14 TUESDAY
Mar 2023 Week 11 / 073-292

Introduction to Advanced Java

Advance → forward movement or a development or improvement.

Java → Core Java (J2SE)
Java → Advanced Java (JEE)

* JEE → Java Enterprise Edition
  ↳ It is a set of specifications, APIs, and technologies designed to develop scalable distributed, and enterprise-level applications. It includes components like Servlets, JSP (Java Server Pages), EJB (Enterprise JavaBeans), JMS (Java Message Service), and more.

* J2SE (Java Platform, Standard Edition).
  ↳ Also known as Cor Java, this is the most basic and standard version of Java.
  ↳ It consists of a wide variety of general purpose APIs (like java.lang, java.util) as well as many special purpose APIs.
  ↳ It is mainly used to create applications for Desktop environment.

* J2ME (Java Platform, Micro Edition)
  ↳ This version of Java is mainly concentrated for the applications running on embedded systems, mobiles and small devices (which was a constraint before its development)

The best defense is a good offence

→ constraints included limited processing power, battery limitations, small display etc.

08.00 → J2ME uses many libraries and API's of J2SE, as well as, many of its own.

09.00 → The basic aim of this editions was to work on mobiles, wireless devices, set top boxes etc

10.00 E.g → Old Nokia phones, which used Symbian OS, uses this technology.

11.00

- Why advance Java?

12.00 (i) It simplifies the complexity of a building n-tier application.

01.00 (ii) Standardizes and API between components and application server container

02.00 (iii) JEE application server and containers provides the framework services

03.00

- Benefits of Advance Java

04.00 (i) JEE (advance Java) provides libraries to understand the concept of Client-Server architecture for web-based applications

05.00 (ii) We can work with web and application servers such as Apache Tomcat and Glassfish. Using these servers, we can understand the working of HTTP protocol. It can't be done in core Java.

06.00 (iii) It is important to understand advance java if we are dealing with trading technologies like Hadoop, cloud-native and Data Science.

EVE (iv) It provides a set of services, API and protocols, that provides the functionality which is necessary for developing multitiered applications, web based application

Notes

Sympathy is the key that fits the lock of any heart

MARCH 2023

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

(iv) There is a number of advance Java frameworks like Spring, Hibernate, Struts, that enables us to develop secure transaction-based web applications such as banking application, inventory mngmt. application.

- Difference between Core Java & Advance Java.

~~Used for.~~

Criteria	Core Java	Advance Java.
(i) Used for.	- It is used to develop general purpose application.	- It is used to develop web-based application
(ii) Purpose.	- It does not deal with database, socket programming, etc.	- It deals with socket programming, DOM, and networking applications
(iii) Architecture.	- It is a single-tier architecture	- It is a multi-tier architecture.
(iv) Editions	- It is a Java Standard Edition	- It is a Java Enterprise Edition.
(v) Package	- It provides java.lang.* package	- It provides java.servlet.* package.

Notes

Applet Programming

* Java Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

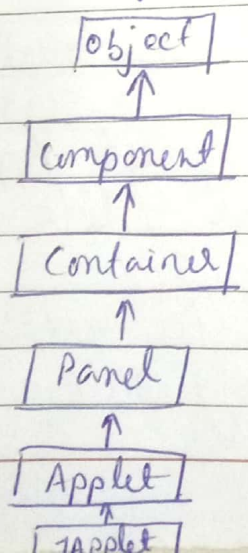
* Advantages of Applet

* It works at client side, so less response time.
* Secured.
* It can be executed by browsers running under many platforms, including Linux, Windows, Mac OS etc.

* Drawback of Applet

* Plug in is required at client browser to execute applet.

* Hierarchy of Applet



As displayed in the above diagram, Applet class extends Panel. Panel class extends Container which is the subclass of Component.

18 SATURDAY
Mar 2023 Week 11 / 077-288

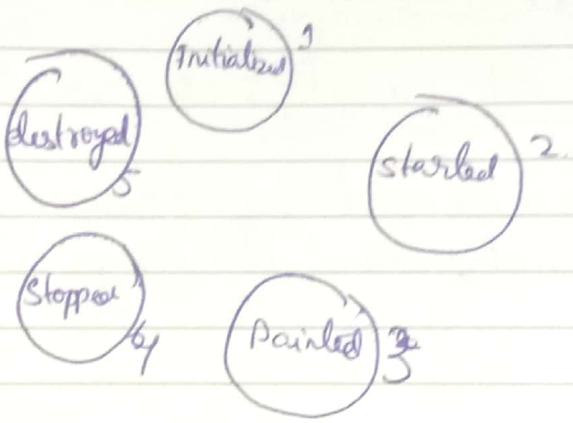
MARCH 2023
S M T W T F S S M T W T F S
1 2 3 4 5 6 7 8 9 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31

* Lifecycle of Java Applet

08.00 (1) Applet is initialized
(2) Applet is started
09.00 (3) Applet is painted
(4) Applet is stopped
10.00 (5) Applet is destroyed.

11.00 Applet Lifecycle

12.00



04.00 * Lifecycle methods for Applet:

05.00 The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle
06.00 methods for an applet

Sunday 19

EVE ↳ java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

Notes

(i) public void init(): is used to initialized the Applet. It is invoked only once.

(2) public void start(); is invoked after the init() method or browser is maximized. It is used to start the Applet.

(3) public void stop(); is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.

(4) public void destroy(); is used to destroy the Applet. It is invoked only once.

↳ java.awt.Component class

The Component class provides 1 life cycle method of applet.

(i) public void paint(Graphics g): is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

* Who is responsible to manage the life cycle of an applet?
Java Plug-in software

* How to run an Applet?

There are two ways to run an applet
(1) By html file.
(2) By appletViewer tool (for testing purpose)

* Simple example of Applet by html file:
To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file.

MARCH 2023

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Now click the html file

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{

public void paint(Graphics g){
g.drawString("welcome",150,150);
}

}
```

NOTE: class must be public because its object is created by Java Plugin software that resides on the browser.

myapplet.html

```
<html>
<body>
<applet code="First.class" width="300" height="300">
</applet>
</body>
</html>
```

* Simple example of Applet by appletviewer tool:

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only

Notes

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{

public void paint(Graphics g){
g.drawString("welcome to applet",150,150);
}

}
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
```

Execution

```
c:\>javac First.java
e:\>appletviewer First.java
```

Notes

25 SATURDAY
Mar 2023 Week 12 / 084-281

MARCH 2023

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Connecting to a Server

08.00 → Connecting to web servers is a fundamental skill for software developers in today's interconnected
09.00 world.
→ Knowing how to interact with web servers programmatically is essential, whether you're building a web
10.00 appl. that needs to fetch data from external sources, testing a web service, or even performing web scraping
11.00 for research.
12.00

• steps for getting Connected to the Web Server

01.00

Step 1: → Define the URL

02.00 Before we can connect to a web server, we need to specify the URL of the server we want to interact
03.00 with.
→ This URL should point to the resource we want to
04.00 access, such as a website, API endpoint, or web service. (Here we've used "https://www.example.com", which can be
05.00 user's choice)

Step 2: Create a URL object

06.00 In Java, we use the URL class to work with Sunday 26
VE URLs. We create a URL object by passing the URL string as a parameter to the URL constructor. This object represents the URL we want to connect to.

otes

Step 3: Open a Connection.

To establish a connection to the web server, we use the 'HttpURLConnection' class, which provides HTTP-specific functionality. We call the 'openConnection()'

method on our URL object to open a connection. We then cast the returned connection to 'HttpURLConnection' for
08.00 HTTP-related operations.

09.00 **Step 4: Set the Request Method**

HTTP requests have different methods, such as
10.00 GET, POST, PUT, and DELETE. In this example we're making a GET request to retrieve data
11.00 from the server. We set the request method to "GET" using the 'setRequestMethod("GET")' method on the
12.00 'HttpURLConnection' object.

01.00 **Step 5: Get the Response Code.**

02.00 To check the status of our request, we obtain the HTTP response code using the 'getResponseCode()'
03.00 method. The response code indicates whether the request was successful or if there were any issues.
04.00

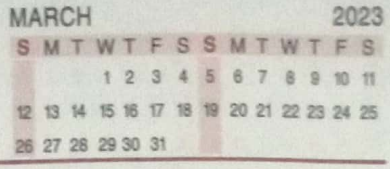
Step 6: Read and Display Response Content

05.00

To retrieve and display the content returned by
06.00 the web server, we create a 'BufferedReader' to read the response content line by line. We append
EVE each line to a 'StringBuilder' to construct the complete response content. Finally, we print the response content to the console.

Notes • **Program to Get Connected to Web server**

```
import java.io.BufferedReader;
import java.io.IOException;
```




```
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.nio.file.Files;
import java.nio.file.Path;

//Driver Class
public class WebServerConnection {
    //main function
    public static void main(String[] args)
    {
        try {
            // Step 1: Define the URL
            // Replace with your desired URL
            String url = "https://www.example.com";

            // Step 2: Create a URL object
            URL serverUrl = new URL(url);

            //Step 3: Open a connection
            HttpURLConnection connection
                = (HttpURLConnection)
                    serverUrl.openConnection();

            //Step 4: Set the request method to GET
            connection.setRequestMethod("GET");

            //Step 6: Read and display response content
            BufferedReader reader
                = new BufferedReader(new InputStreamReader(
                    connection.getInputStream()));
```

```
String line;
StringBuilder responseContent
        = new StringBuilder();

while ((line = reader.readLine()) != null) {
    responseContent.append(line);
}

reader.close();

//Defining the file name of the file
Path fileName = Path.of("... /temp.html");

//Writing into the file
Files.writeString(fileName, responseContent.toString());

//Print the response content
System.out.println(
        "Response Content:\n"
        + responseContent.toString());

}

catch (IOException e) {
    e.printStackTrace();
}
}
```

Notes

```
→ //Step 5: Get the HTTP response code
int responseCode = connection.getResponse();
System.out.println("Response Code:" + responseCode);
```

(i) java.io.BufferedReader Class in Java.

↳ Reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines. The buffer size may be specified, or the default size may be used

- How to take i/p using BufferedReader in Java.

```
BufferedReader input = new BufferedReader(new
    InputStreamReader(System.in));
```

- Once we've created a BufferedReader we can use its method readLine() to read one line of characters at a time from the keyboard and store it as a String object.

```
String inputString = input.readLine();
```

- Is BufferedReader faster than Scanner in Java?

BufferedReader is a bit faster than scanner since scanner does parsing of input data and BufferedReader simply reads sequence of characters

(ii) java.io.IOException is a checked exception in Java that indicates a problem while performing Input/Output (I/O) operations.

→ This usually ~~occurs~~ happens when a failure occurs while performing read, write or search operations in files or directories.

(iii) java.io.InputStreamReader class

It is a bridge from byte streams to character streams. It reads bytes and decodes them into characters using a specified charset

Notes

Sometimes the best gain is to lose

Declaration:

public class InputStreamReader
extends Reader.

(iv) java.net.HttpURLConnection Class

HttpURLConnection class is an abstract class directly extending from URLConnection class. It includes all the functionality of its parent class with additional HTTP-specific features. HttpsURLConnection is another class that is used for the more secured HTTPS protocol.

NOTE: InputStream gives you the bytes
InputStreamReader gives you already chars

(v) java.net.URL Class in Java.

It is an acronym of URL. It is a pointer to locate resource in WWW. A resource can be anything from a simple text file to any other like images, file directory etc.

URL has the following parts:

- Protocol → HTTP, HTTPS.
- Hostname → www.example.com
- Port Number → It is an optional attribute. If not specified then it returns -1. In the above case the port no. is 80.
- Resource name → It is the name of a resource located on the given server

Notes

http://www.example.com:80/index.html

07 FRIDAY
Apr 2023 Week 14 / 097-268

APRIL 2023
S M T W T F S S M T W T F S
1 2 3 4 5 6 7 8
9 10 11 12 13 14 15 16 17 18 19 20 21 22
23 24 25 26 27 28 29 30

→ abstract representation of file and directory pathnames.
File file = new File("baeldung/tutorial.txt");

(iv) java.nio.file package
Java Files is a part of this package

08.00

- Java Files class contains static methods that work on files and directories.
- This class is used for basic file operations like create, read, write, copy and delete the files or directories of the file system.

09.00

10.00

11.00

- The File class is an abstract representation of file and directory pathnames.

↳ Java IO and Java NIO

12.00

Java IO (Input/output) is used to perform read and write operations. The java.io package contains all the classes required for input and output operation

01.00

02.00

Java NIO (New IO) was introduced from JDK 4 to implement high speed IO operations.

03.00

04.00 (v) java.nio.file.Paths class contains static methods for converting path string or URI into path.

05.00

java.nio.file.Path

06.00

- ~~public static Path get (URI uri)~~ :

EVE

java.nio.file.Path class
↳ It delivers an entirely new API to work with I/O. Moreover, like the legacy File class, Path also creates an object that may be used to locate a file in a file system.

Notes

MAY

SATURDAY 08

Week 14 / 098-267 Apr 2023

Likewise, it can perform all the operations that can be done with the File class:

```
Path path =
    Paths.get ("baeldung/tutorial.txt");
```

Sunday 09 16

10 MONDAY
Apr 2023 Week 15 / 100-265

APRIL 2023

S	M	T	W	T	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

Socket Programming in Java

08.00 ~~Client-Side~~

→ One-way Client and Server setup where a Client connects, sends messages to the server and the server shows them using a socket connection.

09.00 → JAVA API networking package (java.net) takes care of all these things, making network programming very easy for programmers.

10.00

11.00

12.00 • Client-Side Programming

01.00 Establish a Socket connection.

→ To connect to another machine we need a socket connection. A socket connection means the two machines have information about each other's network location (IP Address) and TCP port. The java.net.Socket class represents a Socked. To open a socket:

02.00

```
Socket socket = new Socket("127.0.0.1", 5000)
```

03.00

04.00

05.00

→ The first argument – IP address of Server. (127.0.0.1 is the IP address of localhost, where code will run on the single stand-alone machine).

06.00 → The second argument – TCP Port. (Just a number representing which application to run on a server. For example, HTTP runs on port 80. Port number can be from 0 to 65535)

EVE

Notes

• → Communication

To communicate over a socket connection, streams are used to both input and output the data.

Closing the connection

The socket connection is closed explicitly once the message to the server is sent.

Java Implementation

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

public class Client {
  // initialize socket and i/p o/p streams
  private Socket socket = null;
  private DataInputStream input = null;
  private DataOutputStream out = null;

// constructor to put ip address and port
public Client (String address, int port)
 {
    // establish a connection
    try {
        socket = new Socket (address, port);
        System.out.println ("connected");

    // takes input from terminal
    input= new DataInputStream (System.in);

// sends output to the socket
    out = new DataOutputStream(
        socket.getOutputStream());
```

```
catch (UnknownHostException u) {
        System.out.println(u);
        return;
    }
    catch (IOException i) {
        System.out.println(i);
        return;
    }

    //string to read message from input
    String line = "";

    //keep reading until "Over" is input
    while (!line.equals("Over")) {
        try {
            line = input.readLine();
            out.writeUTF(line);
        }
        catch (IOException i) {
            System.out.println(i);
        }
    }

    //close the connection
    try {
        input.close();
        out.close();
        socket.close();
    }
```

Notes

```
catch (IOException i) {
    System.out.println(i);
}

public static void main (String args[])
{
    Client client = new Client("127.0.0.1", 5000);
}
}
```

- Server Programming

Establish a Socket Connection

To write a server application two sockets are needed.

→ A ServerSocket which waits for the client requests (when a client makes a new Socket())

→ A plain old socket to use for communication with the client.

Sunday 16

Communication

getOutputStream() method is used to send the output through the socket.

Close the Connection

Notes

After finishing, it is important to close the connection by closing the socket as well as input/output streams.

```
// A Java program for a Server
import java.net.*;
import java.io.*;

public class Server
{
    // initialize socket and input stream
    private Socket socket = null;
    private ServerSocket server = null;
    private DataInputStream in = null;

    // constructor with port
    public Server(int port)
    {
        // starts server and waits for a connection
        try
        {
            server = new ServerSocket(port);
            System.out.println("Server started");
            System.out.println("Waiting for a client ...");

            socket = server.accept()
            System.out.println("Client accepted");

            // takes input from the client socket
            in = new DataInputStream(new BufferedInputStream
                                     (socket.getInputStream()));

            String line = "";
            // reads message from client until "Over" is sent
```

```
while (!line.equals("over"))
  {
08.00
    try
09.00 {
    {
10.00 line = in.readUTF();
    System.out.println(line);
11.00 }
    catch (IOExceptim i)
12.00 {
    System.out.println(i);
01.00 }
  }
02.00
  System.out.println("Closing connection");
03.00
  //close connection
04.00 socket.close();
  in.close();
05.00 }
06.00 catch(IOExceptim i)
  {
EVE System.out.println(i);
  }
}
Notes
  public static void main(String args[])
  {
    Server server= new Server(5000);
  }
}
```

Important Points

- Server application makes a ServerSocket on a specific port which is 5000. This starts our server listening for client requests coming in for port 5000.
- Then Server makes a new socket to communicate with the client.

```
socket = server.accept()
```

- The accept() method blocks (just sits there) until a client connects to the server.
- Then we take input from the socket using getInputStream() method. Our server keeps receiving messages until the client sends "Over".
- After we're done we close the connection by closing the socket and the input stream.
- To run the Client and Server application on your machine, compile both of them. Then first run the Server application and then run the Client application.

To run on Terminal or Command Prompt.

Open two windows one for Server and another for Client

(1) First run the server application as.

```
$ java Server
Server started
Waiting for a Client.
```

(2) Then run the Client application on another terminal as,

[4 java Client]

It will show - Connected. and the server accepts the client and shows,
Client accepted.

(3) Then you can start typing messages in the Client window. Here is a sample input to the Client

```
Hello
I made my first socket connection
Over
```

which the Server simultaneously receives and shows,

```
Hello
I made my first socket connection
over
Closing connection
```

If you're using Eclipse or likes of such-

(1.) Compile both of them on two different terminals or tabs

(2) Run the server program first

(3) Then run the client program

(4) Type messages in the Client Window which will be received and shown by the Server Window simultaneously.

(5) Type Over to end.

21 FRIDAY
Apr 2023 Week 16 / 111-254

Implementing a server

08.00 The server is the program that starts first and waits for incoming connections. Implementing a server
09.00 consists of six basic steps:
(1) Create a ServerSocket object.
10.00 (2) Create a Socket object from the ServerSocket.
(3) Create an input stream to read input from the
11.00 client.
(4) Create an output stream that can be used to
12.00 send information back to the client.
(5) Do I/O with input and output streams.
01.00 (6) Close the Socket when done.

02.00 (1) Create a ServerSocket object.
With a client socket, you actively go out and
03.00 connect to a particular system. With a server, however, you passively sit and wait for someone to
04.00 come to you. So, creation requires only a port number, not a host, as follows:

05.00
```
ServerSocket listenSocket = A@
    new ServerSocket(portNumber);
```
06.00

EVE • On Unix, if you're a non privileged user, this port number must be greater than 1023 (lower numbers are reserved) and should be greater than 5000 (numbers from 1024 to 5000 are most likely to already be in use). • In addition, you should check /etc/services to make sure your selected port doesn't conflict with other services running on the same port number.

Notes

- If you try to listen on a socket that is already in use, an IOException will be thrown.

08.00

(2) Create a Socket object from the ServerSocket.

09.00

↳ Many servers allow multiple connections, continuing to accept connections until some termination condition is reached. The

10.00

11.00

↳ The ServerSocket accept method blocks until a connection is established, then returns a normal Socket object. Here is the basic idea:

12.00

01.00

```
while (some Condition) {
    Socket server = listenSocket.accept();
    doSomething With (server);
    }
```

02.00

03.00

↳ If you want to allow multiple simultaneous connections to the socket, then socket should be passed to a separate thread to create the input/output streams.

04.00

↳.

05.00

06.00

Sunday 23

(3) Create an input stream to read input from the client.

EVE

→ Once you have a Socket, you can use the saked in the same way as with the client.

Notes → The example here shows the creation of an input stream before an output stream, assuming that most servers will read data before transmitting a reply

24 MONDAY
Apr 2023 Week 17 / 114-251

APRIL 2023
S M T W T F S S M T W T F S
1 2 3 4 5 6 7 8
9 10 11 12 13 14 15 16 17 18 19 20 21 22
23 24 25 26 27 28 29 30

→ You can switch the order of this step and the next if you send data before reading, and even omit this step if your server only transmits information

→ A BufferedReader is more efficient to use underneath the InputStreamReader, as follows:

```
BufferedReader in = new BufferedReader(new InputStreamReader(server.getInputStream()));
```

→ Java also lets you use ObjectInputStream to receive complex objects from another Java program.
↳ An ObjectInputStream connected to the network is used in exactly the same way as one connected to a file; simply use readObject and cast the result to the appropriate type.

(3) Create an output stream that can be used to send information back to the client.

→ You can use a generic OutputStream if you want to send binary data.
→ If you want to use the familiar print and println commands, create a PrintWriter. Here is an example of creating a PrintWriter:

```
PrintWriter out = new
PrintWriter(server.getOutputStream());
```

In Java, you can use an ObjectOutputStream if the client is written in the Java programming language and is

Notes

expecting complex Java objects.

08.00 (5) Do I/O with input and output streams

09.00 ↳ The BufferedReader, DataInputStream, and PrintWriter classes can be used in the same ways ~~as discussed in the client section earlier in this chapter~~.

10.00 ↳ BufferedReader provides read and readLine methods

11.00 for reading characters or strings.

↳ DataInputStream has readByte and readFully methods for

12.00 reading characters or strings.

↳ Use print and println for sending high-level data

01.00 through a PrintWriter; use write to send a byte or byte array.

02.00

(6) Close the socket when done.

03.00 When finished, close the socket.

04.00 `server.close();`

05.00 This method closes the associated input and output streams but does not terminate any loop that

06.00 listens for additional incoming connections.

EVE Example: A Generic Network Server

→ Processing starts with the listen method, which waits until receiving a connection, then passes the socket to handleConnection to do the actual communication.

Notes → Real servers might have handleConnection operate in a separate thread to allow multiple simultaneous connections, but even if not, they would override the method to

26 WEDNESDAY
Apr 2023 Week 17 / 116-249

APRIL 2023
S M T W T F S S M T W T F S
1 2 3 4 5 6 7 8
9 10 11 12 13 14 15 16 17 18 19 20 21 22
23 24 25 26 27 28 29 30

provide the server with the desired behavior.
↳ The generic version of handleConnection simply reports the hostname of the system that made the connection, shows the first line of input received from the client, sends a single line to the client ("Generic Network Server"), then closes the connection.

08.00
09.00
10.00
11.00
12.00
01.00
02.00
03.00
04.00
05.00
06.00
EVE

- NetworkServer.java:

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

/** A starting point for network servers. You'll need to
override handleConnection, but in many cases listen can
remain unchanged. NetworkServer uses SocketUtil to
simplify the creation of the PrintWriter and BufferedReader
*/

public class NetworkServer {
  private int port, maxConnections;

/** Build a server on specified port. It will continue to
accept connections, passing each to handleConnection
until an explicit exit command is sent (e.g., System.exit)
or the maximum number of connections is reached.
Specify 0 for maxConnections if you want the server to
run indefinitely. */

Notes
public NetworkServer(int port, int maxConnections) {
  setPort(port);
  setMaxConnections(maxConnections);
}
```

```
/** Monitor a port for connections. Each time one is
    is established, pass resulting Socket to handleConnection.
08.00 */

09.00 public void listen(){
        int i=0;
10.00   try {
          ServerSocket listener = new ServerSocket(port);
11.00     Socket server;
          while((i++< maxConnections) || (maxConnections==0)){
12.00       server = listener.accept();
            handleConnection(server);
01.00       }
          } catch(IOException ioe){
02.00     System.out.println("IOException: " + ioe);
03.00       ioe.printStackTrace();
          }
04.00   }

05.00
/** This is the method that provides the behavior to the
06.00 server, since it determines what is done with the
    resulting socket. <B> Override this method in servers
EVE you write. </B>
    <P>
    This generic version simply reports the host that made
    the connection, shows the first line the client sent,
Notes and sends a single line in response.
    */
```

```
protected void handleConnection(Socket server)
    throws IOException{
08.00 BufferedReader in = SocketUtil.getReader(server);
      PrintWriter out = SocketUtil.getWriter(server);
09.00 System.out.println
        ("Generic Network server: got connected from " +
10.00     server.getInetAddress().getHostName() + "\n" +
        "with first line '" + in.readLine() + "'");
11.00   out.println("Generic Network Server");
        server.close();
12.00 }

01.00 /** Gets the max connections server will handle
      before exiting. A value of 0 indicates that server
02.00 should run until explicitly killed.
      */
03.00
      public int getMaxConnections(){
04.00   return(maxConnections);
      }
05.00

06.00 public void setMaxConnections(int maxConnections){
EVE     this.maxConnections = maxConnections;
      }

      /** Gets port on which server is listening. */
otes  public int getPort(){
        return(port);
      }
```

/* Sets port. * You can only do before "connect" is called. That usually happens in the constructor */

```
protected void setPort(int port){
    this.port = port;
  }
}
```

Finally, the NetworkServerTest class provides a way to invoke the NetworkServer class on a specified port.

- NetworkServerTest.java

```
public class NetworkServerTest {
public static void main (static[] args){
    int port = 8088;
    if(args.length > 0){
    port = Integer.parseInt(args[0]);
    }
    NetworkServer nwServer = new NetworkServer
                                (port, 1);
    nwServer.listen();
  }
}
```

Output: Accepting a Connection from a WWW Browser

Suppose the test program is started on port 8088 of system1.

```
[system1> java NetworkServerTest]
```

Then, a standard Web browser on system2.vm requests http://system1.vm:8088/foo/, yielding the following back on system1.vm

08.00

09.00 | Generic Network Server:
| got connection from system2.vm
10.00 | with first line 'GET /foo/HTTP/1.0'

11.00

12.00

01.00

02.00

03.00

04.00

05.00

06.00

EVE

Notes

02 TUESDAY
May 2023 Week 18 / 122-243

MAY 2023

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

• Java URL Connection Class

08.00 The Java URL Connection class represents a communication link between the URL and the
09.00 application. It can be used to read and write data to the specified resource referred by the URL.
10.00

What is the URL?

11.00

• URL is an abbreviation for Uniform Resource
12.00 Locator. An URL is a form of string that helps to find a resource on the WWW.
01.00 • It has 2 components:
(i) The protocol required to access the resource ~~component~~
02.00 (ii) The location of the resource.

Features of URLConnection class

03.00

04.00 (i) URLConnection is an abstract class. The two subclasses HttpURLConnection and ~~JarURL~~
05.00 JarURLConnection makes the connection between the client Java program and URL resource on the
06.00 internet.
(ii) With the help of URLConnection class, a user
EVE can read and write to and from any resource referenced by an URL object.
(iii) Once a connection is established and the Java program has an URLConnection object, we can use it
Notes to read or write or get further information like content length, etc.

Constructors

Constructors	Description
(1) protected URLConnection(URL url)	It constructs a URL connection to the specified URL.

URL Connection class Methods

Method

08.00
09.00
10.00
11.00
12.00
01.00
02.00
03.00
04.00
05.00
06.00
EVE

Notes

URLConnection Class Methods

Method	Description
void addRequestProperty(String key, String value)	It adds a general request property specified by a key-value pair
void connect()	It opens a communications link to the resource referenced by this URL, if such a connection has not already been established.
boolean getAllowUserInteraction()	It returns the value of the allowUserInteraction field for the object.
int getConnectionTimeout()	It returns setting for connect timeout.
Object getContent()	It retrieves the contents of the URL connection.
Object getContent(Class[] classes)	It retrieves the contents of the URL connection.
String getContentEncoding()	It returns the value of the content-encoding header field.
int getContentLength()	It returns the value of the content-length header field.
long getContentLengthLong()	It returns the value of the content-length header field as long.
String getContentType()	It returns the value of the date header field.
long getDate()	It returns the value of the date header field.
static boolean getDefaultAllowUserInteraction()	It returns the default value of the allowUserInteraction field.
boolean getDefaultUseCaches()	It returns the default value of an URLConnetion's useCaches flag.
boolean getDoInput()	It returns the value of the URLConnection's doInput flag.
boolean getDoInput()	It returns the value of the URLConnection's doOutput flag.
long getExpiration()	It returns the value of the expires header files.
static FileNameMap getFilenameMap()	It loads the filename map from a data file.

String getHeaderField(int n)	It returns the value of n^{th} header field
String getHeaderField(String name)	It returns the value of the named header field.
long getHeaderFieldDate(String name, long Default)	It returns the value of the named field parsed as a number.
int getHeaderFieldInt(String name, int Default)	It returns the value of the named field parsed as a number.
String getHeaderFieldKey(int n)	It returns the key for the n^{th} header field.
long getHeaderFieldLong(String name, long Default)	It returns the value of the named field parsed as a number.
Map<String, List<String>> getHeaderFields()	It returns the unmodifiable Map of the header field.
long getIfModifiedSince()	It returns the value of the object's ifModifiedSince field.
InputStream getInputStream()	It returns an input stream that reads from the open condition.
long getLastModified()	It returns the value of the last-modified header field.
OutputStream getOutputStream()	It returns an output stream that writes to the connection.
Permission getPermission()	It returns a permission object representing the permission necessary to make the connection represented by the object.
int getReadTimeout()	It returns setting for read timeout.
Map<String, List<String>> getRequestProperties()	It returns the value of the named general request property for the connection.
URL getURL()	It returns the value of the URLConnection's URL field.
boolean getUseCaches()	It returns the value of the URLConnection's useCaches field.
Static String guessContentTypeFromName(String fname)	It tries to determine the content type of an object, based on the specified **file** component of a URL.
static String	It tries to determine the type of an input stream based on

guessContentTypeFromStream(InputStream is)	the characters at the beginning of the input stream.
void setAllowUserInteraction(boolean allowuserinteraction)	It sets the value of the allowUserInteraction field of this URLConnection.
static void setContentHandlerFactory(ContentHandlerFactory fac)	It sets the ContentHandlerFactory of an application.
static void setDefaultAllowUserInteraction(boolean defaultallowuserinteraction)	It sets the default value of the allowUserInteraction field for all future URLConnection objects to the specified value.
void steDafaultUseCaches(boolean defaultusecaches)	It sets the default value of the useCaches field to the specified value.
void setDoInput(boolean doinput)	It sets the value of the doInput field for this URLConnection to the specified value.
void setDoOutput(boolean dooutput)	It sets the value of the doOutput field for the URLConnection to the specified value.

- How to get the object of URLConnection class?

The openConnection() method of the URL class returns the object of URLConnection class.

Syntax:

```
public URLConnection openConnection() throws IOException{}
```

- Displaying Source code of a Webpage by URLConnection class

The URLConnection class provides many methods. We can display all the data of a webpage by using the getInputStream() method. It returns all the data of the specified URL in the stream that can be read and displayed.

Example of Java URLConnection Class

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

public class URLConnectionExample {
public static void main(String[] args){
try{

URL url = new URL("http://www.javatpoint.com/java-tutorial");
URLConnection urlcon = url.openConnection();
InputStream stream = urlcon.getInputStream();
int i;
```

```
while ((i=stream.read()) != -1){
    System.out.print((char)i);
}
}catch(Exception e)
{
System.out.println(e);}
}
}
```

08.00

09.00

10.00

11.00

12.00

01.00

url.openConnection()

02.00

03.00

04.00

05.00

06.00

EVE

Notes