```
/*
 * @(#)Socket.java      1.108 04/05/18
 *
 * Copyright 2004 Sun Microsystems, Inc. All rights reserved.
 * SUN PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
 */

package java.net;

import java.io.InputStream;
import java.io.OutputStream;
import java.io.IOException;
import java.io.InterruptedIOException;
import java.nio.channels.SocketChannel;
import java.security.AccessController;
import java.security.PrivilegedExceptionAction;

/**
 * This class implements client sockets (also called just
 * "sockets"). A socket is an endpoint for communication
 * between two machines.
 * <p>
 * The actual work of the socket is performed by an instance of the
 * <code>SocketImpl</code> class. An application, by changing
 * the socket factory that creates the socket implementation,
 * can configure itself to create sockets appropriate to the local
 * firewall.
 *
 * @author  unascribed
 * @version 1.108, 05/18/04
 * @see     java.net.Socket#setSocketImplFactory(java.net.SocketImplFactory)
 * @see     java.net.SocketImpl
 * @see     java.nio.channels.SocketChannel
 * @since   JDK1.0
 */
public
class Socket {
    /**
     * Various states of this socket.
     */
    private boolean created = false;
    private boolean bound = false;
    private boolean connected = false;
    private boolean closed = false;
    private Object closeLock = new Object();
    private boolean shutIn = false;
    private boolean shutOut = false;

    /**
     * The implementation of this Socket.
     */
    SocketImpl impl;

    /**
     * Are we using an older SocketImpl?
     */
    private boolean oldImpl = false;

    /**
     * Creates an unconnected socket, with the
```

```
     * @since   JDK1.1
     * @revised 1.4
     */
    public Socket() {
        setImpl();
    }

    /**
     * Creates an unconnected socket, specifying the type of proxy, if any,
     * that should be used regardless of any other settings.
     * <P>
     * If there is a security manager, its <code>checkConnect</code> method
     * is called with the proxy host address and port number
     * as its arguments. This could result in a SecurityException.
     * <P>
     * Examples:
     * <UL> <LI><code>Socket s = new Socket(Proxy.NO_PROXY);</code> will create
     * a plain socket ignoring any other proxy configuration.</LI>
     * <LI><code>Socket s = new Socket(new Proxy(Proxy.Type.SOCKS, new
InetSocketAddress("socks.mydom.com", 1080)));</code>
     * will create a socket connecting through the specified SOCKS proxy
     * server.</LI>
     * </UL>
     *
     * @param proxy a {@link java.net.Proxy Proxy} object specifying what kind
     *              of proxying should be used.
     * @throws IllegalArgumentException if the proxy is of an invalid type
     *          or <code>null</code>.
     * @throws SecurityException if a security manager is present and
     *                           permission to connect to the proxy is
     *                           denied.
     * @see java.net.ProxySelector
     * @see java.net.Proxy
     *
     * @since   1.5
     */
    public Socket(Proxy proxy) {
        if (proxy != null && proxy.type() == Proxy.Type.SOCKS) {
            SecurityManager security = System.getSecurityManager();
            InetSocketAddress epoint = (InetSocketAddress) proxy.address();
            if (security != null) {
                if (epoint.isUnresolved())
                    security.checkConnect(epoint.getHostName(),
                                          epoint.getPort());
                else
                    security.checkConnect(epoint.getAddress().getHostAddress(),
                                          epoint.getPort());
            }
            impl = new SocksSocketImpl(proxy);
            impl.setSocket(this);
        } else {
            if (proxy == Proxy.NO_PROXY) {
                if (factory == null) {
                    impl = new PlainSocketImpl();
                    impl.setSocket(this);
                } else
                    setImpl();
            } else
                throw new IllegalArgumentException("Invalid Proxy");
        }
    }
```

```
    /**
     * Creates an unconnected Socket with a user-specified
     * SocketImpl.
     * <P>
     * @param impl an instance of a <B>SocketImpl</B>
     * the subclass wishes to use on the Socket.
     *
     * @exception SocketException if there is an error in the underlying protocol,
     * such as a TCP error.
     * @since   JDK1.1
     */
    protected Socket(SocketImpl impl) throws SocketException {
        this.impl = impl;
        if (impl != null) {
            checkOldImpl();
            this.impl.setSocket(this);
        }
    }

    /**
     * Creates a stream socket and connects it to the specified port
     * number on the named host.
     * <p>
     * If the specified host is <tt>null</tt> it is the equivalent of
     * specifying the address as <tt>{@link java.net.InetAddress#getByName
InetAddress.getByName}(null)</tt>.
     * In other words, it is equivalent to specifying an address of the
     * loopback interface. </p>
     * <p>
     * If the application has specified a server socket factory, that
     * factory's <code>createSocketImpl</code> method is called to create
     * the actual socket implementation. Otherwise a "plain" socket is created.
     * <p>
     * If there is a security manager, its
     * <code>checkConnect</code> method is called
     * with the host address and <code>port</code>
     * as its arguments. This could result in a SecurityException.
     *
     * @param      host   the host name, or <code>null</code> for the loopback address.
     * @param      port   the port number.
     *
     * @exception  UnknownHostException if the IP address of
     * the host could not be determined.
     *
     * @exception  IOException  if an I/O error occurs when creating the socket.
     * @exception  SecurityException  if a security manager exists and its
     *             <code>checkConnect</code> method doesn't allow the operation.
     * @see        java.net.Socket#setSocketImplFactory(java.net.SocketImplFactory)
     * @see        java.net.SocketImpl
     * @see        java.net.SocketImplFactory#createSocketImpl()
     * @see        SecurityManager#checkConnect
     */
    public Socket(String host, int port)
        throws UnknownHostException, IOException
    {
        this(host != null ? new InetSocketAddress(host, port) :
             new InetSocketAddress(InetAddress.getByName(null), port),
             new InetSocketAddress(0), true);
    }

    /**
```

```
     * number at the specified IP address.
     * <p>
     * If the application has specified a socket factory, that factory's
     * <code>createSocketImpl</code> method is called to create the
     * actual socket implementation. Otherwise a "plain" socket is created.
     * <p>
     * If there is a security manager, its
     * <code>checkConnect</code> method is called
     * with the host address and <code>port</code>
     * as its arguments. This could result in a SecurityException.
     *
     * @param      address   the IP address.
     * @param      port      the port number.
     * @exception  IOException  if an I/O error occurs when creating the socket.
     * @exception  SecurityException  if a security manager exists and its
     *             <code>checkConnect</code> method doesn't allow the operation.
     * @see        java.net.Socket#setSocketImplFactory(java.net.SocketImplFactory)
     * @see        java.net.SocketImpl
     * @see        java.net.SocketImplFactory#createSocketImpl()
     * @see        SecurityManager#checkConnect
     */
    public Socket(InetAddress address, int port) throws IOException {
        this(address != null ? new InetSocketAddress(address, port) : null,
             new InetSocketAddress(0), true);
    }

    /**
     * Creates a socket and connects it to the specified remote host on
     * the specified remote port. The Socket will also bind() to the local
     * address and port supplied.
     * <p>
     * If the specified host is <tt>null</tt> it is the equivalent of
     * specifying the address as <tt>{@link java.net.InetAddress#getByName
InetAddress.getByName}(null)</tt>.
     * In other words, it is equivalent to specifying an address of the
     * loopback interface. </p>
     * <p>
     * If there is a security manager, its
     * <code>checkConnect</code> method is called
     * with the host address and <code>port</code>
     * as its arguments. This could result in a SecurityException.
     *
     * @param host the name of the remote host, or <code>null</code> for the loopback
address.
     * @param port the remote port
     * @param localAddr the local address the socket is bound to
     * @param localPort the local port the socket is bound to
     * @exception  IOException  if an I/O error occurs when creating the socket.
     * @exception  SecurityException  if a security manager exists and its
     *             <code>checkConnect</code> method doesn't allow the operation.
     * @see        SecurityManager#checkConnect
     * @since   JDK1.1
     */
    public Socket(String host, int port, InetAddress localAddr,
                  int localPort) throws IOException {
        this(host != null ? new InetSocketAddress(host, port) :
               new InetSocketAddress(InetAddress.getByName(null), port),
             new InetSocketAddress(localAddr, localPort), true);
    }

    /**
```

```
     * the specified remote port. The Socket will also bind() to the local
     * address and port supplied.
     * <p>
     * If there is a security manager, its
     * <code>checkConnect</code> method is called
     * with the host address and <code>port</code>
     * as its arguments. This could result in a SecurityException.
     *
     * @param address the remote address
     * @param port the remote port
     * @param localAddr the local address the socket is bound to
     * @param localPort the local port the socket is bound to
     * @exception  IOException  if an I/O error occurs when creating the socket.
     * @exception  SecurityException  if a security manager exists and its
     *             <code>checkConnect</code> method doesn't allow the operation.
     * @see        SecurityManager#checkConnect
     * @since   JDK1.1
     */
    public Socket(InetAddress address, int port, InetAddress localAddr,
                  int localPort) throws IOException {
        this(address != null ? new InetSocketAddress(address, port) : null,
             new InetSocketAddress(localAddr, localPort), true);
    }

    /**
     * Creates a stream socket and connects it to the specified port
     * number on the named host.
     * <p>
     * If the specified host is <tt>null</tt> it is the equivalent of
     * specifying the address as <tt>{@link java.net.InetAddress#getByName
InetAddress.getByName}(null)</tt>.
     * In other words, it is equivalent to specifying an address of the
     * loopback interface. </p>
     * <p>
     * If the stream argument is <code>true</code>, this creates a
     * stream socket. If the stream argument is <code>false</code>, it
     * creates a datagram socket.
     * <p>
     * If the application has specified a server socket factory, that
     * factory's <code>createSocketImpl</code> method is called to create
     * the actual socket implementation. Otherwise a "plain" socket is created.
     * <p>
     * If there is a security manager, its
     * <code>checkConnect</code> method is called
     * with the host address and <code>port</code>
     * as its arguments. This could result in a SecurityException.
     * <p>
     * If a UDP socket is used, TCP/IP related socket options will not apply.
     *
     * @param      host     the host name, or <code>null</code> for the loopback address.
     * @param      port     the port number.
     * @param      stream   a <code>boolean</code> indicating whether this is
     *                      a stream socket or a datagram socket.
     * @exception  IOException  if an I/O error occurs when creating the socket.
     * @exception  SecurityException  if a security manager exists and its
     *             <code>checkConnect</code> method doesn't allow the operation.
     * @see        java.net.Socket#setSocketImplFactory(java.net.SocketImplFactory)
     * @see        java.net.SocketImpl
     * @see        java.net.SocketImplFactory#createSocketImpl()
     * @see        SecurityManager#checkConnect
     * @deprecated Use DatagramSocket instead for UDP transport.
```

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.