3) Seleccione dos objetos involucrados en la situación problemática e implemente la comunicación entre los mismos. Indique si la comunicación es síncrona o asíncrona. Justifique conceptualmente su selección.

Archivo Adjunto: Parlante.zip

```
package estacionamiento;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.URLDecoder;
import java.nio.charset.StandardCharsets;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Parlante {
    private String mensaje;
 // codOP = 1
    public void EstablecerMensaje(String msg) {
        mensaje = msg;
    }
   // codOP = 2
    public void TransmitirMensaje() {
        System.out.println(mensaje);
    }
    public static void main(String[] args) {
        try {
// Se crea un socket de contacto
            Parlante objParlante = new Parlante();
            ServerSocket server = new ServerSocket(4567);
// Espera hasta ser contactado y crea socket de comunicacion con cliente
            while (true) {
                Socket socket = server.accept();
                Conexion cnx = new Conexion(socket, objParlante);
            }
        } catch (IOException ex) {
            Logger.getLogger(Parlante.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }
}
```

```
class Conexion extends Thread {
    Socket sk;
    Parlante p;
    DataInputStream entrada;
    DataOutputStream salida;
    public Conexion(Socket socket, Parlante parlante) {
        try {
            sk = socket;
            p = parlante;
            entrada = new DataInputStream(sk.getInputStream()); //Para
leer mensajes del cliente
//  En este caso, no utilizamos el DataOutputStream  porque no enviamos
nada al cliente
//            salida = new DataOutputStream(sk.getOutputStream());
            start();
        } catch (IOException ex) {
            Logger.getLogger(Conexion.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }
    public void run() {
// Durante la comunicacion
        try {
            boolean fin = false;
            while (true && !fin) {
                short codOp = entrada.readShort(); // Lee la entrada del
cliente
                switch (codOp) {
                    case 1:
                        String mensaje = entrada.readUTF();
                        p.EstablecerMensaje(URLDecoder.decode(mensaje,
StandardCharsets.UTF_8.name()));
                        break;
                    case 2:
                        p.TransmitirMensaje();
                        break;
                    case 100:
                        fin = true;
                        break;
                }
            }
// Cerramos coneccion
            sk.close();
        } catch (IOException ex) {
            Logger.getLogger(Conexion.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }
}
```

Archivo Adjunto: SistemaControl1.zip

```
package estacionamiento;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.*;
import java.nio.charset.StandardCharsets;
import java.util.logging.Level;
import java.util.logging.Logger;
public class SistemaControl1 {
    public SistemaControl1() {
    }
    public void EmitirSaludo() {
        try {
// crea un socket para contactar con el servidor Parlante
            InetAddress direccionServidorParlante =
InetAddress.getByName("localhost");
            Socket conexionParlante = new
Socket(direccionServidorParlante, 4567);
// el cliente solo escribira datos, no espera leer nada del servidor

            DataOutputStream salida = new
DataOutputStream(conexionParlante.getOutputStream());
            salida.writeShort(1); // Envia el codOP = 1 para
EstablecerMensaje()
            String mensaje = URLEncoder.encode("\nBuenos Dias\nPor Favor
Retire El Ticket\n", StandardCharsets.UTF_8.name());
            salida.writeUTF(mensaje); // Envia el argumento del metodo
EstablecerMensaje()
            salida.writeShort(2); // Envia el codOP = 2 para
TransmitirMensaje()
            salida.writeShort(100); // Envia el codOP = 100 para finalizar
conexion
            conexionParlante.close(); // cierra conexion
        } catch (UnknownHostException ex) {

Logger.getLogger(SistemaControl1.class.getName()).log(Level.SEVERE, null,
ex);
        } catch (IOException ex) {

Logger.getLogger(SistemaControl1.class.getName()).log(Level.SEVERE, null,
ex);
        }
    }
    public static void main(String[] args) {
        SistemaControl1 objEstacionamiento = new SistemaControl1();
        objEstacionamiento.EmitirSaludo();
    }
}
```

La comunicación es asincrona, el cliente solo envia mensajes pero no espera respuestas y el servidor simplemente lee los mensajes sin contestar

NOTA:

Primero debe ejecutarce el proyecto **Parlante.zip**, luego **SistemaControl1.zip.**