Agenti in Java RMI

- Un agente è una computazione che agisce per conto di un host presso un altro host.
- Gli agenti in Java RMI sfruttano due peculiarità di Java:
  - *Serializzazione*, attraverso cui gli oggetti vengono trasmessi per valore (per deep copy).
  - *Polimorfismo*, che consente di utilizzare una sottoclasse B laddove è previsto l'utilizzo di una superclasse A (con B<A). Questo perchè la segnatura di una sottoclasse estende quella di una sua superclasse, sebbene l'implementazione dei metodi può essere completamente diversa.
- In Java RMI, un agente può essere passato dal client al server come parametro di un'invocazione remota, oppure può essere il valore tornato dal server al client attraverso un'invocazione remota.
- In Java RMI è possibile modellare per lo meno tre diverse forme di agenti:

- Mobile Agents
- Callbacks
- Mobile Servers.

Mobile Agents

- Un mobile agent è un oggetto **locale** serializzabile passato attraverso un'invocazione remota.
- Poichè è serializzabile esso verrà trasmesso per deep copy ed *andrà in esecuzione a destinazione*.
- Ovvero, presso il server, se l'agente è stato passato come parametro dal client, presso il client, se l'agente è stato passato come risultato dal server.
- L'utilizzo di agenti mobili richiede che entrambi, mittente e destinatario, siano a conoscenza di una classe astratta o di un'interfaccia dell'agente mobile.

- Questa condizione è simile a quella per cui un server ed un client devono condividere l'interfaccia remota del server remoto.

- Vediamo uno schema dell'entità coinvolte quando si lavora con agenti mobili. In questo esempio, l'agente viene passato dal client al server come parametro di un'invocazione remota. L'agente viene perciò eseguito presso il server:

```
public interface MobileAgent {
   public void act();
}
public interface Receiver extends Remote {
   public void receive(Agent agent);
}

public class ConcreteAgent implements MobileAgent,
                                      Serializable {
   public void act() { ...... }
}

public class Sender {
   public void send(Receiver rcvr) {
      rcvr.receive(new ConcreteAgent());} .....
}

public class ConcreteReceiver extends UnicastRemoteObject
                              implement Receiver {
   public void receive(MobileAgent agent) { agent.act(); }
   ....}
```

Callback

- Una callback è una sorta di “agente immobile”.
- Una callback è un server **remoto** esportato di cui una referenza remota viene passata come parametro, oppure ritornata come risultato di un’invocazione remota.
- Poichè un server remoto esportato è passato per referenza remota (si passa lo stub), l’agente vero e proprio “rimane dietro” (cioè non si muove) ed agisce alla sorgente. Vediamo un esempio di callback:

```
public interface Callback extends Remote {
  public void callback() throws RemoteException;
}
public interface Receiver extends Remote {
  public void receive(Callback agent) throws RemoteException;
}

public class CallbackClient extends UnicastRemoteObject
                            implements Callback {
  public void callback () { ....};
  public void main(String[] args) {
     Receiver rcv = ....lookup...
     rcv.receive(this);}
  .....
}
public class ServerReceiver extends UnicastRemoteObject
                            implements Receiver {
  public void receive(Callback agent) { agent.callback(); }
  ...........
}
```

- Notate che il client invoca un metodo remoto del server che a sua volta reinvoca il client, attraverso il metodo callback.

- Situazioni del genere sono auspicabili quando durante l'esecuzione del metodo remoto del server si rende necessario un'interazione col cliente per ricavare informazioni che il client può aver acquisito successivamente. Oppure informazioni in possesso del client che all'inizio dell'invocazione del metodo del server non si ritenevano necessarie.

- Le callback funzionano senza problemi ogni qual volta si lavora con oggetti locali.

- Se invece lavoriamo con server remoti, allora possono insorgere problemi di *stallo*.

- Ad esempio, consideriamo la variante dello schema precedente in cui l'agente callback contiene due metodi `run` e `callback` entrambi *synchronized*.

```
public interface Callback extends Remote {
  public void callback() throws RemoteException;
}
public interface Receiver extends Remote {
  public void receive(Callback agent) throws RemoteException;
}
public class CallbackClient extends UnicastRemoteObject
                            implements Callback {
  public synchronized void run(Receiver rcv) {
     ........
     rcv.receive(this);}
  public synchronized void callback () { ....};
  .....
}
public class ServerReceiver extends UnicastRemoteObject
                            implements Receiver {
  public void receive(Callback agent) { agent.callback(); }
  public void main (String[] args)
     Callback cb =...lookup...;
     cb.run(this);
}
```

- In una tale situazione, se `ServerReceiver` fosse un oggetto locale, allora il metodo `callback` verrebbe invocato nello stesso thread del metodo `run`, senza che ciò provochi problemi di deadlock.

- Ma se invece `ServerReceiver` è un server remoto, allora il sistema RMI invoca il metodo `callback` all'interno di un nuovo thread.

- Tale thread si blocca nel tentativo di entrare nel metodo `callback` visto che il thread originale è ancora in attesa all'interno del metodo synchronized `run`. In questo modo viene bloccato anche il thread originale sulla `run`. Quindi abbiamo uno stallo!

Mobile Server

- In questo caso l'agente in questione è un server **remoto serializzabile** ma non *non esportato* nel momento in cui viene passato.

- In tal caso sappiamo che il server viene passato per deep copy.

- A destinazione, ci sono due possibilità:

- Se il mobile server è un Unicast de-esportato, allora al momento della deserializzazione il server viene automaticamente esportato sulla medesima porta dov'era esportato originalmente e funziona come un server remoto presso l'host in cui è stato ricevuto.
- Se invece il mobile server è un qualche server remoto che non è stato esportato al momento della spedizione. Allora a destinazione bisognerà esportarlo esplicitamente.

- La struttura di un mobile server è, in un certo qual modo simile a quella vista per le callback eccetto per il fatto che l'oggetto `CallbackClient` non è esportato nel momento in cui si invoca il metodo `receive`.
- Vediamo un esempio di Mobile Server del secondo tipo.

```
public interface MobileServer extends Remote {
  public void remoteMethod() throws RemoteException;}
public interface Receiver extends Remote {
  public void receive(Client agent) throws RemoteException;}
public class ConcreteServer extends UnicastRemoteObject
                          implements MobileServer,Serializable{
  public void remoteMethod () { ....}; }

public class ClientSender {
  public void main(String[] args) {
     ConcreteServer ms = new ConcreteServe();
     .....
     unexport(ms,true);
     Receiver rcv = ...lookup..;
     rcv.receive(ms);}}

public class ServerReceiver extends UnicastRemoteObject
                          implements Receiver {
  public void receive(MobileServer agent){
     agent.remoteMethod();}
  ........}
```

- Si noti che l'invocazione `agent.remoteMethod()` avviene presso l'host in cui gira ServerReceiver ed in cui è stato esportato un server remoto `ConcreteServer`.

- In pratica, il client lancia un server remoto presso un'altra JVM (su un'altra macchina).

- Similmente un server può lanciare un'altro server presso il client.