Socket Programming in Java

Web Engineering
Wasim Ahmad Khan

Objectives:

- Learn about the basic java classes (java.net package) that supports Socket programming, namely:
  - InetAddress
  - Socket
  - ServerSocket
  - DatagramSocket
  - DatagramPacket
  - MulticastSocket
- Learn how to use these classes to create Tcp, Udp and Multicast Client-Server applications.

1. The InetAddress class:

The java.net.InetAddress class is used to represent an IP address as an object. It is also used to resolve from host name to IP address (via the DNS) and vice versa.

No Constructors, instead, factory methods are provided.

The following are some of the methods of the **InetAddress** class.

Static InetAddress getByName(String host)	Takes a hostname and returns InetAddress object representing its IP address.
Static InetAddress[] getAllByName(String host)	Takes a hostname and returns an array of InetAddress objects representing its IP addresses.
static InetAddress getLocalHost()	Returns an InetAddress object representing the IP address of the local host
String getHostAddress()	Returns IP address string of this InetAddress object
string getHostName()	Returns the hostname of this InetAddress object.
boolean isLoopbackAddress()	Checks if the InetAddress is a loopback address.
boolean isMulticastAddress()	Checks if the InetAddress is an IP multicast

	address.
String toString()	Converts this InetAddress to a String.

Example 1: IP Address Resolver.

The following example first prints the Address of the current machine and then in a loop reads a host or an address from a user and resolving it.

```
import java.net.*;
import java.io.*;

public class IPAddressResolver {

   public static void main (String args[]) {
      try {
         BufferedReader stdin = new BufferedReader(new
InputStreamReader(System.in));

         InetAddress myself = InetAddress.getLocalHost();
         System.out.println("MyAddress is :"+myself);

         while (true) {
            System.out.print("host name or IP to resolve - <exit> to quit:
");
            String input = stdin.readLine();
            if (input.equalsIgnoreCase("exit"))
               break;

            InetAddress address = InetAddress.getByName(input);
            if (isHostName(input))
               System.out.println("IP Addrsss is: "+
address.getHostAddress());
            else
               System.out.println("Host name is: "+ address.getHostName());

            System.out.println("addresses for "+input+ " are: ");
            InetAddress[] addresses =
InetAddress.getAllByName(address.getHostName());
            for (int i = 0; i < addresses.length; i++)
               System.out.println(addresses[i]);
         }
      }
      catch (UnknownHostException e) {
         System.out.println("Exception: "+e);
      }
      catch (Exception ex) {
         System.out.println("Exception: "+ex);
      }
   }
```

```
   private static boolean isHostName(String input) {
      char[] ca = input.toCharArray();
      for (int i = 0; i < ca.length; i++) {
         if (!Character.isDigit(ca[i]) && ca[i] != '.')
            return true;
      }
      return false;
   }
}
```

2. TCP Sockets (Stream Sockets)

Java provides two classes for creating TCP sockets, namey, Socket and ServerSocket.

The java.net.Socket class:

This class is used by clients to make a connection with a server

Socket constructors are:

Socket(String hostname, int port)
Socket(InetAddress addr, int port)
Socket(String hostname, int port, InetAddress localAddr, int localPort)
Socket(InetAddress addr, int port, InetAddress localAddr, int localPort)

Creating socket

Socket client = new Socket("www.microsoft.com", 80);

Note that the Socket constructor attempts to connect to the remote server - no separate connect() method is provided.

Data is sent and received with output and input streams.

The Socket class has the following methods, that retruns **InputStream** and the **OutputStream** for reading and writing to the socket

public InputStream getInputStream()
public OutputStream getOutputStream()

There's also a method to close a socket:

public synchronized void close()

The following methods are also provided to set socket options:

void setReceiveBufferSize()

void setSendBufferSize()
void setTcpNoDelay()
void setSoTimeout()

The java.net.ServerSocket class

The **ServerSocket** class is used to by server to accept client connections
The constructors for the class are:

public ServerSocket(int port)
public ServerSocket(int port, int backlog)
public ServerSocket(int port, int backlog, InetAddress networkInterface)

Creating a ServerSocket

ServerSocket server = new ServerSocket(80, 50);

Note: a closed ServerSocket cannot be reopened

ServerSocket objects use their accept() method to connect to a client

public Socket accept()

accept() method returns a **Socket** object, and its ***getInputStream()*** and ***getOutputStream()*** methods provide streams for reading and writing to the client.

Note: There are no getInputStream() or getOutputStream() methods for ServerSocket

Example 2: The following examples show how to create TcpEchoServer and the corresponding TcpEchoClient.

```
import java.net.*;
import java.io.*;
import java.util.*;

public class TcpEchoServer
{
   public static void main(String[] args)
   {
      int port = 9090;

      try {
         ServerSocket server = new ServerSocket(port);

         while(true) {
            System.out.println("Waiting for clients on port " + port);
            Socket client = server.accept();
```

```
            System.out.println("Got connection from
"+client.getInetAddress()+":"+client.getPort());

          BufferedReader reader = new BufferedReader(new
InputStreamReader(client.getInputStream()));
          PrintWriter  writer = new PrintWriter(client.getOutputStream());

          writer.println("Welcome to my server");
          writer.flush();

          String  message = reader.readLine();

          while (!(message == null || message.equalsIgnoreCase("exit"))) {
            System.out.println("MessageReceived: "+message);
            writer.println(message);
            writer.flush();

            message = reader.readLine();
          }
          client.close();
        }
      } catch(Exception ex) {
        System.out.println("Connection error: "+ex);
      }
   }
}
```

```
import java.net.*;
import java.io.*;
import java.util.*;

public class TcpEchoClient
{
  public static void main(String[] args) {

     int port = 9090;

     try {
        String host = InetAddress.getLocalHost().getHostName();
          Socket client = new Socket(host, port);

        PrintWriter writer = new PrintWriter(client.getOutputStream());
        BufferedReader reader = new BufferedReader(new
InputStreamReader(client.getInputStream()));

        BufferedReader stdin = new BufferedReader(new
InputStreamReader(System.in));

        System.out.println(reader.readLine()); //read welcome message
```

```
        String message;

        while (true) {
           System.out.print("Enter message to echo or Exit to end : ");
           message = stdin.readLine();

           if (message == null || message.equalsIgnoreCase("exit"))
              break;

           writer.println(message);
           writer.flush();
           System.out.println("Echo from server: "+reader.readLine());
        }
        client.close();

     }catch (Exception ex) {
        System.out.println("Exception: "+ex);
     }
   }
}
```

Example 3: Multi-Client Tcp Servers

The following example shows how to create a multi-client TCP server.

```
import java.net.*;
import java.io.*;
import java.util.*;

public class MultiClientTcpEchoServer
{
   public static void main(String[] args)
   {
      int port = 9090;

      try {
         ServerSocket server = new ServerSocket(port);


         while(true) {
            System.out.println("Waiting for clients on port " + port);
            Socket client = server.accept();
            ConnectionHandler handler = new ConnectionHandler(client);
            handler.start();
         }
      } catch(Exception ex) {
         System.out.println("Connection error: "+ex);
      }
   }
}

class ConnectionHandler extends Thread {
   private Socket client;
```

```
   BufferedReader reader;
   PrintWriter writer;

   static int count;

   public ConnectionHandler(Socket client) {
      this.client = client;
        System.out.println("Got connection from
"+client.getInetAddress()+":"+client.getPort());
      count++;
      System.out.println("Active Connections = " + count);
   }

   public void run() {
      String message=null;

      try {
        reader = new BufferedReader(new
InputStreamReader(client.getInputStream()));
        writer = new PrintWriter(client.getOutputStream());

        writer.println("Welcome to my server");
        writer.flush();

        message = reader.readLine();

        while (!(message == null || message.equalsIgnoreCase("exit"))) {
           writer.println(message);
           writer.flush();

           message = reader.readLine();
        }

         client.close();
        count--;
        System.out.println("Active Connections = " + count);

      } catch (Exception ex) {
        count--;
        System.out.println("Active Connections = " + count);
      }
   }
 }
```

Example 4: Handling Bytes

The following example shows how to deal with non-text data.

```
import java.net.*;
import java.io.*;
import java.util.*;

public class FileServer
```

```
{
   public static void main(String[] args)
   {
      int port = 9070;
      BufferedReader reader;
      PrintWriter writer;
      InputStream inStream;
      OutputStream outStream;

      try {
         ServerSocket server = new ServerSocket(port);

         while(true) {
            System.out.println("Waiting for clients on port " + port);
            Socket client = server.accept();

            inStream = client.getInputStream();
            outStream = client.getOutputStream();
            reader = new BufferedReader(new InputStreamReader(inStream));
            writer = new PrintWriter(outStream);

            writer.println("Welcome to my file server");
            writer.flush();

            String filename = reader.readLine();
            File file = new File(filename);
            if (!file.exists()){
               writer.println("ERROR");
               writer.flush();
            }
            else {
               FileInputStream  fileInStream = new
FileInputStream(filename);
               writer.println(""+file.length());
               writer.flush();
               sendFile(fileInStream, file.length(), outStream);
               fileInStream.close();
            }
            client.close();
         }
      } catch(Exception ex) {
         System.out.println("Connection error: "+ex);
      }
   }

   public static void sendFile(FileInputStream file, long size,
OutputStream outStream) {
      byte[] buffer = new byte[1024];
      int sofar = 0;

      try {
```

```
        while (sofar < size) {
           int read = file.read(buffer, 0, buffer.length);
           sofar+=read;
           outStream.write(buffer, 0, read);
           outStream.flush();
        }
     }
     catch (Exception ex) {
        System.out.println("Exception: "+ex);
     }
  }

 }
```

```
     btnDownload.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent event) {
           boolean downloaded = false;
           try {
              String host = fldHost.getText();
              int port = Integer.parseInt(fldPort.getText());

              Socket client = new Socket(host, port);

              inStream = client.getInputStream();
              outStream = client.getOutputStream();
              writer = new PrintWriter(outStream);
              reader = new BufferedReader(new InputStreamReader(inStream));

              reader.readLine(); //reads welcome message and discard it

              String filename = fldFile.getText();

              writer.println(filename); //sends the file name
              writer.flush();

              String fileSize = reader.readLine(); //reads the file size or
error message

              if (!fileSize.equalsIgnoreCase("ERROR")){
                 receiveFile("CopyOf_"+filename, Long.parseLong(fileSize),
inStream);
                 downloaded = true;
              }
              else
                 downloaded = false;


              if (downloaded)
                 if (filename.toLowerCase().endsWith("txt"))
```

```
                displayFile("CopyOf_"+filename);
             else
                areaResult.setText(filename+ " has been downloaded and
saved as " + "CopyOf_"+filename);
             else
                areaResult.setText("File "+filename + " Could not be
downloaded");

             client.close();

          }catch (Exception ex) {
             System.out.println("Exception: "+ex);
          }
       }
    });


  public void displayFile(String fname){
     try {
        BufferedReader file = new BufferedReader(new FileReader(fname));
        String s;
        areaResult.setText("");
        while ((s=file.readLine()) != null)
           areaResult.append(s+"");
        file.close();
     }
     catch(Exception ex) {
        System.out.println("Exception: "+ex);
     }
  }

  public static void main(String[] args) {
     FileClient fileClient = new FileClient();
     fileClient.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     fileClient.setVisible(true);
  }
```

3. UDP Sockets (Datagram Sockets)

Java provides two classes for creating UDP Sockets:

- **DatagramPacket** class, used to represent the data for sending and receiving.
- **DatagramSocket** class for creating a socket used to send and receive DatagramPackets.

The java.net.DatagramPacket class:

Constructor for receiving:

public DatagramPacket(byte[] data, int length)

Constructor for sending :

public DatagramPacket(byte[] data, int length, InetAddress addr, int port)

Some methods provided by the DatagramPacket class are:

public synchronized void setAddress(InetAddress addr)
public synchronized void setPort(int port)
public synchronized void setData(byte data[])
public synchronized void setLength(int length)

The java.net.DatagramSocket class:

Constructor for sending:

public DatagramSocket()

Constructor for receiving :

public DatagramSocket(int port)
public DatagramSocket(int port, InetAddress addr)

Sending UDP Datagrams involves the following steps:

- Convert the data into byte array.
- Create a **DatagramPacket** using the array
- Create a **DatagramSocket** using the packet and then call ***send()*** method

Receiving UDP Datagrams involves the following steps:

- Construct a **DatagramSocket** object on the **port** on which you want to listen
- Pass an empty **DatagramPacket** object to the DatagramSocket's ***receive()*** method

  public synchronized void receive(DatagramPacket dp)

  - The calling thread blocks until a datagram is received
  - dp is filled with the data from that datagram

Notes:

- After receiving, use the ***getPort()*** and ***getAddress()*** on the received packet to know where the packet came from. Also use ***getData()*** to retrieve the data, and ***getLength()*** to see how many bytes were in the data
- The received packet could be truncated to fit the buffer

Example 5: The following examples show how to create a UdpEchoServer and the corresponding UdpEchoClient.

```
import java.net.*;
import java.io.*;

public class UdpEchoServer
{
   static final int port = 9999;
   static final int packetSize = 1024;


   public static void main(String args[]) throws SocketException{

      DatagramPacket packet;
      DatagramSocket socket;
      byte[] data;
      int clientPort;
      InetAddress address;
      String str;
      int recvSize;

      socket = new DatagramSocket(port);

      while(true){
         data = new byte[packetSize];

         // Create packets to receive the message
         packet = new DatagramPacket(data,packetSize);
         System.out.println("to receive the packets or port: "+port);

         try{
            socket.receive(packet);
         }catch(IOException ie){
            System.out.println(" Could not Receive:"+ie.getMessage());
            System.exit(0);
         }

         // get data about client in order to echo data back
         address = packet.getAddress();
         clientPort = packet.getPort();
         recvSize = packet.getLength();

         str = new String(data,0,recvSize);
         System.out.println("Message  from "+ address+":"+clientPort+":
"+str.trim());

         // echo data back to the client
         packet = new DatagramPacket(data,recvSize,address,clientPort);
         try{
            socket.send(packet);
```

```
        }catch(IOException ex){
           System.out.println("Could not Send "+ex.getMessage());
           System.exit(0);
        }
     }
  }
}
```

```
import java.net.*;
import java.io.*;

public class UdpEchoClient {
   static final int packetSize = 1024;
   static BufferedReader stdin = new BufferedReader(new
InputStreamReader(System.in));

   public static void main(String args[]) throws UnknownHostException,
SocketException{
     DatagramSocket socket;
     DatagramPacket packet;
     InetAddress address;
     String messageSend;
     String messageReturn;
     byte[] data;
     int port;

     try {
        System.out.print("Enter server name: ");
        address = InetAddress.getByName(stdin.readLine());
        System.out.print("Enter server port: ");
        port = Integer.parseInt(stdin.readLine());

        while (true) {
           System.out.print("Enter message for the server or press enter to
exit: ");
           messageSend = stdin.readLine();
           if(messageSend.length() == 0){
              System.exit(0);
           }

           socket = new DatagramSocket();
           data = messageSend.getBytes();
           packet = new DatagramPacket(data,data.length,address,port);
           socket.send(packet);

           //packet is reinitialized to use it for recieving
           data = new byte[packetSize];
           packet = new DatagramPacket(data,data.length);
           socket.receive(packet);
```

```
            messageReturn = new String(data,0,packet.getLength());
            System.out.println("Message Returned : "+ messageReturn);
          }
       }catch(IOException iee){
          System.out.println("Could not receive : "+iee.getMessage() );
          System.exit(0);
       }
    }
}
```

3. Multicast Sockets

Multicasting is achieved in Java using the same **DatagramPacket** class used for normal Datagrams. This is used together with the **MulticastSocket** class.

The **MulticastSocket** class is a subclass of the **DatagramSocket** class, with the following methods for joining and leaving a multicast group added.

void joinGroup(InetAddress mcastaddr)
void leaveGroup(InetAddress mcastaddr)
void setTimeToLive(int ttl)

Example 6: The following examples shows a sample Multicast sender and receiver.

```
import java.io.*;
import java.net.*;

public class MulticastReceiver {

   static MulticastSocket receiver;
   static InetAddress group;

   public static void main(String[] args) {
      try {
       group = InetAddress.getByName("224.100.0.5");
        receiver = new MulticastSocket(9095);
        System.out.println("Joined group at 224.100.0.5");
        receiver.joinGroup(group);

     while (true) {
          byte[] buffer = new byte[1024];
           DatagramPacket recvPacket = new DatagramPacket(buffer,
buffer.length);
           receiver.receive(recvPacket);

         String message = new String(buffer, 0, recvPacket.getLength());
        if (message.length() == 0) {
            receiver.leaveGroup(group);
```

```
            receiver.close();
            break;
         }
            System.out.println(message);
      }
        }
        catch (Exception ex) {
           System.out.println("Exception: "+ ex);
        }
     }
}
```

```
import java.io.*;
import java.net.*;

public class MulticastSender {
   public static void main(String[] args) {
      MulticastSocket sender;
      InetAddress group;
    try {
         group = InetAddress.getByName("224.100.0.5");
         sender = new MulticastSocket(9095);
         sender.setTimeToLive(32);

      BufferedReader stdin = new BufferedReader(new
InputStreamReader(System.in));

     while (true) {
        System.out.print("Enter message for Multicast: ");
           String message = stdin.readLine();
        if (message.length() == 0) {
           sender.leaveGroup(group);
           sender.close();
           break;
        }
         DatagramPacket packet =
           new DatagramPacket(message.getBytes(), message.length(), group,
9095);
          sender.send(packet);
     }
     }
     catch (Exception ex) {
        System.out.println("Exception: "+ ex);
     }
   }
}
```