

# CSCE 515: Computer Network Programming

----- Java Network Programming  
reference: Dave Hollinger

Wenyuan Xu

Department of Computer Science and  
Engineering  
University of South Carolina

## Java Network Programming

- Introduction
- Java Socket Programming
  - InetAddress
  - Socket
  - ServerSocket
  - DatagramSocket
- Multicast Socket

2007

CSCE515 – Computer Network Programming

## Crash Course in Java

## Why Java?

- Network Programming in Java is very different than in C/C++
  - much more language support
  - error handling
  - no pointers! (garbage collection)
  - Threads are part of the language.
  - some support for common application level protocols (HTTP).

Netprog: Java Intro

4

## Java notes for C++ programmers

- Everything is an object.
- No code outside of class definition!
- Single inheritance
  - an additional kind of inheritance: interfaces
- All classes are defined in .java files
  - one top level public class per file
- To print to stdout:
  - System.out.println();

Netprog: Java Intro

5

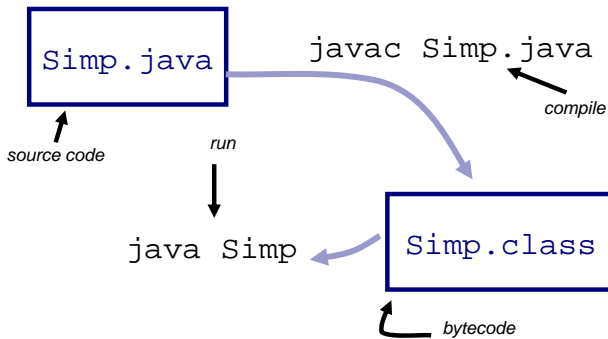
## First Program: `simp.java`

```
public class simp {  
  
    public static void main(String args[]) {  
        System.out.println("Hello, Netprog");  
    }  
  
}
```

Netprog: Java Intro

6

## Compiling and Running



## Java bytecode and interpreter

- bytecode is an intermediate representation of the program (class).
- The Java interpreter starts up a new “Virtual Machine”.
- The VM starts executing the users class by running it’s `main()` method.

## Java Data Types

### ■ Primitive Data Types:

- `boolean` `true` or `false`
- `char` unicode! (16 bits)
- `byte` signed 8 bit integer
- `short` signed 16 bit integer
- `int` signed 32 bit integer
- `long` signed 64 bit integer
- `float, double` IEEE 754 floating point

*not an int!*

## Other Data Types

### ■ Reference types (composite)

- classes
- arrays
- strings are supported by a built-in class named `String`
  - Not an array of chars!**
- string literals are supported by the language (as a special case).

## Classes and Objects

- “All Java statements appear within methods, and all methods are defined within classes”.
- Java classes are very similar to C++ classes (same concepts).
- Instead of a “standard library”, Java provides a lot of Class implementations.

## Defining a Class

- One top level public class per .java file.
  - typically end up with many .java files for a single program.
  - One (at least) has a **static public main()** method.
- **Class name must match the file name!**
  - compiler/interpreter use class names to figure out what file name is.

## Sample Class

(from Java in a Nutshell)

```
public class Point {
    public double x,y;
    public Point(double x, double y) {
        this.x = x; this.y=y;
    }
    public double distanceFromOrigin(){
        return Math.sqrt(x*x+y*y);
    }
}
```

Netprog: Java Intro

13

## Objects and new

You can declare a variable that can hold an object:

```
Point p;
```

but this doesn't create the object! You have to use new:

```
Point p = new Point(3.1,2.4);
```

there are other ways to create objects...

Netprog: Java Intro

14

## Using objects

### ■ Just like C++:

- `object.method()`
- `object.field`

### ■ BUT, never like this (no pointers!)

- `object->method()`
- `object->field`

Netprog: Java Intro

15

## Reference Types

- Objects and Arrays are *reference types*
- Primitive types are stored as values.
- Reference type variables are stored as references (pointers that we can't mess with).
- There are significant differences!

Netprog: Java Intro

16

## Primitive vs. Reference Types

```
int x=3;
```

```
int y=x;
```

*There are two copies of the value  
3 in memory*

```
Point p = new Point(2.3,4.2);
```

```
Point t = p;
```

*There is only one Point object in  
memory!*

Netprog: Java Intro

17

## Passing arguments to methods

- Primitive types: the method gets a copy of the value. Changes won't show up in the caller.
- Reference types: the method gets a copy of the reference, the method accesses the same object!

Netprog: Java Intro

18

## Packages

- You can organize a bunch of classes into a *package*.
  - defines a namespace that contains all the classes.
- You need to use some java packages in your programs
  - java.lang java.io, java.util

## Importing classes and packages

- Instead of `#include`, you use `import`
- You don't have to import anything, but then you need to know the complete name (not just the class, the package).
  - if you `import java.io.File` you can use `File` objects.
  - If not – you need to use `java.io.File` objects.

## Exceptions

- Terminology:
  - *throw an exception*: signal that some condition (possibly an error) has occurred.
  - *catch an exception*: deal with the error (or whatever).
- In Java, exception handling is necessary (forced by the compiler)!

## Try/Catch/Finally

```
try {  
    // some code that can throw  
    // an exception  
} catch (ExceptionType1 e1) {  
    // code to handle the exception  
} catch (ExceptionType2 e2) {  
    // code to handle the exception  
} finally {  
    // code to run after the stuff in try  
    // can handle other exception types  
}
```

## Exceptions and Network Programming

- Exceptions take care of handling errors
  - instead of returning an error, some method calls will throw an exception.
- A little hard to get used to, but forces the programmer to be aware of what errors can occur and to deal with them.

## Socket Programming

## Java Sockets Programming

- The package `java.net` provides support for sockets programming (and more).
- Typically you import everything defined in this package with:

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

## Classes

`InetAddress`  
`Socket`  
`ServerSocket`  
`DatagramSocket`  
`DatagramPacket`

## `java.net.InetAddress` class

- static methods you can use to create new `InetAddress` objects.

```
public static InetAddress getByName(String host)
public static InetAddress getLocalHost()
public static InetAddress[] getAllByName(String
    hostName)
```

```
InetAddress x = InetAddress.getByName(
    "cse.sc.edu");
```

## Sample Code: `Lookup.java`

- Uses `InetAddress` class to lookup hostnames found on command line.

```
> java Lookup www.yahoo.com
```

```
www.yahoo.com:209.191.93.52
```

## Sample Code: `getLocalhost.java`

- Uses `InetAddress` class to lookup localhost

```
> java getLocalhost
```

```
broad/129.252.130.139
```

```
try {
    InetAddress a = InetAddress.getByName(hostname);
    System.out.println(hostname + ":" +
        a.getHostAddress());
} catch (UnknownHostException e) {
    System.out.println("No address found for " +
        hostname);
}
```

## getLocalhost()

```
try {
    InetAddress address =
    InetAddress.getLocalHost();

    System.out.println(address);
} catch (UnknownHostException e) {
    System.out.println("Could not find this
computer's address."); }
}
```

## getAllByName()

```
try {
    InetAddress[] addresses =
    InetAddress.getAllByName("www.microsoft.com");
    for (int i = 0; i < addresses.length; i++)
    { System.out.println(addresses[i]); }
} catch (UnknownHostException e) {
    System.out.println("Could not find
www.microsoft.com");
}
```

## How to open a socket?

- **Socket** class: corresponds to active TCP sockets only!
  - client sockets
  - socket returned by accept();
- Passive sockets are supported by a different class: **ServerSocket**
- UDP sockets are supported by **DatagramSocket**

## Client Socket Constructors

- Constructor creates a **TCP connection** to a named TCP server.

- There are a number of constructors:

```
Socket(InetAddress server, int port);
```

```
Socket(InetAddress server, int port,
    InetAddress local, int localport);
```

```
Socket(String hostname, int port);
```

## Client Socket Constructors

```
Socket(String hostname, int port);
```

```
Socket MyClient;

try {
    MyClient = new Socket("www.yahoo.com", 80);
} catch (UnKnowHostException e) {

    System.out.println(e);
} catch (IOException e) {

    System.out.println(e); }
```

## Client Socket Constructors

```
Socket(InetAddress server, int port);
```

```
InetAddress myServerAddr;
Socket MyClient;

try {
    myServerAddr =
    InetAddress.getByName("www.yahoo.com");
    MyClient = new Socket(myServerAddr, 80);
} catch (UnKnowHostException e) {
    System.out.println(e);
} catch (IOException e) {
    System.out.println(e);
}
```

## Socket Methods

```
void close();
InetAddress getInetAddress(); getpeername
InetAddress getLocalAddress(); getsockname
InputStream getInputStream();
OutputStream getOutputStream();
```

- Lots more (setting/getting socket options, partial close, etc.)

## Socket I/O

- Socket I/O is based on the Java I/O support (in the package `java.io`).
- `InputStream` and `OutputStream` are abstract classes
  - common operations defined for all kinds of `InputStreams`, `OutputStreams`...

## InputStream Basics

```
// reads some number of bytes and
// puts in buffer array b
int read(byte[] b);

// reads up to len bytes
int read(byte[] b, int off, int len);
```

Both methods can throw `IOException`.  
Both return `-1` on `EOF`.

## OutputStream Basics

```
// writes b.length bytes
void write(byte[] b);

// writes len bytes starting
// at offset off
void write(byte[] b, int off, int
len);
```

Both methods can throw `IOException`.

## Output stream example

```
InetAddress myServerAddr;
Socket MyClient;

DataOutputStream output;

try {
    ...
    output = new
    DataOutputStream(MyClient.getOutputStream());
    output.writeBytes("hello");
    output.writeBytes("DATA\n");
} catch (IOException e) {
    System.out.println(e);
}
```

## Sample Client

- `smtpClient.java`
- Simple client code to send an email via an smtp server
- Client:
  - Open a socket.
  - Open an input and output stream to the socket.
  - Read from and write to the socket according to the server's protocol.
  - Clean up.

## ServerSocket Class –Servers (TCP Passive Socket)

### ■ Constructors:

```
ServerSocket(int port);
```

```
ServerSocket(int port, int backlog);
```

```
ServerSocket(int port, int backlog,  
             InetAddress bindAddr);
```

## ServerSocket Constructors

```
ServerSocket(int port, int backlog);
```

```
ServerSocket MyService;  
  
try {  
    MyService = new ServerSocket(PortNumber, 90);  
} catch (IOException e) {  
    System.out.println(e);  
}
```

## ServerSocket Methods

```
Socket accept();
```

```
void close();
```

```
InetAddress getInetAddress();
```

```
int getLocalPort();
```

```
throw IOException, SecurityException
```

## Sample Echo Server

### ■ TCPEchoServer.java

### ■ Server:

1. Create a `ServerSocket`
2. `accept` incoming request, get a `Socket`
3. Open an input and output stream to the `Socket`.
4. Read from and write to the `Socket` according to the server's protocol.
5. Close the `Socket`
6. Return to step 2

## UDP Sockets

### ■ `DatagramSocket` class

- `DatagramPacket` class needed to specify the payload (incoming or outgoing).

## DatagramSocket Constructors

```
DatagramSocket();
```

```
DatagramSocket(int port);
```

```
DatagramSocket(int port, InetAddress a);
```

All can throw `SocketException` or `SecurityException`.

*When should each constructors be used?*

## Client UDP Socket constructor

```
try {
    DatagramSocket client = new DatagramSocket();
    //send packets...
} catch (SocketException e) {
    System.out.println(e);
}
```

2007

CSCE515 – Computer Network Programming

## UDP port scanner

```
for (int port = 1024; port<=65535; port++){
    try {
        DatagramSocket server = new DatagramSocket(port);
        server.close();
    } catch (SocketException e) {
        System.out.println("there is a server on port"+port+
            ".");
    }
}
```

2007

CSCE515 – Computer Network Programming

## DatagramSocket Methods

```
void receive(DatagramPacket p);

void send(DatagramPacket p);

void connect (int port, InetAddress);

void close();
```

*Lots more!*

```
ssize_t sendto( int sockfd,
                const void *buff,
                size_t nbytes,
                int flags,
                const struct sockaddr* to,
                socklen_t addrlen);
```

Netprog: Java Sockets

51

## DatagramPacket

- Contain the payload (a byte array).
- Can also be used to specify the destination address (when not using connected mode UDP).

Netprog: Java Sockets

52

## DatagramPacket Constructors

For receiving:

```
DatagramPacket( byte[] buf, int len);
```

For sending:

```
DatagramPacket( byte[] buf, int len
                InetAddress a, int port);
```

Netprog: Java Sockets

53

## DatagramPacket methods

```
byte[] getData();
void setData(byte[] buf);
void setAddress(InetAddress a);
void setPort(int port);

InetAddress getAddress();
int getPort();
```

*destination address*

*could be either address (depends on context)*

Netprog: Java Sockets

54

## Sample UDP code

UDPEchoServer.java

Simple UDP Echo server.

Test using nc as the client (netcat):

```
> nc -u hostname port
```

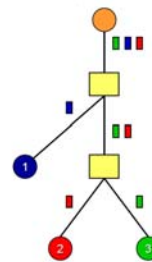
## Multicast

## Multicasting

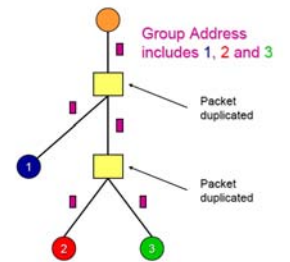
- Unicast
  - A flow from one source to one destination
  - IP packets contain destination IP address
- Broadcast
  - A flow from one source to all destinations
  - IP packets contain broadcast address 255.255.255.255
- Multicast
  - A flow from one source transmits to a *Group of destinations*
  - IP packets contain a class D address for destination
  - Ranges from 224.0.0.0 to 239.255.255.255 (256K addresses)

## Multicast vs. Multiple Unicast

Multiple Unicast:



Multicast:



## Advantages of Multicasting

- Advantages:
  - Lower overhead at the source
    - Source sends only one packet
  - Bandwidth is conserved on shared links
    - Only one copy of each packet is sent on each link
- Requirements:
  - Group address management
    - – Network/router participation
  - Packet duplication at routing nodes
- Disadvantages
  - Security
  - Business models
  - No Incentives for deployment

## Multicast Addressing

Class D 

1	1	1	0
---	---	---	---

28  
multicast group ID

224.0.0.0 -- 239.255.255.255

The set of hosts listening to a particular multicast address is called a *host group*

Multicast packets, at least for now, are sent only as UDP packets (*WHY?*)

Some of the addresses 224.0.0.0 to 224.0.0.15 are reserved for well-known groups  
239.0.0.0 to 239.255.255.255 are for local/administratively scoped applications

Like any IP address, a multicast address can have a hostname,  
e.g. 224.0.0.2 -- all-routers.mcast.net (All routers on the local subnet)

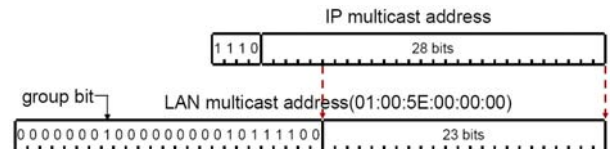
## Mapping to Ethernet Addresses

- Ethernet has a 48-bit address field
  - It has its own multicast address range
  - 01.00.5e.00.00.00 through 01.00.5e.7f.ff.ff
- Lower order 23 bits can be used for multicast addresses
  - IP multicast address has 28 bits for specifying a group address
  - Thus, only the lower order 23 bits of IP multicast address are copied into the Ethernet address

CSCE515 – Computer Network Programming

## Link-Layer Multicast Addresses

- Ranges from 01:00:5E:00:00:00 to 01:00:5E:7F:FF:FF
  - Map low-order 23 bits of class D address to lower order 23 bits of ethernet multicast address space
  - Upper 5 bits of multicast group ID are ignored in the mapping, thus mapping is not unique
- For point-to-point links: no mapping needed.



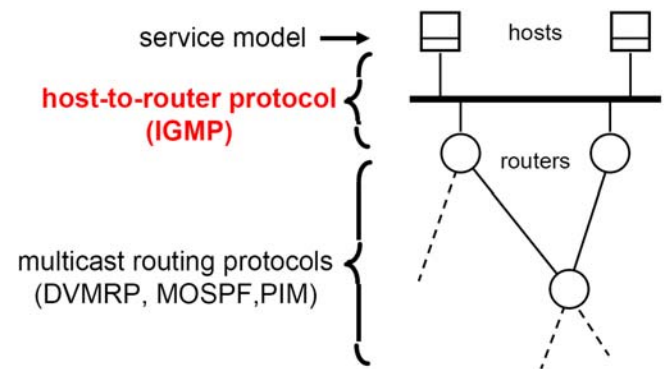
CSCE515 – Computer Network Programming

## Multicast service model

- Uses the notion of host groups
  - Sender sends to a group address
  - Any receiver who has joined this group gets this packets
- Implications?
  - No limits on number or location of receivers
  - Best effort delivery (same as in unicast)
- Dynamic membership
  - Host can join/leave at will (no synchronization required among group members)
- Scope control
  - Can limit the distribution using TTL

CSCE515 – Computer Network Programming

## Components of the IP Multicast Architecture



CSCE515 – Computer Network Programming

## Multicast Routers

- Biggest restriction on multicasting:
  - Availability of special multicast routers
- Check whether your routers support multicasting:
 

```
wyxc@broad % ping all-routers.mcast.net
all-routers.mcast.net is alive
```

CSCE515 – Computer Network Programming

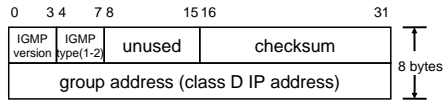
## IGMP

- Internet Group Management Protocol (IGMP)
- Allows a router to know which of its directly connected hosts belongs to which multicast group
- IGMP is required to support TRPB, RPM, CBT and PIM protocols

CSCE515 – Computer Network Programming

## IGMPv1 Message Format

- IGMP is part of the IP layer
- IGMP messages are transmitted in IP packets



67

## IGMP Host Queries

- Routers use IGMP “query” messages to periodically query hosts on their subnets and learn if they are members of *any multicast group*
  - Queries are addressed to all hosts group (224.0.0.1) and carry an IP TTL of 1 (no more than once a minute)
  - Hosts who are members of multicast groups respond with one IGMP “report” message for each group they are a member of
  - To improve efficiency, hosts wait a random amount of time before responding
    - – During this waiting time, hosts listen to other host responses
    - – If another host reports membership in the same group, then the first host aborts its report

CSC515 – Computer Network Programming

## IGMP Host Queries

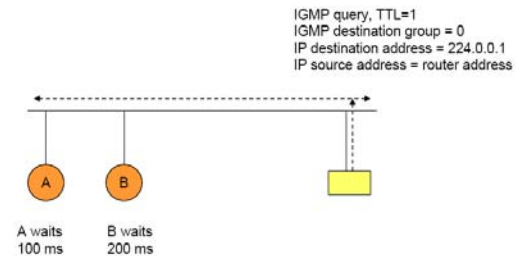
Example: A and B are members of multicast group  $G_1$



CSC515 – Computer Network Programming

## IGMP Host Queries

Example: A and B are members of multicast group  $G_1$

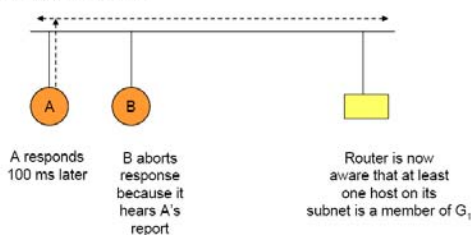


CSC515 – Computer Network Programming

## IGMP Host Queries

Example: A and B are members of multicast group  $G_1$

IGMP report, TTL=1  
 IGMP destination group =  $G_1$   
 IP destination address =  $G_1$   
 IP source address = A



CSC515 – Computer Network Programming

## Communicating with a Multicast Group

- Typical four key operations
  - Join a multicast group
  - Send data to the members of the group
  - Receive data from the group
  - Leave the multicast group
- To receive data from a multicast group, you have to join the group
- To send data, you don't have to join the group

CSC515 – Computer Network Programming

## Class MulticastSocket

- Extend class DatagramSocket and add support for IP multicast
- Multiple MulticastSockets can listen to same port on same machine

### Constructors

```
MulticastSocket()  
MulticastSocket(int port)  
MulticastSocket(SocketAddress  
bindAddress)
```

## Class MulticastSocket

```
try {  
  
    MulticastSocket ms = new MulticastSocket();  
    //send some datagrams  
  
} catch (SocketException e) {  
  
    System.out.println(e);  
  
}
```

## Class MulticastSocket

### Methods

```
void joinGroup(InetAddress group) throws IOException  
void leaveGroup(InetAddress group) throws IOException  
void setTimeToLive(int ttl) throws IOException  
void setTTL(byte ttl) throws IOException  
int getTimeToLive() throws IOException  
byte getTTL() throws IOException  
void send(DatagramPacket packet, byte ttl) throws IOException  
void setInterface(InetAddress address) throws SocketException  
InetAddress getInterface() throws SocketException  
void receive(DatagramPacket p)
```

### Exceptions

```
IOException  
SecurityException
```

75

## Sending Multicast Packets

```
// byte[] data  
// InetAddress multicastGroup  
// int multicastPort  
MulticastSocket socket = new MulticastSocket();
```



```
DatagramPacket packet = new DatagramPacket  
(data, data.length, multicastGroup,  
multicastPort);
```



```
socket.send(packet, (byte) 64);
```



```
socket.close();
```

76

## Receiving Multicast Packets

```
MulticastSocket socket = new  
MulticastSocket(multicastPort);  
Socket.joinGroup(multicastGroup);
```



```
byte buffer[] = new byte[65508];  
DatagramPacket packet = new DatagramPacket();
```



```
socket.receive(packet);
```



```
InetAddress fromAddress = packet.getAddress();  
int fromPort = packet.getPort();  
int length = packet.getLength();  
byte[] data = packet.getData();  
// ...
```



```
socket.leaveGroup(multicastGroup);  
socket.close();
```

77

## Sample MulticastSocket code

MulticastSender.java

MulticastSniffer.java

Receiver:

```
Broad % java MulticastSniffer all-  
systems.mcast.net 4000
```

Sender:

```
Broad % java MulticastSender all-  
systems.mcast.net 4000
```

Netprog: Java Sockets

78



## A Peer-to-Peer Multicast Chat System

- Each client multicasts its message to other clients
- No server is involved; all clients communicate as peers
- Open a chat frame and start a thread that listens for incoming packets

79



## Threads

- Subclassing thread

```
□ public class MyThread extends Thread{}  
□ public void run(){..  
□ public static void main(String[] args) {  
    ■ new Mythread.start();}
```