JAVA PROGRAMMING

Week 10: Networking

Lecturer:

- HO Tuan Thanh, M.Sc.

Plan

1. Networking basics
2. java.net networking Classes and Interfaces
3. InetAddress class
4. Datagrams
5. Introducing java.net.http

Plan

1. Networking basics
2. java.net networking Classes and Interfaces
3. InetAddress class
4. Datagrams
5. Introducing java.net.http

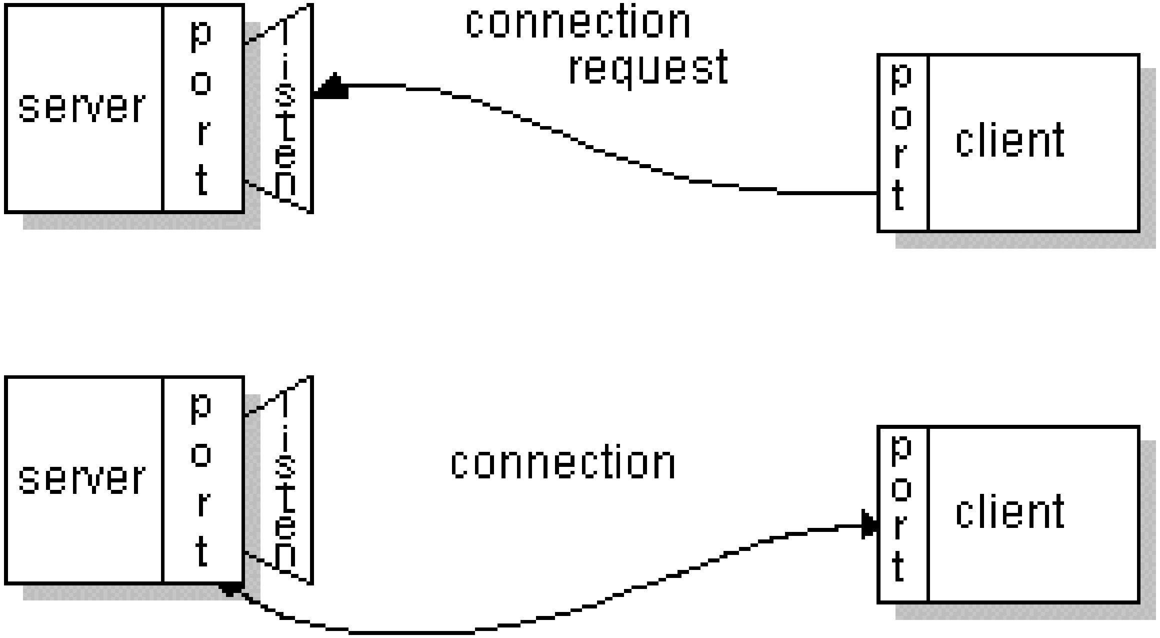
Introduction

- Since its beginning, Java has been associated with Internet programming.
    - generate secure, crossplatform, portable code
- Network programming classes defined in the java.net package.
    - Provide a convenient means by which programmers of all skill levels can access network resources.
    - From JDK 11: Java has also provided enhanced networking support for HTTP clients in the java.net.http package
    - Called the HTTP Client API, it further solidifies Java's networking capabilities.
- Networking is a very large and at times complicated topic → focus on several of its core classes and interfaces.

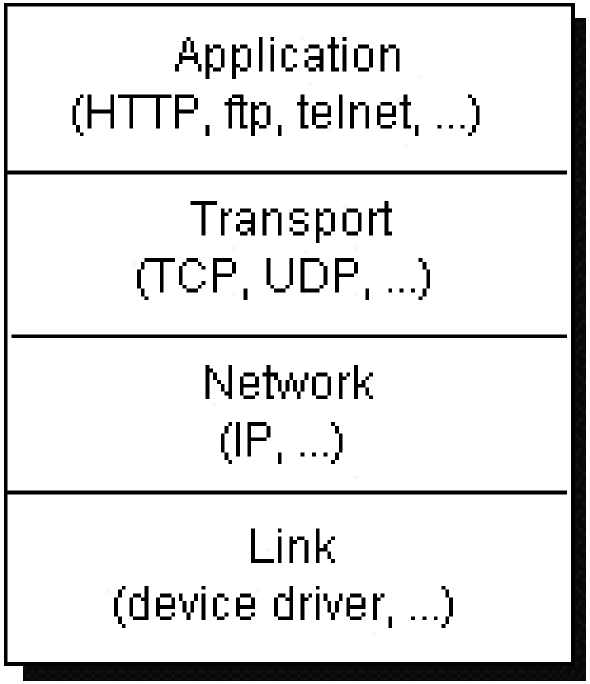
Networking Basics

- A **socket** identifies an end-point in a network.
- Sockets are at the foundation of modern networking.
  - This is accomplished through the use of 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.

Socket


server
port
listen
connection
request
port
client
server
port
listen
connection
port
client

- When you write Java programs that communicate over the network, you are programming at the application layer.
  - Don't need to concern yourself with the TCP and UDP layers.
  - Use the classes in the java.net package.



Application
(HTTP, ftp, telnet, ...)

Transport
(TCP, UDP, ...)

Network
(IP, ...)

Link
(device driver, ...)

Protocols

- **Internet Protocol** (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.

Ports

- Once a connection has been established: a higher-level protocol ensues, which is dependent on which port you are using.
- TCP/IP reserves the lower 1024 ports for specific protocols.
  - Many of these will seem familiar to you if you have spent any time surfing the Internet.
  - 21 is for FTP;
  - 23 is for Telnet;
  - 25 is for email;
  - 43 is for whois;
  - 80 is for HTTP;
  - 119 is for netnews...
  - It is up to each protocol to determine how a client should interact with the port.

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.
- Originally, all **Internet addresses** consisted of 32-bit values, organized as four 8-bit values.
  - This address type was specified by IPv4 (Internet Protocol, version 4).
- A new addressing scheme, called IPv6 (Internet Protocol, version 6) uses a 128-bit value to represent an address, organized into eight 16-bit chunks.
  - Supports a much larger address space than does IPv4.
  - When using Java: do not need to worry about whether IPv4 or IPv6 addresses are used because Java handles the details.

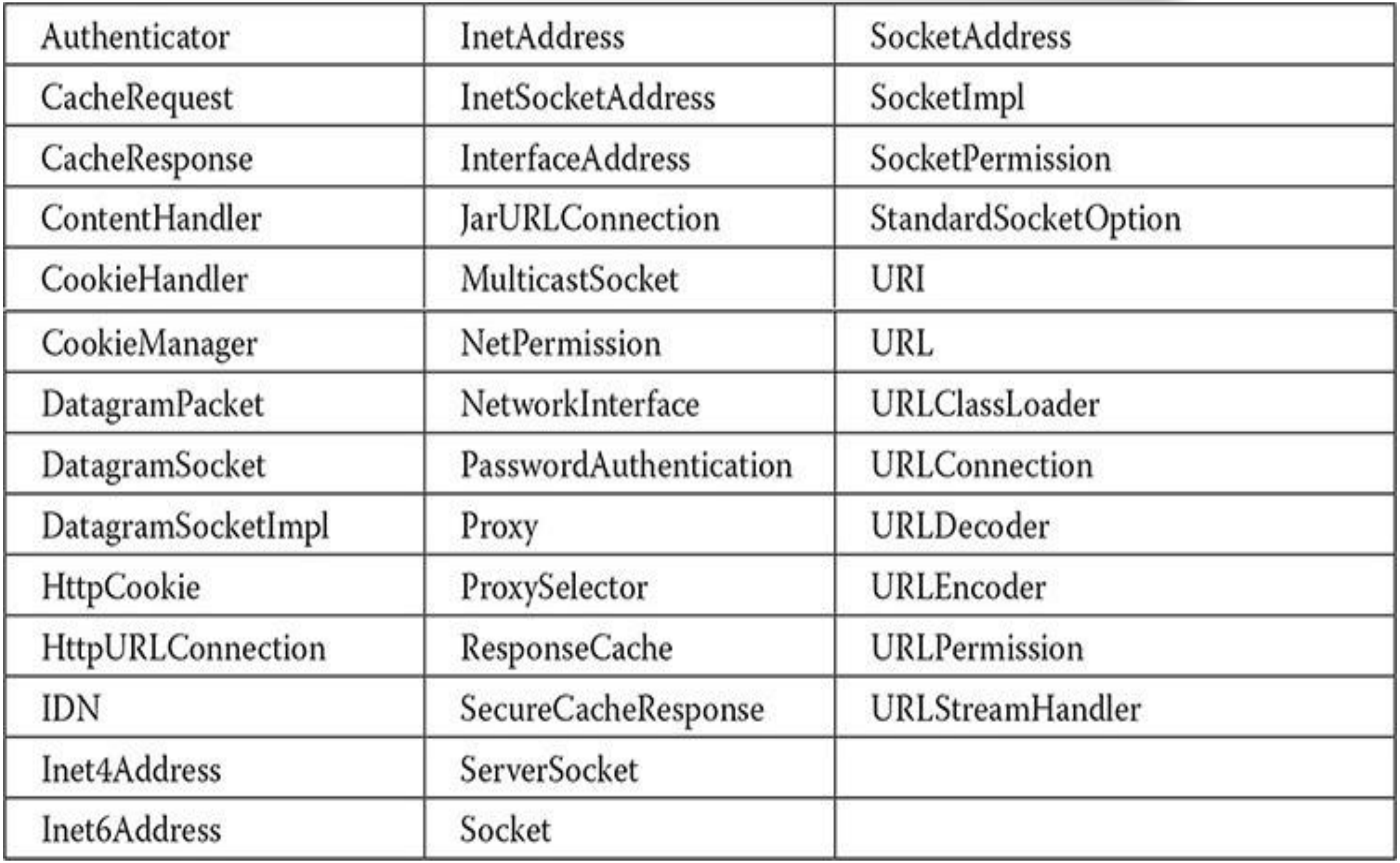
DNS

- 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.
- 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.

Plan

1. Networking basics
2. java.net networking Classes and Interfaces
3. InetAddress class
4. Datagrams
5. Introducing java.net.http

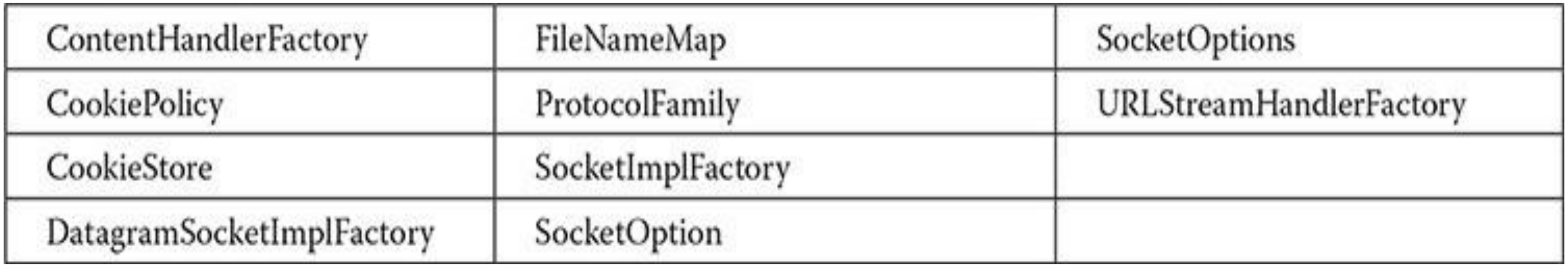
java.net classes



Authenticator	InetAddress	SocketAddress
CacheRequest	InetSocketAddress	SocketImpl
CacheResponse	InterfaceAddress	SocketPermission
ContentHandler	JarURLConnection	StandardSocketOption
CookieHandler	MulticastSocket	URI
CookieManager	NetPermission	URL
DatagramPacket	NetworkInterface	URLClassLoader
DatagramSocket	PasswordAuthentication	URLConnection
DatagramSocketImpl	Proxy	URLDecoder
HttpCookie	ProxySelector	URLEncoder
HttpURLConnection	ResponseCache	URLPermission
IDN	SecureCacheResponse	URLStreamHandler
Inet4Address	ServerSocket	
Inet6Address	Socket	

java.net package’s interfaces

- From JDK 9: java.net is part of the java.base module.



ContentHandlerFactory	FileNameMap	SocketOptions
CookiePolicy	ProtocolFamily	URLStreamHandlerFactory
CookieStore	SocketImplFactory	
DatagramSocketImplFactory	SocketOption	

Plan

1. Networking basics
2. java.net networking Classes and Interfaces
3. InetAddress class
4. Datagrams
5. Introducing java.net.http

InetAddress class

- Encapsulate both the numerical IP address and the domain name for that address.
- Interact with this class by using the name of an IP host
  - More convenient and understandable than its IP address.
- Hides the number inside.
- Handle both IPv4 and IPv6 addresses.

Factory Methods

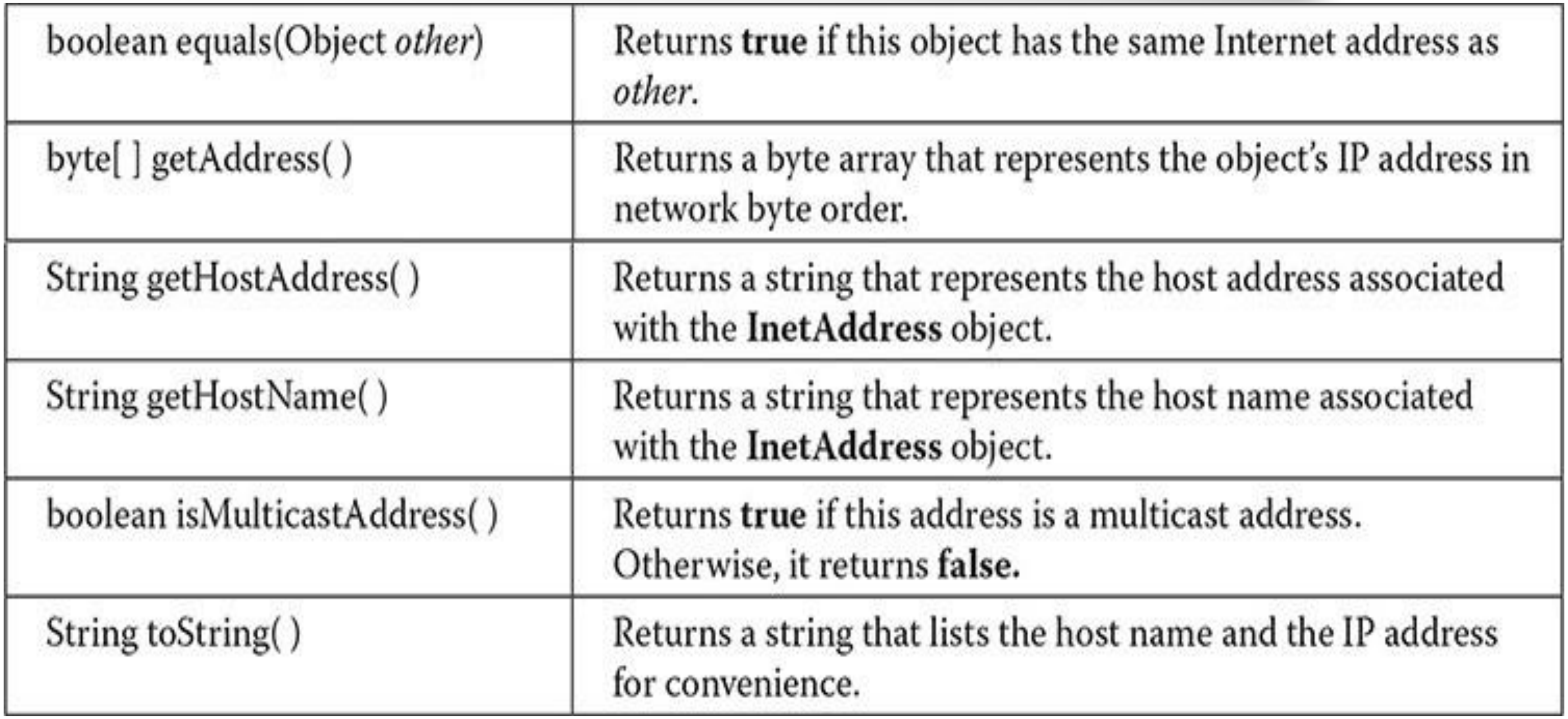
- No visible constructors.
- To create an InetAddress object: use one of the available factory methods:

```
static InetAddress getLocalHost()
                              throws UnknownHostException

static InetAddress getByName(String hostName)
                              throws UnknownHostException

static InetAddress[] getAllByName(String hostName)
                              throws UnknownHostException
```

Example

```
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.vnexpress.net");
        System.out.println(address);
        InetAddress SW[] =
                InetAddress.getAllByName("www.hcmus.edu.vn");
        for (int i = 0; i < SW.length; i++)
            System.out.println(SW[i]);
    }
}
```

Instance 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

- Two subclasses of InetAddress were created: Inet4Address and Inet6Address.
  - Inet4Address represents a traditionalstyle IPv4 address.
  - Inet6Address encapsulates a newer IPv6 address.
- They are subclasses of InetAddress → an InetAddress reference can refer to either.

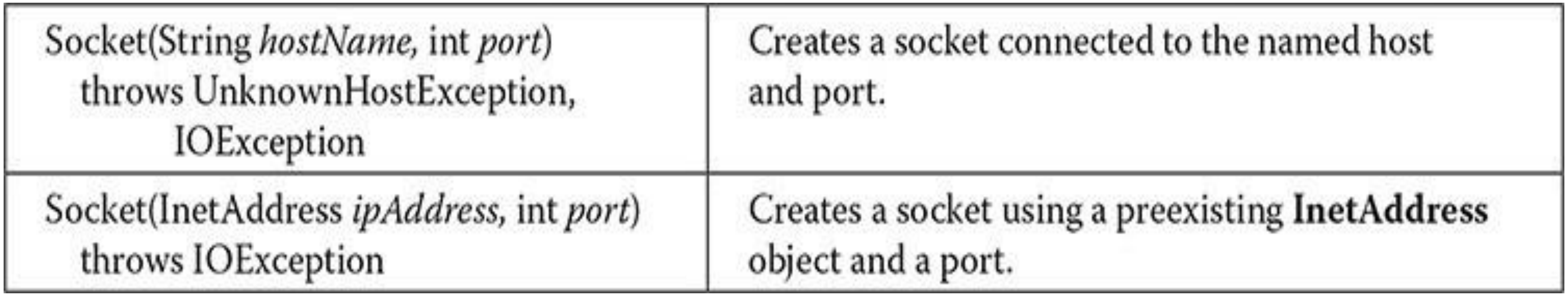
  → Java was able to add IPv6 functionality without breaking existing code or adding many more classes.
  - For the most part, you can simply use InetAddress when working with IP addresses because it can accommodate both styles.

TCP/IP Client Sockets [1]

- TCP/IP sockets are used to implement reliable, bi-directional, persistent, point-to-point, stream-based connections between hosts on the Internet.
- A socket can be used to connect Java's I/O system to other programs that may reside either on the local machine or on any other machine on the Internet, subject to security constraints.
- There are two kinds of TCP sockets in Java: One is for servers, and the other is for clients.
- The ServerSocket class is designed to be a "listener," which waits for clients to connect before doing anything → ServerSocket is for servers.

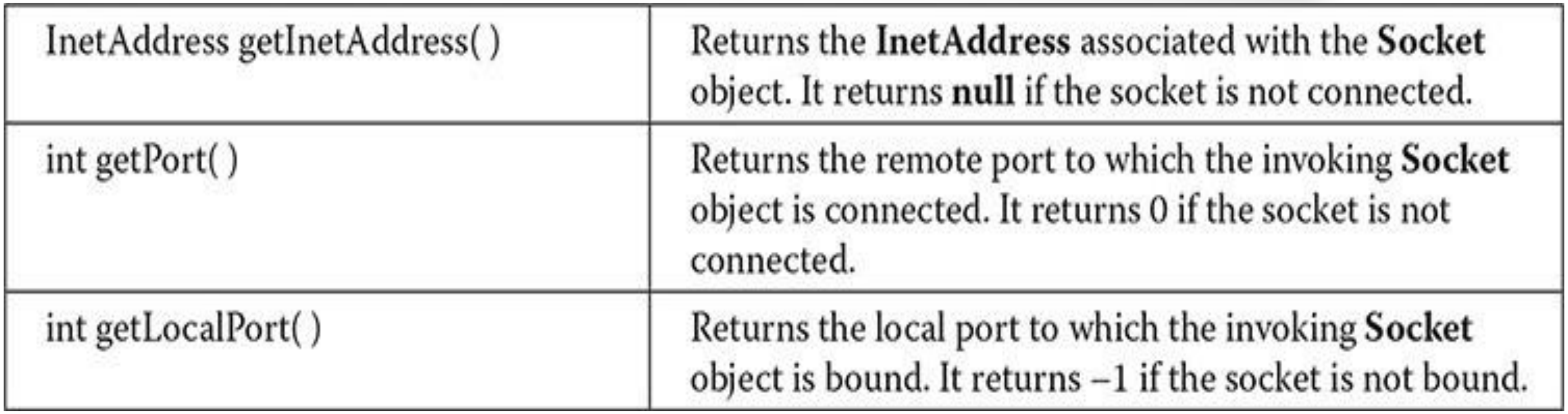
TCP/IP Client Sockets [2]

- The Socket class is for clients.
  - It is designed to connect to server sockets and initiate protocol exchanges.
- 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.
- Constructors used to create client sockets:

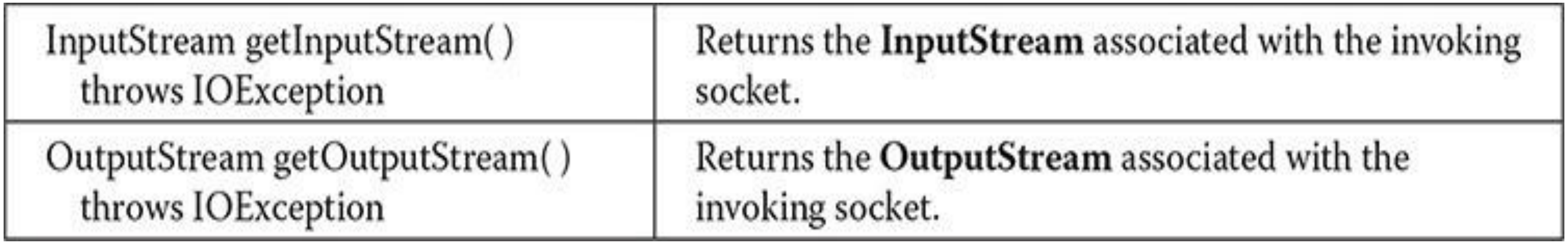


Socket(String *hostName*, int *port*) throws UnknownHostException, IOException	Creates a socket connected to the named host and port.
Socket(InetAddress *ipAddress*, int *port*) throws IOException	Creates a socket using a preexisting **InetAddress** object and a port.

Methods defined by Socket



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.



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

Example [1]

```
class Whois {
    public static void main(String args[])
                                              throws Exception {
        int c;
        //Create a socket connected to internic.net, port 43.
        Socket s = new Socket("whois.internic.net", 43);
        // Obtain input and output streams.
        InputStream in = s.getInputStream();
        OutputStream out = s.getOutputStream();
        // Construct a request string.
        String str = (args.length == 0
                ? "MHProfessional.com" : args[0]) + "\n";
```

Example [2]

```
        byte buf[] = str.getBytes(); // Convert to bytes.
        // Send request.
        out.write(buf);
        // Read and display response.
        while ((c = in.read()) != -1) {
            System.out.print((char) c);
        }
        s.close();
    }
}
```

with try-with-resources

```java
try(Socket s = new Socket("whois.internic.net", 43)){
        InputStream in = s.getInputStream();
        OutputStream out = s.getOutputStream();
        String str = (args.length == 0
                                ? "MHProfessional.com" : args[0]) + "\n";
        byte buf[] = str.getBytes();
        out.write(buf);
        while ((c = in.read()) != -1) {
                System.out.print((char) c);
        }
}
```

URL

- The Web is a loose collection of higher-level protocols and file formats, all unified in a web browser.
- The URL provides a reasonably intelligible form to uniquely identify or address information on the Internet.
- All URLs share the same basic format, although some variation is allowed. Example:
  - http://www.HerbSchildt.com
http://www.HerbSchildt.com:80/index.htm
- A URL specification is based on four components
  - The protocol
  - The host name or IP address of the host
  - The port number,
  - The actual file path

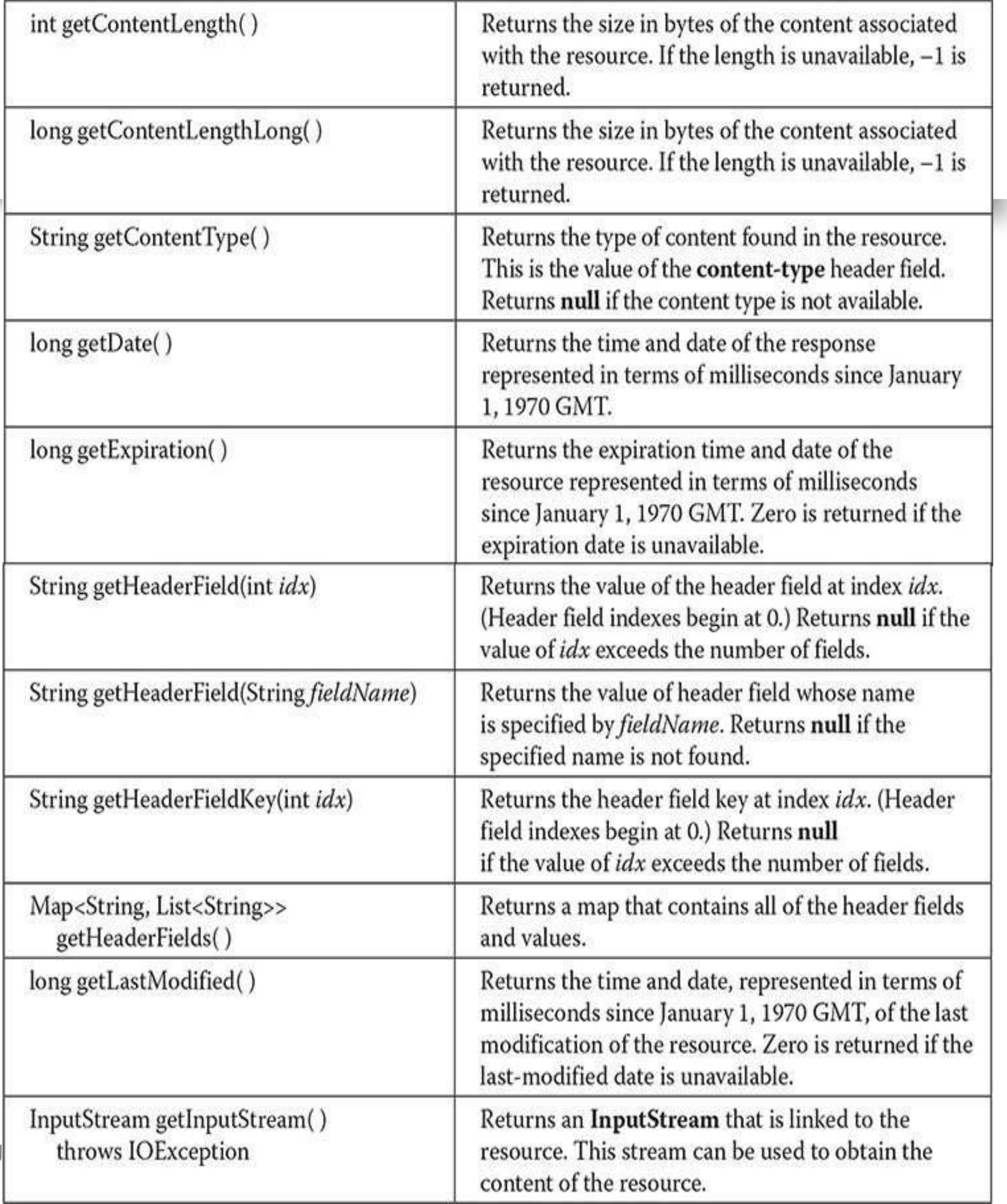
URL constructors

```
URL(String urlSpecifier)
                          throws MalformedURLException

URL(String protocolName, String hostName,
                          int port, String path )
                          throws MalformedURLException

URL(String protocolName, String hostName,
                          String path)
                          throws MalformedURLException

URL(URL urlObj, String urlSpecifier)
                          throws MalformedURLException
```

```
//Demonstrate URL.
import java.net.*;
class URLDemo {
        public static void main(String args[])
                                                        throws MalformedURLException {
        URL hp = new URL("http://www.HerbSchildt.com/WhatsNew");
                System.out.println("Protocol: " + hp.getProtocol());
                System.out.println("Port: " + hp.getPort());
                System.out.println("Host: " + hp.getHost());
                System.out.println("File: " + hp.getFile());
                System.out.println("Ext:" + hp.toExternalForm());
        }
}
```

URLConnection

- General-purpose class for accessing the attributes of a remote resource.
- Once you make a connection to a remote server: use URLConnection to inspect the properties of the remote object before actually transporting it locally.
- These attributes are exposed by the HTTP protocol specification and, only make sense for URL objects that are using the HTTP protocol.



int getContentLength( )	Returns the size in bytes of the content associated with the resource. If the length is unavailable, –1 is returned.
long getContentLengthLong( )	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.

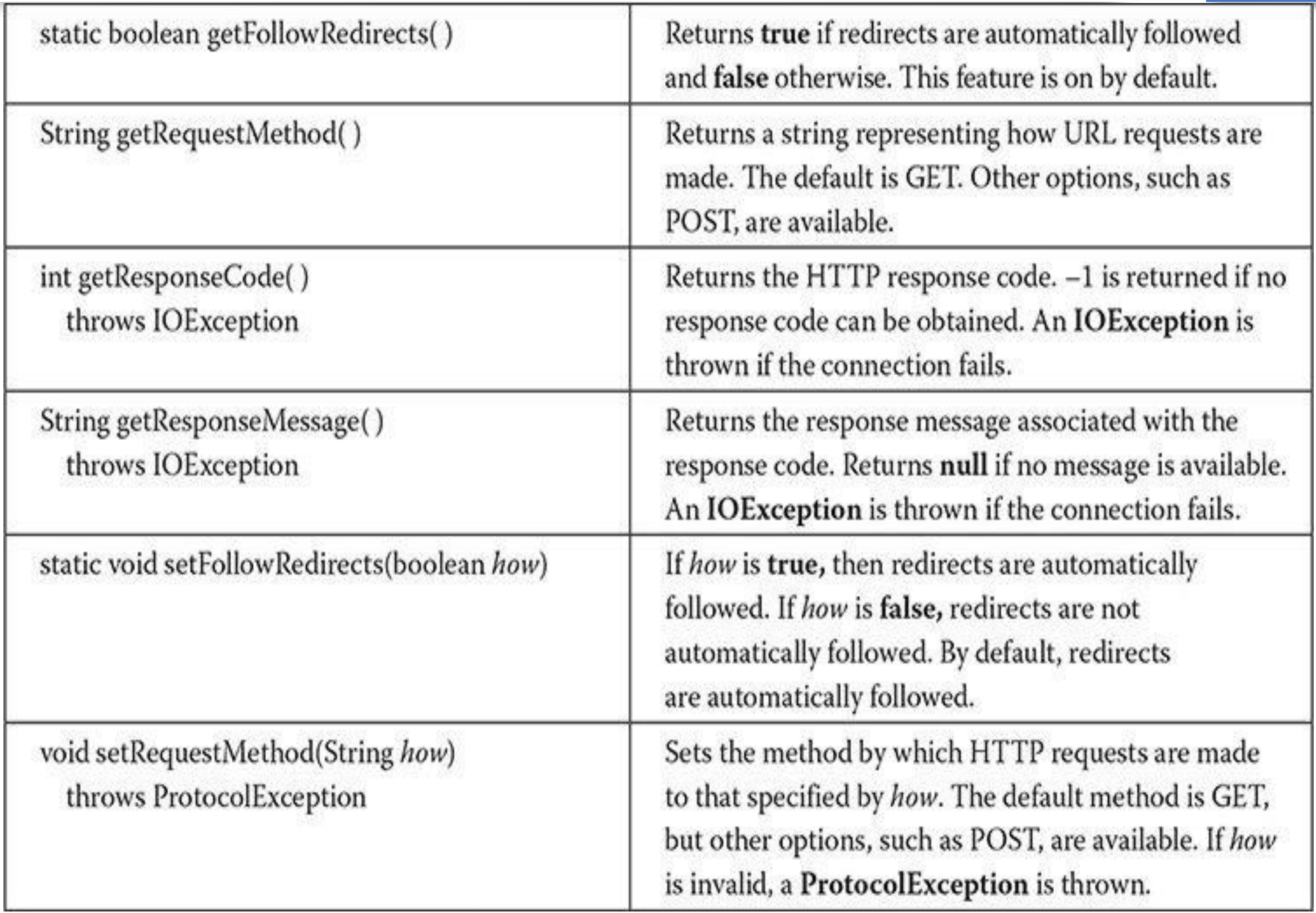
```
class UCDemo {
    public static void main(String args[])
                                        throws Exception {
        int c; URL hp = new URL("http://www.internic.net");
        URLConnection hpCon = hp.openConnection();
        long d = hpCon.getDate(); // get date
        if (d == 0)
            System.out.println("No date information.");
        else System.out.println("Date: " + new Date(d));
        // get content type
        System.out.println("Content-Type: " +
                                        hpCon.getContentType());
        d = hpCon.getExpiration(); // get expiration date
        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));
// get content length
long len = hpCon.getContentLengthLong();
if (len == -1)
        System.out.println("Content length unavailable.");
else
        System.out.println("Content-Length: " + len);
        if (len != 0) {
            System.out.println("=== Content ===");
            InputStream input = hpCon.getInputStream();
            while (((c = input.read()) != -1)) {
                System.out.print((char) c);
            }
            input.close();
        } else {
            System.out.println("No content available.");
        }
    }
}
```

HttpURLConnection

- HttpURLConnection is a subclass of URLConnection that provides support for HTTP connections.
- To obtain an HttpURLConnection:
  - by calling openConnection( ) on a URL object, but you must cast the result to HttpURLConnection.
  - Once obtained a reference to an HttpURLConnection object: use any of the methods inherited from URLConnection.
  - Can also use any of the several methods defined by HttpURLConnection.

HttpURLConnection methods



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.

Example [1]

```
class HttpURLDemo {
    public static void main(String args[]) throws Exception {
        URL hp = new URL("http://www.google.com");
        HttpURLConnection hpCon =
                    (HttpURLConnection) hp.openConnection();
        // Display request method.
        System.out.println("Request method is " +
                    hpCon.getRequestMethod());
        // Display response code.
        System.out.println("Response code is " +
                    hpCon.getResponseCode());
        // Display response message.
        System.out.println("Response Message is " +
                    hpCon.getResponseMessage());
```

Example [2]

```
        // Get a list of the header fields and a set
        // of the header keys.
        Map<String, List<String>> hdrMap =
                                            hpCon.getHeaderFields();
        Set<String> hdrField = hdrMap.keySet();
        System.out.println("\nHere is the header:");
        // Display all header keys and values..
        for (String k : hdrField) {
        System.out.println("Key: " + k + "  Value: " +
                                                    hdrMap.get(k));
        }
    }
}
```

URI Class

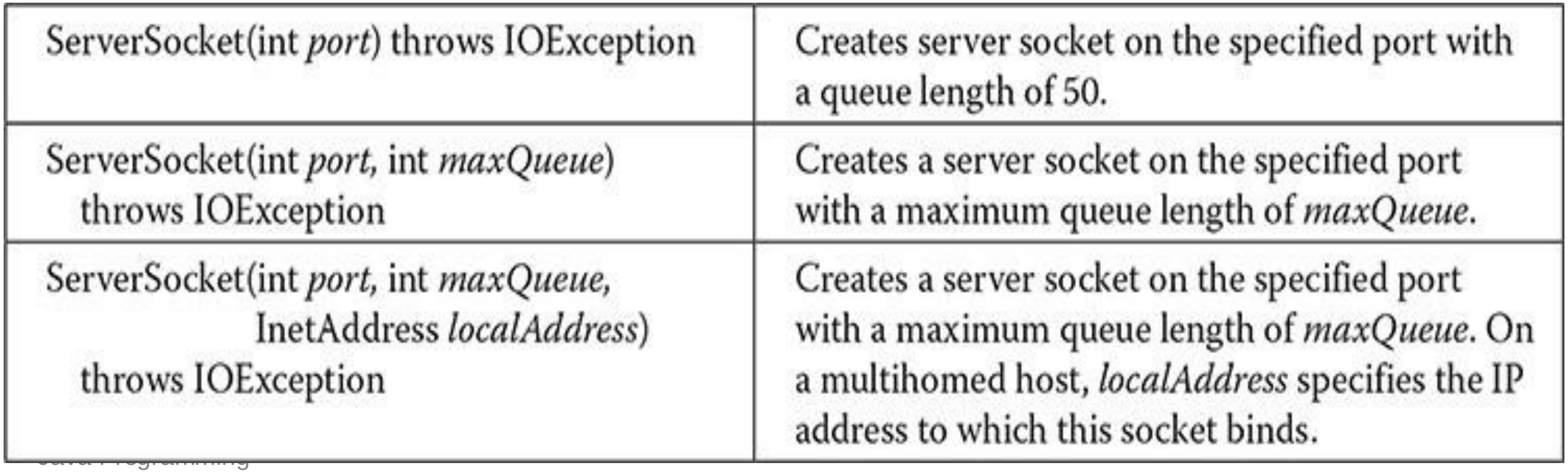
- Encapsulates a Uniform Resource Identifier (URI).
- Are similar to URLs.
  - URLs constitute a subset of URIs.
- Represents a standard way to identify a resource.
- A URL also describes how to access the resource.

TCP/IP Server Sockets [1]

- Java has a different socket class that must be used for creating server applications.
- The ServerSocket class is used to create servers that listen for either local or remote client programs to connect to them on published ports.
- ServerSockets are quite different from normal Sockets.

TCP/IP Server Sockets [2]

- When you create a ServerSocket:
  - it will register itself with the system as having an interest in client connections.
  - The constructors for ServerSocket reflect the port number that you want to accept connections on and, optionally, how long you want the queue for said port to be.
  - The queue length tells the system how many client connections it can leave pending before it should simply refuse connections. The default is 50.



ServerSocket(int *port*) throws IOException	Creates server socket on the specified port with a queue length of 50.
ServerSocket(int *port*, int *maxQueue*) throws IOException	Creates a server socket on the specified port with a maximum queue length of *maxQueue*.
ServerSocket(int *port*, int *maxQueue*, InetAddress *localAddress*) throws IOException	Creates a server socket on the specified port with a maximum queue length of *maxQueue*. On a multihomed host, *localAddress* specifies the IP address to which this socket binds.

- ServerSocket has the method accept()
  - Is a blocking call that will wait for a client to initiate communications and
  - return with a normal Socket that is then used for communication with the client.

Plan

1. Networking basics
2. java.net networking Classes and Interfaces
3. InetAddress class
4. Datagrams
5. Introducing java.net.http

Datagrams [1]

- TCP/IP-style networking is appropriate for most networking needs.
  - It provides a serialized, predictable, reliable stream of packet data.
  - 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.
  - 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.

Datagrams [2]

- Java implements datagrams 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 DatagramPackets.

DatagramSocket

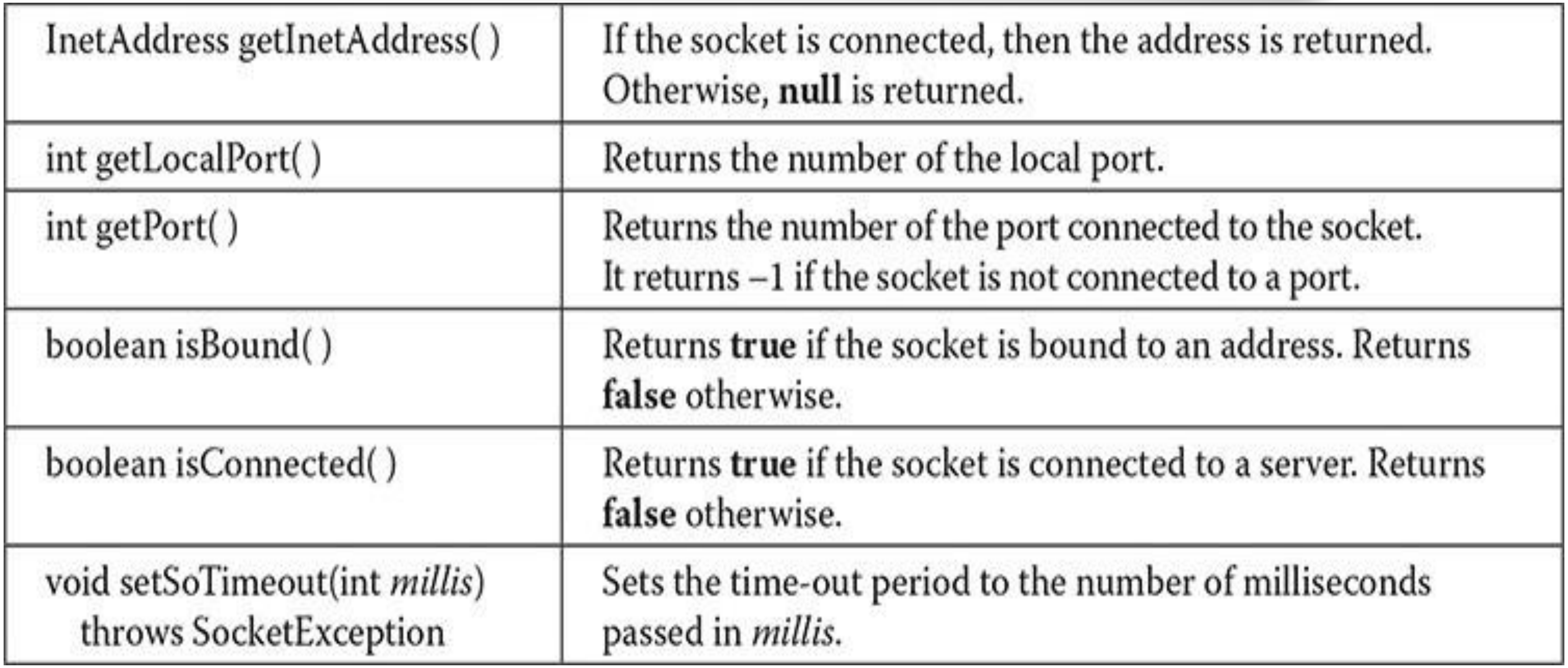
- Constructors:

```
DatagramSocket( ) throws SocketException
DatagramSocket(int port)
                            throws SocketException
DatagramSocket(int port, InetAddress ipAddress)
                            throws SocketException
DatagramSocket(SocketAddress address)
                            throws SocketException
```

- Two important methods

```
void send(DatagramPacket packet)
                                    throws IOException
void receive(DatagramPacket packet)
                                    throws IOException
```

Methods of DatagramSocket



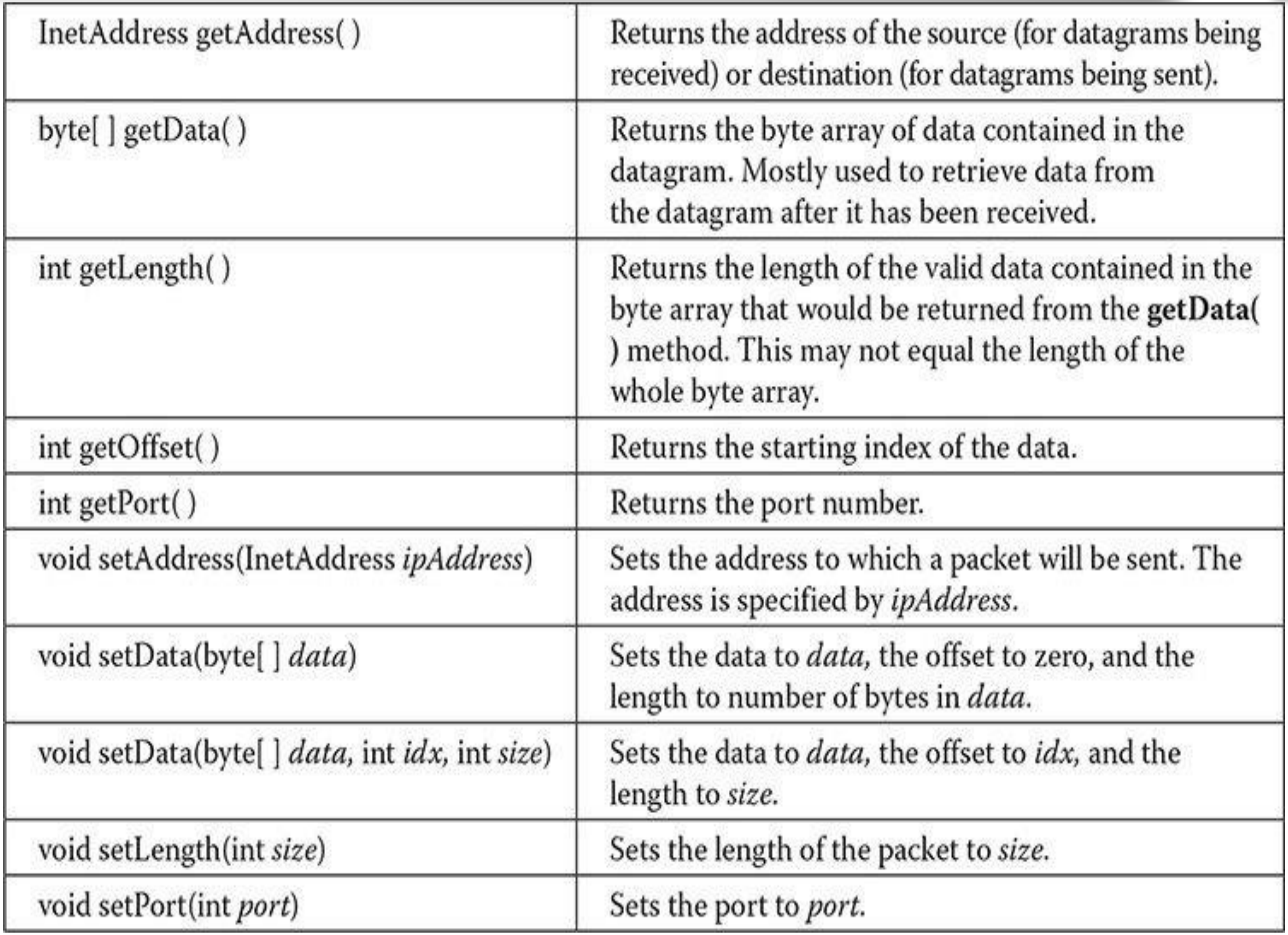
InetAddress getInetAddress( )	If the socket is connected, then the address is returned. Otherwise, **null** is returned.
int getLocalPort( )	Returns the number of the local port.
int getPort( )	Returns the number of the port connected to the socket. It returns –1 if the socket is not connected to a port.
boolean isBound( )	Returns **true** if the socket is bound to an address. Returns **false** otherwise.
boolean isConnected( )	Returns **true** if the socket is connected to a server. Returns **false** otherwise.
void setSoTimeout(int *millis*) throws SocketException	Sets the time-out period to the number of milliseconds passed in *millis*.

DatagramPacket

- Constructors:

```
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)
```

DatagramPacket methods



InetAddress getAddress( )	Returns the address of the source (for datagrams being received) or destination (for datagrams being sent).
byte[ ] getData( )	Returns the byte array of data contained in the datagram. Mostly used to retrieve data from the datagram after it has been received.
int getLength( )	Returns the length of the valid data contained in the byte array that would be returned from the **getData( )** method. This may not equal the length of the whole byte array.
int getOffset( )	Returns the starting index of the data.
int getPort( )	Returns the port number.
void setAddress(InetAddress *ipAddress*)	Sets the address to which a packet will be sent. The address is specified by *ipAddress*.
void setData(byte[ ] *data*)	Sets the data to *data*, the offset to zero, and the length to number of bytes in *data*.
void setData(byte[ ] *data*, int *idx*, int *size*)	Sets the data to *data*, the offset to *idx*, and the length to *size*.
void setLength(int *size*)	Sets the length of the packet to *size*.
void setPort(int *port*)	Sets the port to *port*.

```
class WriteServer {
    public static int serverPort = 998;
    public static int clientPort = 999;
    public static int buffer_size = 1024;
    public static DatagramSocket ds;
    public static byte buffer[] = new byte[buffer_size];
    public static void TheServer() throws Exception {
        int pos = 0;
        while (true) {
            int c = System.in.read();
            switch (c) {
                case -1: System.out.println("Server Quits.");
                         ds.close(); return;
                case '\r': break;
                case '\n':
                  ds.send(new DatagramPacket(buffer, pos,
                   InetAddress.getLocalHost(), clientPort));
                  pos = 0; break;
            default: buffer[pos++] = (byte) c;
            }
        }
    }
    public static void TheClient() throws Exception {
        while (true) {
            DatagramPacket p = new DatagramPacket(buffer,
                                                 buffer.length);
            ds.receive(p);
            System.out.println(new String(p.getData(), 0,
                                                 p.getLength()));
        }
    }
    public static void main(String args[]) throws Exception {
        if (args.length == 1) {
            ds = new DatagramSocket(serverPort);
            TheServer();
        } else {
            ds = new DatagramSocket(clientPort);
            TheClient();
        }
    }
}
```

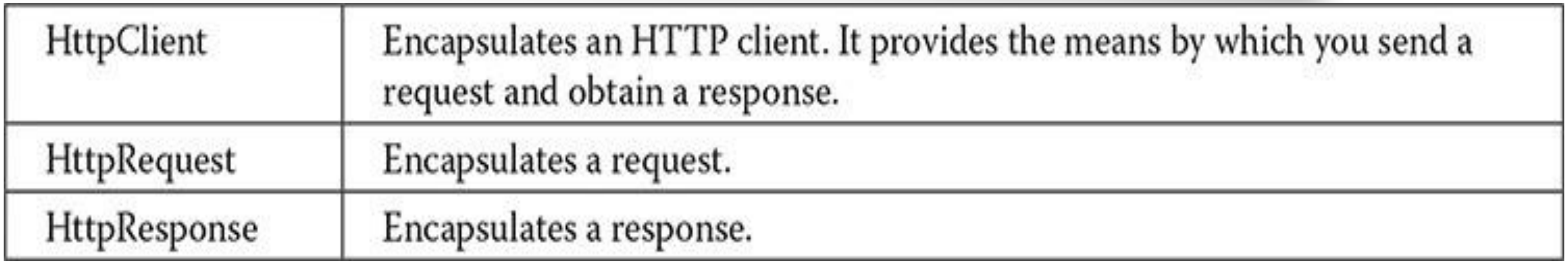
Plan

1. Networking basics
2. java.net networking Classes and Interfaces
3. InetAddress class
4. Datagrams
5. Introducing java.net.http

Introducing java.net.http

- From JDK 11: java.net.http has been added.
  - It provides enhanced, updated networking support for HTTP clients.
  - This new API is generally referred to as the HTTP Client API.
- For many types of HTTP networking:
  - The capabilities defined by the API in java.net.http can provide superior solutions.
  - In addition to offering a stream-lined, easy-to-use API, other advantages include support for asynchronous communication, HTTP/2, and flow control.
- The HTTP Client API is designed as a superior alternative to the functionality provided by HttpURLConnection.
  - It also supports the WebSocket protocol for bi-directional communication.

Three Key Elements



HttpClient	Encapsulates an HTTP client. It provides the means by which you send a request and obtain a response.
HttpRequest	Encapsulates a request.
HttpResponse	Encapsulates a response.

HttpClient

- Encapsulates the HTTP request/response mechanism.
- Supports both synchronous and asynchronous communication.
- Once a HttpClient object is created: Use it to send requests and obtain responses.
- It is at the foundation of the HTTP Client API.
- HttpClient is an abstract class, and instances are not created via a public constructor.
  - Use a factory method to build one.
  - HttpClient supports builders with the HttpClient.Builder interface, which provides several methods that let you configure the HttpClient.

```
HttpClient myHC =
        HttpClient.newBuilder().build();
```

HttpClient.Builder [1]

- Defines a number of methods that let you configure the builder.
- Method followRedirects(): passing in the new redirect setting, which must be a value in the HttpClient.Redirect enumeration.
  - Values: ALWAYS, NEVER, and NORMAL.
  - ALWAYS and NEVER are self explanatory.
  - NORMAL setting causes redirects to be followed unless a redirect is from an HTTPS site to an HTTP site.
- Example:

```
HttpClient.Builder myBuilder =
            HttpClient.newBuilder().followRedirects(
                                                  HttpClient.Redirect.NORMAL);
HttpClient myHC = myBuilder.build();
```

HttpClient.Builder [2]

- Builder configuration settings include authentication, proxy, HTTP version, and priority → you can build an HTTP client to fit virtually any need.
- In cases in which the default configuration is sufficient, you can obtain a default HttpClient directly by calling :

  ```
  static HttpClient newHttpClient()
  ```

  - An HttpClient with a default configuration is returned.
- Example:

  ```
  HttpClient myHC = HttpClient.newHttpClient();
  ```

Method send

- Once an HttpClient instance is created: send a synchronous request by :

```
abstract <T> HttpResponse<T> send(
                                        HttpRequest request,
                                        HttpResponse.BodyHandler<T>
                                        responseBodyHandler)
```

  - request encapsulates the request and handler specifies how the response body is handled.
  - You can use one of the predefined body handlers provided by the HttpResponse.BodyHandlers class.
  - An HttpResponse object is returned.
  - send() provides the basic mechanism for HTTP communication.

HttpRequest

- Encapsulates requests in the HttpRequest abstract class.
- To create an HttpRequest object:

```
static HttpRequest.Builder newBuilder( )
static HttpRequest.Builder newBuilder(URI uri)
```

- HttpRequest.Builder lets you specify various aspects of the request, such as what request method to use.
- To actually construct a request: call build() on the builder instance:

```
HttpRequest build()
```

- Once an HttpRequest instance is created: use it in a call to HttpClient’s send() method.

HttpResponse [1]

- Generic interface:

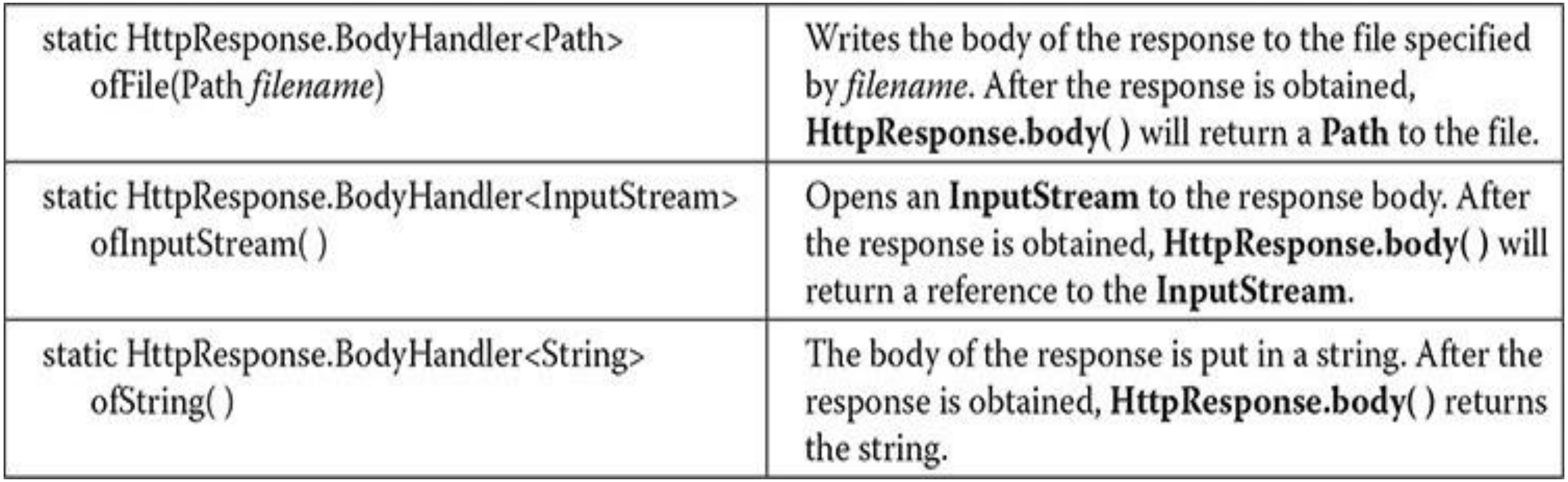
  ```
  HttpResponse<T>
  ```

  - T specifies the type of body.
- When a request is sent: an HttpResponse instance is returned that contains the response.
- Methods:

  ```
  T body()
  int statusCode( )
  HttpHeaders headers( )
  ```

HttpResponse [2]

- Responses are handled by implementations of the HttpResponse.BodyHandler interface.
- Some pre-defined body handler factory methods:



static HttpResponse.BodyHandler<Path> ofFile(Path *filename*)	Writes the body of the response to the file specified by *filename*. After the response is obtained, **HttpResponse.body( )** will return a **Path** to the file.
static HttpResponse.BodyHandler<InputStream> ofInputStream( )	Opens an **InputStream** to the response body. After the response is obtained, **HttpResponse.body( )** will return a reference to the **InputStream**.
static HttpResponse.BodyHandler<String> ofString( )	The body of the response is put in a string. After the response is obtained, **HttpResponse.body( )** returns the string.

HttpResponse [3]

- The stream returned by ofInputStream() should be read in its entirety.
    - Doing so enables associated resources to be freed.
    - If the entire body cannot be read for some reason, call close() to close the stream, which may also close the HTTP connection.
    - In general, it is best to simply read the entire stream.

```
import java.net.*; import java.net.http.*;
import java.io.*; import java.util.*;
public class HttpClientDemo {
    public static void main(String args[]) throws Exception{
        // Obtain a client that uses the default settings.
        HttpClient myHC = HttpClient.newHttpClient();
        // Create a request
        HttpRequest myReq = HttpRequest.newBuilder(new
                                URI("http://www.google.com/")).build();
        /* Send the request and get the response.
         * Here, an InputStream is used for the body */
        HttpResponse<InputStream> myResp = myHC.send(myReq,
                    HttpResponse.BodyHandlers.ofInputStream());
        // Display response code and response method.
        System.out.println("Response code is" +
                                                myResp.statusCode());
        System.out.println("Request method is " +
                                                myReq.method());
```

```
// Get headers from the response.
HttpHeaders hdrs = myResp.headers();
// Get a map of the headers.
Map<String, List<String>> hdrMap = hdrs.map();
Set<String> hdrField = hdrMap.keySet();
System.out.println("\nHere is the header");
// Display all header keys and values
for(String k: hdrField) {
    System.out.println("Key: " + k + " Value: "
                                              + hdrMap.get(k));
}
```

```
        // Display the body
        System.out.println("\nHere is the body: ");
        InputStream input = myResp.body(); int c;
        // Read and display the entire body.
        while ((c = input.read()) != -1) {
            System.out.print((char)c);
        }
    }
}
```

QUESTION ?