1.SOCKETS ORIENTATS A CONNEXIO

Per poder treballar amb Sockets bàsicament necessitarem les següents classes:

java.net.Socket
java.net.ServerSocket
java.io.DataOutputStream
java.io.DataInputStream

La classe Socket ens permet establir connexions a servidors.

La classe ServerSocket es la que crea un servidor que escolta les diferents peticions de clients. Per cada petició acceptada pel servidor es crearà un objecte Socket que representarà l'altre extrem del Socket del client.

Les classes DataOutputStream i DataInputStream son classes que hereten de OutputStream i InputStream respectivament, i que donen més comoditat a la transferència de dades a traves del stream.

Socket:

Les operacions bàsiques que farem servir son

- **Socket(String host, int port)**
- **InputStream getInputStream()**
- **OutputStream getOutputStream()**

Socket es la constructora, i només tenim que proporcionar-li el host i el port, ella mateixa ja s'ocupa d'establir la connexió i tot.

getInputStream i getOutputStream ens proporcionen un stream de lectura i un altre d'escriptura, per poder comunicar-se amb el servidor. Després aquests streams els transformarem a unes subclasses (DataOutputStream i DataInputStream) per poder treballar millor amb ells.

ServerSocket:

Les operacions bàsiques que farem servir son:

- **ServerSocket(int port, int backlog)**
- **Socket accept()**

La constructora li passem el port que vol escoltar i el numero màxim (backlog) de peticions a la cua d'espera.

Cridant sobre accept, aquest agafa de la cua de peticions i retorna un Socket pq ens puguem comunicar amb el client. Si la cua de peticions esta buida, es queda bloquejat.

DataInputStream:

Les operacions bàsiques que farem servir són:

- **DataInputStream(InputStream in)**
- **boolean readBoolean()**
- **byte readByte()**
- **char readChar()**
- **double readDouble()**
- **float readFloat()**
- **int readInt()**
- **long readLong()**
- **short readShort()**
- **String readUTF()**

Com veiem la constructora necessita que li passem un InputStream, i serà el InputStream que ens retorna el Socket cridant getInputStream().

La resta d'operacions son per llegir diferents tipus de dades; booleans, bytes, Strings, enters... Aquesta es la gran facilitat d'utilitzar els DataInputStream, ja que la classe s'ocupa tota sola d'espavilar-se per poder rebre aquest tipus de dada. En la documentacio de java encara hi ha més operacions per tractar amb diferents tipus de dades, només hem ficat les bàsiques.

DataOutputStream:

Les operacions bàsiques que farem servir són:

- **DataOutputStream(OutputStream out)**
- **void writeBoolean(boolean v)**
- **void writeByte(int v)**
- **void writeChar(int v)**
- **void writeDouble(double v)**
- **void writeFloat(float v)**
- **void writeInt(int v)**
- **void writeLong(long v)**
- **void writeShort(int v)**
- **void writeUTF(String str)**

Com veiem la constructora necessita que li passem un OutputStream, i serà el OutputStream que ens retorna el Socket cridant getOutputStream().

La resta d'operacions son per escriure diferents tipus de dades; booleans, bytes, Strings, enters... Aquesta es la gran facilitat d'utilitzar els DataOutputStream, ja que la classe s'ocupa tota sola d'espavilar-se per poder enviar aquest tipus de dada. En la documentacio de java encara hi ha més operacions per tractar amb diferents tipus de dades, només hem ficat les bàsiques.

1.1. CLIENT:

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

public class ClientSocket {
  private static final int PORT = 2323;
  private static final int BUFFER_SIZE = 256;

  public static void main(String[] args) {
    Socket s = null;
    DataInputStream dis = null;
    DataOutputStream dos = null;

    try {
      s = new Socket("127.0.0.1", PORT);
      dis = new DataInputStream(s.getInputStream());
      dos = new DataOutputStream(s.getOutputStream());

      dos.writeInt(100);
      System.out.println("Rebem msg = " + dis.readUTF());
      s.close();
      dis.close();
      dos.close();
    }
    catch(IOException e) {
      try {
        if (s != null)
          s.close();
        if (dis != null)
          dis.close();
        if (dos != null)
          dos.close();
      }
      catch(IOException e2) {}
      System.exit(-1);
    }
  }
}
```

1.2. SERVIDOR:

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

public class ServerSocketClass {
  private static final int PORT = 2323;
  private static final int BUFFER_SIZE = 256;

  public static void main(String[] args) {
    ServerSocket ss = null;
    try {
      ss = new ServerSocket(PORT, 10);

      while(true) {
        Socket s = null;
        DataInputStream dis = null;
        DataOutputStream dos = null;
        s = ss.accept();
        dis = new DataInputStream(s.getInputStream());
        dos = new DataOutputStream(s.getOutputStream());

        System.out.println("Rebem int = " + dis.readInt());
        dos.writeUTF("Enter rebut");
        s.close();
      }
    }
    catch(IOException e) {
      try {
          ss.close();
          if(s != null) s.close();
          if(dis != null) dis.close();
          if(dos != null) dos.close();
        }
        catch(IOException e2) {}
      System.exit(-1);
    }
  }
}
```

2. SOCKETS DE DATAGRAMES

Per enviar per datagrames, basicament hem de tenir en compte tres classes d'objectes:

java.net.DatagramSocket
java.net.DatagramPacket
java.net.InetAddress

El DatagramPacket es l'objecte on anirà la informació, així com el host i el port on volem enviar-ho.

La InetAddress es el objecte encarregat de gestionar el host.

El DatagramSocket s'encarrega d'enviar i rebre informació a través de l'objecte DatagramPacket.

DatagramPacket:

Les operacions bàsiques que farem servir son:

- **DatagramPacket (byte[] buf, int length)**
- **DatagramPacket (byte[] buf, int length, InetAddress address, int port)**
- **byte [] getData()**
- **int getLength()**
- **int getPort()**
- **InetAddress getAddress()**

Les dos primeres son les constructores; la primera li passem el buffer i el tamany d'aquest, i es utilitzada per la recepció de paquets a través d'un DatagramSocket. La segona a més li indiquem una @ i port, i es utilitzada quan volem enviar informació a traves d'un DatagramSocket.

getData retorna la informació del buffer, getLength retorna la longitud de la informacio enviada o rebuda.

getPort retorna el port al que hem enviat la informació o el port de la maquina remota de la que hem rebut la informació. Això depèn de si s'ha utilitzat el DatagramPacket per rebre o enviar informació.

getAddress retorna o el host al que hem enviat o el host que ens ha enviat la informació.

DatagramSocket:

- **DatagramSocket (int port)**
- **void receive(DatagramPacket p)**
- **void send(DatagramPacket p)**

La primera de totes es la constructora, i només tenim que passar-li per paràmetre el port pel que rebrà informació.

El receive i send, només tenim que passar-li el DatagramPacket, on s'escriurà la informació o es llegirà per poder-la enviar.

InetAddress:

- **static InetAddress getByName(String host)**

De fet aquesta classe no requereix massa explicació, només fer constar que si volem construir un objecte InetAddress a partir d'una @, ho farem amb aquesta operació.

2.1. CLIENT:

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

public class ClientDatagrama {
  private static final int PORT = 2323;
  private static final int BUFFER_SIZE = 256;

  public static void main(String[] args) {
    String s = "Hola que tal?";
    byte [] bBuffer = new byte[BUFFER_SIZE];
    DatagramSocket ds = null;
    DatagramPacket dp = null;
    try {
      ds = new DatagramSocket();
    }
    catch(SocketException e) {
      System.exit( -1);
    }

    try {
      dp = new DatagramPacket(s.getBytes(), s.getBytes().length,
                              InetAddress.getByName("127.0.0.1"),
                              PORT);
    }
    catch(UnknownHostException e) {
      System.exit( -1);
    }

    try {
      ds.send(dp);
      dp = new DatagramPacket(bBuffer, bBuffer.length);
      ds.receive(dp);
    }
    catch(IOException e) {
        System.exit( -1);
    }
  }
}
```

2.2. SERVIDOR:

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

public class ServerDatagrama {
  private static final int PORT = 2323;
  private static final int BUFFER_SIZE = 256;

  public static void main(String[] args) {
    byte[] bBuffer = new byte[BUFFER_SIZE];
    DatagramSocket ds = null;
    try {
      ds = new DatagramSocket(PORT);
    }
    catch (SocketException e) {
      System.exit( -1);
    }
    DatagramPacket dp = new DatagramPacket(bBuffer,
bBuffer.length);

    while (true) {
      try {
        ds.receive(dp);
        bBuffer = new String("Gracies per fer servir el
servidor").getBytes();
        dp = new DatagramPacket(bBuffer, bBuffer.length,
dp.getAddress(), dp.getPort());
        ds.send(dp);
      }
      catch(IOException e) {
        System.exit(-1);      }
    }
  }
}
```