**ES Local -> Multi con socket**

1. Le classi che devono comunicare tra client e server sono messe serializable
2. La classe comune per il controllo ha metodi synchronized
3. Il server:
   1. Crea un serversocket con la porta
   2. Crea un elemento per il controllo degli altri (tipo la banca)
   3. Crea un socket all’infinito per ogni connessione
   4. Crea un thread passandogli l’elemento comune e il socket
   5. Finally si chiude
4. Il serverThread
   1. Memorizza alla creazione il socket e l’elemento comune, creando in e out come ObjInput e ObjOutput streams e starta
   2. All’infinito prova a leggere una richiesta e se è end esce (il readobj viene castato con il tipo necessario per creare l’oggetto inviato)
   3. Esegue la richiesta
   4. Scrive la risposta
   5. Finally prova a chiudere il socket
5. Il Multiclient
   1. Crea un inetaddress con nome nullo
   2. Con un while crea n threads e gli passa l’indirizzo, dormendo se si è già al numero massimo
   3. Ha un main che fa partire questa funzione (o questo è il main)
6. Il clientThread
   1. Salva l’id del client
   2. Crea un nuovo socket con address e porta default
   3. Crea nuovi in e out e starta
   4. In un loop (prob in un try too) esegue n volte delle operazioni con writeobject e leggendo il risultato, printando localmente l’esito
   5. Finally close il socket
7. Se si ha un proxy, esso viene creato con un socket (gli in e out) dal client thread e implementa le chiamate per le operazioni con semplicemente il read e writeobject.
8. Se c’è necessità dell’attesa di una risposta un client slave verrà creato al posto della lettura della risposta con il socket input del proxy e il risultato; esso sarà letto col readobj e setterà un flag di completamento nel risultato, copiando i valori da esso nel res passato da clientthead al proxy

**ES Local -> Multi con RMI**

1. La classe server-side ha deve avere un’interfaccia che estenda Remote e indichi tutti i metodi che vengono chiamati da remoto; ciò vale anche per il client
2. La classe server deve
   1. estendere UnicastRemoteObject e implementare l’interfaccia
   2. ridefinire i metodi dell’interfaccia
   3. (nel main) creare un oggetto del tipo della classe
   4. Crare un registry (con LocateReg.createReg(porta))
   5. Rebindare il registro con nome e oggetto
   6. Se si vuole avere un callback serve una lista dei socket client accettati e un metodo per aggiungerli (chiamato dal client) e un metodo client per invocare il callback (tipo reportnews)
3. La classe client deve
   1. (estendere thread)
   2. Creare un registro che localizzi il registro con LocateReg.getReg(porta))
   3. Creare un oggetto a partire dal cast del lookup del registro col nome dato (stub)
   4. (Implementare eventualmente i metodi dell’interfaccia che il server può chiamare; si passa il riferimento con un oggetto della stessa classe castato da UnicastRemoteObject.exportObject(this, PORT))