JavaOne℠

Sun's Worldwide Java Developer Conference

JavaOne℠

Sun's Worldwide Java Developer Conference

Network-based Applications

Pavani Diwanji
David Brown
JavaSoft

Networking in Java™

- Introduction
- Datagrams, Multicast
- TCP: Socket, ServerSocket
- Issues, Gotchas
- URL, URLConnection
- Protocol Handlers
- Q & A

InetAddress

- InetAddress Object: address, family
- Name service methods
  - getByName
  - getAllByName (multihomed hosts)
- Cache the name, address mapping

Datagrams

- Packet oriented: send and receive
- No connection setup/teardown overhead
- Datagrams can get lost in the internet
- Reliability does not come for free
  - Latency: retransmissions
  - Bandwidth: duplicates, forward error correction
- Example- Video: a frame is lost!

java.net.DatagramPacket

- Represents datagram packet
  - Data buffer, packet length, IP address, port
- Sender creates a DatagramPacket with the data, length, destination IP address and port number

DatagramPacket (cont.)

- On receiving side: application allocates a byte buffer for receiving the packet and passes it to the DatagramPacket constructor
- Receiving buffer can be reused over multiple datagram receives

DatagramSocket

```
Class DatagramSocket {
    public DatagramSocket();
    public DatagramSocket(int port);
    public void send(DatagramPacket p);
    public void synchronized recieve(DatagramPacket p);
    ...
}
```

Simple Datagram Client

```
import java.net.DatagramSocket;
import java.net.DatagramPacket;
class DatagramClient {
        …
        DatagramSocket s = new DatagramSocket();
        DatagramPacket dp = formatPacket(destinationHost,
    port);
        s.send(dp);
        ...
        s.close();
}
```

•

Simple Datagram Server

- import java.net.DatagramSocket;
- import java.net.DatagramPacket;
- class DatagramServer {

```
...
DatagramSocket s = new DatagramSocket();
System.out.println("Server receiving on port: "+
s.getLocalPort())
DatagramPacket dp = new DatagramPacket(buf, length);
s.receive(dp);
...
s.close();
}
```

IP Multicast

- Deliver a packet to a set of destinations
- Destination set is identified by a single group address. Membership can change over time
- More efficient than unicasting seperate copies to all destinations
- Logical/location independent addressing
- RFC’s 966, 1112

IP Multicast (cont.)

- Addressing
  - Range: 224.x.x.x - 239.x.x.x
  - Addresses assigned by hand!
- Scoped Multicasting
  - time-to-live (TTL) field
  - TTL=0: same host
  - TTL=1: same subnet
  - TTL=255: unrestricted

Useful Multicast Examples

- Querying a distributed file store, when the actual location of data is unknown
- Location Service: locate objects on the net
- Sending video over the internet: MBONE
- Real-time networked games

Multicast Class in Java

- sun.net.MulticastSocket
- Extends java.net.DatagramSocket
- New methods:
  - joinGroup(InetAddress mcast);
  - leaveGroup(InetAddress mcast);
  - send(DatagramPacket dp, byte ttl);

Simple Multicast Example

```
import sun.net.MulticastSocket;
class VideoReceiver {
    // listen to the video on the MBONE
    InetAddress receivers = InetAddress.getByName("224.1.2.3");
    MulticastSocket mcast = new MulticastSocket();
    mcast.joinGroup(receivers);
    ….
    mcast.receive(packet); // Receive multicast packet
    ….
    mcast.leaveGroup(receivers);  // clean up when finished
    mcast.close();
}
```

TCP Socket Classes

- Socket: Client Endpoint for doing TCP based communication
- ServerSocket: Server endpoint for doing TCP based communication
- SocketImpl: Site-specific implementation of sockets (e.g. SOCKS, Kona)
- PlainSocketImpl: used by default

Socket, ServerSocket

- Connect to destination by:
  - Constructing a socket given the InetAddress and port.
- read/write done through:
  - Socket.getInputStream(), Socket.getOutputStream()
- ServerSocket.accept():
  - Blocks until a client asks to connect
  - Returns a new client socket handle

Keep in Mind

- Java Applets!= Java Applications
- Applets are restricted in what they can do.
- Security manager in the browser restricts who Applets are allowed to talk to: checkConnect, checkAccept, checkListen...
- No restrictions, though on Network based Applications: write native code to do whatever they want!

Issues…

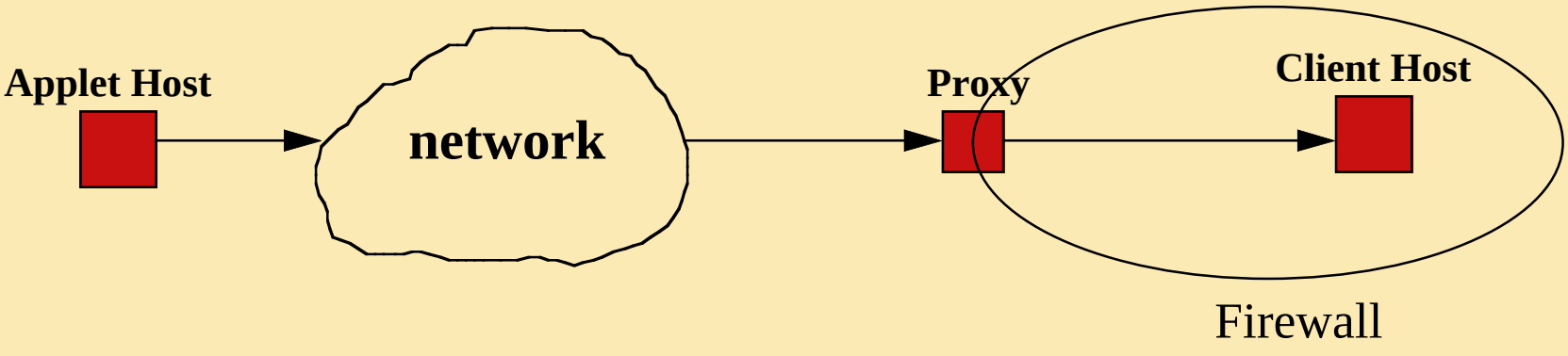
- SecurityManager Checks in datagram and multicast receive:
  - Can getting data hurt?
- Multicast Security
  - Group address-based
  - TTL-based

Issues…

- Name service is not secure!
- Firewalls, Proxies: a necessary evil?
  - Firewalls only let HTTP traffic through.
  - Name resolution within firewall
    - Can resolve an external hostname
    - Cannot resolve an external hostname

Issues…

- Stamp where the applets came from, and never call the name service again!
- Where did the applet come from, if proxy is in the picture?



Gotchas…

- Timers: ms level granularity, relative.
  - Need finer grained (us) timers
  - Absolute timers
- No Poll/Select
- Setting socket options

Gotchas…

- Timeouts on read/write not available
- Use of synchronized buffered streams
- PrintStream eats exceptions! Check explictly for errors

Conclusions

- Sample Code Examples:
  - http://www.javasoft.com/people/pavani
- Questions?

JavaOne℠

Sun's Worldwide Java Developer Conference

Network-based Applications: Extending java.net.URL*

David Brown
JavaSoft

The Classes

- URL
- URLStreamHandler
- URLConnection
- URLStreamHandlerFactory

Why Extend the URL Classes?

- Your own web protocols
- Your own content handling
- Link legacy servers to web
- Simple, small, powerful
- Leverage Java™
- More resources than applets

java.net.URL

- Constructed from String

```
new URL(“http://www.javasoft.com”);
```

- Knows its own URLStreamHandler
- Data members:

```
String protocol; String host;
int port; String file;
URLStreamHandler handler;
```

URLStreamHandler

- Abstract class
- One for each URL protocol
- Create URLConnection given a URL

```
URLConnection openConnection(URL);
```

- Parse a URL in protocol-specific way
- Unparse a URL back to String

```
String toExternalForm(URL u);
```

URLConnection

- Abstract class
- No public constructor
- Instantiable only by StreamHandler
- Can dynamically plug in content handlers

```
protected URLConnection(URL u);
String getContentType();
ContentHandler getContentHandler();
setContentHandlerFactory();
```

URLStreamHandlerFactory

- One way to add protocol handlers
- Interface – one method:

```
URLStreamHandler
createURLStreamHandler(String protocol);
```

- Set the Factory in class URL:

```
URL.setURLStreamHandlerFactory()
```

CGI POST by URLConnection

- HttpURLConnection can POST

```
URL url = new URL("http://java.sun.com/cgi-bin/reverse");
URLConnection conn = url.openConnection();
PrintStream ps = new PrintStream(conn.getOutputStream());
ps.println("string=StringToReverse");
ds = new DataInputStream(conn.getInputStream());
String result = ds.readLine();
/* result should be "esreveRoTgnirtS" */
```

- Applets can connect home from behind firewall

Restrictions/Gotcha's

- Only certain sequences of I/O allowed
- Interpreted as POST:

  get output [write output], get input [read input]

  get output [write output]
- Interpreted as GET:

  get input [read input]

Restrictions/Gotcha's

- Disallowed:

  get input [read input], get output [write output],
- HttpURLConnection has state
- Why? HTTP 1.0 vs 1.1
- No KEEPALIVE/persistent connections

Simple Example: Finger Protocol

- A protocol for fingering users
- URL `finger:brown@monkey,pavani@sai`
- Produces output like:

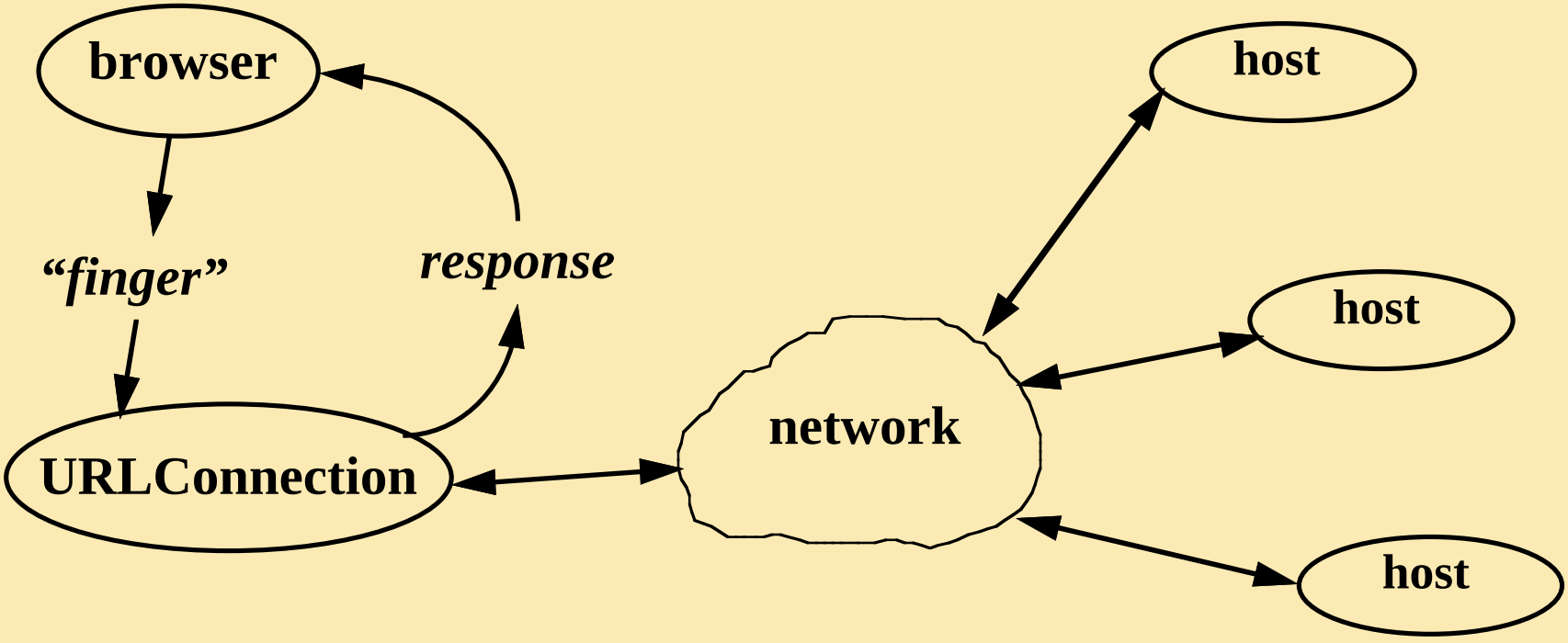
Finger data for “brown@monkey”:

Login	Name	TTY	Idle	When
brown	Dave Brown	console	0:18	Wed 18:53

Finger data for “pavani@sai”:

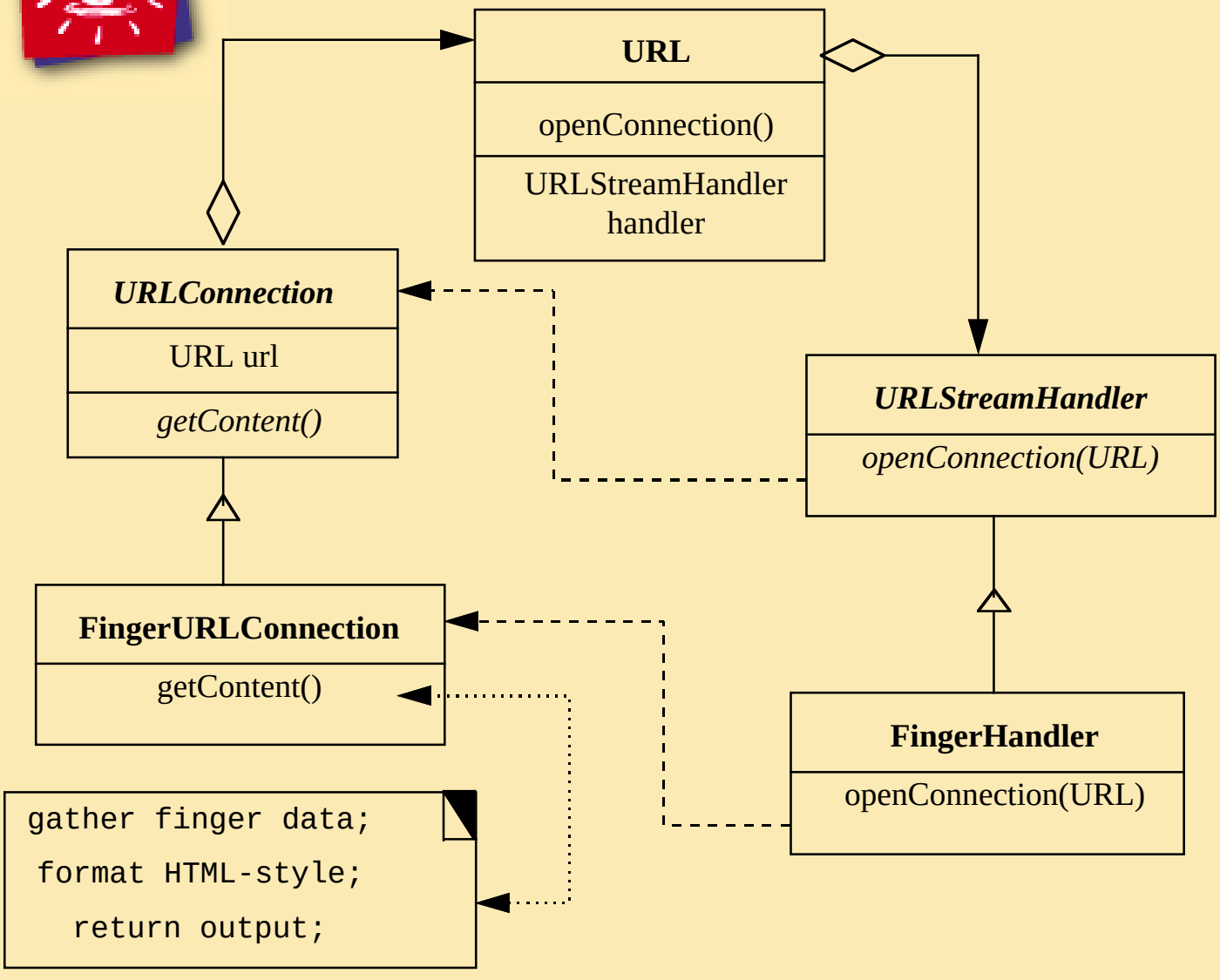
Login	Name	TTY	Idle	When
pavani	Pavani Diwanji	console	0:13	Tue 9:54

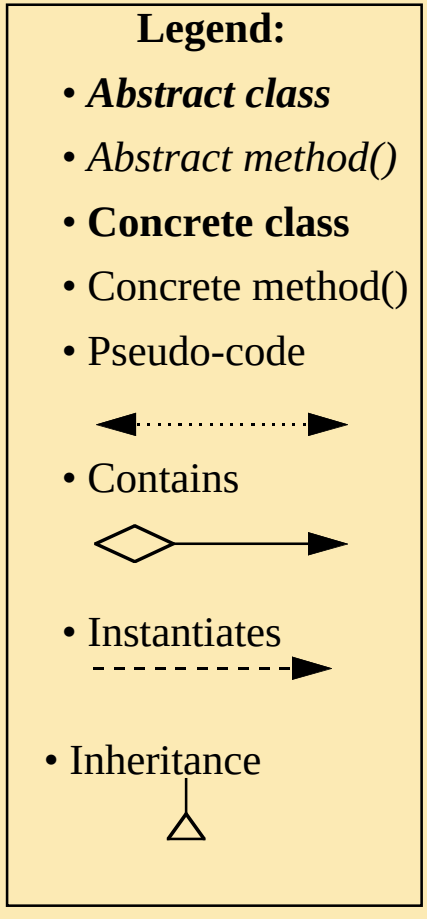
The (web) Finger Protocol



Gather data from multiple connections

Class Relationships





Step 1: Create a New URL

- finger:joe@dinosaur,brown@monkey
- Is there a “finger” protocol handler?
- Ask the handler to parse the URL
- Default does usual HTTP parsing: <protocol>://<host>/<file>

Step 2: Connect the URL

- URL.openConnection():
- Type of URLConnection delegated to URL's handler
- Here's the extensiblity:
  - Data gathering done by an instance of FingerURLConnection
  - Format raw data for browser viewing

What Must URLConnection Do?

- URLConnection defines ~53 methods
- Only override 4:
- FingerURLConnection(URL)
- String getContentType():

```
return “text/html”;
```

What Must URLConnection Do?

- Object getContent():

```
Create a StringBuffer;
for each entry in list {
     Connect to host;
     finger user;
     append finger result to StringBuffer
     (with HTML formatting);
}
return new String(StringBuffer);
```

- InputStream getInputStream():

```
return new
StringBufferInputStream((String)getContent(
));
```

Interesting Protocols

- Newsgroup binary formatter

  listen:alt.binaries.sounds.birds

- Meta search engine

  search://yahoo,lycos,webcrawler/java AND protocol

- Filter protocol - filtered HTML
- Database queries

  jdbc://<query-string>

Sample Code Available

- http://www.javasoft.com/people/brown
- Questions/Flames?
- brown@monkey.eng.sun.com