Module java.base
Package java.net

Class Socket

java.lang.Object
    java.net.Socket

All Implemented Interfaces:

Closeable, AutoCloseable

Direct Known Subclasses:

SSLSocket

```
public class Socket
extends Object
implements Closeable
```

This class implements client sockets (also called just "sockets"). A socket is an endpoint for communication between two machines.

The actual work of the socket is performed by an instance of the `SocketImpl` class.

The `Socket` class defines convenience methods to set and get several socket options. This class also defines the `setOption` and `getOption` methods to set and query socket options. A `Socket` support the following options:

Option Name	Description
SO_SNDBUF	The size of the socket send buffer
SO_RCVBUF	The size of the socket receive buffer
SO_KEEPALIVE	Keep connection alive
SO_REUSEADDR	Re-use address
SO_LINGER	Linger on close if data is present (when configured in blocking mode only)
TCP_NODELAY	Disable the Nagle algorithm

Additional (implementation specific) options may also be supported.

Since:

1.0

See Also:

SocketImpl, SocketChannel

Constructor Summary

Constructors

Modifier	Constructor	Description
	`Socket()`	Creates an unconnected Socket.
	`Socket(String host, int port)`	Creates a stream socket and connects it to the specified port number on the named host.
	`Socket(String host, int port, boolean stream)`	**Deprecated.** Use DatagramSocket instead for UDP transport.
	`Socket(String host, int port, InetAddress localAddr, int localPort)`	Creates a socket and connects it to the specified remote host on the specified remote port.
	`Socket(InetAddress address, int port)`	Creates a stream socket and connects it to the specified port number at the specified IP address.
	`Socket(InetAddress host, int port, boolean stream)`	**Deprecated.** Use DatagramSocket instead for UDP transport.
	`Socket(InetAddress address, int port, InetAddress localAddr, int localPort)`	Creates a socket and connects it to the specified remote address on the specified remote port.
	`Socket(Proxy proxy)`	Creates an unconnected socket, specifying the type of proxy, if any, that should be used regardless of any other settings.
protected	`Socket(SocketImpl impl)`	Creates an unconnected Socket with a user-specified SocketImpl

SocketImpl.

Method Summary

All Methods Static Methods Instance Methods Concrete Methods Deprecated Methods

Modifier and Type	Method	Description
void	**bind**(**SocketAddress** bindpoint)	Binds the socket to a local address.
void	**close**()	Closes this socket.
void	**connect**(**SocketAddress** endpoint)	Connects this socket to the server.
void	**connect**(**SocketAddress** endpoint, int timeout)	Connects this socket to the server with a specified timeout value.
SocketChannel	**getChannel**()	Returns the unique `SocketChannel` object associated with this socket, if any.
InetAddress	**getInetAddress**()	Returns the address to which the socket is connected.
InputStream	**getInputStream**()	Returns an input stream for this socket.
boolean	**getKeepAlive**()	Tests if `SO_KEEPALIVE` is enabled.
InetAddress	**getLocalAddress**()	Gets the local address to which the socket is bound.
int	**getLocalPort**()	Returns the local port number to which this socket is bound.
SocketAddress	**getLocalSocketAddress**()	Returns the address of the endpoint this socket is bound to.
boolean	**getOOBInline**()	Tests if `SO_OOBINLINE` is enabled.
<T> T	**getOption**(**SocketOption**<T> name)	Returns the value of a socket option.

OutputStream	getOutputStream()	Returns an output stream for this socket.
int	getPort()	Returns the remote port number to which this socket is connected.
int	getReceiveBufferSize()	Gets the value of the SO_RCVBUF option for this Socket, that is the buffer size used by the platform for input on this Socket.
SocketAddress	getRemoteSocketAddress()	Returns the address of the endpoint this socket is connected to, or null if it is unconnected.
boolean	getReuseAddress()	Tests if SO_REUSEADDR is enabled.
int	getSendBufferSize()	Get value of the SO_SNDBUF option for this Socket, that is the buffer size used by the platform for output on this Socket.
int	getSoLinger()	Returns setting for SO_LINGER.
int	getSoTimeout()	Returns setting for SO_TIMEOUT. 0 returns implies that the option is disabled (i.e., timeout of infinity).
boolean	getTcpNoDelay()	Tests if TCP_NODELAY is enabled.
int	getTrafficClass()	Gets traffic class or type-of-service in the IP header for packets sent from this Socket
boolean	isBound()	Returns the binding state of the socket.
boolean	isClosed()	Returns the closed state of the socket.
boolean	isConnected()	Returns the connection state of the socket.
boolean	isInputShutdown()	Returns whether the read-half of the socket connection is closed.
boolean	isOutputShutdown()	Returns whether the write-half of the socket connection is closed.

void	**sendUrgentData**(int data)	Send one byte of urgent data on the socket.
void	**setKeepAlive**(boolean on)	Enable/disable SO_KEEPALIVE.
void	**setOOBInline**(boolean on)	Enable/disable SO_OOBINLINE (receipt of TCP urgent data) By default, this option is disabled and TCP urgent data received on a socket is silently discarded.
<T> **Socket**	**setOption**(**SocketOption**<T> name, T value)	Sets the value of a socket option.
void	**setPerformancePreferences**(int connectionTime, int latency, int bandwidth)	Sets performance preferences for this socket.
void	**setReceiveBufferSize**(int size)	Sets the SO_RCVBUF option to the specified value for this Socket.
void	**setReuseAddress**(boolean on)	Enable/disable the SO_REUSEADDR socket option.
void	**setSendBufferSize**(int size)	Sets the SO_SNDBUF option to the specified value for this Socket.
static void	**setSocketImplFactory**(**SocketImplFactory** fac)	**Deprecated.** Use a SocketFactory and subclass Socket directly.
void	**setSoLinger**(boolean on, int linger)	Enable/disable SO_LINGER with the specified linger time in seconds.
void	**setSoTimeout**(int timeout)	Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds.
void	**setTcpNoDelay**(boolean on)	Enable/disable TCP_NODELAY (disable/enable Nagle's algorithm).
void	**setTrafficClass**(int tc)	Sets traffic class or type-of-service octet in the IP header for packets sent from this Socket.
void	**shutdownInput**()	Places the input stream for this socket at "end of stream"

		stream".
void	**shutdownOutput**()	Disables the output stream for this socket.
Set<**SocketOption**<?>>	**supportedOptions**()	Returns a set of the socket options supported by this socket.
String	**toString**()	Converts this socket to a `String`.

Methods declared in class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Details

Socket

```
public Socket()
```

Creates an unconnected Socket.

If the application has specified a client socket implementation factory, that factory's `createSocketImpl` method is called to create the actual socket implementation. Otherwise a system-default socket implementation is created.

Since:

1.1

Socket

```
public Socket(Proxy proxy)
```

Creates an unconnected socket, specifying the type of proxy, if any, that should be used regardless of any other settings.

If there is a security manager, its `checkConnect` method is called with the proxy host address and port number as its arguments. This could result in a SecurityException.

Examples:

- `Socket s = new Socket(Proxy.NO_PROXY);` will create a plain socket ignoring any other proxy configuration.
- `Socket s = new Socket(new Proxy(Proxy.Type.SOCKS, new InetSocketAddress("socks.mydom.com", 1080)));` will create a socket connecting through the specified SOCKS proxy server.

Parameters:

`proxy` - a `Proxy` object specifying what kind of proxying should be used.

Throws:

`IllegalArgumentException` - if the proxy is of an invalid type or `null`.

`SecurityException` - if a security manager is present and permission to connect to the proxy is denied.

Since:

1.5

See Also:

`ProxySelector`, `Proxy`

Socket

```
protected Socket(SocketImpl impl)
          throws SocketException
```

Creates an unconnected Socket with a user-specified SocketImpl.

Parameters:

`impl` - an instance of a **SocketImpl** the subclass wishes to use on the Socket.

Throws:

`SocketException` - if there is an error in the underlying protocol, such as a TCP error.

`SecurityException` - if `impl` is non-null and a security manager is set and its `checkPermission` method doesn't allow `NetPermission("setSocketImpl")`.

Since:

1.1

Socket

```
public Socket(String host,
              int port)
       throws UnknownHostException,
              IOException
```

Creates a stream socket and connects it to the specified port number on the named host.

If the specified host is `null` it is the equivalent of specifying the address as `InetAddress.getByName(null)`. In other words, it is equivalent to specifying an address of the loopback interface.

If the application has specified a client socket implementation factory, that factory's `createSocketImpl` method is called to create the actual socket implementation. Otherwise a system-default socket implementation is created.

If there is a security manager, its `checkConnect` method is called with the host address and `port` as its arguments. This could result in a SecurityException.

Parameters:

`host` - the host name, or `null` for the loopback address.

`port` - the port number.

Throws:

`UnknownHostException` - if the IP address of the host could not be determined.

`IOException` - if an I/O error occurs when creating the socket.

`SecurityException` - if a security manager exists and its `checkConnect` method doesn't allow the operation.

`IllegalArgumentException` - if the port parameter is outside the specified range of valid port values, which is between 0 and 65535, inclusive.

See Also:

`SocketImpl`,
`SecurityManager.checkConnect(java.lang.String, int)`

Socket

```
public Socket(InetAddress address,
              int port)
       throws IOException
```

Creates a stream socket and connects it to the specified port number at the specified IP address.

If the application has specified a client socket implementation factory, that factory's `createSocketImpl` method is called to create the actual socket implementation. Otherwise a system-default socket implementation is created.

If there is a security manager, its `checkConnect` method is called with the host address and `port` as its arguments. This could result in a SecurityException.

Parameters:

`address` - the IP address.

`port` - the port number.

Throws:

`IOException` - if an I/O error occurs when creating the socket.

`SecurityException` - if a security manager exists and its `checkConnect` method doesn't allow the operation.

`IllegalArgumentException` - if the port parameter is outside the specified range of valid port values, which is between 0 and 65535, inclusive.

`NullPointerException` - if `address` is null.

See Also:

`SocketImpl`,
`SecurityManager.checkConnect(java.lang.String, int)`

Socket

```
public Socket(String host,
              int port,
              InetAddress localAddr,
              int localPort)
       throws IOException
```

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.

If the specified host is `null` it is the equivalent of specifying the address as `InetAddress.getByName(null)`. In other words, it is equivalent to specifying an address of the loopback interface.

A local port number of `zero` will let the system pick up a free port in the `bind` operation.

If there is a security manager, its `checkConnect` method is called with the host address and `port` as its arguments. This could result in a SecurityException.

Parameters:

`host` - the name of the remote host, or `null` for the loopback address.

`port` - the remote port

`localAddr` - the local address the socket is bound to, or `null` for the `anyLocal` address.

`localPort` - the local port the socket is bound to, or `zero` for a system selected free port.

Throws:

`IOException` - if an I/O error occurs when creating the socket.

`SecurityException` - if a security manager exists and its `checkConnect` method doesn't allow the connection to the destination, or if its `checkListen` method doesn't allow the bind to the local port.

`IllegalArgumentException` - if the port parameter or localPort parameter is outside the specified range of valid port values, which is between 0 and 65535, inclusive.

Since:

1.1

See Also:

`SecurityManager.checkConnect(java.lang.String, int)`

Socket

```
public Socket(InetAddress address,
              int port,
              InetAddress localAddr,
              int localPort)
       throws IOException
```

Creates a socket and connects it to the specified remote address on the specified remote port. The Socket will also bind() to the local address and port supplied.

If the specified local address is `null` it is the equivalent of specifying the address as the AnyLocal address (see `InetAddress.isAnyLocalAddress()`).

A local port number of `zero` will let the system pick up a free port in the `bind` operation.

If there is a security manager, its `checkConnect` method is called with the host address and `port` as its arguments. This could result in a SecurityException.

Parameters:

`address` - the remote address

`port` - the remote port

`localAddr` - the local address the socket is bound to, or `null` for the `anyLocal` address.

`localPort` - the local port the socket is bound to or `zero` for a system selected free port.

Throws:

`IOException` - if an I/O error occurs when creating the socket.

`SecurityException` - if a security manager exists and its `checkConnect` method doesn't allow the connection to the destination, or if its `checkListen` method doesn't allow the bind to the local port.

`IllegalArgumentException` - if the port parameter or localPort parameter is outside the specified range of valid port values, which is between 0 and 65535, inclusive.

`NullPointerException` - if `address` is null.

Since:

1.1

See Also:

`SecurityManager.checkConnect(java.lang.String, int)`

Socket

```
@Deprecated
public Socket(String host,
              int port,
              boolean stream)
       throws IOException
```

Deprecated.
Use DatagramSocket instead for UDP transport.

Creates a stream socket and connects it to the specified port number on the named host.

If the specified host is `null` it is the equivalent of specifying the address as `InetAddress.getByName(null)`. In other words, it is equivalent to specifying an address of the loopback interface.

If the stream argument is `true`, this creates a stream socket. If the stream argument is `false`, it creates a datagram socket.

If the application has specified a client socket implementation factory, that factory's `createSocketImpl` method is called to create the actual socket implementation. Otherwise a system-default socket implementation is created.

If there is a security manager, its `checkConnect` method is called with the host address and `port` as its arguments. This could result in a SecurityException.

If a UDP socket is used, TCP/IP related socket options will not apply.

Parameters:

`host` - the host name, or `null` for the loopback address.

`port` - the port number.

`stream` - a `boolean` indicating whether this is a stream socket or a datagram socket.

Throws:

`IOException` - if an I/O error occurs when creating the socket.

`SecurityException` - if a security manager exists and its `checkConnect` method doesn't allow the operation.

`IllegalArgumentException` - if the port parameter is outside the specified range of valid port values, which is between 0 and 65535, inclusive.

See Also:

`SocketImpl`,
`SecurityManager.checkConnect(java.lang.String, int)`

Socket

```
@Deprecated
public Socket(InetAddress host,
              int port,
              boolean stream)
       throws IOException
```

Deprecated.
Use DatagramSocket instead for UDP transport.

Creates a socket and connects it to the specified port number at the specified IP address.

If the stream argument is `true`, this creates a stream socket. If the stream argument is `false`, it creates a datagram socket.

If the application has specified a client socket implementation factory, that factory's `createSocketImpl` method is called to create the actual socket implementation. Otherwise a system-default socket implementation is created.

If there is a security manager, its `checkConnect` method is called with `host.getHostAddress()` and `port` as its arguments. This could result in a SecurityException.

If UDP socket is used, TCP/IP related socket options will not apply.

Parameters:

`host` - the IP address.

`port` - the port number.

`stream` - if `true`, create a stream socket; otherwise, create a datagram socket.

Throws:

`IOException` - if an I/O error occurs when creating the socket.

`SecurityException` - if a security manager exists and its `checkConnect` method doesn't allow the operation.

`IllegalArgumentException` - if the port parameter is outside the specified range of valid port values, which is between 0 and 65535, inclusive.

`NullPointerException` - if `host` is null.

See Also:

`SocketImpl`,
`SecurityManager.checkConnect(java.lang.String, int)`

Method Details

connect

```
public void connect(SocketAddress endpoint)
             throws IOException
```

Connects this socket to the server.

Parameters:

`endpoint` - the `SocketAddress`

Throws:

`IOException` - if an error occurs during the connection

`IllegalBlockingModeException` - if this socket has an associated channel, and the channel is in non-blocking mode

`IllegalArgumentException` - if endpoint is null or is a SocketAddress subclass not supported by this socket

Since:

1.4

connect

```
public void connect(SocketAddress endpoint,
                    int timeout)
             throws IOException
```

Connects this socket to the server with a specified timeout value. A timeout of zero is interpreted as an infinite timeout. The connection will then block until established or an error occurs.

Parameters:

`endpoint` - the `SocketAddress`

`timeout` - the timeout value to be used in milliseconds.

Throws:

`IOException` - if an error occurs during the connection

`SocketTimeoutException` - if timeout expires before connecting

`IllegalBlockingModeException` - if this socket has an associated channel, and the channel is in non-blocking mode

`IllegalArgumentException` - if endpoint is null or is a SocketAddress subclass not supported by this socket, or if `timeout` is negative

Since:

1.4

bind

```
public void bind(SocketAddress bindpoint)
          throws IOException
```

Binds the socket to a local address.

If the address is `null`, then the system will pick up an ephemeral port and a valid local address to bind the socket.

Parameters:

`bindpoint` - the `SocketAddress` to bind to

Throws:

`IOException` - if the bind operation fails, or if the socket is already bound.

`IllegalArgumentException` - if bindpoint is a SocketAddress subclass not supported by this socket

`SecurityException` - if a security manager exists and its `checkListen` method doesn't allow the bind to the local port.

Since:

1.4

See Also:

`isBound()`

getInetAddress

```
public InetAddress getInetAddress()
```

Returns the address to which the socket is connected.

If the socket was connected prior to being `closed`, then this method will continue to return the connected address after the socket is closed.

Returns:

the remote IP address to which this socket is connected, or `null` if the socket is not connected.

getLocalAddress

```
public InetAddress getLocalAddress()
```

Gets the local address to which the socket is bound.

If there is a security manager set, its `checkConnect` method is called with the local address and `-1` as its arguments to see if the operation is allowed. If the operation is not allowed, the `loopback` address is returned.

Returns:

the local address to which the socket is bound, the loopback address if denied by the security manager, or the wildcard address if the socket is closed or not bound yet.

Since:

1.1

See Also:

`SecurityManager.checkConnect(java.lang.String, int)`

getPort

```
public int getPort()
```

Returns the remote port number to which this socket is connected.

If the socket was connected prior to being `closed`, then this method will continue to return the connected port number after the socket is closed.

Returns:

the remote port number to which this socket is connected, or 0 if the socket is not connected yet.

getLocalPort

```
public int getLocalPort()
```

Returns the local port number to which this socket is bound.

If the socket was bound prior to being `closed`, then this method will continue to return the local port number after the socket is closed.

Returns:

the local port number to which this socket is bound or -1 if the socket is not bound yet.

getRemoteSocketAddress

```
public SocketAddress getRemoteSocketAddress()
```

Returns the address of the endpoint this socket is connected to, or `null` if it is unconnected.

If the socket was connected prior to being `closed`, then this method will continue to return the connected address after the socket is closed.

Returns:

a `SocketAddress` representing the remote endpoint of this socket, or `null` if it is not connected yet.

Since:

1.4

See Also:

`getInetAddress()`,
`getPort()`,
`connect(SocketAddress, int)`,
`connect(SocketAddress)`

getLocalSocketAddress

```
public SocketAddress getLocalSocketAddress()
```

Returns the address of the endpoint this socket is bound to.

If a socket bound to an endpoint represented by an `InetSocketAddress` is `closed`, then this method will continue to return an `InetSocketAddress` after the socket is closed. In that case the returned `InetSocketAddress`'s address is the `wildcard` address and its port is the local port that it was bound to.

If there is a security manager set, its `checkConnect` method is called with the local address and `-1` as its arguments to see if the operation is allowed. If the operation is not allowed, a `SocketAddress` representing the `loopback` address and the local port to which this socket is bound is returned.

Returns:

a `SocketAddress` representing the local endpoint of this socket, or a `SocketAddress` representing the loopback address if denied by the security manager, or `null` if the socket is not bound yet.

Since:

1.4

See Also:

`getLocalAddress()`,
`getLocalPort()`,
`bind(SocketAddress)`,
`SecurityManager.checkConnect(java.lang.String, int)`

getChannel

```
public SocketChannel getChannel()
```

Returns the unique `SocketChannel` object associated with this socket, if any.

A socket will have a channel if, and only if, the channel itself was created via the `SocketChannel.open` or `ServerSocketChannel.accept` methods.

Returns:

the socket channel associated with this socket, or `null` if this socket was not created for a channel

Since:

1.4

getInputStream

```
public InputStream getInputStream()
                           throws IOException
```

Returns an input stream for this socket.

If this socket has an associated channel then the resulting input stream delegates all of its operations to the channel. If the channel is in non-blocking mode then the input stream's `read` operations will throw an `IllegalBlockingModeException`.

Under abnormal conditions the underlying connection may be broken by the remote host or the network software (for example a connection reset in the case of TCP connections). When a broken connection is detected by the network software the following applies to the returned input stream :-

- The network software may discard bytes that are buffered by the socket. Bytes that aren't discarded by the network software can be read using `read`.
- If there are no bytes buffered on the socket, or all buffered bytes have been consumed by `read`, then all subsequent calls to `read` will throw an `IOException`.
- If there are no bytes buffered on the socket, and the socket has not been closed using `close`, then `available` will return `0`.

Closing the returned `InputStream` will close the associated socket.

Returns:

an input stream for reading bytes from this socket.

Throws:

`IOException` - if an I/O error occurs when creating the input stream, the socket is closed, the socket is not connected, or the socket input has been shutdown using `shutdownInput()`

getOutputStream

```
public OutputStream getOutputStream()
                            throws IOException
```

Returns an output stream for this socket.

If this socket has an associated channel then the resulting output stream delegates all of its operations to the channel. If the channel is in non-blocking mode then the output stream's `write` operations will throw an `IllegalBlockingModeException`.

Closing the returned `OutputStream` will close the associated socket.

Returns:

an output stream for writing bytes to this socket.

Throws:

`IOException` - if an I/O error occurs when creating the output stream or if the socket is not connected.

setTcpNoDelay

```
public void setTcpNoDelay(boolean on)
                   throws SocketException
```

Enable/disable `TCP_NODELAY` (disable/enable Nagle's algorithm).

Parameters:

`on` - `true` to enable TCP_NODELAY, `false` to disable.

Throws:

`SocketException` - if there is an error in the underlying protocol, such as a TCP error.

Since:

1.1

See Also:

`getTcpNoDelay()`

getTcpNoDelay

```
public boolean getTcpNoDelay()
                     throws SocketException
```

Tests if `TCP_NODELAY` is enabled.

Returns:

a `boolean` indicating whether or not `TCP_NODELAY` is enabled.

Throws:

`SocketException` - if there is an error in the underlying protocol, such as a TCP error.

Since:

1.1

See Also:

`setTcpNoDelay(boolean)`

setSoLinger

```
public void setSoLinger(boolean on,
                        int linger)
                 throws SocketException
```

Enable/disable SO_LINGER with the specified linger time in seconds. The maximum timeout value is platform specific. The setting only affects socket close.

Parameters:

on - whether or not to linger on.

linger - how long to linger for, if on is true.

Throws:

SocketException - if there is an error in the underlying protocol, such as a TCP error.

IllegalArgumentException - if the linger value is negative.

Since:

1.1

See Also:

getSoLinger()

getSoLinger

```
public int getSoLinger()
                throws SocketException
```

Returns setting for SO_LINGER. -1 returns implies that the option is disabled. The setting only affects socket close.

Returns:

the setting for SO_LINGER.

Throws:

SocketException - if there is an error in the underlying protocol, such as a TCP error.

Since:

1.1

See Also:

`setSoLinger(boolean, int)`

sendUrgentData

```
public void sendUrgentData(int data)
                    throws IOException
```

Send one byte of urgent data on the socket. The byte to be sent is the lowest eight bits of the data parameter. The urgent byte is sent after any preceding writes to the socket OutputStream and before any future writes to the OutputStream.

Parameters:

`data` - The byte of data to send

Throws:

`IOException` - if there is an error sending the data.

Since:

1.4

setOOBInline

```
public void setOOBInline(boolean on)
                  throws SocketException
```

Enable/disable `SO_OOBINLINE` (receipt of TCP urgent data) By default, this option is disabled and TCP urgent data received on a socket is silently discarded. If the user wishes to receive urgent data, then this option must be enabled. When enabled, urgent data is received inline with normal data.

Note, only limited support is provided for handling incoming urgent data. In particular, no notification of incoming urgent data is provided and there is no capability to distinguish between normal data and urgent data unless provided by a higher level protocol.

Parameters:

`on` - `true` to enable `SO_OOBINLINE`, `false` to disable.

Throws:

SocketException - if there is an error in the underlying protocol, such as a TCP error.

Since:

1.4

See Also:

getOOBInline()

getOOBInline

```
public boolean getOOBInline()
                     throws SocketException
```

Tests if SO_OOBINLINE is enabled.

Returns:

a boolean indicating whether or not SO_OOBINLINE is enabled.

Throws:

SocketException - if there is an error in the underlying protocol, such as a TCP error.

Since:

1.4

See Also:

setOOBInline(boolean)

setSoTimeout

```
public void setSoTimeout(int timeout)
                  throws SocketException
```

Enable/disable SO_TIMEOUT with the specified timeout, in milliseconds. With this option set to a positive timeout value, a read() call on the InputStream associated with this Socket will block for only this amount of time. If the timeout expires, a **java.net.SocketTimeoutException** is raised, though the Socket is still valid. A timeout of zero is interpreted as an infinite timeout. The option **must** be enabled prior to entering the blocking operation to have effect.

Parameters:

`timeout` - the specified timeout, in milliseconds.

Throws:

`SocketException` - if there is an error in the underlying protocol, such as a TCP error

`IllegalArgumentException` - if `timeout` is negative

Since:

1.1

See Also:

`getSoTimeout()`

getSoTimeout

```
public int getSoTimeout()
                 throws SocketException
```

Returns setting for `SO_TIMEOUT`. 0 returns implies that the option is disabled (i.e., timeout of infinity).

Returns:

the setting for `SO_TIMEOUT`

Throws:

`SocketException` - if there is an error in the underlying protocol, such as a TCP error.

Since:

1.1

See Also:

`setSoTimeout(int)`

setSendBufferSize

```
public void setSendBufferSize(int size)
                      throws SocketException
```

Sets the SO_SNDBUF option to the specified value for this Socket. The SO_SNDBUF option is used by the platform's networking code as a hint for the size to set the underlying network I/O buffers.

Because SO_SNDBUF is a hint, applications that want to verify what size the buffers were set to should call getSendBufferSize().

Parameters:

size - the size to which to set the send buffer size. This value must be greater than 0.

Throws:

SocketException - if there is an error in the underlying protocol, such as a TCP error.

IllegalArgumentException - if the value is 0 or is negative.

Since:

1.2

See Also:

getSendBufferSize()

getSendBufferSize

```
public int getSendBufferSize()
                    throws SocketException
```

Get value of the SO_SNDBUF option for this Socket, that is the buffer size used by the platform for output on this Socket.

Returns:

the value of the SO_SNDBUF option for this Socket.

Throws:

SocketException - if there is an error in the underlying protocol, such as a TCP error.

Since:

1.2

See Also:

setSendBufferSize(int)

setReceiveBufferSize

```
public void setReceiveBufferSize(int size)
                          throws SocketException
```

Sets the SO_RCVBUF option to the specified value for this Socket. The SO_RCVBUF option is used by the platform's networking code as a hint for the size to set the underlying network I/O buffers.

Increasing the receive buffer size can increase the performance of network I/O for high-volume connection, while decreasing it can help reduce the backlog of incoming data.

Because SO_RCVBUF is a hint, applications that want to verify what size the buffers were set to should call getReceiveBufferSize().

The value of SO_RCVBUF is also used to set the TCP receive window that is advertised to the remote peer. Generally, the window size can be modified at any time when a socket is connected. However, if a receive window larger than 64K is required then this must be requested **before** the socket is connected to the remote peer. There are two cases to be aware of:

1. For sockets accepted from a ServerSocket, this must be done by calling ServerSocket.setReceiveBufferSize(int) before the ServerSocket is bound to a local address.
2. For client sockets, setReceiveBufferSize() must be called before connecting the socket to its remote peer.

Parameters:

size - the size to which to set the receive buffer size. This value must be greater than 0.

Throws:

IllegalArgumentException - if the value is 0 or is negative.

SocketException - if there is an error in the underlying protocol, such as a TCP error.

Since:

1.2

See Also:

getReceiveBufferSize(),
ServerSocket.setReceiveBufferSize(int)

getReceiveBufferSize

```
public int getReceiveBufferSize()
                          throws SocketException
```

Gets the value of the SO_RCVBUF option for this Socket, that is the buffer size used by the platform for input on this Socket.

Returns:

the value of the SO_RCVBUF option for this Socket.

Throws:

SocketException - if there is an error in the underlying protocol, such as a TCP error.

Since:

1.2

See Also:

setReceiveBufferSize(int)

setKeepAlive

```
public void setKeepAlive(boolean on)
                  throws SocketException
```

Enable/disable SO_KEEPALIVE.

Parameters:

on - whether or not to have socket keep alive turned on.

Throws:

SocketException - if there is an error in the underlying protocol, such as a TCP error.

Since:

1.3

See Also:

getKeepAlive()

getKeepAlive

```
public boolean getKeepAlive()
                     throws SocketException
```

Tests if SO_KEEPALIVE is enabled.

Returns:

a boolean indicating whether or not SO_KEEPALIVE is enabled.

Throws:

SocketException - if there is an error in the underlying protocol, such as a TCP error.

Since:

1.3

See Also:

setKeepAlive(boolean)

setTrafficClass

```
public void setTrafficClass(int tc)
                     throws SocketException
```

Sets traffic class or type-of-service octet in the IP header for packets sent from this Socket. As the underlying network implementation may ignore this value applications should consider it a hint.

The tc **must** be in the range `0 <= tc <= 255` or an IllegalArgumentException will be thrown.

Notes:

For Internet Protocol v4 the value consists of an `integer`, the least significant 8 bits of which represent the value of the TOS octet in IP packets sent by the socket. RFC 1349 defines the TOS values as follows:

- `IPTOS_LOWCOST (0x02)`
- `IPTOS_RELIABILITY (0x04)`
- `IPTOS_THROUGHPUT (0x08)`
- `IPTOS_LOWDELAY (0x10)`

The last low order bit is always ignored as this corresponds to the MBZ (must be zero) bit.

Setting bits in the precedence field may result in a SocketException indicating that the operation is not permitted.

As RFC 1122 section 4.2.4.2 indicates, a compliant TCP implementation should, but is not required to, let application change the TOS field during the lifetime of a connection. So whether the type-of-service field can be changed after the TCP connection has been established depends on the implementation in the underlying platform. Applications should not assume that they can change the TOS field after the connection.

For Internet Protocol v6 `tc` is the value that would be placed into the sin6_flowinfo field of the IP header.

Parameters:

`tc` - an `int` value for the bitset.

Throws:

`SocketException` - if there is an error setting the traffic class or type-of-service

Since:

1.4

See Also:

`getTrafficClass()`, `SocketOptions.IP_TOS`

getTrafficClass

```
public int getTrafficClass()
                    throws SocketException
```

Gets traffic class or type-of-service in the IP header for packets sent from this Socket

As the underlying network implementation may ignore the traffic class or type-of-service set using `setTrafficClass(int)` this method may return a different value than was previously set using the `setTrafficClass(int)` method on this Socket.

Returns:

the traffic class or type-of-service already set

Throws:

`SocketException` - if there is an error obtaining the traffic class or type-of-service value.

Since:

1.4

See Also:

`setTrafficClass(int)`, `SocketOptions.IP_TOS`

setReuseAddress

```
public void setReuseAddress(boolean on)
                     throws SocketException
```

Enable/disable the `SO_REUSEADDR` socket option.

When a TCP connection is closed the connection may remain in a timeout state for a period of time after the connection is closed (typically known as the `TIME_WAIT` state or `2MSL` wait state). For applications using a well known socket address or port it may not be possible to bind a socket to the required `SocketAddress` if there is a connection in the timeout state involving the socket address or port.

Enabling `SO_REUSEADDR` prior to binding the socket using `bind(SocketAddress)` allows the socket to be bound even though a previous connection is in a timeout state.

When a `Socket` is created the initial setting of `SO_REUSEADDR` is disabled.

The behaviour when `SO_REUSEADDR` is enabled or disabled after a socket is bound (See `isBound()`) is not defined.

Parameters:

`on` - whether to enable or disable the socket option

Throws:

`SocketException` - if an error occurs enabling or disabling the `SO_REUSEADDR` socket option, or the socket is closed.

Since:

1.4

See Also:

`getReuseAddress()`, `bind(SocketAddress)`, `isClosed()`, `isBound()`

getReuseAddress

```
public boolean getReuseAddress()
                        throws SocketException
```

Tests if SO_REUSEADDR is enabled.

Returns:

a `boolean` indicating whether or not SO_REUSEADDR is enabled.

Throws:

SocketException - if there is an error in the underlying protocol, such as a TCP error.

Since:

1.4

See Also:

setReuseAddress(boolean)

close

```
public void close()
           throws IOException
```

Closes this socket.

Any thread currently blocked in an I/O operation upon this socket will throw a SocketException.

Once a socket has been closed, it is not available for further networking use (i.e. can't be reconnected or rebound). A new socket needs to be created.

Closing this socket will also close the socket's InputStream and OutputStream.

If this socket has an associated channel then the channel is closed as well.

Specified by:

close in interface AutoCloseable

Specified by:

`close` in interface `Closeable`

Throws:

`IOException` - if an I/O error occurs when closing this socket.

See Also:

`isClosed()`

shutdownInput

```
public void shutdownInput()
                   throws IOException
```

Places the input stream for this socket at "end of stream". Any data sent to the input stream side of the socket is acknowledged and then silently discarded.

If you read from a socket input stream after invoking this method on the socket, the stream's `available` method will return 0, and its `read` methods will return `-1` (end of stream).

Throws:

`IOException` - if an I/O error occurs when shutting down this socket.

Since:

1.3

See Also:

`shutdownOutput()`,
`close()`,
`setSoLinger(boolean, int)`,
`isInputShutdown()`

shutdownOutput

```
public void shutdownOutput()
                    throws IOException
```

Disables the output stream for this socket. For a TCP socket, any previously written data will be sent followed by TCP's normal connection termination sequence. If you write to a socket output stream after invoking shutdownOutput() on the socket, the stream will throw an IOException.

Throws:

`IOException` - if an I/O error occurs when shutting down this socket.

Since:

1.3

See Also:

`shutdownInput()`,
`close()`,
`setSoLinger(boolean, int)`,
`isOutputShutdown()`

toString

```
public String toString()
```

Converts this socket to a `String`.

Overrides:

`toString` in class `Object`

Returns:

a string representation of this socket.

isConnected

```
public boolean isConnected()
```

Returns the connection state of the socket.

Note: Closing a socket doesn't clear its connection state, which means this method will return `true` for a closed socket (see `isClosed()`) if it was successfully connected prior to being closed.

Returns:

true if the socket was successfully connected to a server

Since:

1.4

isBound

```
public boolean isBound()
```

Returns the binding state of the socket.

Note: Closing a socket doesn't clear its binding state, which means this method will return `true` for a closed socket (see `isClosed()`) if it was successfully bound prior to being closed.

Returns:

true if the socket was successfully bound to an address

Since:

1.4

See Also:

`bind(java.net.SocketAddress)`

isClosed

```
public boolean isClosed()
```

Returns the closed state of the socket.

Returns:

true if the socket has been closed

Since:

1.4

See Also:

close()

isInputShutdown

```
public boolean isInputShutdown()
```

Returns whether the read-half of the socket connection is closed.

Returns:

true if the input of the socket has been shutdown

Since:

1.4

See Also:

shutdownInput()

isOutputShutdown

```
public boolean isOutputShutdown()
```

Returns whether the write-half of the socket connection is closed.

Returns:

true if the output of the socket has been shutdown

Since:

1.4

See Also:

shutdownOutput()

setSocketImplFactory

```
@Deprecated(since="17")
public static void setSocketImplFactory(SocketImplFactory fac)
                                 throws IOException
```

Deprecated.
Use a `SocketFactory` and subclass `Socket` directly.
This method provided a way in early JDK releases to replace the system wide implementation of `Socket`. It has been mostly obsolete since Java 1.4. If required, a `Socket` can be created to use a custom implementation by extending `Socket` and using the protected constructor that takes an implementation as a parameter.

Sets the client socket implementation factory for the application. The factory can be specified only once.

When an application creates a new client socket, the socket implementation factory's `createSocketImpl` method is called to create the actual socket implementation.

Passing `null` to the method is a no-op unless the factory was already set.

If there is a security manager, this method first calls the security manager's `checkSetFactory` method to ensure the operation is allowed. This could result in a SecurityException.

Parameters:

`fac` - the desired factory.

Throws:

`IOException` - if an I/O error occurs when setting the socket factory.

`SocketException` - if the factory is already defined.

`SecurityException` - if a security manager exists and its `checkSetFactory` method doesn't allow the operation.

See Also:

`SocketImplFactory.createSocketImpl()`,
`SecurityManager.checkSetFactory()`

setPerformancePreferences

```
public void setPerformancePreferences(int connectionTime,
                                       int latency,
                                       int bandwidth)
```

Sets performance preferences for this socket.

Sockets use the TCP/IP protocol by default. Some implementations may offer alternative protocols which have different performance characteristics than TCP/IP. This method allows the application to express its own preferences as to how these tradeoffs should be made when the implementation chooses from the available protocols.

Performance preferences are described by three integers whose values indicate the relative importance of short connection time, low latency, and high bandwidth. The absolute values of the integers are irrelevant; in order to choose a protocol the values are simply compared, with larger values indicating stronger preferences. Negative values represent a lower priority than positive values. If the application prefers short connection time over both low latency and high bandwidth, for example, then it could invoke this method with the values `(1, 0, 0)`. If the application prefers high bandwidth above low latency, and low latency above short connection time, then it could invoke this method with the values `(0, 1, 2)`.

Invoking this method after this socket has been connected will have no effect.

Parameters:

`connectionTime` - An `int` expressing the relative importance of a short connection time

`latency` - An `int` expressing the relative importance of low latency

`bandwidth` - An `int` expressing the relative importance of high bandwidth

Since:

1.5

setOption

```
public <T> Socket setOption(SocketOption<T> name,
                            T value)
                     throws IOException
```

Sets the value of a socket option.

Type Parameters:

`T` - The type of the socket option value

Parameters:

`name` - The socket option

`value` - The value of the socket option. A value of `null` may be valid for some options.

Returns:

this Socket

Throws:

`UnsupportedOperationException` - if the socket does not support the option.

`IllegalArgumentException` - if the value is not valid for the option.

`IOException` - if an I/O error occurs, or if the socket is closed.

`NullPointerException` - if name is `null`

`SecurityException` - if a security manager is set and if the socket option requires a security permission and if the caller does not have the required permission. `StandardSocketOptions` do not require any security permission.

Since:

9

getOption

```
public <T> T getOption(SocketOption<T> name)
                throws IOException
```

Returns the value of a socket option.

Type Parameters:

`T` - The type of the socket option value

Parameters:

`name` - The socket option

Returns:

The value of the socket option.

Throws:

`UnsupportedOperationException` - if the socket does not support the option.

`IOException` - if an I/O error occurs, or if the socket is closed.

`NullPointerException` - if name is `null`

`SecurityException` - if a security manager is set and if the socket option requires a security permission and if the caller does not have the required permission. `StandardSocketOptions` do not require any security permission.

Since:

9

supportedOptions

```
public Set<SocketOption<?>> supportedOptions()
```

Returns a set of the socket options supported by this socket. This method will continue to return the set of options even after the socket has been closed.

Returns:

A set of the socket options supported by this socket. This set may be empty if the socket's SocketImpl cannot be created.

Since:

9