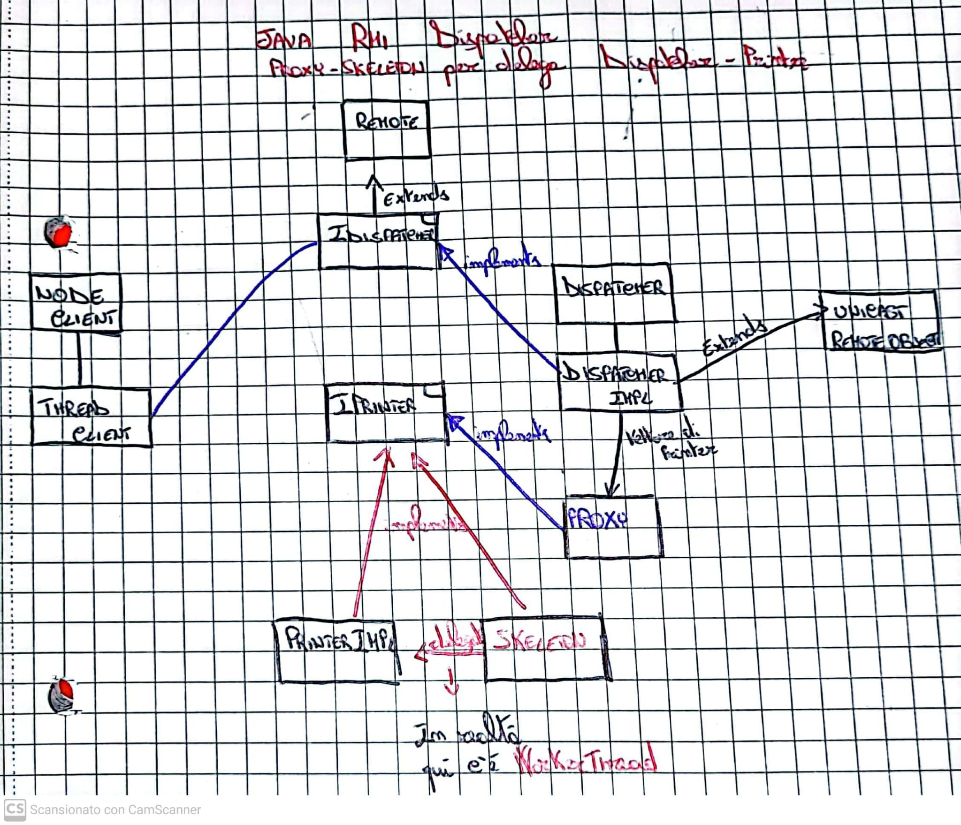
**Schema generale**

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

**Class Diagram**



* **Dispatcher**
  + **addPrinter**(IPrinter printer): metodo che permette l’**aggiunta** di una **stampante** al **Dispatcher**.
  + **printRequest**(String docName): consente la **richiesta** di **stampa** del documento **docName**.
* **Printer**
  + **print**(String docName) : ciascuna printer può eseguire **1 print** in maniera concorrente (un’unica print alla volta). Dura un tempo casuale tra 5 e 10 secondi.
    - Se occupata: return false.
    - Se libera: **scrivere** sul **file** **docName** il **docName**.
* **Node**

Ci saranno un **numero arbitrario** di **Printer** lanciate all’avvio o durante l’esecuzione del programma.  
Ogni **printer** è avviata passando come parametro dell’avvio il **portNumber** e il **nomeFile** su cui saranno salvati i nomi dei **documenti stampati**:

java nomePackage.PrinterServer 8000 documentiPrinterUno.txt

Seguiamo una printRequest inviata dal **node** per capire come avviene la comunicazione tra:

**Node – Dispatcher – Printer**

La comunicazione tra **Node**, o meglio **ThreadClient** e **Dispatcher** avviene sfruttando il **meccanismo** di comunicazione remota **JavaRMI** easy.

**Node  
Main** Client, in cui andremo a creare il **vettore** di **5 Thread** che andremo ad avviare, passandogli il **riferimento** del **servizio remoto** ricavato tramite **lookup** sull’**RMI Registry**, myDispatcher.

**ClientThread**Ogni **thread** invoca **3 printRequest**, con **docName** generato in maniera casuale in modo da avere prefisso ***doc*** + un numero intero tra 1 e 50, ogni **3 secondi**.

**Dispatcher**

**Main Server**, andremo semplicemente a pubblicare il **servizio remoto** implementato nella classe **DispatcherImpl** nell’RMI registry.

**DispatcherImpl**  
Implementando per **ereditarietà** la definizione di un **servizio remoto**, l’oggetto **servente** implementerà i **metodi remoti** **addPrinter** e **printRequest** e avrà un vettore di **attachedPrinters**.

* addPrinter(int port): l’invocazione di questo metodo deve avvenire tramite **RMI** e non tramite **proxy-skeleton**. Quello che accade è che quando una **printer** invoca questo metodo è che il **dispatcher** aggiungerà un **proxy** **associato alla stampante** (che si distingue per il **port** su cui è in ascolto).

1.    IPrinter printer = new Proxy(port); //Costruiamo il proxy associato.

2.    attachedPrinters.add(printer);

Questo perché abbiamo bisogno di un **riferimento** al proxy per poter poi andare ad invocare la **printRequest**.   
E’ questo che si intende per **callback attraverso socket**!

Praticamente **ad ogni stampante** viene assegnato un **port number specifico**, dunque avremo un **proxy** creato **dinamicamente** quando una **nuova stampante** si registra.

* printRequest(String docName): All’arrivo di una **richiesta di stampa**, il dispatcher itera attraverso i **proxy** e **inoltra** le **richieste** utilizzando le informazioni memorizzate nei **proxy**:

1.  result = attachedPrinters.get(i).print(docName);

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

Praticamente il **dispatcher** agirà come **intermediario** per gestire la **comunicazione** bidirezionale tra i **threadClient** e le **stampanti**.

**Proxy**  
Implementa ovviamente l’**interfaccia IPrinter**, dato che deve fare da intermediario nella comunicazione tra il **dispatcher** (che riceve le **richieste** di stampa e le inoltra alle stampanti fino a chè non gli tornano un **true**, cioè che la stampa va a buon fine, oppure finiscono le stampanti da interrogare).

Nel **costruttore** del **proxy** andiamo a impostare il valore del **portNumber**, che verrà poi utilizzato nel metodo **boolean** **print**(String docName) per la creazione della **socket** per inviare la **richiesta** di **print** e il **nome del documento** alle **stampanti corrispondenti**.

Letteralmente in questo esercizio, avremo **tanti proxy** e **tanti skeleton** quante sono le stampanti!

**Skeleton**  
Implementato per **delega**, dunque sarà una classe semplice che **implementa** l’interfaccia **IPrinter**.  
Nel costruttore ovviamente gli passiamo l’implementazione di **IPrinter** ma anche il **numero di port** su cui poi andrà creato nel **runSkeleton**() la **serverSocket** su cui **la singola stampante** andrà ad ascoltare.

**PrinterThread**  
Ad ogni **ThreadWorker** passiamo la **socket** (che rappresenta la connessione) e l’implementazione IPrinter (cioè un oggetto **PrinterImpl**).  
Questo farà l’**unmarshalling** dei messaggi che provengono dal **proxy corrispondente** alla stampante e andremo dunque a **invocare** il metodo **print** sulla stampante finalmente!  
Manderà indietro poi una **booleana**, che verrà ricevuta dal **proxy** corrispondente, e poi tornato al **dispatcher** che deciderà se chiedere ad un’altra stampante o meno.

**PrinterImpl**

Semplicemente implementa la scrittura su file del **docName**, che deve avvenire in maniera **mutuamente esclusiva**. Se è già occupata la stampante, deve restituire false.

Conviene farlo con il **Semaphore**.