**Esercizio Java RMI – Pattern Observer**

Considero un sistema, composto da **Client-Server** e degli Observer.

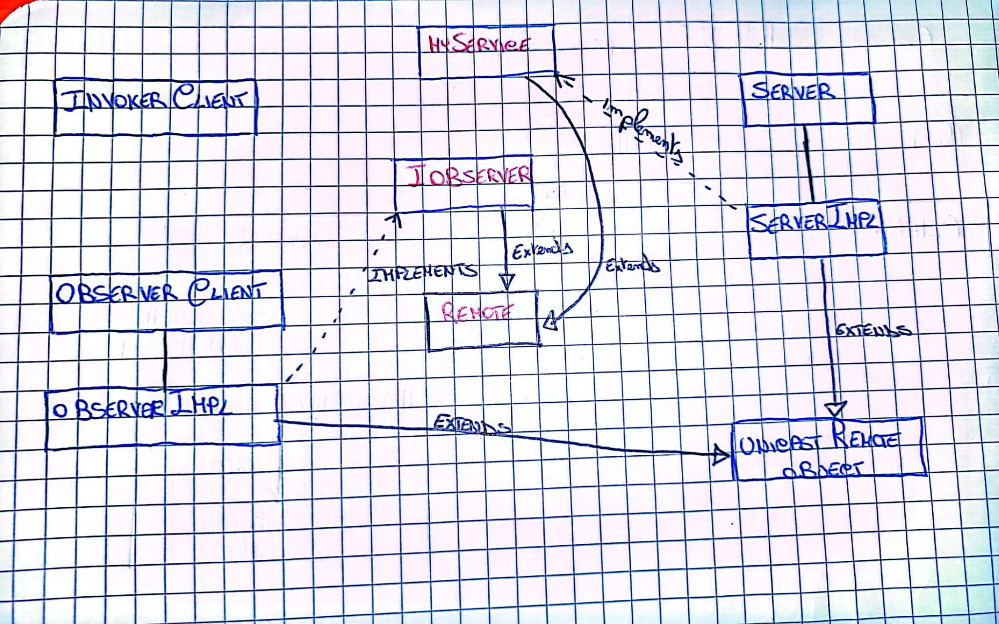
* **Server**:
  + espone un **servizio remoto** (service\_method()).
  + **Registra** in una **lista** una serie di **ClientObserver** interessati all’invocazione del servizio remoto.
  + Notifica tutti gli **observer** registrati quando il **servizio remoto** viene **invocato**.
* **Client Invoker**: invocano il servizio remoto (service\_method()).
* **ClientObserver**: sarà ovviamente il soggetto per il servizio remoto di Callback.

**Pattern Comportamentale Observer**

Definisce una relazione **uno a molti** tra oggetti, in modo che quando un **oggetto** **cambia** **stato** tutti i suoi dipendenti vengono notificati.  
Si parla di:

* **Molti osservatori**, **subscriber**, che osservano un particolare **subject**.  
  Questi si **registrano** presso un subject per ricevere notifica quando viene apportato una modifica all’interno di quel soggetto.  
  Così come possono registrarsi ovviamente possono anche **annullare** la registrazione, ciò li rende oggetti debolmente accoppiati.

**Class Diagram**



Noto che la **definizione** dei **servizi remoti** avviene per **ereditarietà**; dunque, le interfacce estenderanno l’interfaccia remote e verranno implementate.

**Interfacce**

* **MyService**:
  + espone il **servizio remoto** vero e proprio, che semplicemente andrà a stampare una stringa.
  + espone il metodo **attachObserver**, che consente l’**attach** di un **observer** al **Subject**.
* **Observer**:
  + Espone il **metodo di callback** observerNotify(), che verrà invocato dal **Server** ad ogni invocazione del metodo service\_method().
    - Ovviamente, per invocare questo metodo, l’**ObserverClient** (main degli observer) creerà quest’**oggetto Observer** (che avrà una determinata implementazione ObserverImpl) il cui metodo **notify** verrà invocato da remoto dal Server.

Partiamo prima con il **Servizio** e tutto ciò che lo riguarda. Semplicemente è un **metodo remoto** invocabile dall’ **InvokerClient** tramite **RMI**.  
Dunque consideriamo:

**Server**

**ServerImpl**  
Come **dato membro privato** c’è il **vettore** di **observer** che sono “**attached**”.

* **Costruttore**: oltre a creare il vettore di observer, viene sempre richiamato il costruttore **super**() per richiamare il costruttore di **UnicastRemoteObject**. Questo perché il costruttore di **UnicastRemoteObject** esporta l’oggetto remoto.
* **service\_method**(): metodo che verrà invocato da **InvokerClient**, e ogni volta che verrà invocato, il **subject** (dunque lo stesso oggetto remoto che espone questo metodo) **notificherà** tutti gli **observer** tramite un metodo che verrà definito sempre all’interno di quest’oggetto (notifyAllObserver).
  + **notifyAllObserver**: è un metodo privato, dato che verrà richiamato solo dallo stesso oggetto, il **subject** in questo caso.   
    Scorriamo il vettori di observer e invochiamo il **metodo ObserverNotify**, che è la **callback** offerta da **Observer**.
* **attachObserver**(): metodo che verrà invocato dagli **Observer** per attaccarsi al subject.

**Server**

Il compito principale sarà quello di **registrare** l’oggetto remoto sull’RMI Registry ovviamente.

Ricorda che:

try{

    //Devo inserire il try-catch dell'eccezione RemoteExcteption perchè l'ho trowata nel costruttore ServerImpl()

    //infatti super(), ovvero il costruttore di UnicastRemoteObject può generare quell'eccezione.

        MyService myService = new ServerImpl();

    }catch(RemoteException e){

        e.printStackTrace();

    }

**Invoker Client**

Ha il compito semplice di dover invocare il **servizio remoto** esposto dall’interfaccia **MyService**.  
Dunque semplicemente effettuare da prassi il **lookup** sull’RMI Registry:

  MyService stub =(MyService) rmiRegistry.lookup("myService")

**ObserverClient**

Ha il compito di creare un **observer** per effettuare poi il successivo **attach** al subject, tramite chiamata remota del servizio offerto da myService.  
Ovviamente, per favorire il meccanismo di **callback** questo attach conterrà un riferimento al **observer attached**, cioè al vero e proprio **oggetto callback**.

**ObserverImpl**

Fornisce l’implementazione della callback semplicemente, ed essendo l’implementazione di un servizio remoto.