

# Proxy - Skeleton

## • Interfaccia Servizio:

- + Contiene unicamente il file `IServizio.java` ossia l'interfaccia dove troveremo le definizioni dei metodi da implementare.

## • Lato Server: (per delega)

### 1. ServerThread:

- Bisognerà creare la classe che "estende" `thread` e che eseguirà le operazioni lato server.
- Contiene:
  - + Costruttore: a cui passeremo gli oggetti `Socket` ed `IServizio`
  - + Metodo `run`:
    - o Contiene i `DataStream` `input` ed `output` (che prendono lo stream dalla `socket`)
    - o Optional: Una stringa contenente dei parametri utili
    - o Ciclo di if: un caso per ogni metodo dell'interfaccia.

### 2. Skeleton:

- Implementa l'interfaccia del servizio; in particolare la comunicazione lato server
- Contiene:
  - + Il costruttore: a cui si affidano gli oggetti `interfaccia` e `port`
  - + Un semplice override dei metodi dell'interfaccia
  - + Metodo `runSkeleton`:
    - o Si inizializza una `ServerSocket` a cui si affida la porta del servizio
    - o All'interno di un `while(true)` si inizializza un oggetto `socket` a `ServerSocket.accept()` e si istanzia un `thread server` a cui passare la `socket` e l'interfaccia del servizio

### 3. ServiceImpl:

- Si occupa dell'effettiva implementazione dei metodi delle interfacce e della gestione in Mutua Esclusione delle risorse.

### 4. Service Server:

- Qui si dovrà eseguire la `runSkeleton`, a cui passeremo 2 istanze:
  - o Un oggetto `ServiceImplementation`
  - o Uno `skeleton` (a cui si affida il `s-impl` e la porta scelta per il servizio)

## • Lato Client:

### 1. Service Proxy:

- Si occupa dell'implementazione del passaggio dei pacchetti contenenti parametri e risultato delle procedure chiamate.

#### - Contiene:

- o Costruttore (a cui passare indirizzo e porta del servizio su cui mettersi in ascolto.)
- o Override dei metodi forniti dall'interfaccia:
  - + Si definisce una `socket` per l'ascolto.
  - + Si usano come canali di comunicazione i `DataInputStream` / `OutputStream`

## 2 ClientThread:

- Definisce i thread lato client ed il loro modo di operare.

- Contiene:

- o Costruttore (a cui passare l'interfaccia ed una stringa per la scelta dei metodi)
- o Override di run(): Cosa fa il thread per ogni metodo

(Per ereditarietà)

Cosa Combinia:

### 1. Skeleton:

- Diventa una classe astratta

- Implementa solo runSkeleton() e all'oggetto ServerThread si passano (this, socket)

- Al costruttore si passa solo il port number.

### 2. Service Implementation: (extends SkeletonClass)

- Il costruttore inizializza anche il numero di porto con "Super(port)"

### 3. Service Server:

- Si dichiara solo un oggetto ServiceImpl a cui si passa il numero di porto

- Si richiama con tale oggetto il metodo runSkeleton().

**Nota: Creare un File**

1. Definisce la variabile per il nome: String docName

2. Definisce il percorso: String path = "./" + docName

3. Istanzia un oggetto File: File file = new File(path)

3.1 Check nel ciclo: if(file.exists == false) { file.createNewFile() }

4 Poi per la stampa si sfrutta il FileWriter (a cui si passa l'oggetto file)

**Lavorare ad un File che dovrà contenere più messaggi:**

1. Istanziando l'oggetto File e gli passiamo il path nome.

2. blocco try:

- while(true){
  - 2.1 Istanziando un FileOutputStream e gli passiamo il file.
  - 2.2 Istanziando un BufferedWriter e gli passiamo un istanza OutputStreamWriter che contiene il file.out.
  - 2.3 Utilizziamo bw.write() e bw.newLine() per scrivere (all'interno di un ciclo for!)
  - 2.4 bw.newLine() finale + bw.close();

## Invio pacchetti:

### Com TCP:

- + Three way handshake
- + Buona affidabilità
- + Uso dei DataStream

### Com UDP:

- + Pessima affidabilità
- + Ricomposizione dei diversi porti su singolo host
- + Uso dei Datagram Packet

## Socket:

Mecanismo di comunicazione tra processi attivi su host e la rete; avremo una socket per ogni processo/porta.

Tramite il **binding** si associa una socket ad un particolare servizio dell'host, affidabile il suo indirizzo ip e rispettivo port number

### in Java:

- + **Socket**: permette di creare la connessione alla porta desiderata e di ricevere/invia dati
- + **ServerSocket**: solo su lato Server, attende il collegamento con il port affidato e restituisce l'oggetto Socket per la comunicazione.

→ La accept andrà messa in un ciclo while.

## Comunicazione:

### + TCP:

- o Si utilizzano gli Input ed output stream; sui quali scriviamo e leggiamo con i reader e writer stream.

### + UDP:

- o Connectionless: Non ci sarà bisogno di instaurare alcuna connessione, la stessa socket è utilizzabile per raggiungere più destinatari!
- o Usa i Datagram Packet.
- o Sfrutta le DatagramSocket

## Astrazione dell' ip:

Si usa la classe **InetAddress** con alcuni metodi utili:

- **Getlocalhost**
- **GetbyName (host nome)**

Poi vorremo gestire le **UnknownHostException** e le **SecurityException**

## Servizi:

I servizi offerti dal server sono sempre implementati tramite un'interfaccia.

## Proxy - Skeleton

- **Proxy:**
  - + Implementazione locale del servizio remoto del server.
  - + Incapsula tutto ciò che serve per instaurare una comunicazione e per inviare e ricevere messaggi
- **Interfaccia Server:**
  - + Implementazione: Contiene i servizi implementati
  - + ServerProxy: Si occupa dell'invio delle richieste da client a server

Come avviene la comunicazione:

**Dato Client:** Il client ha un oggetto **InterfacciaServer ServerProxy** che si occupa della comunicazione

**Dato Server:** Si ha un analogo del proxy, detto **Skeleton**

**Skeleton:**

Per **esecutività**: Implementa solo i servizi di comunicazione; lascia i restanti a **ServerImpl**, sua sottoclasse

Per **delega**: Contiene solo la firma dei metodi che non implementa

» **Abstract Class**

## Annotazioni dagli esercizi

### Socket UDP.

+ Server Side:

- o Thread:

1. Vanno passati: `Interfaccia`, `DatagramPacket` e `DatagramSocket`
2. Nel `run()`:
  - 2.1 Creare array di byte che contiene i dati del packet offset: da che byte comincia
  - 2.2 Creare oggetto `String` a cui passare: `(buffer, 0, packet.getLength())`
  - 2.3 Usare interfaccia `Logger`
  - 2.4 Creo un pacchetto per la risposta, a cui passare:
    - 2.4.1 `risposta.getBytes()`
    - 2.4.2 `risposta.getBytes().length`
    - 2.4.3 `packet.getAddress()`
    - 2.4.4 `packet.getPort()`

- 2.5 Send della risposta

- o `Skeleton`

+ Per delega:

1. Si possono interfaccia e port al costruttore.
2. Override dei metodi dell'interfaccia
3. Nel `runSkeleton()`:

3.1 Creiamo una `DatagramSocket` a cui affidiamo la port

3.2 Inizializziamo un array byte a [65508] byte.

3.3 While (+true):

3.3.1 Inizializza pacchetto a cui passare array e array.length

3.3.2 `Socket.receive(pkt)`

3.3.3 Istruziona thread specifico

3.3.4 Avvia thread

- o Implementazione:

1. Costruttore con oggetti utili

2. Implementazione Metodi

- o Main Server:

1. Passaggio parametri `args[ ]`

2. Creazione oggetto `impl.`

3. `Skeleton obj`

4. `runSkeleton`