# ADVANCED JAVA (CBCS)

**BCA-V SEM**

NOTES

**Prepared By**

Prof. Prabhu Kichadi, M, Tech
KLES SSMS BCA College Athani

# UNIT – V

## CONTENTS

**Networking Basics, InetAddress, TCP/IP Client-Server Socket, URLConnection, HTTPURLConnection, Datagram, Introduction To EJB, Types of EJB.**

## Networking

### Networking Basics

**Socket** : A socket identifies an endpoint in a network. Sockets are at the foundation of modern networking because a socket allows a single computer to serve many different clients at once, as well as to serve many different types of information. This is accomplished using a port, which is a numbered socket on a particular machine.

A server process is said to "listen" to a port until a client connects to it. A server is allowed to accept multiple clients connected to the same port number, although each session is unique.

To manage multiple client connections, a server process must be multithreaded or have some other means of multiplexing the simultaneous I/O. Socket communication takes place via a protocol

**Internet Protocol (IP) :** IP is a low-level routing protocol that breaks data into small packets and sends them to an address across a network, which does not guarantee to deliver said packets to the destination.

**Transmission Control Protocol (TCP)** is a higher-level protocol that manages to robustly string together these packets, sorting and retransmitting them as necessary to reliably transmit data.

**User Datagram Protocol (UDP),** sits next to TCP and can be used directly to support fast, connectionless, unreliable transport of packets.
HTTP is the protocol that web browsers and servers use to transfer hypertext pages and images. It is a quite simple protocol for a basic page-browsing web server.

### Internet Address:

A key component of the Internet is the address. Every computer on the Internet has one. An Internet address is a number that uniquely identifies each computer on the Net.

Internet addresses consisted of 32-bit values, organized as four 8-bit values. This address type was specified by IPv4 (Internet Protocol, version 4).

However, a new addressing scheme, called IPv6 (Internet Protocol, version 6) has come into play.

IPv6 uses a 128-bit value to represent an address, organized into eight 16-bit chunks.

## Domain name

Just as the numbers of an IP address describe a network hierarchy,
The name of an Internet address, called its domain name, describes a machine's location in a name space.
For example, www.osborne.com is in the COM domain (reserved for U.S. commercial sites)
It is called osborne (after the company name), and www identifies the server for web requests.
An Internet domain name is mapped to an IP address by the Domain Naming Service (DNS).
This enables users to work with domain names, but the Internet operates on IP addresses.

## The Networking Classes and Interfaces

The classes contained in the java.net package are shown here:

| | | |
|---|---|---|
| Authenticator | Inet6Address | ServerSocket |
| CacheRequest | InetAddress | Socket |
| CacheResponse | InetSocketAddress | SocketAddress |
| ContentHandler | InterfaceAddress (Added by Java SE 6.) | SocketImpl |
| CookieHandler | JarURLConnection | SocketPermission |
| CookieManager (Added by Java SE 6.) | MulticastSocket | URI |
| DatagramPacket | NetPermission | URL |
| DatagramSocket | NetworkInterface | URLClassLoader |
| DatagramSocketImpl | PasswordAuthentication | URLConnection |
| HttpCookie (Added by Java SE 6.) | Proxy | URLDecoder |
| HttpURLConnection | ProxySelector | URLEncoder |
| IDN (Added by Java SE 6.) | ResponseCache | URLStreamHandler |
| Inet4Address | SecureCacheResponse | |

The **java.net** package's interfaces are listed here:

| | | |
|---|---|---|
| ContentHandlerFactory | DatagramSocketImplFactory | SocketOptions |
| CookiePolicy (Added by Java SE 6.) | FileNameMap | URLStreamHandlerFactory |
| CookieStore (Added by Java SE 6.) | SocketImplFactory | |

## InetAddress

The InetAddress class is used to encapsulate both the numerical IP address and the domain name for that address.

You interact with this class by using the name of an IP host, which is more convenient and understandable than its IP address. The InetAddress class hides the number inside.

InetAddress can handle both IPv4 and IPv6 addresses.

## Factory Methods

The InetAddress class has no visible constructors. To create an InetAddress object, you have to use one of the available factory methods.

Factory methods are merely a convention where by static methods in a class return an instance of that class. This is done in lieu of overloading a constructor with various parameter lists when having unique method names makes the results much clearer.

**Three commonly used InetAddress factory methods are shown here:**

1. static InetAddress getLocalHost( ) throws UnknownHostException

2. static InetAddress getByName(String hostName) throws UnknownHostException

3. static InetAddress[ ] getAllByName(String hostName) throws UnknownHostException

The getLocalHost( ) method simply returns the InetAddress object that represents the local host.
The getByName( ) method returns an InetAddress for a host name passed to it. If these methods are unable to resolve the host name, they throw an UnknownHostException.
The getAllByName( ) factory method returns an array of InetAddresses that represent all of the addresses that a particular name resolves to. It will also throw an UnknownHostException if it can't resolve the name to at least one address.

InetAddress also includes the factory method getByAddress( ), which takes an IP address and returns an InetAddress object. Either an IPv4 or an IPv6 address can be used.

The following example prints the addresses and names of the local machine and two well-known Internet web sites:

```
// Demonstrate InetAddress.
import java.net.*;
class InetAddressTest
{
        public static void main(String args[]) throws UnknownHostException
        {
        InetAddress Address = InetAddress.getLocalHost();
        System.out.println(Address);
        Address = InetAddress.getByName("www.google.com");
        System.out.println(Address);
        InetAddress SW[] = InetAddress.getAllByName("www.nba.com");
        for (int i=0; i<SW.length; i++)
        {
                System.out.println(SW[i]);
        }
        }
}
```

Here is the output produced by this program. (Of course, the output you see may be slightly different.)

```
PRABHU/192.168.0.130
www.google.com/142.250.196.68
www.nba.com/23.207.133.49
```

## Instance Methods

The InetAddress class has several other methods, which can be used on the objects returned by the methods just discussed. Here are some of the more commonly used methods:

| | |
|---|---|
| boolean equals(Object *other*) | Returns **true** if this object has the same Internet address as *other*. |
| byte[ ] getAddress( ) | Returns a byte array that represents the object's IP address in network byte order. |
| String getHostAddress( ) | Returns a string that represents the host address associated with the **InetAddress** object. |
| String getHostName( ) | Returns a string that represents the host name associated with the **InetAddress** object. |
| boolean isMulticastAddress( ) | Returns **true** if this address is a multicast address. Otherwise, it returns **false**. |
| String toString( ) | Returns a string that lists the host name and the IP address for convenience. |

## Inet4Address and Inet6Address

Beginning with version 1.4, Java has included support for IPv6 addresses. Because of this, two subclasses of InetAddress were created: Inet4Address and Inet6Address.

Inet4Address represents a traditional-style IPv4 address. Inet6Address encapsulates a new-style IPv6 address. Because they are subclasses of InetAddress, an InetAddress reference can refer to either. This is one way that Java was able to add IPv6 functionality without breaking existing code or adding many more classes.

For the most part, you can simply use InetAddress when working with IP addresses because it can accommodate both styles.

## TCP/IP Client Sockets

TCP/IP sockets are used to implement reliable, bidirectional, persistent, point-to-point, stream-based connections between hosts on the Internet.
There are two kinds of TCP sockets in Java.

Servers
clients.

The ServerSocket class is designed to be a "listener," which waits for clients to connect before doing anything. Thus, ServerSocket is for servers.

The Socket class is for clients. It is designed to connect to server sockets and initiate protocol exchanges. Because client sockets are the most used by Java applications, they are examined here.

The creation of a Socket object implicitly establishes a connection between the client and server. There are no methods or constructors that explicitly expose the details of establishing that connection. Here are two constructors used to create client sockets:

| | |
|---|---|
| Socket(String *hostName*, int *port*) throws UnknownHostException, IOException | Creates a socket connected to the named host and port. |
| Socket(InetAddress *ipAddress*, int *port*) throws IOException | Creates a socket using a preexisting **InetAddress** object and a port. |

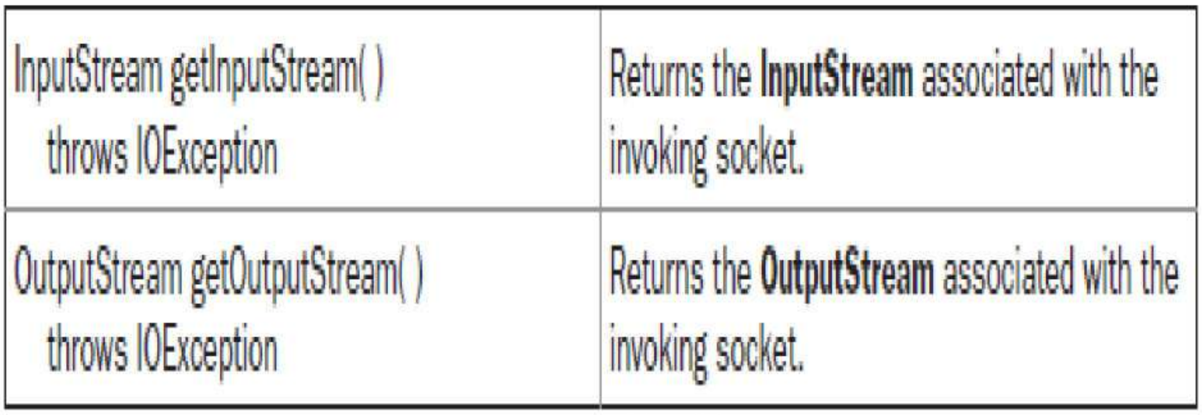
## Socket defines several instance methods.

For example, a Socket can be examined at any time for the address and port information associated with it, by use of the following methods:

| | |
|---|---|
| InetAddress getInetAddress( ) | Returns the **InetAddress** associated with the **Socket** object. It returns null if the socket is not connected. |
| int getPort( ) | Returns the remote port to which the invoking **Socket** object is connected. It returns 0 if the socket is not connected. |
| int getLocalPort( ) | Returns the local port to which the invoking **Socket** object is bound. It returns –1 if the socket is not bound. |

You can gain access to the input and output streams associated with a Socket by use of the getInputStream( ) and getOuptutStream( ) methods, as shown here.

Each can throw an IOException if the socket has been invalidated by a loss of connection.



| | |
|---|---|
| InputStream getInputStream( ) throws IOException | Returns the **InputStream** associated with the invoking socket. |
| OutputStream getOutputStream( ) throws IOException | Returns the **OutputStream** associated with the invoking socket. |

**Several other methods are available,**
1) connect( ), which allows you to specify a new connection.
2) isConnected( ) which returns true if the socket is connected to a server.
3) isBound( ), which returns true if the socket is bound to an address.
4) isClosed( ), which returns true if the socket is closed.

## Uniform Resource Locator (URL)

The Web is a loose collection of higher-level protocols and file formats, all unified in a web browser. One of the most important aspects of the Web is that Tim Berners-Lee devised a scaleable way to locate all of the resources of the Net.

Once you can reliably name anything and everything, it becomes a very powerful paradigm. The Uniform Resource Locator (URL) does exactly that.

The URL provides a reasonably intelligible form to uniquely identify or address information on the Internet.

URLs are everywhere, every browser uses them to identify information on the Web. Within Java's network class library,

All URLs share the same basic format Here are two examples: http://www.osborne.com/ and http://www.osborne.com:80/index.htm.

A URL specification is based on four components.
**Protocol:** to use, separated from the rest of the locator by a colon (:). Common protocols are HTTP, FTP,
**Host name or IP address of the host to use:** this is delimited on the left by double slashes (//) and on the right by a slash (/) or optionally a colon (:).
**Port number:** is an optional parameter, delimited on the left from the host name by a colon (:) and on the right by a slash (/).
**File path.** Most HTTP servers will append a file named index.html or index.htm to URLs that refer directly to a directory resource.
Thus, http://www.osborne.com/ is the same as http://www.osborne.com/index.htm.

URL Connection is a general-purpose class for accessing the attributes of a remote resource.

Once you make a connection to a remote server, you can use URL Connection to inspect the properties of the remote object before actually transporting it locally.

These attributes are exposed by the HTTP protocol specification and, as such, only make sense for URL objects that are using the HTTP protocol.
URLConnection defines several methods. Here is a sampling:

| | |
|---|---|
| int getContentLength( ) | Returns the size in bytes of the content associated with the resource. If the length is unavailable, –1 is returned. |
| String getContentType( ) | Returns the type of content found in the resource. This is the value of the **content-type** header field. Returns null if the content type is not available. |
| long getDate( ) | Returns the time and date of the response represented in terms of milliseconds since January 1, 1970 GMT. |
| long getExpiration( ) | Returns the expiration time and date of the resource represented in terms of milliseconds since January 1, 1970 GMT. Zero is returned if the expiration date is unavailable. |
| String getHeaderField(int *idx*) | Returns the value of the header field at index *idx*. (Header field indexes begin at 0.) Returns null if the value of *idx* exceeds the number of fields. |
| String getHeaderField(String *fieldName*) | Returns the value of header field whose name is specified by *fieldName*. Returns null if the specified name is not found. |
| String getHeaderFieldKey(int *idx*) | Returns the header field key at index *idx*. (Header field indexes begin at 0.) Returns null if the value of *idx* exceeds the number of fields. |
| Map<String, List<String>> getHeaderFields( ) | Returns a map that contains all of the header fields and values. |
| long getLastModified( ) | Returns the time and date, represented in terms of milliseconds since January 1, 1970 GMT, of the last modification of the resource. Zero is returned if the last-modified date is unavailable. |
| InputStream getInputStream( ) throws IOException | Returns an **InputStream** that is linked to the resource. This stream can be used to obtain the content of the resource. |

```
// Demonstrate URLConnection.
      import java.io.*;
      import java.net.*;
      import java.util.Date;


      public class InetAddressDemo
      {

            public static void main(String args[]) throws IOException
            {
                  int c;
```

```
                URL hp = new URL("http://www.internic.net");
                URLConnection hpCon = hp.openConnection();
                // get date
                        long d = hpCon.getDate();
                        if(d==0)
                        System.out.println("No date information.");
                                else
                        System.out.println("Date: " + new Date(d));
                // get content type
                System.out.println("Content-Type:                " +
hpCon.getContentType());
                // get expiration date
                d = hpCon.getExpiration();
                if(d==0)
                System.out.println("No expiration information.");
                else
                System.out.println("Expires: " + new Date(d));
                // get last-modified date
                d = hpCon.getLastModified();
                if(d==0)
                {
                        System.out.println("No                last-modified
information.");
                }
                else
                {
                        System.out.println("Last-Modified:    "    +    new
Date(d));
                }
        }
    }
```

**Output**

```
Date: Tue Jan 31 13:30:11 IST 2023
Content-Type: text/html; charset=UTF-8
Expires: Tue Feb 28 13:30:11 IST 2023
```

```
Last-Modified: Sun Jun 11 05:31:00 IST 2017
```

Notice that URLConnection defines several methods that handle header information.

A header consists of pairs of keys and values represented as strings.

By using getHeaderField( ), you can obtain the value associated with a header key.

By calling getHeaderFields( ), you can obtain a map that contains all of the headers.

Several standard header fields are available directly through methods such as getDate( ) and getContentType( ).

## HttpURLConnection

Java provides a subclass of URLConnection that provides support for HTTP connections. This class is called HttpURLConnection. You obtain an HttpURLConnection in the same way just shown, by calling openConnection( ) on a URL object, but you must cast the result to HttpURLConnection.

Once you have obtained a reference to an HttpURLConnection object, you can use any of the methods inherited from URLConnection.

| | |
|---|---|
| static boolean getFollowRedirects( ) | Returns **true** if redirects are automatically followed and **false** otherwise. This feature is on by default. |
| String getRequestMethod( ) | Returns a string representing how URL requests are made. The default is GET. Other options, such as POST, are available. |
| int getResponseCode( ) throws IOException | Returns the HTTP response code. –1 is returned if no response code can be obtained. An **IOException** is thrown if the connection fails. |
| String getResponseMessage( ) throws IOException | Returns the response message associated with the response code. Returns null if no message is available. An **IOException** is thrown if the connection fails. |
| static void setFollowRedirects(boolean *how*) | If *how* is **true**, then redirects are automatically followed. If how is **false**, redirects are not automatically followed. By default, redirects are automatically followed. |
| void setRequestMethod(String *how*) throws ProtocolException | Sets the method by which HTTP requests are made to that specified by *how*. The default method is GET, but other options, such as POST, are available. If *how* is invalid, a **ProtocolException** is thrown. |

## The URI Class

A relatively recent addition to Java is the URI class, which encapsulates a Uniform Resource Identifier (URI). URIs are similar to URLs. In fact, URLs constitute a subset of URIs. A URI represents a standard way to identify a resource. A URL also describes how to access the
resource.

## Cookies

The java.net package includes classes and interfaces that help manage cookies and can be used to create a stateful (as opposed to stateless) HTTP session. The classes are CookieHandler, CookieManager, and HttpCookie. The interfaces are CookiePolicy and CookieStore. All but CookieHandler was added by Java SE 6.

## Datagrams

TCP/IP-style networking is appropriate for most networking needs. It provides a serialized, predictable, reliable stream of packet data. This is not without its cost, however. TCP includes many complicated algorithms for dealing with congestion control on crowded networks, as well as pessimistic expectations about packet loss. This leads to a somewhat inefficient way to transport data. Datagrams provide an alternative.

Datagrams are bundles of information passed between machines. They are some what like a hard throw from a well-trained but blindfolded catcher to the third baseman. Once the datagram has been released to its intended target, there is no assurance that it will arrive or even that someone will be there to catch it. Likewise, when the datagram is received, there is no assurance that it hasn't been damaged in transit or that whoever sent it is still there to receive a response.

Java implements datagram's on top of the UDP protocol by using two classes:

DatagramPacket object is the data container,

DatagramSocket is the mechanism used to send or receive the Datagram Packets.

## DatagramSocket

DatagramSocket defines four public constructors. They are shown here:

DatagramSocket( ) throws SocketException

DatagramSocket(int port) throws SocketException

DatagramSocket(int port, InetAddress ipAddress) throws SocketException

DatagramSocket(SocketAddress address) throws SocketException

The first creates a DatagramSocket bound to any unused port on the local computer.

The second creates a DatagramSocket bound to the port specified by port.

The third constructs a DatagramSocket bound to the specified port and InetAddress.

The fourth constructs a DatagramSocket bound to the specified SocketAddress.

## DatagramSocket defines many methods.

**send( )**

**receive( ),**

**void send(DatagramPacket packet) throws IOException**

**void receive(DatagramPacket packet) throws IOException**

The send( ) method sends packet to the port specified by packet.

The receive method waits for a packet to be received from the port specified by packet and returns the result.

## DatagramPacket

DatagramPacket defines several constructors. Four are shown here:

DatagramPacket(byte data[ ], int size)
DatagramPacket(byte data[ ], int offset, int size)
DatagramPacket(byte data[ ], int size, InetAddress ipAddress, int port)
DatagramPacket(byte data[ ], int offset, int size, InetAddress ipAddress, int port)

The first constructor specifies a buffer that will receive data and the size of a packet. It is used for receiving data over a DatagramSocket.

The second form allows you to specify an offset into the buffer at which data will be stored.

The third form specifies a target address and port, which are used by a DatagramSocket to determine where the data in the packet will be sent.

The fourth form transmits packets beginning at the specified offset into the data

## Introduction to EJB

## Types of EJB

## Advanced Java Question Bank & Assignment No - 4

## UNIT-V

## Submission Due Date: 06/02/2023

## 2 Marks

1. **Define IP and TCP**
2. **Define InetAddress**
3. **Define Inet4Address and Inet6Address**
4. **Define TCP/IP client sockets**
5. **Define URL**
6. **Define URL class**
7. **Define TCP/IP socket**
8. **Define datagram socket**
9. **Define RMI**
10. **Which package contains classes and interfaces for networking ?List some classes**
11. **What is socket? Mention two methods used for sockets**
12. **Which package contains classes and interfaces for networking?List some classes**
13. **What is DNS?**
14. **Using which method can you get the IP address of www.google.com**
15. **Define EJB?**
16. **List the types of EJB?**
17. **List the interfaces of EJB?**
18. **What is JAR file?**

## 5/10 Marks

1. **Explain various URL connection methods**
2. **What is use of URL class ?Explain with its basic form and constructor**
3. **Explain different HttpURLConnection methods.**
4. **Write a datagram example**

5. **Explain RMI concept**
6. **Explain socket and server socket**
7. **Explain the steps followed before running a client in RMI**
8. **Explain Inetaddress class and its commonly used factory methods**
9. **Explain the types of EJB?**
10. **Explain the types of EJB Interfaces?**
11. **Explain Entity Java Bean?**
12. **Explain Session Java Bean and Message Driven Java Bean?**
13. **Write a short note on JAR file?**
14. **Write short notes on a)Datagram socket b)DatagramPacket**

**Note: Refer all advanced java Lab journal programs on this unit.**