ESERCITAZIONE TIGA: RMI, unico host, caricamento locale delle classi

Un `Conto` bancario è caratterizzato esclusivamente da un `nome` ed un `saldo`. Il `nome` identifica univocamente un `Conto`, tra gli oggetti istanziati. Un oggetto di classe `Conto` viene implementato come un *oggetto remoto*, ossia con (alcuni) metodi invocabili da applicazioni in esecuzione su altre JVM (inizialmente supposte, per semplicità, residenti su un unico host) tramite l'*interfaccia remota* `Conto` (Fig.1):

```
public interface Conto extends Remote {
  void versa(double importo)     throws RemoteException;
  void preleva(double importo    throws RemoteException,
                                 NegativeAmountException;
  double ritornaSaldo()          throws RemoteException;
}
```

Nella creazione dell'oggetto `ContoImpl`, si definisce il nome dell'oggetto e si imposta il saldo iniziale al valore zero. Il nome del conto viene anche adoperato come *nome pubblico* per registrare l'oggetto *Conto* presso l'RMI Registry.

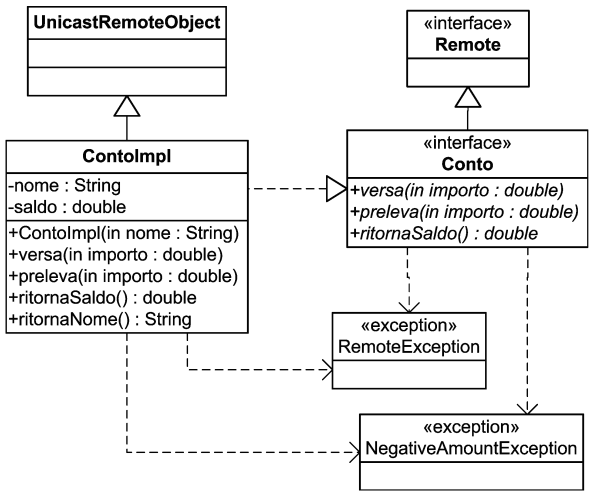

Fig. 1 – Diagramma delle Classi.

PROGETTARE E REALIZZARE:

• La classe `ContoImpl`, implementazione **thread-safe** dell' interfaccia. A tale scopo, si dichiarino semplicemente i metodi come `synchronized` per garantire un accesso mutuamente esclusivo.

• La classe `ContoServer`, che implementa il programma *server* in accordo alle seguenti specifiche:
  i) *crea* (ed *esporta*) un insieme di conti, dai nomi passati da riga di comando;
  ii) li *registra* presso l'RMI Registry.

• La classe `ContoClient`, che implementa il programma *client* in accordo alle seguenti specifiche:
  i) riceve *`nome`*, *`importo1`* ed *`importo2`* come parametri di ingresso, da riga di comando;
  ii) recupera un oggetto di classe `ContoImpl_Stub` (che funge da proxy per l'oggetto `ContoImpl` residente sul server) con tale *`nome`*, e lo riferisce con un riferimento interfaccia `conto`.
  iii) invoca remotamente i seguenti metodi:
  `conto.versa(importo1); conto.preleva(importo2);`
  `System.out.println(conto.ritornaSaldo());`

• La gestione delle seguenti eccezioni: `NotBoundException` e `java.net.MalformedURLException` (metodo `Naming.`**`lookup`**), `RemoteException` (metodi dell' interfaccia `Conto`), `AlreadyBound Exception` (metodo `Naming.`**`bind`**), `NegativeAmountException` (classe eccezione sviluppata "ad hoc" per intercettare un prelievo superiore al saldo, quindi sollevata dal metodo `Conto.`**`preleva`**).

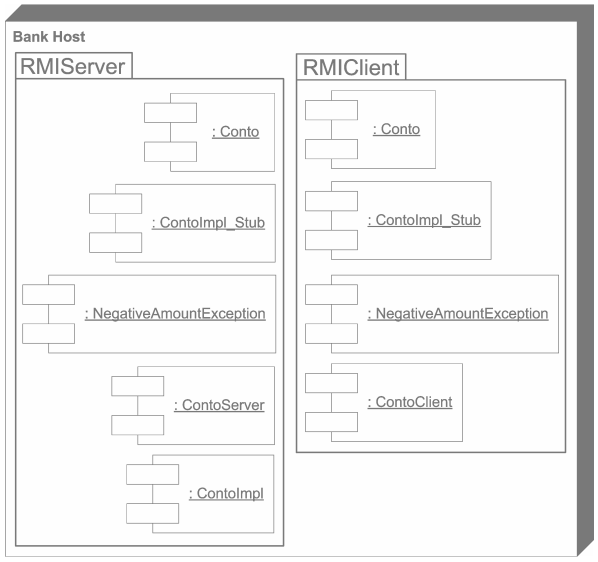

Fig. 2 – Diagramma di distribuzione dei componenti.

SUGGERIMENTI:

• Compilare tutti i sorgenti (`javac *.java`) in un' unica cartella, successivamente produrre la classe *stub* mediante l'applicazione *rmic* della directory *bin* (`rmic -v1.2 ContoImpl`), e distribuire il bytecode in tre cartelle, *RMIregistry, RMIserver* ed *RMIclient*, come indicato in Fig.1-a e Tab.1.

Tab. 1 – Moduli necessari ad ogni sottosistema, a compile e run time.

	ContoServer	ContoClient	RMI Registry
compile time	server, interfaccia remota e relativa implementazione, altre classi adoperate (da questi moduli).	client, interfaccia remota, altre classi adoperate (da questi moduli).	
run time	server, interfaccia remota e relativa implementazione, altre classi adoperate, stub	client, interfaccia remota, altre classi adoperate, stub.	interfaccia, altre classi adoperate, stub.

• Adoperare l'applicazione *rmiregistry* della directory *bin*, digitando `rmiregistry 1099` [1] per far partire l'RMI Registry, prima del client e del server, dalla cartella *RMIregistry*.
• Avviare i processi `ContoServer` e `ContoClient` dai rispettivi percorsi (quindi su JVM differenti).
• In `ContoClient` passare come parametro di ingresso anche l'host di residenza del registry, in modo da costruire, per il metodo `lookup`, una URL come nel seguente esempio `//10.114.109.11:1099/pippo`[2].

[1] Se vi sono problemi di connettività con RMIRegistry, provare a cambiare porta (es. `1100`, `1101`,...).
[2] Cerca l'oggetto `pippo` nel registry che risiede in `10.114.109.11` sulla porta di default `1099`).