



CHAPTER 13

java.net

Package **java.net**

Java 1.0

The **java.net** package provides a powerful and flexible infrastructure for networking. The most common classes are briefly described in the following sections. Note that as of Java 1.4, the New I/O API of **java.nio** and **java.nio.channels** can be used for high-performance non-blocking networking. See also the **javax.net.ssl** package for classes for secure networking using SSL.

The **URL** class represents an Internet uniform resource locator (URL). It provides a very simple interface to networking: the object referred to by the URL can be downloaded with a single call, or streams may be opened to read from or write to the object. At a slightly more complex level, a **URLConnection** object can be obtained from a given **URL** object. The **URLConnection** class provides additional methods that allow you to work with URLs in more sophisticated ways. Java 1.4 introduces the **URI** class; it provides a powerful API for manipulating URI and URL strings, but does not have any networking capabilities itself.

If you want to do more than simply download an object referenced by a URL, you can do your own networking with the **Socket** class. This class allows you to connect to a specified port on a specified Internet host and read and write data using the **InputStream** and **OutputStream** classes of the **java.io** package. If you want to implement a server to accept connections from clients, you can use the related **ServerSocket** class. Both **Socket** and **ServerSocket** use the **InetAddress** address class, which represents an Internet address. In Java 1.4, **Inet4Address** and **Inet6Address** are subclasses that represent the addresses used by Versions 4 and 6 of the IP protocol. Java 1.4 also introduces the **SocketAddress** class as a high-level representation of a network address that is not tied to a specific networking protocol. There is also an IP-specific **InetSocketAddress** subclass that encapsulates an **InetAddress** and a port number.

The **java.net** package allows you to do low-level networking with **DatagramPacket** objects, which may be sent and received over the network through a **DatagramSocket** object. **MulticastSocket** extends **DatagramSocket** to support multicast networking.

Interfaces:

```
public interface ContentHandlerFactory;  
public interface DatagramSocketImplFactory;  
public interface FileNameMap;  
public interface SocketImplFactory;  
public interface SocketOptions;  
public interface URLStreamHandlerFactory;
```

Classes:

```
public abstract class Authenticator;  
public abstract class ContentHandler;  
public final class DatagramPacket;  
public class DatagramSocket;  
    public class MulticastSocket extends DatagramSocket;  
public abstract class DatagramSocketImpl implements SocketOptions;  
public class InetAddress implements Serializable;  
    public final class Inet4Address extends InetAddress;  
    public final class Inet6Address extends InetAddress;  
public final class NetPermission extends java.security.BasicPermission;  
public final class NetworkInterface;  
public final class PasswordAuthentication;  
public class ServerSocket;  
public class Socket;  
public abstract class SocketAddress implements Serializable;  
    public class InetSocketAddress extends SocketAddress;  
public abstract class SocketImpl implements SocketOptions;  
public final class SocketPermission extends java.security.Permission implements Serializable;  
public final class URI implements Comparable, Serializable;  
public final class URL implements Serializable;  
public class URLLoader extends java.security.SecureClassLoader;  
public abstract class URLConnection;  
    public abstract class HttpURLConnection extends URLConnection;  
    public abstract class JarURLConnection extends URLConnection;  
public class URLDecoder;  
public class URLEncoder;  
public abstract class URLStreamHandler;
```

Exceptions:

```
public class MalformedURLException extends java.io.IOException;  
public class ProtocolException extends java.io.IOException;  
public class SocketException extends java.io.IOException;  
    public class BindException extends SocketException;  
    public class ConnectException extends SocketException;  
    public class NoRouteToHostException extends SocketException;  
    public class PortUnreachableException extends SocketException;  
public class SocketTimeoutException extends java.io.InterruptedIOException;  
public class UnknownHostException extends java.io.IOException;  
public class UnknownServiceException extends java.io.IOException;  
public class URISyntaxException extends Exception;
```

Authenticator

Java 1.2

java.net

This abstract class defines a customizable mechanism for requesting and performing password authentication when required in URL-based networking. The static `setDefault()` method establishes the systemwide `Authenticator`. An `Authenticator` implementation can obtain the required authentication information from the user however it wants (e.g., through a text- or a GUI-based interface). `setDefault()` can be called only once; subsequent calls are ignored. Calling `setDefault()` requires an appropriate `NetPermission`.

When an application or the Java runtime system requires password authentication (to read the contents of a specified URL, for example), it calls the static `requestPasswordAuthentication()` method, passing arguments that specify the host and port for which the password is required and a prompt that may be displayed to the user. This method looks up the default `Authenticator` for the system and calls its `getPasswordAuthentication()` method. Calling `requestPasswordAuthentication()` requires an appropriate `NetPermission`.

`Authenticator` is an abstract class; its default implementation of `getPasswordAuthentication()` always returns null. To create an `Authenticator`, you must override this method so that it prompts the user to enter a username and password and returns that information in the form of a `PasswordAuthentication` object. Your implementation of `getPasswordAuthentication()` may call the various `getRequesting()` methods to find who is requesting the password and what the recommended user prompt is. In Java 1.4, a new version of the static `requestPasswordAuthentication()` method has been defined to allow specification of the requesting hostname. A corresponding `getRequestingHost()` instance method has also been added.

```
public abstract class Authenticator {
    // Public Constructors
    public Authenticator();
    // Public Class Methods
    public static PasswordAuthentication requestPasswordAuthentication(InetAddress addr, int port,
                                                                    String protocol, String prompt, String scheme);
    1.4 public static PasswordAuthentication requestPasswordAuthentication(String host, InetAddress addr, int port,
                                                                    String protocol, String prompt, String scheme);
    public static void setDefault(Authenticator a);                                synchronized
    // Protected Instance Methods
    protected PasswordAuthentication getPasswordAuthentication();                constant
    1.4 protected final String getRequestingHost();
    protected final int getRequestingPort();
    protected final String getRequestingPrompt();
    protected final String getRequestingProtocol();
    protected final String getRequestingScheme();
    protected final InetAddress getRequestingSite();
}
```

Passed To: `Authenticator.setDefault()`

BindException

Java 1.1

java.net

serializable checked

This exception signals that a socket cannot be bound to a local address and port. This often means that the port is already in use.



```
public class BindException extends SocketException {
// Public Constructors
    public BindException();
    public BindException(String msg);
}
```

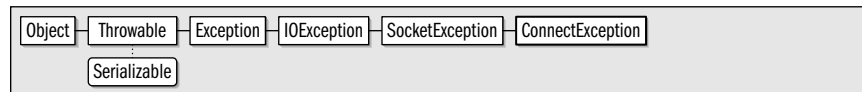
ConnectException

Java 1.1

java.net

serializable checked

This exception signals that a socket cannot be connected to a remote address and port. This means that the remote host can be reached, but is not responding, perhaps because there is no process on that host that is listening on the specified port.



```
public class ConnectException extends SocketException {
// Public Constructors
    public ConnectException();
    public ConnectException(String msg);
}
```

ContentHandler

Java 1.0

java.net

This abstract class defines a method that reads data from a `URLConnection` and returns an object that represents that data. Each subclass that implements this method is responsible for handling a different type of content (i.e., a different MIME type). Applications never create `ContentHandler` objects directly; they are created, when necessary, by the registered `ContentHandlerFactory` object. Applications should also never call `ContentHandler` methods directly; they should call `URL.getContent()` or `URLConnection.getContent()` instead. You need to subclass `ContentHandler` only if you are writing a web browser or similar application that needs to parse and understand some new content type.

```
public abstract class ContentHandler {
// Public Constructors
    public ContentHandler();
// Public Instance Methods
    public abstract Object getContent(URLConnection urlc) throws java.io.IOException;
1.3 public Object getContent(URLConnection urlc, Class[ ] classes) throws java.io.IOException;
}
```

Returned By: `ContentHandlerFactory.createContentHandler()`

ContentHandlerFactory

Java 1.0

java.net

This interface defines a method that creates and returns an appropriate `ContentHandler` object for a specified MIME type. A systemwide `ContentHandlerFactory` interface may be specified using the `URLConnection.setContentHandlerFactory()` method. Normal applications never need to use or implement this interface.

ContentHandlerFactory

```
public interface ContentHandlerFactory {  
    // Public Instance Methods  
    public abstract java.net.ContentHandler createContentHandler(String mimetype);  
}
```

Passed To: URLConnection.setContentHandlerFactory()

DatagramPacket

Java 1.0

java.net

This class implements a packet of data that may be sent or received over the network through a `DatagramSocket`. Create a `DatagramPacket` to be sent over the network with one of the constructor methods that includes a network address. Create a `DatagramPacket` into which data can be received using one of the constructors that does not include a network address argument. The `receive()` method of `DatagramSocket` waits for data and stores it in a `DatagramPacket` created in this way. The contents and sender of a received packet can be queried with the `DatagramPacket` instance methods.

New constructors and methods were added to this class in Java 1.4 to support the `SocketAddress` abstraction of a network address.

```
public final class DatagramPacket {  
    // Public Constructors  
    public DatagramPacket(byte[] buf, int length);  
    1.4 public DatagramPacket(byte[] buf, int length, SocketAddress address) throws SocketException;  
    1.2 public DatagramPacket(byte[] buf, int offset, int length);  
        public DatagramPacket(byte[] buf, int length, InetAddress address, int port);  
    1.4 public DatagramPacket(byte[] buf, int offset, int length, SocketAddress address) throws SocketException;  
    1.2 public DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port);  
    // Property Accessor Methods (by property name)  
    public InetAddress getAddress(); synchronized  
    1.1 public void setAddress(InetAddress iaddr); synchronized  
    public byte[] getData(); synchronized  
    1.1 public void setData(byte[] buf); synchronized  
    1.2 public void setData(byte[] buf, int offset, int length); synchronized  
        public int getLength(); synchronized  
    1.1 public void setLength(int length); synchronized  
    1.2 public int getOffset(); synchronized  
        public int getPort(); synchronized  
    1.1 public void setPort(int iport); synchronized  
    1.4 public SocketAddress getSocketAddress(); synchronized  
    1.4 public void setSocketAddress(SocketAddress address); synchronized  
}
```

Passed To: DatagramSocket.{receive(), send()}, DatagramSocketImpl.{peekData(), receive(), send()}, MulticastSocket.send()

DatagramSocket

Java 1.0

java.net

This class defines a socket that can receive and send unreliable datagram packets over the network using the UDP protocol. A *datagram* is a very low-level networking interface: it is simply an array of bytes sent over the network. A datagram does not implement any kind of stream-based communication protocol, and there is no connection established between the sender and the receiver. Datagram packets are called unreliable because the protocol does not make any attempt to ensure they arrive or to resend them if they don't. Thus, packets sent through a `DatagramSocket` are not guaranteed to

arrive in the order sent or even to arrive at all. On the other hand, this low-overhead protocol makes datagram transmission very fast. See `Socket` and `URL` for higher-level interfaces to networking. This class was introduced in Java 1.0, and was enhanced in Java 1.4 to allow local and remote addresses to be specified using the protocol-independent `SocketAddress` class.

`send()` sends a `DatagramPacket` through the socket. The packet must contain the destination address to which it should be sent. `receive()` waits for data to arrive at the socket and stores it, along with the address of the sender, in the specified `DatagramPacket`. `close()` closes the socket and frees the local port for reuse. Once `close()` has been called, the `DatagramSocket` should not be used again, except to call the `isClosed()` method, which returns `true` if the socket has been closed.

Each time a packet is sent or received, the system must perform a security check to ensure that the calling code has permission to send data to or receive data from the specified host. In Java 1.2 and later, if you are sending multiple packets to or receiving multiple packets from a single host, use `connect()` to specify the host with which you are communicating. This causes the security check to be done a single time but does not allow the socket to communicate with any other host until `disconnect()` is called. Use `getRemoteSocketAddress()`, or `getInetAddress()` and `getPort()`, to obtain the network address, if any, that the socket is connected to. Use `isConnected()` to determine if the socket is currently connected in this way.

By default, a `DatagramSocket` sends data through a local address assigned by the system. If desired, however, you can *bind* the socket to a specified local address. Do this by using one of the constructors other than the no-arg constructor or by binding the `DatagramSocket` to a local `SocketAddress` with the `bind()` method. You can determine whether a `DatagramSocket` is bound with `isBound()` and obtain the local address of the socket with `getLocalSocketAddress()` or with `getLocalAddress()` and `getLocalPort()`.

This class defines a number of get/set method pairs for setting and querying a variety of “socket options” for datagram transmission. `setSoTimeout()` specifies the number of milliseconds that `receive()` waits for a packet to arrive before throwing an `InterruptedIOException`. Specify 0 milliseconds to wait forever. `setSendBufferSize()` and `setReceiveBufferSize()` set hints as to the underlying size of the networking buffers. `setBroadcast()`, `setReuseAddress()`, and `setTrafficClass()` set more complex socket options; use of these options requires a sophisticated understanding of low-level network protocols, and an explanation of them is beyond the scope of this reference.

In Java 1.4 and later, `getChannel()` returns a `java.nio.channels.DatagramChannel` associated with this `DatagramSocket`. Sockets created with one of the `DatagramSocket()` constructors always return null from this method. `getChannel()` returns only a useful value for sockets that were created by and belong to a `DatagramChannel`.

```
public class DatagramSocket {
    // Public Constructors
    public DatagramSocket() throws SocketException;
    1.4 public DatagramSocket(SocketAddress bindaddr) throws SocketException;
    public DatagramSocket(int port) throws SocketException;
    1.1 public DatagramSocket(int port, InetAddress laddr) throws SocketException;
    // Protected Constructors
    1.4 protected DatagramSocket(DatagramSocketImpl impl);
    // Public Class Methods
    1.3 public static void setDatagramSocketImplFactory(DatagramSocketImplFactory fac)           synchronized
        throws java.io.IOException;
    // Property Accessor Methods (by property name)
    1.4 public boolean isBound();                                                             default:true
    1.4 public boolean getBroadcast() throws SocketException;                               synchronized default:true
```

DatagramSocket

```
1.4 public void setBroadcast(boolean on) throws SocketException;           synchronized
1.4 public java.nio.channels.DatagramChannel getChannel();                 constant default:null
1.4 public boolean isClosed();                                           default:false
1.4 public boolean isConnected();                                       default:false
1.2 public InetAddress getInetAddress();                                default:null
1.1 public InetAddress getLocalAddress();                             default:InetAddress
    public int getLocalPort();
1.4 public SocketAddress getLocalSocketAddress();                     default:InetSocketAddress
1.2 public int getPort();                                              default:-1
1.2 public int getReceiveBufferSize() throws SocketException;         synchronized default:32767
1.2 public void setReceiveBufferSize(int size) throws SocketException;   synchronized
1.4 public SocketAddress getRemoteSocketAddress();                     default:null
1.4 public boolean getReuseAddress() throws SocketException;           synchronized default:false
1.4 public void setReuseAddress(boolean on) throws SocketException;     synchronized
1.2 public int getSendBufferSize() throws SocketException;             synchronized default:32767
1.2 public void setSendBufferSize(int size) throws SocketException;     synchronized
1.1 public int getSoTimeout() throws SocketException;                 synchronized default:0
1.1 public void setSoTimeout(int timeout) throws SocketException;       synchronized
1.4 public int getTrafficClass() throws SocketException;               synchronized default:0
1.4 public void setTrafficClass(int tc) throws SocketException;         synchronized
// Public Instance Methods
1.4 public void bind(SocketAddress addr) throws SocketException;         synchronized
    public void close();
1.4 public void connect(SocketAddress addr) throws SocketException;
1.2 public void connect(InetAddress address, int port);
1.2 public void disconnect();
    public void receive(DatagramPacket p) throws java.io.IOException;     synchronized
    public void send(DatagramPacket p) throws java.io.IOException;
}
```

Subclasses: MulticastSocket

Returned By: java.nio.channels.DatagramChannel.socket()

DatagramSocketImpl

Java 1.1

java.net

This abstract class defines the methods necessary to implement communication through datagram and multicast sockets. System programmers may create subclasses of this class when they need to implement datagram or multicast sockets in a nonstandard network environment, such as behind a firewall or on a network that uses a nonstandard transport protocol. Normal applications never need to use or subclass this class.

Object — DatagramSocketImpl — SocketOptions

```
public abstract class DatagramSocketImpl implements SocketOptions {
// Public Constructors
    public DatagramSocketImpl();
// Protected Instance Methods
    protected abstract void bind(int lport, InetAddress laddr) throws SocketException;
    protected abstract void close();
1.4 protected void connect(InetAddress address, int port) throws SocketException;   empty
    protected abstract void create() throws SocketException;
1.4 protected void disconnect();                                                  empty
    protected java.io.FileDescriptor getFileDescriptor();
    protected int getLocalPort();
}
```

```

1.2 protected abstract int getTimeout() throws java.io.IOException;
    protected abstract void join(InetAddress inetaddr) throws java.io.IOException;
1.4 protected abstract void joinGroup(SocketAddress mcastaddr, NetworkInterface netif) throws java.io.IOException;
    protected abstract void leave(InetAddress inetaddr) throws java.io.IOException;
1.4 protected abstract void leaveGroup(SocketAddress mcastaddr, NetworkInterface netif) throws java.io.IOException;
    protected abstract int peek(InetAddress i) throws java.io.IOException;
1.4 protected abstract int peekData(DatagramPacket p) throws java.io.IOException;
    protected abstract void receive(DatagramPacket p) throws java.io.IOException;
    protected abstract void send(DatagramPacket p) throws java.io.IOException;
1.2 protected abstract void setTimeout(int ttl) throws java.io.IOException;
// Protected Instance Fields
    protected java.io.FileDescriptor fd;
    protected int localPort;
// Deprecated Protected Methods
#    protected abstract byte getTTL() throws java.io.IOException;
#    protected abstract void setTTL(byte ttl) throws java.io.IOException;
}

```

Passed To: DatagramSocket.DatagramSocket()

Returned By: DatagramSocketImplFactory.createDatagramSocketImpl()

DatagramSocketImplFactory

Java 1.3

java.net

This interface defines a method that creates DatagramSocketImpl objects. You can register an instance of this factory interface with the static setDataagramSocketImplFactory() method of DatagramSocket. Application-level code never needs to use or implement this interface.

```

public interface DatagramSocketImplFactory {
// Public Instance Methods
    public abstract DatagramSocketImpl createDatagramSocketImpl();
}

```

Passed To: DatagramSocket.setDataagramSocketImplFactory()

FileNameMap

Java 1.1

java.net

This interface defines a single method that is called to obtain the MIME type of a file based on the name of the file. The fileNameMap field of the URLConnection class refers to an object that implements this interface. The filename-to-file-type map it implements is used by the static URLConnection.guessContentTypeFromName() method.

```

public interface FileNameMap {
// Public Instance Methods
    public abstract String getContentTypeFor(String fileName);
}

```

Passed To: URLConnection.setFileNameMap()

Returned By: URLConnection.getFileNameMap()

URLConnection

Java 1.1

java.net

This class is a specialization of URLConnection. An instance of this class is returned when the openConnection() method is called for a URL object that uses the HTTP protocol. The

HttpURLConnection

many constants defined by this class are the status codes returned by HTTP servers. `setRequestMethod()` specifies what kind of HTTP request is made. The contents of this request must be sent through the `OutputStream` returned by the `getOutputStream()` method of the superclass. Once an HTTP request has been sent, `getResponseCode()` returns the HTTP server's response code as an integer, and `getResponseMessage()` returns the server's response message. The `disconnect()` method closes the connection. The static `setFollowRedirects()` specifies whether URL connections that use the HTTP protocol should automatically follow redirect responses sent by HTTP servers. In order to successfully use this class, you need to understand the details of the HTTP protocol.

```
Object — HttpURLConnection — HttpURLConnection

public abstract class HttpURLConnection extends URLConnection {
// Protected Constructors
    protected HttpURLConnection(URL u);
// Public Constants
    public static final int HTTP_ACCEPTED;                =202
    public static final int HTTP_BAD_GATEWAY;             =502
    public static final int HTTP_BAD_METHOD;             =405
    public static final int HTTP_BAD_REQUEST;            =400
    public static final int HTTP_CLIENT_TIMEOUT;         =408
    public static final int HTTP_CONFLICT;              =409
    public static final int HTTP_CREATED;               =201
    public static final int HTTP_ENTITY_TOO_LARGE;       =413
    public static final int HTTP_FORBIDDEN;             =403
    public static final int HTTP_GATEWAY_TIMEOUT;       =504
    public static final int HTTP_GONE;                 =410
    public static final int HTTP_INTERNAL_ERROR;        =500
    public static final int HTTP_LENGTH_REQUIRED;       =411
    public static final int HTTP_MOVED_PERM;            =301
    public static final int HTTP_MOVED_TEMP;            =302
    public static final int HTTP_MULT_CHOICE;           =300
    public static final int HTTP_NO_CONTENT;            =204
    public static final int HTTP_NOT_ACCEPTABLE;        =406
    public static final int HTTP_NOT_AUTHORITY;         =203
    public static final int HTTP_NOT_FOUND;             =404
    1.3 public static final int HTTP_NOT_IMPLEMENTED;    =501
    public static final int HTTP_NOT_MODIFIED;          =304
    public static final int HTTP_OK;                   =200
    public static final int HTTP_PARTIAL;              =206
    public static final int HTTP_PAYMENT_REQUIRED;     =402
    public static final int HTTP_PRECON_FAILED;        =412
    public static final int HTTP_PROXY_AUTH;           =407
    public static final int HTTP_REQ_TOO_LONG;         =414
    public static final int HTTP_RESET;                =205
    public static final int HTTP_SEE_OTHER;            =303
    public static final int HTTP_UNAUTHORIZED;         =401
    public static final int HTTP_UNAVAILABLE;          =503
    public static final int HTTP_UNSUPPORTED_TYPE;     =415
    public static final int HTTP_USE_PROXY;            =305
    public static final int HTTP_VERSION;              =505
// Public Class Methods
    public static boolean getFollowRedirects();
    public static void setFollowRedirects(boolean set);
// Property Accessor Methods (by property name)
    1.2 public java.io.InputStream getErrorStream();    constant
```

```

1.3 public boolean getInstanceFollowRedirects();
1.3 public void setInstanceFollowRedirects(boolean followRedirects);
1.2 public java.security.Permission getPermission() throws java.io.IOException; Overrides:URLConnection
    public String getRequestMethod();
    public void setRequestMethod(String method) throws ProtocolException;
    public int getResponseCode() throws java.io.IOException; constant
    public String getResponseMessage() throws java.io.IOException;
// Public Instance Methods
    public abstract void disconnect();
    public abstract boolean usingProxy();
// Public Methods Overriding URLConnection
1.3 public long getHeaderFieldDate(String name, long Default);
// Protected Instance Fields
1.3 protected boolean instanceFollowRedirects;
    protected String method;
    protected int responseCode;
    protected String responseMessage;
// Deprecated Public Fields
# public static final int HTTP_SERVER_ERROR; =500
}

```

Subclasses: javax.net.ssl.HttpsURLConnection

Inet4Address

Java 1.4

java.net

serializable

Inet4Address implements methods defined by its superclass to make them specific to Internet Protocol version 4 (IPv4) Internet addresses. Inet4Address does not have a constructor. Create instances with the static methods of InetAddress, which return instances of Inet4Address or Inet6Address as appropriate.



```

public final class Inet4Address extends InetAddress {
// No Constructor
// Public Methods Overriding InetAddress
    public boolean equals(Object obj);
    public byte[] getAddress();
    public String getHostAddress();
    public int hashCode();
    public boolean isAnyLocalAddress();
    public boolean isLinkLocalAddress();
    public boolean isLoopbackAddress();
    public boolean isMCGlobal();
    public boolean isMCLinkLocal();
    public boolean isMCNodeLocal; constant
    public boolean isMCOrgLocal;
    public boolean isMCSiteLocal;
    public boolean isMulticastAddress();
    public boolean isSiteLocalAddress();
}

```

Inet6Address

Inet6Address

Java 1.4

java.net

serializable

Inet4Address implements methods defined by its superclass to make them specific to Internet Protocol version 6 (IPv6) Internet addresses. See RFC 2373 for complete details about Internet addresses of this type. Inet6Address does not have a constructor. Create instances with the static methods of InetAddress, which return instances of Inet4Address or Inet6Address as appropriate.



```
public final class Inet6Address extends InetAddress {
    // No Constructor
    // Public Instance Methods
    public boolean isIPv4CompatibleAddress();
    // Public Methods Overriding InetAddress
    public boolean equals(Object obj);
    public byte[] getAddress();
    public String getHostAddress();
    public int hashCode();
    public boolean isAnyLocalAddress();
    public boolean isLinkLocalAddress();
    public boolean isLoopbackAddress();
    public boolean isMCGlobal();
    public boolean isMCLinkLocal();
    public boolean isMCNodeLocal();
    public boolean isMCOrgLocal();
    public boolean isMCSiteLocal();
    public boolean isMulticastAddress();
    public boolean isSiteLocalAddress();
}
```

InetAddress

Java 1.0

java.net

serializable

This class represents an Internet Protocol (IP) address. The class does not have a public constructor but instead supports static factory methods for obtaining InetAddress objects. getLocalHost() returns the InetAddress of the local computer. getByName() returns the InetAddress of a host specified by name. getAllByName() returns an array of InetAddress objects that represents all the available addresses for a host specified by name. getByAddress() returns an InetAddress that represents the IP address defined by the specified array of bytes.

Once you have obtained an InetAddress object, its instance methods provide various sorts of information about it. Two of the most important are getHostName(), which returns the hostname, and getAddress(), which returns the Internet IP address as an array of bytes, with the highest-order byte as the first element of the array. getHostAddress() returns the IP address formatted as a string rather than as an array of bytes. The various methods whose names begin with “is” determine whether the address falls into any of the named categories. The “isMC” methods are all related to multicast addresses.

This class was originally defined in Java 1.0, but many of its methods were added in Java 1.4. Java 1.4 also defines two subclasses, `Inet4Address` and `Inet6Address`, representing IPv4 and IPv6 (Versions 4 and 6) addresses.

```

Object | InetSocketAddress | Serializable
public class InetSocketAddress implements Serializable {
// No Constructor
// Public Class Methods
    public static InetSocketAddress[] getAllByName(String host) throws java.net.UnknownHostException;
    1.4 public static InetSocketAddress getByAddress(byte[] addr) throws java.net.UnknownHostException;
    1.4 public static InetSocketAddress getByAddress(String host, byte[] addr) throws java.net.UnknownHostException;
    public static InetSocketAddress getByName(String host) throws java.net.UnknownHostException;
    public static InetSocketAddress getLocalHost() throws java.net.UnknownHostException;           synchronized
// Property Accessor Methods (by property name)
    public byte[] getAddress();           constant
    1.4 public boolean isAnyLocalAddress();           constant
    1.4 public String getCanonicalHostName();
    public String getHostAddress();           constant
    public String getHostName();
    1.4 public boolean isLinkLocalAddress();           constant
    1.4 public boolean isLoopbackAddress();           constant
    1.4 public boolean isMCGlobal();           constant
    1.4 public boolean isMCLinkLocal();           constant
    1.4 public boolean isMCNodeLocal();           constant
    1.4 public boolean isMCOrgLocal();           constant
    1.4 public boolean isMCSiteLocal();           constant
    1.1 public boolean isMulticastAddress();           constant
    1.4 public boolean isSiteLocalAddress();           constant
// Public Methods Overriding Object
    public boolean equals(Object obj);           constant
    public int hashCode();           constant
    public String toString();
}

```

Subclasses: `Inet4Address`, `Inet6Address`

Passed To: Too many methods to list.

Returned By: Too many methods to list.

Type Of: `SocketImpl.address`

InetSocketAddress

Java 1.4

java.net

serializable

`InetSocketAddress` represents the combination of an Internet Protocol (IP) address and a port number. The constructors allow you to specify the IP address as an `InetAddress` or as a hostname and also allow you to omit the IP address, in which case the wildcard address is used (this is useful for server sockets).

```

Object | SocketAddress | InetSocketAddress
      | :
      | Serializable
public class InetSocketAddress extends SocketAddress {
// Public Constructors
    public InetSocketAddress(int port);
    public InetSocketAddress(InetAddress addr, int port);
}

```

InetSocketAddress

```
public InetSocketAddress(String hostname, int port);  
// Public Instance Methods  
public final InetSocketAddress getAddress();  
public final String getHostName();  
public final int getPort();  
public final boolean isUnresolved();  
// Public Methods Overriding Object  
public final boolean equals(Object obj);  
public final int hashCode();  
public String toString();  
}
```

JarURLConnection

Java 1.2

java.net

This class is a specialized `URLConnection` that represents a connection to a jar: URL. A jar: URL is a compound URL that includes the URL of a JAR archive and, optionally, a reference to a file or directory within the JAR archive. The jar: URL syntax uses the ! character to separate the pathname of the JAR archive from the filename within the JAR archive. Note that a jar: URL contains a subprotocol that specifies the protocol that retrieves the JAR file itself. For example:

```
jar:http://my.jar.com/my.jar/           // The whole archive  
jar:file:/usr/java/lib/my.jar!/com/jar/ // A directory of the archive  
jar:ftp://ftp.jar.com/pub/my.jar!/com/jar/Jar.class // A file in the archive
```

To obtain a `JarURLConnection`, define a `URL` object for a jar: URL, open a connection to it with `openConnection()`, and cast the returned `URLConnection` object to a `JarURLConnection`. The various methods defined by `JarURLConnection` allow you to read the manifest file of the JAR archive and look up attributes from that manifest for the archive as a whole or for individual entries in the archive. These methods make use of various classes from the `java.util.jar` package.

Object — `URLConnection` — `JarURLConnection`

```
public abstract class JarURLConnection extends URLConnection {  
    // Protected Constructors  
    protected JarURLConnection(URL url) throws MalformedURLException;  
    // Property Accessor Methods (by property name)  
    public java.util.jar.Attributes getAttributes() throws java.io.IOException;  
    public java.security.cert.Certificate[] getCertificates() throws java.io.IOException;  
    public String getEntryName();  
    public java.util.jar.JarEntry getJarEntry() throws java.io.IOException;  
    public abstract java.util.jar.JarFile getJarFile() throws java.io.IOException;  
    public URL getJarFileURL();  
    public java.util.jar.Attributes getMainAttributes() throws java.io.IOException;  
    public java.util.jar.Manifest getManifest() throws java.io.IOException;  
    // Protected Instance Fields  
    protected URLConnection jarFileURLConnection;  
}
```

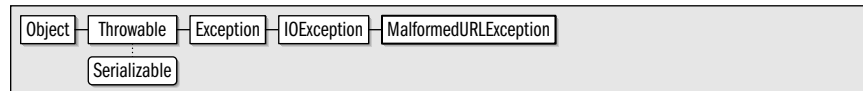
MalformedURLException

Java 1.0

java.net

serializable checked

This exception signals that an unparseable URL specification has been passed to a method.



```

public class MalformedURLException extends java.io.IOException {
    // Public Constructors
    public MalformedURLException();
    public MalformedURLException(String msg);
}
  
```

Thrown By: Too many methods to list.

MulticastSocket

Java 1.1

java.net

This subclass of `DatagramSocket` can send and receive multicast UDP packets. It extends `DatagramSocket` by adding `joinGroup()` and `leaveGroup()` methods to join and leave multicast groups. You do not have to join a group to send a packet to a multicast address, but you must join the group to receive packets sent to that address. Note that the use of a `MulticastSocket` is governed by a security manager.

Use `setTimeToLive()` to set a time-to-live value for any packets sent through a `MulticastSocket`. This constrains the number of network hops a packet can take and controls the scope of a multicast. Use `setInterface()` or `setNetworkInterface()` to specify the `InetAddress` or the `NetworkInterface` that outgoing multicast packets should use: this is useful for servers or other computers that have more than one Internet address or network interface. `setLoopbackMode()` specifies whether a multicast packet sent through this socket should be sent back to this socket or not. This method should really be named “`setLoopbackModeDisabled()`”: passing an argument of `true` requests (but does not require) that the system disable loopback packets.



```

public class MulticastSocket extends DatagramSocket {
    // Public Constructors
    public MulticastSocket() throws java.io.IOException;
    1.4 public MulticastSocket(SocketAddress bindaddr) throws java.io.IOException;
    public MulticastSocket(int port) throws java.io.IOException;
    // Property Accessor Methods (by property name)
    public InetAddress getInterface() throws SocketException;           default:InetAddress
    public void setInterface(InetAddress intf) throws SocketException;
    1.4 public boolean getLoopbackMode() throws SocketException;         default:true
    1.4 public void setLoopbackMode(boolean disable) throws SocketException;
    1.4 public NetworkInterface getNetworkInterface() throws SocketException;
    1.4 public void setNetworkInterface(NetworkInterface netIf) throws SocketException;
    1.2 public int getTimeToLive() throws java.io.IOException;           default:1
    1.2 public void setTimeToLive(int ttl) throws java.io.IOException;
    // Public Instance Methods
    public void joinGroup(InetAddress mcastaddr) throws java.io.IOException;
    1.4 public void joinGroup(SocketAddress mcastaddr, NetworkInterface netIf) throws java.io.IOException;
    public void leaveGroup(InetAddress mcastaddr) throws java.io.IOException;
}
  
```

MulticastSocket

```
1.4 public void leaveGroup(SocketAddress mcastaddr, NetworkInterface netIf) throws java.io.IOException;
// Deprecated Public Methods
# public byte getTTL() throws java.io.IOException;                                default:1
# public void send(DatagramPacket p, byte ttl) throws java.io.IOException;
# public void setTTL(byte ttl) throws java.io.IOException;
}
```

NetPermission

Java 1.2

java.net

serializable permission

This class is a `java.security.Permission` that represents various permissions required for Java's URL-based networking system. See also `SocketPermission`, which represents permissions to perform lower-level networking operations. A `NetPermission` is defined solely by its name; no actions list is required or supported. As of Java 1.2, there are three `NetPermission` targets defined: "setDefaultAuthenticator" is required to call `Authenticator.setDefault()`; "requestPasswordAuthentication" to call `Authenticator.requestPasswordAuthentication()`; and "specifyStreamHandler" to explicitly pass a `URLStreamHandler` object to the `URL()` constructor. The target "*" is a wildcard that represents all defined `NetPermission` targets.

System administrators configuring security policies must be familiar with this class and the permissions it represents. System programmers may use this class, but application programmers never need to use it explicitly.



```
public final class NetPermission extends java.security.BasicPermission {
// Public Constructors
    public NetPermission(String name);
    public NetPermission(String name, String actions);
}
```

NetworkInterface

Java 1.4

java.net

Instances of this class represent a network interface on the local machine. `getName()` and `getDisplayName()` return the name of the interface, and `getInetAddresses()` returns a `java.util.Enumeration` of the Internet addresses for the interface. Obtain a `NetworkInterface` object with one of the static methods defined by this class. `getNetworkInterfaces()` returns an enumeration of all interfaces for the local host. Typically, this class is used only in advanced networking applications.

```
public final class NetworkInterface {
// No Constructor
// Public Class Methods
    public static NetworkInterface getByInetAddress(InetAddress addr) throws SocketException;    native
    public static NetworkInterface getByName(String name) throws SocketException;            native
    public static java.util.Enumeration getNetworkInterfaces() throws SocketException;
// Public Instance Methods
    public String getDisplayName();
    public java.util.Enumeration getInetAddresses();
    public String getName();
}
```

```
// Public Methods Overriding Object
public boolean equals(Object obj);
public int hashCode();
public String toString();
}
```

Passed To: DatagramSocketImpl.{joinGroup(), leaveGroup()}, MulticastSocket.{joinGroup(), leaveGroup(), setNetworkInterface()}

Returned By: MulticastSocket.getNetworkInterface(), NetworkInterface.{getByInetAddress(), getByName()}

NoRouteToHostException

Java 1.1

java.net

serializable checked

This exception signals that a socket cannot be connected to a remote host because the host cannot be contacted. Typically, this means that some link in the network between the local machine and the remote host is down or that the host is behind a firewall.



```
public class NoRouteToHostException extends SocketException {
// Public Constructors
public NoRouteToHostException();
public NoRouteToHostException(String msg);
}
```

PasswordAuthentication

Java 1.2

java.net

This simple immutable class encapsulates a username and a password. The password is stored as a character array rather than as a `String` object so that the caller can erase the contents of the array after use for increased security. Note that the `PasswordAuthentication()` constructor clones the specified password character array, but `getPassword()` returns a reference to the object's internal array.

Application programmers defining an `Authenticator` object for their application need to create and return a `PasswordAuthentication` object from the `getPasswordAuthentication()` method of that object. System programmers writing `URLStreamHandler` implementations or otherwise interacting with a network server that requests password authentication may obtain a `PasswordAuthentication` object representing the user's name and password by calling the static `Authenticator.requestPasswordAuthentication()` method.

```
public final class PasswordAuthentication {
// Public Constructors
public PasswordAuthentication(String userName, char[] password);
// Public Instance Methods
public char[] getPassword();
public String getUserName();
}
```

Returned By: Authenticator.{getPasswordAuthentication(), requestPasswordAuthentication()}

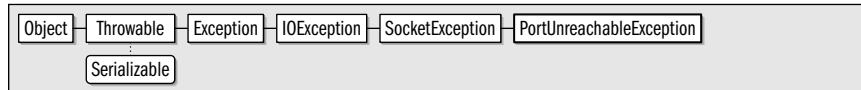
PortUnreachableException

Java 1.4

java.net

serializable checked

An exception of this type may be thrown by a `send()` or `receive()` call on a `DatagramSocket` if the `connect()` method of that socket has been called and if the connection attempt resulted in an ICMP “port unreachable” message.



```

public class PortUnreachableException extends SocketException {
    // Public Constructors
    public PortUnreachableException();
    public PortUnreachableException(String msg);
}

```

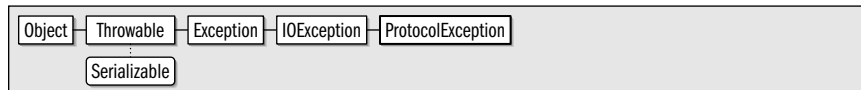
ProtocolException

Java 1.0

java.net

serializable checked

This exception signals a protocol error in the `Socket` class.



```

public class ProtocolException extends java.io.IOException {
    // Public Constructors
    public ProtocolException();
    public ProtocolException(String host);
}

```

Thrown By: `URLConnection.setRequestMethod()`

ServerSocket

Java 1.0

java.net

This class is used by servers to listen for connection requests from clients. Before you can use a `ServerSocket`, it must be *bound* to the local network address on which it will listen. All of the `ServerSocket()` constructors, except for the no-argument constructor, create a server socket and bind it to the specified local port, optionally specifying a “connection backlog” value: the number of client connection attempts that may be queued before subsequent connection attempts are rejected.

In Java 1.4 and later, the no-argument `ServerSocket()` constructor allows you to create an unbound socket. Doing this allows you to bind the socket using the `bind()` method, which uses a `SocketAddress` object rather than a port number. It also allows you to call `setReuseAddress()`, which is useful only when done before the socket is bound. Call `isBound()` to determine whether a server socket has been bound. If it has, use `getLocalSocketAddress()`, or `getLocalPort()` and `getInetAddress()`, to obtain the local address it is bound to.

Once a `ServerSocket` has been bound, you can call the `accept()` method to listen on the specified port and block until the client requests a connection on the port. When this happens, `accept()` accepts the connection, creating and returning a `Socket` the server can use to communicate with the client. A typical server starts a new thread to handle the communication with the client and calls `accept()` again to listen for another connection.

ServerSocket defines several methods for setting socket options that affect the socket's behavior. `setSoTimeout()` specifies the number of milliseconds that `accept()` should block before throwing an `InterruptedIOException`. A value of 0 means that it should block forever. `setReceiveBufferSize()` is an advanced option that suggests the desired size for the internal receive buffer of the `Socket` objects returned by `accept()`. This is only a hint and may be ignored by the system. `setReuseAddress()` is another advanced option; it specifies that a `bind()` operation should succeed even if the local bind address is still nominally in use by a socket that is in the process of shutting down.

Like all sockets, a `ServerSocket` should be closed with the `close()` method when it is no longer needed. Once closed, a `ServerSocket` should not be used, except to call the `isClosed()` method, which returns `true` if it has been closed.

The `getChannel()` method is a link between this `ServerSocket` class and the New I/O `java.nio.channels.ServerSocketChannel` class. It returns the `ServerSocketChannel` associated with this `ServerSocket`, if there is one. Note, however, that this method always returns `null` for sockets created with any of the `ServerSocket()` constructors. If you create a `ServerSocketChannel` object, and obtain a `ServerSocket` from it, however, then the `getChannel()` method provides a way to link back to the parent channel.

```
public class ServerSocket {
    // Public Constructors
    1.4 public ServerSocket() throws java.io.IOException;
        public ServerSocket(int port) throws java.io.IOException;
        public ServerSocket(int port, int backlog) throws java.io.IOException;
    1.1 public ServerSocket(int port, int backlog, InetAddress bindAddr) throws java.io.IOException;
    // Public Class Methods
        public static void setSocketFactory(SocketImplFactory fac) throws java.io.IOException;           synchronized
    // Property Accessor Methods (by property name)
    1.4 public boolean isBound();                               default:false
    1.4 public java.nio.channels.ServerSocketChannel getChannel();       constant default:null
    1.4 public boolean isClosed();                               default:false
        public InetAddress getInetAddress();                   default:null
        public int getLocalPort();                             default:-1
    1.4 public SocketAddress getLocalSocketAddress();           default:null
    1.4 public int getReceiveBufferSize() throws SocketException;   synchronized default:32767
    1.4 public void setReceiveBufferSize(int size) throws SocketException;   synchronized
    1.4 public boolean getReuseAddress() throws SocketException;       default:true
    1.4 public void setReuseAddress(boolean on) throws SocketException;
    1.1 public int getSoTimeout() throws java.io.IOException;       synchronized default:0
    1.1 public void setSoTimeout(int timeout) throws SocketException;   synchronized
    // Public Instance Methods
        public Socket accept() throws java.io.IOException;
    1.4 public void bind(SocketAddress endpoint) throws java.io.IOException;
    1.4 public void bind(SocketAddress endpoint, int backlog) throws java.io.IOException;
        public void close() throws java.io.IOException;
    // Public Methods Overriding Object
        public String toString();
    // Protected Instance Methods
    1.1 protected final void implAccept(Socket s) throws java.io.IOException;
}
```

Subclasses: `javax.net.ssl.SSLServerSocket`

Returned By: `java.nio.channels.ServerSocketChannel.socket()`,
`java.rmi.server.RMIServerSocketFactory.createServerSocket()`,
`java.rmi.server.RMISocketFactory.createServerSocket()`,
`javax.net.ServerSocketFactory.createServerSocket()`

Socket

Java 1.0

java.net

This class implements a socket for stream-based communication over the network. See [URL](#) for a higher-level interface to networking and [DatagramSocket](#) for a lower-level interface.

Before you can use a socket for communication, it must be *bound* to a local address and *connected* to a remote address. Binding and connection are done automatically when you call any of the `Socket()` constructors, except the no-argument constructor. These constructors allow you to specify either the name or the `InetAddress` of the computer to connect to and require you to specify the port number to connect to. Two of these constructors also allow you to specify the local `InetAddress` and port number to bind the socket to. Most applications do not need to specify a local address and can simply use one of the two-argument versions of `Socket()` and allow the constructor to choose an ephemeral local port to bind the socket to.

The no-argument `Socket()` constructor is different from the others; it creates an unbound and unconnected socket. In Java 1.4 and later, you can explicitly call `bind()` and `connect()` to bind and connect the socket. It can be useful to do this when you want to set a socket option (described later) that must be set before binding or connection. `bind()` uses a `SocketAddress` object to describe the local address to bind to, and `connect()` uses a `SocketAddress` to specify the remote address to connect to. There is also a version of `connect()` that takes a timeout value in milliseconds: if the connection attempt takes longer than the specified amount of time, `connect()` throws an `IOException`. See [ServerSocket](#) for a description of how to write server code that accepts socket connection requests from client code.

Use `isBound()` and `isConnected()` to determine whether a `Socket` is bound and connected. Use `getInetAddress()` and `getPort()` to determine the IP address and port number that the socket is connected to. In Java 1.4, use `getRemoteSocketAddress()` to obtain the remote address as a `SocketAddress` object. Similarly, use `getLocalAddress()` and `getLocalPort()`, or `getLocalSocketAddress()`, to find the address a socket is bound to.

Once you have a `Socket` object that is bound and connected, use `getInputStream()` and `getOutputStream()` to obtain `InputStream` and `OutputStream` objects you can use to communicate with the remote host. Use these streams just as you would similar streams for file input and output. When you are done with a `Socket`, use `close()` to close it. Once a socket has been closed, it is not possible to call `connect()` again to reuse it, and you should not call any of its methods except `isClosed()`. Because networking code can throw many exceptions, it is common practice to `close()` a socket in the `finally` clause of a `try/catch` statement to ensure that the socket always gets closed. Note, however, that the `close()` method itself can throw an `IOException`, and you may need to put it in its own `try` block. In Java 1.3 and later, `shutdownInput()` and `shutdownOutput()` allow you to close the input and output communication channels individually without closing the entire socket. In Java 1.4, `isInputShutdown()` and `isOutputShutdown()` allow you to test for this.

The `Socket` class defines a number of methods that allow you to set (and query) “socket options” that affect the low-level networking behavior of the socket. `setSendBufferSize()` and `setReceiveBufferSize()` provide hints to the underlying networking system as to what buffer size is best to use with this socket. `setSoTimeout()` specifies the number of milliseconds a `read()` call on the input stream returned by `getInputStream()` waits for data before throwing an `InterruptedException`. The default value of 0 specifies that the stream blocks indefinitely. `setSoLinger()` specifies what to do when a socket is closed while there is still data waiting to be transmitted. If lingering is turned on, the `close()` call blocks for up to the specified number of seconds while attempting to transmit the remaining data. Call-

ing `setTcpNoDelay()` with an argument of `true` causes data to be sent through the socket as soon as it is available, instead of waiting for the TCP packet to fill up more before sending it. In Java 1.3, use `setKeepAlive()` to enable or disable the periodic exchange of control messages across an idle socket connection. The keepalive protocol enables a client to determine if its server has crashed without closing the socket and vice versa. In Java 1.4, pass `true` to `setOOBInline()` if you want to receive “out of band” data sent to this socket “inline” on the input stream of the socket (by default, such data is simply discarded). This can be used to receive bytes sent with `sendUrgentData()`. Java 1.4 also adds `setReuseAddress()`, which you can use before binding the socket to specify that the socket should be allowed to bind to a port that is still nominally in use by another socket in the process of shutting down. `setTrafficClass()` is also new in Java 1.4; it sets the “traffic class” field for the socket and requires an understanding of the low-level details of the IP protocol.

The `getChannel()` method is a link between this `Socket` class and the New I/O `java.nio.channels.SocketChannel` class. It returns the `SocketChannel` associated with this `Socket` if there is one. Note, however, that this method always returns `null` for sockets created with any of the `Socket()` constructors. If you create a `SocketChannel` object, and obtain a `Socket` from it, then the `getChannel()` method provides a way to link back to the parent channel.

```
public class Socket {
// Public Constructors
1.1 public Socket();
    public Socket(InetAddress address, int port) throws java.io.IOException;
    public Socket(String host, int port) throws java.net.UnknownHostException, java.io.IOException;
#   public Socket(InetAddress host, int port, boolean stream) throws java.io.IOException;
#   public Socket(String host, int port, boolean stream) throws java.io.IOException;
1.1 public Socket(InetAddress address, int port, InetAddress localAddr, int localPort) throws java.io.IOException;
1.1 public Socket(String host, int port, InetAddress localAddr, int localPort) throws java.io.IOException;
// Protected Constructors
1.1 protected Socket(SocketImpl impl) throws SocketException;
// Public Class Methods
    public static void setSocketImplFactory(SocketImplFactory fac) throws java.io.IOException;           synchronized
// Property Accessor Methods (by property name)
1.4 public boolean isBound();                                     default:false
1.4 public java.nio.channels.SocketChannel getChannel();          constant default:null
1.4 public boolean isClosed();                                    default:false
1.4 public boolean isConnected();                                default:false
    public InetAddress getInetAddress();                          default:null
1.4 public boolean isInputShutdown();                             default:false
    public java.io.InputStream getInputStream() throws java.io.IOException;
1.3 public boolean getKeepAlive() throws SocketException;       default:false
1.3 public void setKeepAlive(boolean on) throws SocketException;
1.1 public InetAddress getLocalAddress();                         default:Inet4Address
    public int getLocalPort();                                    default:-1
1.4 public SocketAddress getLocalSocketAddress();               default:null
1.4 public boolean getOOBInline() throws SocketException;       default:false
1.4 public void setOOBInline(boolean on) throws SocketException;
1.4 public boolean isOutputShutdown();                           default:false
    public java.io.OutputStream getOutputStream() throws java.io.IOException;
    public int getPort();                                         default:0
1.2 public int getReceiveBufferSize() throws SocketException;   synchronized default:32767
1.2 public void setReceiveBufferSize(int size) throws SocketException;   synchronized
```

Socket

```
1.4 public SocketAddress getRemoteSocketAddress(); default:null
1.4 public boolean getReuseAddress() throws SocketException; default:false
1.4 public void setReuseAddress(boolean on) throws SocketException;
1.2 public int getSendBufferSize() throws SocketException; synchronized default:32767
1.2 public void setSendBufferSize(int size) throws SocketException; synchronized
1.1 public int getSoLinger() throws SocketException; default:-1
1.1 public int getSoTimeout() throws SocketException; synchronized default:0
1.1 public void setSoTimeout(int timeout) throws SocketException; synchronized
1.1 public boolean getTcpNoDelay() throws SocketException; default:false
1.1 public void setTcpNoDelay(boolean on) throws SocketException;
1.4 public int getTrafficClass() throws SocketException; default:0
1.4 public void setTrafficClass(int tc) throws SocketException;
// Public Instance Methods
1.4 public void bind(SocketAddress bindpoint) throws java.io.IOException;
    public void close() throws java.io.IOException; synchronized
1.4 public void connect(SocketAddress endpoint) throws java.io.IOException;
1.4 public void connect(SocketAddress endpoint, int timeout) throws java.io.IOException;
1.4 public void sendUrgentData(int data) throws java.io.IOException;
1.1 public void setSoLinger(boolean on, int linger) throws SocketException;
1.3 public void shutdownInput() throws java.io.IOException;
1.3 public void shutdownOutput() throws java.io.IOException;
// Public Methods Overriding Object
    public String toString();
}
```

Subclasses: javax.net.ssl.SSLSocket

Passed To: ServerSocket.implAccept(), javax.net.ssl.SSLSocketFactory.createSocket(),
javax.net.ssl.X509KeyManager.chooseClientAlias(), chooseServerAlias()

Returned By: ServerSocket.accept(), java.nio.channels.SocketChannel.socket(),
java.rmi.server.RMIClientSocketFactory.createSocket(), java.rmi.server.RMISocketFactory.createSocket(),
javax.net.SocketFactory.createSocket(), javax.net.ssl.SSLSocketFactory.createSocket()

SocketAddress

Java 1.4

java.net

serializable

Instances of this abstract class are opaque representations of network socket addresses. The only concrete subclass in the core Java platform is `InetSocketAddress`, which represents an Internet address and port number. See `InetSocketAddress`.



```
public abstract class SocketAddress implements Serializable {
// Public Constructors
    public SocketAddress();
}
```

Subclasses: InetSocketAddress

Passed To: Too many methods to list.

Returned By: DatagramPacket.getSocketAddress(), DatagramSocket.{getLocalSocketAddress(),
getRemoteSocketAddress()}, ServerSocket.getLocalSocketAddress(), Socket.{getLocalSocketAddress(),
getRemoteSocketAddress()}, java.nio.channels.DatagramChannel.receive()

SocketException

Java 1.0

java.net

serializable checked

This exception signals an exceptional condition while using a socket.



```

public class SocketException extends java.io.IOException {
// Public Constructors
    public SocketException();
    public SocketException(String msg);
}
  
```

Subclasses: BindException, java.net.ConnectException, NoRouteToHostException, PortUnreachableException

Thrown By: Too many methods to list.

SocketImpl

Java 1.0

java.net

This abstract class defines the methods necessary to implement communication through sockets. Different subclasses of this class may provide different implementations suitable in different environments (such as behind firewalls). These socket implementations are used by the `Socket` and `ServerSocket` classes. Normal applications never need to use or subclass this class.



```

public abstract class SocketImpl implements SocketOptions {
// Public Constructors
    public SocketImpl();
// Public Methods Overriding Object
    public String toString();
// Protected Instance Methods
    protected abstract void accept(SocketImpl s) throws java.io.IOException;
    protected abstract int available() throws java.io.IOException;
    protected abstract void bind(InetAddress host, int port) throws java.io.IOException;
    protected abstract void close() throws java.io.IOException;
    protected abstract void connect(String host, int port) throws java.io.IOException;
    protected abstract void connect(InetAddress address, int port) throws java.io.IOException;
    1.4 protected abstract void connect(SocketAddress address, int timeout) throws java.io.IOException;
    protected abstract void create(boolean stream) throws java.io.IOException;
    protected java.io.FileDescriptor getFileDescriptor();
    protected InetAddress getInetAddress();
    protected abstract java.io.InputStream getInputStream() throws java.io.IOException;
    protected int getLocalPort();
    protected abstract java.io.OutputStream getOutputStream() throws java.io.IOException;
    protected int getPort();
    protected abstract void listen(int backlog) throws java.io.IOException;
    1.4 protected abstract void sendUrgentData(int data) throws java.io.IOException;
    1.3 protected void shutdownInput() throws java.io.IOException;
    1.3 protected void shutdownOutput() throws java.io.IOException;
    1.4 protected boolean supportsUrgentData();
}
  
```

constant

SocketImpl

```
// Protected Instance Fields
protected InetAddress address;
protected java.io.FileDescriptor fd;
protected int localport;
protected int port;
}
```

Passed To: Socket.Socket(), SocketImpl.accept()

Returned By: SocketImplFactory.createSocketImpl()

SocketImplFactory

Java 1.0

java.net

This interface defines a method that creates `SocketImpl` objects. `SocketImplFactory` objects may be registered to create `SocketImpl` objects for the `Socket` and `ServerSocket` classes. Normal applications never need to use or implement this interface.

```
public interface SocketImplFactory {
    // Public Instance Methods
    public abstract SocketImpl createSocketImpl();
}
```

Passed To: ServerSocket.setSocketFactory(), Socket.setSocketImplFactory()

SocketOptions

Java 1.2

java.net

This interface defines constants that represent low-level BSD Unix-style socket options and methods that set and query the value of those options. In Java 1.2, `SocketImpl` and `DatagramSocketImpl` implement this interface. Any custom socket implementations you define should also provide meaningful implementations for the `getOption()` and `setOption()` methods. Your implementation may support options other than those defined here. Only custom socket implementations need to use this interface. All other code can use methods defined by `Socket`, `ServerSocket`, `DatagramSocket`, and `MulticastSocket` to set specific socket options for those socket types.

```
public interface SocketOptions {
    // Public Constants
    public static final int IP_MULTICAST_IF;                =16
    1.4 public static final int IP_MULTICAST_IF2;           =31
    1.4 public static final int IP_MULTICAST_LOOP;          =18
    1.4 public static final int IP_TOS;                     =3
    public static final int SO_BINDADDR;                   =15
    1.4 public static final int SO_BROADCAST;               =32
    1.3 public static final int SO_KEEPAIVE;                 =8
    public static final int SO_LINGER;                      =128
    1.4 public static final int SO_OOBINLINE;               =4099
    public static final int SO_RCVBUF;                     =4098
    public static final int SO_REUSEADDR;                  =4
    public static final int SO_SNDBUF;                     =4097
    public static final int SO_TIMEOUT;                    =4102
    public static final int TCP_NODELAY;                   =1
    // Public Instance Methods
    public abstract Object getOption(int optID) throws SocketException;
}
```

```
public abstract void setOption(int optID, Object value) throws SocketException;
}
```

Implementations: DatagramSocketImpl, SocketImpl

SocketPermission

Java 1.2

java.net

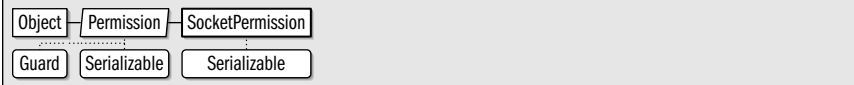
serializable permission

This class is a `java.security.Permission` that governs all networking operations performed with sockets. Like all permissions, a `SocketPermission` consists of a name, or target, and a list of actions that may be performed on that target. The target of a `SocketPermission` is the host and, optionally, the port or ports for which permission is being granted or requested. The target consists of a hostname optionally followed by a colon and a port specification. The host may be a DNS domain name, a numerical IP address, or the string "localhost". If you specify a host domain name, you may use * as a wildcard as the leftmost portion of the hostname. The port specification, if present, must be a single port number or a range of port numbers in the form `n1-n2`. If `n1` is omitted, it is taken to be 0, and if `n2` is omitted, it is taken to be 65535. If no port is specified, the socket permission applies to all ports of the specified host. Here are some legal `SocketPermission` targets:

```
java.sun.com:80
*.sun.com:1024-2000
*:1024-
localhost:1023
```

In addition to a target, each `SocketPermission` must have a comma-separated list of actions, which specify the operations that may be performed on the specified host(s) and port(s). The available actions are "connect", "accept", "listen", and "resolve". "connect" represents permission to connect to the specified target. "accept" indicates permission to accept connections from the specified target. "listen" represents permission to listen on the specified ports for connection requests. This action is only valid when used for ports on "localhost". Finally, the "resolve" action indicates permission to use the DNS name service to resolve domain names into IP addresses. This action is required for and implied by all other actions.

System administrators configuring security policies must be familiar with this class and understand the risks of granting the various permissions it represents. System programmers writing new low-level networking libraries or connecting to native code that performs networking may need to use this class. Application programmers, however, should never need to use it directly.



```
public final class SocketPermission extends java.security.Permission implements Serializable {
    // Public Constructors
    public SocketPermission(String host, String action);
    // Public Methods Overriding Permission
    public boolean equals(Object obj);
    public String getActions();
    public int hashCode();
    public boolean implies(java.security.Permission p);
    public java.security.PermissionCollection newPermissionCollection();
}
```


SocketTimeoutException

Java 1.4

java.net

serializable checked

This exception signals that a timeout value was exceeded for a socket read or accept operation. See the `setSoTimeout()` method of `Socket`.



```

public class SocketTimeoutException extends java.io.InterruptedIOException {
// Public Constructors
    public SocketTimeoutException();
    public SocketTimeoutException(String msg);
}
  
```

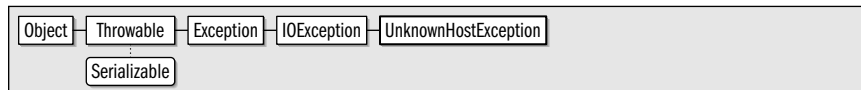
UnknownHostException

Java 1.0

java.net

serializable checked

This exception signals that the name of a specified host could not be resolved.



```

public class UnknownHostException extends java.io.IOException {
// Public Constructors
    public UnknownHostException();
    public UnknownHostException(String host);
}
  
```

Thrown By: `InetAddress.getAllByName()`, `InetAddress.getByAddress()`, `InetAddressByName()`, `InetAddress.getLocalHost()`, `Socket.Socket()`, `javax.net.SocketFactory.createSocket()`, `javax.net.ssl.SSLSocket.SSLSocket()`

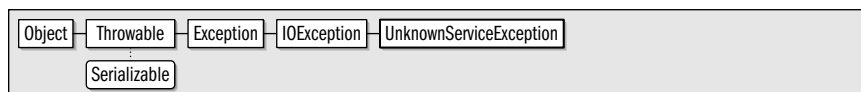
UnknownServiceException

Java 1.0

java.net

serializable checked

This exception signals an attempt to use an unsupported service of a network connection.



```

public class UnknownServiceException extends java.io.IOException {
// Public Constructors
    public UnknownServiceException();
    public UnknownServiceException(String msg);
}
  
```

URI

Java 1.4

java.net

serializable comparable

The `URI` class is an immutable representation of a Uniform Resource Identifier (URI). A URI is a generalization of the Uniform Resource Locators (URLs) used on the World Wide Web. The `URI` supports parsing and textual manipulation of URI strings but does not have any direct networking capabilities the way that the `URL` class does. The advantages of the `URI` class over the `URL` class are that the `URI` class provides more general facilities for parsing and manipulating URLs than the `URL` class; it can represent relative

URIs, which do not include a scheme (or protocol); and it can manipulate URIs that include unsupported or even unknown schemes.

Obtain a URI with one of the constructors, which allow a URI to be parsed from a single string, or allow the specification of the individual components of a URI. These constructors can throw `URISyntaxException`, which is a checked exception. When using hard-coded URIs (rather than URIs based on user input), you may prefer to use the static `create()` method, which does not throw any checked exceptions.

Once you have created a URI object, you can use the various `get` methods to query the various portions of the URI. The `getRaw()` methods are like the `get()` methods, except that they do not decode hexadecimal escape sequences of the form `%xx` that appear in the URI. `normalize()` returns a new URI object that has “.” and unnecessary “..” sequences removed from its path component. `resolve()` interprets its URI (or string) argument relative to this URI and returns the result. `relativize()` performs the reverse operation. It returns a new URI that represents the same resource as the specified URI argument but is relative to this URI. Finally, the `toURL()` method converts an absolute URI object to the equivalent URL. Since the URI class provides superior textual manipulation capabilities for URLs, it can be useful to use the URI class to resolve relative URLs (for example) and then convert those URI objects to URL objects when they are ready for networking.



```
public final class URI implements Comparable, Serializable {
    // Public Constructors
    public URI(String str) throws URISyntaxException;
    public URI(String scheme, String ssp, String fragment) throws URISyntaxException;
    public URI(String scheme, String host, String path, String fragment) throws URISyntaxException;
    public URI(String scheme, String authority, String path, String query, String fragment) throws URISyntaxException;
    public URI(String scheme, String userInfo, String host, int port, String path, String query, String fragment)
        throws URISyntaxException;

    // Public Class Methods
    public static URI create(String str);

    // Property Accessor Methods (by property name)
    public boolean isAbsolute();
    public String getAuthority();
    public String getFragment();
    public String getHost();
    public boolean isOpaque();
    public String getPath();
    public int getPort();
    public String getQuery();
    public String getRawAuthority();
    public String getRawFragment();
    public String getRawPath();
    public String getRawQuery();
    public String getRawSchemeSpecificPart();
    public String getRawUserInfo();
    public String getScheme();
    public String getSchemeSpecificPart();
    public String getUserInfo();

    // Public Instance Methods
    public URI normalize();
    public URI parseServerAuthority() throws URISyntaxException;
    public URI relativize(URI uri);
}
```

URI

```
public URI resolve(URI uri);
public URI resolve(String str);
public String toASCIIString();
public URL toURL() throws MalformedURLException;
// Methods Implementing Comparable
public int compareTo(Object ob);
// Public Methods Overriding Object
public boolean equals(Object ob);
public int hashCode();
public String toString();
}
```

Passed To: java.io.File.File(), URI.{relativize(), resolve()}, javax.print.attribute.URISyntax.URISyntax(), javax.print.attribute.standard.Destination.Destination(), javax.print.attribute.standard.PrinterMoreInfo.PrinterMoreInfo(), javax.print.attribute.standard.PrinterMoreInfoManufacturer.PrinterMoreInfoManufacturer(), javax.print.attribute.standard.PrinterURI.PrinterURI()

Returned By: java.io.File.toURI(), URI.{create(), normalize(), parseServerAuthority(), relativize(), resolve()}, javax.print.URIException.getUnsupportedURI(), javax.print.attribute.URISyntax.getURI()

URISyntaxException

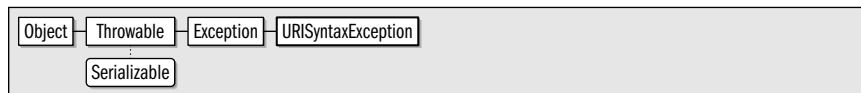
Java 1.4

java.net

serializable checked

This class signals that a string could not be parsed as a valid URI. `getInput()` returns the string that could not be parsed. `getReason()` returns an error message. `getIndex()` returns the character position at which the syntax error occurred, if that information is available. `getMessage()` returns a human-readable string that includes the information from each of the other three methods.

This is a checked exception thrown by all the `URI()` constructors. If you are parsing a hardcoded URI that you do not believe contains any syntax errors and wish to avoid the checked exception, you can use the `URI.create()` factory method instead of the one-argument version of the `URI()` constructor.



```
public class URISyntaxException extends Exception {
// Public Constructors
public URISyntaxException(String input, String reason);
public URISyntaxException(String input, String reason, int index);
// Public Instance Methods
public int getIndex();
public String getInput();
public String getReason();
// Public Methods Overriding Throwable
public String getMessage();
}
```

Thrown By: URI.{parseServerAuthority(), URI()}

URL

Java 1.0

java.net

serializable

This class represents a uniform resource locator and allows the data referred to by the URL to be downloaded. A URL can be specified as a single string or with separate

protocol, host, port, and file specifications. Relative URLs can also be specified with a `String` and the `URL` object to which it is relative. `getFile()`, `getHost()`, `getProtocol()` and related methods return the various portions of the URL specified by a `URL` object. `sameFile()` determines whether a `URL` object refers to the same file as this one. `getDefaultPort()` returns the default port number for the protocol of the `URL` object; it may differ from the number returned by `getPort()`. Use `openConnection()` to obtain a `URLConnection` object with which you can download the content of the URL. For simple cases, however, the `URL` class defines shortcut methods that create and invoke methods on a `URLConnection` internally. `getContent()` downloads the URL data and parses it into an appropriate Java object (such as a string or image) if an appropriate `ContentHandler` can be found. In Java 1.3 and later, you can pass an array of `Class` objects that specify the type of objects that you are willing to accept as the return value of this method. If you wish to parse the URL content yourself, call `openStream()` to obtain an `InputStream` from which you can read the data.

Object	URL	Serializable
--------	-----	--------------

```

public final class URL implements Serializable {
// Public Constructors
    public URL(String spec) throws MalformedURLException;
    public URL(URL context, String spec) throws MalformedURLException;
    public URL(String protocol, String host, String file) throws MalformedURLException;
    1.2 public URL(URL context, String spec, URLStreamHandler handler) throws MalformedURLException;
    public URL(String protocol, String host, int port, String file) throws MalformedURLException;
    1.2 public URL(String protocol, String host, int port, String file, URLStreamHandler handler)
        throws MalformedURLException;

// Public Class Methods
    public static void setURLStreamHandlerFactory(URLStreamHandlerFactory fac);
// Property Accessor Methods (by property name)
    1.3 public String getAuthority();
    public final Object getContent() throws java.io.IOException;
    1.3 public final Object getContent(Class[] classes) throws java.io.IOException;
    1.4 public int getDefaultPort();
    public String getFile();
    public String getHost();
    1.3 public String getPath();
    public int getPort();
    public String getProtocol();
    1.3 public String getQuery();
    public String getRef();
    1.3 public String getUserInfo();
// Public Instance Methods
    public URLConnection openConnection() throws java.io.IOException;
    public final java.io.InputStream openStream() throws java.io.IOException;
    public boolean sameFile(URL other);
    public String toExternalForm();
// Public Methods Overriding Object
    public boolean equals(Object obj);
    public int hashCode();
    public String toString();
// Protected Instance Methods
    protected void set(String protocol, String host, int port, String file, String ref);
    1.3 protected void set(String protocol, String host, int port, String authority, String userInfo, String path, String query,
        String ref);
}

```

URL

Passed To: Too many methods to list.

Returned By: Too many methods to list.

Type Of: URLConnection.url

URLClassLoader

Java 1.2

java.net

This `ClassLoader` provides a useful way to load untrusted Java code from a search path of arbitrary URLs, where each URL represents a directory or JAR file to search. Use the inherited `loadClass()` method to load a named class with a `URLClassLoader`. Classes loaded by a `URLClassLoader` have whatever permissions are granted to their `java.security.CodeSource` by the system `java.security.Policy`, plus they have one additional permission that allows the class loader to read any resource files associated with the class. If the class is loaded from a local file: URL that represents a directory, the class is given permission to read all files and directories below that directory. If the class is loaded from a local file: URL that represents a JAR file, the class is given permission to read that JAR file. If the class is loaded from a URL that represents a resource on another host, that class is given permission to connect to and accept network connections from that host. Note, however, that loaded classes are not granted this additional permission if the code that created the `URLClassLoader` in the first place would not have had that permission.

You can obtain a `URLClassLoader` by calling one of the `URLClassLoader()` constructors or one of the static `newInstance()` methods. If you call `newInstance()`, the `loadClass()` method of the returned `URLClassLoader` performs an additional check to ensure that the caller has permission to access the specified package.

Object	ClassLoader	SecureClassLoader	URLClassLoader
--------	-------------	-------------------	----------------

```
public class URLClassLoader extends java.security.SecureClassLoader {  
    // Public Constructors  
    public URLClassLoader(URL[] urls);  
    public URLClassLoader(URL[] urls, ClassLoader parent);  
    public URLClassLoader(URL[] urls, ClassLoader parent, URLStreamHandlerFactory factory);  
    // Public Class Methods  
    public static URLClassLoader newInstance(URL[] urls);  
    public static URLClassLoader newInstance(URL[] urls, ClassLoader parent);  
    // Public Instance Methods  
    public URL[] getURLs();  
    // Protected Methods Overriding SecureClassLoader  
    protected java.security.PermissionCollection getPermissions(java.security.CodeSource codesource);  
    // Public Methods Overriding ClassLoader  
    public URL findResource(String name);  
    public java.util.Enumeration findResources(String name) throws java.io.IOException;  
    // Protected Methods Overriding ClassLoader  
    protected Class findClass(String name) throws ClassNotFoundException;  
    // Protected Instance Methods  
    protected void addURL(URL url);  
    protected Package definePackage(String name, java.util.jar.Manifest man, URL url)  
        throws IllegalArgumentException;  
}
```

Returned By: `URLClassLoader.newInstance()`

URLConnection**Java 1.0****java.net**

This abstract class defines a network connection to an object specified by a URL. `URL.openConnection()` returns a `URLConnection` instance. You should use a `URLConnection` object when you want more control over the downloading of data than is available through the simpler `URL` methods. `connect()` actually establishes the network connection. Some methods must be called before the connection is made, and others depend on being connected. The methods that depend on being connected call `connect()` themselves if no connection exists yet, so you never need to call this method explicitly. The `getContent()` methods are just like the same-named methods of the `URL` class; they download the data referred to by the URL and parse it into an appropriate type of object (such as a string or an image). In Java 1.3 and later, there is a version of `getContent()` that allows you to specify the types of parsed objects you are willing to accept by passing an array of `Class` objects. If you prefer to parse the URL content yourself instead of calling `getContent()`, you can call `getInputStream()` (and `getOutputStream()` if the URL protocol supports writing) to obtain a stream through which you can read (or write) data from (or to) the resource identified by the URL.

Before a connection is established, you may want to set request fields (such as HTTP request headers) to refine the URL request. Use `setRequestProperty()` to set a new value for a named header. In Java 1.4 and later, you can use `addRequestProperty()` to add a new comma-separated item to an existing header. Java 1.4 also adds `getRequestProperties()`, which returns the current set of request properties in the form of an unmodifiable `Map` object that maps request header names to `List` objects that contain the string value or values for the named header.

Once a connection has been established, there are a number of methods you can call to obtain information from the “response headers” of the URL. `getContentLength()`, `getContentType()`, `getContentEncoding()`, `getExpiration()`, `getDate()`, and `getLastModified()` return the appropriate information about the object referred to by the URL, if that information can be determined (e.g., from HTTP header fields). `getHeaderField()` returns an HTTP header field specified by name or by number. `getHeaderFieldInt()` and `getHeaderFieldDate()` return the value of a named header field parsed as an integer or a date. In Java 1.4 and later, `getHeaderFields()` returns an unmodifiable `Map` object that maps response header names to an unmodifiable `List` that contains the string value or values for the named header.

There are a number of options you can specify to control how the `URLConnection` behaves. These options are set with the various `set()` methods and may be queried with corresponding `get()` methods. The options must be set before the `connect()` method is called. `setDoInput()` and `setDoOutput()` allow you to specify whether you are using the `URLConnection` for input and/or output (input-only by default). `setAllowUserInteraction()` specifies whether user interaction (such as typing a password) is allowed during the data transfer (`false` by default). `setDefaultAllowUserInteraction()` is a class method that allows you to change the default value for user interaction. `setUseCaches()` allows you to specify whether a cached version of the URL can be used. You can set this to `false` to force a URL to be reloaded. `setDefaultUseCaches()` sets the default value for `setUseCaches()`. `setIfModifiedSince()` allows you to specify that a URL should not be fetched unless it has been modified since a specified time (if it is possible to determine its modification date).

```
public abstract class URLConnection {
    // Protected Constructors
    protected URLConnection(URL url);
    // Public Class Methods
    public static boolean getDefaultAllowUserInteraction();
```

URLConnection

```
1.1 public static FileNameMap getFileNameMap(); synchronized
    public static String guessContentTypeFromName(String fname);
    public static String guessContentTypeFromStream(java.io.InputStream is) throws java.io.IOException;
    public static void setContentHandlerFactory(ContentHandlerFactory fac); synchronized
    public static void setDefaultAllowUserInteraction(boolean defaultallowuserinteraction);
1.1 public static void setFileNameMap(FileNameMap map);
// Property Accessor Methods (by property name)
    public boolean getAllowUserInteraction();
    public void setAllowUserInteraction(boolean allowuserinteraction);
    public Object getContent() throws java.io.IOException;
1.3 public Object getContent(Class[] classes) throws java.io.IOException;
    public String getContentEncoding();
    public int getContentLength();
    public String getContentType();
    public long getDate();
    public boolean getDefaultUseCaches();
    public void setDefaultUseCaches(boolean defaultusecaches);
    public boolean getDoInput();
    public void setDoInput(boolean doinput);
    public boolean getDoOutput();
    public void setDoOutput(boolean dooutput);
    public long getExpiration();
1.4 public java.util.Map getHeaderFields();
    public long getIfModifiedSince();
    public void setIfModifiedSince(long ifmodifiedsince);
    public java.io.InputStream getInputStream() throws java.io.IOException;
    public long getLastModified();
    public java.io.OutputStream getOutputStream() throws java.io.IOException;
1.2 public java.security.Permission getPermission() throws java.io.IOException;
1.4 public java.util.Map getRequestProperties();
    public URL getURL();
    public boolean getUseCaches();
    public void setUseCaches(boolean usecaches);
// Public Instance Methods
1.4 public void addRequestProperty(String key, String value);
    public abstract void connect() throws java.io.IOException;
    public String getHeaderField(String name); constant
    public String getHeaderField(int n); constant
    public long getHeaderFieldDate(String name, long Default);
    public int getHeaderFieldInt(String name, int Default);
    public String getHeaderFieldKey(int n); constant
    public String getRequestProperty(String key);
    public void setRequestProperty(String key, String value);
// Public Methods Overriding Object
    public String toString();
// Protected Instance Fields
    protected boolean allowUserInteraction;
    protected boolean connected;
    protected boolean doInput;
    protected boolean doOutput;
    protected long ifModifiedSince;
    protected URL url;
    protected boolean useCaches;
// Deprecated Public Methods
# public static String getDefaultRequestProperty(String key); constant
```

```
# public static void setDefaultRequestProperty(String key, String value); empty
}
```

Subclasses: HttpURLConnection, JarURLConnection

Passed To: java.net.ContentHandler.getContent()

Returned By: URL.openConnection(), URLStreamHandler.openConnection()

Type Of: JarURLConnection, JarURLConnection

URLDecoder**Java 1.2**

java.net

This class defines a static `decode()` method that reverses the encoding performed by `URLEncoder.encode()`. It decodes 8-bit text with the MIME type “x-www-form-urlencoded”, which is a standard encoding used by web browsers to submit form contents to CGI scripts and other server-side programs.

```
public class URLDecoder {
    // Public Constructors
    public URLDecoder();
    // Public Class Methods
    1.4 public static String decode(String s, String enc) throws java.io.UnsupportedEncodingException;
    // Deprecated Public Methods
    # public static String decode(String s);
}
```

URLEncoder**Java 1.0**

java.net

This class defines a single static method that converts a string to its URL-encoded form. That is, spaces are converted to +, and nonalphanumeric characters other than underscore are output as two hexadecimal digits following a percent sign. Note that this technique works only for 8-bit characters. This method canonicalizes a URL specification so that it uses only characters from an extremely portable subset of ASCII that can be correctly handled by computers around the world.

```
public class URLEncoder {
    // No Constructor
    // Public Class Methods
    1.4 public static String encode(String s, String enc) throws java.io.UnsupportedEncodingException;
    // Deprecated Public Methods
    # public static String encode(String s);
}
```

URLStreamHandler**Java 1.0**

java.net

This abstract class defines the `openConnection()` method that creates a `URLConnection` for a given URL. A separate subclass of this class may be defined for various URL protocol types. A `URLStreamHandler` is created by a `URLStreamHandlerFactory`. Normal applications never need to use or subclass this class.

```
public abstract class URLStreamHandler {
    // Public Constructors
    public URLStreamHandler();
    // Protected Instance Methods
```


URLStreamHandler

```
1.3 protected boolean equals(URL u1, URL u2);
1.3 protected int getDefaultPort(); constant
1.3 protected InetAddress getHostAddress(URL u); synchronized
1.3 protected int hashCode(URL u);
1.3 protected boolean hostsEqual(URL u1, URL u2);
    protected abstract URLConnection openConnection(URL u) throws java.io.IOException;
    protected void parseURL(URL u, String spec, int start, int limit);
1.3 protected boolean sameFile(URL u1, URL u2);
1.3 protected void setURL(URL u, String protocol, String host, int port, String authority, String userInfo, String path,
    String query, String ref);
    protected String toExternalForm(URL u);
// Deprecated Protected Methods
# protected void setURL(URL u, String protocol, String host, int port, String file, String ref);
}
```

Passed To: URL.URL()

Returned By: URLStreamHandlerFactory.createURLStreamHandler()

URLStreamHandlerFactory

Java 1.0

java.net

This interface defines a method that creates a `URLStreamHandler` object for a specified protocol. Normal applications never need to use or implement this interface.

```
public interface URLStreamHandlerFactory {
// Public Instance Methods
    public abstract URLStreamHandler createURLStreamHandler(String protocol);
}
```

Passed To: URL.setURLStreamHandlerFactory(), URLClassLoader.URLClassLoader()