Important Java Packages

- **java.net**  
  TCP/IP networking

- **java.io**  
  I/O streams & utilities

- **java.rmi**  
  Remote Method Invocation

- **java.security**  
  Security policies

- **java.lang**  
  Threading classes
Networking Basics

- Application Layer
  - Standard apps
    - HTTP
    - FTP
    - Telnet
  - User apps
- Transport Layer
  - TCP
  - UDP
  - Programming Interface:
    - Sockets
- Network Layer
  - IP
- Link Layer
  - Device drivers

**TCP/IP Stack**

- Application
  (http, ftp, telnet, …)
- Transport
  (TCP, UDP, ..)
- Network
  (IP, ..)
- Link
  (device driver, ..)
Networking Basics

- **TCP (Transport Control Protocol)** is a connection-oriented protocol that provides a reliable flow of data between two computers.

- **Example applications:**
  - HTTP
  - FTP
  - Telnet

---

**TCP/IP Stack**

- Application (http, ftp, telnet, …)
- Transport (TCP, UDP, …)
- Network (IP, …)
- Link (device driver, …)
Networking Basics

- UDP (User Datagram Protocol) is a protocol that sends independent packets of data, called *datagrams*, from one computer to another with no guarantees about arrival.

- Example applications:
  - Clock server
  - Ping

TCP/IP Stack

- Application (http, ftp, telnet, …)
- Transport (TCP, UDP, …)
- Network (IP, …)
- Link (device driver, …)
Java Sockets Programming

• Java uses BSD-style sockets to interface with TCP/IP services (java.net package)

• Java distinguishes between UDP, TCP server & TCP client sockets

• Behind-the-scenes classes do the actual work & can be updated or swapped out transparently
IP Addresses & Hostnames

- java.net.InetAddress class
- Represents a single IP address
- Factory class – no public constructor
- Performs transparent DNS lookups or reverse lookups
- java.net.UnknownHostException thrown if DNS system can’t find IP address for specific host
TCP Server Sockets

• java.net.ServerSocket class

• Binds to a local port to listen for initial connections

• Can be bound to a local IP for multi-homed machines

• accept() method returns a java.net.Socket, not an integer descriptor
TCP Client Sockets

• java.net.Socket class

• Combines socket with socket options (timeout, linger, keep alive, no delay, etc)

• Encapsulates a java.io.InputStream and a java.io.OutputStream – can be retrieved for use in a layered I/O system
Socket functional calls

- `socket()`: Create a socket
- `bind()`: bind a socket to a local IP address and port #
- `listen()`: passively waiting for connections
- `connect()`: initiating connection to another socket
- `accept()`: accept a new connection
- `Write()`: write data to a socket
- `Read()`: read data from a socket
- `sendto()`: send a datagram to another UDP socket
- `recvfrom()`: read a datagram from a UDP socket
- `close()`: close a socket (tear down the connection)
JAVA TCP Sockets

• In Package java.net
  – java.net.Socket
    • Implements client sockets (also called just “sockets”).
    • An endpoint for communication between two machines.
    • Constructor and Methods
      – Socket(String host, int port): Creates a stream socket and connects it to the specified port number on the named host.
      – InputStream getInputStream()
      – OutputStream getOutputStream()
      – close()

  – java.net.ServerSocket
    • Implements server sockets.
    • Waits for requests to come in over the network.
    • Performs some operation based on the request.
    • Constructor and Methods
      – ServerSocket(int port)
      – Socket Accept(): Listens for a connection to be made to this socket and accepts it. This method blocks until a connection is made.
TCPClient.java

```java
import java.io.*;
import java.net.*;
class TCPClient {
    public static void main(String argv[]) throws Exception {
        String sentence;
        String modifiedSentence;
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));
        Socket clientSocket = new Socket("hostname", 6789);
        DataOutputStream outToServer =
            new DataOutputStream(clientSocket.getOutputStream());
        BufferedReader inFromServer =
            new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
        sentence = inFromUser.readLine();
        outToServer.writeBytes(sentence + '\n');
        modifiedSentence = inFromServer.readLine();
        System.out.println("FROM SERVER: " + modifiedSentence);
        clientSocket.close();
    }
}
```
TCP\texttt{Server.java}

```java
import java.io.*;
import java.net.*;
class TCPServer {
    public static void main(String argv[]) throws Exception {
        String clientSentence;
        String capitalizedSentence;
        ServerSocket welcomeSocket = new ServerSocket(6789);
        while(true) {
            Socket connectionSocket = welcomeSocket.accept();
            BufferedReader inFromClient = new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient = new DataOutputStream(connectionSocket.getOutputStream());
            clientSentence = inFromClient.readLine();
            capitalizedSentence = clientSentence.toUpperCase() + '
';
            outToClient.writeBytes(capitalizedSentence);
        }
    }
}
```
Another simple server

// SimpleServer.java: a simple server program
import java.net.*;
import java.io.*;
public class SimpleServer {
    public static void main(String args[]) throws IOException {
        // Register service on port 1254
        ServerSocket s = new ServerSocket(1254);
        Socket s1 = s.accept(); // Wait and accept a connection
        // Get a communication stream associated with the socket
        OutputStream s1out = s1.getOutputStream();
        DataOutputStream dos = new DataOutputStream(s1out);
        // Send a string!
        dos.writeUTF("Hi there");
        // Close the connection, but not the server socket
        dos.close();
        s1out.close();
        s1.close();
    }
}
Another simple client

// SimpleClient.java: a simple client program
import java.net.*;
import java.io.*;
public class SimpleClient {
    public static void main(String args[]) throws IOException {
        // Open your connection to a server, at port 1254
        Socket s1 = new Socket("mundroo.cs.mu.oz.au",1254);
        // Get an input file handle from the socket and read the input
        InputStream s1In = s1.getInputStream();
        DataInputStream dis = new DataInputStream(s1In);
        String st = new String (dis.readUTF());
        System.out.println(st);
        // When done, just close the connection and exit
        dis.close();
        s1In.close();
        s1.close();
    }
}

Slides prepared by: Farzana Rahman
UDP Sockets

- java.net.DatagramSocket class

- Java makes no distinction between client/server for UDP sockets

- Connected mode UDP supported in Java 2

- Can be bound to both a local port & a local IP address – multi-homed support
UDP Datagrams

- java.net.DatagramPacket class
- Expects a byte array of data
- Address optional for connected-mode UDP
- This class is final – can’t be extended!
- java.net.DatagramSocket instances can only send instances of java.net.DatagramPacket
Threading

- Java doesn’t support the notion of forking processes; how do we support concurrency?
  - Java was designed to support multi-threading!
  - In server environments we can spawn new threads to handle each client
  - Thread groups allow for collective control of many threads
Java Servlets

• Servlets are the Java analog to CGI

• Advantages of servlets: full access to other Java APIs, persistence between invocations, guaranteed portability

• Servlets can be generic services or specific to HTTP
HTTP Servlets

- javax.servlet.http.HttpServlet class
- Uses HTTP to receive requests and generate responses
- Full support for all HTTP methods, cookies, sessions, persistent connections
Java Applets

• Client-side Java programs that run in a browser

• Applets have special security restrictions called the applet sandbox

• Only applets loaded over the network are subject to the applet sandbox

• The applet sandbox is controlled by a java.lang.SecurityManager
Applet Sandbox

- Can’t load libraries or define native methods
- Can’t access local host filesystem
- Can’t open sockets to hosts other than originating host
- Can’t use Runtime.exec()
- Applet windows have a unique appearance
- Restricted access to certain system properties
Remote Method Invocation (RMI)

• RMI is the Java analog to RPC

• RMI servers use a naming service (rmiregistry) to register remote objects

• RMI servers use a special security policy implemented by RMISecurityManager

• The default RMI transport mechanism is via TCP sockets – this is transparent to RMI code!

• Any object transferred in an RMI call must implement the Serializable interface
Java Naming & Directory Interface (JNDI)

- JNDI provides a generic API that can be used to interface with any naming system.
- JNDI uses SPIs (service provider interfaces) to access many different types of naming & directory services from the JNDI API.
- Sun supplies JNDI SPIs for LDAP, NIS, COS (CORBA naming), RMI registry & local filesystem.