Java Programming, Comprehensive

Lecture 10

Bineet Sharma

Agenda: Networking with Java

- TCP/IP
- HTTP
- Socket
- UDP
- What's New In JDK 8: Overview
- Default Methods in Interface
- Lamda Expression

- Example codes are inspired either from "Java The Complete Reference" by Herbert Schildt, Murach's or Oracle tutorial http://docs.oracle.com/javase/tutorial/

Event Driven Programming

Objectives

Applied

- Use URL class. Parse URLs
- Use URLConnection, HttpURLConnect clases to read and write into Internet resources
- Use Sockets, and SocketServer classes to create low level client/server applications
- Use InetAddress class to encapsulated Internet address
- Use DatagramSocket

Event Driven Programming

Objectives (cont.)

Knowledge

- Describe how HTTP works
- Describe how sockets work
- Differentiate between HTTP, UDP, TCP/IP protocols
- Implement whois functionality
- Implement a client server knock, knock joke using client server programming

Networking with Java

- Motivation of Networking with Java. It:
  - Supports distributed model of computation
  - Supports client(s) server model of computation
  - Allows programs to download documents (files, images)
  - Allows request-response model as in Web browsers and Web servers
    - When you enter a link in Web browser (a client application), essentially a request is sent to the appropriate Web server (a server application) – servers responds by sending the appropriate HTML page
  - Allows programs to run as Applets in client machine
- What will we cover?
  - Learn how to use Java to write a client-server application
  - We will cover both sides of client-server model

Demonstrate

A

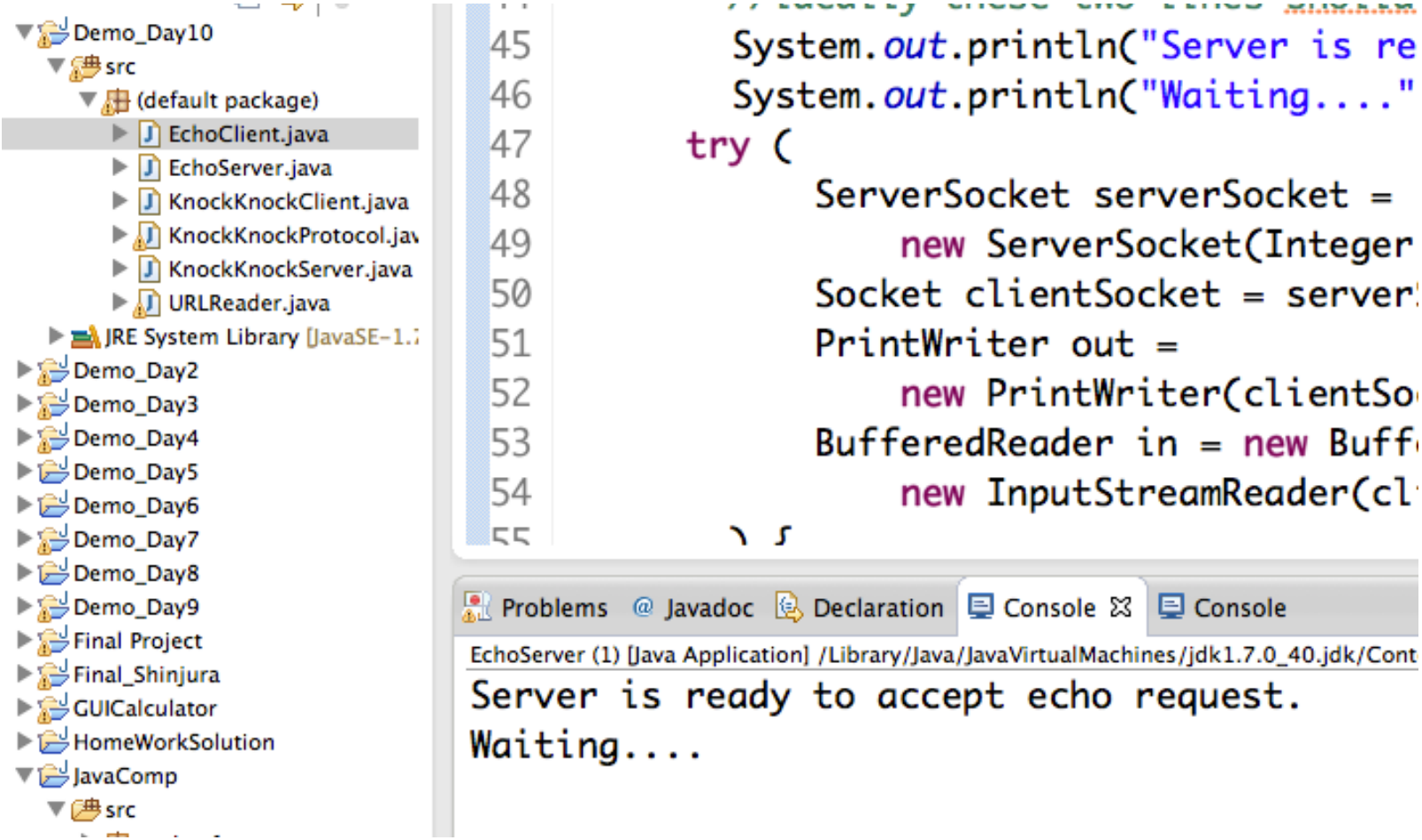
Basic

Client/Server Networking

Application

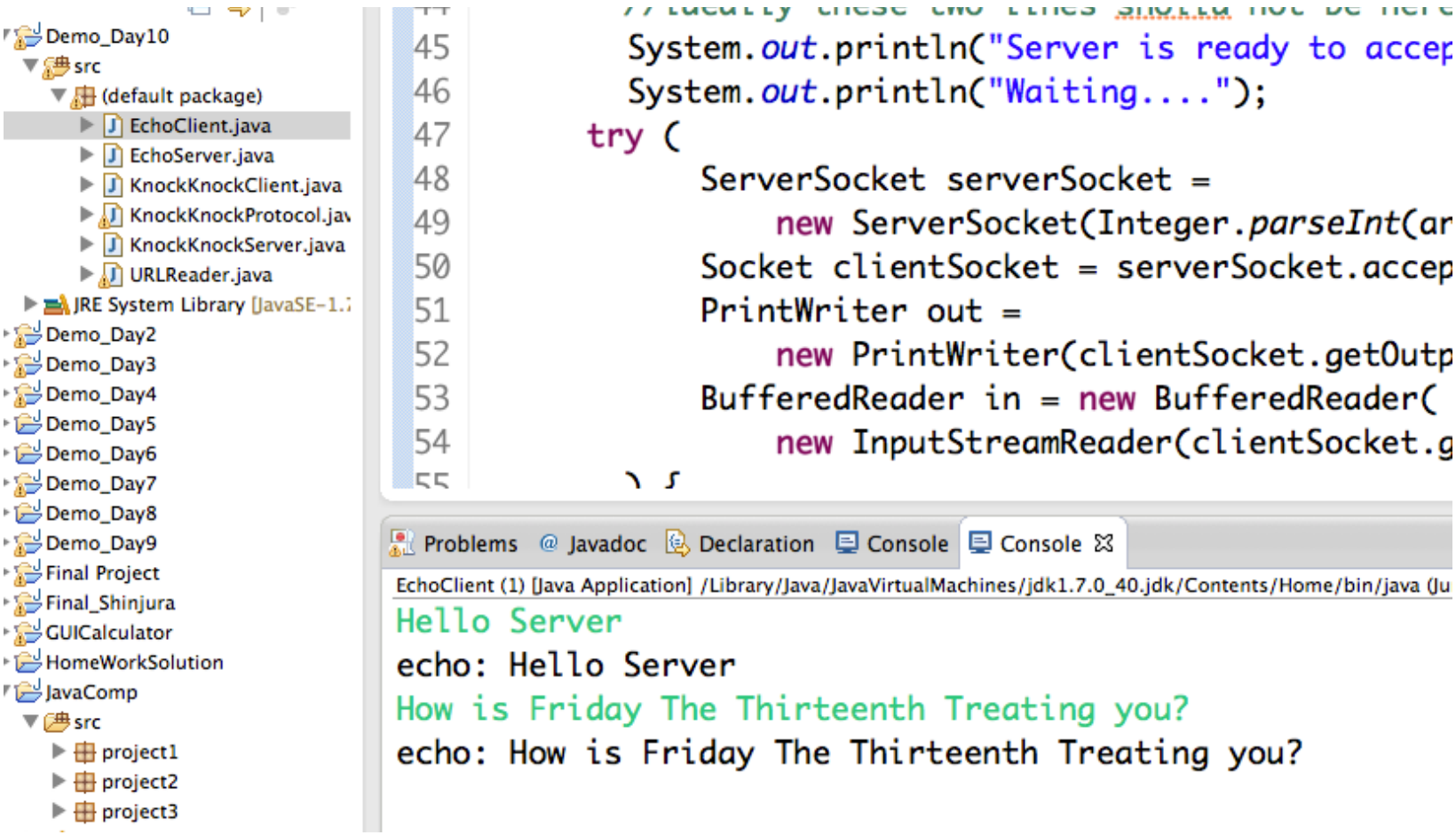
Networking with Java

- Run EchoServer first



Networking with Java

- Run EchoClient next.



Networking with Java

- Now client and server can communicate with each other
  - Client type something

```
Problems  @ Javadoc  Declaration  Console  Console
EchoClient (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_40.jdk/Contents/Home/bin/java (Jun 13, 2014, 12:26:56 PM)
Today is Friday the thirteenth, 06/13/2014
echo: Today is Friday the thirteenth, 06/13/2014
How is Friday The Thirteenth Treating you?
echo: How is Friday The Thirteenth Treating you?
Did you know that it is Full moon as well and is called 'Honey' Moon
echo: Did you know that it is Full moon as well and is called 'Honey' Moon
Lots of headlines on "Spooky 'Honey' Moon Casts Glow on Friday the 13th"
echo: Lots of headlines on "Spooky 'Honey' Moon Casts Glow on Friday the 13th"
Happens only 14.3 on average, next will be in 2049
echo: Happens only 14.3 on average, next will be in 2049
```

  - Server can respond

```
Problems  @ Javadoc  Declaration  Console  Console
EchoServer (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_40.jdk/Contents/Home/bin/java (Jun 13, 2014, 12:26:41 PM)
Received a echo request of: Today is Friday the thirteenth, 06/13/2014
Received a echo request of: How is Friday The Thirteenth Treating you?
Received a echo request of: Did you know that it is Full moon as well and is called 'Honey' Moo
Received a echo request of: Lots of headlines on "Spooky 'Honey' Moon Casts Glow on Friday the
Received a echo request of: Happens only 14.3 on average, next will be in 2049
```

Networking with Java

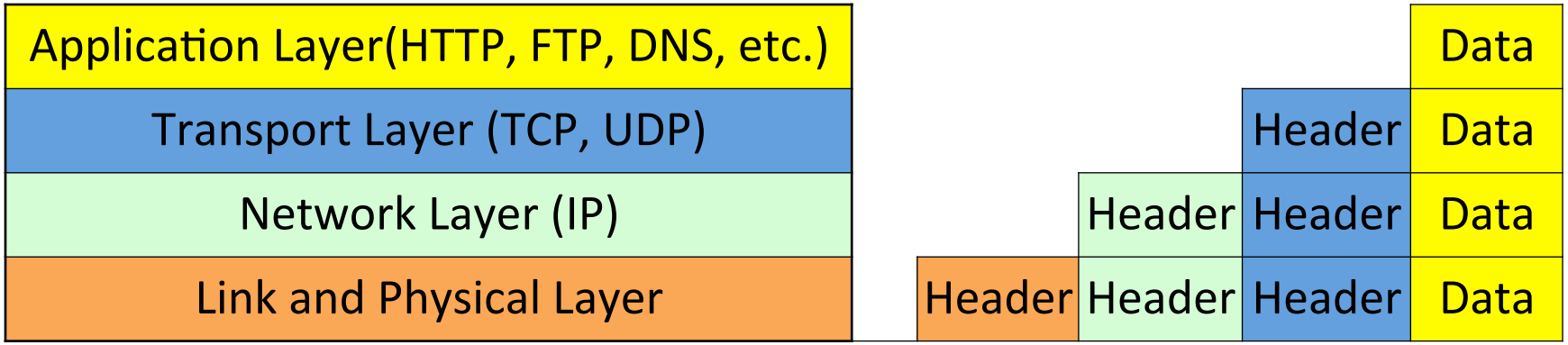
- Networking basics: Sockets and Ports
  - At the core of networking there is a concept of ***sockets***.
    - A software socket identifies an endpoint in a network (of computers)
    - Sockets allow a single computer to serve many computers at once
    - This is accomplished by the use of ***port***. Port is really a numbered socket in a computer
    - A server computer, *listens* to a port until a client computer latches onto it
    - A server computer, can connect to multiple client computer using same port
    - Each client machine will be served differently using unique session
    - A server machine uses multi-threading to accomplish this

Networking with Java

- Networking basics – Protocols - Network communication happens via protocols (set of rules)
  - Internet Protocol (IP): is a low-level routing protocol that breaks data into small packets and sends them to an address across network – delivery is not guaranteed
  - Transmission Control Protocol (TCP): is a high-level protocol that manages to robustly string together these (IP) packets, sorting and re-transmitting them as necessary to reliably transmit data
  - User Datagram Protocol (UDP): is another protocol which supports fast, connectionless, and unreliable transport of packets over IP
  - HTTP, FTP etc.: higher-level protocols used by web applications

Networking with Java

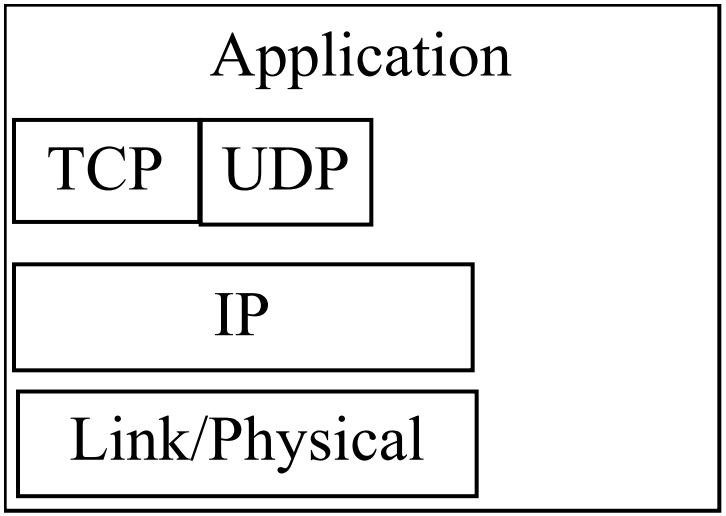
- Networking basics:
  - Computers running on the Internet communicate with each other using various networking protocols:



Networking with Java

Java Networking Model:

- Java networking model is flexible
  - Java programs communicate with each other at the application layer, however, you can choose to communicate at any layer as well
  - You typically use classes in the **java.net** package
  - The classes provide system-independent network communication



Networking with Java

Java Networking Model:

- The classes in the java.net packages are designed to support, varieties of application models:
  - Client-Server
  - Multicasting
  - Web-based
- Socket classes support client-server model, Sockets are considered low level. You will use these classes for:
  - Input/Output model using host address and ports
  - Protocols like TCP or UDP
- Universal Resource Locator (URL) classes support Web-based model
- Basically, Java classes support both stream-based communications, and packet-based communications

Networking with Java

Classes in java.net package

- There are two types of classes which helps programming in different layers of Java networking model:
  - High-level APIs support in application layer use in FTP, HTTP protocols
  - Low-level APIs, supporting:
    - Socket-based and packet-based communications
- Socket-based communications, supports client-server model:
  - Allows applications to view the communications as streams of data (like a disk file)
  - Is a connection-based protocol and uses TCP
- Packet-based communications, supports broadcast model:
  - Allows applications to communicate using packets of data (chunks of data)
  - It is a connectionless protocol and uses UDP

Networking with Java

Transmission-Control Protocol (TCP)

- TCP is a connection-based protocol that provides a reliable flow of data between two applications (endpoints)
- Tasks performed by TCP:
  - Handshake (connection established)
  - Data divided into packets. Use IP to transfer them
  - Assemble and de-assemble. Error correction, retransmit if needed. Uses checksum.
  - Connection terminated

User Datagram Protocol (UDP)

- UDP is a connectionless protocol and provides an unreliable flow of data between two endpoints (why even consider something which is considered unreliable?)
- Used when speed is essential: streaming media, games, voip

Networking with Java

Identification of endpoints (applications): Internet Address

- Key to establish connections between computers is to identify these computers by an unique name
- Yes, each computer in the internet has unique name, called *Internet address*
- The IPV4 (Internet Protocol, version 4) specifies this address by 32-bit values, organized in 4 8-bit values
  - 192.168.1.1, 129.0.0.1 etc.
- The IPV6 specifies this address by 128-bit values, organized in 8 16-bit values supporting much wider ranges of IP addresses
- When working with Java these details, largely, don't matter to you

Networking with Java

Identification of endpoints (applications): Internet Address (Cont.)

- The dotted quad (or dotted decimal) format describes the network hierarchies (192.198.1.1 etc.)
- IP address is not easily remembered. So, a table of names is used to represent IP numbers
- These names are easy to remember and are in a hierarchical order (though they are no one-to-one mapping between name hierarchies and dotted quad number hierarchies)
- Domain name describes a machine's location in a name space
- www.oracle.com is in the COM top-level domain (reserved for US commercial website). The name of end point is 'oracle' and 'www' identifies the server for web requests
- Domain Name Service (DNS) maps the IP to internet name

Networking with Java

Identification of endpoints (applications): Ports

- Most computers have single internet connection (hence single IP address) used for all data
- However, a server machine can run multiple network applications at once
- There is a problem of directing the traffic to the right application – how does the computer know which application gets this data (packet)?
- Solution is to provide a unique number to each channel endpoint known to both the communicating computers
- This unique number is called *port*
- A port number is a 16 bit number
- While creating a connection between two end points (application), we need an IP address and a port number

Networking with Java

Reserved Ports:

- TCP/IP reserves the lower 1, 024 ports for specific protocols
- Some well known ports are:
  - 20, 21: FTP (File Transfer Protocol)
  - 22: SSH (Secure Shell)
  - 23: Telnet
  - 25: SMTP (Simple Mail Transport Protocol)
  - 80: HTTP (HyperText Transport Protocol)
  - 110: POP3 (Post Office Protocol 3)
  - 119: NNTP (Network News Transport Protocol)
- Good idea or bad idea to have these known ports?

Networking with Java

Important Java Networking classes defined in java.net package

- InetAddress: Encapsulates both the numerical IP address and the domain name for that address (handles IPv4, IPv6)
- Socket: Encapsulates a connected client socket which would like to connect to a server socket at specified port
- SocketServer: Encapsulates a connected server socket which would accepts a client socket connections in a specified port
- SocketImpl: Socket and SocketServer classes use this class for all the services
- DatagramSocket: Encapsulates supports for sending and receiving Datagrams (used for User Datagram Protocol:UDP)
- DatagramPacket: Encapsulates a self-describing packet
- URL: Encapsulates Unified Resource Locator, points to a 'resource' on the World Wide Web

Networking with Java

Working with I. Addresses: InetAddress class

- Encapsulates both the address (IP add and domain name)
- It has no visible constructor so use factory methods (really a static method) to instantiate the object of InetAddress
- There are three common factory methods:

static InetAddress getLocalHost() – returns InetAddress object representing this local server

static InetAddress getByName(String hostName) - returns InetAddress object for a host name passed to it.

static InetAddress [] getAllByName(String hostName) – returns an array of InetAddress object that name resolves to.

Some hosts name resolved to multiple IP addresses

If these methods fails they throw UnknownHostException

Networking with Java

Using: InetAddress class

```
public static void testInetAddress(
                        String strDomainName)
                        throws UnknownHostException
{
  InetAddress iAddress = InetAddress.getLocalHost();
  System.out.println(iAddress);

  iAddress = InetAddress.getByName(strDomainName);
  System.out.println(iAddress);

  InetAddress []iArrayAddress =
            InetAddress.getAllByName(strDomainName);
  for (InetAddress ia: iArrayAddress)
        System.out.println(ia);
}
```

Networking with Java

Using: InetAddress class

```
Call the method:

testInetAddress("www.apple.com");

The output will look like this: (may vary for you)

bineetsarmasmbp/192.168.1.85
www.apple.com/23.72.205.15
www.apple.com/23.72.205.15
www.apple.com/2600:1406:22:190:0:0:0:c77
www.apple.com/2600:1406:22:192:0:0:0:c77
```

Why do you think you have multiple ip addresses?

Networking with Java

Sockets

- A *socket* is one endpoint of a two-way communication link between two programs running on the network.
- A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to
- There are two broad categories of Sockets:
  - Stream Sockets
    - Maintains connection between two endpoints
    - Guarantees reliability, Bi-directional (uses TCP/IP)
  - Datagram Sockets
    - Connection less. Used for one-way messages
    - Does not offer reliability (uses (UDP/IP)
- Most Internet services (protocols), e.g. FTP, TELNET, HTTP are build on top of Stream Sockets. While, SNMP (Simple Network Management Protocol) uses Datagram Sockets

Networking with Java

Java Stream Sockets

- Java's stream socket enables applications to view networking as if it were merely file I/O
- That means you really can read and write from sockets as if you were reading and writing to a file
- Stream sockets enable a process to establish a connection to another process and data flows continuously between them
- Stream connection provides a connection-oriented services using TCP providing reliable, in-order byte-stream service
- Java wraps OS sockets (over TCP) by the objects of class java.net.Socket

Two constructors:

Socket (String hostname, int port) throws UnknownHostException, IOException

Socket(InetAddress ipAddress, int port) throws IOException

Networking with Java

Sockets

- Examine the Socket by using methods

  InetAddress getInetAddress(): returns InetAdress

  int getPort(): returns remote port

  int getLocalPort(): returns the local port to which the invoking Socket object is bound

- Write and read using streams:

  InputStream getInputStream()

  OutputStream getOutputStream()

- Other available methods:

  connect(): allows new connection

  isConnected(): returns true if the socket is connected

  close(): call it to close a connection

Socket example: InterNic website

InterNIC

Home Registrars FAQ Whois

Whois Search Results

Search again (.aero, .arpa, .asia, .biz, .cat, .com, .coop, .edu, .info, .int, .jobs, .mobi, .museum, .name, .net, .org, .pro, or .travel) :

alibaba.com

- (•) Domain (ex. internic.net)
- ( ) Registrar (ex. ABC Registrar, Inc.)
- ( ) Nameserver (ex. ns.example.com or 192.16.0.192)

Submit

```
Whois Server Version 2.0

Domain names in the .com and .net domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.

   Domain Name: ALIBABA.COM
   Registrar: MARKMONITOR INC.
   Whois Server: whois.markmonitor.com
   Referral URL: http://www.markmonitor.com
   Name Server: NS8.ALIBABAONLINE.COM
   Name Server: NSHZ.ALIBABAONLINE.COM
   Name Server: NSP.ALIBABAONLINE.COM
   Name Server: NSP2.ALIBABAONLINE.COM
   Status: clientDeleteProhibited
   Status: clientTransferProhibited
   Status: clientUpdateProhibited
   Status: serverDeleteProhibited
   Status: serverTransferProhibited
   Status: serverUpdateProhibited
   Updated Date: 20-jun-2012
   Creation Date: 15-apr-1999
   Expiration Date: 23-may-2022

>>> Last update of whois database: Fri, 29 Aug 2014 20:32:14 UTC <<<
```

Networking with Java

Socket example: create a whois look-a-like

```
public static void testWhoIs(String hostName)
                                throws IOException {
        Socket s = new Socket("whois.internic.net",
                                              43);
        InputStream in = s.getInputStream();
        OutputStream out = s.getOutputStream();
        hostName = "www.alibaba.com" + "\n";
        byte buf[] = hostName.getBytes();
        out.write(buf);

        int c;
        while ((c = in.read()) != -1){
            System.out.print((char) c);
        }
        s.close();
}
```

Socket example: an example output

```
Whois Server Version 2.0

Domain names in the .com and .net domains can now be
registered
with many different competing registrars. Go to
http://www.internic.net
for detailed information.

   Server Name: WWW.ALIBABA.COM.CN
   Registrar: XIN NET TECHNOLOGY CORPORATION
   Whois Server: whois.paycenter.com.cn
   Referral URL: http://www.xinnet.com

>>> Last update of whois database: Fri, 21 Mar 2014
05:30:21 UTC <<<
```

Networking with Java

Client/Server Programming Using Sockets:

Create a ServerSocket object first

ServerSocket server=new ServerSocket(port,queueLength);

//will refuse connection if queueLength exceeds

The server listens indefinitely (or blocks) for an attempt by a client to connect

Socket connection = server.accept();

For input and output between sockets, get the OutputStream and InputStream objects

InputStream input = connection.getInputStream();

OutputStream output = connection.getOutputStream();

After the communication completes, the server closes the connection by invoking *close()* on the Socket and the corresponding streams

Networking with Java

Client/Server Programming Using Sockets (cont.)

Create a client Socket object next

Socket clientConnection=new Socket(serverAddress, port);

Do exactly same thing for input/output in client side as well

Get the OutputStream and InputStream objects

InputStream input = clientConnection.getInputStream();

OutputStream output = clientConnection.getOutputStream();

The server and the client communicate via the InputStream and the OutputStream objects

Networking with Java

- echo client and server application
  - Client type something

```
Problems  @ Javadoc  Declaration  Console  Console
EchoClient (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_40.jdk/Contents/Home/bin/java (Jun 13, 2014, 12:26:56 PM)
Today is Friday the thirteenth, 06/13/2014
echo: Today is Friday the thirteenth, 06/13/2014
How is Friday The Thirteenth Treating you?
echo: How is Friday The Thirteenth Treating you?
Did you know that it is Full moon as well and is called 'Honey' Moon
echo: Did you know that it is Full moon as well and is called 'Honey' Moon
Lots of headlines on "Spooky 'Honey' Moon Casts Glow on Friday the 13th"
echo: Lots of headlines on "Spooky 'Honey' Moon Casts Glow on Friday the 13th"
Happens only 14.3 on average, next will be in 2049
echo: Happens only 14.3 on average, next will be in 2049
```

  - Server can respond

```
Problems  @ Javadoc  Declaration  Console  Console
EchoServer (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_40.jdk/Contents/Home/bin/java (Jun 13, 2014, 12:26:41 PM)
Received a echo request of: Today is Friday the thirteenth, 06/13/2014
Received a echo request of: How is Friday The Thirteenth Treating you?
Received a echo request of: Did you know that it is Full moon as well and is called 'Honey' Moo
Received a echo request of: Lots of headlines on "Spooky 'Honey' Moon Casts Glow on Friday the
Received a echo request of: Happens only 14.3 on average, next will be in 2049
```

Networking with Java

- echo client and server application

Echo! Server code

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

        if (args.length != 1) {
            System.err.println("Usage: java EchoServer
                        <port number>");
            System.exit(1);
        }
        try (
            ServerSocket serverSocket = new ServerSocket
                      (Integer.parseInt(args[0]));
            Socket clientSocket = serverSocket.accept();
            PrintWriter out = new PrintWriter
                 (clientSocket.getOutputStream(), true);
            BufferedReader in = new BufferedReader
                            (new InputStreamReader
                     (clientSocket.getInputStream()));
        )
```

Networking with Java

- echo client and server application

Echo! Server code (cont.)

```
public class EchoServer {
    public static void main(String[] args) throws
      ….
      try (. . .
        ) {
            String inputLine;
            while ((inputLine = in.readLine()) != null)
            {
                   System.out.println("Received a echo
                        request of: " + inputLine);
                   out.println(inputLine);
            }
        } catch (IOException e) {
            System.out.println("Exception caught when" +
                + "trying to listen on port "
                + portNumber + " or listening for a "
                + "connection");
            System.out.println(e.getMessage());
        }
    }
}
```

Networking with Java

- echo client and server application

Echo! Client code

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

        try (
            Socket echoSocket =
                   new Socket(hostName, portNumber);
            PrintWriter out =  new PrintWriter
                  (echoSocket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                    new InputStreamReader
                      (echoSocket.getInputStream()));
            BufferedReader stdIn =
                new BufferedReader(
                    new InputStreamReader(System.in))
        )
```

Networking with Java

- echo client and server application

Echo! Client code (cont.)

```
public class EchoClient {
    public static void main(String[] args) throws
                                        IOException {
. . .
        try (. . . ) {
            String userInput;
            while ((userInput = stdIn.readLine())
                                        != null) {
                out.println(userInput);
                System.out.println("echo: " +
                                        in.readLine());
            }
        } catch (UnknownHostException e) {
                . . .
            System.exit(1);
        } catch (IOException e) {
                . . .
            System.exit(1);
        }
    }
}
```

```
bin — bash — 61×6
bineetsarmasmbp:bin bineetsharma$ ls
EchoClient.class                KnockKnockProtocol.class
EchoServer.class                KnockKnockServer.class
KnockKnockClient.class          URLReader.class
bineetsarmasmbp:bin bineetsharma$ java KnockKnockServer 4444
bineetsarmasmbp:bin bineetsharma$
```

Knock, Knock Joke! A Client/Server Socket Ex.

```
bin — bash — 80×30
^Cbineetsarmasmbp:bin bineetsharma$ java KnockKnockClient 127.0.0.1 4444
Server: Knock! Knock!
Who's there?
Client: Who's there?
Server: Turnip
Turnip who?
Client: Turnip who?
Server: Turnip the heat, it's cold in here! Want another? (y/n)
y
Client: y
Server: Knock! Knock!
Who?
Client: Who?
Server: You're supposed to say "Who's there?"! Try again. Knock! Knock!
Who's there?
Client: Who's there?
Server: Little Old Lady
Who?
Client: Who?
Server: You're supposed to say "Little Old Lady who?"! Try again. Knock! Knock!
Who's there?
Client: Who's there?
Server: Little Old Lady
Little Old Lady who?
Client: Little Old Lady who?
Server: I didn't know you could yodel! Want another? (y/n)
n
Client: n
Server: Bye.
bineetsarmasmbp:bin bineetsharma$
```

Knock, Knock Joke! Establish a protocol first

```
public class KnockKnockProtocol {
    //state
    private static final int WAITING = 0;
    private static final int SENTKNOCKKNOCK = 1;
    …
    //Q & A
    private String[] clues = { "Turnip", "Little Old
                               Lady", "Atch", "Who", "Who" };
    private String[] answers = { "Turnip the heat,
                                 it's cold in here!",
                                 "I didn't know you
                                  could yodel!",
                                 "Bless you!",
                                 "Is there an owl in
                                   here?",
                                 "Is there an echo in
                                   here?" };
```

Networking with Java

Knock, Knock Joke! Protocol (Cont.)

```
public String processInput(String theInput) {
        String theOutput = null;

        if (state == WAITING) {
            theOutput = "Knock! Knock!";
            state = SENTKNOCKKNOCK;
        } else if (state == SENTKNOCKKNOCK) {
            if (theInput.equalsIgnoreCase("Who's
                                        there?")) {
                theOutput = clues[currentJoke];
                state = SENTCLUE;
            } else {
                theOutput = "You're supposed to say
                        \"Who's there?\"! " +
                    "Try again. Knock! Knock!";
            }
        } else if (state == SENTCLUE) {
//more testing …..
        return theOutput;
    }
}
```

Networking with Java

Knock, Knock Joke! Client

```
public class KnockKnockClient {
    public static void main(String[] args) throws
                                        IOException {
        Socket kkSocket = new Socket(hostName,
                                    portNumber);
        PrintWriter out = new
            PrintWriter(kkSocket.getOutputStream(),
                                            true);
        BufferedReader in = new
            BufferedReader(new InputStreamReader(
                            kkSocket.getInputStream()));

        BufferedReader stdIn = new BufferedReader(new
            InputStreamReader(System.in));

        String fromServer;
        String fromUser;
```

Networking with Java

Knock, Knock Joke! Client (Cont.)

```
while ((fromServer = in.readLine()) != null) {
        System.out.println("Server: " +
                            fromServer);
        if (fromServer.equals("Bye."))
            break;

        fromUser = stdIn.readLine();
        if (fromUser != null) {
            System.out.println("Client: " +
                                fromUser);
            out.println(fromUser);
        }
    }
```

Knock, Knock Joke! Server

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

    int portNumber = Integer.parseInt(args[0]);

    ServerSocket serverSocket = new
                        ServerSocket(portNumber);
    Socket clientSocket = serverSocket.accept();
    PrintWriter out = new
        PrintWriter(clientSocket.getOutputStream(),
                                            true);
     BufferedReader in = new BufferedReader(new
        InputStreamReader(
                clientSocket.getInputStream()));

     String inputLine, outputLine;
```

Networking with Java: Multiple Clients

- Quite often multiple clients connect to a single server
- This is achieved by a constantly running server which serves to all clients simultaneously
- How is that possible?
- As you know the accept() method is a blocking method, meaning, it will block until a client connects
- You can achieve multiple clients services using threads
- When a client connects to a server, server will create a new thread for this client, and go back to listen to other clients to connect to

Networking with Java: Working With URL

- Java allows you to work in higher level protocols than TCP/IP
- You usually work with HTTP protocol in connecting with different computers in World Wide Web
- Uniform Resource Locator (URL) allows you to connect with another machine and exchange information
- URL is a reference (address) to a resource on the internet
- A URL has two main components:
  - Protocol identifier: For the URL http://oracle.com, the protocol identifier is http (HyperText Transfer Protocol). Other protocols:
    - FTP (File Transfer Protocol), Gopher, File, and News.
  - Resource name: For the URL http://oracle.com, the resource name is oracle.com.

Networking with Java: Working With URL

- Resource name is the complete address of the resource and usually includes the following:
  - Host Name: Name of the machine which has resources
  - Filename: Pathname to the file (resource) on the machine
  - Port Number: A port number to connect for this application
  - Reference: Reference to a named anchor within a resource. Usually identifies a specific location of a file (path)
  - Query: Additional parameters starting with a '?'
- Examples:
  - http://course.ucsc-extension.edu:80/modules/shop/index.html?action=courseSearch

Networking with Java: URL Definition

(from WIKI)

```
  foo://username:password@example.com:8042/over/there/index.dtb?type=animal&name=narwhal#nose
  \_/   \_______________/ \_________/ \__/            \___/ \_/ \_______________________/ \__/
   |           |               |       |                |    |           |                |
   |       userinfo         hostname  port              |    |          query          fragment
   |    \_______________________________/\_____________|____|/ \__/        \__/
   |                    |                         |    |    |    |          |
   |                    |                         |    |    |    |          |
scheme              authority                    path   |    |    interpretable as keys
 name   \_________________________________________|____|/       \____/     \_____/
   |                         |                    |    |          |           |
   |                 hierarchical part            |    |    interpretable as values
   |                                              |    |
   |            path               interpretable as filename |
   |   ___________|____________                   |
  / \ /                        \                  |
  urn:example:animal:ferret:nose               interpretable as extension

                path
          ________|________
 scheme /                 \
  name  userinfo  hostname      query
  _|__   ___|__   ____|____   _____|_____
 /    \ /      \ /         \ /           \
 mailto:username@example.com?subject=Topic
```

Networking with Java: Working With URL

- Working with URL in Java:
  - You typically use URL class in Java to parse url. To create a URL object use any of its constructors
  - The constructor throws MalformedURLException in case it finds errors in the URL provided
- Constructing URLs:
  - URL url1 =new URL("http://course.ucsc-extension.edu");
  - URL url2 =new URL("http","course.ucsc-extension.edu",
    80,"modules");
  - URL url3=new URL(url2, "shop/index.html?
    action=courseSearch/");
    - If the string is not an absolute URL, then it is considered relative to the URL

Networking with Java: Working With URL

- URL class provides many useful methods for parsing URLs
  - getProtocol(), getHost(), getPort(), getPath(), getQuery()

Parsing a URL:

```
public static void parsingAnURL(String urlString)
                            throws Exception{
     URL aURL = new URL(urlString);

     System.out.println("protocol = " +
                              aURL.getProtocol());
     System.out.println("authority = " +
                              aURL.getAuthority());
     System.out.println("host = " + aURL.getHost());
     System.out.println("port = " + aURL.getPort());
     System.out.println("path = " + aURL.getPath());
     System.out.println("query = " +
                                aURL.getQuery());
    System.out.println("filename = " +
                                aURL.getFile());
     System.out.println("ref = " + aURL.getRef());
}
```

Networking with Java: Working With URL

Parsing a URL, an example:

```
parsingAnURL("http://example.com:80/docs/books/tutorial"
            +"/index.html?name=networking#DOWNLOADING");


//Output will look like this:

protocol = http
authority = example.com:80
host = example.com
port = 80
path = /docs/books/tutorial/index.html
query = name=networking
filename =
/docs/books/tutorial/index.html?name=networking
ref = DOWNLOADING
```

Networking with Java: Working with URL

- The HTTP protocol specifies that strings passed as arguments has to be in an allowable format
  - For example: a string like this
    - st=california/area=bay/uc=SC
  - Is translated as:
    - st%3dcalifornia%2farea%3dbay%2fuc%3dSC
- You typically use a URLEncoder class which will examine each character in search string and:
  - Space is converted to + sign
  - The bytes of all special characters are replaced by hexadecimal numbers, preceded by %
  - The encoding and decoding is done by: encode(), decode() methods

Networking with Java: Working with URL

- How do you retrieve data associated with an URL?
- Use URLConnection class to retrieve actual content of a resource identified by a URL
- The URLConnection class is a general purpose class for accessing the attributes of a remote resource
- Call openConnection() method of URL reference to establish the connection to the actual resource – it returns an object of URLConnection class for that protocol
  - For HTTP it will return object of HttpURLConnection
- This class encapsulates all socket management and HTTP directions required to obtain the resource. You don't need to worry about low level calls

Networking with Java: Working with URL

- URLConnection class has several methods to get information about the resources and its headers:
  - Long getContentLength(): returns size of content
  - String getContentType(): returns content-type header field
  - String getHeaderField(int idx): value of header field at idx
  - String getHeaderFieldKey(int idx): value of header key at idx
  - Map<String, List<String>> getHeaderFields(): returns a map that contains all of the header fields and values

Networking with Java: Working with URL

- HttpURLConnection is subclass of URLConnection and encapsulates all HTTP transaction over sockets, e.g.:
  - Content decoding, redirection, proxy indirection
- The object of HttpURLConnection is also obtained using openConnection()method
- The HttpURLConnection adds:
  - String getRequestMethod(): Returns a string representing how URL requests are made. Default is GET (POST is also available)
  - void setRequestMethod(String how): Sets the HTTP requests specified by how, default is GET (POST is also available)

Networking with Java: Working with URL

- A typical HTTP message structure looks like this:

  Request/Status-Line \r\n

  Header1: value1 \r\n

  Header2: value2 \r\n

  ...

  HeaderN: valueN \r\n

  \r\n

- Reading HTTP message: Several ways to interpret bytes of the body:
  - Binary: images, compressed files, class files etc
  - Text: ASCII, Latin-1, UTF-8, etc

Networking with Java: Working with URL

- Applications parse the headers of the message, and process the body according to information supplied by the headers:
  - Content-Type
  - Content-Encoding
  - Transfer-Encoding

Test HttpURLConnection

```
URL hp = new URL(urlString);
HttpURLConnection hpCon =
            (HttpURLConnection)hp.openConnection();

System.out.println("Request method is " +
                   hpCon.getRequestMethod());
System.out.println("Response Code is " +
                   hpCon.getResponseCode());
System.out.println("Response Message is " +
                   hpCon.getResponseMessage());
```

Test HttpURLConnection (Cont.)

```
//get a list of the header fields
//and a set of header keys

        Map<String, List<String>> hdrMap =
                hpCon.getHeaderFields();
        Set<String> hdrField = hdrMap.keySet();
        System.out.println("\nHere is the header:");

//display all header keys and values

        for (String k: hdrField) {
            System.out.println("Key: " + k + "  Values:
                                " + hdrMap.get(k));
        }
```

Test HttpURLConnection (Cont.)

- Output from a www.google.com URL

```
//get a list of the header fields
//and a set of header keys

Request method is GET
Request response is 200
Request response message is OK

Here is the header:
Key: null  Values: [HTTP/1.1 200 OK]
Key: X-Frame-Options  Values: [SAMEORIGIN]
Key: Transfer-Encoding  Values: [chunked]
Key: Date  Values: [Fri, 21 Mar 2014 19:39:21 GMT]
Key: P3P  Values: [CP="This is not a P3P policy! See
http://www.google.com/support/accounts/bin/answer.py?hl=
en&answer=151657 for more info."]
Key: X-XSS-Protection  Values: [1; mode=block]
Key: Expires  Values: [-1]
…… //more
```

Networking with Java

Reading directly from URL

- Call openStream() method of a URL object, which returns a java.io.InputStream object
- This makes reading from a URL as easy as reading from an input stream

```
URL oracle = new URL(strURL);
 BufferedReader in = new BufferedReader(
 new InputStreamReader(oracle.openStream()));

 String inputLine;
 while ((inputLine = in.readLine()) != null)
     System.out.println(inputLine);

 in.close();
```

Networking with Java

An example output from http://oracle.com/

```
<!DOCTYPE html>

<html  lang="en-US">
<head><meta content="text/html; charset=utf-8"
       http-equiv="Content-Type" />
<script type="text/javascript">
   var _U = "undefined";
   var g_HttpRelativeWebRoot = "/ocom/";
   var SSContributor = false;
   var SSForceContributor = false;
   var SSHideContributorUI = false;
   var ssUrlPrefix = "/us/";
   var ssUrlType = "2";

   var g_navNode_Path = new Array();
       g_navNode_Path[0] = '8';
   var g_ssSourceNodeId = "8";
   var g_ssSourceSiteId = "ocomen";
</script>
<script id="SSNavigationFunctionsScript"></script>
```

Networking with Java: Datagrams

- A *datagram* is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed
- Why would you need it?
  - There are many application who will prefer this because of speed
  - SMTP etc., mail, broadcast messages
- Java implements datagram on top of the User Datagram Protocol (UDP) by using two classes:
  - DatagramSocket: used to send and receive both
  - DatagramPacket: the packet (data) itself

Networking with Java: Datagrams

- In UDP, there is no 'connection' between servers or handshaking between them
- The sender explicitly attaches the IP address and port of the destination inside each packet
- The server must extract the IP address and port of the sender from the receiver packet to identify sender
- From application viewpoint, UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server
- A datagram's arrival, arrival time and order of arrival is not guaranteed

Putting it all together

Network Programming

Code walk through

JDK 8 Specifics

- What's New In JDK 8: Overview
- Default Methods in Interface
- Lambda Expression

What's New in JDK 8: Overview

- It is a major upgrade since Java inception, adds plenty of new features in:
  - Language
    - Lambda Expressions: enables functionality as a method argument
    - Method references: provides easy-to-read lambda expressions for methods that already have a name
    - Default methods: Added in the interface
  - Libraries
    - Collections
      - Stream API is integrated into the Collections API
      - Performance improvement for HashMaps with Key Collisions
      - Date-Time package
  - Security, JavaFX, Internationalization, JVM, tools (jjs, java, javadoc)
- More here: http://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html

Default Interface Method

- JDK 8 allows you to define methods in the interface instead of leaving them as abstract

```
interface MyIF {
    // This is a "normal" interface method declaration.
    // It does NOT define a default implementation.
    int getNumber();

    // This is a default method. Notice that it provides
    // a default implementation.
    default String getString() {
        return "Default String";
    }
}
```

- Provides a mechanism to add new methods to existing interfaces without breaking backwards compability

Default Interface Method

- Allows you to include static methods in an interface
- Static methods, by definition, are not abstract

```
interface MyIF2 {
    // This is a "normal" interface method declaration.
    // It does NOT define a default implementation.
    int getNumber();

    // This is a default method. Notice that it provides
    // a default implementation.
    default String getString() {
        return "Default String";
    }

    // This is a static interface method.
    static int getDefaultNumber() {
        return 0;
    }
}
```

Lambdas

- Lambdas are the most important new addition to Java 8
- Lambda expressions:
  - Enables functional programming
  - Allows leaner and more compact code
  - Facilitates parallel programming
  - Helps in developing more generic, flexible and reusable APIs
  - Allows you to pass behaviors as well as data to functions

Lambda Expression: Fundamentals

- Lambda expression introduces a new syntax element and operator into the Java language.
  - The new operator, *lambda operator or* the *arrow operator, is* **->**
  - It divides a lambda expressions into two parts.
  - The left side specifies any parameters required by the lambda expression (() is used if there are no parameters)
  - On the right side is the *lambda body,* which specifies the actions of the lambda expression.
    - Two types of body, single expression or block of code
  - -> can be verbalized as "becomes" or "go to"

Lambda Expression: Fundamentals

- A Java 8 lambda expression is basically a method in Java without a declaration, usually written as:
  - (parameters) -> {body}
- Simplest lambda expression example:

  ```
  () -> 123.45
  ```

  - This lambda expression takes no parameters, thus the parameter list is empty.
  - It returns the constant value 123.45
  - Thus, it is equivalent to the following method:

  ```
  double myMeth() { return 123.45; }
  ```

Lambda Expression: Fundamentals

- More examples:

```
1.    ( ) -> k
2.    (k) -> k
3.    k -> 2 % k
4.    (int k, int l) -> { return k * l; }
```

- A lambda can have zero or more parameters, separated by commas and their type can be explicitly declared or inferred from the context.
- Parenthesis are not needed around a single parameter.
- () is used to denote zero parameters.
- Body can contain zero or more statements.
- Braces are not needed around a single-statement body.

Lambdas: Functional Interface

- A *functional interface* is an interface that contains one and only one abstract method.
- Normally, this method specifies the intended purpose of the interface.
- Thus, a functional interface typically represents a single action. For example:
  - The standard interface **Runnable** is a functional interface because it defines only one method: **run()**. Therefore, **run()** defines the action of **Runnable**
  - Furthermore, a functional interface defines a *target type* of a lambda expression.
  - A lambda expression can only be used in a context in which its target type is specified

Lambdas: Functional Interface

- Previously all interface methods were abstract
- Now, an interface method is abstract only if it does not specify a default implementation.
- Because non-default interface methods are implicitly abstract, there is no need to use the **abstract** modifier.
- Example of a *functional interface:*

```
interface MyNumber {
    double getValue();
}
```

  - Here, method **getValue()** is implicitly abstract, and only method defined by **MyNumbe**r. Hence **MyNumber** is a functional interface and its function is defined by **getValue()**

Lambdas: Functional Interface

- Lambda expression is not executed on its own.
- It forms the implementation of the abstract method defined by the functional interface that specifies the target type
- As a result, a lambda expression can be specified only in a context in which a target type is defined.
- One of these contexts is created when a lambda expression is assigned to a functional interface.
- Examples of other target type contexts:
  - Variable initialization, return statements, and method arguments

Lambdas: Functional Interface

- Working through an example:
  - A functional interface

```
interface MyNumber {
    double getValue();
}
```

  - Create a reference to the functional interface

```
MyNumber myNum;
```

  - Assign a lambda expression to that interface reference:

```
myNum = () -> 123.45;
```

  - Now, use the reference to call the lamda espression

```
System.out.println(" " + myNum.getValue());
```

Lambdas: Functional Interface

- When a lambda expression occurs in a target type context, an instance of a class is automatically created that implements the functional interface, with the lambda expression defining the behavior of the abstract method declared by the functional interface.
- When that method is called through the target, the lambda expression is executed. Thus a lambda expression gives us a way to transform a code segment into an object.
- In the preceding example, the lambda expression becomes the implementation for the **getValue()** method.
- As a result, the following displays the value 123.45;

```
System.out.println(" " + myNum.getValue());
```

Lambda Expression: Examples

- Complete example as described earlier

```
interface MyNumber {
  double getValue();
}

public class Listing_1 {
  public static void main(String args[])
  {
    MyNumber myNum;//declare an interface reference
  // Here, the lambda expression is simply a
  // constant expression. When it is assigned to
  //myNum, a class instance is constructed in which
  //the lambda expression provides an override
  // of the getValue() method in MyNumber.
    myNum = () -> 123.45;

 // Call getValue(), which is overridden by the
 // previously assigned lambda expression.
    System.out.println("A fixed value: " +
                       myNum.getValue());
   //more code
  }
}
```

Lambda Expression: Examples

- Complete example as described earlier (contd.)

```
 // Continuing
 // …
// Here, a more complex expression is used.
    myNum = () -> Math.random() * 100;

 // These call the lambda exp in the previous line.
    System.out.println("A random value: " +
                       myNum.getValue());
    System.out.println("Another random value: " +
                       myNum.getValue());

 // A lambda expression must be compatible with the
 //method defined by the functional interface.
 //Therefore, this won't work:
//  myNum = () -> "123.03"; // Error!
  }
}
```

```
<terminated> Listing_1 [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/
A fixed value: 123.45
A random value: 27.923134433441977
Another random value: 44.49348602752959
```

Lambda Expression: Examples

- Another example showing use of parameters with the lambda expression

```
interface NumericTest {
    boolean test(int n);
}

public class Listing_2 {
    public static void main(String args[]) {
        NumericTest isEven = (n) -> (n % 2) == 0;

        if (isEven.test(10))
            System.out.println("10 is even");
        if (!isEven.test(9))
            System.out.println("9 is not even");

        NumericTest isNonNeg = (n) -> n >= 0;

        if (isNonNeg.test(1))
            System.out.println("1 is non-
                                negative");
        if (!isNonNeg.test(-1))
            System.out.println("-1 is negative");
    }
}
```

```
10 is even
9 is not even
1 is non-negative
-1 is negative
```

Anonymous Inner Classes

- Inner class listeners can be shortened using anonymous inner classes.
- An *anonymous inner class* is an inner class without a name. It combines declaring an inner class and creating an instance of the class in one step.
- An anonymous inner class is declared as follows:

```
new SuperClassName/InterfaceName() {
  // Implement or override methods in superclass or interface
  // Other methods if necessary
}
```

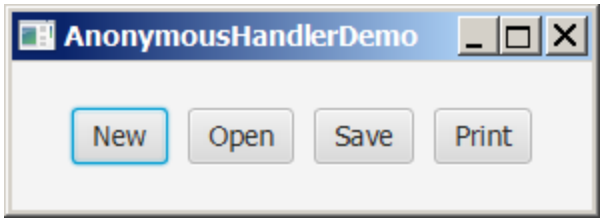
Anonymous Inner Classes (cont.)

```java
public void start(Stage primaryStage) {
  // Omitted

  btEnlarge.setOnAction(
    new EnlargeHandler());
}

class EnlargeHandler
    implements EventHandler<ActionEvent> {
  public void handle(ActionEvent e) {
    circlePane.enlarge();
  }
}
```

(a) Inner class `EnlargeListener`

```java
public void start(Stage primaryStage) {
  // Omitted

  btEnlarge.setOnAction(
    new class EnlargeHandlner
      implements EventHandler<ActionEvent>() {
      public void handle(ActionEvent e) {
        circlePane.enlarge();
      }
    });
}
```

(b) Anonymous inner class



Simplifying Event Handing Using Lambda Expressions

- For example, the following code in (a) can be greatly simplified using a lambda expression in (b) in three lines.

```
btEnlarge.setOnAction(
  new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent e) {
      // Code for processing event e
    }
  }
});
```

(a) Anonymous inner class event handler

```
btEnlarge.setOnAction(e -> {
  // Code for processing event e
});
```

(b) Lambda expression event handler

Single Abstract Method Interface (SAM)

- The statements in the lambda expression is all for that method.
- If it contains multiple methods, the compiler will not be able to compile the lambda expression.
- So, for the compiler to understand lambda expressions, the interface must contain exactly one abstract method.
- Such an interface is known as a *functional interface*, or a *Single Abstract Method* (SAM) interface.

Summary: Networking with Java

- TCP/IP
- HTTP
- Socket
- UDP
- What's New In JDK 8: Overview
- Default Methods in Interface
- Lamda Expression

Next Lecture

- ☺
- Apply what you learned:
  - OOP is easy! Peg yourself to a functioning home
    - Un-learning procedural programming might be most difficult
  - Generics, Multi-threading, JDBC, Networking, GUI
- Be fearless
  - Well, that is tough! Humankind survived because of the fear
  - Just a tad fearless to take next venture for which you are not fully ready!

Next Lecture (cont.) ☺

- Where do you go from here?
  - You have graduated Java, ready to fly away!
  - Stay current in Java. Venture around time to time here
    - http://docs.oracle.com/javase/tutorial/
    - Read more on Java Concurrency, Generic gotchas
  - Refresh your Java by looking at Java online lectures from prestigious Universities – Harvard, Sanford
    - Stay with sites:*.edu during Google search
  - Join coding meet-ups, participate in Hackathons, venture around Hacker Dojo types of places
  - Take Java EE, Hadoop, Android Development courses
  - Read OOD & A, Agile SD Methodologies, Patterns