Università degli Studi dell'Insubria
Dipartimento di Scienze Teoriche e Applicate

Programmazione Concorrente e Distribuita
RMI

Luigi Lavazza
Dipartimento di Scienze Teoriche e Applicate
luigi.lavazza@uninsubria.it

Esercizio 1

- Si desidera realizzare un sistema client-server RMI in cui il server offre la funzione per calcolare il massimo comune divisore (MCD) di due interi dati.

- Si modifichi il codice dato, che implementa l'interazione tra un 'client' e il 'server' localmente.

Es. 1 – codice dato

- File MCD.java

```
public interface MCD {
 public int mcd(int n, int m);
}
```

- File MCDimpl.java

```
public class MCDimpl implements MCD{
   public int mcd(int n, int m){
        int r;
        while(m != 0) {
              r = n % m;
              n = m;
              m = r;
        }
        return n;
   }
}
```

Es. 1 – codice dato

- File MCD.java

```
public class MyMain {
  public static void main(String[] args) {
    MCDimpl euclMCD= new MCDimpl();
    int x, y;
    x=18; y=3;
    System.out.println("MCD("+x+","+y+")="+euclMCD.mcd(x,y));
    x=18; y=6;
    System.out.println("MCD("+x+","+y+")="+euclMCD.mcd(x,y));
    x=18; y=7;
    System.out.println("MCD("+x+","+y+")="+euclMCD.mcd(x,y));
    x=18765; y=345435;
    System.out.println("MCD("+x+","+y+")="+euclMCD.mcd(x,y));
  }
}
```

Esercizio 2

- Si implementi usando RMI un sistema client server in cui il server pubblica informazioni e i client le leggono.
- Il server crea un'informazione (una stringa di testo) e la mette in un buffer. Ogni nuova informazione rimpiazza la precedente, indipendentemente dal fatto che sia stata letta o meno.
- Ogni volta che crea una nuova informazione, il server manda una notifica ai client che si sono "abbonati" al servizio.
- Inizialmente, i client --dopo aver identificato il server attraverso il registry-- gli mandano la richiesta di "abbonamento" cioè gli mandano la richiesta di essere avvertiti ogni volta che risulta disponibile una nuova informazione.
- In seguito, quando ricevono la notifica, i client utilizzano un metodo del server per leggere la notizia.

Esercizio 2

- Si può realizzare il sistema richiesto modificando il codice dato, che implementa un server che si limita a mettere a disposizione un metodo per leggere le notizie. I Client non sanno quando c'è una nuova notizia da leggere.

Interfaccia remota **`NewsService`**

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface NewsService extends Remote {
  public String readNews() throws RemoteException;
}
```

NewsServiceImpl

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class NewsServiceImpl extends UnicastRemoteObject
                             implements NewsService {
  private int ncount=0;
  private String s="abc"+ncount;
  public NewsServiceImpl() throws RemoteException {
    super();
  }
  public synchronized void updateNews()
                                 throws RemoteException {
    s="abc"+(++ncount);
  }
```

NewsServiceImpl

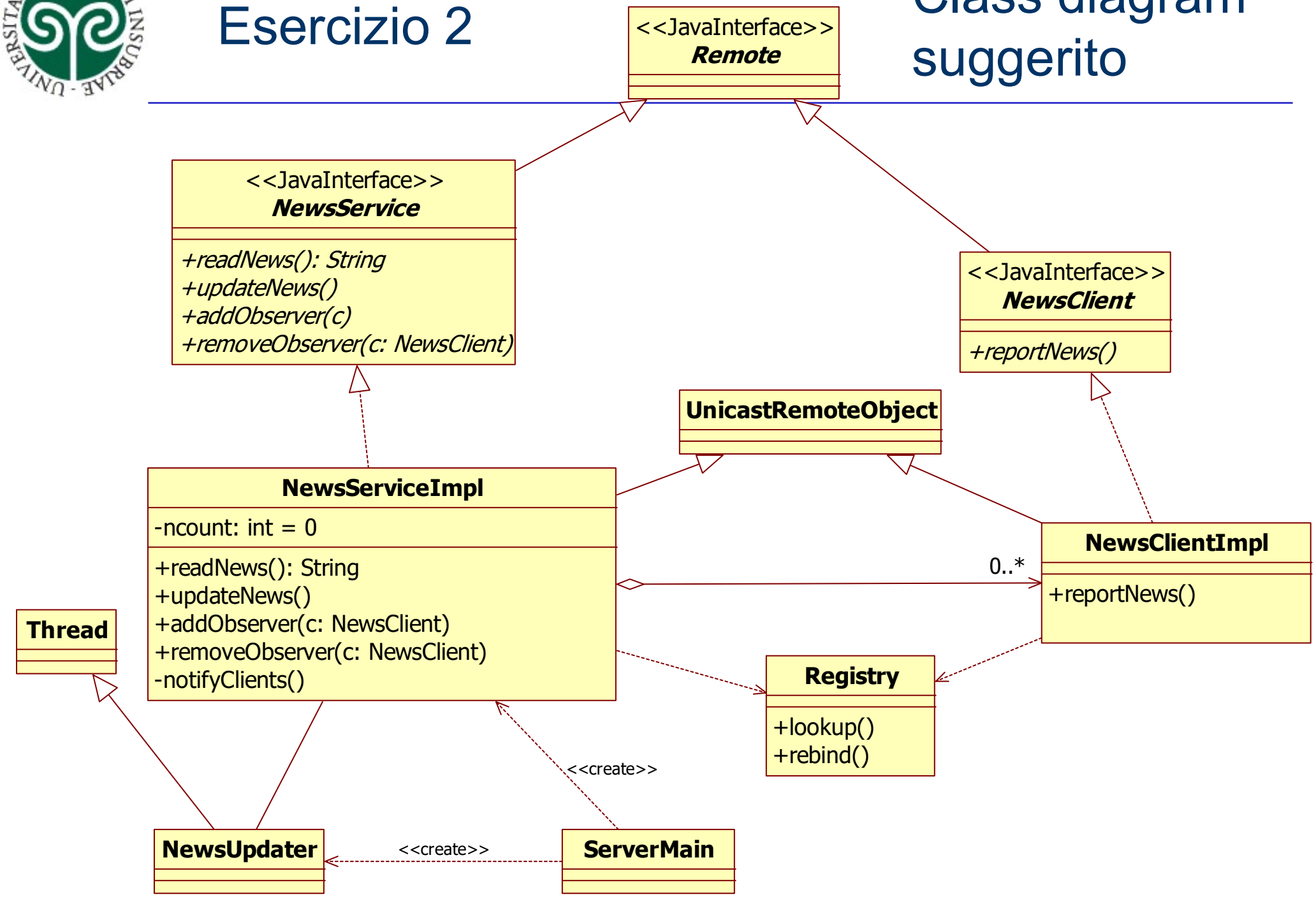
```
  public synchronized String readNews()
                                    throws RemoteException {
    String theNews=s;
    updateNews();
    return theNews;
  }
  public static void main(String args[]) {
    try {
      NewsServiceImpl obj = new NewsServiceImpl();
      Registry reg = LocateRegistry.createRegistry(1099);
      reg.rebind("NEWS", obj);
      System.err.println("Server ready");
    } catch (Exception e) {
       System.err.println("Server exception: " +
                          e.toString());
       e.printStackTrace();
    }
} }
```

Esercizio 2

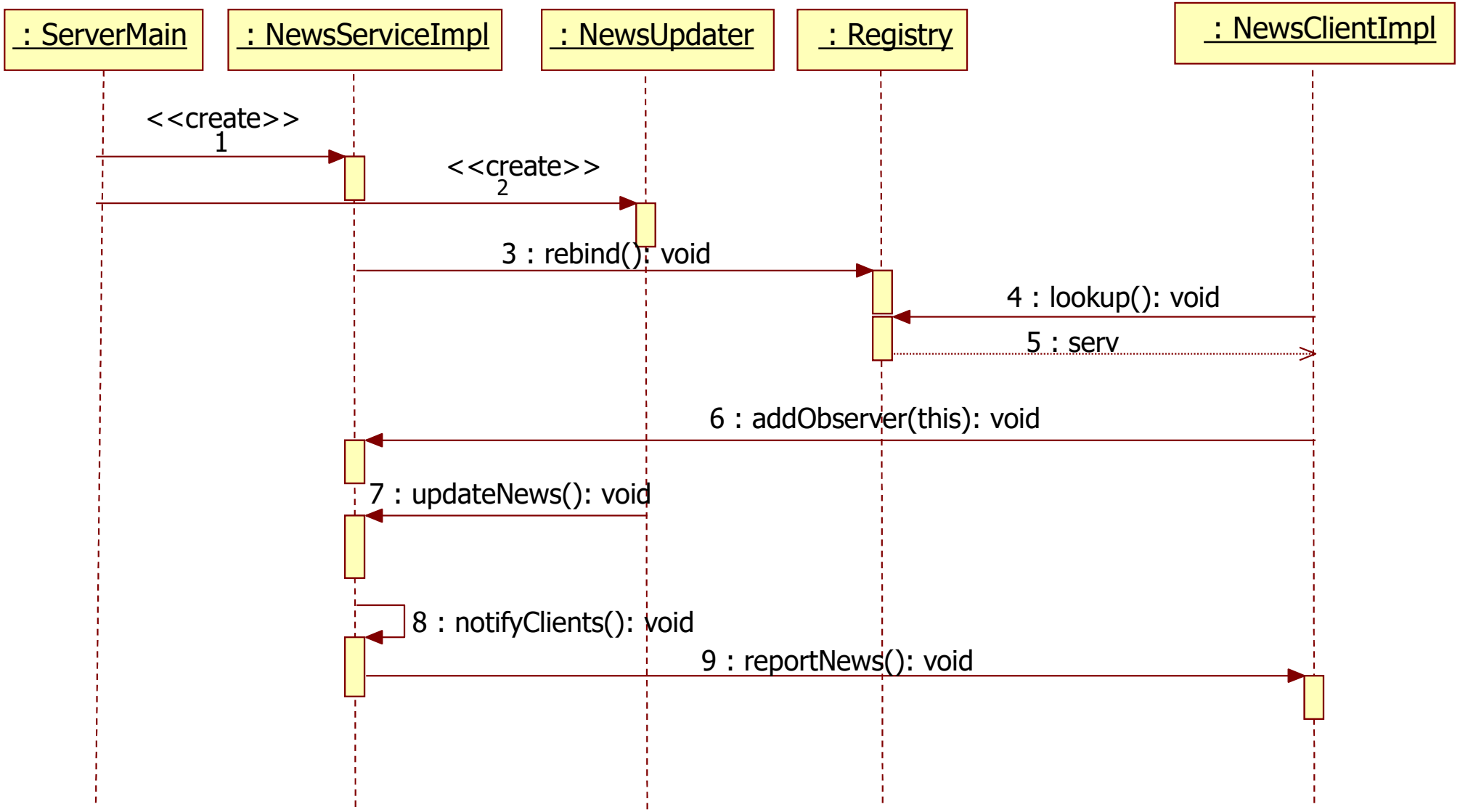
- Le slide seguenti illustrano una possibile struttura del sistema.

Esercizio 2

Class diagram suggerito



Esercizio 2: sequence diagram suggerito



Esercizio 2: primo passo

- Modifichiamo il programma dato
  - In modo che sia multiclient
    - NB: solo per testing, i client stanno su machine diverse
  - Un thread si occupa dell'aggiornamento delle notizie
    - Asincrono. Il server non sa quando le notizie saranno modificate
  - Un main lancia il server e il thread di aggiornamento
- Non introduciamo ancora le callback

Esercizio 2: callback

- I client si registrano presso il server
- Ogni volta che la notizia viene aggiornata, il server manda la notizia a tutti i client registrati.

Esercizio 3

- Si scriva il codice di un sistema client server che funzioni come segue.
- Il server fornisce un metodo **`warnAt`**.
  - Un client chiama **`warnAt(X)`** al tempo **`T`**.
  - Il server chiamerà il metodo **`notifyWarn()`** del client **`X`** secondi dopo, cioè al tempo **`T+X`**.
- Si implementi il sistema usando RMI.
- Una volta ottenuto un sistema funzionante con un solo client, lo si testi usando più client.

Esercizio 3: soluzione es03_bloccante

- Soluzione banale. Il client resta bloccato per tutto il tempo.
- Testabile con più clienti lanciandoli a mano

- Domanda:
  - Il metodo **`warnAt`** del server chiama il metodo **`notifyWarn`** del client mentre il client sta ancora attendendo la terminazione del metodo **`warnAt`**.
  - Come è possibile che il client esegua **`notifyWarn`** se sta eseguendo il **`main`**?

Esercizio 3: soluzione es3_bloccante_multi

- Per test con tanti client
- Il server genera implicitamente un thread per ogni chiamata di **`warnAt`**, come si vede grazie all'istruzione

```
System.out.println("server "+
Thread.currentThread().getName()+
" riceve richiesta da "+c+" per "+X+" sec.");
```

- Si vede che il server «riutilizza» i thread usati per i client terminati.

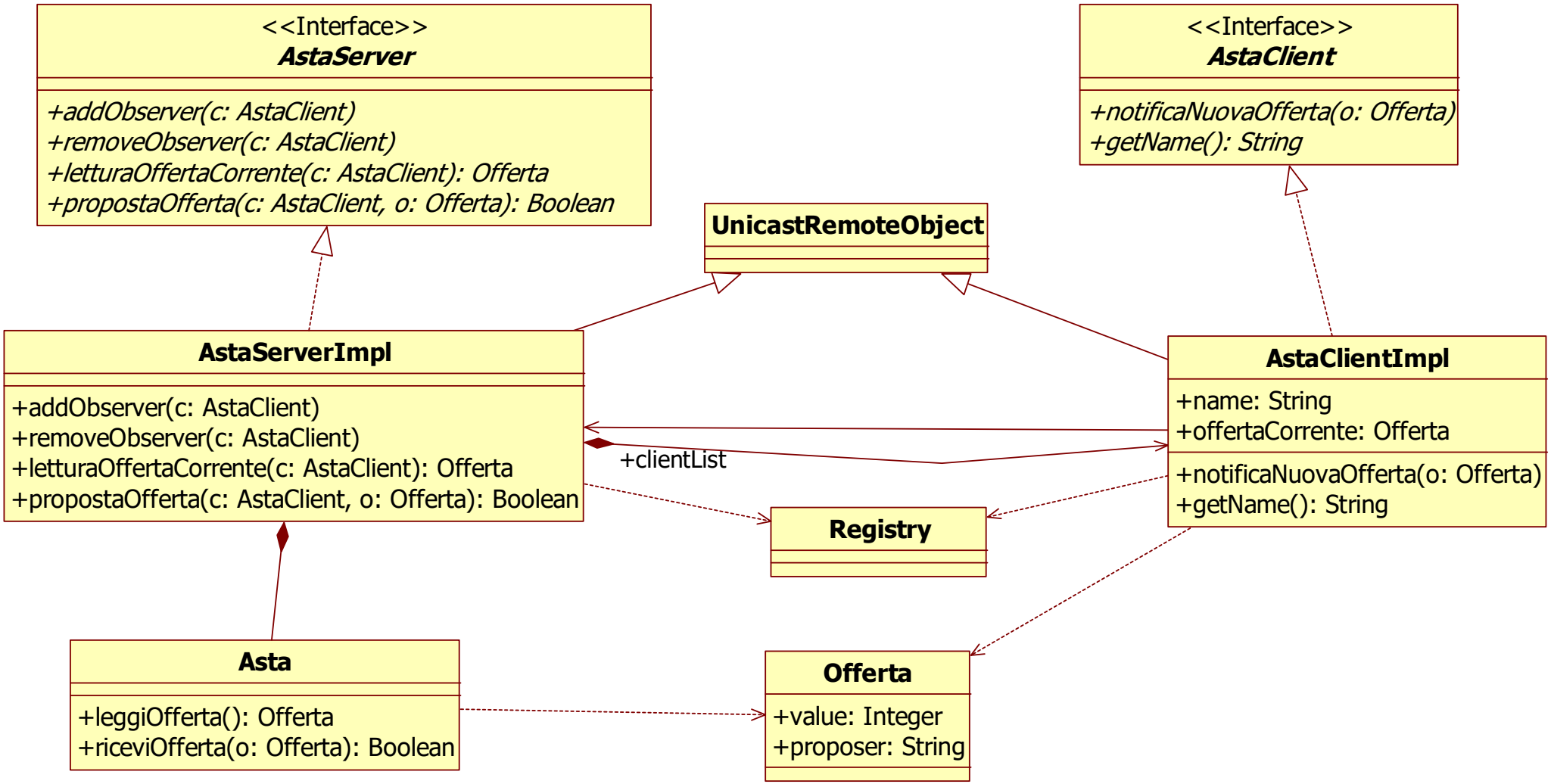
Esercizio 3: soluzione es3_non_bloccante

- Il server genera un thread che manderà la notifica al momento giusto. Il metodo **`warnAt`** termina subito, così il client può proseguire le sue elaborazioni.
- Testabile con più clienti lanciandoli a mano
- Si vede che il client prosegue le sue elaborazioni dopo aver richiesto **`warnAt`**, e che la notifica arriva durante queste elaborazioni.

Esercizio 4

- Implementare usando RMI l'asta semplificata già realizzata con i socket.

Class diagram suggerito



- Rifare con RMI gli esercizi già fatti con Socket