Introduction to Network Programming Using Java

Java starting point

- Development platform
  - Unix/Linux/Windows available in the department or computing center
  - More information http://www.tkk.fi/cc/computers/
  - Using Sun JDK
- Deployment platform
  - Your program must run on desktop at Maari-A

Java starting point (2)

- Working with development tools
  - Using IDE (Eclipse, NetBeans, JCreator ...)
  - Use existing libraries (Apache Commons ...)
  - Use of existing protocol implementations is forbidden
  - Automate compiling (Apache Ant) and testing (JUnit)
  - Both programs are available in TKK linux machines
- Try version control systems to share your code with in your group
  - http://goblin.tkk.fi/c++/tutorials/svn.html

Java starting point (3)

- Information sources
  - Today's slides and examples
  - Sun Java Documentation
  - Examples and tutorials available via search engines
  - Send mail to assistants (if everything else has failed)

Basic concepts

- ... concerning Java programming in general
  - Environment
  - Handling Streams
  - Handling Channels
  - Handling byte arrays
- ... concerning network programming
  - Resolving hostname
  - Handling address information
  - Creating Sockets
  - Sending and receiving data using blocking / non-blocking methods

Parse Command Line in Java

```
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:

- Apache Commons CLI, see http://commons.apache.org/cli/

Resolve hostname

- Transform a symbolic name into a protocol-specific address
- Select the most suitable implementation for the specific task
- InetAddress class for 32-bit and 128-bit IP addresses used for unicast or multicast traffic
- InetSocketAddress class is an implementation for the IP address and port number pair used by sockets for binding and connecting

- API classes
  - java.net.InetAddress
  - java.net.InetSocketAddress

Socket Creation (blocking)

```
java.net.Socket
java.net.ServerSocket
java.net.DatagramSocket
java.net.MulticastSocket
```

Opening a socket and using a stream for communication

java.net.Socket()
Creates an unconnected socket, with the system-default type of SocketImpl.

java.net.Socket(InetAddress address, int port)
Creates a stream socket and connects it to the specified port number at the specified IP address.

java.net.ServerSocket()
Creates an unbound server socket.

java.net.ServerSocket(int port)
Creates a server socket, bound to the specified port.

Socket Creation (non-blocking)

java.nio.channels.SocketChannel
java.nio.channels.ServerSocketChannel

Opening a socket and using a channel for communication

```
InetSocketAddress isa
            = new InetSocketAddress(targetAddrs, targetPort);
// Connect
SocketChannel sChannel
            = SocketChannel.open();
sChannel.configureBlocking(false);
boolean connected = sChannel.connect(isa);

if(connected == false){
    sChannel.finishConnect();
}
```

Sending data (blocking)

- Connection-oriented (TCP)
  - java.net.Socket(InetAddress address, int port)
    - Creates a stream socket and connects it to the
    - specified port number at the specified IP address.
  - java.net.Socket.getOutputStream()
    - Write into OutputStream using suitable Stream writers

Sending data (blocking)

- Connectionless (UDP)
  - java.net.DatagramSocket(int port)
    - Constructs a datagram socket and binds it to the specified port on the local host machine.
  - java.net.DatagramPacket(byte[] buf, int length, InetAddress address, int port)
    - Constructs a datagram packet for sending packets of length to the specified port number on the specified host.
  - java.net.DatagramSocket.send(DatagramPacket p)
    - Sends a datagram packet from this socket.

Receiving (blocking)

- Data reception (TCP) using a Socket
  - InputStream Socket.getInputStream()
  - Read InputStream using suitable classes
- Data reception (UDP) using a DatagramSocket
  - DatagramSocket.receive(DatagramPacket pPacket)
  - Receives a datagram packet from this socket. The DatagramPacket contains the bytes transmitted.
  - To modify socket behavior check the setter methods of the specified implementation

Sending data (non-blocking)

```
//
// SocketChannel sChannel

try {
        String message = "PD course";
        ByteBuffer buf = ByteBuffer.wrap( message.getBytes() );
        sChannel.write(content);
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Receiving data (non-blocking)

```
//
// SocketChannel sChannel
// CharsetDecoder decoder

ByteBuffer dbuf = ByteBuffer.allocateDirect(1024);
CharBuffer cb = null;
int readCount = -1;
try {
                    dbuf.clear();
                    readCount = sChannel.read(dbuf);
                    dbuf.flip();
                    cb = decoder.decode(dbuf);
                    dbuf.flip();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Byte array operations

Using byte array or java.nio.ByteBuffer

```
// array operations
byte[] array = new byte[64];
int arrayLength = array.lenght;
byte[] content = new byte[arrayLength];
System.arraycopy(array, 0, content, 0, arrayLength);

// ByteBuffer
String example = "Hello";
ByteBuffer buffer = ByteBuffer.wrap( example.getBytes() );
ByteBuffer buffer2 = buffer.dublicate();
buffer2.order( ByteOrder.BIG_ENDIAN);
byte[] array2 = buffer2.array();
```

Or use existing libraries like Apache Commons IO
http://commons.apache.org/io/api-release/index.html

Concurrency

Event Based (Single Thread Handling many connections)

```
See event based solution from examples using java.nio.Channels
```

Using Threads

```
//
// ReceiverThread implements Runnable interface
ReceiverThread reveicerConnection = new ReceiverThread();

receiver = new Thread(reveicerConnection);
receiver.start();
```

For the beginners read tutorials like

http://java.sun.com/docs/books/tutorial/essential/concurrency/
http://java.sun.com/j2se/1.5.0/docs/guide/concurrency/index.html
http://www.ibm.com/developerworks/edu/j-dw-javathread-i.html

Others (1)

- Try to keep your classes as simply as possible
  - group a certain set of functionalities into a specified class
- Use design patterns to get a controlled structure for your program
  - For example Observer – Observable pattern can be used to deliver the received data for multiple users
  - i.e. Server must replicate data for multiple receivers:
    - Socket container (source) implements Observable interface
    - Client connection creates an instance of the client container that implements an Observer interface
    - When client container is created the client observer is registered to the observable source

Others (2)

Remember always to terminate program and release resources

```
To handle shutdown signal use 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 to catch signals
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!!");
      }
    });
  //
  }
```