Professional Programming in Java

Session: 18

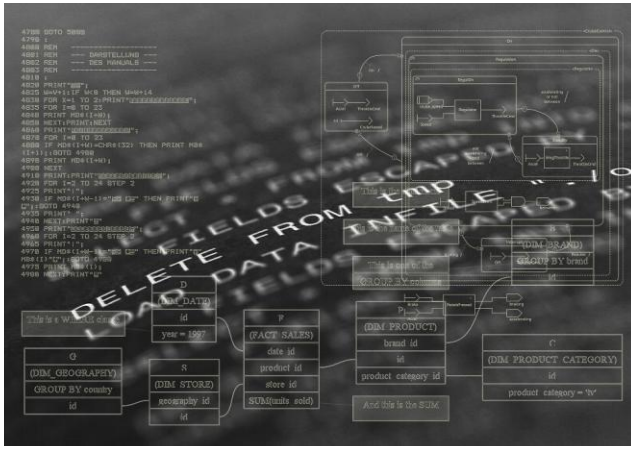
Java Documentation and Networking

Objectives

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

Javadoc Tool

- 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



Documentation in Java [1-2]

- 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

Documentation in Java [2-2]

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

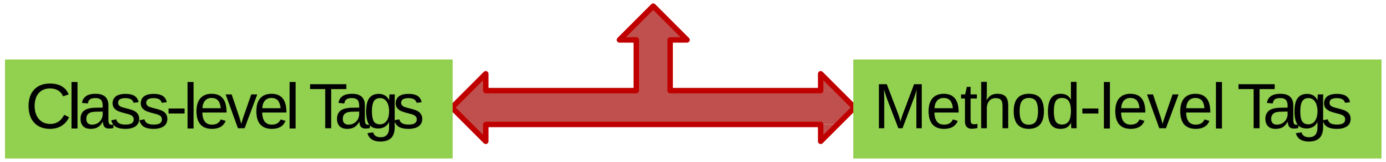
Generates API documentation in HTML

Can be found in the bin folder of the JDK path

Examples: Eclipse and NetBeans also have support for Javadoc

Javadoc Tags [1-14]

- The Javadoc tags can be primarily divided into:



Class-level Tags

Tag	Description
`@author`	Inserts the author name ofthe class
`{@code}`	Inserts text in codeformat
`@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

Javadoc Tags [2-14]

- 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.

Javadoc Tags [3-14]

Method-level Tags

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

Javadoc Tags [4-14]

- Following code snippet demonstrates the use ofthe 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.

Javadoc Tags [5-14]

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

```
/**
  *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() { }
```

Javadoc Tags [6-14]

```
      /**
        *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 num1 and num2.
        */
      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.
       *@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));
      }
}
```

Javadoc Tags [7-14]

- The two ways to generate Java documentation are:

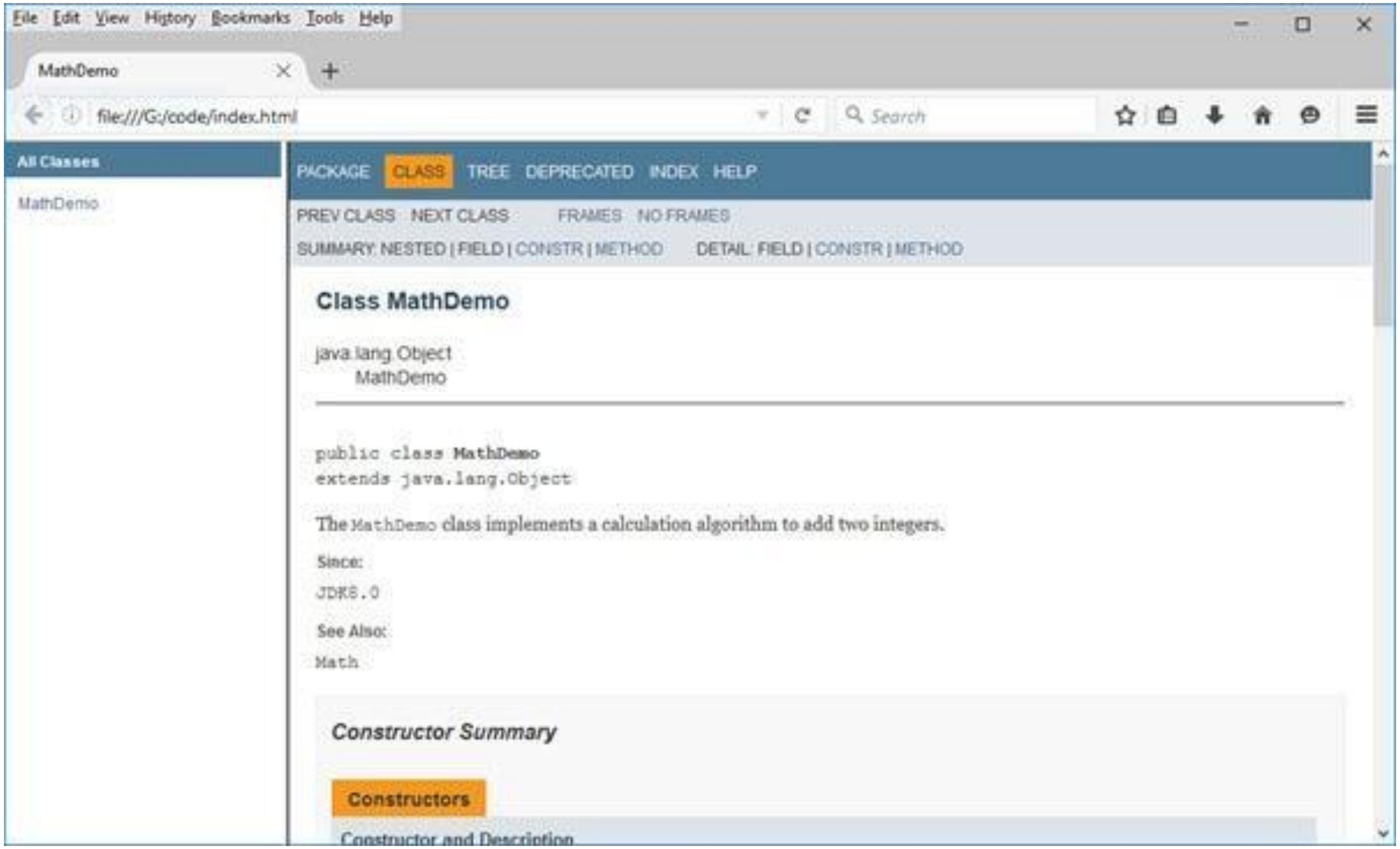
At the command prompt using the Javadoc tool or using the IDE option.

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.

Javadoc Tags [9-14]

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


File Edit View History Bookmarks Tools Help
MathDemo
file:///G:/code/index.html
Search
All Classes
MathDemo
PACKAGE CLASS TREE DEPRECATED INDEX HELP
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class MathDemo

java.lang.Object
    MathDemo

```
public class MathDemo
extends java.lang.Object
```

The MathDemo class implements a calculation algorithm to add two integers.

Since:

JDK8.0

See Also:

Math

Constructor Summary

Constructors

Constructor and Description

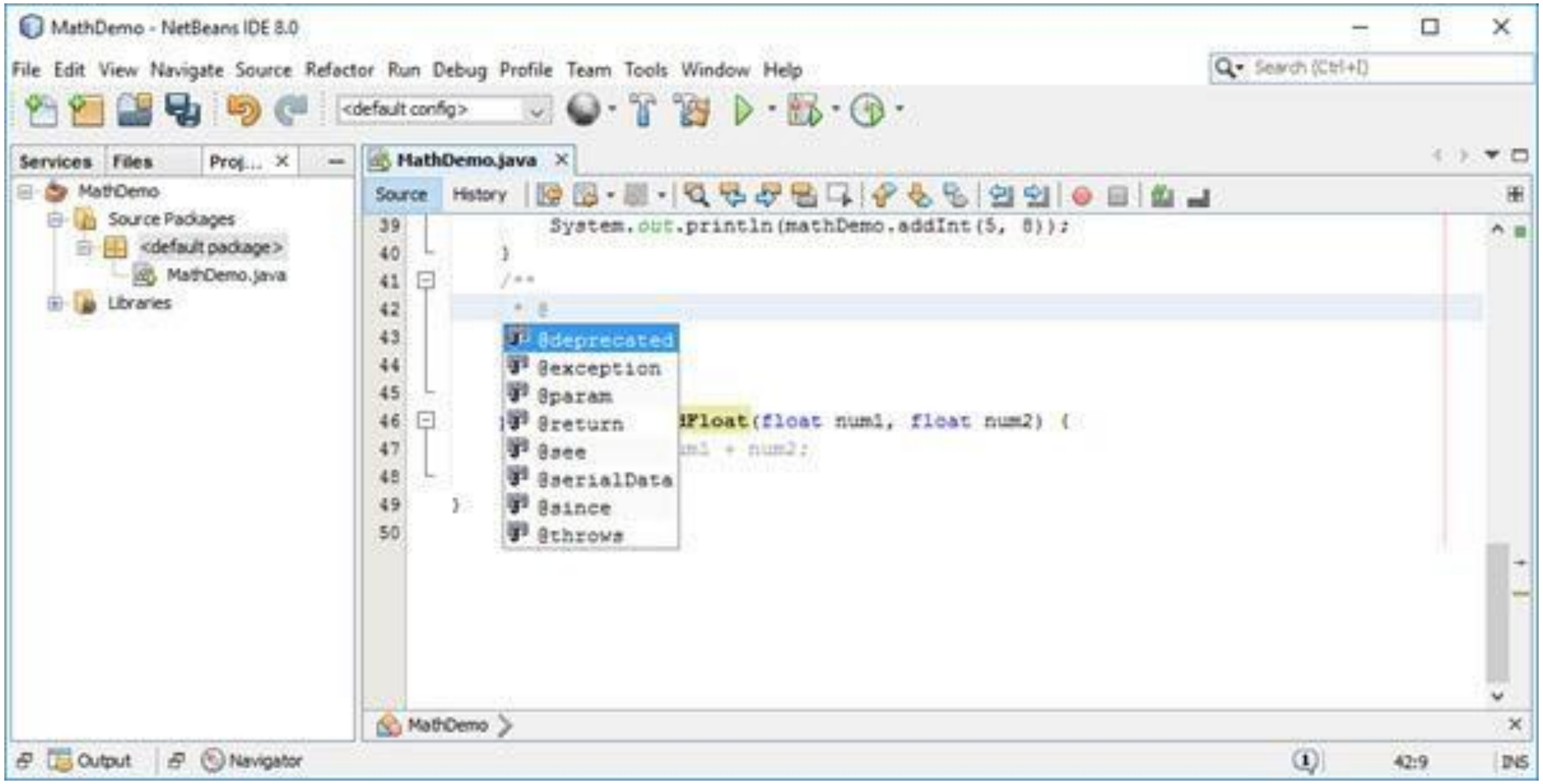
Javadoc Tags [10-14]

- 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

Javadoc Tags [11-14]

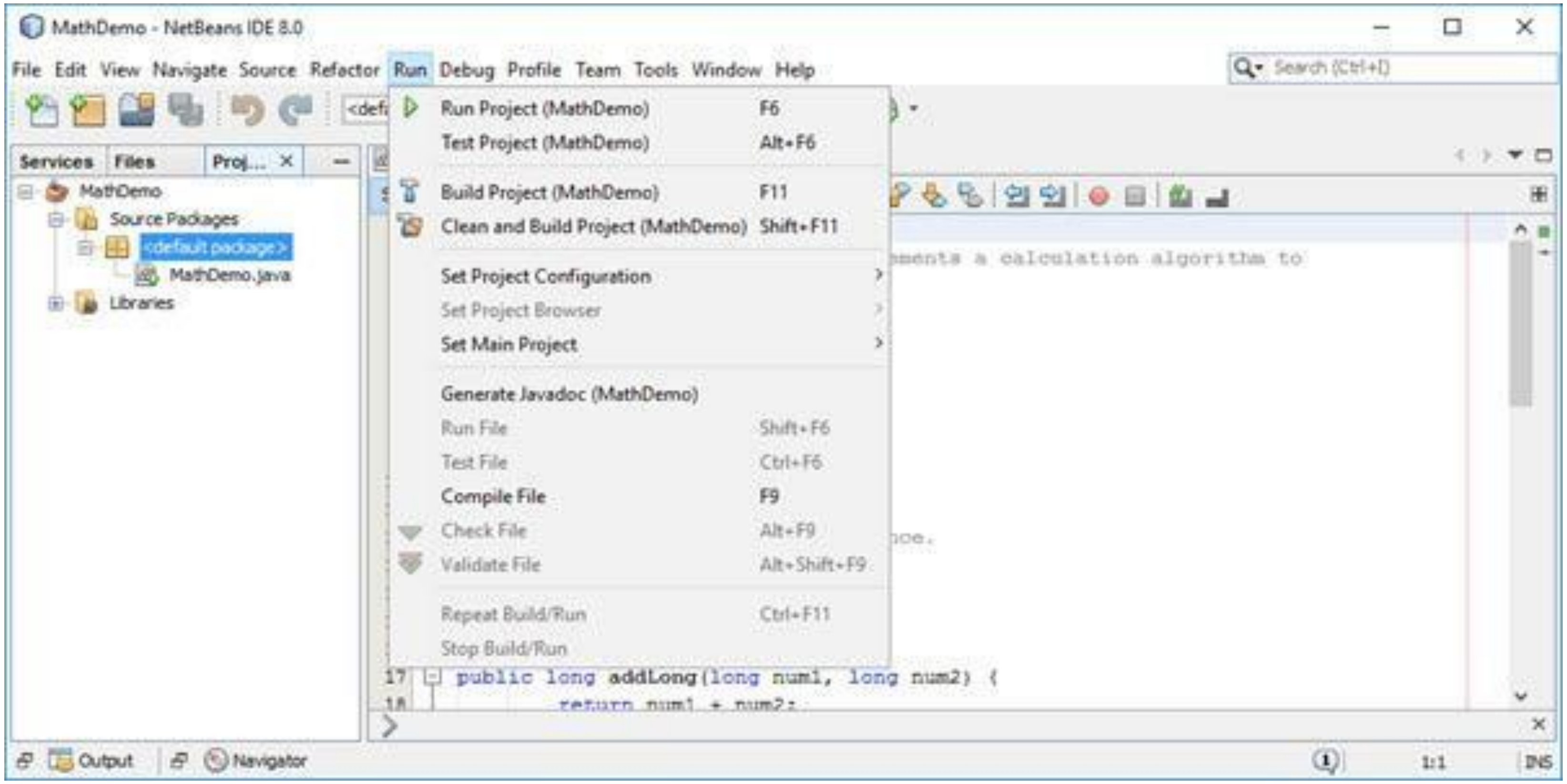
- Following figure displays the Javadoc tag code-completion pop-up:



```
        System.out.println(mathDemo.addInt(5, 8));
    }
    /**
     * @



    dFloat(float num1, float num2) {
    um1 + num2;

}
```

Javadoc Tags [12-14]

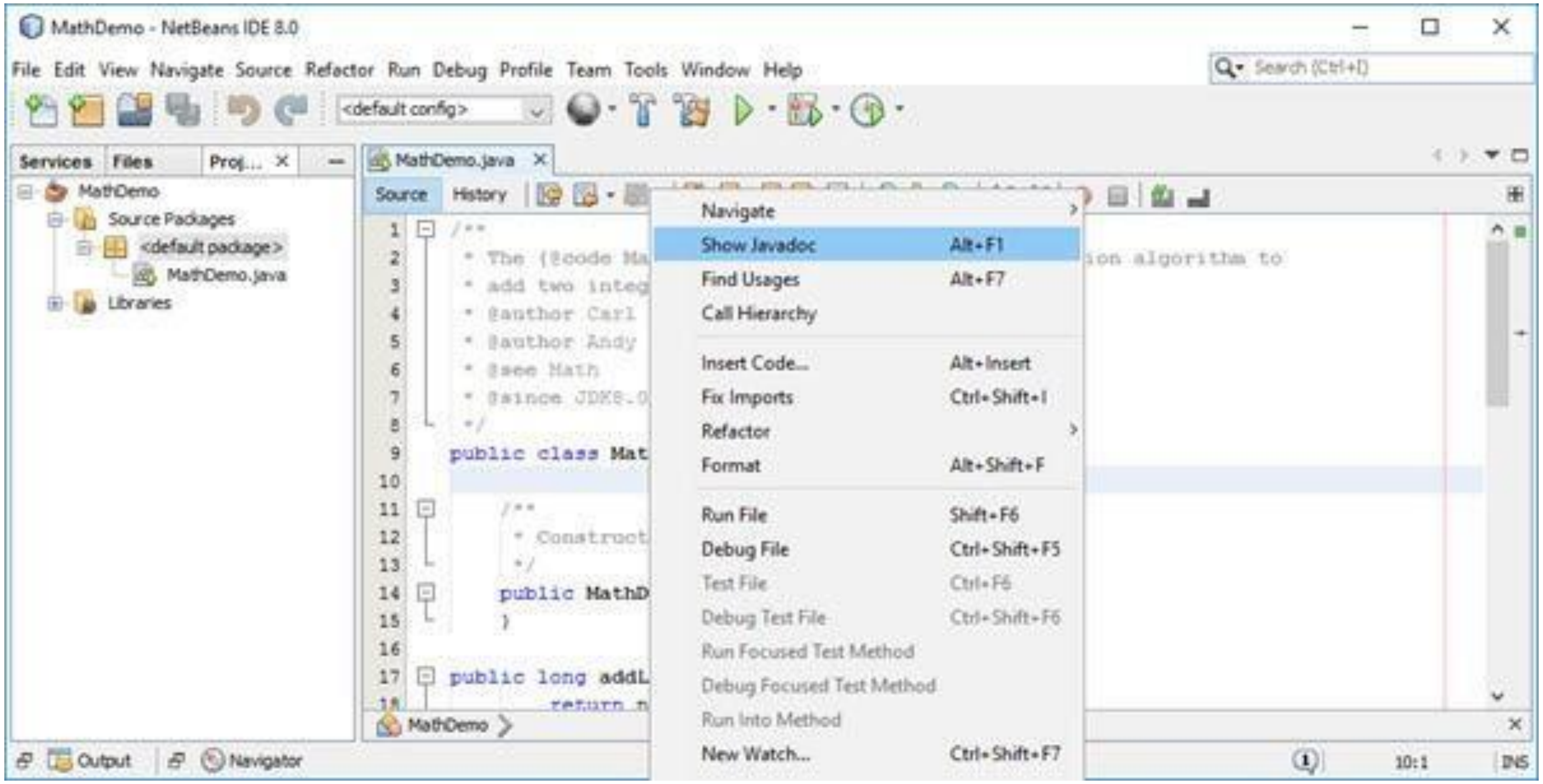
- Select Run → Generate Javadoc from the main menu of NetBeans to show the following figure that displays the Javadoc Documentation Generation in NetBeans:


MathDemo - NetBeans IDE 8.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Services Files Proj...
MathDemo
Source Packages
<default package>
MathDemo.java
Libraries
Run Project (MathDemo) F6
Test Project (MathDemo) Alt+F6
Build Project (MathDemo) F11
Clean and Build Project (MathDemo) Shift+F11
Set Project Configuration
Set Project Browser
Set Main Project
Generate Javadoc (MathDemo)
Run File Shift+F6
Test File Ctrl+F6
Compile File F9
Check File Alt+F9
Validate File Alt+Shift+F9
Repeat Build/Run Ctrl+F11
Stop Build/Run
Output
Navigator
1:1
INS

```
ments a calculation algorithm to
nce.
public long addLong(long num1, long num2) {
    return num1 + num2;
```

Javadoc Tags [13-14]

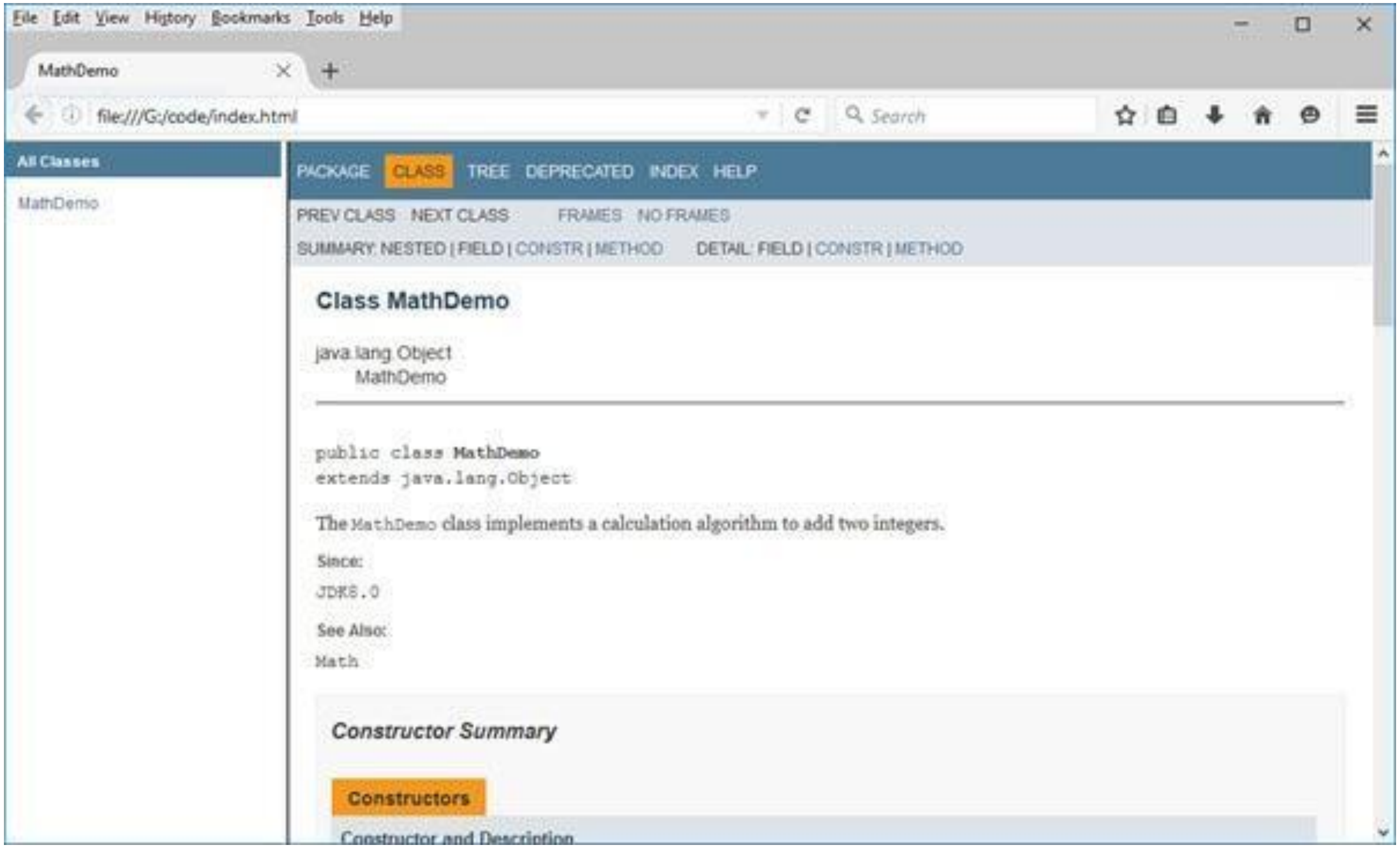
- Select **Show Javadoc** from the contextual menu to show the following figure that displays Viewing Javadoc in NetBeans:



```
/**
 * The {@code Ma                    ion algorithm to
 * add two integ
 * @author Carl
 * @author Andy
 * @see Math
 * @since JDK8.0
 */
public class Mat

    /**
     * Construct
     */
    public MathD
    }

public long addL
        return n
```

Javadoc Tags [14-14]

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


File Edit View History Bookmarks Tools Help
MathDemo
file:///G:/code/index.html
Search
All Classes
MathDemo
PACKAGE CLASS TREE DEPRECATED INDEX HELP
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class MathDemo

java.lang.Object
    MathDemo

```
public class MathDemo
extends java.lang.Object
```

The `MathDemo` class implements a calculation algorithm to add two integers.

Since:

`JDK8.0`

See Also:

`Math`

Constructor Summary

Constructors

Constructor and Description

java.net Package

- 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:

DatagramPacket

DatagramSocket

Socket

ServerSocket

InetAddress

URL

URLConnection

DatagramPacket Class [1-2]

- The DatagramPacket class:

  - Represents datagram packets for communication using the User Datagram Protocol (UDP) protocol
  - Contains information about itself and its destination
  - Are routed differently

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

DatagramPacket Class [2-2]

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

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

DatagramSocket Class

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

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

Socket Class [1-3]

- The Socket class:

Represents the socket used by both the clientand server for communicating

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

- The TCP protocol:

Maintains a connection between endpoints that Socketobjects represents

Guarantees both because both the client and serversockets remains connected

Socket Class [2-3]

- 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

Socket Class [3-3]

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

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

ServerSocket Class

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

InetAddress Class

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

URL Class

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

URLConnection Class [1-2]

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

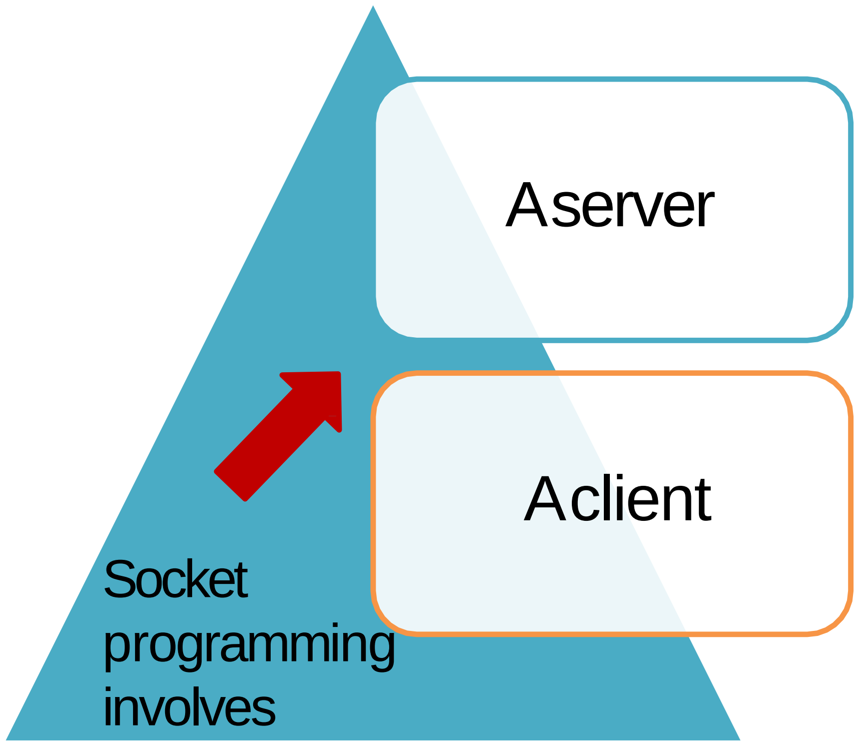
Method	Description
`getURL()`	Returns the URLthat the `URLConnection` object is connected to as a URL object
`setDoInput(boolean input)`	Accepts a boolean value to indicate whether the `URLConnection` object will be used for input. The default value is true
`setDoOutput(boolean output)`	Accepts a boolean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

URLConnection Class [2-2]

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

Socket Programming and URL Processing

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



Client/Server Program [1-6]

- 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:

Step 1
- Create the server socket as a `ServerSocket` object

Step 2
- Wait for a client request

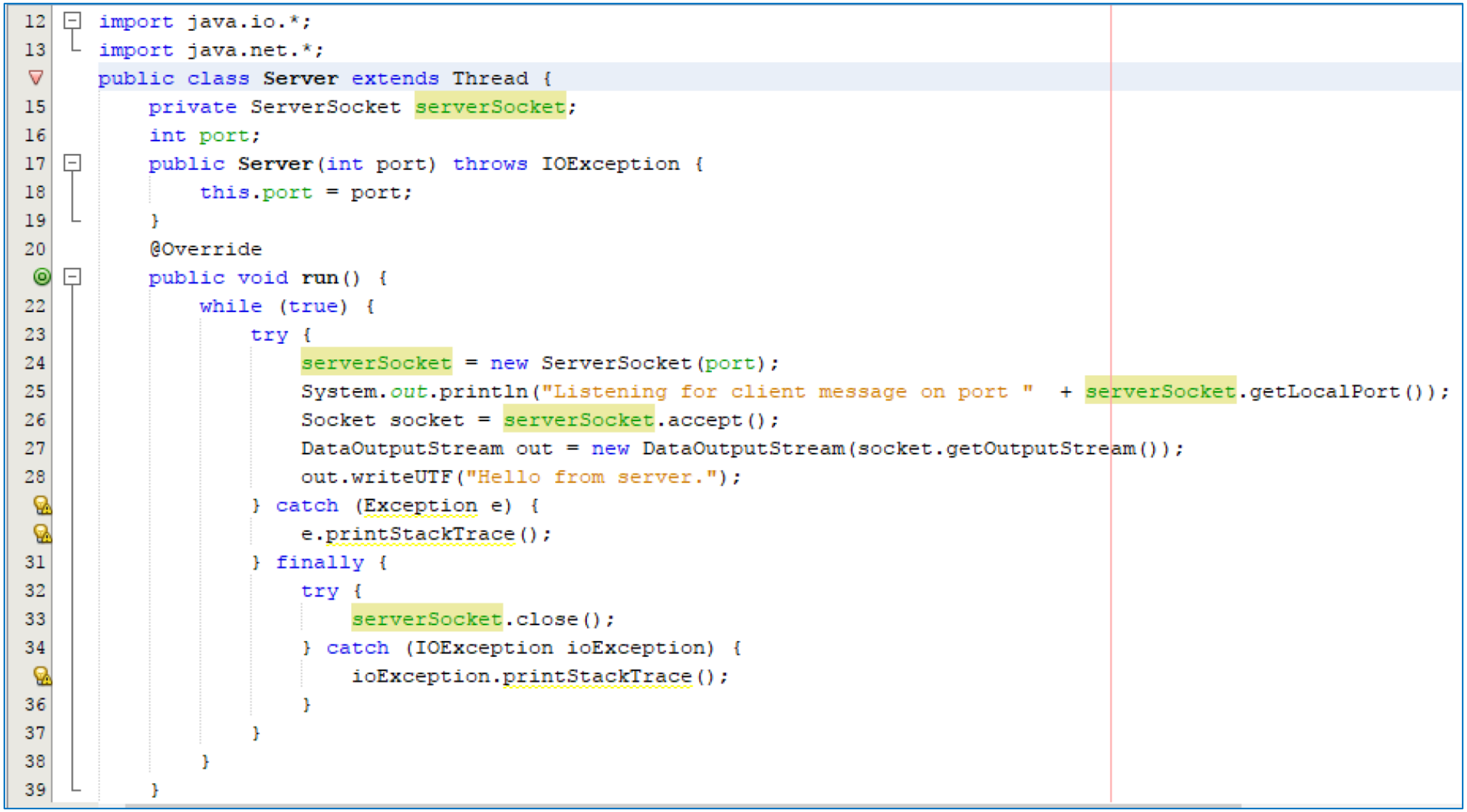
Step 3
- Create input and output streams to receive and send data respectively

Step 4
- Perform communication with the client

Step 5
- Close the socket

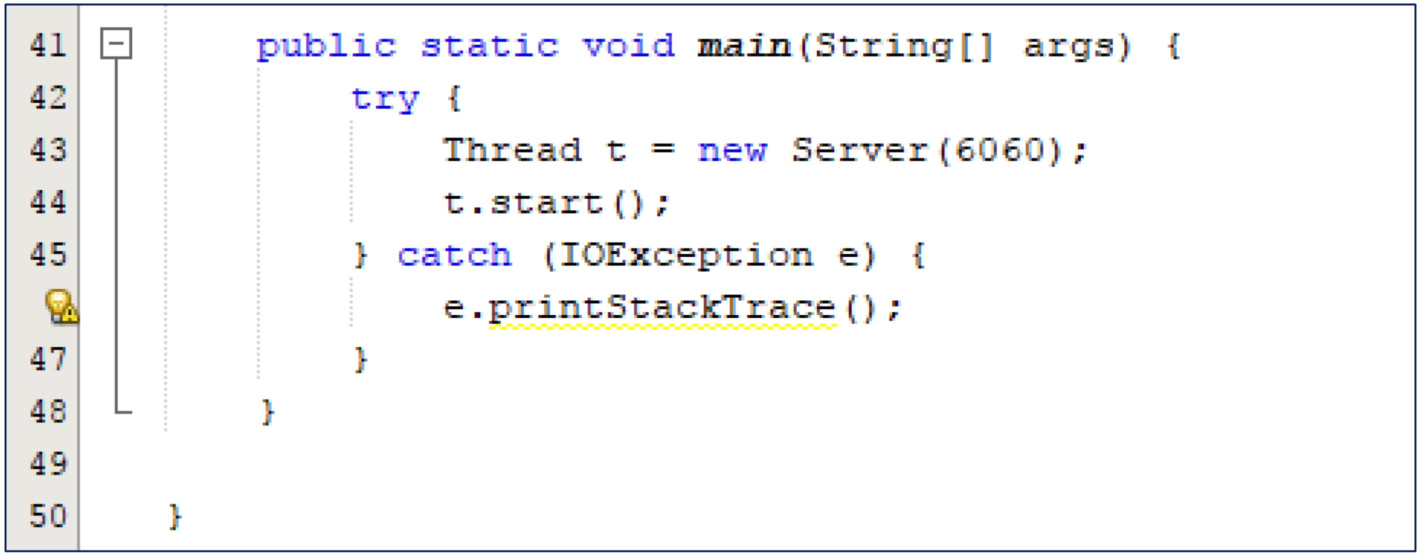
Client/Server Program [2-6]

- Following code snippet demonstrates using ServerSocket to create a server:

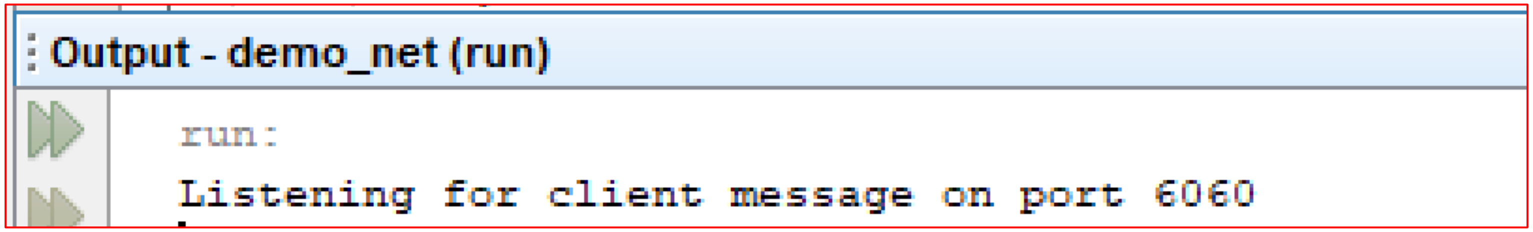


```java
import java.io.*;
import java.net.*;
public class Server extends Thread {
    private ServerSocket serverSocket;
    int port;
    public Server(int port) throws IOException {
        this.port = port;
    }
    @Override
    public void run() {
        while (true) {
            try {
                serverSocket = new ServerSocket(port);
                System.out.println("Listening for client message on port "  + serverSocket.getLocalPort());
                Socket socket = serverSocket.accept();
                DataOutputStream out = new DataOutputStream(socket.getOutputStream());
                out.writeUTF("Hello from server.");
            } catch (Exception e) {
                e.printStackTrace();
            } finally {
                try {
                    serverSocket.close();
                } catch (IOException ioException) {
                    ioException.printStackTrace();
                }
            }
        }
    }
```

Client/Server Program [3-6]



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

}
```

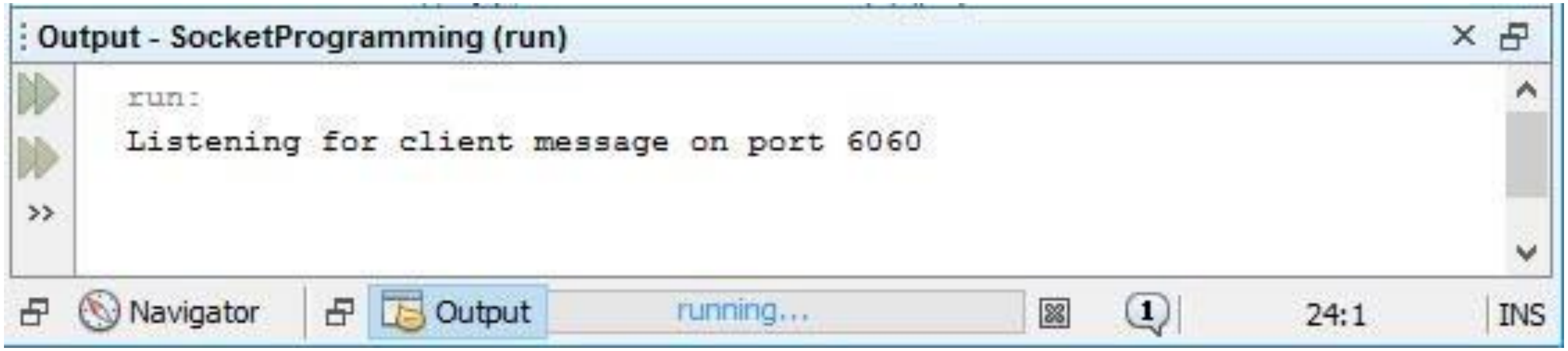


Output - demo_net (run)

```
run:
Listening for client message on port 6060
```

Client/Server Program [6-6]

- Following figure displays the output of the server:



```
run:
Listening for client message on port 6060
```

Creating a Client [1-3]

- Following are the steps to create a client:

Step 1
- Create a socket as a Socket object

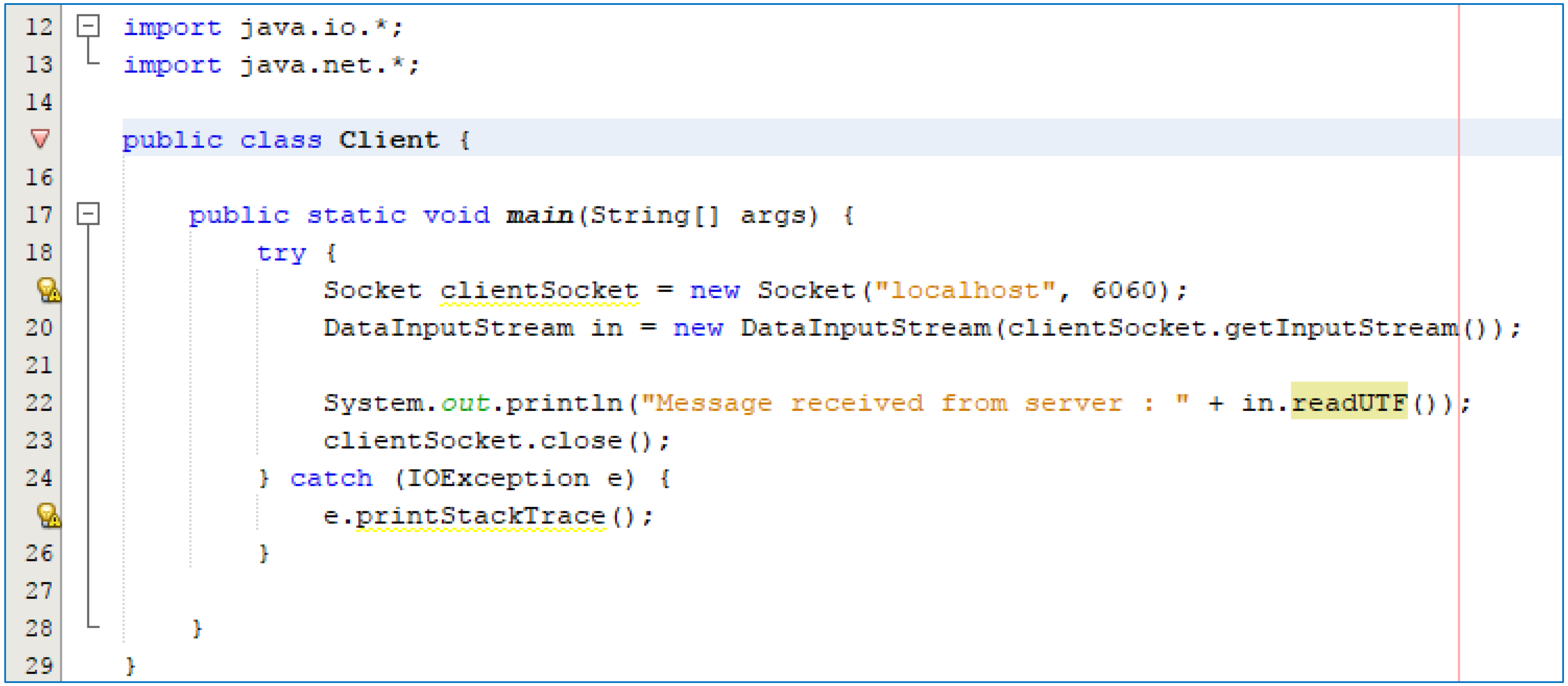
Step 2
- Create input and output streams to receive and send data respectively

Step 3
- Perform communication with the server

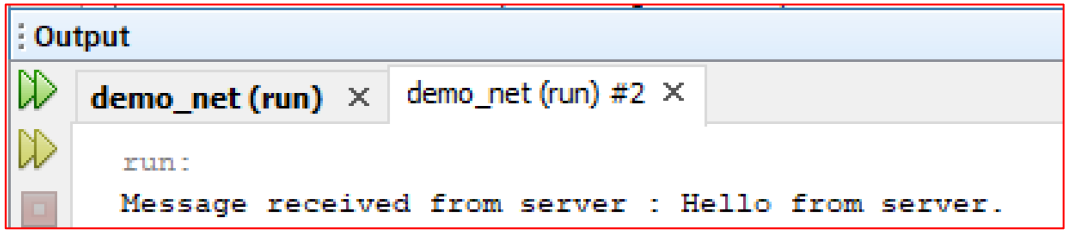
Step 4
- Close the socket

Creating a Client [2-3]

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



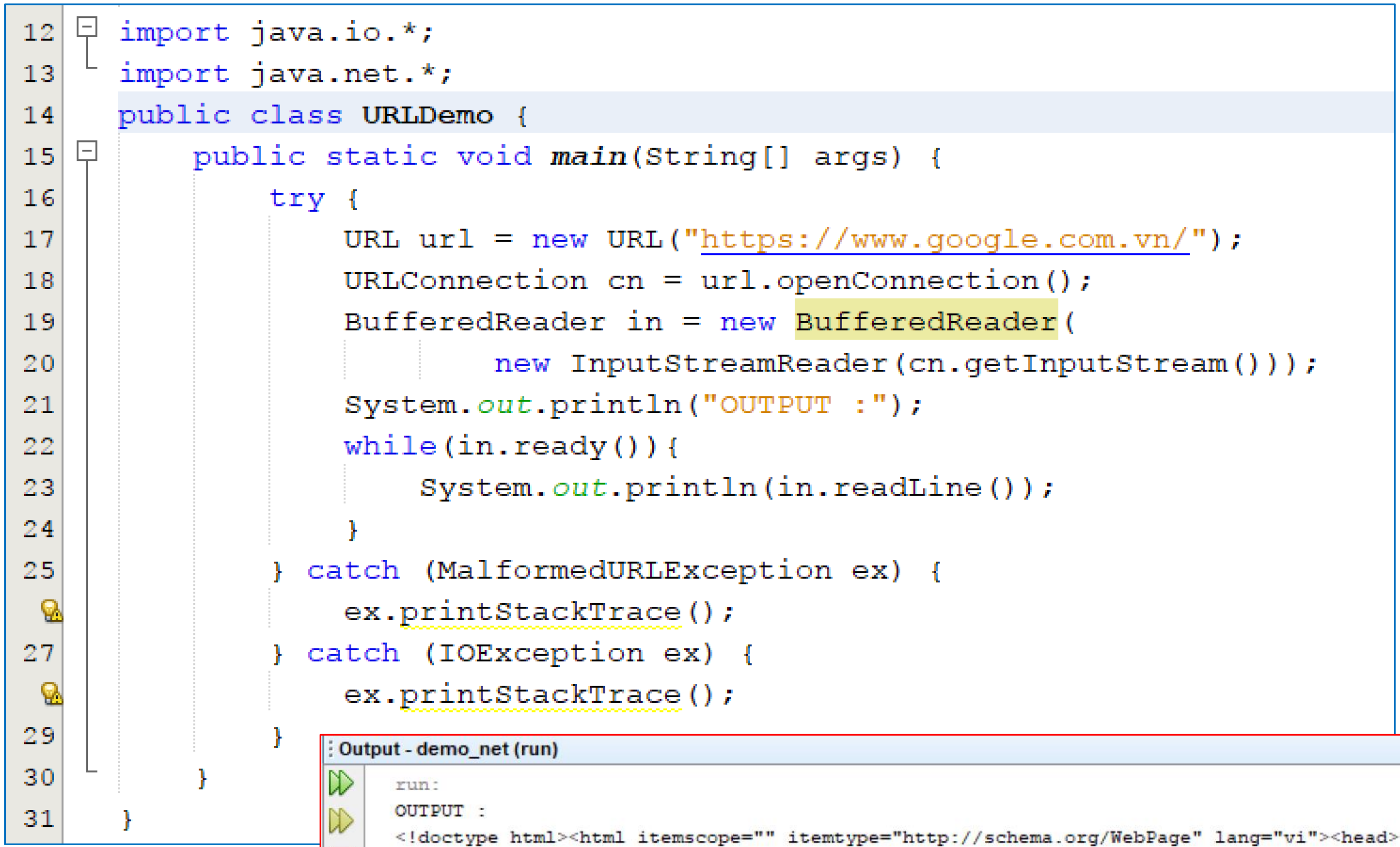
```
import java.io.*;
import java.net.*;

public class Client {

    public static void main(String[] args) {
        try {
            Socket clientSocket = new Socket("localhost", 6060);
            DataInputStream in = new DataInputStream(clientSocket.getInputStream());

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

    }
}
```



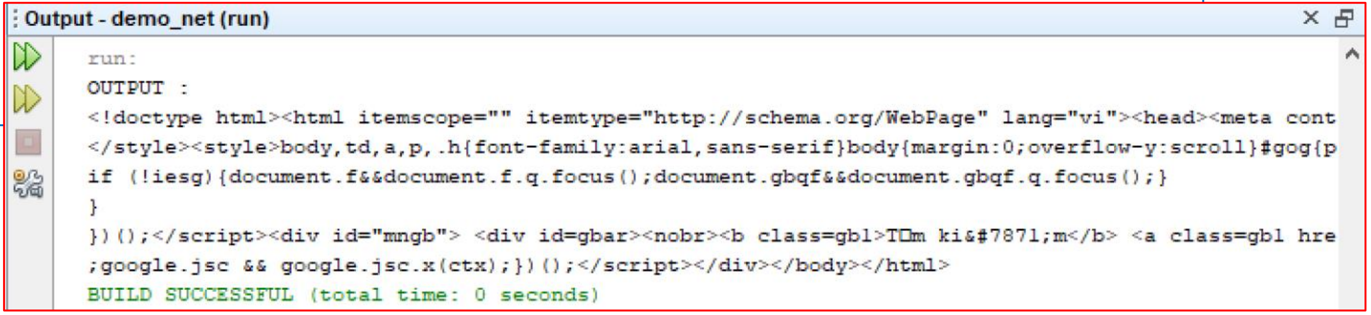
```
run:
Message received from server : Hello from server.
```

URL Processing [1-2]

- Following code snippet demonstrates the use of the `URL` and `URLConnection` classes:



```
import java.io.*;
import java.net.*;
public class URLDemo {
    public static void main(String[] args) {
        try {
            URL url = new URL("https://www.google.com.vn/");
            URLConnection cn = url.openConnection();
            BufferedReader in = new BufferedReader(
                    new InputStreamReader(cn.getInputStream()));
            System.out.println("OUTPUT :");
            while(in.ready()){
                System.out.println(in.readLine());
            }
        } catch (MalformedURLException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```



```
run:
OUTPUT :
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="vi"><head><meta cont
</style><style>body,td,a,p,.h{font-family:arial,sans-serif}body{margin:0;overflow-y:scroll}#gog{p
if (!iesg){document.f&&document.f.q.focus();document.gbqf&&document.gbqf.q.focus();}
}
})();</script><div id="mngb"> <div id=gbar><nobr><b class=gbl>T□m ki&#7871;m</b> <a class=gbl hre
;google.jsc && google.jsc.x(ctx);})();</script></div></body></html>
BUILD SUCCESSFUL (total time: 0 seconds)
```

Summary

- 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.