

Chapter: 1

Introduction

Contents:

- Network Programming Features and Scope
- Network Programming Language, Tools & Platforms
- Client and Server Application
- Client Server Model and Software Design

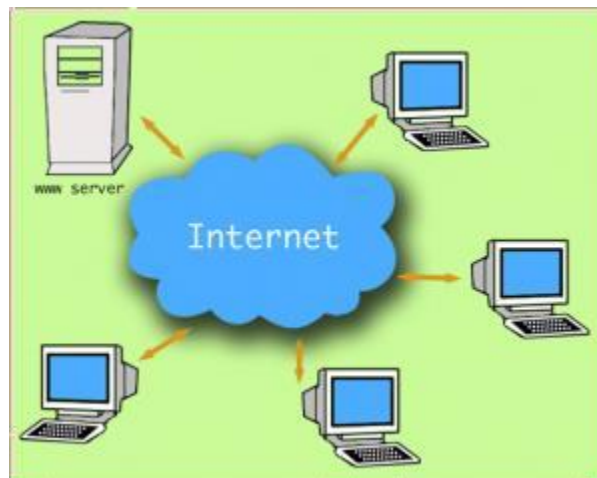
Definition:

Network: A network is a group of two or more computers or other electronic devices that are

interconnected for the purpose of exchanging data and sharing resources.

Group of Device:

- Connect
- Communicate



Network Programming:

Network programming is a type of programming that involves writing code to communicate with other devices over a network.

Network Programming Feature and Scope:

some key features of network programming:

- **Socket Programming:** Sockets are a fundamental concept in network programming. Sockets are endpoints for sending and receiving data across a network. Network programming often involves using sockets to establish connections between devices.
- **Client-Server Architecture:** Network programming often involves creating client-server applications, where one device acts as the server, providing resources or services, and the other devices act as clients, accessing those resources or services.
- **Protocols:** Network programming involves using network protocols to ensure that devices can communicate with each other. Protocols specify the rules and procedures for transmitting and receiving data over a network.

- **Multi-threading:** Network programming often requires the use of multiple threads to handle simultaneous connections and requests. Multi-threading allows an application to handle multiple tasks at the same time.
- **Security:** Network programming involves implementing security measures to protect against unauthorized access and data breaches. This may include encryption, authentication, and access controls.
- **Asynchronous I/O:** Network programming often requires performing I/O operations without blocking the execution of the program. Asynchronous I/O allows an application to continue processing while waiting for I/O operations to complete.
- **Remote Procedure Calls (RPC):** Network programming can involve using remote procedure calls to allow programs on different devices to communicate with each other. RPC allows a program

to call a function on a remote device as if it were a local function call.

The **scope** of network programming is vast and covers a range of topics related to the communication between two or more computing devices over a network.

Network programming is essential for building a wide range of **networked applications and systems**, and it plays a critical role in the modern digital world.

Network Programming Language Tools and Platform:

- Python (open-source programming language)
- Java (general-purpose, object-oriented programming language)

- Perl (general-purpose programming language)
- Bash (command-line-interface tool)

In addition to these programming languages, several **platforms** like Apache Kafka, RabbitMQ, and Redis are also commonly used for network programming.

Apache Kafka:

- Open source distributed streaming
- Operate as a distributed system
- Stream processing platform
- High throughput
- Low latency platform for handling real-time feeds.

RabbitMQ:

- RabbitMQ is designed to handle messaging between applications, particularly in distributed systems.
- High-throughput messaging systems.

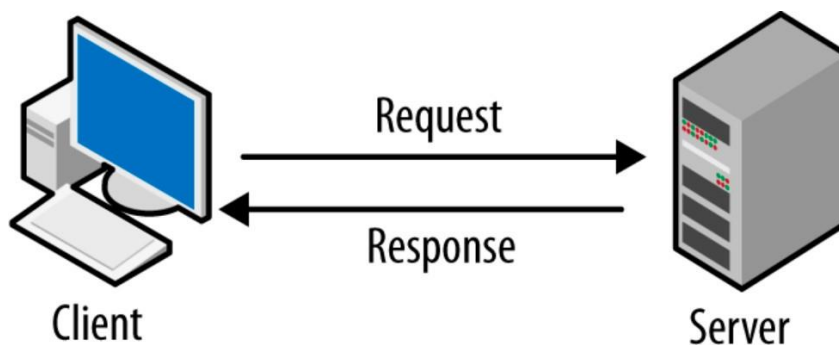
Redis:

- Redis is an open-source, in-memory data structure store that can be used as a database, cache, and message broker.
- It is designed to be fast, efficient, and reliable, making it an excellent choice for applications that require real-time data processing.

These platforms provide **easy-to-use APIs** and infrastructure for building distributed and scalable applications.

Client and Server Application:

- A client-server application is a distributed computing architecture that consists of two main components: a client and a server.
- The client is a program or application that requests services or resources from the server, while the server is a program or application that provides those services or resources to the client.
- A client-server application is **a program that runs on the client-side while accessing the information over a remote server.**



Client Server Model and Software Design:

- The client-server model is a **software architecture** in which a client sends requests to a server, which responds by sending data or performing some operation.
- In this model, the client and server communicate over a network using a standardized protocol such as **TCP/IP**. The client sends a request to the server, which processes the request and sends a response back to the client.
- In software design, the client-server model is often used to create distributed systems where different parts of the system are located on different machines.
- In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and deliver the data packets requested back to the client. Clients do not share any of their resources.

Examples of Client-Server Model are Email, World Wide Web, etc.

Advantages of Client-Server model:

- Centralized system with all data in a single place.
- Cost efficient requires less maintenance cost and Data recovery is possible.
- The capacity of the Client and Servers can be changed separately.

Disadvantages of Client-Server model:

- Clients are prone to viruses, Trojans and worms if present in the Server or uploaded into the Server.
- In case of server failure entire network will be failed.

- Scalability
- Phishing or capturing login credentials or other useful information of the user are common and MITM(Man in the Middle) attacks are common.

2 Tier Architecture:

In 2-tier architecture, the client communicates directly with the server.

The server provides the necessary services or resources to the client, which then uses them locally.

In this model, the client performs the presentation layer and the application layer, while the server performs the data layer.

For example, in a simple client-server system like a file sharing system, the client application running on a user's computer directly accesses the file server to retrieve or upload files.

3 Tier Architecture:

In 3-tier architecture, the client communicates with an application server that serves as an intermediary between the client and the database server.

This model separates **the presentation layer, application layer, and data layer** into three separate tiers.

The first tier is the presentation tier, which includes the user interface and the client application.

The second tier is the application tier, which includes the business logic and processing of data.

The third tier is the data tier, which includes the database server where data is stored.



For example, in a web-based e-commerce application, the client interacts with the presentation tier (e.g., the user interface on a web page), which sends requests to the application server.

The application server processes these requests and interacts with the database server to retrieve or update data.

The application server then sends the updated information back to the presentation tier, which displays it to the user.

Design Consideration of Client Server Application Architecture:

- Client and Server machines need different amounts of hardware and software resources.

- Client and server machines may belong to different vendors.
- Horizontal scalability (increase of the client machine) and vertical scalability (migration to a more powerful server or to a multiserver solution)
- A client or server application interacts directly with a transport layer protocol to establish communication and to send or receive information.
- The transport protocol then uses lower layer protocols to send or receive individual message. Thus, a computer needs a complete stack of protocols to run either a client or a server.
- A single server-class computer can offer multiple services at the same time, a separate server program is needed for each service.

Few Protocols and Description:

IPv4:

Internet Protocol version 4. IPv4 uses 32-bit addresses and provides packet delivery service for TCP, UDP, SCTP, ICMP (Internet Control Message Protocol), and IGMP. (Internet Group Management Protocol).

ICMP is a protocol used for error reporting and diagnostics in IP networks, while IGMP is a protocol used for managing multicast groups in IP networks.

IPv6:

Internet Protocol version 6. IPv6 uses 128-bit addresses.

TCP:

Transmission Control Protocol. TCP is a **connection-oriented** protocol that provides a **reliable, full-duplex** byte stream to its users

UDP:

User Datagram Protocol. UDP is a **connectionless** protocol, and UDP sockets are an example of datagram sockets.

SCTP:

Stream Control Transmission Protocol. SCTP is a **connection-oriented** protocol that provides a reliable full-duplex association.

SCTP provides a reliable, message-oriented transport layer protocol that is well-suited for a wide range of applications that require message-oriented communication over IP networks.

ICMP:

Internet Control Message Protocol. ICMP handles error and control information between routers and hosts.

Chapter: 2

Internet Addresses

Contents:

- The InetAddress Class: Creating New Inet Address Object, Getter
- Method, Address Types, Testing Reachability and Object Methods
- Inet4Address and Inet6Address
- The Network Interface Class: Factory Method & Getter Method
- Some Useful Programs: SpamCheck, Processing Web Server Logfiles

IP Address:

- An **IP address**, or **Internet Protocol address**, is a **numerical label** assigned to devices that are connected to a computer network.
- The IP address serves as a **unique identifier** that enables devices to communicate with each other over the network.

There are two main types of IP addresses:

- IPv4
- IPv6.

IPv4:

- (Internet Protocol version 4) addresses are **32-bit binary numbers**.
- It consists of 4 numbers separated by the dots.
- Each number can be from 0-255 in decimal numbers.
But computers do not understand decimal numbers,

they instead change them to binary numbers which are only 0 and 1. Therefore, in binary, this (0-255) range can be written as (00000000 – 11111111).

- A total of **(2³²) devices approximately** = 4,294,967,296 can be **assigned** with IPv4.
- IP Address range: 0.0.0.0 to 255.255.255.255
- An example of an IPv4 address is 192.0.2.1.

IPv6:

- (Internet Protocol version 6) addresses are **128-bit binary numbers**, which are typically represented in hexadecimal format.
- IPv6 is written as a group of **8 hexadecimal numbers** separated with **colons(:)**
- A total of **(2¹²⁸) devices** can be assigned with **unique addresses** which are actually more than enough for upcoming future generations.

- An example of an IPv6 address is 2001:0db8:85a3:0000:0000:8a2e:0370:7334.

InetAddress:

- The **InetAddress** class is a Java class that represents an IP address, which can be either an IPv4 or IPv6 address.
- The **java.net.InetAddress** class provides methods to get the IP of any host name.
- For example www.google.com, www.facebook.com, etc.
- An IP address is represented by 32-bit or 128-bit.
- InetAddress can **handle** both **IPv4 and IPv6 addresses**.

There are two types of addresses:

- Unicast - An identifier for a **single interface**.
- Multicast -An identifier for a **set of interfaces**.

Creating new InetAddress Objects:

Steps:

1. Import the **java.net.InetAddress** class:

```
import java.net.InetAddress;
```

2. Declare object of InetAddress e.g. address and call the static method `getByName("domain_name")` of InetAddress

```
InetAddress address =  
InetAddress.getByName("www.example.com");
```

3. Print the result

```
System.out.println(address);
```

Note: that the **getByName()** method can throw an **UnknownHostException** if the host name is not valid. You should handle this exception appropriately in your code.

Example:

```
import java.io.*;
import java.net.*;
public class InetAddressExample{
public static void main(String[] args){
try{
InetAddress ip=InetAddress.getByName("www.google.
com");

System.out.println("Host Name: "+ip.getHostName());
System.out.println("IP Address: "+ip.getHostAddress())
;
}catch(Exception e){System.out.println(e);}
}
}
```

Getter Methods:

1. **getHostName():** Returns the host name associated with the IP address as a string. If the host name is not available, the method returns the IP address in textual form.
2. **getHostAddress():** Returns the IP address in textual form as a string.
3. **getAddress():** Returns the raw IP address as a byte array. For IPv4 addresses, the array will contain four bytes; for IPv6 addresses, it will contain 16 bytes.
4. **getCanonicalHostName():** Returns the fully qualified domain name (FQDN) of the host associated with the instance of InetAddress class.

Example:

```
import java.io.*;
import java.net.*;
public class InetAddressExample{
public static void main(String[] args){
try{
InetAddress ip=InetAddress.getByName("www.
google.com");

System.out.println("Host Name: "+ip.getHostNa
me());
System.out.println("IP Address: "+ip.getHostAd
dress());
System.out.println("Canonical Host
Name: "+ip.getCanonicalHostName());
System.out.println("IP Address: "+ip.getHostAd
dress());

}catch(Exception e){System.out.println(e);}
}
}
```


Address Types:

- **Wildcard Address:** 0.0.0.0 or :: => isAnyLocalAddress()
- **Loopback Address :** address connects to the same computer i.e. 0::1; 127.0.0.1 => isLoopbackAddress()
- **link-local address:** address used to help IPv6 networks self-configure without necessarily using a server e.g. FE80:: => isLinkLocalAddress()
- **site-local address:** ip forwarded by routers within a site or campus e.g. FEC0:: => isSiteLocalAddress();
- **Multicast Address:** broadcasts content to all subscribed computers rather than to one particular computer e.g. 224.0.0.0 to 239.255.255.255 or IPv6 multicast addresses start with FF. FF00:: => isMulticastAddress();

- **global multicast address:** subscribers around the world e.g. 224.0.2.1; begins FF0E or FF1E => isMCGlobal();
- **organization-wide multicast address:** subscribers within all the sites of a company or organization e.g. begin with FF08 or FF18, => isMCOrgLocal()
- **site-wide multicast address:** Packets addressed to a site-wide address will only be transmitted within their local site e.g. 224.0.2.1; FF05 or FF15 => isMCSiteLocal();
- **subnet-wide multicast address:** Packets addressed to a link-local address will only be transmitted within their own subnet e.g. FF02 or FF12 => isMCLinkLocal();
- **interface-local multicast address:** Packets addressed to an interface-local address are not sent beyond the network interface from which they originate, not even to a different network interface on the same node e.g. FF01 or FF11 => isMCNodeLocal();

Methods:

- isAnyLocalAddress()
- isLoopbackAddress()
- isLinkLocalAddress()
- isMCGlobal()
- isMCLinkLocal()
- isMCNodeLocal()
- isMCOrgLocal()
- isMCSiteLocal()
- isMulticastAddress():

Testing Reachability:

- Connections can be blocked for many reasons, including firewalls, proxy server, misbehaving routers, and broken cables, or simply because the remote host is not turned on
- Testing whether a particular node is reachable from the current host (i.e., whether a network connection can be made)
- If the host responds within timeout milliseconds, the methods return true; otherwise, they return false.

- **Methods:**

```
public boolean isReachable(int timeout) throws  
IOException
```

```
public boolean isReachable(NetworkInterface  
interface, int ttl, int timeout) throws IOException
```

ttl(time to live) - the maximum number of network hops the connection will attempt before being discarded.

Inet4Address:

Example:

```
import java.io.*;
import java.net.*;
public class Inet4AddressExample{
public static void main(String[] args){
try{
    //Get the Inet4Address object for a given IP
    address string
    Inet4Address ip=(Inet4Address)Inet4Address.get
    ByName("192.168.0.1");

    System.out.println("Host Name: "+ip.getHostNa
    me());
    System.out.println("IP Address: "+ip.getHostAd
    dress());

catch(Exception e){System.out.println(e);}
}
}
```

Inet6Address:

This class represents IPv6 address and extends the InetAddress class.

Methods of this class provide facility to represent and interpret IPv6 addresses.

Methods of this class takes input in the following formats:

1. **x:x:x:x:x:x:x:x** –This is the general form of IPv6 address where each x can be replaced with a 16 bit hexadecimal value of the address.

For example:

4B0C:0:0:0:880C:99A8:4B0:4411

2. When the address contains multiple set of 8 bits as '0'. In such cases '::' is replaced in place of 0's to make the address shorter.

For example:

4B0C::880C:99A8:4B0:4411

3. **x:x:x:x:x:x:d.d.d.d** –A third format is used when hybrid addressing(IPv6 + IPv4) has to be taken care of. In such cases the first 12 bytes are used for IPv6 addressing and remaining 4 bytes are used for IPv4 addressing.

For example:

2001:0db8:85a3::8a2e:0370:192.168.0.1

InetAddress – Factory Methods:

- The **InetAddress** class is used to encapsulate both, the numerical IP address and the domain name for that address.
- The InetAddress class has no visible constructors.
- The InetAddress class has the inability to create objects directly, hence **factory methods** are used for the purpose.

- Factory Methods are **static methods** in a class that return an object of that class.

Method	Description
<code>public static InetAddress getLocalHost() throws <i>UnknownHostException</i></code>	This method returns the instance of <code>InetAddress</code> containing the local hostname and address.
<code>public static InetAddress getByName (String host) throws <i>UnknownHostException</i></code>	This method returns the instance of <code>InetAddress</code> containing IP and Host name of host represented by host argument.

Method	Description
<pre>public static InetAddress[] getAllByName(String hostName) <i>throws</i> UnknownHostException</pre>	<p>This method returns the array of the instance of InetAddress class which contains IP addresses.</p>
<pre>public static InetAddress getByAddress(byte IPAddress[]) <i>throws</i> UnknownHostException</pre>	<p>This method returns an InetAddress object created from the raw IP address.</p>
<pre>public static InetAddress getByAddress(String hostName, byte IPAddress[]) <i>throws</i> UnknownHostException</pre>	<p>This method creates and returns an InetAddress based on the provided hostname and IP address.</p>

Example:

```
// To get and print InetAddress of Local Host
InetAddress address1 = InetAddress.getLocalHost();
System.out.println("InetAddress of Local Host : "
    + address1);

// To get and print InetAddress of Named Host
InetAddress address2
    = InetAddress.getByName("45.22.30.39");
System.out.println("InetAddress of Named Host : "
    + address2);

// To get and print ALL InetAddresses of Named
Host
InetAddress address3[]
    = InetAddress.getAllByName("172.19.25.29");
for (int i = 0; i < address3.length; i++) {
    System.out.println(
        "ALL InetAddresses of Named Host : "
        + address3[i]);
}

// To get and print InetAddresses of Host with
specified IP Address
byte IPAddress[] = { 125, 0, 0, 1 };
InetAddress address4
    = InetAddress.getByAddress(IPAddress);
System.out.println(
```

```

        "InetAddresses of Host with specified IP Address
: "
        + address4);

// To get and print InetAddresses of Host with
specified IP Address and hostname
byte[] IPAddress2
    = { 105, 22, (byte)223, (byte)186 };
InetAddress address5 = InetAddress.getByAddress(
    "gfg.com", IPAddress2);
System.out.println(
    "InetAddresses of Host with specified IP Address
and hostname : "
    + address5);

```

Output:

InetAddress of Local Host : localhost/127.0.0.1

InetAddress of Named Host : /45.22.30.39

ALL InetAddresses of Named Host : /172.19.25.29

InetAddresses of Host with specified IP Address :
/125.0.0.1

InetAddresses of Host with specified IP Address and
hostname : gfg.com/105.22.223.186

SpamCheck:

- A number of services monitor spammers, and inform clients whether a host attempting to connect to them is a known spammer or not.
- These real-time blackhole lists need to respond to queries extremely quickly, and process a very high load.
- Thousands, maybe millions, of hosts query them repeatedly to find out whether an IP address attempting a connection is or is not a known spammer.
- To find out if a certain IP address is known spammer, reverse the byte of the address, add the domain of the blackhole service, and look it up. If the address is found, it's a spammer. If it isn't, it's not.
- For instance, if you want to ask `sbl.spamhaus.org` if `192.168.0.1` is a spammer, you would look up the hostname `1.0.168.192.sbl.spamhaus.org`.

- If the DNS query succeeds, if it returns the address 127.0.0.2 then the host is known to be a spammer. Otherwise it isn't.

Example:

```
public class SpamCheckIp {  
  
    public static boolean isSpam(String ipAddress) {  
        // Reverse the IP address and append the blacklist  
        domain  
        String blacklistDomain = "sbl.spamhaus.org";  
        String reversedIpAddress = new  
        StringBuilder(ipAddress).reverse().toString();  
        String query = reversedIpAddress + "." +  
        blacklistDomain;  
  
        try {  
            // Perform a DNS lookup on the query string  
            InetAddress address =  
            InetAddress.getByName(query);  
  
            // Return true if the IP address is listed in the  
            blacklist
```

```
        return  
address.getHostAddress().equals("127.0.0.2");  
    } catch (UnknownHostException ex) {  
        // The DNS lookup failed, so assume the IP  
        address is not listed  
        return false;  
    }  
}
```

```
public static void main(String[] args) {  
    String ipAddress = "192.0.2.1";  
    boolean isSpam = isSpam(ipAddress);  
    System.out.println("Is spam \n"+isSpam);  
}  
}
```

Output:

Is spam?

False

Processing Web Server Logfiles:

- Web Server logs **track the hosts that access a website.**
- By default, the log reports the IP addresses of the sites that connect to the server.
- They can be used to analyze user behavior, track website performance, and identify potential security issues.

Example:

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

public class WebLogTest {

    public static void main(String[] args) {
        String file = "logfile.txt";
        try (FileInputStream fin = new FileInputStream(file);
            Reader in = new InputStreamReader(fin);
            BufferedReader bin = new BufferedReader(in);) {

            for (String entry = bin.readLine(); entry != null; entry = bin.readLine())
            {
                // separate out the IP address
                int index = entry.indexOf(' '); // position of the first space
```

```
String ip = entry.substring(0, index); // read IP
String theRest = entry.substring(index); // reads remain (extra
information)
```

```
// Ask DNS for the hostname and print it out
try {
    InetAddress address = InetAddress.getByName(ip);
    System.out.println(address.getHostName() + theRest);
} catch (UnknownHostException ex) {
    System.err.println(entry);
}
}
} catch (IOException ex) {
    System.out.println("Exception: " + ex);
}
}
}
```

Sample Log:

205.160.186.76 unknown

[11/Mar/2023:22:53:76 -0500] “GET

/bgs/greenbg.gif HTTP 1.0” 200 50

Chapter -3

URLs and URIs

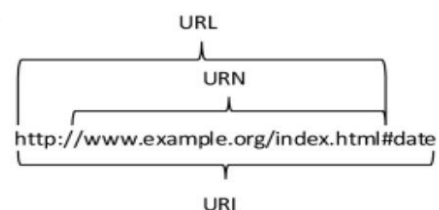
(5 Hours)

Contents

- **URIs:** URLs and Relative URLs
- **The URL Class:** Creating New URLs, Retrieving Data From a URL, Splitting a URL into Pieces, Equality & Comparison and Conversion
- **The URI Class:** Constructing a URI, The Parts of the URI, Resolving Relative URIs, Equality & Comparison and String Representation
- **x-www-form-urlencoded:** URL Encoder and URL Decoder
- **Proxies:** System Properties, The ProxyClass and The ProxySelector Class
- Communicating with Server-Side Programs Through GET
- **Accessing Password-Protected Sites:** The Authenticator Class, The Password Authentication Class and The JPasswordField Class

URLs

- The **URL** class is the simplest way for a Java program to **locate and retrieve data from the network**.
- We do not need to worry about the details of the protocol being used, the **format of the data being retrieved, or how to communicate with the server**; simply tell Java the URL and it gets the data



URL Class

BCA
6th

- URL Class represents a Uniform Resource Locator, a pointer to a “resource” on the world wide web.
- **Absolute URL:** contains all of the information necessary to reach the resource.

E.g. /about

- **Relative URL:** contains only enough information to reach the resource relative to (or in the context) another URL.

E.g. <http://www.example.com/about>

Components of URL

BCA
6th

- **Protocol:** HTTP is the protocol here
- **Hostname:** Name of the machine on which the resource lives.
- **File Name:** The path name to the file on the machine.
- **Port Number:** Port number to which to connect (typically optional).

<http://www.studytonight.com:80/index.html>

protocol hostname portnumber file name

<http://www.ambition.edu.np/notices/entrance.pdf>

- **Base URL**

<http://www.ambition.edu.np/notices/>

- **Relative URL String**

[entrance.pdf](#)

- **Resolved URL**

<http://www.ambition.edu.np/notices/entrance.pdf>

In JAVA

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

```
String baseURLStr = "http://www.ietf.org/rfc/rfc3986.txt";
```

```
String relativeURLStr = "rfc2732.txt";
```

```
URL baseURL = new URL(baseURLStr);
```

```
URL resolvedRelativeURL = new URL(baseURL, relativeURLStr);
```

```
System.out.println("Base URL:" + baseURL);
```

```
System.out.println("Relative URL String:" + relativeURLStr);
```

```
System.out.println("Resolved Relative URL:" + resolvedRelativeURL);
```

Creating URL

BCA
6th

(a) creates a URL with string url representation

- URL url1 = new
URL("<https://www.google.com/search?q=computer+engineer&sclient=gws-wiz>");

(b) creates a URL with a protocol, hostname, and path

- URL url2 = new URL("http", "[www.google.com](http://www.google.com/contact/)", "/contact/");

(b) creates a URL with a protocol, hostname, port and path

- URL url2 = new URL("http", "www.google.com", 8008, "/contact/");

(b) creates a URL with a url and string relative

- URL u1 = new URL("<http://www.ibiblio.org/javafaq/index.html>");
- URL u2 = new URL (u1, "[mailinglists.html](http://www.ibiblio.org/javafaq/index.html#mailinglists.html)");

Constructing a URL from a string

BCA
6th

Which protocols does a virtual machine support?

```
try {  
    URL u1 = new URL("http://www.ambition.edu.np/");  
    System.out.println(u1.getProtocol()); // http  
    URL u2 = new URL("verbatim:http://www.adc.org/");  
    System.out.println(u2.getProtocol()); // error  
}  
catch (MalformedURLException ex) {  
    System.err.println(ex);  
}
```

Methods that retrieve data from a URL

BCA
6th

- `public InputStream openStream()`: connects to the resource referenced by the URL, performs any necessary handshaking between the client and the server, and returns an Input Stream from which data can be read.
- `public URLConnection openConnection()`: opens a socket to the specified URL and returns a URLConnection object.
- `public Object getContent()`: method retrieves the data referenced by the URL and tries to make it into some type of object.

openStream()

BCA
6th

- `public final InputStream openStream()` throws IOException

```
try {
    URL u = new URL("http://www.tufohss.edu.np");
    InputStream in = u.openStream();
    int c;
    while ((c = in.read()) != -1)
        System.out.write(c);
    in.close();
} catch (IOException ex) {
    System.err.println(ex);
}
```


openConnection()

BCA
6th

- **public URLConnection openConnection()** throws IOException

```
try {
    URL u = new URL("http://www.tufohss.edu.np");
    try {
        URLConnection uc = u.openConnection();
        InputStream in = uc.getInputStream();
        // read from the connection...
    } catch (IOException ex) {
        System.err.println(ex);
    }
} catch (MalformedURLException ex) {
    System.err.println(ex);
}
```

getContent()

BCA
6th

- **public final Object getContent()** throws IOException

```
try {
    URL u = new URL("http://www.oreilly.com/graphics_new/animation.gif");
    Object o = u.getContent();
    System.out.println("I got a " + o.getClass().getName());
} catch (MalformedURLException ex) {
    System.err.println(args[0] + " is not a parseable URL");
} catch (IOException ex) {
    System.err.println(ex);
}
```

Retrieving Data From a URL

BCA
6th

- Now, create a **new URL object** and pass the desired URL that we want to access.
`URL url = new URL(theUrl);`
- Now, using this url object, create a **URLConnection object**.
`URLConnection urlConnection = url.openConnection();`
- Use the **InputStreamReader and BufferedReader** to read from the URL connection. `BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(urlConnection.getInputStream()));`
- The **readLine method** of `BufferedReader` returns a String that we can access. If this string is null, it means we have reached the end of the document. `while ((line = bufferedReader.readLine()) != null)`
- Now, **Append the string series** that we have received as output from the URL to your `StringBuilder` object. `content.append(line + "\n");`
- print **output**: `content.toString();`

Example:

BCA
6th

```
URL url = new URL(theUrl); // creating a url object
URLConnection urlConnection = url.openConnection(); // creating a urlconnection object

// wrapping the urlconnection in a bufferedreader
BufferedReader bufferedReader = new BufferedReader(new InputStreamReader( urlConnection.getInputStream()));
String line;

// reading from the urlconnection using the bufferedreader
while ((line = bufferedReader.readLine()) != null)
{
    content.append(line + "\n");
}
bufferedReader.close();
System.out.println(content.toString());
```

Splitting a URL into Pieces

BCA
6th

URLs are composed of **FIVE pieces**:

- The scheme, also known as the protocol
- The authority
- The path
- The fragment identifier, also known as the section or ref
- The query string

E.g.

<http://www.ibiblio.org:8080/javafaq/books/jnp/index.html? isbn=1565922069#toc>

scheme = http,

authority = www.ibiblio.org:8080,

path = /javafaq/books/jnp/index.html, fragment identifier = toc,

query string = isbn=1565922069

Methods to access pieces

BCA
6th

- Read-only access to these parts of a URL is provided by nine public methods:
- `getFile()`, `getHost()`, `getPort()`, `getProtocol()`, `getRef()`, `getQuery()`, `getPath()`, `getUserInfo()`, and `getAuthority()`.

Example: Splitting a URL into Pieces

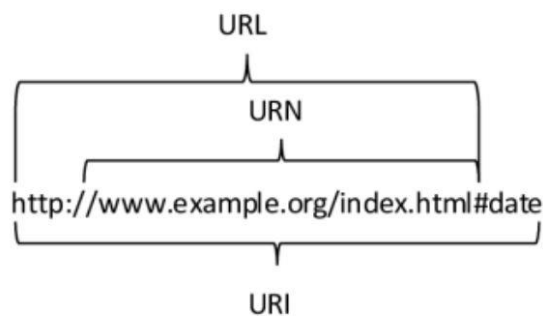
BCA
6th

```
URL u = new URL("http://www.ibiblio.org/wc/compositions.phtml?category=Piano");
System.out.println("The URL is " + u); // http://www.ibiblio.org/wc/compositions.phtml?category=Piano
System.out.println("The scheme is " + u.getProtocol()); //http
System.out.println("The user info is " + u.getUserInfo()); //null
String host = u.getHost();
if (host != null) {
    int atSign = host.indexOf('@');
    if (atSign != -1) host = host.substring(atSign+1);
    System.out.println("The host is " + host); //www.ibiblio.org
} else {
    System.out.println("The host is null.");
}
System.out.println("The port is " + u.getPort()); //-1
System.out.println("The path is " + u.getPath()); //wc/compositions.phtml
System.out.println("The ref is " + u.getRef()); //null
System.out.println("The query string is " + u.getQuery()); //Piano
```

URI Class

BCA
6th

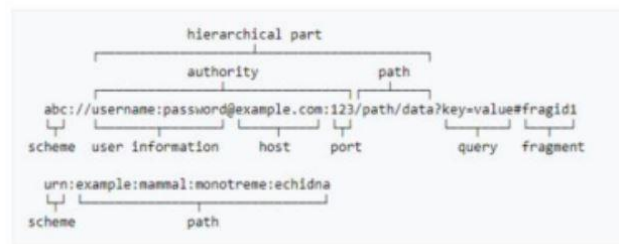
- A Uniform Resource Identifier (URI) is a sequence of characters used for **identification of a particular resource**.
- It enables for the interaction of the representation of the resource over the network using specific protocols.



URI Constructors

BCA
6th

- **URI(String str)** : Constructs a URI object by parsing the specified string.
- **URI(String scheme, String ssp, String fragment)** A component may be left undefined by passing null. Initially the result string is empty. If scheme is not null it is appended.
- **URI(String scheme, String userInfo, String host, int port, String path, String query, String fragment)**
- **URI(String scheme, String host, String path, String fragment)**
- **URI(String scheme, String authority, String path, String query, String fragment)**
 - **scheme** : string representing scheme
 - **userInfo** : userinfo of URI
 - **host** : host component of URI
 - **port** : listening port number
 - **path** : path of URI
 - **query** : String representing the query part
 - **fragment** : optional fragment



Constructing URI

BCA
6th

// Constructor to create a new URI by parsing the string

```
String uri = "http://www.ibiblio.org";
```

```
URI uriBase = new URI(uri);
```

// create() method

```
URI uriBase = URI.create(uri);
```

Resolving Relative URIs

BCA
6th

```
String uribase = "https://www.test.org/";
String urirelative = "languages/../java";
URI uriBase = new URI(uribase);

// create() method
URI uri = URI.create(str);
// toString() method
System.out.println("Base URI = " + uriBase.toString()); //https://www.test.org/

URI uriRelative = new URI(urirelative);
System.out.println("Relative URI = " + uriRelative.toString()); //languages/../java

// resolve() method
URI uriResolved = uriBase.resolve(uriRelative);
System.out.println("Resolved URI = " + uriResolved.toString()) //https://www.geeksforgeeks.org/java
https://www.geeksforgeeks.org/java-net-uri-class-java/
```

URL Encoder

BCA
6th

- **Java.net.URLEncoder**
- This class is a utility class for HTML form encoding.
- Encoding makes the form of URL more reliable and secure.
- When the user request is triggered by a get method, the form parameters and their values are appended at the end of URL after a '?' sign.

Example

bca@network programming => bca%40network+programming

Rules when encoding a string

BCA
6th

- Alphanumeric characters and certain special characters such as '*', '_', '-' and '.' remains **unchanged**.
- **Spaces** are converted into '+' signs.
- All other characters are encoded by one or more bytes using the encoding scheme specified. They are converted in a three character string of the form %xy, where xy represents the hexadecimal representation of the encoding character.

Example

bca@ambition academy => bca%40ambition+academy

Method: **encode()**

BCA
6th

(a) Syntax : `public static String encode(String s)`

Parameters :

s : String to be encoded

```
URLEncoder.encode("hello world", "UTF-8")
```

(b) Syntax : `public static String encode(String s, String enc)`

Parameters :

s : string to be encoded

enc : encoding to be used e.g. UTF-8

```
URLEncoder.encode("hello world", "UTF-8")
```

Example Code

BCA
6th

```
// base URL
String baseUrl = "https://www.ambition.edu.np?q=";

// String to be encoded
String query = "info@gmail for bca";

System.out.println("URL without encoding :");
URL url = new URL(baseUrl + query);
System.out.println(url);

// encode() method
System.out.println("URL after encoding :");
url = new URL(baseUrl + URLEncoder.encode(query, "UTF-8"));
System.out.println(url);
```

26

URL Decoder

BCA
6th

- **Java.net.URLDecoder class**
- utility class for HTML form decoding.
- It just performs the reverse of what URLEncoder class do, i.e. given an encoded string, it decodes it using the scheme specified.

Example

bca%40network+programming => bca@network programming

Steps while decoding the strings

BCA
6th

- Alphanumeric characters and certain special characters such as '*', '_', '-' and '.' remains unchanged.
- '+' signs are converted into spaces.
- All other characters are decoded using the encoding scheme specified. The string of the form %xy, is converted to the character whose encoding would have resulted in this three character representation.

Example

bca%40ambition+academy => bca@ambition academy

decode()

BCA
6th

(a) Syntax : `public static String decode(String s)`

Parameters :

s : encoded string to be decoded

`URLDecoder.decode("hello+world")`

(b) Syntax : `public static String decode(String s, String enc)`

Parameters :

s : string to be decoded

enc : encoding to be used e.g. UTF-8

`URLDecoder.decode("hello+world", "UTF-8")`

Example Code

BCA
6th

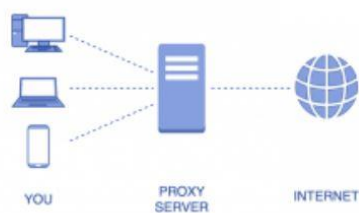
```
// encoded string
String encodedString = "info%40gmail+for+bca";
System.out.println("Encoded String :");
System.out.println(encodedString);

// decode() method
System.out.println("Decoded String :");
System.out.println(URLEncoder.decode(encodedString, "UTF-8"));
```

Proxies

BCA
6th

- Proxy means **‘in place of’**, representing’ or **‘in place of’** or **‘on behalf of’**
- A real world example can be a cheque or credit card is a proxy for what is in our bank account.
- Proxy pattern does – **“Controls and manage access to the object they are protecting”**.



System Properties

BCA
6th

- For basic operations, all you have to do is set a few system properties to point to the addresses of your local proxy servers.
- If you are using a **HTTP proxy**, set `http.proxyHost` to the domain name or the IP address of your proxy server and `http.proxyPort` to the port of the proxy server (the default is 80).
- `System.setProperty("http.proxyHost", "192.168.254.254");`
- `System.setProperty("http.proxyPort", "9000");`
- `System.setProperty("http.nonProxyHosts", "java.oreilly.com|xml.oreilly.com");`

Proxy Class

BCA
6th

- The Proxy class allows more fine-grained control of proxy servers from within a Java program.
- Specifically, it allows you to choose different proxy servers for different remote hosts.
- The proxies themselves are represented by instances of the `java.net.Proxy` class.
- **Example:**

```
SocketAddress address = new InetSocketAddress("proxy.example.com", 80);  
Proxy proxy = new Proxy(Proxy.Type.HTTP, address);
```


Proxy Selector

BCA
6th

- Each running virtual machine has a single `java.net.ProxySelector` object it uses to *locate the proxy server for different connections*
- To change the Proxy Selector, pass the new selector to the static `ProxySelector.setDefault()` method, like so:

```
ProxySelector selector = new LocalProxySelector(); // returns list of proxies  
ProxySelector.setDefault(selector);
```

Communicating with Server-Side Programs Through GET

BCA
6th

- The URL class makes it easy for Java applets and applications to **communicate with serverside programs** such as CGI's, servlets, PHP pages, and others that use the GET method.

```
<form name="search" action="http://www.google.com/search" method="get">  
    <input name="q" />  
    <input type="submit" value="Search" />  
</form>
```

Basic Syntax

BCA
6th

`http://www.google.com/search?q=computer`

```
QueryString query = new QueryString();
```

```
query.add("q", target);
```

```
URL u = new URL("http://www.google.com/search?" + query);
```

... write code for reading webpage

Accessing Password-Protected Site

BCA
6th

- Java's URL class **can access** sites that use **HTTP authentication**, though you'll of course need to *tell it what username and password to use*.
- **cookie-based authentication** is more challenging, not least because this varies a lot from one site to another

Authenticator Class: **Authenticator()**

BCA
6th

- the **java.net** package includes an Authenticator class you can use to provide a username and password for sites that protect themselves using HTTP authentication:

```
public abstract class Authenticator extends Object
```

- **Methods**

```
Authenticator.setDefault(new DialogAuthenticator());
```

```
// Sets the authenticator to be used when a HTTP server requires authentication.
```

Methods from the Authenticator superclass

BCA
6th

- protected final **InetAddress** **getRequestingSite()** // requesting for the authorization,
- protected final int **getRequestingPort()**
- protected final String **getRequestingProtocol()**
- protected final String **getRequestingPrompt()**
- protected final String **getRequestingScheme()**
- protected final String **getRequestingHost()**
- protected final String **getRequestingURL()**
- protected **RequestorType** **getRequestorType()** // requester is a Proxy or a Server.

Example: Methods from the Authenticator

BCA
6th

```
ClassAuthenticator obj1 = new ClassAuthenticator();  
Authenticator.setDefault(new ClassAuthenticator());  
obj1.getPasswordAuthentication() ;
```

```
public static class ClassAuthenticator extends Authenticator  
{  
    protected PasswordAuthentication getPasswordAuthentication()  
    {  
        System.out.println("Port Requesting : " + getRequestingPort());  
        String username = "javaTpoint";  
        String password = "java";  
        return new PasswordAuthentication(username, password.toCharArray());  
    }  
}
```

PasswordAuthentication Class

BCA
6th

- **PasswordAuthentication** is a very simple final class that supports **two read-only properties**: username and password.

- **Syntax**

```
public PasswordAuthentication(String userName, char[] password)
```

- Each is **accessed** via a getter method:

```
public String getUserNames( )
```

```
public char[] getPassword( )
```

Example: PasswordAuthentication Class

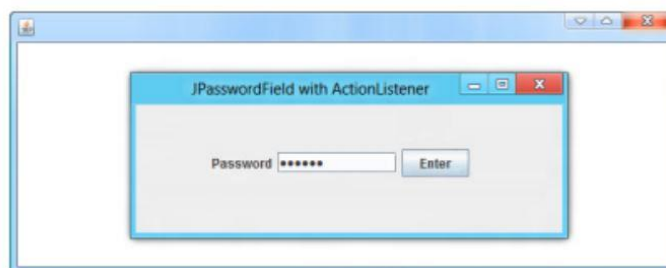
BCA
6th

```
String userName = "user";  
char[] password = { 'p', 'a', 's', 's' };  
PasswordAuthentication auth = new PasswordAuthentication(userName, password);  
System.out.println("UserName: " + auth.getUserName());  
System.out.println("Password: " + passwordAuthentication.getPassword());
```

JPasswordField CLASS

BCA
6th

- One useful tool for asking users for their passwords in a more or less secure fashion is the **JPasswordField** component from Swing:
- **public class JPasswordField** extends **JTextField**



Example: JPasswordField

BCA
6th

```
import javax.swing.*;

public class PasswordFieldExample {

    public static void main(String[] args) {
        JFrame f=new JFrame("Password Field Example");
        JPasswordField value = new JPasswordField();
        JLabel l1=new JLabel("Password:");
        l1.setBounds(20,100, 80,30);
        value.setBounds(100,100,100,30);
        f.add(value); f.add(l1);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```



The End

BCA
6th























Unit – 4 HTTP

4 HTTP

4.1 The Protocol

Keep-Alive

4.2 HTTP Methods

The Request Body

Cookies

CookieManager

CookieStore





Unit - 4

HTTP

The Hypertext Transfer Protocol (HTTP) is a standard that defines how a web client talks to a server and how data is transferred from the server back to the client.

HTTP is usually thought of as a means of transferring HTML files and the pictures embedded in them.

It can be used to transfer pictures, Microsoft Word documents, Windows .exe files, or *anything else that can be* represented in bytes.

Unit - 4

The Protocol

HTTP is the standard protocol for communication between web browsers and web servers.


- HTTP specifies how a client and server establish a connection,
- how the client requests data from the server,
- how the server responds to that request, and
- finally, how the connection is closed.

HTTP connections use the TCP/IP protocol for data transfer.



The Protocol


For each request from client to server, there is a sequence of four steps:

1. Client opens a TCP connection to the server on port 80, by default; other ports may be specified in the URL.
 2. Client sends a message to the server requesting the resource at a specified path.
 3. Server sends a response to the client.
 4. The server closes the connection.
- 



The Protocol

Each request and response has the same basic form:

- ✓ A header line
 - ✓ An HTTP header
 - ✓ Containing metadata
 - ✓ A blank line
 - ✓ A message body
- 



The Protocol

A typical client request:

GET /index.html HTTP/1.1

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:20.0)

Gecko/20100101 Firefox/20.0

Host: en.wikipedia.org

Connection: keep-alive

Accept-Language: en-US,en;

Accept-Encoding: gzip, deflate

Accept: text/html,application/xhtml+xml,application/xml



The Protocol

A typical successful response :

HTTP/1.1 200 OK

Date: Sun, 21 Apr 2013 15:12:46 GMT

Server: Apache

Connection: close

Content-Type: text/html; charset=ISO-8859-1

Content-length: 115





The Protocol

A typical successful response :

HTTP/1.1 200 OK

Date: Sun, 21 Apr 2013 15:12:46 GMT

Server: Apache

Connection: close

Content-Type: text/html; charset=ISO-8859-1

Content-length: 115

A response code :

- ✓ 100 to 199 always indicates an informational response
- ✓ 200 to 299 always indicates success
- ✓ 300 to 399 always indicates redirection
- ✓ 400 to 499 always indicates a client error
- ✓ 500 to 599 indicates a server error



Keep-Alive

HTTP 1.0 opens a new connection for each request.

The time taken to open and close all the connections in a typical web session can outweigh the time taken to transmit the data, especially for sessions with many small documents.

This is a problematic for encrypted HTTPS connections using Secure Sockets Layer & Transport Layer Security.

In HTTP 1.1 and later, the server doesn't have to close the socket after it sends its response.

A client indicates that it's willing to reuse a socket by including a *Connection field in the HTTP request header* with the value *Keep-Alive*:

Connection: Keep-Alive



Keep-Alive

We can control Java's use of HTTP Keep-Alive with several system properties:

- `http.keepAlive` to “true or false” to enable/disable. Default enable.
- `http.maxConnections` to the number of sockets. The default is 5.
- `http.keepAlive.remainingData` to true, It is false by default.
- `sun.net.http.errorstream.enableBuffering` to true, It is false by default.
- `sun.net.http.errorstream.bufferSize`, The default is 4,096 bytes.
- `sun.net.http.errorstream.timeout` , It is 300 milliseconds by default.



HTTP Methods

Communication with an HTTP server follows a request-response pattern: one stateless request followed by one stateless response.

Each HTTP request has two or three parts:

- Start line containing the HTTP method and a path to the resource.
- Header of name-value fields that provide meta-information.
- Request body containing a representation of a resource (POST and PUT only)

There are four main HTTP methods, operations that can be performed:

- GET
- POST
- PUT
- DELETE



The Request Body

The GET method retrieves a representation of a resource identified by a URL.

The exact location of the resource we want to GET from a server is specified by the various parts of the path and query string.

POST and PUT are more complex. In these cases, the client supplies the representation of the resource, in addition to the path and the query string.

The representation of the resource is sent in the body of the request, after the header. That is, it sends these four items in order:

1. A starter line including the method, path and query string, and HTTP version
2. An HTTP header
3. A blank line
4. The body

For example, this POST request sends form data to a server:

```
POST /cgi-bin/register.pl HTTP 1.0
Date: Sun, 27 Apr 2013 12:32:36
Host: www.cafeaulait.org
Content-type: application/x-www-form-urlencoded
Content-length: 54
username=Elliotte+Harold&email=elharo%40ibiblio.org
```



The Request Body

However, the HTTP header should include two fields that specify the nature of the body:

- A Content-length field that specifies how many bytes are in the body.
- A Content-type field that specifies the MIME media type of the bytes

For example, here's a PUT request that uploads an Atom document:

```
PUT /blog/software-development/the-power-of-pomodoros/ HTTP/1.1
Host: elharo.com
User-Agent: AtomMaker/1.0
Authorization: Basic ZGFmZnk6c2VjZXJldA==
Content-Type: application/atom+xml;type=entry
Content-Length: 322
```

```
<?xml version="1.0"?>
<entry xmlns="http://www.w3.org/2005/Atom">
<title>The Power of Pomodoros</title>
<id>urn:uuid:101a41a6-722b-4d9b-8afb-ccfb01d77499</id>
<updated>2013-02-22T19:40:52Z</updated>
<author><name>Elliotte Harold</name></author>
<content>I hadn't paid much attention to Pomodoro...</content>
</entry>
```

Cookies

Many websites use small strings of text known as *cookies* to store *persistent client-side* state between connections.

Cookies are passed from server to client and back again in the HTTP headers of requests and responses.

Cookies are limited to nonwhitespace ASCII text, and may not contain commas or semicolons. To set a cookie in a browser, the server includes a Set-Cookie header line in the HTTP header.

For example, this HTTP header sets the cookie “cart” to the value “ATVPDKIKX0DER”:

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: cart=ATVPDKIKX0DER
```

If a browser makes a second request to the same server, it will send the cookie back in a Cookie line in the HTTP request header like so:

```
GET /index.html HTTP/1.1
Host: www.example.org
Cookie: cart=ATVPDKIKX0DER
Accept: text/html
```

Cookies

As long as the server doesn't reuse cookies, this enables it to track individual users and sessions across multiple, otherwise stateless, HTTP connections. Servers can set more than one cookie.

For example, a request made to Amazon fed my browser five cookies:

```
Set-Cookie:skin=noskin  
Set-Cookie:ubid-main=176-5578236-9590213  
Set-Cookie:session-token=Zg6afPNqbaMv2WmYFOv57zCU1O6Ktr  
Set-Cookie:session-id-time=2082787201l  
Set-Cookie:session-id=187-4969589-3049309
```

In addition to a simple name=value pair, cookies can have several attributes that control their scope including:

- expiration date,
- path,
- domain,
- port,
- version, and
- security options

Cookies

For example, this request sets a user cookie for the entire *foo.example.com* domain:

```
Set-Cookie: user=elharo;Domain=.foo.example.com
```

```
Set-Cookie: user=elharo; Path=/restricted
```

```
Set-Cookie: user=elharo;Path=/restricted;Domain=.example.com
```

```
Cookie: user=elharo; Path=/restricted;Domain=.foo.example.com
```

```
Set-Cookie: user=elharo; expires=Wed, 21-Dec-2015 15:23:00 GMT
```

```
Set-Cookie: user="elharo"; Max-Age=3600
```

```
Set-Cookie: key=etrogl7*;Domain=.foo.example.com; secure
```

```
Set-Cookie: key=etrogl7*;Domain=.foo.example.com; secure; httponly
```




Cookies

Here's a complete set of cookies sent by Amazon:

Set-Cookie: skin=noskin; path=/; domain=.amazon.com; expires=Fri, 03-May-2013 21:46:43 GMT

Set-Cookie: ubid-main=176-5578236-9590213; path=/; domain=.amazon.com; expires=Tue, 01-Jan-2036 08:00:01 GMT

Set-Cookie: session-token=Zg6afPNqbaMv2WmYFOv57zCU1O6KtrMMdskcmllbZ
cY4q6t0PrMywqO82PR6AgtfIJhtBABhomNUW2dITwuLfOZuhXILp7Toya+
AvWaYJxpfY1lj4ci4cnJxiuUZTev1WV31p5bcwzRM1Cmn3QOCezNNqenhzZD8TZUnOL/9Ya;
path=/; domain=.amazon.com; expires=Thu, 28-Apr-2033 21:46:43 GMT

Set-Cookie: session-id-time=2082787201l; path=/; domain=.amazon.com; expires=Tue, 01-Jan-2036 08:00:01 GMT

Set-Cookie: session-id=187-4969589-3049309; path=/; domain=.amazon.com;
expires=Tue, 01-Jan-2036 08:00:01 GMT



CookieManager

Java 5 includes an abstract `java.net.CookieHandler` class that defines an API for storing and retrieving cookies.

Java 6 fleshes this out by adding a concrete `java.net.CookieManager` subclass of `CookieHandler` can be use.

Before Java will store and return cookies, you need to enable it:

```
CookieManager manager = new CookieManager();  
CookieHandler.setDefault(manager);
```

Three policies are predefined:

- `CookiePolicy.ACCEPT_ALL` All: cookies allowed
- `CookiePolicy.ACCEPT_NONE`: No cookies allowed
- `CookiePolicy.ACCEPT_ORIGINAL_SERVER`: Only first party cookies allowed



CookieManager

For example, this code fragment tells Java to block third-party cookies but accept firstparty cookies:

```
CookieManager manager = new CookieManager();  
manager.setCookiePolicy(CookiePolicy.ACCEPT_ORIGINAL_SERVER);  
CookieHandler.setDefault(manager);
```

you can implement the CookiePolicy interface yourself and override the shouldAccept() method:

```
public boolean shouldAccept(URL uri, HttpCookie cookie)
```

Example 6-1. A cookie policy that blocks all .gov cookies but allows others

```
import java.net.*;
public class NoGovernmentCookies implements CookiePolicy {
    @Override
    public boolean shouldAccept(Uri uri, HttpCookie cookie) {
        if (uri.getAuthority().toLowerCase().endsWith(".gov")
            || cookie.getDomain().toLowerCase().endsWith(".gov")) {
            return false;
        }
        return true;
    }
}
```



CookieStore

We can retrieve the store in which the CookieManager saves its cookies with the `getCookieStore()` method:

```
CookieStore store = manager.getCookieStore();
```

The `CookieStore` class allows you to add, remove, and list cookies so you can control the cookies that are sent outside the normal flow of HTTP requests and responses:

```
public void add(Uri uri, HttpCookie cookie)
public List<HttpCookie> get(Uri uri)
public List<HttpCookie> getCookies()
public List<Uri> getURIs()
public boolean remove(Uri uri, HttpCookie cookie)
public boolean removeAll()
```

Example 6-2. The HTTPCookie class

```
package java.net;  
public class HttpCookie implements Cloneable {  
    public HttpCookie(String name, String value)
```

```
    public boolean hasExpired()  
    public void setComment(String comment)  
    public String getComment()  
    public void setCommentURL(String url)  
    public String getCommentURL()  
    public void setDiscard(boolean discard)  
    public boolean getDiscard()  
    public void setPortlist(String ports)  
    public String getPortlist()  
    public void setDomain(String domain)  
    public String getDomain()  
    public void setMaxAge(long expiry)  
    public long getMaxAge()
```

```
    public void setPath(String path)  
    public String getPath()  
    public void setSecure(boolean flag)  
    public boolean getSecure()  
    public String getName()  
    public void setValue(String value)  
    public String getValue()  
    public int getVersion()  
    public void setVersion(int v)
```

```
    public static boolean domainMatches(String domain,  
    String host)  
    public static List<HttpCookie> parse(String header)
```

```
    public String toString()  
    public boolean equals(Object obj)  
    public int hashCode()  
    public Object clone()  
}
```

Unit - 4

4 HTTP

4.1 The Protocol

Keep-Alive

4.2 HTTP Methods

The Request Body

Cookies

CookieManager

CookieStore

Review unit -4
The End