# 75:42 - Taller de Programación I

Ejercicio Nº ___ Padrón _____

Alumno _____ Firma _____

| **Nota**: | | **Corrige**: | | **Entrega #1** |
|---|---|---|---|---|
| | | | | Fecha de entrega |
| | | | | |
| | | | | Fecha de devolución |
| | | | | |
| | | | | |

| **Nota**: | | **Corrige**: | | **Entrega #2** |
|---|---|---|---|---|
| | | | | Fecha de entrega |
| | | | | |
| | | | | Fecha de devolución |
| | | | | |
| | | | | |

El presente trabajo, así como la entrega electrónica correspondiente al mismo, constituyen una obra de creación completamente personal, no habiendo sido inspirada ni siendo copia completa o parcial de ninguna fuente pública, privada, de otra persona o naturaleza.

# Índice general

**1 Autoridad Certificante** **3**
1.1 Introducción . . . . . 3
1.2 Explicación de las clases implementadas . . . . . 3
1.3 Diseño utilizado y relacion entre clases . . . . . 4
1.3.1 Cliente . . . . . 4
1.3.2 Servidor . . . . . 5
1.4 Dificultades afrontadas . . . . . 6
1.5 Aclaraciones . . . . . 7

# 1 Autoridad Certificante

## 1.1. Introducción

El trabajo practico consto en crear una autoridad certificante que soporte la conexión de multiples clientes. El servidor puede agregar o quitar clientes siempre y cuando los mismos envien la información requeridad para llevar la operación a cabo.

## 1.2. Explicación de las clases implementadas

- ArchivoCertificado: Hereda de la clase Archivo. Es la encargada de parsear y contener el certifcado de un cliente. Se utiliza en el modo revoke.
- ClaveCliente: Hereda de la clase Archivo. Es la encargada de parsear y contener la clave publica y privada de un cliente.
- ClavePublicaServer: Hereda de la clase Archivo. Es la encargada de parsear y contener la clave publica del servidor.
- ComandoCliente: Es la encargada de llevar a cabo los modos new y revoke del cliente.
- RequestCliente: Hereda de la clase Archivo. Es la encargada de parsear y contener los datos del cliente. Se utiliza en el modo new.
- Archivo: Es una clase abstracta con el objetivo de que las clases que heredan de ella tengo un metodo para leer y el archivo.
- Certificado: Es la clase encargada de manejar los certificados tanto el en cliente como en el servidor.
- Fecha: En caso que el cliente no tenga fechas en su request esta clase contendra la fecha actual y calculara la fecha en 30 dias.
- Hash: Es la clase encargada de calcular el hash de un string.
- Protocolo: Es la clase que se encarga de cumplir el protocolo de comunicación.
- Rsa: Es la clase encargada de realizar el algoritmo de encriptación.

- SocketConnect: Es la clase que contiene los sockets de comunicación. Tanto el cliente como cada hilo del servidor requeriran una instacia de esta clase para poder comunicarse.
- AcceptadorDeConexiones: Se encarga de aceptar nuevas conexiones al servidor, crea los sockets de conexiín y le deriva la comunicación a otra clase.
- Claves: Es la clase encargada de parsear y contener las claves del servidor.
- Cliente: Es la clase encargada de contener la informacion necesaria de cada cliente.
- ComandoServidor: Es la encargada de llevar a cabo los modos new y revoke del servidor.
- Comunicador: Es la clase que se comunica desde el servidor con cada cliente en particular.
- Indice: Es la clase que contiene la informacion de los clientes en el servidor.
- SocketAccept: Es la clase que acepta nuevas conexiones.
- Thread: Es una clase abstracta. Las clases que heredan de ella podran ejecutarse en varios hilos distintos del hilo principal.

## 1.3. Diseño utilizado y relacion entre clases

### 1.3.1. Cliente

El cliente puede tener dos modos, el modo new en caso que se ejecute de forma exitosa, se iba a obtener un certificado de parte del servidor. El siguiente diagrama muestra las clases intervinientes en este modo.

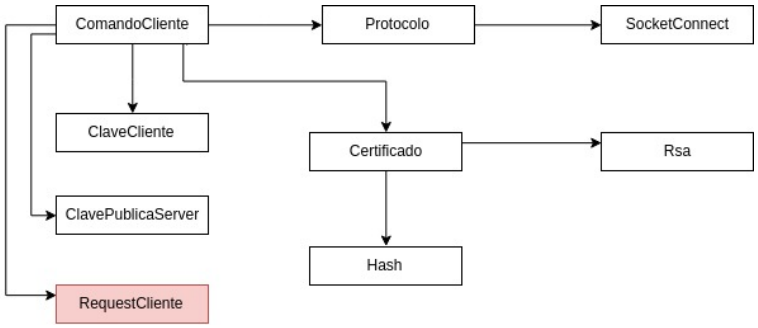


Figura 1.1: Diagrama cliente modo new

El otro modo posible es el modo revoke. En este modo el cliente en caso de que se ejecute en forma exitosa, tiene la posibilidad de dar de baja su certificado. El driagrama es casi identico al del cliente en modo new, la diferencia es la clase que este marcada en rojo, pasa a ser la clase que esta marcada en verde en el siguiente diagrama.

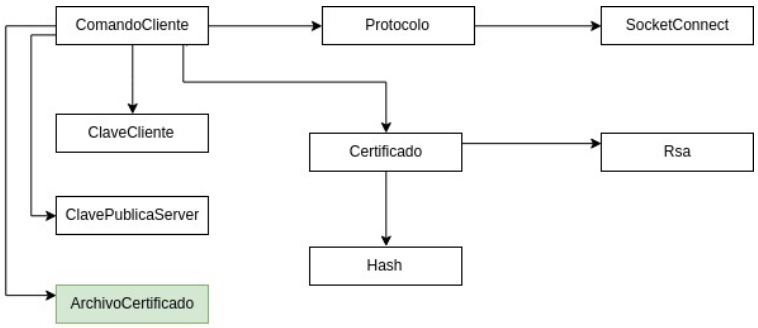


Figura 1.2: Diagrama cliente modo revoke

### 1.3.2. Servidor

El servidor fue la parte mas complicada del trabajo practico ya que hubo que crear un servidor que pueda aceptar varios clientes al mismo tiempo. El servidor tendra varios hilos ejecutandose al mismo tiempo donde todos los hilos van a estar accediento a la misma base de datos, que vendria a ser en mi caso la clase Indice, la cual contiene información de todos los clientes registrados. El siguiente diagrama muestra la situación del servidor teniendo n conexiones al mismo tiempo.

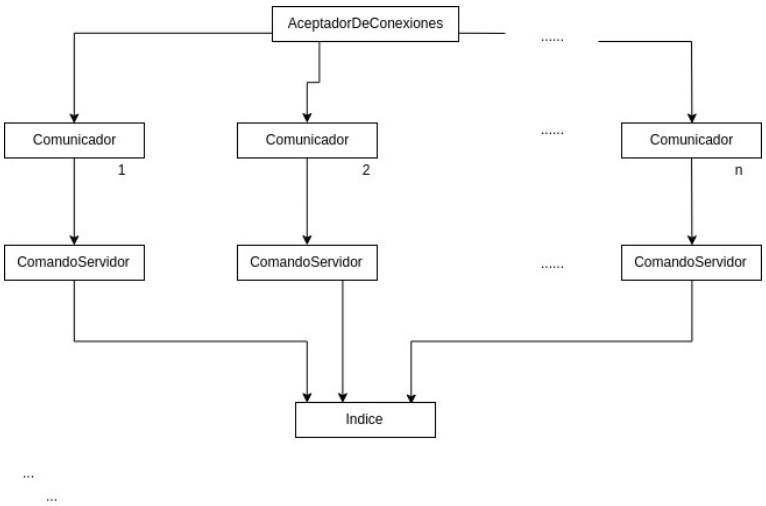


Figura 1.3: Servidor soportando n clientes al mismo tiempo

La clase encargada de aceptar nuevas conexiones cuando un nuevo cliente se quiere conectar crea un nuevo socket de conexión y un nuevo comunicador, la clase comunicador hereda de la clase hilo y sobrecarga el metodo run de forma que cuando se comience a ejecutar el comunicador utilice un socket de conexion (el que creo que aceptador) para comunicarse con el cliente y un ComandoServidor para realizar la logica necesaria. El siguiente diagrama muestra las clases intervinientes cuando se crea una nueva conexión.

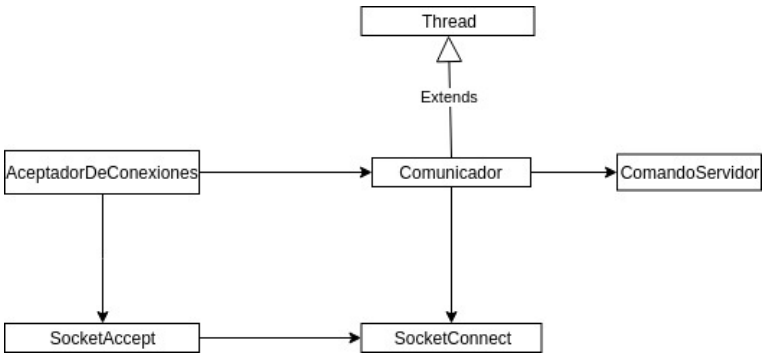


Figura 1.4: Nueva conexion

Finalmente el siguiente diagrama muestra las clases que intervienen en el servidor ya sea cuando se ejecuta el comando revoke o el comando new. Al igual que para el cliente el ComandoServidor utiliza la clase protocolo para comunicarse y la clase certificado para poder calcular los datos necesarios para tener un nuevo cliente. La clase indice es la encargada de tener en memoria la información de cada cliente registrado. En caso que el servidor se detenga esta información se guarda en un archivo.

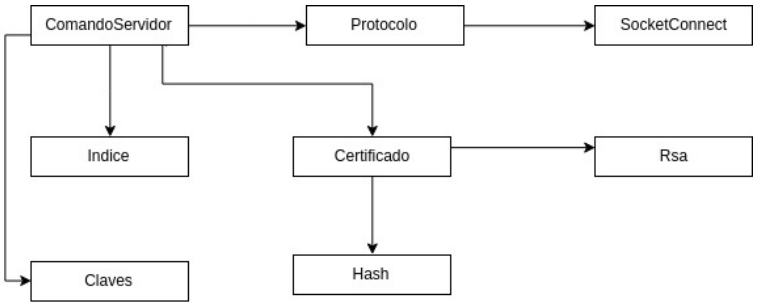


Figura 1.5: Servidor modo new o revoke

## 1.4. Dificultades afrontadas

Las dificultades mas dificiles de resolver fue evitar distintas condiciones de carrera en la clase indice. Las dos primeras condiciones de carrera que se pensaron fueron similares, una de ellas era si dos clientes quiere agregar un mismo sujeto, primero se preguntaba si el sujeto pertenecia al indice y luego se agregaba, entonces para los dos hilos se respondia que no estaba dicho sujeto y luego uno de ellos lo agregaba. Lo que causaba era que el sujeto pueda tener distintos indices en cada ejecución. Para solucionar esto se hizo una funcion bloqueante que pregunte si no esta y lo agregue. El segundo caso posible era cuando se eliminaba un cliente, este caso era pero ya que se preguntaba si pertenecia, para ambos se respondia que si y luego solo uno podia elimnar, se resolvio de forma similar, creando un metodo bloqueante que elimine el cliente si pertenece al indice.

Un caso de carrera de condición mas dificil de resolver, que se vio despues de los casos anteriores, era cuando un cliente se agregaba pero luego se debia quitar porque su hash era invalido. La solucion dicha anteriormente no fue del todo correcta, y gracias a esta condición de carrera lo pude notar, el problema era que se agregaba el cliente al indice si no estaba pero podia haber una doble eliminacion ya que hasta que en un hilo se veia que el hash era invalido y se queria remover el cliente del indice, otro hilo podia venir en modo revoke y quitar dicho cliente. Ademas de la doble eliminación era incorrecto que sea valido quitar dicho cliente del indice ya que este nunca debia estar agregado. Para solucionar este problema la solucion fue crear una "lista de espera"donde el cliente es agregado cuando todavia no se verifico que sea valido su hash. Luego recien cuando se verifica que el hash es valido se agrega al indice, de esta forma si otro hilo viene en modo revoke nunca lo va poder sacar del indice ya que esta guardado en otro lado. La desventaja es que cada vez que quiero agregar un cliente debo iterar sobre la lista de espera y sobre el indice que ya contiene los clientes validos. La gran ventaja de esta lista de espera es no tener una operacion bloqueante desde que se pregunta si el cliente se puede agregar al indice hasta que se verifica que son validos los datos del clie

Proteger el nte.indice, que vendria a ser el serial number de cada certificado tambien tuvo una complicacion, ya que se debia mantener el mismo indice desde que se agregaba a esta lista de espera hasta que, en caso de que el cliente no tenga problemas, se pase al indice de clientes registrados. Para solucionar esto cuando el cliente es agregado a la lista de espera se guarda el indice actual.

Fuera de las carreras de condicion el trabajo puede haber sido mas extenso que complicado ya que no hubo que realizar nuevas tareas, sino aplicar conceptos del trabajo practico 0,1 y 2. En la siguiente seccion se aclara con mejor detalle las tareas que no se llegaron a realizar.

## 1.5. Aclaraciones

Faltaron agregar las siguientes tareas:

- La clase archivo deberia levantar una excepción en caso que no se pueda abrir el archivo. Para ello falto crear una clase error para se lanzado.
- En la clase AceptadorDeConexiones no se pudo hacer join de los hilos que ya finalizaron su ejecución.
- No se pudo implementar la sobrecarga de los operadores « y ». Se quizo realizar en la clase protocolo para enviar y recibir elementos.
- Sobre el final de la entrega se vio que la clase Claves y ClaveCliente tienen la misma responsabilidad.