Socket Programming in Java
Learning Objectives

• The InetAddress Class
• Using sockets
  – TCP sockets
  – Datagram Sockets
Classes in java.net

- The core package java.net contains a number of classes that allow programmers to carry out network programming
  - ContentHandler
  - DatagramPacket
  - DatagramSocket
  - DatagramSocketImplHttpURLConnection
  - InetAddress
  - MulticastSocket
  - ServerSocket
  - Socket
  - SocketImpl
  - URL
  - URLConnection
  - URLDecoder
  - URLStreamHandler
Exceptions in Java

- BindException
- ConnectException
- MalformedURLException
- NoRouteToHostException
- ProtocolException
- SocketException
- UnknownHostException
- UnknownServiceException
The InetAddress Class

- Handles Internet addresses both as host names and as IP addresses
- Static Method getByName returns the IP address of a specified host name as an InetAddress object
- Methods for address/name conversion:
  ```java
  public static InetAddress getByName(String host) throws UnknownHostException
  public static InetAddress[] getAllByName(String host) throws UnknownHostException
  public static InetAddress getLocalHost() throws UnknownHostException
  public boolean isMulticastAddress()
  public String getHostName()
  public byte[] getAddress()
  public String getHostAddress()
  public int hashCode()
  public boolean equals(Object obj)
  public String toString()
  ```
import java.net.*;
import java.io.*;

public class IPFinder {
    public static void main(String[] args) throws IOException {
        String host;
        BufferedReader input =
            new BufferedReader(
                new InputStreamReader(System.in));
        System.out.print("Enter host name: ");
        host = input.readLine();
        try {
            InetAddress address = InetAddress.getByName(host);
            System.out.println("IP address: "+address.toString());
        } catch (UnknownHostException e) {
            System.out.println("Could not find "+host);
        }
    }
}
import java.net.*;

public class MyLocalIPAddress
{
    public static void main(String[] args)
    {
        try {
            InetAddress address = InetAddress.getLocalHost();
            System.out.println(address);
        } catch (UnknownHostException e) {
            System.out.println("Could not find local address!");
        }
    }
}
The Java.net.Socket Class

• Connection is accomplished through the constructors. Each Socket object is associated with exactly one remote host. To connect to a different host, you must create a new Socket object.

  public Socket(String host, int port) throws UnknownHostException, IOException
  public Socket(InetAddress address, int port) throws IOException
  public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException
  public Socket(InetAddress address, int port, InetAddress localAddress, int localPort) throws IOException

• Sending and receiving data is accomplished with output and input streams. There are methods to get an input stream for a socket and an output stream for the socket.

  public InputStream getInputStream() throws IOException
  public OutputStream getOutputStream() throws IOException

• There's a method to close a socket:

  public void close() throws IOException
The Java.net.SocketSocket Class

- The `java.net.ServerSocket` class represents a server socket. It is constructed on a particular port. Then it calls `accept()` to listen for incoming connections.
  - `accept()` blocks until a connection is detected.
  - Then `accept()` returns a `java.net.Socket` object that is used to perform the actual communication with the client.

```java
public ServerSocket(int port) throws IOException
public ServerSocket(int port, int backlog) throws IOException
public ServerSocket(int port, int backlog, InetAddress bindAddr) throws IOException
```

```java
public Socket accept() throws IOException
public void close() throws IOException
```
TCP Sockets

SERVER:

1. Create a ServerSocket object
   ```java
   ServerSocket servSocket = new ServerSocket(1234);
   ```

2. Put the server into a waiting state
   ```java
   Socket link = servSocket.accept();
   ```

3. Set up input and output streams

4. Send and receive data
   ```java
   out.println-awaiting data...);
   String input = in.readLine();
   ```

5. Close the connection
   ```java
   link.close()
   ```
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`:

  ```java
  BufferedReader in =
  new BufferedReader(
    new InputStreamReader(link.getInputStream()));
  PrintWriter out =
  new PrintWriter(link.getOutputStream(),true);
  ```
TCP Sockets

CLIENT:

1. Establish a connection to the server
   
   \[
   \text{Socket link = new Socket(inetAddress.getLocalHost(),1234) ;}
   \]

2. Set up input and output streams

3. Send and receive data

4. Close the connection
The UDP classes

• 2 classes:
  – java.net.DatagramSocket class
    • is a connection to a port that does the sending and receiving. Unlike TCP sockets, there is no distinction between a UDP socket and a UDP server socket. Also unlike TCP sockets, a DatagramSocket can send to multiple, different addresses. The address to which data goes is stored in the packet, not in the socket.
  
    public DatagramSocket() throws IOException
    public DatagramSocket(int port) throws IOException
    public DatagramSocket(int port, InetAddress laddr) throws IOException
  
  – java.net.DatagramPacket class
    • is a wrapper for an array of bytes from which data will be sent or into which data will be received. It also contains the address and port to which the packet will be sent.

    public DatagramPacket(byte[] data, int length)
    public DatagramPacket(byte[] data, int length, InetAddress host, int port)
Datagram Sockets

SERVER:

1. Create a DatagramSocket object
   
   ```java
   DatagramSocket dgramSocket = new DatagramSocket(1234);
   ```

2. Create a buffer for incoming datagrams
   
   ```java
   byte[] buffer = new byte[256];
   ```

3. Create a `DatagramPacket` object for the incoming datagram
   
   ```java
   DatagramPacket inPacket = new DatagramPacket(buffer, buffer.length);
   ```

4. Accept an incoming datagram
   
   ```java
   dgramSocket.receive(inPacket)
   ```
Datagram Sockets

SERVER:
5. Accept the sender’s address and port from the packet
   InetAddress clientAddress = inPacket.getAddress();
   int clientPort = inPacket.getPort();
6. Retrieve the data from the buffer
   string message =
       new String(inPacket.getData(), 0, inPacket.getLength());
7. Create the response datagram
   DatagramPacket outPacket =
       new DatagramPacket(
           response.getBytes(), response.length(),
           clientAddress, clientPort);
8. Send the response datagram
   dgramSocket.send(outPacket)
9. Close the DatagramSocket: dgram.close();
Datagram Sockets

CLIENT:
1. Create a DatagramSocket object
   
   ```java
   DatagramSocket dgramSocket = new DatagramSocket;
   ```

2. Create the outgoing datagram
   
   ```java
   DatagramPacket outPacket = new DatagramPacket(
       message.getBytes(), message.length(), host, port);
   ```

3. Send the datagram message
   
   ```java
   dgramSocket.send(outPacket)
   ```

4. Create a buffer for incoming datagrams
   
   ```java
   byte[] buffer = new byte[256];
   ```
Datagram Sockets

CLIENT:
5. Create a `DatagramPacket` object for the incoming datagram

   ```java
   DatagramPacket inPacket =
       new DatagramPacket(buffer, buffer.length);
   ```

6. Accept an incoming datagram

   ```java
   dgramSocket.receive(inPacket)
   ```

7. Retrieve the data from the buffer

   ```java
   string response = new String(inPacket.getData(), 0,
       inPacket.getLength());
   ```

8. Close the `DatagramSocket`:

   ```java
   dgram.close();
   ```
References
