

# Professional Programming in Java

**Session: 18**

**Java Documentation  
and Networking**



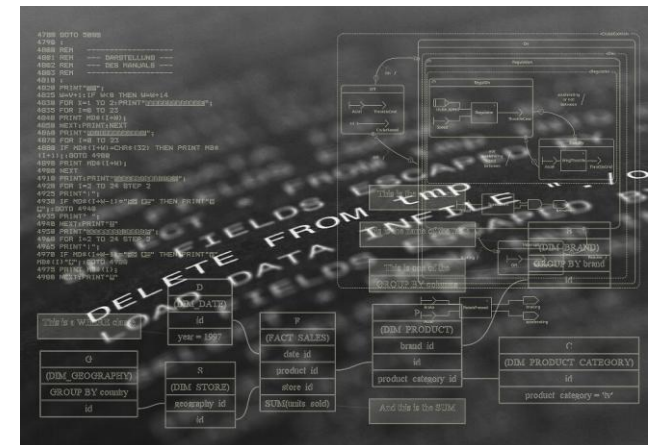


- ◆ Explain the Javadoc tool
- ◆ Describe the key classes of the java.net package
- ◆ Explain socket programming
- ◆ Explain URL processing



## ◆ The Javadoc tool:

- ◆ Enables creating HTML-based API documentation of Java source code
- ◆ Relies on documentation tags present in the source code
- ◆ Is used to create documentation of packages, classes, interfaces, methods, and fields





- ◆ The Java API is a large collection of types where each type can have a large number of constructs, such as constructors, fields, and methods.
- ◆ Developers need to:

Know the purpose of the types and its constructs

Provide documentation of the classes and their constructs

Describe everything that another developer would require to use the API



- ◆ In Java:
  - ◆ API documentation is created using documentation tags
- ◆ The Javadoc tool:

Generates API documentation in HTML

Can be found in the bin folder of the JDK path

Examples: Eclipse and NetBeans also have support for Javadoc



- ◆ The Javadoc tags can be primarily divided into:



Class-level Tags	
Tag	Description
@author	Inserts the author name of the class
{@code}	Inserts text in code format
@since	Inserts a Since heading used to specify from when this class exists
@deprecated	Inserts a comment to indicate that this class is deprecated and should no longer be used



- ◆ Following code snippet demonstrates the use of class-level tags:

## Code Snippet

```
/**
 * @author Carl Boynton
 * @author Andy Payne
 * @see Collection
 * @see Vector
 * @since JDK1.0
 */
public class MathDemo {
    /*Code implementation*/
}
```

The class `MathDemo` will have information indicating who are the authors of the code, which classes to see further, and since which version the class has been existing.



## Method-level Tags

Tag	Description
<code>@param</code>	Inserts a parameter that the method accepts
<code>@return</code>	Inserts the return type of the method
<code>@throws</code>	Inserts any exception that the method throws
<code>@see</code>	Inserts a See Also heading with a link or text points to closely related methods
<code>@since</code>	Inserts a Since heading with a text to specify from when this class exists
<code>@deprecated</code>	Inserts a comment to indicate that this method is deprecated, and should no longer be used





- ◆ Following code snippet demonstrates the use of the method-level tags:

## Code Snippet

```
/**
 * @param num1 This is the first paramter to
 * addInt method
 * @param num2 This is the second parameter to
 * addInt method
 * @return int This returns sum of numA and numB.
 * @see MathDemo#addLong(long,long)
 */
public int addInt(int num1, int num2) {
    return num1 + num2;
}
```

- ◆ The `@see` annotation in the code specifies an `addLong(long, long)` method in the `MathDemo` class. This method must be defined in the `MathDemo` class failing which the Javadoc tool will report a compilation error.



- ◆ Following code snippet demonstrates the use of Javadoc tags and documentation comments:

## Code Snippet

```
/**
 * The {@code MathDemo} class implements a calculation
 * algorithm to
 * add two integers.
 * @author Carl Boynton
 * @author Andy Payne
 * @see Math
 * @since JDK8.0
 */
public class MathDemo {
    /**
     * Constructs a MathDemo instance.
     */
}
```



```
public MathDemo() { }
public long addLong(long num1, long num2) {
    return num1 + num2;
}
/**
 * This method is used to add two integers.
 * @param num1 This is the first parameter to addInt
method
 * @param num2 This is the second parameter to addInt
method
 * @return int This returns sum of numA and numB.
 */
public int addInt(int num1, int num2) {
    return num1 + num2;}
/**
 * This is the main method to use addInt method.
 * @param args Unused.
 * @exception java.io.IOException on input error.
```



```
/**
 * This is the main method to use addInt method.
 * @param args Unused.
 * @exception java.io.IOException on input error.
 * @see java.io.IOException
 */
public static void main(String[] args) throws
java.io.IOException{
    MathDemo mathDemo = new MathDemo();
    System.out.println(mathDemo.addInt(5, 8)); }
}
```

For Aptech Center Use Only



- ◆ The two ways to generate Java documentation are:

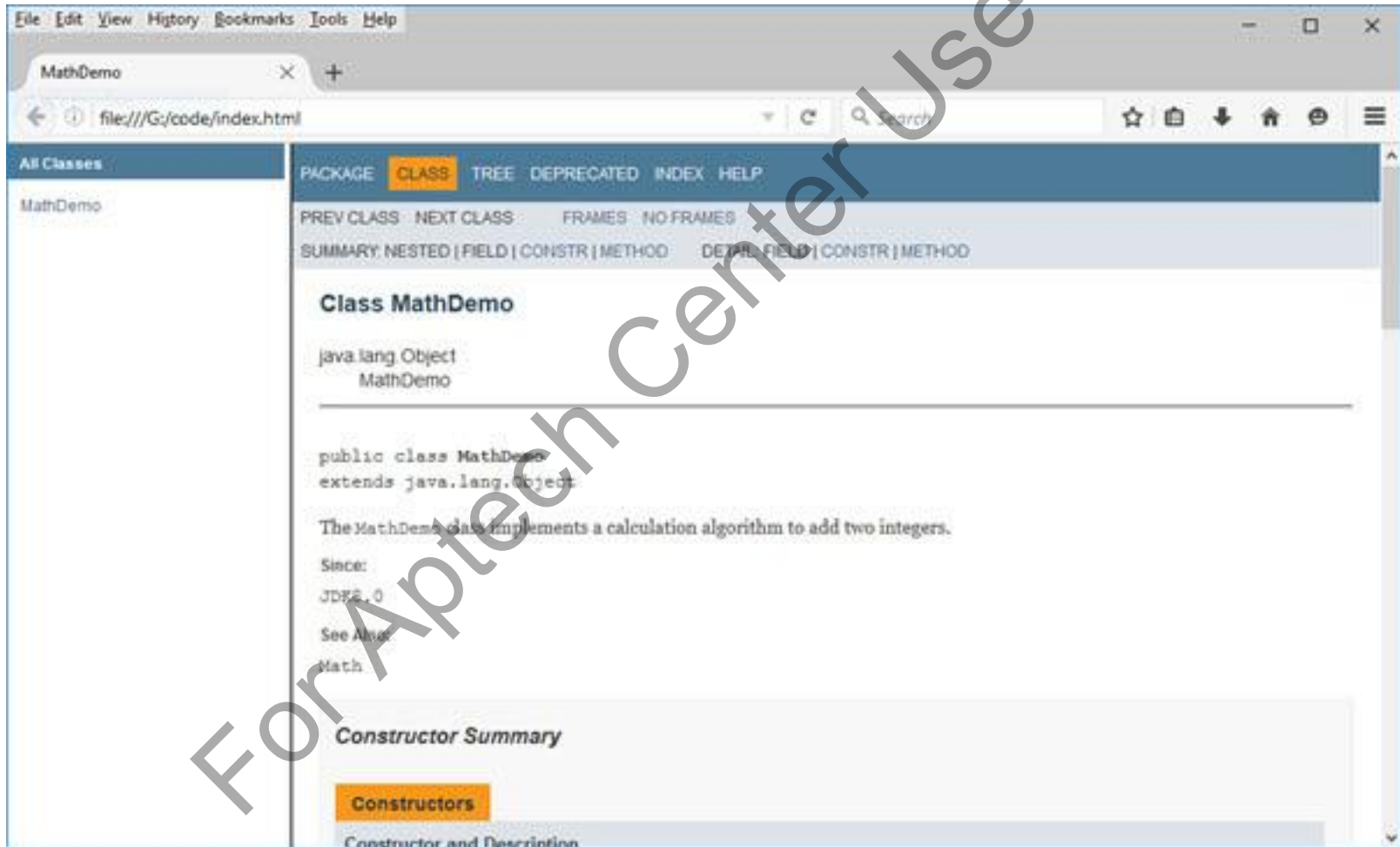
At the command prompt using the Javadoc tool or using the IDE option.

Generating Javadoc is to use the Javadoc generation features of an IDE, such as NetBeans.

- ◆ The command `javadoc MathDemo.java` given at the command prompt results in an HTML file containing the Javadoc generated documentation.



- ◆ Following figure displays the Javadoc generated documentation opened in the browser:





## ◆ NetBeans:

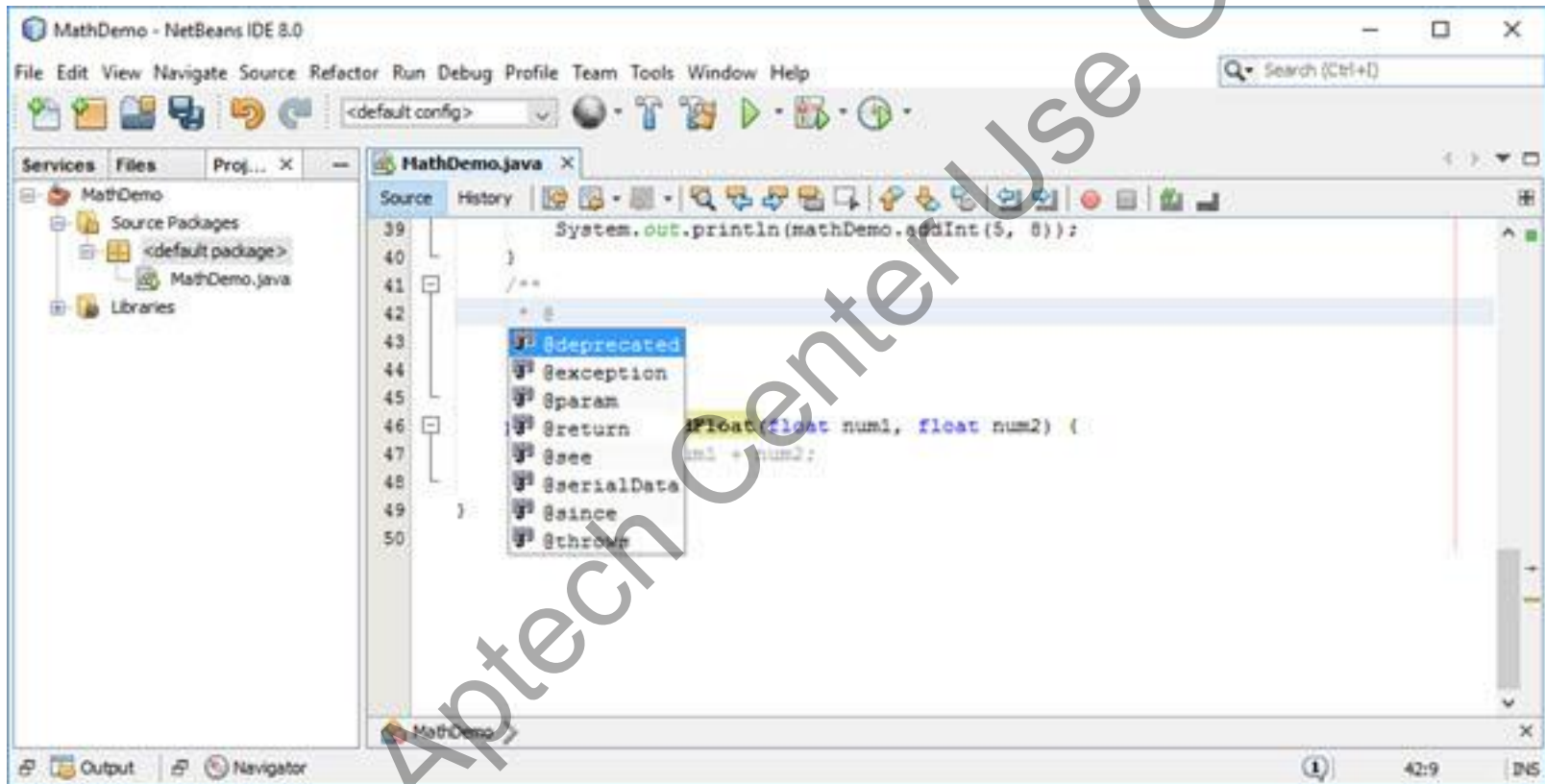
- ◆ Enables automatically inserting Javadoc comments and tags, generating Javadoc, and viewing Javadoc documentation
- ◆ Assists in writing Javadoc through hints and the code-completion feature
- ◆ Can be automatically generated in source files



- ◆ For a Javadoc comment, typing `/**` and pressing the TAB and ENTER key:
  - ◆ Automatically generates a Javadoc comment block
- ◆ For a method, typing `/**` and pressing the TAB and ENTER key:
  - ◆ `@param` and `@return` tags
- ◆ For other tags:
  - ◆ A hint appears as a pop up as a Javadoc tag is typed
- ◆ On clicking a tag or pressing the ENTER key:
  - ◆ The tag is inserted in the source file



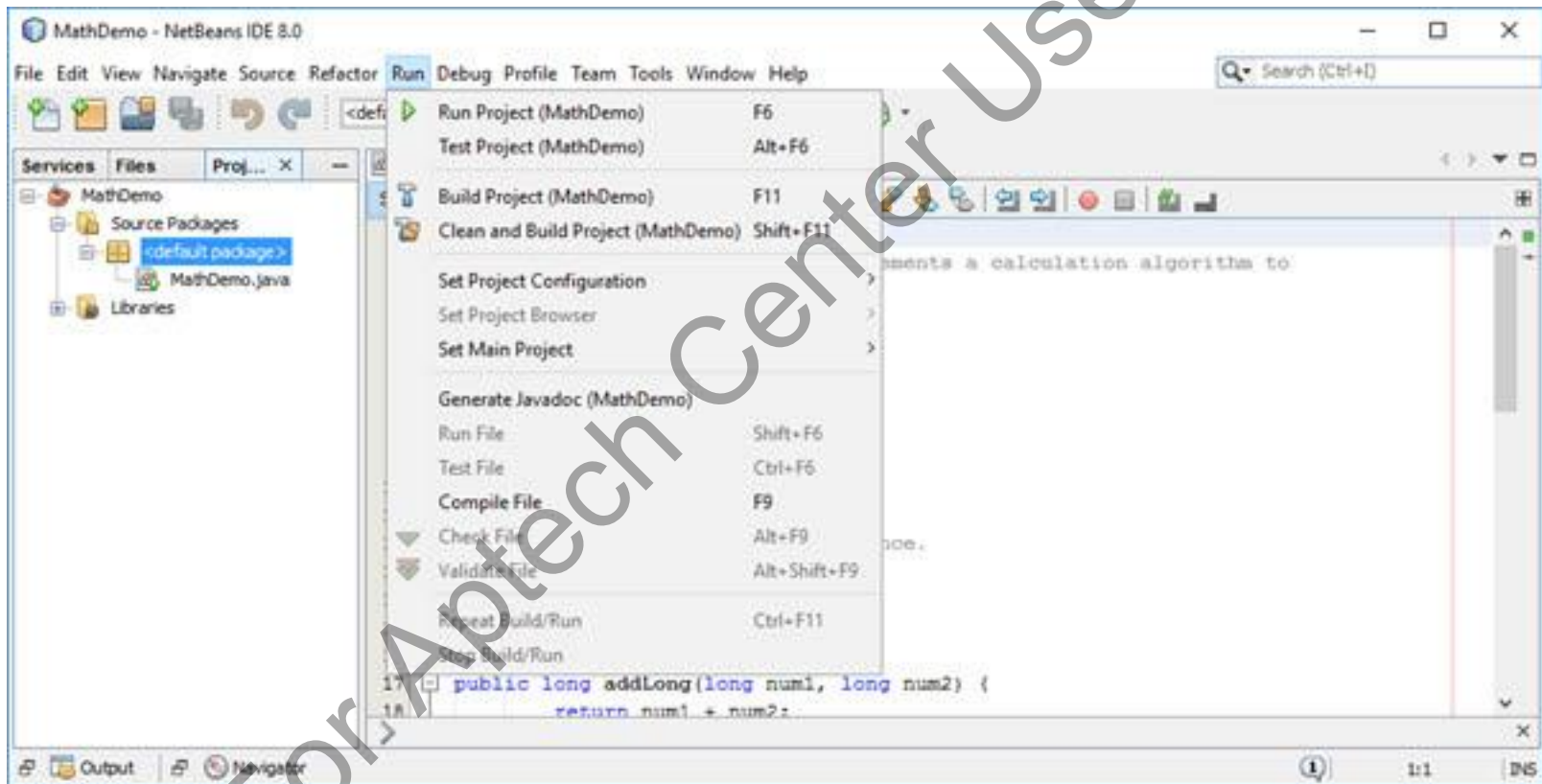
- ◆ Following figure displays the Javadoc tag code-completion pop-up:





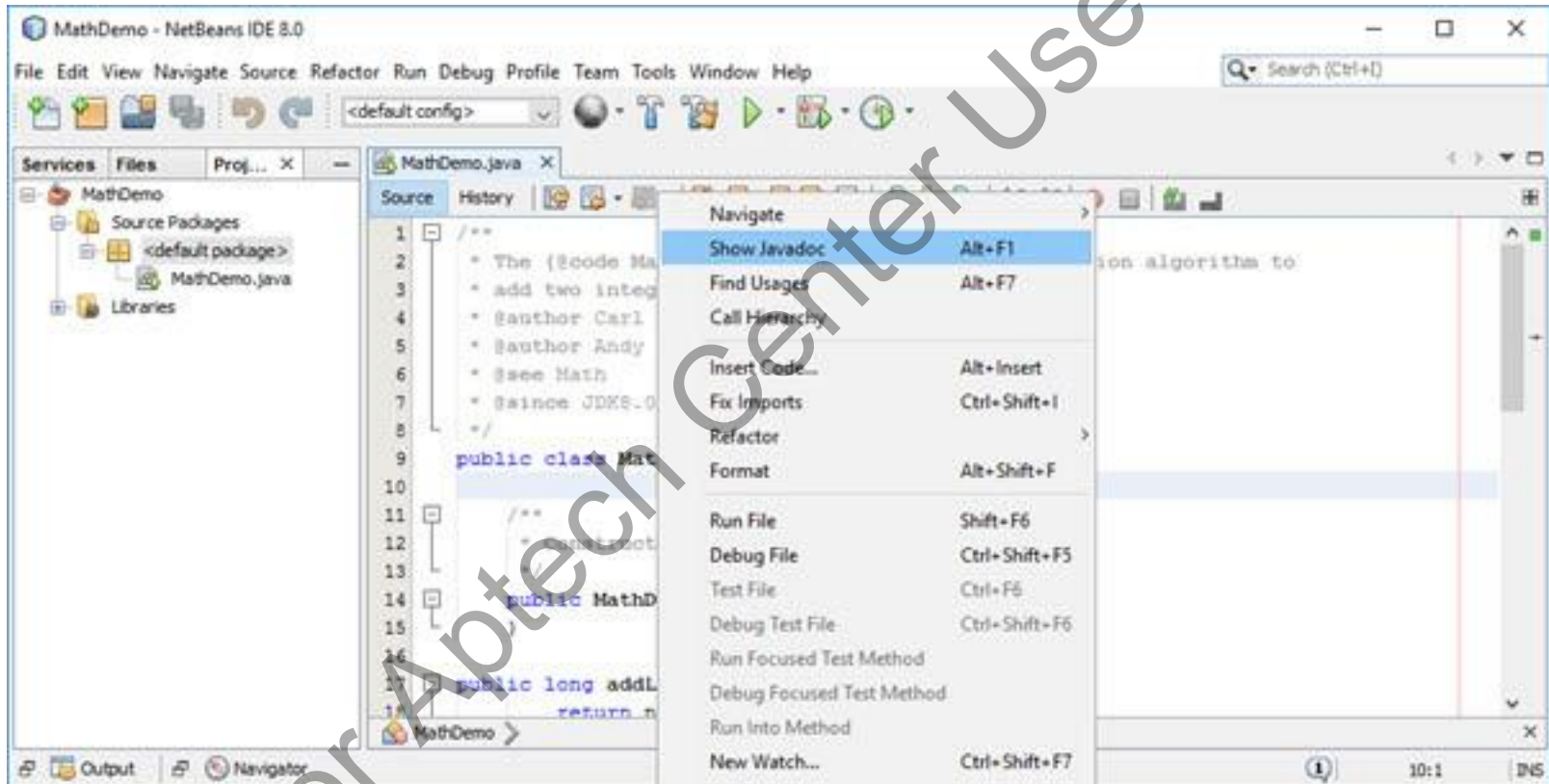


- ◆ Select Run → Generate Javadoc from the main menu of NetBeans to show the following figure that displays the Javadoc Documentation Generation in NetBeans:



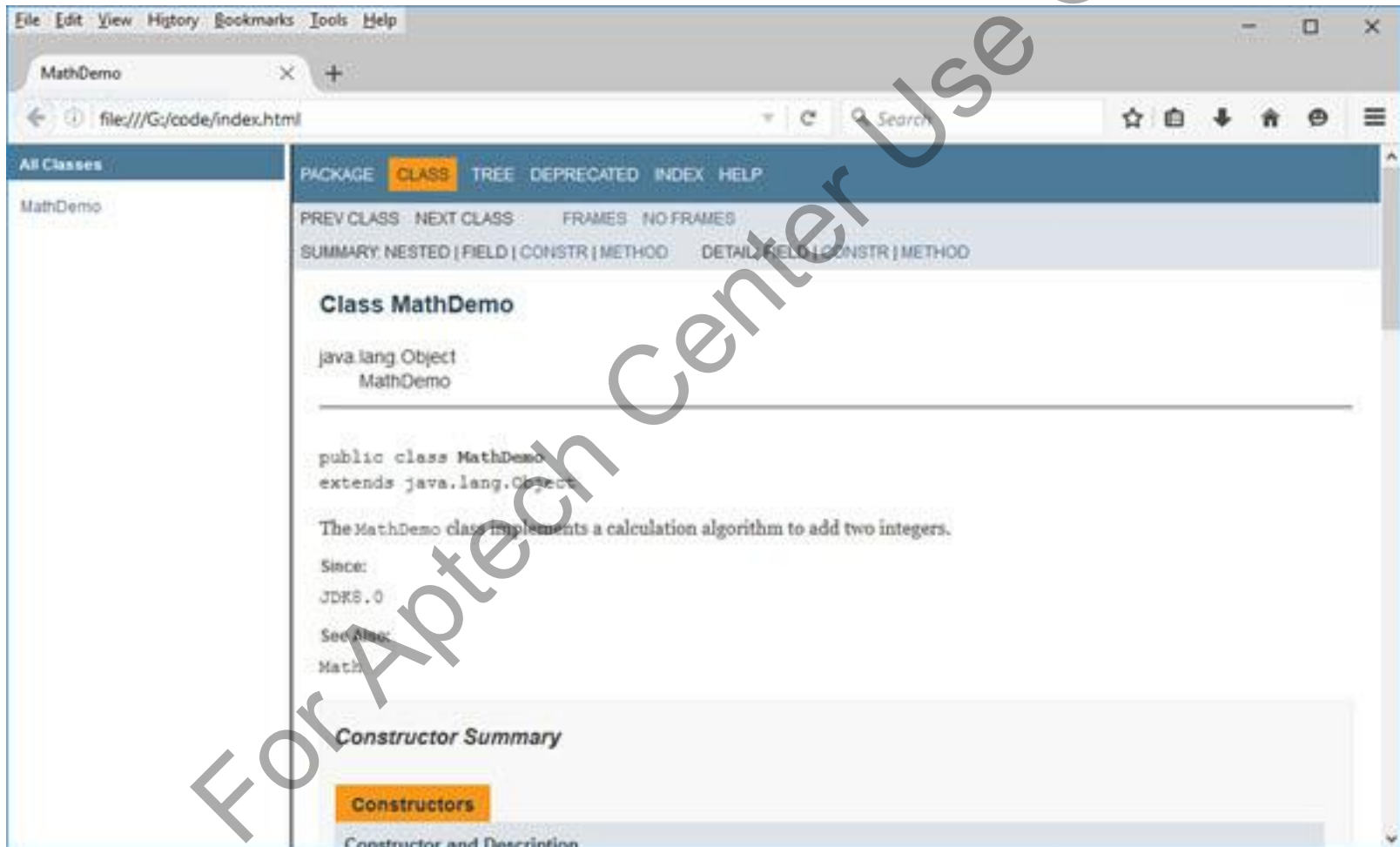


- ◆ Select **Show Javadoc** from the contextual menu to show the following figure that displays Viewing Javadoc in NetBeans:



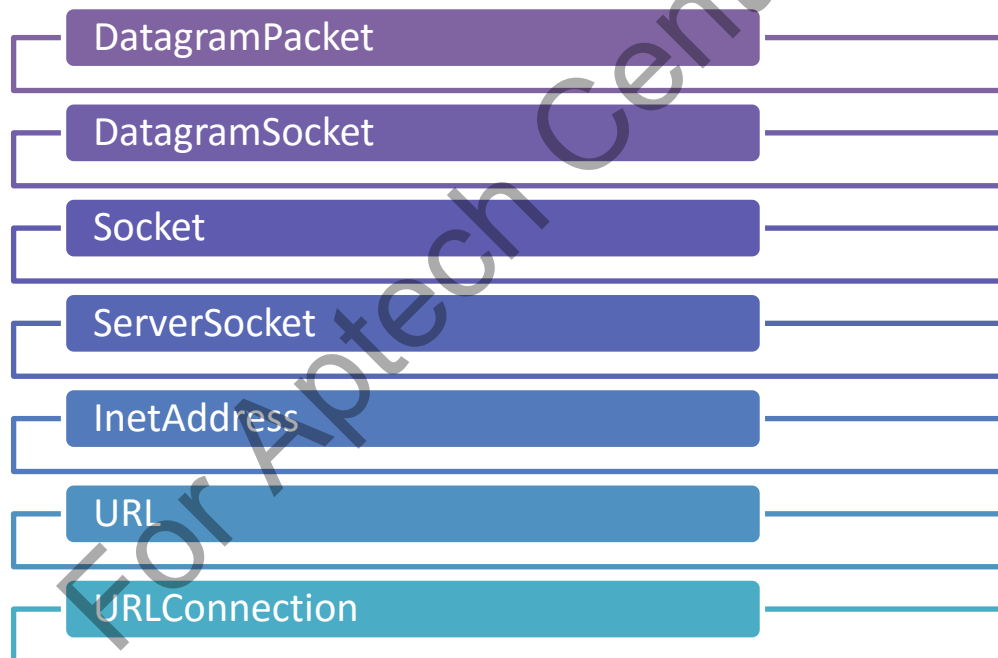


- ◆ Following figure displays the Javadoc generated documentation in the browser:



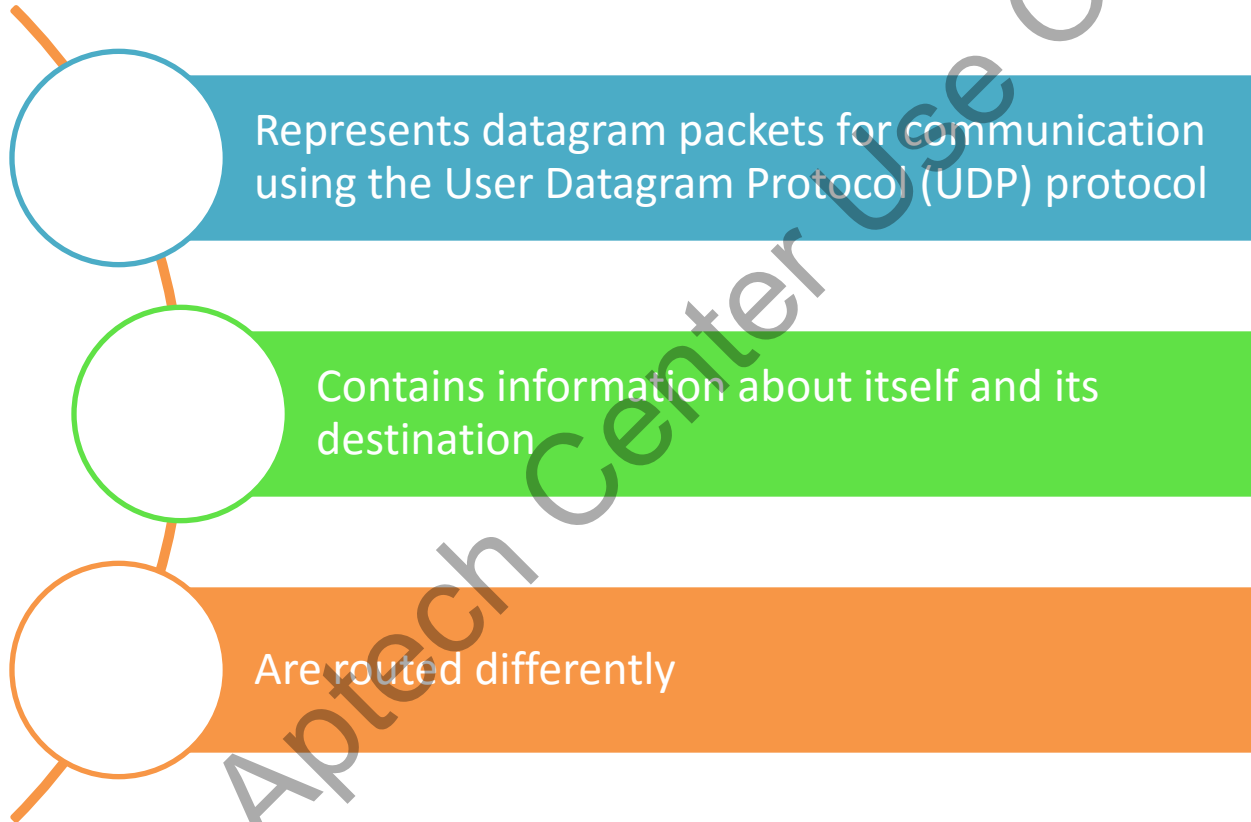


- ◆ The java.net package:
  - ◆ Contains classes and interfaces for network programming
  - ◆ Creates transport layer client and server sockets
  - ◆ Performs communication over the Internet
- ◆ Some of the important classes of the java.net package are:





- ◆ The DatagramPacket class:



**Note:** DatagramPacket object can have a maximum size of 65507 bytes.



- ◆ Following table demonstrates methods of the `DatagramPacket` class:

Method	Description
<code>setData (byte [] buf)</code>	Sets the data of a packet as a <code>byte []</code>
<code>setAddress (InetAddress iaddr)</code>	Sets the IP address of the computer to which a datagram packet needs to be sent
<code>setLength (int length)</code>	Sets the length of a packet as an <code>int</code> value
<code>getData ()</code>	Returns the data of the packet as a <code>byte []</code>
<code>getLength ()</code>	Returns the length of data in a packet to be sent or in a packet that has been received
<code>getAddress ()</code>	Returns the IP address of the computer to which a datagram packet is sent or the computer that sends a datagram packet



- ◆ The `DatagramSocket` class is responsible for sending and receiving datagram packets as `DatagramPacket` objects.
- ◆ Following table demonstrates methods of the `DatagramPacket` class:

Method	Description
<code>connect(InetAddress address, int port)</code>	Connects the socket to the IP address and port of a remote computer
<code>disconnect()</code>	Disconnects the socket
<code>send(DatagramPacket packet)</code>	Sends a <code>DatagramPacket</code> object to a destination
<code>receive(DatagramPacket packet)</code>	Receives a <code>DatagramPacket</code> object



- ◆ The Socket class:

Represents the socket used by both the client and server for communicating

Is used for communication over the Transmission Control Protocol (TCP) protocol

- ◆ The TCP protocol:

Maintains a connection between endpoints that Socket objects represents

Guarantees both because both the client and server sockets remains connected





- ◆ To transmit data to a server:

A client creates an object of the `Socket` class.

The server obtains a `Socket` object by calling the `accept()` method of the `ServerSocket` class.

- ◆ A client can create a `Socket` to represent a connection to the server by:

Invoking the public `Socket(String host, int port)` constructor of the `Socket` class



- ◆ Following table explains the key methods of the `Socket` class:

Method	Description
<code>connect(SocketAddress host, int timeout)</code>	Connects the client socket to the server socket. This method is required if a <code>Socket</code> object is created without initializing it with a connection to the server.
<code>getInputStream()</code>	Returns an <code>InputStream</code> object of the <code>Socket</code> . Both clients and servers use the <code>getInputStream()</code> method to receive data.
<code>getOutputStream()</code>	Returns an <code>OutputStream</code> object of the <code>Socket</code> . Both clients and servers use the <code>getOutputStream()</code> method to send data.
<code>close()</code>	Closes the <code>Socket</code> connection.



- ◆ The `ServerSocket` class is used by servers to listen for incoming connections from clients.
- ◆ The following table explains the key methods of the `Socket` class:

Method	Description
<code>bind(SocketAddress endPoint)</code>	Binds a <code>ServerSocket</code> object to a specified IP address and port number that the <code>SocketAddress</code> parameter represents.
<code>accept()</code>	Listens for a connection to be made to this socket and accepts it. The <code>accept()</code> method blocks until either a client connects to the server on the specified port or the socket times out.
<code>getLocalPort()</code>	Returns the port number as an <code>int</code> value that a <code>ServerSocket</code> object is listening to.
<code>setSoTimeout(int timeout)</code>	Sets a timeout in milliseconds after which a <code>ServerSocket</code> object stops accepting client connections.
<code>isClosed()</code>	Returns a boolean value to indicate whether or not a <code>ServerSocket</code> object is closed.



- ◆ The `InetAddress` class represents an Internet address to perform a Domain Name System (DNS) look-up and reverse look-up.
- ◆ Following table explains the important methods of the `InetAddress` class:

Method	Description
<code>getAddress()</code>	Returns the IP address of the <code>InetAddress</code> object as a <code>byte[]</code>
<code>getByName(String host)</code>	Returns the IP address of the host passed as parameter as an <code>InetAddress</code> object
<code>getHostName()</code>	Returns the host name of the <code>InetAddress</code> object
<code>getAllByName(String host)</code>	Returns an array of its IP addresses for the host passed as parameter
<code>isReachable(int timeout)</code>	Returns a boolean to indicate whether or not the IP address represented by <code>InetAddress</code> is reachable



- ◆ The `URL` class represents a Uniform Resource Locator (URL) that points to a resource on the Web.
- ◆ Following table explains the important methods of the `URL` class:

Method	Description
<code>getPath()</code>	Returns the path of the URL as a String
<code>getQuery()</code>	Returns the query part of the URL as a String
<code>getPort()</code>	Returns the port of the URL as an int value
<code>getDefaultPort()</code>	Returns the default port for the protocol of the URL as an int value
<code>getProtocol()</code>	Returns the protocol of the URL as a String
<code>getHost()</code>	Returns the host of the URL as a String
<code>getFile()</code>	Returns the filename of the URL as a String
<code>openConnection()</code>	Opens a connection to the URL and returns a <code>URLConnection</code> object

# URLConnection Class [1-2]



- ◆ The `openConnection()` method of the `URL` class returns an implementation of the `URLConnection` class.
- ◆ Following table explains the important methods of the `URLConnection` class:

Method	Description
<code>getURL ()</code>	Returns the URL that the <code>URLConnection</code> object is connected to as a <code>URL</code> object
<code>setDoInput (boolean input)</code>	Accepts a boolean value to indicate whether the <code>URLConnection</code> object will be used for input. The default value is <code>true</code>
<code>setDoOutput (boolean output)</code>	Accepts a boolean value to indicate whether the <code>URLConnection</code> object will be used for output. The default value is <code>false</code>
<code>getInputStream ()</code>	Returns the input stream of the <code>URLConnection</code> as an <code>InputStream</code> object. This method is called to read from a URL
<code>getOutputStream ()</code>	Returns the output stream of the <code>URLConnection</code> as a <code>OutputStream</code> object. This method is called to write to a URL
<code>getContent ()</code>	Returns an Object of the contents of the <code>URLConnection</code>

# URLConnection Class [2-2]

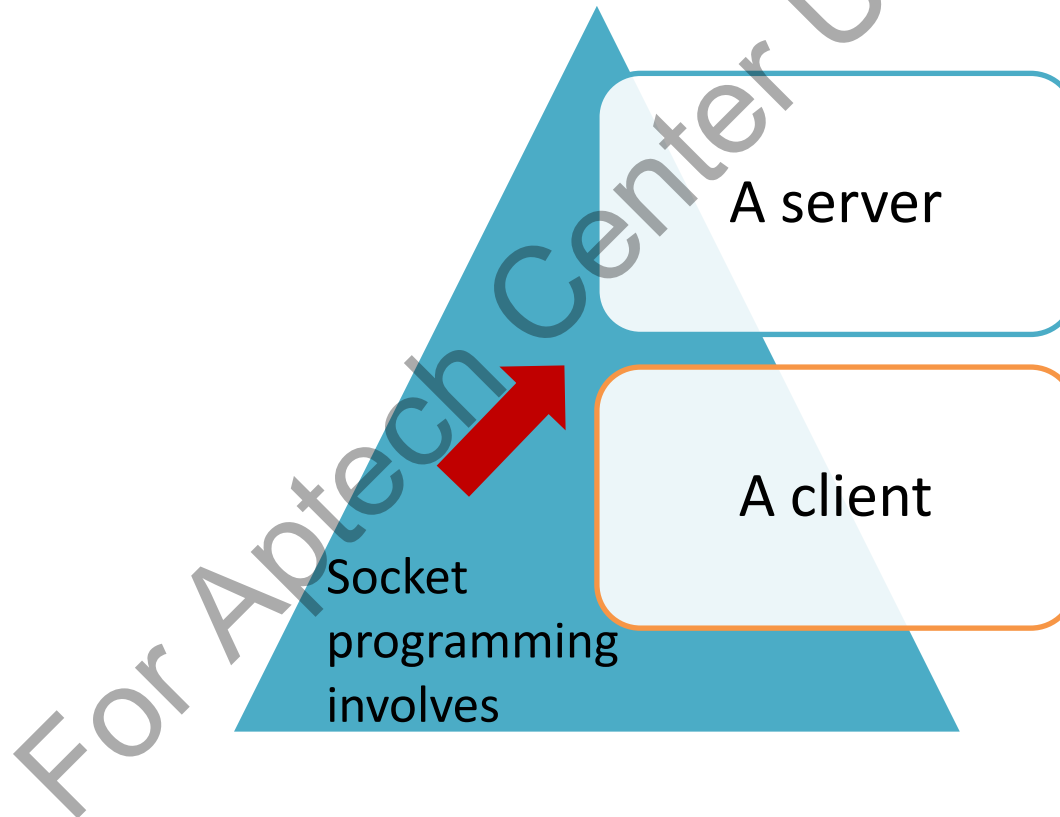


Method	Description
<code>getContentEncoding()</code>	Returns the content-encoding header field of the of the <code>URLConnection</code> as a <code>String</code> object
<code>getContentLength()</code>	Returns the content-length header field of the of the <code>URLConnection</code> as an <code>int</code> value
<code>getContentType()</code>	Returns the content-type header field of the of the <code>URLConnection</code> as a <code>String</code> object
<code>getLastModified()</code>	Returns the last-modified header field as an <code>int</code> value
<code>getExpiration()</code>	Returns the expires header field as a <code>long</code> value
<code>getIfModifiedSince()</code>	Returns the <code>ifModifiedSince</code> field of the <code>URLConnection</code> object as a <code>long</code> value

For Aptech Center Use Only



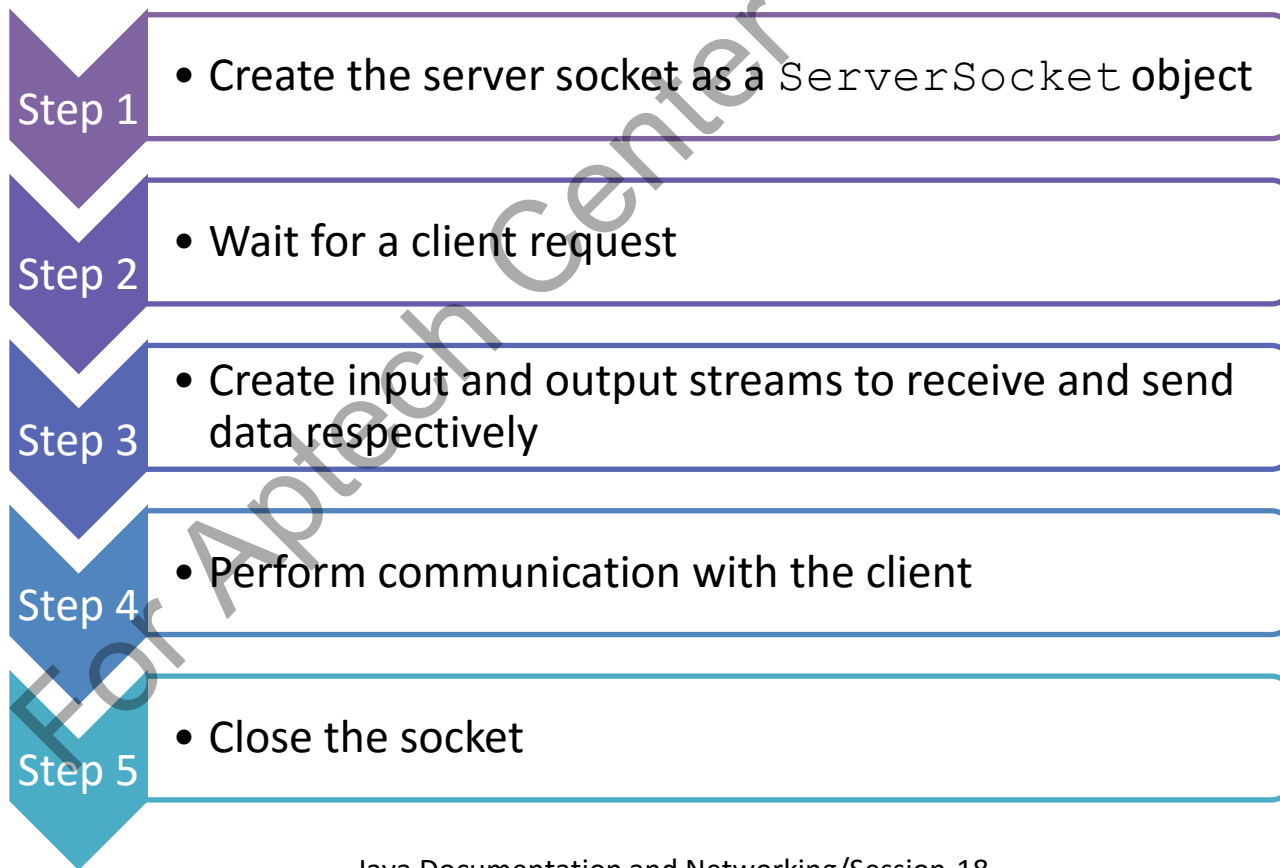
- ◆ The `openConnection()` method of the `URL` class returns an implementation of the `URLConnection` class.







- ◆ In a client/server programs that use TCP/IP:
  - ◆ A server is created to listen for client connections
  - ◆ Then, a client is created to connect with the server and exchange data packages.
- ◆ Following are the steps to create a server class:





- ◆ Following code snippet demonstrates the use of `ServerSocket` to create a server:

## Code Snippet

```
package com.io.demo;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketTimeoutException;
public class Server extends Thread {
    private ServerSocket serverSocket;
    public Server(int port) throws IOException {
        serverSocket = new ServerSocket(port);
    }
    public void run() {
```



## Code Snippet

```
while (true) {  
    try {  
        System.out.println("Listening for  
        client message on port " +  
        serverSocket.getLocalPort());  
        Socket =  
        serverSocket.accept();  
        DataInputStream in = new  
        DataInputStream(  
        socket.getInputStream());  
        DataOutputStream out = new  
        DataOutputStream(socket.  
        getOutputStream());  
        out.writeUTF("Hello from server.");  
    }  
    catch (SocketTimeoutException stException)  
    {  
    }
```



## Code Snippet

```
sTException.printStackTrace();  
}  
catch (IOException ioException) {  
    ioException.printStackTrace();  
} finally {  
    try {  
        serverSocket.close();  
    }  
    catch (IOException ioException) {  
        ioException.printStackTrace();  
    }  
}  
}  
}  
}  
public static void main(String[] args) {
```



## Code Snippet

```
public static void main(String[] args) {  
    try {  
        Thread = new Server(6060);  
        thread.start();  
    }  
    catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

For Apteck Center Use Only



- ◆ Following figure displays the output of the server:



# Creating a Client [1-3]



- ◆ Following are the steps to create a client:

Step 1

- Create a socket as a Socket object

Step 2

- Create input and output streams to receive and send data respectively

Step 3

- Perform communication with the server

Step 4

- Close the socket



- ◆ Following code snippet demonstrates the use of the Socket class to create a client:

## Code Snippet

```
package com.io.demo;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;

public class Client {
    public static void main(String[] args) {
        try {
            Socket clientSocket = new
            Socket("localhost", 6060);
            InputStream inFromServer ==
            clientSocket.getInputStream();
```



## Creating a Client [3-3]



```
        System.out.println("Message received from  
server: " + in.readUTF());  
        clientSocket.close();  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

- ◆ Following figure displays the output of the client:





- ◆ URL is an address of a resource in the Internet.
- ◆ Following code snippet demonstrates the use of the URL and URLConnection classes:

## Code Snippet

```
package com.io.demo;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;
public class Client {
    public static void main(String[] args) {
        try {
            Socket clientSocket = new
            Socket("localhost", 6060);
            InputStream inFromServer =
            clientSocket.getInputStream();
```



```
        DataInputStream in = new
        DataInputStream(inFromServer);
        System.out.println("Message received from
        server: " + in.readUTF());
        clientSocket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
```

- ◆ Following figure displays the output of the URLProcessingDemo class:

```
run:
null<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en-IN"><head><meta content="text/ht
function _gjh(){!_gjuc()&&window.google&&google.x&&google.x({id:"GJH"},function(){google.nav&&google.nav.gjh&&google
</style><style>body,td,a,p,h{font-family:arial,sans-serif}body{margin:0;overflow-y:scroll}#gog{padding:3px 8px 0}td
if (!iesg){document.f&&document.f.q.focus();document.gbqf&&document.gbqf.q.focus();}
}
})();</script><div id="mngb">    <div id=gbar><noabr><b class=gbl>Search</b> <a class=gbl href="http://www.google.co.
</script></div></body></html>

BUILD SUCCESSFUL (total time: 1 second)
```



- ◆ The Javadoc tool relies on documentation tags present in the source code to create API documentation.
- ◆ Javadoc can be generated using the Javadoc tool from the command line or the in-built Javadoc options of NetBeans.
- ◆ Classes and interfaces of the java.net package supports network programming.
- ◆ Socket programming over UDP is supported by the DatagramPacket and DatagramSocket classes.
- ◆ Socket programming over TCP is supported by the Socket and ServerSocket classes of the java.net package.
- ◆ URL processing can be done by the URL and URLConnection classes.

