## Java.net.DatagramPacket class in Java

This class provides facility for connection less transfer of messages from one system to another. Each message is routed only on the basis of information contained within the packet and it may be possible for different packets to route differently. There is also no guarantee as to whether the message will be delivered or not, and they may also arrive out of order. This class provides mechanisms for creation of datagram packets for connectionless delivery using datagram socket class.
**Constructors :** The constructors vary according to their use, i.e. for receiving or for sending the packet. The important parameters used in these constructors are-

1. **buf** : This is the actual message that is to be delivered or received. Datagram packets receive or send data in stuffed in a byte array. If this is used in constructor for sending the message, than this represents the message to be delivered, when used for receiving purpose, it represents the buffer where the received message will be stored.
2. **offset** : It represents the offset into the buffer array.
3. **length :** It is the actual size of packet to receive. This must be less than or equal to the size of buffer array or else there will be overflow as received message wont fit into the array.
4. **InetAddress address** : This is the destination to which the message is to be delivered.
5. **port** : This is the port to which the message will be delivered.
6. **SocketAddress address** : The information about the address and port can be represented wih the help of socket address. Same function as the above two combined.

**For sending purpose**, following constructors are used :

1. **Syntax :**public DatagramPacket(byte[] buf,
2. int offset,
3. int length,
4. InetAddress address,
5. int port)
6. **Parameters :**
7. buf : byte array
8. offset : offset into the array
9. length : length of message to deliver
10. address : address of destination
11. port : port number of destination
12. 
13. **Syntax :**public DatagramPacket(byte[] buf,

```
            int offset,
            int length,
            SocketAddress address)
Parameters :
buf : byte array
offset : offset into the array
length : length of message to deliver
address : socket address of destination

Syntax :public DatagramPacket(byte[] buf,
            int length,
            InetAddress address,
            int port)
Parameters :
buf : byte array
length : length of message to deliver
address : address of destination
port : port number of destination

Syntax :public DatagramPacket(byte[] buf,
            int length,
            SocketAddress address)
Parameters :
buf : byte array
length : length of message to deliver
address : socket address of destination
```

**For receiving purpose**, following constructors are used :

```
Syntax :public DatagramPacket(byte[] buf,
            int offset,
            int length)
Parameters :
buf : byte array
offset : offset into the array
length : length of message to deliver

Syntax :public DatagramPacket(byte[] buf,
            int length)
Parameters :
buf : byte array
length : length of message to deliver
```

**Methods :**

1. **getAddress() :** Returns the IP address to which this packet is sent to or from which it was received.
   **Syntax :**public InetAddress getAddress()
2. **getPort() :** Returns the port to which this packet is sent to or from which it was received. This method is specifically used on the server from getting the port of the client who sent the request.
   **Syntax :** public int getPort()
3. **getData() :** Returns the data contained in this packet as a byte array. The data starts from the offset specified and is of length specified.
   **Syntax :** public byte[] getData()
4. **getOffset() :** Returns the offset specified.
   **Syntax :** public int getOffset()
5. **getLength() :** Returns the length of the data to send or receive
   **Syntax :** public int getLength()
6. **setData() :** Used to set the data of this packet.
7. **Syntax :** public void setData(byte[] buf,
8. int offset,
9. int length)
10. **Parameters :**
11. buf : data buffer
12. offset :offset into the data
13. length : length of the data

Another overloaded method in which offset is set to 0 and lenght is set to length of the buffer.

1. **Syntax :** public void setData(byte[] buf)
2. **Parameters :**
3. buf : data buffer
4. **setAddress() :** Used to set the address to which this packet is sent.
5. **Syntax :** public void setAddress(InetAddress iaddr)
6. **Parameters :**
7. iaddr : InetAddress of the recipient
8. **setPort() :** Set the port on which destination will receive this packet.
9. **Syntax :**public void setPort(int iport)
10. **Parameters :**
    iport : the port number
11. **setSocketAddress() :** Used to set the socket address of the destination(IP address+ port number).

12. **Syntax :** public void setSocketAddress(SocketAddress address)
13. **Parameters :**
    address : socket address
14. **getSocketAddress() :** Returns the socket address of this packet. In case it was received, return the socket address of the host machine.
    **Syntax :** public SocketAddress getSocketAddress()
15. **setLength() :** Used to set the length for this packet.
16. **Syntax :**public void setLength(int length)
17. **Parameters :**
    length : length of the packet

**Java Implementation :**

```
//Java program to illustrate various
//DatagramPacket class methods
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.util.Arrays;

public class datapacket
{
   public static void main(String[] args) throws IOException
   {

      byte ar[] = { 12, 13, 15, 16 };
      byte buf[] = { 15, 16, 17, 18, 19 };
      InetAddress ip = InetAddress.getByName("localhost");

      // DatagramPacket for sending the data
      DatagramPacket dp1 = new DatagramPacket(ar, 4, ip, 1052);

      // setAddress() method
      // I have used same address as in initiation
      dp1.setAddress(ip);

      // getAddress() method
      System.out.println("Address : " + dp1.getAddress());

      // setPort() method
      dp1.setPort(2525);

      // getPort() method
      System.out.println("Port : " + dp1.getPort());

      // setLength() method
      dp1.setLength(4);
```

```
        // getLength() method
        System.out.println("Length : " + dp1.getLength());

        // setData() method
        dp1.setData(buf);

        // getData() method
        System.out.println("Data : " + Arrays.toString(dp1.getData()));

        // setSocketAddress() method
        //dp1.setSocketAddress(address.getLocalSocketAddress());

        // getSocketAddress() method
        System.out.println("Socket Address : " + dp1.getSocketAddress());

        // getOffset() method
        System.out.println("Offset : " + dp1.getOffset());

    }
}
```

**Output :**

Address : localhost/127.0.0.1

Port : 2525

Length : 4

Data : [15, 16, 17, 18, 19]

Socket Address : localhost/127.0.0.1:2525

Offset : 0

**Client Side Implementation**

filter_none
edit
play_arrow
brightness_4

```
// Java program to illustrate Client side
// Implementation using DatagramSocket
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

public class udpBaseClient_2
{
    public static void main(String args[]) throws IOException
    {
        Scanner sc = new Scanner(System.in);
```

```
        // Step 1:Create the socket object for
        // carrying the data.
        DatagramSocket ds = new DatagramSocket();

        InetAddress ip = InetAddress.getLocalHost();
        byte buf[] = null;

        // loop while user not enters "bye"
        while (true)
        {
            String inp = sc.nextLine();

            // convert the String input into the byte array.
            buf = inp.getBytes();

            // Step 2 : Create the datagramPacket for sending
            // the data.
            DatagramPacket DpSend =
                  new DatagramPacket(buf, buf.length, ip, 1234);

            // Step 3 : invoke the send call to actually send
            // the data.
            ds.send(DpSend);

            // break the loop if user enters "bye"
            if (inp.equals("bye"))
                break;
        }
    }
}
```

**Output:**

```
Hello
I am Client.
...
bye
```

**Server side Implementation**

```
filter_none
edit
play_arrow
brightness_4
// Java program to illustrate Server side
// Implementation using DatagramSocket
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;

public class udpBaseServer_2
{
    public static void main(String[] args) throws IOException
```

```
    {
        // Step 1 : Create a socket to listen at port 1234
        DatagramSocket ds = new DatagramSocket(1234);
        byte[] receive = new byte[65535];

        DatagramPacket DpReceive = null;
        while (true)
        {

            // Step 2 : create a DatgramPacket to receive the data.
            DpReceive = new DatagramPacket(receive, receive.length);

            // Step 3 : revieve the data in byte buffer.
            ds.receive(DpReceive);

            System.out.println("Client:-" + data(receive));

            // Exit the server if the client sends "bye"
            if (data(receive).toString().equals("bye"))
            {
                System.out.println("Client sent bye.....EXITING");
                break;
            }

            // Clear the buffer after every message.
            receive = new byte[65535];
        }
    }

    // A utility method to convert the byte array
    // data into a string representation.
    public static StringBuilder data(byte[] a)
    {
        if (a == null)
            return null;
        StringBuilder ret = new StringBuilder();
        int i = 0;
        while (a[i] != 0)
        {
            ret.append((char) a[i]);
            i++;
        }
        return ret;
    }
}
```

In a nutshell, we can summarize the steps of sending and recieving data over UDP as follows:-

1. For sending a packet via UDP, we should know 4 things, **the message to send, its length, ipaddress of destination, port at which destination is listening.**

2. Once we know all these things, we can create the socket object for carrying the packets and packets which actually possess the data.
3. Invoke send()/receive() call for actually sending/recieving packets.
4. Extract the data from the received packet.

**Output:**

```
Client:- Hello
Client:- I am client.
...
Client:- bye
Client sent bye.....EXITING
```

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

/**
 * This program demonstrates a simple TCP/IP socket server.
 *
 * @author www.codejava.net
 */
public class TimeServer {

    public static void main(String[] args) {
        if (args.length < 1) return;

        int port = Integer.parseInt(args[0]);

        try (ServerSocket serverSocket = new ServerSocket(port)) {

            System.out.println("Server is listening on port " + port);

            while (true) {
                Socket socket = serverSocket.accept();

                System.out.println("New client connected");

                OutputStream output = socket.getOutputStream();
                PrintWriter writer = new PrintWriter(output, true);

                writer.println(new Date().toString());
            }

        } catch (IOException ex) {
            System.out.println("Server exception: " + ex.getMessage());
            ex.printStackTrace();
        }
    }
}
import java.net.*;
```

```
import java.io.*;

/**
 * This program demonstrates a simple TCP/IP socket client.
 *
 * @author www.codejava.net
 */
public class TimeClient {

    public static void main(String[] args) {
        if (args.length < 2) return;

        String hostname = args[0];
        int port = Integer.parseInt(args[1]);

        try (Socket socket = new Socket(hostname, port)) {

            InputStream input = socket.getInputStream();
            BufferedReader reader = new BufferedReader(new
InputStreamReader(input));

            String time = reader.readLine();

            System.out.println(time);


        } catch (UnknownHostException ex) {

            System.out.println("Server not found: " + ex.getMessage());

        } catch (IOException ex) {

            System.out.println("I/O error: " + ex.getMessage());
        }
    }
}
```

```
import java.io.*;

02    import java.net.*;

03

04    class TCPServer

05    {
```

```
06    public static void main(String argv[]) throws Exception
07    {
08    String clientSentence;
09    String capitalizedSentence;
10    ServerSocket welcomeSocket= new ServerSocket(6789);
11    while (true)
12    {
13    Socket connectionSocket=welcomeSocket.accept();
14    BufferedReader inFromClient=
15          new BufferedReader(new InputStreamReader(
16          connectionSocket.getInputStream()));
17    DataOutputStream outToClient=
18          new DataOutputStream(
19          connectionSocket.getOutputStream());
20    clientSentence=inFromClient.readLine();
21    System.out.println("From Client: "+clientSentence);
22    capitalizedSentence=
23          clientSentence.toUpperCase()+'\n';
24
```

```
25      outToClient.writeBytes(capitalizedSentence);
26
27      }
28
29      }
30
31      }
```

```
import java.io.*;
06
07 import java.net.*;
08 class TCPClient
09 {
10      public static void main(String argv[]) throws Exception
11      {
12          String sentence;
13          String modifiedSentence;
14
15          BufferedReader inFromUser=
16              new BufferedReader(new InputStreamReader(System.in));
```

```
17
18      InetAddress inetAddress=InetAddress.getLocalHost();
19      //.getByName(String hostname); "CL11"
20      System.out.println(inetAddress);
21
22      Socket clientSocket = new Socket(inetAddress,6789);
23      DataOutputStream outToServer=
24          new  DataOutputStream(clientSocket.getOutputStream());
25
26      BufferedReader inFromServer=
27          new  BufferedReader(new InputStreamReader
28                          (clientSocket.getInputStream()));
29
30
31      sentence=inFromUser.readLine();
32      outToServer.writeBytes(sentence+'\n');
33
34      modifiedSentence=inFromServer.readLine();
35      System.out.println("From Server: "+modifiedSentence );
36      clientSocket.close();
```

```
37

38

39    }

40 }
```

The following GreetingServer program is an example of a server application that uses the Socket class to listen for clients on a port number specified by a command-line argument

```
// File Name GreetingServer.java
import java.net.*;
import java.io.*;


public class GreetingServer extends Thread {
  private ServerSocket serverSocket;

  public GreetingServer(int port) throws IOException {
     serverSocket = new ServerSocket(port);
     serverSocket.setSoTimeout(10000);
  }


  public void run() {
     while(true) {
        try {
           System.out.println("Waiting for client on port " +
              serverSocket.getLocalPort() + "...");
           Socket server = serverSocket.accept();

           System.out.println("Just connected to " +
server.getRemoteSocketAddress());
           DataInputStream in = new
DataInputStream(server.getInputStream());

           System.out.println(in.readUTF());
           DataOutputStream out = new
DataOutputStream(server.getOutputStream());
           out.writeUTF("Thank you for connecting to " +
server.getLocalSocketAddress()
              + "\nGoodbye!");
           server.close();

        } catch (SocketTimeoutException s) {
```

```
            System.out.println("Socket timed out!");
            break;
         } catch (IOException e) {
            e.printStackTrace();
            break;
         }
      }
   }

   public static void main(String [] args) {
      int port = Integer.parseInt(args[0]);
      try {
         Thread t = new GreetingServer(port);
         t.start();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

```
java GreetingServer 6066
Waiting for client on port 6066...
```

Check the client program as follows −

Output

```
$ java GreetingClient localhost 6066
Connecting to localhost on port 6066
Just connected to localhost/127.0.0.1:6066
Server says Thank you for connecting to /127.0.0.1:6066
Goodbye!
```

The following GreetingClient is a client program that connects to a server by using a socket and sends a greeting, and then waits for a response.

Example

```
// File Name GreetingClient.java
import java.net.*;
import java.io.*;


public class GreetingClient {


   public static void main(String [] args) {
      String serverName = args[0];
      int port = Integer.parseInt(args[1]);
      try {
         System.out.println("Connecting to " + serverName + " on port " +
port);
```

```
        Socket client = new Socket(serverName, port);

        System.out.println("Just connected to " +
client.getRemoteSocketAddress());
        OutputStream outToServer = client.getOutputStream();
        DataOutputStream out = new DataOutputStream(outToServer);

        out.writeUTF("Hello from " + client.getLocalSocketAddress());
        InputStream inFromServer = client.getInputStream();
        DataInputStream in = new DataInputStream(inFromServer);

        System.out.println("Server says " + in.readUTF());
        client.close();
      } catch (IOException e) {
        e.printStackTrace();
      }
   }
}
```