Networking

Outline: Networking

Presents the network programming in Java language.

Main points:

1) Review the basic network concepts and Java Implementation.
2) Discuss the usage of java.net package.
3) Introduce the Secure Socket.
4) Introduce the New I/O API.
5) Introduce the Java implementation for UDP protocol.

Why Java

Why use Java for Networking?

a) Java was the first programming language designed from the ground up with networking in mind.

b) Java provides easy solutions to two crucial problem for Internet networking — platform independence and security.

c) It is far easier to write network programs in Java than in almost any other language.

- In the fully functional applications, very little code is devoted to networking.

Network programs with Java 1

Examples that a Network Program can do:

a) Server-Client

Examples: RssOwl (http://rssowl.sourceforge.net/)

b) Peer-to-Peer

Examples: LimeWire (http://limewire.org/)

Azureus (http://azureus.sourceforge.net/)

Concepts for network program

Important concepts needed for writing network program in Java

1) Communication protocols: TCP and UDP
2) Ports and Internet Addresses
3) Sockets
4) Uniform Resource Locator (URL)
5) Uniform Resource Identifier (URI)
6) Streams and Threads (Covered in previous sessions)
7) Classes in java.net and java.io packages

TCP and UDP

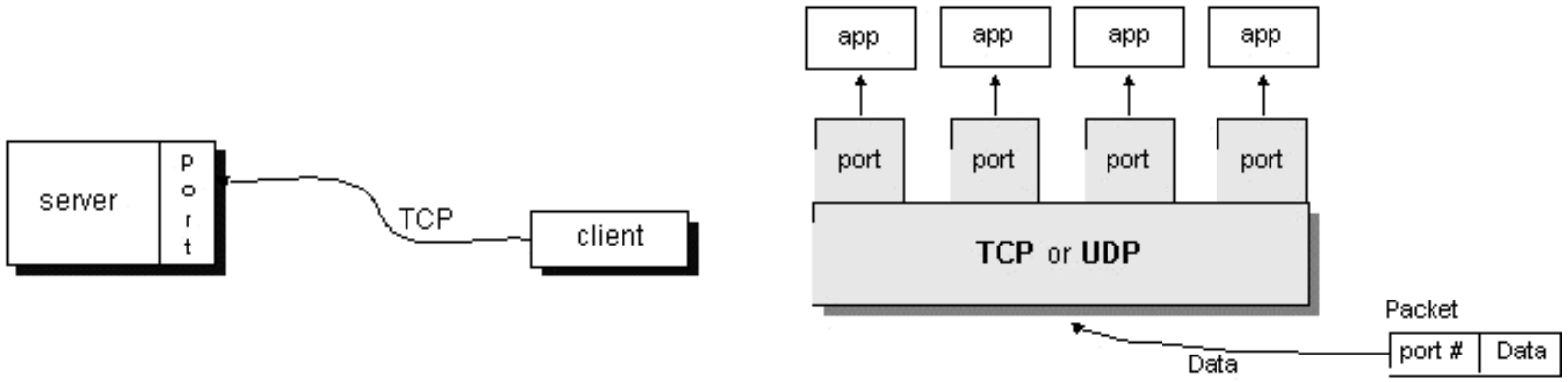
Java only supports TCP (Transmission Control Protocol), UDP (User Datagram Protocol) and application layer protocols built on top of these.

Characteristics of TCP and UDP :

TCP	UDP
• Provides the ability to acknowledge receipt of IP packets and request retransmission of lost or corrupted packets. • Allows the received packets to be put back together in the order they were sent. • Requires a lot of overhead. • Supported classes in java.net : URL, URLConnection, Socket, and ServerSocket	• Is an unreliable protocol that does not guarantee that packets will arrive at their destination. • Allow the receiver to detect corrupted packets but does not guarantee that packets are delivered in the correct order. • Requires less overhead and faster. • Supported classes in java.net : DatagramPacket, DatagramSocket, and MulticastSocket

Ports

Each port from the server can be treated by the clients as a separate machine offering different services.



Port numbers are represented by 16-bit numbers. (0 to 65,535)

The port numbers ranging from 0 - 1023 reserved for use by well-known services such as HTTP and FTP and other system services.

Sockets

You can reach required service via its network and port IDs. what then?

a) If you are a client
   - you need an API that will allow you to send messages to that service and read replies from it

b) If you are a server
   - you need to be able to create a port and listen at it.
   - you need to be able to read the message comes in and reply to it.

The **Socket** and **ServerSocket** are the Java client and server classes to do this.

Example : Sending Email 1

E-mail is sent by socket communication with port 25 on a computer system.

open a socket connected to port 25 on some system, and speak “mail protocol” to the daemon at the other end.

Example : Sending Email 2

```
import java.io.*;
import java.net.*;
public class SendEmail {
  public static void main(String args[]) throws
   IOException {
      Socket sock;
      DataInputStream dis;
      BufferedReader br;
      PrintStream ps;
      System.out.println(">>> Connect
                  mailhost.test.com");
      sock = new Socket("mailhost.test.com", 25);
      dis = new DataInputStream(sock.getInputStream());
```

Example : Sending Email 3

```
 br = new BufferedReader (new
    InputStreamReader(dis));
 ps = new PrintStream( sock.getOutputStream());
System.out.println( br.readLine() );
System.out.println(">>> Hello ITB");
ps.println("Hello UNU/ITB");
System.out.println( br.readLine() );
System.out.println(">>> Mail From: sparkymarky
 @gmail.com");
ps.println("MAIL FROM:sparkymarky@gmail.com");
System.out.println( br.readLine() );
String Addressee= "sparky@test.com";
```

Example : Sending Email 4

```
System.out.println(">>> Rcpt to: " + Addressee);
ps.println("RCPT TO: " + Addressee );
System.out.println( br.readLine() );
System.out.println(">>> Send \"data\"");
ps.println("DATA");
System.out.println( br.readLine() );
System.out.println(">>>>>>>>>>");
System.out.println(">>> This is the message\n that
 Java sent");
System.out.println(">>> We are testing Socket
 Programming");
System.out.println(">>>>>>>>>>");
```

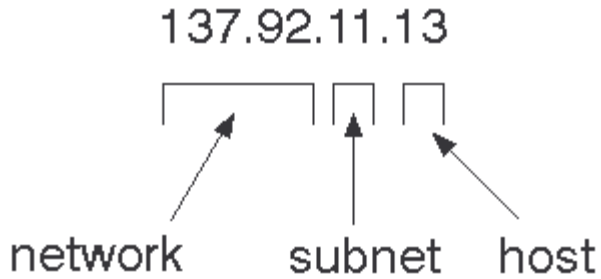
Example : Sending Email 5

```
    ps.println("This is the message\n that Java
    sent");
     ps.println("We are testing Socket Programming");
     System.out.println(">>> .");
     ps.println(".");
     System.out.println( br.readLine() );
     System.out.println(">>> QUIT");
     ps.println("QUIT");
     System.out.println( br.readLine() );
     ps.flush();
     sock.close();
   }
}
```

Internet Addressing

Internet address (IP address) is a unique number for identifying a device connected to the Internet.

The current standard is IPv4 which are four bytes long.



The hostname and IP address is, in Java, represented by java.net.InetAddress.

InetAddress is used by many other networking classes, including Socket, ServerSocket, URL, DatagramSocket, DatagramPacket, and more.

Example

The following program will print out the IP address of the www.iist.unu.edu

```
import java.net.*;
public class IISTByName {
  public static void main (String[] args) {
   try { InetAddress address =
     InetAddress.getByName   ("www.test.com") ;
     System.out.println(address);
   } catch (UnknownHostException ex) {
       System.out.println("Could not find
          www.test.com ");
   }
  }
}
```

InetAddress methods

Useful methods:

a) static InetAddress getByName(String host)

b) static InetAddress getLocalHost()

c) String getHostAddress(); // in dotted form

d) String getHostName();

The URL Class 1

The `java.net.URL` is an abstraction of a Uniform Resource Locator (URL).

URLs are composed of five pieces:

1. The scheme, also known as the protocol
2. The authority
3. The path
4. The query string
5. The fragment identifier, also known as the section or ref

<scheme>://<authority><path>?<query>#<fragment>

The URL Class 2

For example, given the URL :

http://www.ibiblio.org/javafaq/javabooks/index.html?isbn=123456789#toc

1. scheme : http
2. authority : www.ibiblio.org
3. path : /javafaq/books/javabooks/index.html
4. query string : isbn=123456789
5. fragment identifier : toc

The URL Class 3

The authority may further be divided into the user info, the host, and the port.

For example, in the URL http://admin@www.blackstar.com:8080/

1. user info : admin
2. host : www.blackstar.com
3. port : 8080

The URL Class 4

The java.net.URL class provides static methods for getting the above mentioned information:

a) getFile( )
b) getHost()
c) getPort( )
d) getProtocol()
e) getRef()
f) getQuery( )
g) getPath( )
h) getUserInfo( )
i) getAuthority( )

The URL Class 5

Unlike the InetAddress objects, you can construct instances of java.net.URL using one of its six constructors.

1) `public URL(String url) throws MalformedURLException`

2) `public URL(String protocol, String hostname, String file) throws MalformedURLException`

3) `public URL(String protocol, String host, int port, String file) throws MalformedURLException`

4) `public URL(URL base, String relative) throws MalformedURLException`

The URL Class 6

5) `public URL(URL base, String relative, URLStreamHandler handler) // 1.2 throws MalformedURLException`

6) `public URL(String protocol, String host, int port, String file, // 1.2 URLStreamHandler handler) throws MalformedURLException`

Example 1

The following program will test the protocol supported by the browser:

```
import java.net.*;
public class ProtocolTester {
    public static void testProtocol(String url) {
     try {
       URL u = new URL(url);
       System.out.println(u.getProtocol( ) + " is
        supported");
     }
     catch (MalformedURLException ex) {
       String protocol =
        url.substring(0, url.indexOf(':'));
       System.out.println(protocol + " is not
        supported");
     }
   }
}
```

Example 2

You can test it with the following Tester:

```
public class Tester {
  public static void main(String[] args) {
    ProtocolTester.testProtocol("http://www.adc.org");
    ProtocolTester.testProtocol("https://www.amazon.co
    m/exec/obidos/order2/");
    ProtocolTester.testProtocol("ftp://metalab.unc.edu
    /pub/languages/java/javafaq/");
```

Retrieving Data from a URL

The URL class provides methods for retrieving data from a URL:

```
public InputStream openStream( ) throws IOException

public URLConnection openConnection( ) throws
IOException

public URLConnection openConnection(Proxy proxy)
throws IOException // 1.5

public Object getContent( ) throws IOException

public Object getContent(Class[] classes) throws
IOException // 1.3
```

Retrieving Data from a URL 1

Procedure to use the methods:

1) Create an URL object

   e.g. `URL u = new URL("http://www.test.com");`

2) Open an InputStream object directly from the URL object

   e.g. `InputStream in = u.openStream( );`

3) Or open an URLConnection object from the URL object and then get an InputStream object from the URLConnection object .

   e.g.
```
URLConnection uc = u.openConnection( );
InputStream in = uc.getInputStream( );
```

4) In either case, you will have an InputStream. What's followed is the normal I/O procedure for getting data.

5) Don't forget to put the try catch block for catching the `MalformedURLException` and `IOException`.

Retrieving Data from a URL 2

What is the difference between using the openStream and openConnection method?

1) openStream method only give you the access to the raw data and cannot detect the encoding information.
2) openConnection method opens a socket to the specified URL and returns a URLConnection object.
3) The URLConnection object gives you access to everything sent by the server. You can access all the metadata specified by the protocol such as the scheme. The URLConnection class also lets you write data to as well as read from a URL.

Retrieving Data from a URL 3

The following methods are used to access the header fields and the contents after the connection is made to the remote object:

1)getContent

2)getHeaderField

3)getInputStream

4)getOutputStream

Retrieving Data from a URL 4

Certain header fields are accessed frequently. The methods:

1) getContentEncoding
2) getContentLength
3) getContentType
4) getDate
5) getExpiration
6) getLastModifed

Example : Reading from URL 1

```
import java.net.*;
import java.io.*;
public class URLConnectionReader {
  ublic static void main(String[] args) throws
   Exception {
        URL yahoo = new URL("http://www.yahoo.com/");
        URLConnection yc = yahoo.openConnection();
        BufferedReader in = new BufferedReader(
        new InputStreamReader (yc.getInputStream()));
        String inputLine;
```

Example : Reading from URL 2

```
while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);
        in.close();
    }
}
```

Uniform Resource Identifier (URI)

A Uniform Resource Identifier (URI) is an abstraction of a URL.

Most URIs used in practice are URLs, but most specifications and standards such as XML are defined in terms of URIs

In Java 1.4 and later, URIs are represented by the `java.net.URI` class.

you should use the URL class when you want to download the content of a URL and the URI class when you want to use the URI for identification rather than retrieval.

When you need to do both, you may convert from a URI to a URL with the toURL( ) method, and in Java 1.5 you can also convert from a URL to a URI using the toURI( ) method of the URL class.

Uniform Resource Identifier (URI)

A URI reference has up to three parts: a scheme, a scheme-specific part, and a fragment identifier. The general format is:
scheme:*scheme-specific-part*:*fragment* .

Getter methods:

1) public String getScheme( )
2) public String getSchemeSpecificPart( )
3) public String getRawSchemeSpecificPart( )
4) public String getFragment( )
5) public String getRawFragment( )

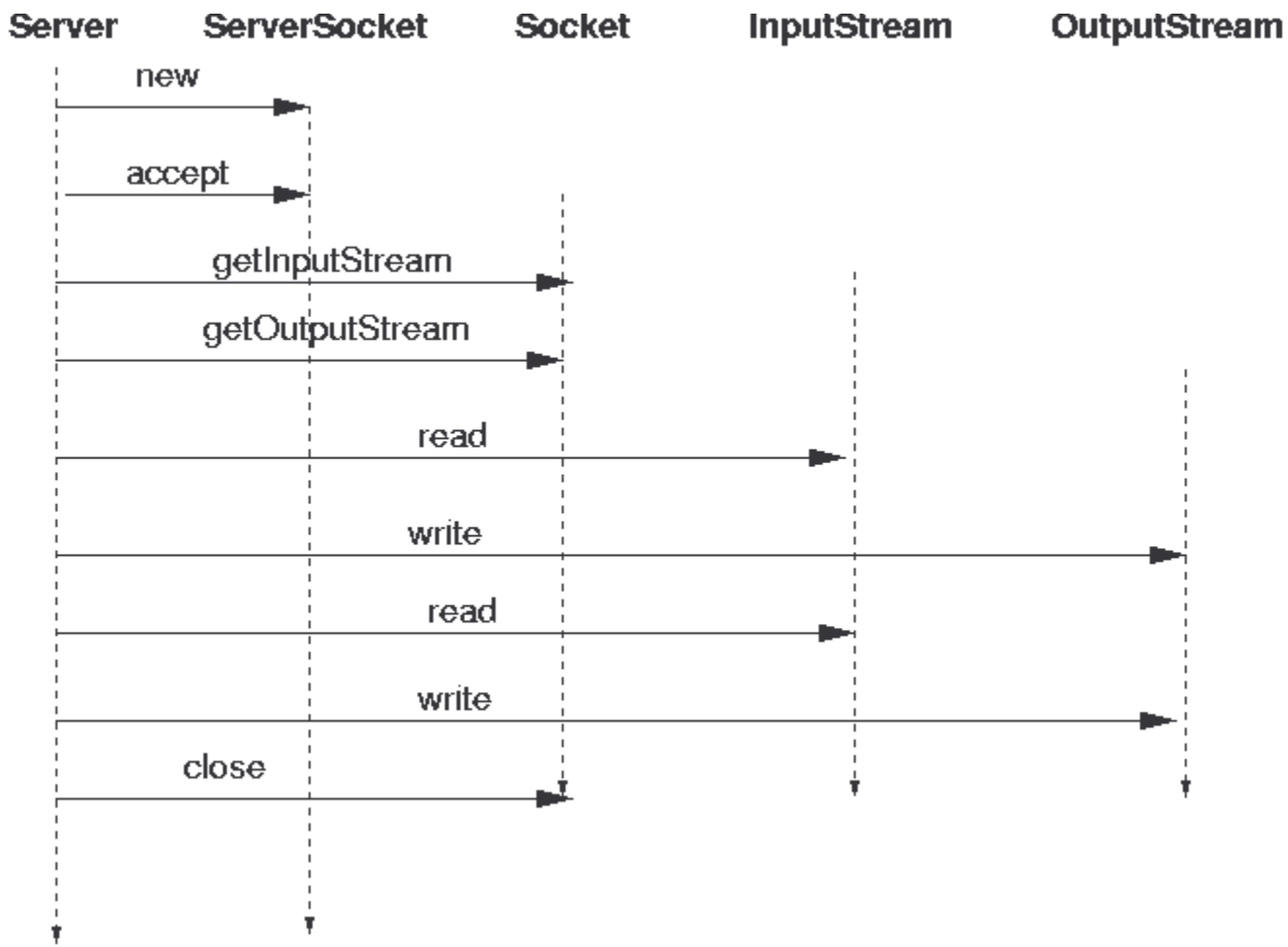
Networking Examples

Now you have the basic concepts for different components for Java networking. Let's try to start some simple experiments.

A Simple Example

Create a Server

How to create a server?



Echo Server 1

```
import java.io.*;
import java.net.*;
public class EchoServer {
  public static int MYECHOPORT = 8189;
    public static void main(String argv[]) {
      ServerSocket s = null;
      try {
          s = new ServerSocket(MYECHOPORT);
      } catch(IOException e) {
          System.out.println(e);
          System.exit(1);
      }
```

Echo Server 2

```
while (true) {
        Socket incoming = null;
        try {
          incoming = s.accept();
        } catch(IOException e) {
          System.out.println(e);
          continue;
        }
        try {
          incoming.setSoTimeout(10000); //10 seconds
        } catch(SocketException e) {
          e.printStackTrace();
        }
```

Echo Server 3

```
try {         handleSocket(incoming);
            } catch(InterruptedIOException e) {
              System.out.println("Time expired " + e);
            } catch(IOException e) {
              System.out.println(e);
            }
            try {
              incoming.close();
            } catch(IOException e) {
              // ignore
            }
      }
    }
```

Echo Server 4

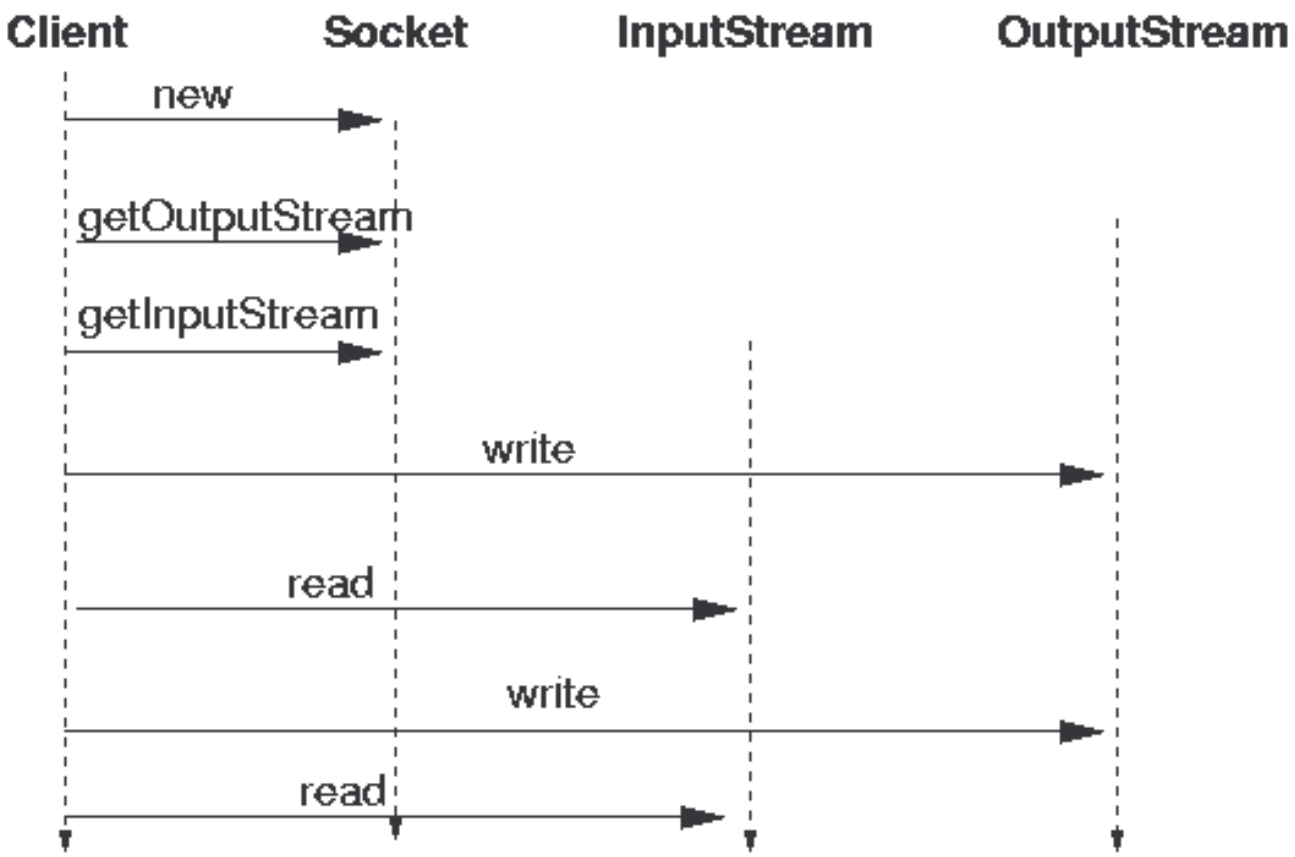
```
public static void handleSocket(Socket incoming)
      throws IOException {
      BufferedReader reader =
          new BufferedReader(new InputStreamReader(
                        incoming.getInputStream()));
      PrintStream out =
          new PrintStream(incoming.getOutputStream());
      out.println("Hello. Enter BYE to exit");

      boolean done = false;
      while ( ! done) {
          String str = reader.readLine();
```

Echo Server 5

```
if (str == null) {
            done = true;
            System.out.println("Null received");
          }
          else {
            out.println("Echo: " + str);
            if (str.trim().equals("BYE"))
                done = true;
          }
            }
      incoming.close();
    }
}
```

Create a Client

How to create a client connected to a sever?



Echo Client 1

```
import java.io.*;
import java.net.*;


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

            Socket echoSocket = null;
            PrintWriter out = null;
            BufferedReader in = null;
            BufferedReader stdIn = null;
```

Echo Client 2

```
try {
        echoSocket = new Socket("localhost", 8189);
        out = new
        PrintWriter(echoSocket.getOutputStream(), true);
        in = new BufferedReader(new
        InputStreamReader(echoSocket.getInputStream()));
        System.out.println (in.readLine());
        } catch (UnknownHostException e) {
              System.err.println("Don't know about host.");
              System.exit(1);
        } catch (IOException e) {
             System.err.println("Couldn't get I/O for "
              + "the connection to server.");
```

Echo Client 3

```
        System.exit(1);
}
try {
        stdIn = new BufferedReader(new InputStreamReader
                (System.in));
        String userInput;
        while ((userInput = stdIn.readLine()) != null) {
                out.println(userInput);
                System.out.println("echo: " + in.readLine());
         }
        } catch (SocketException e) {
                        System.err.println("Socket closed");
        }
```

Echo Client 4

```
finally {
                    if (out != null)
                        out.close();
                    if (in != null)
                        in.close();
                    if (stdIn != null)
                        stdIn.close();
                    if (echoSocket != null)
                        echoSocket.close();
                }
        }
}
```

Secure Sockets

Starting from JDK 1.4, Java Secure Sockets Extension (JSSE) is part of the standard distribution.

JSSE uses the Secure Sockets Layer (SSL) Version 3 and Transport Layer Security (TLS) protocols and their associated algorithms to secure network communications.

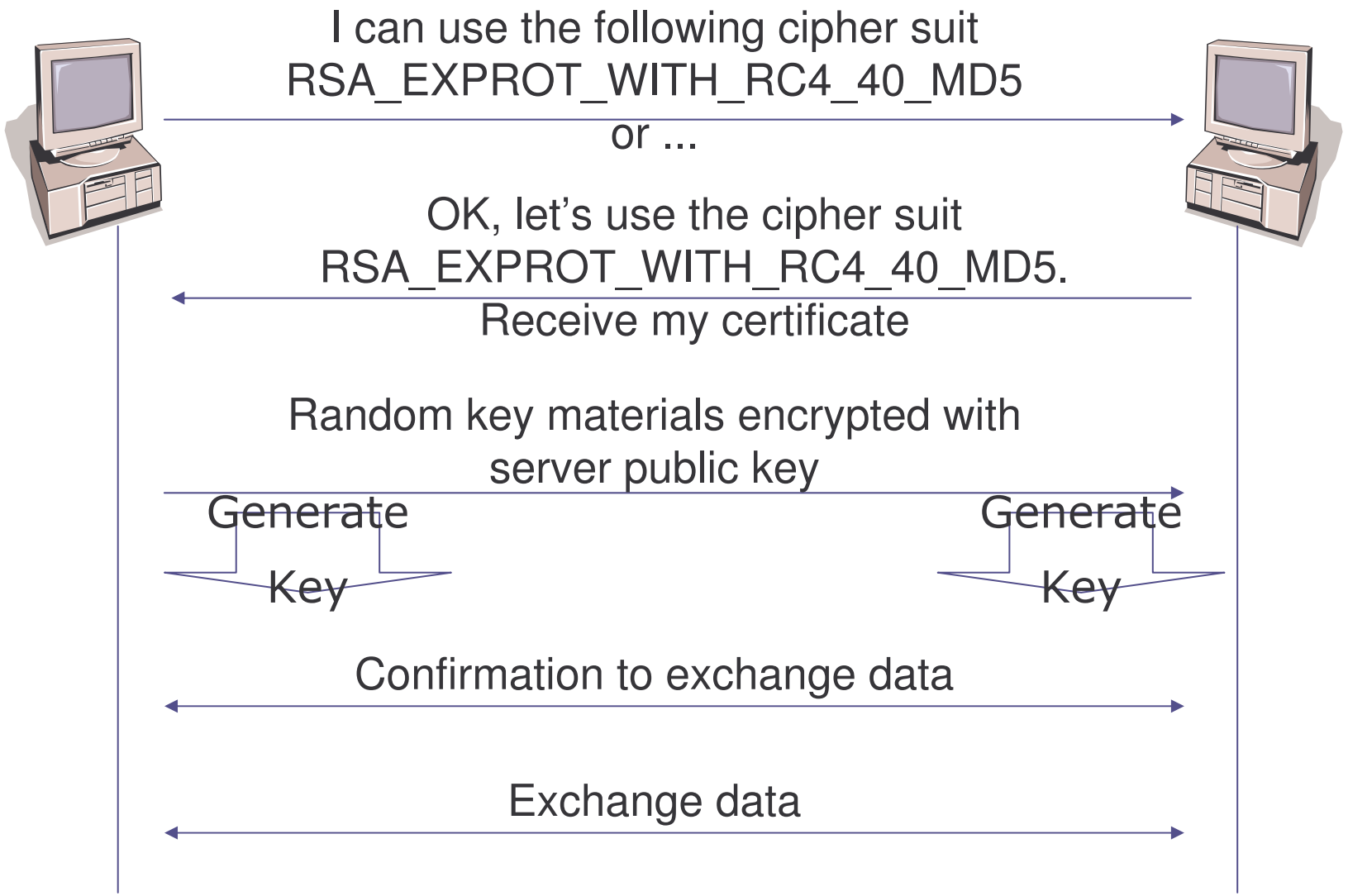
JSSE abstracts all the low-level details such as keys exchange, authentication, and data encryption. All you have to do is to send your data over the streams from the secured sockets obtained.

Java Secure Socket Extension

The Java Secure Socket Extension is divided into four packages:

1) javax.net.ssl :The abstract classes that define Java's API for secure network communication.
2) javax.net :The abstract socket factory classes used instead of constructors to create secure sockets.
3) javax.security.cert : A minimal set of classes for handling public key certificates that's needed for SSL in Java 1.1. (In Java 1.2 and later, the java.security.cert package should be used instead.)
4) com.sun.net.ssl : The concrete classes that implement the encryption algorithms and protocols in Sun's reference implementation of the JSSE.

SSL Handshake



Secure Client Sockets 1

Procedures to create a secure client socket:

1) get an instance of `SocketFactory` by invoking the static `SSLSocketFactory.getDefault()` method. e.g.

```
SocketFactory sf = SSLSocketFactory.getDefault();
```

2) use one of these five overloaded `createSocket( )` methods to build an SSLSocket:

1. `public abstract Socket createSocket(String host, int port) throws IOException, UnknownHostException`
2. `public abstract Socket createSocket(InetAddress host, int port) throws IOException`

Secure Client Sockets 2

3. `public abstract Socket createSocket(String host, int port, InetAddress interface, int localPort) throws IOException, UnknownHostException`
4. `public abstract Socket createSocket(InetAddress host, int port, InetAddress interface, int localPort) throws IOException, UnknownHostException`
5. `public abstract Socket createSocket(Socket proxy, String host, int port, boolean autoClose) throws IOException`

Secure Client Sockets 3

3) Once the socket has been created, you use it just like any other socket, through its getInputStream( ), getOutputStream(), and other methods.

   For example, if the following purchasing information is required to be sent over the network:

   a) Name: John Smith
   b) Product-ID: 67X-89
   c) Address: 123 Fake Street, Dub 15
   d) Card number: 4000-1234-5678-9017
   e) Expires: 08/10

Using JSSE, the following code will do the work for you:

Secure Client Sockets 4

```
try {
    SSLSocketFactory factory
    = (SSLSocketFactory) SSLSocketFactory.getDefault( );
    Socket socket = factory.createSocket("localhost",
       7000);
    Writer out = new
       OutputStreamWriter(socket.getOutputStream( ),
       "ASCII");
    out.write("Name: John Smith\r\n");
```

Secure Client Sockets 5

```
        out.write("Product-ID: 67X-89\r\n");
        out.write("Address: 123 Fake Street, Dub 15\r\n");
    out.write("Card number: 4000-1234-5678-9017\r\n");
    out.write("Expires: 08/10\r\n");
    out.flush( );
    out.close( );
    socket.close( );
} catch (IOException ex) {
      ex.printStackTrace( );
}
```

Configuring Secure Sockets

Methods are available for configuring how much and what kind of authentication and encryption is performed.

a) getSupportedCipherSuites() method tells you which combination of algorithms is available on a given socket

b) getEnabledCipherSuites() method tells you which suites this socket is willing to use

c) You can change the suites the client attempts to use via the setEnabledCipherSuites(String[] suites) method

- Sun's JDK 1.4 supports 23 cipher suites. For the list of the supported cipher suites, please check with JavaDoc.

d) There are still methods for handling handshaking and sessions, and I will open these for your further study.

Secure Server Sockets 1

Procedures to create a secure server socket:

1) get an instance of `ServerSocketFactory` by invoking the static `SSLSocketFactory.getDefault()` method. e.g.

```
ServerSocketFactory sf =
  SSLServerSocketFactory.getDefault();
```

2) use one of these three overloaded `createServerSocket( )` methods to build an SSLServerSocket:

```
1.public abstract ServerSocket
  createServerSocket(int port) throws IOException

2.public abstract ServerSocket
  createServerSocket(int port, int queueLength)
  throws IOException

3.public abstract ServerSocket
  createServerSocket(int port, int queueLength,
  InetAddress interface) throws IOException
```

Secure Server Sockets 2

3) Unlike creating the client socket, you need to do more to set up the encryption for the server socket.

   This setup varies between different JSSE implementations. In Sun's implementation, you may need to do the followings:

   1. Generate public keys and certificates using *keytool*.
   2. Pay money to have your certificates authenticated by a trusted third party such as Verisign.
   3. Create an SSLContext for the algorithm you'll use.
   4. Create a TrustManagerFactory for the source of certificate material you'll be using.

Secure Server Sockets 3

5. Create a KeyManagerFactory for the type of key material you'll be using.
6. Create a KeyStore object for the key and certificate database. (Sun's default is JKS.)
7. Fill the KeyStore object with keys and certificates; for instance, by loading them from the filesystem using the pass phrase they're encrypted with.
8. Initialize the KeyManagerFactory with the KeyStore and its pass phrase.
9. Initialize the context with the necessary key managers from the KeyManagerFactory, trust managers from the TrustManagerFactory, and a source of randomness. (The last two can be null if you're willing to accept the defaults.)

New I/O (NIO) API

Java introduce the new I/O (NIO) API in v1.4.

New features:

a) Buffers for data of primitive types

b) Character-set encoders and decoders

c) A pattern-matching facility based on Perl-style regular expressions

d) Channels, a new primitive I/O abstraction

e) A file interface that supports locks and memory mapping

f) A multiplexed, non-blocking I/O facility for writing scalable servers

Why NIO?

Allow Java programmers to implement high-speed I/O.

NIO deals with data in blocks which can be much faster than processing data by the (streamed) byte.

NIO Components

1) Buffers
2) Channels
3) Selectors
4) Regular Expressions
5) Character Set Coding

Buffers

In the NIO library, all data is handled with buffers.

A buffer is essentially an array. Generally, it is an array of bytes, but other kinds of arrays can be used.

A buffer also provides structured access to data and also keeps track of the system's read/write processes.

Types:

a) ByteBuffer
b) CharBuffer
c) ShortBuffer
d) IntBuffer
e) LongBuffer·
f) FloatBuffer
g) DoubleBuffer

Channels

Channel is like a stream in original I/O.

You can read a buffer from and write a buffer to a channel.

Unlike streams, channels are bi-directional.

Read from a file

Codes for reading from a file:

```
FileInputStream fin = new
FileInputStream( "readandshow.txt" );

FileChannel fc = fin.getChannel();

ByteBuffer buffer = ByteBuffer.allocate( 1024 );

fc.read( buffer );
```

Write to a file

Codes for writing to a file:

```
FileOutputStream fout = new
FileOutputStream( "writesomebytes.txt" );

FileChannel fc = fout.getChannel();

ByteBuffer buffer = ByteBuffer.allocate( 1024 );

for (int i=0; i<message.length; ++i) {

buffer.put( message[i] );

}

buffer.flip();//prepares the buffer to have the newly-
              //read data written to another channel

fc.write( buffer );
```

Server with NIO

Channels and buffers are really intended for server systems that need to process many simultaneous connections efficiently.

Handling servers requires the new selectors that allow the server to find all the connections that are ready to receive output or send input.

Asynchronous I/O

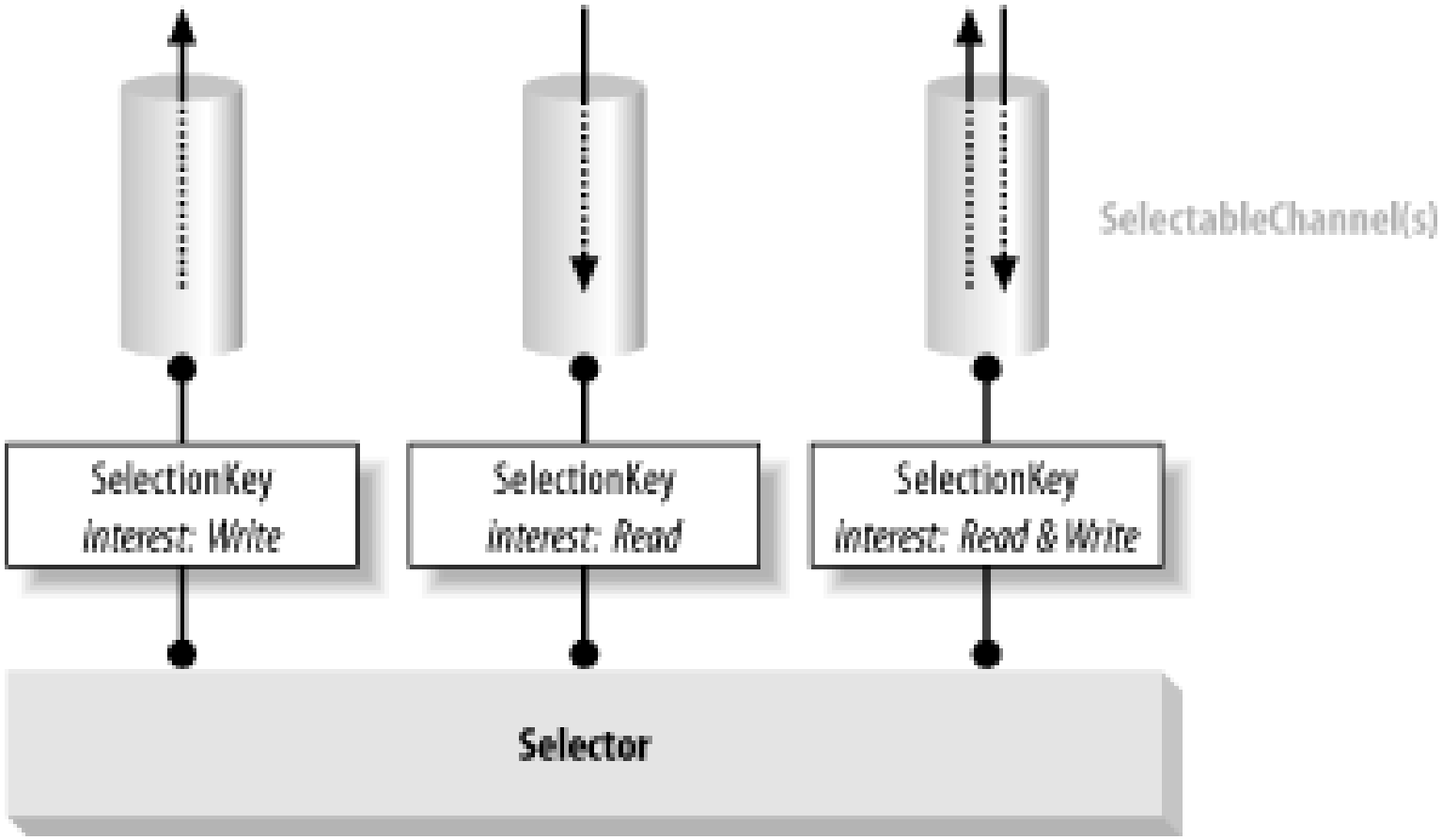
Asynchronous I/O is made possible in NIO with scalable sockets, which consist of the following components:

a) Selectable Channel - A channel that can be multiplexed

b) Selector - A multiplexor of selectable channel

c) Selection key - A token representing the registration of a selectable channel with a selector

Selectors, Keys and Channels



The Selection Process 1

1) Create a Selector and register channels with it

   Use `Open()` method to create a Selector.

   The `register()` method is on `SelectableChannel`, not `Selector`

2) Invoke `select()` method on the `Selector` object

3) Retrieve the Selected Set of keys from the Selector

   Selected set: Registered keys with non-empty Ready Sets

   ```
   keys = selector.selectedKeys()
   ```

The Selection Process 2

4) Iterate over the Selected Set
    1) Check each key's Ready Set
    2) Remove the key from the Selected Set (`iterator.remove()`)
        1) Bits in the Ready Sets are never reset while the key is in the Selected Set
        2) The Selector never removes keys from the Selected Set – you must do so
    3) Service the channel (`key.channel()`) as appropriate (read, write, etc)

Example : NIO Server 1

Skeleton dodes for a simple server with NIO:

```
  //Open a ServerSocketChannel
  ServerSocketChannel serverChannel =
  ServerSocketChannel.open( );
//make the ServerSocketChannel non-blocking.
//necessary for asynchronous i/o
  serverChannel.configureBlocking(false);
  ServerSocket ss = serverChannel.socket( );
  // bind the socket to a specific port
  ss.bind(new InetSocketAddress(PORT_NO));
```

Example : Create NIO Server 2

```
//Open the selector

  Selector selector = Selector.open( );

//use the channel's register() method to register the
  //ServerSocketchannel with the selector.


  serverChannel.register(selector,
  SelectionKey.OP_ACCEPT);
```

Example : Create NIO Server 3

```
//check whether anything is ready to be acted on, call
//the selector's select( ) method. For a long-running
//server, this normally goes in an infinite loop:
while (true) {
    try {
      selector.select( );
    }
    catch (IOException ex) {
      ex.printStackTrace( );
      break;
    }
```

Example : Create NIO Server 4

```
 // process selected keys...

//selectedKeys( ) method returns a java.util.Set
//containing one SelectionKey object for each ready
//channel

 Set readyKeys = selector.selectedKeys( );
Iterator iterator = readyKeys.iterator( );
while (iterator.hasNext( )) {

    SelectionKey key = (SelectionKey)

    (iterator.next( ));

     // Remove key from set

    iterator.remove( );
```

Example : Create NIO Server 5

```
  // You can obtain the channel using the channel()
  //methods of the SelectionKey. Catch the IOExcepton.
try{
      if (key.isAcceptable( ))
        { ServerSocketChannel
          server = (ServerSocketChannel ) key.channel( );
           SocketChannel client = server.accept( );
        // Data manipulation
         System.err.println("Got connection from
  "+client.socket().getInetAddress().getHostName());
```

Example : Create NIO Server 6

```
                //Send some message to client

                ByteBuffer bb =
                ByteBuffer.allocateDirect(1024);

                byte[] message = "Hello... You are
                reaching NIO Server".getBytes();

                bb.put( message);

                bb.flip();

                client.write( bb);

                }

            } catch (IOException e) {   }

    }

}
```

User Datagram Protocol

The User Datagram Protocol (UDP) is an alternative protocol for sending data over IP

UDP is very quick, but not reliable.

Java's implementation of UDP is split into two classes: DatagramPacket and DatagramSocket.

The DatagramPacket class stuffs bytes of data into UDP packets called datagrams and lets you unstuff datagrams that you receive.

A DatagramSocket sends as well as receives UDP datagrams.

Constructors for DatagramPacket

For receiving datagrams:

a) `public DatagramPacket(byte[] buffer, int length)`

b) `public DatagramPacket(byte[] buffer, int offset, int length)`

For sending datagrams:

a) `public DatagramPacket(byte[] data, int length, InetAddress destination, int port)`

b) `public DatagramPacket(byte[] data, int offset, int length, InetAddress destination, int port) // Java 1.2`

c) `public DatagramPacket(byte[] data, int length, SocketAddress destination, int port) // Java 1.4`

d) `public DatagramPacket(byte[] data, int offset, int length, SocketAddress destination, int port) //Java 1.4`

Constructors for DatagramSocket

For socket bound to an anonymous port:

a) `public DatagramSocket( ) throws SocketException`

For socket listen for incoming datagrams on a particular port :

a) `public DatagramSocket(int port) throws SocketException`

Others constructors:

a) `public DatagramSocket(int port, InetAddress interface) throws SocketException`

b) `public DatagramSocket(SocketAddress interface) throws SocketException // Java 1.4`

c) `protected DatagramSocket(DatagramSocketImpl impl) throws SocketException // Java 1.4`

Sending and Receiving

After constructed the DatagramPacket, you can send and receive datagram from it :

a) `public void send(DatagramPacket dp) throws IOException`

b) `public void receive(DatagramPacket dp) throws IOException`