

# Ejemplo de como podría ser el examen de Dad

Probablemente se nos pida algo del estilo a la implementación de un protocolo, en el que un servidor deberá recibir comandos de un cliente y responder en función a estos.

## Elemento

Este será la clase con el que trabajaremos:

```
public class Elemento{
    //Atributos:
    String nombre
    double cantidad;
    double veces = 0;

    //Constructor:
    public Elemento(String nombre, double cantidad){
        this.nombre = nombre;
        this.cantidad = cantidad;
    }
}
```

## Cliente

Este será el cliente que utilizaremos para enviar señales al serverSocket, este cliente permanecerá activo permitiendo enviar mensajes al servidor hasta que se envíe el comando EXIT y se reciba la señal "cerrado" del servidor.

```
public class Cliente{
    public void ejecutar{
        try{//Se debe hacer siempre try Catch cuando andamos con sockets
            socket = new Socket("localhost", 5000);//Inicializamos nuevo socket
con IP y Puerto
            //Buffers de lectura y escritura:
            BufferedReader br = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter pw = new PrintWriter(new
OutputStreamWriter(socket.getOutputStream()));
            Scanner sc = new Scanner(System.in);//Para leer por teclado
            String rl = ""; //Aqui guardamos cada línea leída
            do{//repetimos contenido hasta que se reciba "Cerrado"
                pw.println(sc.nextLine());//Leemos del teclado con sc y enviamos al
servidor con pw
            }
        }
    }
}
```

```

        pw.flush(); Limpiamos salida
        //Recibimos línea del servidor con br, la almacenamos en rl y la
mostramos en pantalla:
        System.out.println((rl = br.readLine()))
    }while(lineaLeida.contentEquals("Cerrado")==false)//Mientras no se
reciba una señal de cierre
    }catch(IOException e){//en caso de salir excepción
        e.printStackTrace();
    }
}
}
public static void main(String[] args){
    (new Cliente()).ejecutar();
}

```

## Servidor

```

public class Server{
    ServerSocket serverSocket;//Declaramos Socket Servidor
    Socket socket;//declaramos Socket standar
    Static Hashtable<string, ArrayList<elemento>> listaElementos = new
Hashtable<String, ArrayList<elemento>>;//Lista de elementos
    public void ejecutar(){
        try{
            serverSocket = new ServerSocket(5000) //Indicamos puerto de escucha
            while(true){
                socket=serverSocket.accept();//Iniciamos escucha
                (new ServerThread(socket)).start();//Arrancamos el servidor en un
hilo;
            }
        }catch(IOException e){
            e.printStackTrace();//mostramos excepción ocurrida
        }
    }
    public static void main(String[] args){
        (new Server()).ejecutar(); //Iniciamos el hilo del servidor
    }
}

```

## ServerThread

En el ServerThread extendemos la funcionalidad de Thread e implementamos la funcionalidad del server Socket. En este caso lo que hará nuestro servidor es reaccionar a los comandos que se le manden y en caso de recibir un comando inválido enviar un mensaje indicando que el comando no es válido.

```

public class ServerThread extends Thread{

```

```
Socket socket; Declaramos Socket
public ServerThread(Socket socket){//constructor de clase
    this.socket = socket
}
public void run(){//Funcionalidad Arranque del hilo servidor
    try{
        System.out.println("CONECTADO");
        BufferedReader br = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        PrintWriter pr = new PrintWriter(new
OutputStreamWriter(socket.getOutputStream()));
        string[] linealeida;
        do{
            linealeida = br.readLine().split("");//separamos para pillar primero
el comando
            switch(linealeida[0]){
                //Comando ADD, formato: ADD nombre cantidad (add patata 1)
                case "ADD":
                    if(linealeida.length==3){Si el comando tiene 3 elementos
(Comando, nombre, cantidad)
                        ArrayList<Elemento> elementos = null;//Declaramos nueva lista
de elementos
                        Elemento elemento = new Elemento(linealeida[1],
Double.parseDouble(linealeida[2]));
                        elementos= Server.listaElementos.get(linealeida[1]);
                        if(elementos==null){//si no se encuentra el elemento
                            elementos = new ArrayList<Elemento>() //creamos nueva lista
                            Server.listaElementos.put(elemento.nombre, Elementos);
                        }
                        Elementos.add(elemento);
                        pw.println("elemento añadido al nombre especificado");
                        pw.flush();
                    }else{
                        pw.println(linealeida[0] + "El comando necesita mas
argumentos\nEstructura: ADD Nombre, Cantidad");
                        pw.flush();
                    }
                    break;
                //Comando GET, formato: Get Nombre (Get patata)
                case "GET":
                    if(linealeida.length==2){ //Si el comando tiene GET + el Nombre
                        ArrayList<Elemento> elementos = null
                        elementos =
Server.listaElementos.get(linealeida[1]);//buscamos nombre en la lista
                        if(elementos == null){//Si no se encuentra el elemento
                            pw.println("0");
                            pw.flush();
                        }else{
                            double total = 0;
                            for(Elemento elemento: elementos){
                                total+=elemento.cantidad;
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```
        }
        pw.println(total);
        pw.flush();
    }
}else{
    pw.println(linealeida[0]+"no tiene estructura valida\nUsó: GET
Nombre");
    pw.flush();
}
break;
//Comando DELETE, uso: DELETE Nombre (DELETE Patata)
case "DELETE":
    if(linealeida.lenght == 2){
        ArrayList<Elemento> Elementos = null;
        Elementos = Server.listaElementos.get(lineaLeida[1]);
        if((Elementos==null || Elementos.size()==0)){
            pw.println("no hay elementos para este nombre");
            pw.flush();
        }else{
            Elementos.remove(elementos.size()-1);
            pw.println("elemento borrado con éxito");
            pw.flush();
        }
    }else{
        pw.println(linealeida[0] + "se ha usado incorrectamente\nUsó:
DELETE Patata")
    }
    break;
//Comando de salida EXIT
case "EXIT":
    if(lineaLeida.lenght==1){
        pw.println("Cerrado");
        pw.flush();
    }else{
        pw.println("el comando EXIT tiene demasiados elementos")
        pw.flush();
    }
    break;
//En caso de introducir comando invalido
default:
    pw.println("comando inexistente");
    pw.flush();
    break;
}
}while(linealleida[0].contentEquals("exit")==false)
}catch(IOException e){
    e.printStackTrace();
}
}
```

## DELETE

```
        case "DELETE":
            if(linealeida.length == 2) {
                ArrayList<Elemento> elementos;
                elementos =
Server.listaElementos.get(linealeida[1]);
                if(elementos == null) {
                    pw.println("El elemento en cuestion no existe");
                    pw.flush();
                }else {
                    elementos.remove(elementos.size()-1);
                    pw.println("Eliminado");
                    pw.flush();
                }
            }else {
                pw.println("Mal uso del comando DELETE");
                pw.flush();
            }
            break;
```

From:

<https://www.knoppia.net/> - **Knoppia**

Permanent link:

<https://www.knoppia.net/doku.php?id=dad:ejercicioprotocolos&rev=1699987853>

Last update: **2023/11/14 18:50**

