NTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE COMPUTO

Nombre de los alumnos: Sandoval Muñoz Emanuel

Matriculas: 2022670104

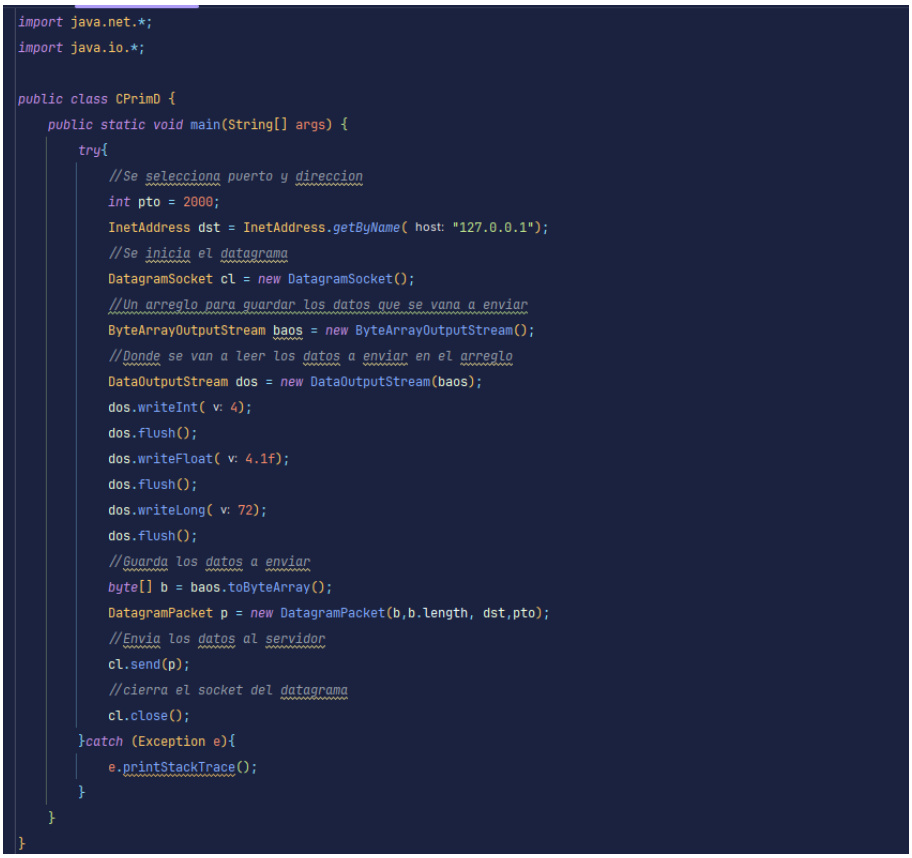
Maestr@: BAUTISTA ROSALES SANDRA IVETTE

Materia: APLICACIONES PARA COMUNICACIONES EN RED

Grupo: 6CM4

Trabajo: Sockets de datagrama (datos primitivos y Datagram Channel) s/a

Código del cliente con primitivas:

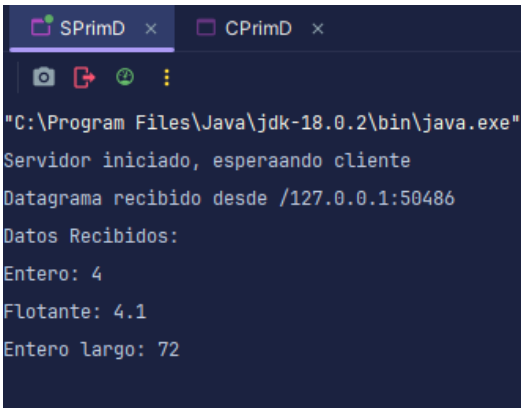


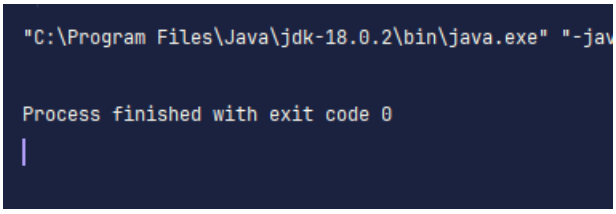
```java
import java.net.*;
import java.io.*;

public class CPrimD {
    public static void main(String[] args) {
        try{
            //Se selecciona puerto y direccion
            int pto = 2000;
            InetAddress dst = InetAddress.getByName( host: "127.0.0.1");
            //Se inicia el datagrama
            DatagramSocket cl = new DatagramSocket();
            //Un arreglo para guardar los datos que se vana a enviar
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            //Donde se van a leer los datos a enviar en el arreglo
            DataOutputStream dos = new DataOutputStream(baos);
            dos.writeInt( v: 4);
            dos.flush();
            dos.writeFloat( v: 4.1f);
            dos.flush();
            dos.writeLong( v: 72);
            dos.flush();
            //Guarda los datos a enviar
            byte[] b = baos.toByteArray();
            DatagramPacket p = new DatagramPacket(b,b.length, dst,pto);
            //Envia los datos al servidor
            cl.send(p);
            //cierra el socket del datagrama
            cl.close();
        }catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

Código del servidor con primitivas:

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

public class SPrimD {
    public static void main(String[] args) {
        try{
            //Se inicia el Socket con el puerto 2000
            DatagramSocket s = new DatagramSocket( port: 2000);
            System.out.println("Servidor iniciado, esperaando cliente");
            for (;;){
                //Se hace el datagrama que va a recibir los datos del cliente
                DatagramPacket p = new DatagramPacket(new byte[2000], length: 2000);
                s.receive(p);
                //Recibe los datos y los muestra en terminal
                System.out.println("Datagrama recibido desde " + p.getAddress() + ":" + p.getPort());
                DataInputStream dis = new DataInputStream(new ByteArrayInputStream(p.getData()));
                int x = dis.readInt();
                float f = dis.readFloat();
                long z = dis.readLong();
                System.out.println("Datos Recibidos \nEntero: "+x+"\nFlotante: "+f+"\nEntero largo: "+z);
            }
        }catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

Ejecución del código cliente al servidor con primitivas:





Código cliente con datagrama Chanel

```java
import java.net.*;
import java.nio.*;
import java.nio.channels.DatagramChannel;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.util.*;

public class UDPCliente {
    public final static int PUERTO = 7;  1 usage
    private final static int LIMITE = 100;  2 usages

    public static void main(String[] args) {
        boolean bandera = false;
        SocketAddress remote = new InetSocketAddress( hostname: "127.0.0.1", PUERTO);
        try {
            DatagramChannel canal = DatagramChannel.open(); // Abre un canal de datagramas
            canal.configureBlocking(false); // Configura el canal para que no sea bloqueante
            canal.connect(remote); // Conecta el canal al servidor remoto
            Selector selector = Selector.open(); // Abre un selector
            canal.register(selector, SelectionKey.OP_WRITE); // Registra el canal en el selector para operaciones de escritura
            ByteBuffer buffer = ByteBuffer.allocateDirect( capacity: 4); // Asigna un buffer de 4 bytes
            int n = 0;
            while (true) {
                selector.select( timeout: 5000); // Espera hasta 5 segundos para que haya canales listos
                Set<SelectionKey> sk = selector.selectedKeys(); // Obtiene las claves de selección
                if (sk.isEmpty() && n == LIMITE || bandera) { // Si no hay claves y se ha alcanzado el límite o la bandera está activa
                    canal.close(); // Cierra el canal
                    break;
                } else {
                    Iterator<SelectionKey> it = sk.iterator();
```

```
                while (it.hasNext()) {
                    SelectionKey key = it.next();
                    it.remove();
                    if (key.isWritable()) { // Si la clave está lista para escritura
                        buffer.clear();
                        buffer.putInt(n);
                        buffer.flip();
                        canal.write(buffer); // Escribe el dato en el canal
                        System.out.println("Escribe el dato: " + n);
                        n++;
                        if (n == LIMITE) { // Si se ha alcanzado el límite
                            buffer.clear();
                            buffer.putInt( value: 1000);
                            buffer.flip();
                            canal.write(buffer); // Escribe el valor de terminación
                            bandera = true;
                            key.interestOps(SelectionKey.OP_READ); // Cambia la clave a operación de lectura
                            break;
                        }
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Código del servidor con datagrama Chanel

```
import java.net.*;
import java.nio.*;
import java.nio.channels.DatagramChannel;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.util.*;

public class UDPServidor {
    public final static int PUERTO = 7;  1 usage
    public final static int TAM_MAXIMO = 65507;  1 usage

    public static void main(String[] args) {
        int port = PUERTO;
        try {
            DatagramChannel canal = DatagramChannel.open(); // Abre un canal de datagramas
            canal.configureBlocking(false); // Configura el canal para que no sea bloqueante
            DatagramSocket socket = canal.socket();
            SocketAddress dir = new InetSocketAddress(port);
            socket.bind(dir); // Asocia el socket a la dirección especificada
            Selector selector = Selector.open(); // Abre un selector
            canal.register(selector, SelectionKey.OP_READ); // Registra el canal en el selector para operaciones de lectura
            ByteBuffer buffer = ByteBuffer.allocateDirect(TAM_MAXIMO); // Asigna un buffer de tamaño máximo
```

```java
        while (true) { // Bucle infinito para mantener el servidor en ejecución
            selector.select( timeout: 5000); // Espera hasta 5 segundos para que haya canales listos
            Set<SelectionKey> sk = selector.selectedKeys(); // Obtiene las claves de selección
            Iterator<SelectionKey> it = sk.iterator();
            while (it.hasNext()) {
                SelectionKey key = it.next();
                it.remove();
                if (key.isReadable()) { // Si la clave está lista para lectura
                    buffer.clear();
                    SocketAddress client = canal.receive(buffer); // Recibe datos del cliente
                    buffer.flip();
                    int eco = buffer.getInt();
                    if (eco == 1000) { // Si el entero leído es 1000
                        canal.close();
                        System.exit( status: 0); // Termina el programa
                    } else {
                        System.out.println("Dato leído: " + eco);
                        buffer.flip();
                        canal.send(buffer, client); // Envía el dato de vuelta al cliente
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Ejecución del código cliente y servidor