methods to resolve host names to their IP addresses and vise versa.

```
try {
   // create an InetAddress object
   InetAddress ia = InetAddress.getByName("www.google.com");
   System.out.println(ia.getHostAddress()); // prints IP address in textual form
   } catch (UnknownHostException uhe) {
     System.out.println("Could not find host");
     uhe.printStackTrace();
   }
```

  - For other APIs, consult http://java.sun.com/j2se/1.5.0/docs/api/java/net/InetAddress.html
  - *Inet4Address* and *Inet6Address* are the subclasses of InetAddress to represent 32-bit IPv4 address and 128-bit IPv6 address respectively.
- Java.net.InetSocketAddress
  - *InetSocketAddress* class represents an IP socket address. It can include an IP address and port, or a hostname and port. In the later case, an attempt will be made to resolve the hostname.

Java Sockets Classes

- Blocking sockets
  - java.net.ServerSocket
  - java.net.Socket
  - } TCP
  - java.net.DatagramSocket
  - java.net.MulticastSocket
  - } UDP
- Non-blocking sockets
  - java.nio.channels.ServerSocketChannel
  - java.nio.channels.SocketChannel
  - } TCP
  - java.nio.channels.DatagramChannel
  - } UDP
- Other important classes used in socket programming
  - Java.net.DatagramPacket
  - Java.nio.ByteBuffer

Stream Sockets (blocking)

- The *Socket* class represents a client socket. Each Socket object is associated with exactly one remote host. To connect to a different host, you must create a new Socket object.
  - Connection can be established in different ways

```
// This constructor will block until the connection succeeds
Socket socket = new Socket("java.sun.com", 80); // throws UnknownHostException
and IOException
Or
// Create an unbound socket
Socket socket = new Socket();
socket.connect(new InetSocketAddress("java.sun.com", 80)); // throws IOException
```

  - It is very good practice to close sockets before quitting the program

```
socket.close() // throws IOException
```

- The *ServerSocket* class represents a server socket. It is constructed on a particular port. Then it calls *accept()* to listen for incoming connections. *accept()* call blocks until a connection request is detected on the specified port.

```
ServerSocket serverSocket = new ServerSocket(80); // throws IOException
// returns Socket object to perform actual communication with the client
Socket socket = serverSocket.accept();
```

- *setSoTimeout(int timeout)* method of both *Socket* and *ServerSocket* class can be enabled to make the blocking APIs (e.g., *accept()*) waituntil specified amount of timeout (>0) in milliseconds. If the timeout expires, a *java.net.SocketTimeoutException* is raised.

Set up input and output streams

- Once a socket has connected you send data to the server via an output stream. You receive data from the server via an input stream.
- Methods *getInputStream* and *getOutputStream* of class *Socket.*
- Reading text from a socket

```
BufferedReader br = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
   String str;
   while ((str = br.readLine()) != null) {
      process(str);
   }
rd.close();
```

- Writing text to a socket

```
BufferedWriter bw = new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream()));
bw.write("aString");
bw.flush();
```

- To process binary data such as ints or doubles

```
DataInputStream dis = new DataInputStream(socket.getInputStream());
DataOutputStream dos = new DataOutputStream(socket.getOutputStream());
double num = dis.readDouble();
dos.writeDouble(num);
dos.flush();
```


Datagram Sockets

- The *DatagramSocket* class represents a socket for sending and receiving datagram packets. Unlike TCP sockets, there is no distinction between a UDP socket and a UDP server socket, and a *DatagramSocket* can send to multiple different addresses. The address to which data goes is stored in the packet, not in the socket.
  - Creating an UDP socket and binding to a port

```
DatagramSocket dgramSocket = new DatagramSocket(null);
dgramSocket.bind(new InetSocketAddress(8888));
or
DatagramSocket dgramSocket = new DatagramSocket(8888);
```

  - *connect(address, port)* can be used to send to and receive from a particular address.
- The *DatagramPacket* class is used to implement a connectionless packet delivery service. Multiple packets sent from one machine to another might be routed differently, and might arrive in any order.
  - Sending a datagram

```
DatagramPacket outDgramPkt = new DatagramPacket(outbuf, outbuf.length,
        InetAddress.getByName("130.233.x.y"), 3048);
dgramSocket.send(outDgramPkt);
```

  - Receiving a datagram

```
DatagramPacket inDgramPkt = new DatagramPacket(inbuf, inbuf.length);
dgramSocket.receive(inDgramPkt);//This method blocks until a datagram is received
```

Multicast Datagram Socket

- The *MuticastSocket* is a (UDP) *DatagramSocket* with additional capabilities for joining "groups" of other multicast hosts on the internet. A multicast group is specified by a class D IP address and by a standard UDP port number.
  - Joining a multicast group

```
InetAddress group = InetAddress.getByName("228.5.6.7");
MulticastSocket mcastSocket = new MulticastSocket(6789);
mcastSocket.joinGroup(group);
```

  - Sending to a multicast group

```
DatagramPacket packet = new DatagramPacket(outbuf, outbuf.length, group, 1234);
mcastSocket.send(packet);
```

  - Receiving from a multicast group

```
DatagramPacket packet = new DatagramPacket(inbuf, inbuf.length);
// Wait for packet
mcastSocket.receive(packet);
```

  - Leaving a multicast group

```
mcastSocket.leaveGroup(group);
```


Non-blocking socket creation

- The *SocketChannel* class allows to create stream-oriented non-blocking client socket.
  - Creating a non-blocking socket channel and connecting to a remote host

```
SocketChannel sChannel = SocketChannel.open();
// set SocketChannel to non-blocking (default blocking)
sChannel.configureBlocking(false);
sChannel.connect(new InetSocketAddress(”www.google.com”, 80));
while (!sChannel.finishConnect()) {
     System.out.println(“connection failure”);
}
```

- The *ServerSocketChannel* class represents stream-oriented server socket to listen for incoming connections without blocking.
  - Creating a non-blocking server socket channel on port 80

```
ServerSocketChannel ssChannel = ServerSocketChannel.open();
ssChannel.configureBlocking(false);
ssChannel.socket().bind(new InetSocketAddress(80));
```

  - Then accept the connection request

```
SocketChannel sChannel = ssChannel.accept();//retruns null if no pending requests
```


Socket Channel I/O Operations

- Writing to a socket channel

```
String message = "aMessage";
ByteBuffer buf = ByteBuffer.wrap(message.getBytes());
int numBytesWritten = sChannel.write(buf);
```

- Reading from a socket channel

```
try {
  ByteBuffer buf = ByteBuffer.allocateDirect(1024);
  buf.clear();
  int numBytesRead = sChannel.read(buf);
  if (numBytesRead == -1) { // No more bytes can be read from the channel
     sChannel.close();
   } else {
       buf.flip(); // To read the bytes, flip the buffer
       byte[] bytesRead = new byte[buf.limit()];
       buf.get(bytesRead, 0, bytesRead.length);
   }
} catch (IOException e) { }
```


Datagram Socket Channel (Non-blocking)

- The *DatagramChannel* class represents a non-blocking datagram-oriented socket channel.
  - Creating datagram channel

```
DatagramChannel dChannel = DatagramChannel.open();
dChannel.configureBlocking(false);
dChannel.socket.bind(new InetSocketAddress(2345));
```

  - Datagram I/O can be performed in two ways

```
dChannel.connect(new InetSocketAddress("time-a.nist.gov", 37));
// A datagram channel must be connected in order to use the read and write
methods
int numBytesWritten = dChannel.write(buf);
or
// A datagram channel need not be connected in order for the send and receive
methods
SocketAddress sa = dChannel.receive(buf);
```

Handling Multiple Connections (Blocking case)

- Multithreading approach
- The server spawns a new thread to manipulate a new connection
- Needs to handle concurrency issues (http://java.sun.com/docs/books/tutorial/essential/concurrency)

```
public void listenSocket() {
    try{
        server = new ServerSocket(4444);
        while(true){
            // ClientWorker class must implement runnable interface
            ClientWorker cw = new ClientWorker(server.accept());
            Thread t = new Thread(cw);
            t.start();
        }
    } catch (IOException e) { }
}
class ClientWorker implements Runnable {
    private Socket client;
    ClientWorker(Socket client) {
        this.client = client;
    }
    public void run(){
        // handle the connection (reading from or writing to a socket)
    }
}
```


Handling Multiple Connections (Non-blocking case)

- Event-based mechanism
- The server can handle multiple simultaneous connections from a single thread with the aid of the *Selector* class

```
try {
    Selector selector = Selector.open(); // Create the selector
    ServerSocketChannel ssChannel = ServerSocketChannel.open();
    ssChannel.configureBlocking(false);
    ssChannel.socket().bind(new InetSocketAddress(80));
    ssChannel.register(selector, SelectionKey.OP_ACCEPT);//Register channel with the selector
    while (true) {
        selector.select(); // Wait for an event
        Iterator it = selector.selectedKeys().iterator();
        while (it.hasNext()) { // Process each key
            SelectionKey selKey = (SelectionKey)it.next();
            it.remove();
            if (selKey.isAcceptable()) { // Check if it's a connection request
                ServerSocketChannel ssChannel = (ServerSocketChannel)selKey.channel();
                SocketChannel sChannel = ssChannel.accept();
                sChannel.configureBlocking(false);
                sChannel.register(selector, SelectionKey.OP_READ);
            }
        }
    }
} catch (IOException e) { }
```

Ctrl-C interrupt

- Remember always to release resources when Ctrl-C interrupt is fired
  - Handling Ctrl-C interrupt by using addShutdownHook() method for Runtime class

```
Runtime.getRuntime().addShutdownHook(new Thread() {
public void run() {
    System.out.println ("Called at shutdown.");
}
});
```

  - Other alternative is to use handle() method in sun.misc.Signal class

```
public static void main(String[] args) throws Exception {
    Signal.handle(new Signal("INT"), new SignalHandler () {
        public void handle(Signal sig) {
        System.out.println("Received a interrupt!!");
    }
    });
}
```


Command Line Parsing

public static void main(String[] args)

```
// String array containing the program arguments
// Example iterating through array
for (int i = 0; i < args.length; i++) {
   String type = args[i++];
   String value = args[i];
   if(type.equalsIgnoreCase("-l")){
      // use value
      setExampleParameter( value );
   }
}
```

- Or use the existing packages like:
  - args4j, see https://args4j.dev.java.net/
  - Apache Commons CLI, see http://commons.apache.org/cli/

Useful Pointers

- http://java.sun.com/docs/books/tutorial/networking/index.html
- http://java.sun.com/j2se/1.4.2/docs/guide/nio/
- http://www.exampledepot.com/egs/java.net/pkg.html?
- http://www.exampledepot.com/egs/java.nio/pkg.html?
- http://www.java2s.com/Tutorial/Java/0320__Network/Catalog0320__Network.htm