The Socket API

Introduction

- The **socket API** is an Interprocessing Communication (IPC) programming interface originally provided as part of the Berkeley UNIX operating system.
- It has been ported to all modern operating systems, including Sun Solaris and Windows systems.
- It is a *de facto* **standard** for programming IPC, and is the **basis** of more sophisticated IPC interface such as remote procedure call (RPC) and remote method invocation (RMI).

The socket API

- A socket API provides a programming construct termed a **socket**. A process wishing to communicate with another process must create an instance, or instantiate, such a construct (**socket**)
- The two processes then issue **operations** provided by the API to **send** and **receive** data (e.g., a message)

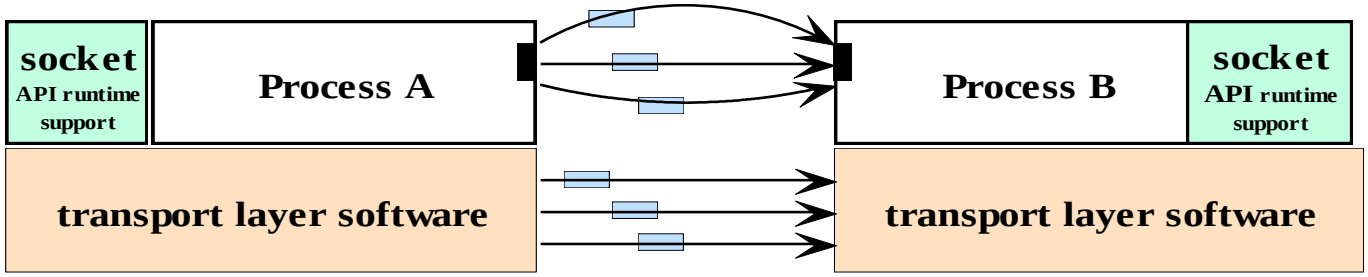
Datagram Socket vs. Stream Socket

- A socket programming construct can make use of either the **UDP** (User Datagram Protocol) or **TCP** (Transmission Control Protocol).
- A **socket** is a generalization of the UNIX file access mechanism that provides an endpoint for communication. A ***datagram*** consists of a datagram header, containing the source and destination IP addresses, and a datagram data area.
- Sockets that use **UDP** for **transport** are known as ***datagram sockets***, while sockets that use **TCP** are termed ***stream sockets***.

UDP vs. TCP

- reliable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
- unreliable, unordered delivery: UDP
  - “best-effort” service
  - loss tolerant; rate sensitive
  - DNS, streaming multimedia apps

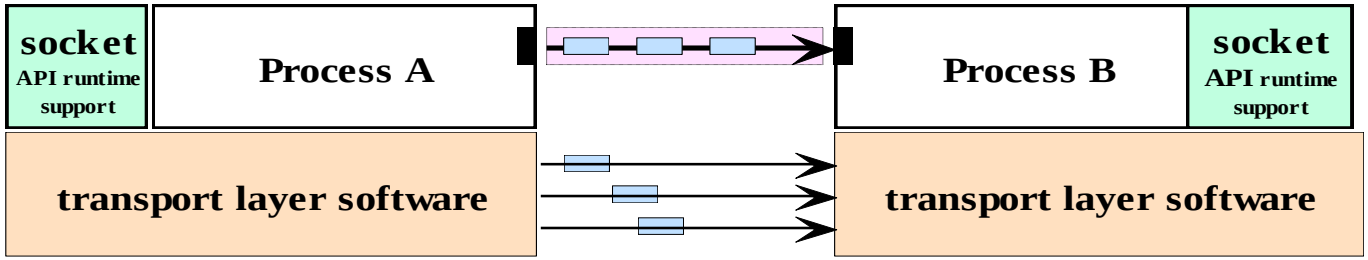
Connection-oriented & connectionless Datagram socket



connectionless datagram socket

a datagram

a logical connection created and maintained by the runtime support of the datagram socket API



connection-oriented datagram socket

The Java Datagram Socket API

- There are two Java classes for the datagram socket API:

  - the ***DatagramSocket*** class for the sockets.

  - the ***DatagramPacket*** class for the datagrams.

- A process wishing to send or receive data using this API must instantiate a
  - DatagramSocket object--a **socket**
  - DatagramPacket object--a **datagram**
- Each **socket** in a receiver process is said to be ***bound*** to a **UDP port** of the machine local to the process.

The Java Datagram Socket API

To **send** a datagram to another process, a process:

- creates a DatagramSocket (**socket**) object, and an object that represents the **datagram** itself. This **datagram object** can be created by instantiating a ***DatagramPacket*** object, which carries a reference to a byte array and the destination address--**host ID** and **port number**, to which the receiver's socket is bound.
- issues a call to the send method in the DatagramSocket object, specifying a reference to the DatagramPacket object as an argument.

The Java Datagram Socket API

- **DatagramSocket mySocket = new DatagramSocket();**
  // any available port number
- **byte[ ] byteMsg = message.getBytes( );**
- **DatagramPacket datagram = new DatagramPacket (byteMsg , byteMsg.length, receiverHost, receiverPort);**
- **mySocket.send(datagram);**
- **mySocket.close( );**

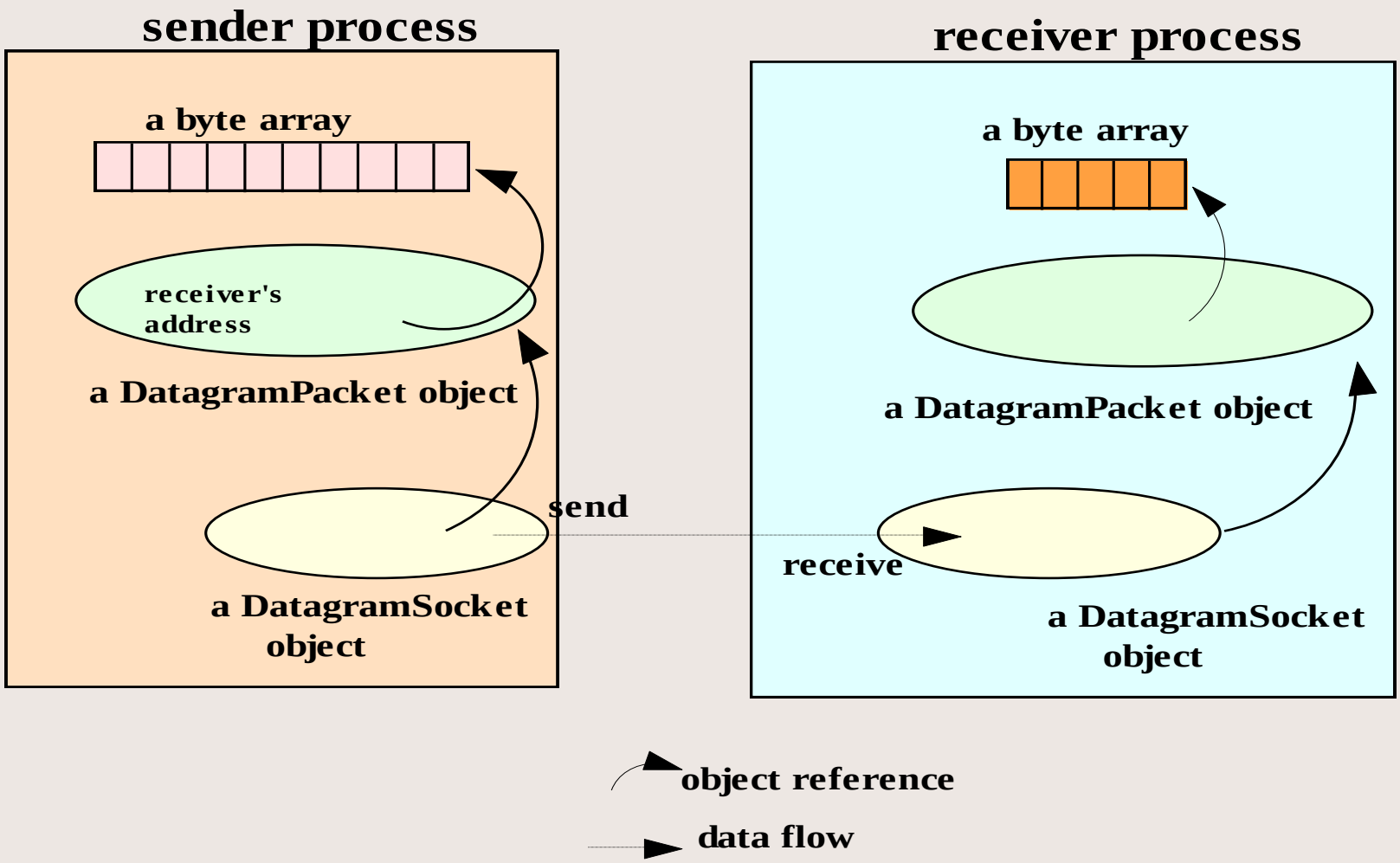
The Java Datagram Socket API

- In the **receiving** process, a ***DatagramSocket*** (socket) object must also be instantiated and bound to a local port, the port number must agree with that specified in the datagram packet of the sender.
- To receive datagrams sent to the socket, the process creates a ***datagramPacket*** object which references a byte array, and calls a ***receive*** method in its ***DatagramSocket*** object, specifying as argument a reference to the ***DatagramPacket*** object.

The Java Datagram Socket API

```
DatagramSocket    mySocket = new DatagramSocket(port);
byte[ ] recMsg = new byte[MAX_LEN];
DatagramPacket datagram =  new DatagramPacket(recMsg,
    MAX_LEN);
mySocket.receive(datagram);  // blocking and waiting
mySocket.close( );
```

The Data Structures in the sender and receiver programs



The program flow in the sender and receiver programs

sender program

create a datagram socket and
bind it to any local port;
place data in a byte array;
create a datagram packet, specifying
the data array and the receiver's
address;
invoke the send method of the
socket with a reference to the
datagram packet;

receiver program

create a datagram socket and
bind it to a specific local port;
create a byte array for receiving the data;
create a datagram packet, specifying
the data array;
invoke the receive method of the
socket with a reference to the
datagram packet;

- **Q**: Why the sender socket needs a local port number?

Setting timeout

To avoid indefinite blocking, a timeout can be set with a socket object:

void **setSoTimeout**(int timeout)

- Set a timeout for the blocking receive from this socket, in milliseconds.
- int timeoutPeriod = 30000; // 30 seconds
  mySocket.setSoTimeout(timeoutPeriod);

Once set, the timeout will be in effect for all blocking operations.

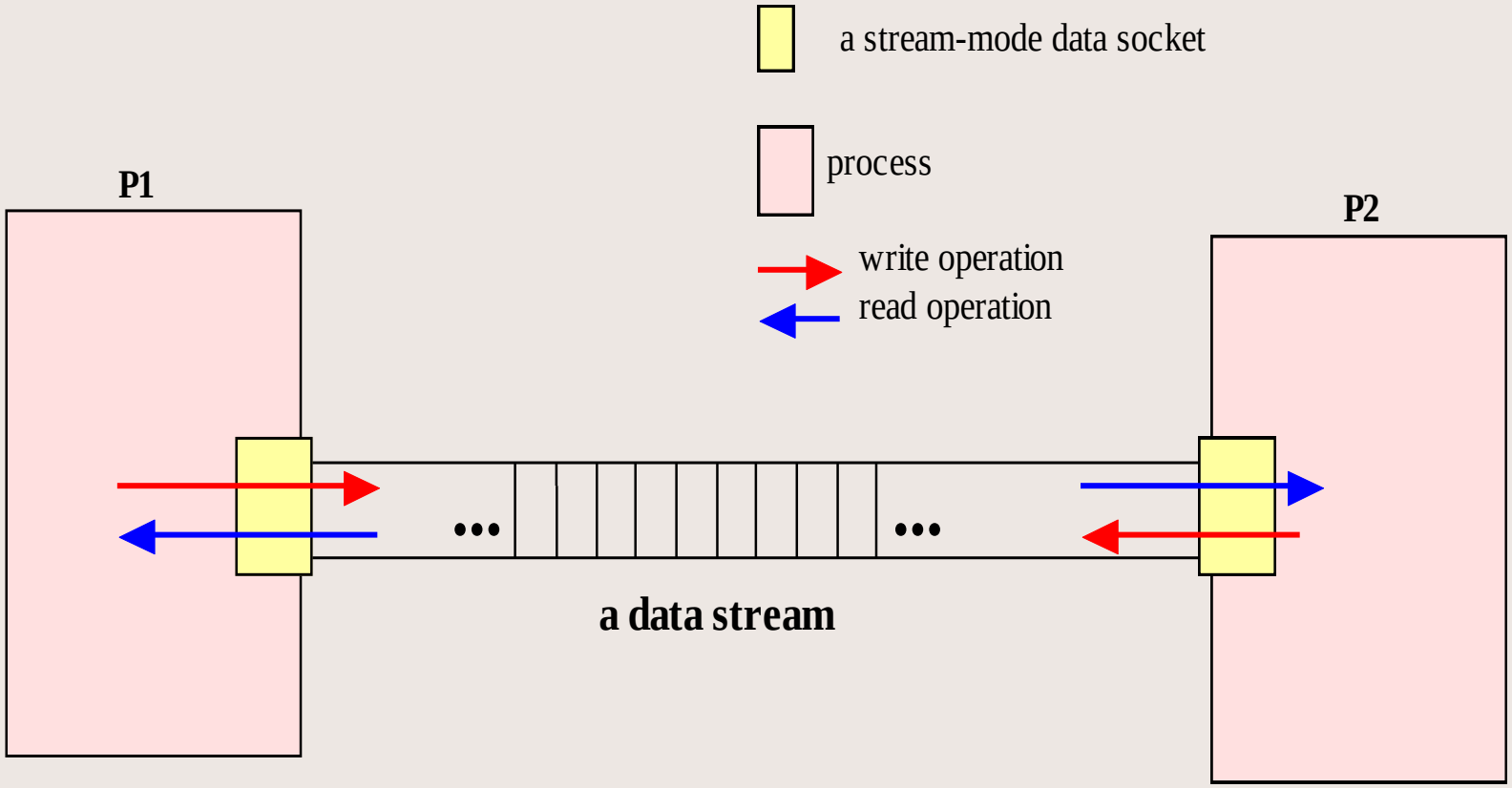
Key Methods and Constructors

Method/Constructor	Description
DatagramPacket (byte[ ] buf, int length)	Construct a datagram packet for receiving packets of length *length;* data received will be stored in the byte array reference by *buf*.
DatagramPacket (byte[ ] buf, int length, InetAddress address, int port)	Construct a datagram packet for sending packets of length *length* to the socket bound to the specified port number on the specified host *;* data received will be stored in the byte array reference by *buf*.
DatagramSocket ( )	Construct a datagram socket and binds it to any available port on the local host machine; this constructor can be used for a process that sends data and does not need to receive data.
DatagramSocket (int port)	Construct a datagram socket and binds it to the specified port on the local host machine; the port number can then be specified in a datagram packet sent by a sender.
void close( )	Close this datagramSocket object
void receive (DatagramPacket p)	Receive a datagram packet using this socket.
void send (DatagramPacket p)	Send a datagram packet using this socket.
void setSoTimeout (int timeout)	Set a timeout for the blocking receive from this socket, in milliseconds.

The Stream-Mode Socket API

- The **datagram socket** API supports the exchange of **discrete** units of data.
- the **stream socket** API provides a model of data transfer based on the stream-mode I/O of the Unix operating systems.
- By definition, a stream-mode socket supports **connection-oriented communication** only.

Stream-Mode Socket API (connection-oriented socket API)



Stream-Mode Socket API

- A stream-mode socket is established for data exchange between two specific processes.
- Data stream is written to the socket at one end, and read from the other end.
- A data stream cannot be used to communicate with more than one process.

Stream-Mode Socket API

In Java, the stream-mode socket API is provided with two classes:

- ServerSocket: for accepting connections; we will call an object of this class a **connection** socket.
- Socket: for data exchange; we will call an object of this class a **data** socket.

Stream-Mode Socket API

- ServerSocket connectionSocket = new ServerSocket(portNo);
- Socket dataSocket = connectionSocket.accept();

// waiting for a connection request

- OutputStream outStream = dataSocket.getOutputStream();
- PrintWriter socketOutput = new PrintWriter(new OutputStreamWriter(outStream));
- socketOutput.println(message);

// send a msg into stream

- socketOutput.flush();
- dataSocket.close( );
- connectionSocket.close( );

- SocketAddress sockAddr = new InetSocketAddress(acceptHost, acceptorPort);
- Socket mySocket = new Socket();
- mySocket.connect (sockAddr, 60000); // 60 sec timeout
- Socket mySocket = new Socket(acceptorHost, acceptorPort);
- InputStream inStream = mySocket.getInputStream();
- BufferedReader socketInput = new BufferedReader(new InputStreamReader(inStream));
- String message = socketInput.readLine( );
- mySocket.close( );

Stream-Mode Socket API program flow

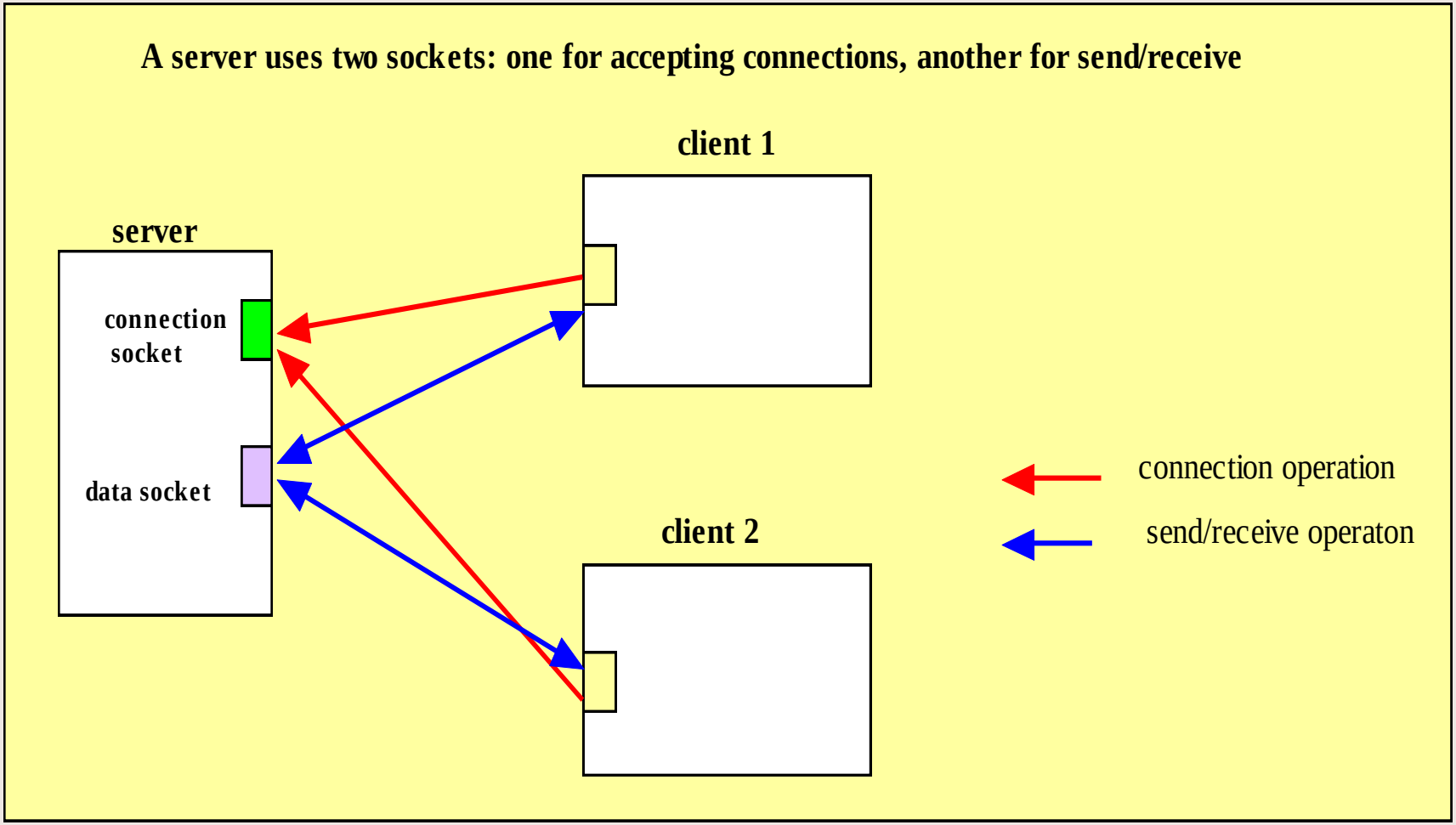
connection listener (server)

- create a connection socket and listen for connection requests;
- accept a connection;
- creates a data socket for reading from or writing to the socket stream;
- get an input stream for reading to the socket;
- read from the stream;
- get an output stream for writing to the socket;
- write to the stream;
- close the data socket;
- close the connection socket.

connection requester (client)

- create a data socket and request for a connection;
- get an output stream for writing to the socket;
- write to the stream;
- get an input stream for reading to the socket;
- read from the stream;
- close the data socket.

The server (the connection listener)



Key methods in the ServerSocket class

Method/constructor	Description
ServerSocket(int port)	Creates a server socket on a specified port.
Socket **accept()** throws IOException	Listens for a connection to be made to this socket and accepts it. The method blocks until a connection is made.
public void close() throws IOException	Closes this socket.
void setSoTimeout(int timeout) throws SocketException	Set a timeout period (in milliseconds) so that a call to accept( ) for this socket will block for only this amount of time. If the timeout expires, a java.io.InterruptedIOException is raised

Note: accept() is a blocking operation.

Key methods in the Socket class

Method/constructor	Description
Socket(InetAddress address, int port)	Creates a stream socket and connects it to the specified port number at the specified IP address
void close() throws IOException	Closes this socket.
InputStream getInputStream( ) throws IOException	Returns an input stream so that data may be read from this socket.
OutputStream getOutputStream( )throws IOException	Returns an output stream so that data may be written to this socket.
void **setSoTimeout**(int timeout) throws SocketException	Set a timeout period for blocking so that a read( ) call on the InputStream associated with this Socket will block for only this amount of time. If the timeout expires, a **java.io.InterruptedIOException** is raised

A **read** operation on an **InputStream** is **blocking**.

A **write** operation on an **OutputStream** is **nonblocking**.

Secure Sockets

- **Secure sockets perform encryption on the data transmitted.**
- **The Java™ Secure Socket Extension (JSSE) is a Java package that enables secure Internet communications.**
- **It implements a Java version of SSL (Secure Sockets Layer) and TLS (Transport Layer Security) protocols**
- **It includes functionalities for data encryption, server authentication, message integrity, and optional client authentication.**
- **Using JSSE, developers can provide for the secure passage of data between a client and a server running any application protocol.**

The Java Secure Socket Extension API

- **Import javax.net.ssl; // provides classes related to creating and configuring secure socket factories.**
- **Class SSLServerSocket is a subclass of ServerSocket, and inherits all its methods.**
- **Class SSLSocket is a subclass of Socket, and inherits all its methods.**
- **There are also classes for**
  - **Certification**
  - **Handshaking**
  - **KeyManager**
  - **SSLsession**