UNIVERSITÀ DI PISA

Programmazione di reti
Corso B

29 Novembre 2016
Lezione 10

Contenuti

- Creare oggetti remoti (alternative)
- Funzionamento RMI
- *Dynamic class loading* e *code mobility*

Oggetti remoti

- Varie alternative per creare la classe remota:
  - Estendere classe `RemoteObject` e implementare interfaccia remota - come abbiamo visto la settimana scorsa
  - Estendere classe `RemoteServer` e implementare interfaccia remota
  - Estendere classe `UnicastRemoteObject` e implementare interfaccia remota
  - Implementare solo l'interfaccia remota, senza estendere nessuna classe

Oggetti remoti

- Estendere **`RemoteServer`** - classe di base per implementazioni dei *server*

- Metodi per *logging* e individuare il cliente

- Si deve usare **`UnicastRemoteObject.exportObject`**() per esportare l'oggetto e ottenere lo *stub*

Definizione della classe:

```
public class PisaStudentManager extends RemoteServer implements StudentManager {

   private static final long serialVersionUID = 1L;
   ArrayList<Remote> students;

   public PisaStudentManager() {
      this.students=new ArrayList<>();
   }
. . .
```

In main():

```
PisaStudentManager manager= new PisaStudentManager();
StudentManager managerStub=
                (StudentManager) UnicastRemoteObject.exportObject(manager, 0);
Registry registry= LocateRegistry.createRegistry(PORT);
registry.rebind(StudentManager.REMOTE_OBJECT_NAME, managerStub);
```

Oggetti remoti

- Estendere `UnicastRemoteObject` - classe di base per implementazioni degli oggetti remoti

- L’esportazione si fa automaticamente nel costruttore

- Si deve richiamare il costruttore della classe di base nella classe derivata

Definizione della classe:

```
public class PisaStudentManager extends UnicastRemoteObject implements
StudentManager {

   private static final long serialVersionUID = 1L;
   ArrayList<Remote> students;

   public PisaStudentManager() throws RemoteException {
      super();
      this.students=new ArrayList<>();
   }

   public PisaStudentManager(int port) throws RemoteException {
      super(port);
      this.students=new ArrayList<>();
   }
. . .
```

In main():

```
PisaStudentManager manager= new PisaStudentManager();
Registry registry= LocateRegistry.createRegistry(PORT);
registry.rebind(StudentManager.REMOTE_OBJECT_NAME, manager);
```

Oggetti remoti

- Non estendere nessuna classe RMI - per poter estendere un'altra classe

- Si devono implementare i metodi `equals`(), `toString`(), e `hashCode`() , se usati dalla applicazione

- Si deve usare `UnicastRemoteObject.exportObject`() per esportare l'oggetto e ottenere lo *stub*

Definizione della classe:

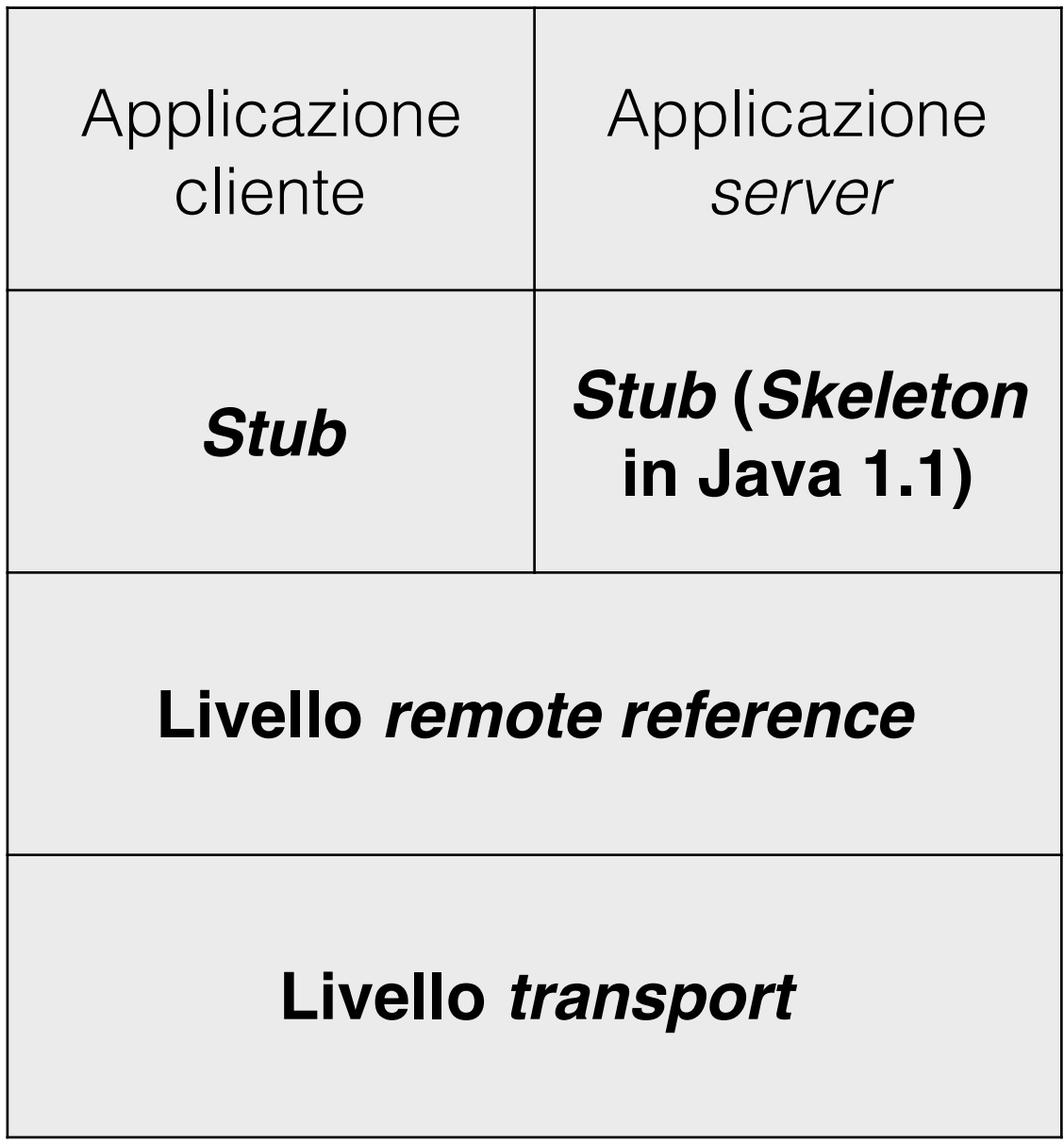
```
public class PisaStudentManager implements StudentManager {

   ArrayList<Remote> students;

   public PisaStudentManager() {
      this.students=new ArrayList<>();
   }
. . .
```

In main():

```
PisaStudentManager manager= new PisaStudentManager();
StudentManager managerStub=
                (StudentManager) UnicastRemoteObject.exportObject(manager, 0);
Registry registry= LocateRegistry.createRegistry(PORT);
registry.rebind(StudentManager.REMOTE_OBJECT_NAME, managerStub);
```

Systema RMI

- 3 livelli
  - *stub* - *proxy* per applicazione
  - *remote reference* - metodi per identificare ed accedere all'oggetto remoto, invocazione del metodo remoto, *marshalling* e *unmarshalling*
  - *transport* (RMI *runtime*)- gestisce la comunicazione di rete (i *socket*, le connessioni, etc.)



Livello *stub*

- Interfaccia tra applicazione e il sistema RMI
- Lato cliente:
    - Prende l'invocazione del metodo remoto e i parametri e li invia al livello *remote reference*, usando una `RemoteRef`
    - Aspetta il risultato e lo restituisce al cliente
- Lato *server*:
    - Riceve l'invocazione e i parametri dal livello *remote reference*, e invoca il metodo sull'oggetto reale.
    - Inoltra il risultato al livello *remote reference*

Stub statici/dinamici

- Fino a Java 1.4, le *stub* dovevano essere create manualmente: *stub* statici
  - usando *rmic* (e.g.: `rmic -keep PisaStudentManager`)
    - si ottiene una classe `*_Stub` con l'implementazione dello *stub* (e.g. `PisaStudentManager_Stub`)
- A partire da Java 1.5 non è più indicato usare *stub* statici: il *framework* RMI genera dei *stub* dinamici (*proxy*), automaticamente.

```
public final class PisaStudentManager_Stub
    extends java.rmi.server.RemoteStub
    implements StudentManager, java.rmi.Remote

{
    private static final long serialVersionUID = 2;

    private static java.lang.reflect.Method $method_searchByLastName_0;
    private static java.lang.reflect.Method $method_setStudentGrade_1;
```

```
abstract public class RemoteStub extends RemoteObject{
  . . . .
}

public abstract class RemoteObject implements Remote, java.io.Serializable {

    /** The object's remote reference. */
    transient protected RemoteRef ref;

. . . .
}
```

```
static {
    try {
        $method_searchByLastName_0 = StudentManager.class.getMethod("searchByLastName",
              new java.lang.Class[] {java.lang.String.class});

       $method_setStudentGrade_1 = StudentManager.class.getMethod("setStudentGrade",
              new java.lang.Class[] {int.class, double.class});

    } catch (java.lang.NoSuchMethodException e) {
         throw new java.lang.NoSuchMethodError(
       "stub class initialization failed");
    }
}
```

```
// implementation of searchByLastName(String)
    public java.util.ArrayList searchByLastName(java.lang.String $param_String_1)
   throws java.rmi.RemoteException
    {
   try {
       Object $result = ref.invoke(this, $method_searchByLastName_0,
            new java.lang.Object[] {$param_String_1}, -3361392531171944 55L);

       return ((java.util.ArrayList) $result);
} catch (java.lang.RuntimeException e) {
    throw e;
} catch (java.rmi.RemoteException e) {
    throw e;
} catch (java.lang.Exception e) {
    throw new java.rmi.UnexpectedException("undeclared checked exception", e);
}
 }
```

```
public boolean setStudentGrade(int $param_int_1, double $param_double_2)
    throws java.rmi.RemoteException
     {
    try {

        Object $result = ref.invoke(this, $method_setStudentGrade_1,
          new java.lang.Object[] {new java.lang.Integer($param_int_1), new
          java.lang.Double($param_double_2)},
          -2316187480729461103L);

        return ((java.lang.Boolean) $result).booleanValue();
    } catch (java.lang.RuntimeException e) {
        throw e;
    } catch (java.rmi.RemoteException e) {
        throw e;
    } catch (java.lang.Exception e) {
        throw new java.rmi.UnexpectedException("undeclared checked exception", e);
    }
     }
```

Livello *remote reference*

- Usato dal livello *stub* per invocare i metodi dell'oggetto remoto

- Rappresentato da un oggetto `RemoteRef`

- Il metodo `invoke()` comunica con il livello *transport* usando una connessione *stream-oriented* messa a disposizione dal livello *transport*

  - l'invocazione e i parametri sono scritti nell'*output stream* ed il risultato viene letto dall'i*nput stream* (*marshalling/unmarshalling*).

```
public class UnicastRef implements RemoteRef {

    protected LiveRef ref;
    […]

    public invoke(Remote obj, Method method, Object[] params, long opnum)
                    throws Exception
    {
         […]
         Connection conn = ref.getChannel().newConnection();
         RemoteCall call = null;
         […]
         call = new StreamRemoteCall(conn, ref.getObjID(), -1, opnum);
         ObjectOutput out = call.getOutputStream();
         marshalCustomCallData(out);
         Class<?>[] types = method.getParameterTypes();
         for (int i = 0; i < types.length; i++) {
                  marshalValue(types[i], params[i], out);
         }
         […]
         ObjectInput in = call.getInputStream();
         Object returnValue = unmarshalValue(rtype, in);
         […]
         return returnValue;
    }
    […]
}
```

`LiveRef` fa parte del livello *trasporto*

Passare parametri/risultati

- Si possono passare solo oggetti serializzabili o riferimenti ad oggetti remoti.
  - `RemoteObject` e sottoclassi implementano `Serializable`
    - Gli stub implementano `Serializable`
    - Quindi alla fine sono tutti oggetti serializzabili
- Un oggetto che implementa interfaccia `Remote` però non implementa `Serializable` non può essere passato direttamente - possiamo passare lo *stub* (che è generato automaticamente e implementa `Serializable`)

Passare parametri/risultati

- Serialisazione si fa usando `ObjectOutputStream/ObjectInputStream`

- Oggetti con lo stesso riferimento sul mittente avranno lo stesso riferimento sul destinatario

- La classe viene annotata con la location (*URL*) della definizione della classe (viene scaricata automaticamente da un *server* HTTP/FTP) - dettagli in seguito

Eccezioni

- Metodi remoti lanciano due tipi di eccezioni
  - `RemoteException`
    - Errore nella comunicazione, etc.
    - Gerarchia di eccezioni: e.g. `ConnectException, ConnectIOException, ExportException, MarshalException, ServerError, ServerException, StubNotFoundException, UnknownHostException, UnmarshalException`
    - Accesso alla *nested exception* - eccezioni lanciate lato *server* - `e.getCause()`
- Altre eccezioni definite dalla classe
  - vengono inviate sulla rete (usando la serializzazione) e lanciate lato client - quindi vengono gestite come le eccezioni locali

Livello *trasporto*

- Pacchetto `sun.rmi.transport`
- Lato cliente: Apre connessioni verso oggetti remoti, gestisce connessioni e *liveness*.
- Lato *server*: aspetta delle richieste di connessione, apre nuove connessioni, gestisce una tabella di oggetti remoti.
- Tutto usando un *endpoint* (indirizzo locale) e un *objectId* (*id* dell'oggetto remoto) - incapsulate in una `LiveRef` per ogni oggetto remoto
- Un oggetto `Transport` gestisce le operazioni - usa `java.rmi.server.RMISocketFactory` per creare dei *socket*
- Una connessione TCP per ogni cliente
- Almeno un *thread* per ogni cliente, i *socket* si riusano.

```
public class LiveRef implements Cloneable {

            private final Endpoint ep;
            private final ObjID id;
            private transient Channel ch;
             private final boolean isLocal;
```

Firewalls

- RMI pensato con server accessibile direttamente e clienti che possono usare *firewall*
- Le comunicazioni RMI usano HTTP *tunnelling*: se accesso diretto al *server* non è possibile, ogni richiesta del cliente è impacchettata in una richiesta HTTP POST inviata
  - direttamente al *host* e *port* del *server* (se possibile)
  - se non possibile, inviata sul *port* HTTP, dove un script la inoltra al *port* coretto.
- La `java.rmi.server.RMISocketFactory` predefinita implementa automaticamente questo meccanismo - server deve essere individuato usando *hostname* (e non l'IP)
- HTTP *tunnelling* è più lento dell'accesso diretto
- Non risolve problema del *server* dietro un *firewall* - ci sono però pacchetti java che permettono anche *server* dietro ai *firewall* (e.g. `com.rmiproxy.*`)

Distributed Garbage Collection

- Difficile sapere quanti riferimenti (*stub*) esistono per lo stesso oggetto

- Ogni cliente informa il *server* quando ottiene uno *stub* e quando non lo usa più. Il *server* conta i clienti. Quando non ci sono più *stub* usati, l'oggetto *server* viene sottoposto all'operazione GC.

- In realtà, una volta un oggetto viene esportato e incluso in un RMI `Registry`, non potrà mai essere rimosso con GC, visto che lo *stub* esiste nel `Registry`

Protocolli

- La comunicazione tra JVM necessaria per RMI si fa usando il protocollo *Java Remote Method Protocol* (JRMP)

- JRMP usa TCP/IP

- JRMP - valido esclusivamente per comunicazioni tra programmi Java

- Per mettere a disposizioni oggetti remoti ad altri linguaggi, si può usare il protocollo RMI-IIOP (*RMI over Internet InterORB Protocol*) - interfaccia tra RMI e CORBA (*Common Object Request Broker Architecture*)

Classe `Naming`

- Accesso più facile al *naming server* (RMI *Registry*) su un *host* e un *port*
- Metodi statici simili all'interfaccia `Registry`
- I nomi degli oggetti devono essere in formatto URL: //host:port/object

Classe Naming

```
static void   bind(String name, Remote obj)

static void   rebind(String name, Remote obj)

static void   unbind(String name)

static String[]  list(String name)

static Remote lookup(String name)
```

lato *server*

```
Registry registry= LocateRegistry.createRegistry(PORT);
registry.rebind(StudentManager.REMOTE_OBJECT_NAME, manager);
```

o

```
Registry registry= LocateRegistry.createRegistry(PORT);
Naming.rebind("//localhost:"+PORT+"/"+StudentManager.REMOTE_OBJECT_NAME,manager);
```

lato cliente

```
Registry registry = LocateRegistry.getRegistry(REGISTRY_HOST, REGISTRY_PORT);
StudentManager manager= (StudentManager)
registry.lookup(StudentManager.REMOTE_OBJECT_NAME);
```

O

```
StudentManager manager= (StudentManager)
Naming.lookup("//"+REGISTRY_HOST+":"+REGISTRY_PORT+"/"+StudentManager.REMOTE_OBJECT_
NAME);
```

Dynamic class loading

- Gli oggetti vengono inviati usando serializzazione - lo *stream* include il nome della classe dell'oggetto

- Una volta arrivati all destinazione, gli oggetti vengono ricreati usando *reflection*:

    - Una nuova istanza della classe viene creata in automatico

    - Gli attributi prendono i valori ricevuti

- La JVM ha bisogno di sapere la definizione della classe (*file .class*)

Dynamic class loading

- La classe viene prima cercata nel CLASSPATH della destinazione
- Se non trovata, RMI la scarica automaticamente, usando un *web server* (HTTP o FTP)
- Il mittente deve pubblicare le classi usate ad un indirizzo accessibile (denominato *codebase*)
- Due possibilità di informare il destinatario:
  - Publicare l'URL della *codebase* che il destinatario usa come parametro JVM
  - Durante la serializzazione si scrive non solo l'oggetto, ma anche URL della *codebase*
- Il lavoro è fato dal `RMIClassLoader`

Dynamic class loading

- Utile in due situazioni:
  - Uso dei *stub* statici (*dynamic stub loading*)
  - Uso del polimorfismo

Esempio stub statici

Dynamic class loading

- Usando dei *stub* statici:
  - Il *server* lancia *rmic* per creare lo *stub*. Ha due possibilità:
    - Include lo *stub* nella distribuzione dell'applicazione cliente
    - Si basa su *dynamic class loading*: mette lo *stub* in una *codebase* disponibile attraverso la rete

Dynamic class loading- stub statico

Server:

```
Alinas-MacBook-Air:RMIServerWeek10 Alina$ ls
PisaStudentManager.class        Student.java                    StudentManagerMain.class
PisaStudentManager.java         StudentManager.class            StudentManagerMain.java
Student.class                   StudentManager.java
Alinas-MacBook-Air:RMIServerWeek10 Alina$ rmic -keep PisaStudentManager
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
Alinas-MacBook-Air:RMIServerWeek10 Alina$ ls
PisaStudentManager.class        Student.class                   StudentManagerMain.class
PisaStudentManager.java         Student.java                    StudentManagerMain.java
PisaStudentManager_Stub.class   StudentManager.class
PisaStudentManager_Stub.java    StudentManager.java
Alinas-MacBook-Air:RMIServerWeek10 Alina$
```

Cliente:

```
Alinas-MacBook-Air:RMIClientWeek10 Alina$ ls
Student.class               StudentClient.class     StudentManager.class
Student.java                StudentClient.java      StudentManager.java
Alinas-MacBook-Air:RMIClientWeek10 Alina$
```

Dynamic class loading- stub statico

Avvio *server* e cliente (come prima):

```
Error in client: error unmarshalling return; nested exception is:
    java.lang.ClassNotFoundException: PisaStudentManager_Stub (no security manager:
RMI class loader disabled)
```

Stack trace:

```
java.rmi.UnmarshalException: error unmarshalling return; nested exception is:
    java.lang.ClassNotFoundException: PisaStudentManager_Stub (no security
manager: RMI class loader disabled)
    at sun.rmi.registry.RegistryImpl_Stub.lookup(Unknown Source)
    at StudentClient.main(StudentClient.java:16)
Caused by: java.lang.ClassNotFoundException: PisaStudentManager_Stub (no
security manager: RMI class loader disabled)
    at sun.rmi.server.LoaderHandler.loadClass(LoaderHandler.java:396)
    at sun.rmi.server.LoaderHandler.loadClass(LoaderHandler.java:186)
    at java.rmi.server.RMIClassLoader$2.loadClass(RMIClassLoader.java:637)
    at java.rmi.server.RMIClassLoader.loadClass(RMIClassLoader.java:264
```

Dynamic class loading

- Primo passo: istallare un `SecurityManager` - nel nostro caso solo sul cliente siccome il *server* non ha bisogno di scaricare classi

- **Necessario** se RMI deve scaricare delle classi dal cliente/*server*

- `SecurityManager` e accessibile tramite la classe `System`

```
if (System.getSecurityManager() == null) {
   System.setSecurityManager(new SecurityManager());
    }
```

```
public static void main(String[] args) {
      try {
         if (System.getSecurityManager() == null) {
               System.setSecurityManager(new SecurityManager());
           }

         Registry registry = LocateRegistry.
            getRegistry(REGISTRY_HOST, REGISTRY_PORT);
        StudentManager manager= (StudentManager)
            registry.lookup(StudentManager.REMOTE_OBJECT_NAME);
   […]
```

Dynamic class loading

Avvio *server* e cliente (come prima):

```
Exception in thread "main" java.security.AccessControlException: access denied
("java.net.SocketPermission" "127.0.0.1:2000" "connect,resolve")
    at java.security.AccessControlContext.checkPermission(AccessControlContext.java:
472)
    at java.security.AccessController.checkPermission(AccessController.java:884)
    at java.lang.SecurityManager.checkPermission(SecurityManager.java:549)
    at java.lang.SecurityManager.checkConnect(SecurityManager.java:1051)
    at java.net.Socket.connect(Socket.java:584)
```

Abbiamo bisogno di un *file* di *policy* e di informare la JVM del nome di questo *file*.

Dynamic class loading - security

File di *policy*:

StudentManagerM | StudentClient.j | permission

```
grant {// Allow everything
permission java.security.AllPermission;
};
```

Impostare la proprietà di sistema j`ava.security.policy` - 2 possibilità:

- Lanciare il programma con l'opzione JVM

`-Djava.security.policy=permission`

- Impostare il valore della proprietà usando `System.setProperty(name, value)`

Dynamic class loading

Avvio *server*. Avvio cliente usando `-Djava.security.policy=permission`

```
Error in client: error unmarshalling return; nested exception is:
    java.lang.ClassNotFoundException: PisaStudentManager_Stub
```

Stack trace:

```
java.rmi.UnmarshalException: error unmarshalling return; nested exception is:
    java.lang.ClassNotFoundException: PisaStudentManager_Stub
    at sun.rmi.registry.RegistryImpl_Stub.lookup(Unknown Source)
    at StudentClient.main(StudentClient.java:20)
Caused by: java.lang.ClassNotFoundException: PisaStudentManager_Stub
    at java.net.URLClassLoader.findClass(URLClassLoader.java:381)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
    at sun.rmi.server.LoaderHandler$Loader.loadClass(LoaderHandler.java:1207)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
```

Abbiamo bisogno di una *codebase*.

Codebase

- Impostazione con 2 modalità:
  - Opzione JVM `-Djava.rmi.server.codebase=[URL]`
  - `System.setProperty(“java.rmi.server.codebase”,”[URL]”)`

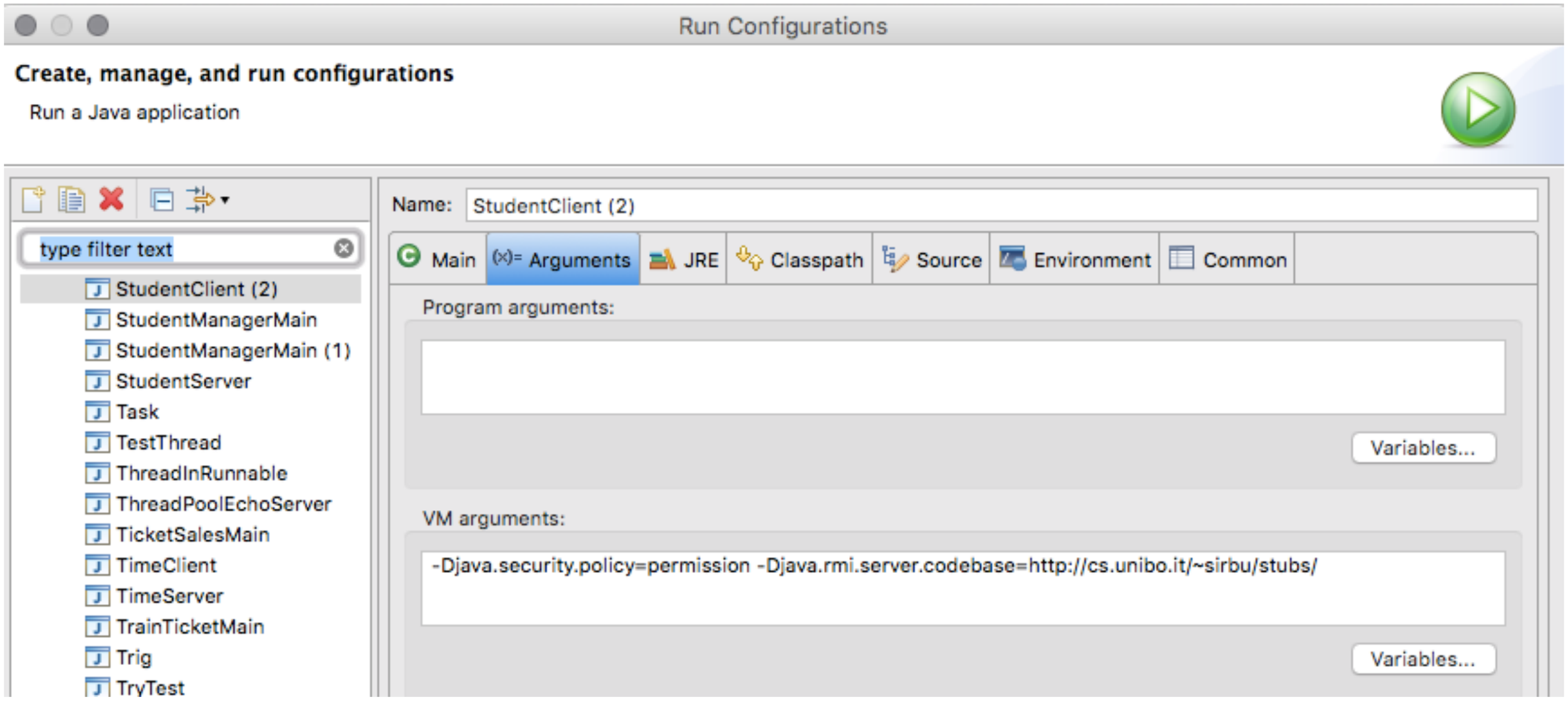
- Il *file .class* deve essere disponibile su un *server*

Dynamic class loading

Avvio *server*.
Avvio cliente usando `-Djava.security.policy=permission -Djava.rmi.server.codebase=http://cs.unibo.it/~sirbu/stubs/`

```
PisaStudentManager_Stub[UnicastRef [liveRef: [endpoint:[192.168.1.73:63330](remote),objID:
[6a5c9522:1548f8747ba:-7fff, 3194544050326330684]]]]
Received students:
Robert Brown living at 12 Dawson street. Student id 0. Current grade -1.0
Ann Brown living at 132 Buffallo street. Student id 3. Current grade -1.0
Received students:
Robert Brown living at 12 Dawson street. Student id 0. Current grade 30.0
Ann Brown living at 132 Buffallo street. Student id 3. Current grade -1.0
```

Dynamic class loading



Dynamic class loading

- Se non si usano i *stub* statici, non c'è bisogno della codebase

```
Alinas-MacBook-Air:RMIServerWeek10 Alina$ ls
PisaStudentManager.class        Student.java                    StudentManagerMain.class
PisaStudentManager.java         StudentManager.class            StudentManagerMain.java
Student.class                   StudentManager.java
Alinas-MacBook-Air:RMIServerWeek10 Alina$
```

- Lancio cliente con *-Djava.security.policy=permission*:

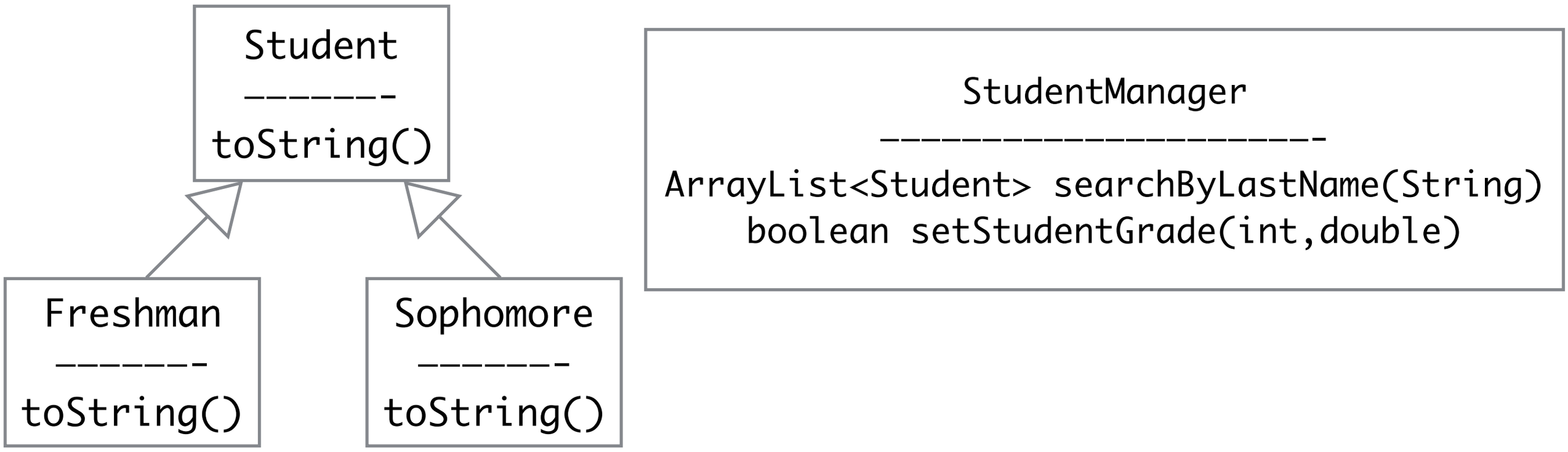
```
Proxy[StudentManager,RemoteObjectInvocationHandler[UnicastRef [liveRef: [endpoint:
[192.168.1.73:63685](remote),objID:[634e6051:1548fb73f45:-7fff,
-9090487137513352728]]]]]
Received students:
Robert Brown living at 12 Dawson street. Student id 0. Current grade -1.0
Ann Brown living at 132 Buffallo street. Student id 3. Current grade -1.0
Received students:
Robert Brown living at 12 Dawson street. Student id 0. Current grade 30.0
Ann Brown living at 132 Buffallo street. Student id 3. Current grade -1.0
```

Esempio polimorfismo

Dynamic class loading

- Usando *stub* dinamici
  - Gli stub vengono creati automaticamente quindi non devono essere inclusi nella *codebase*.
  - Se usiamo polimorfismo, può succedere che il *cliente/server* non ha a disposizione tutte le classi necessarie
  - Si devono includere queste classi nella *codebase*

Dynamic class loading- polimorfismo



- Il *server* contiene 2 tipi di studenti.
- Possono essere restituiti in `searchByLastName`()
- Il cliente non conosce queste 2 classi, quindi li deve scaricare da qualche parte.

```
public class Student implements Serializable{

   private static final long serialVersionUID = 2L;

   protected String fname;

   protected String lname;
   protected String streetAddress;
   protected int addressNo;
   protected double grade;
   protected int studentId;
   protected Lock lock;


[……]
```

```
public class Freshman extends Student {

   private static final long serialVersionUID = 1L;

   public Freshman(String fname, String lname,
            String street, int no, int id) {
      super(fname, lname, street, no, id);
   }

   public String toString(){
      return "Freshman - "+super.toString();
   }

   public Student copy(){
      Freshman result=
            new Freshman(this.fname,this.lname,this.streetAddress,
            this.addressNo,this.studentId);
      this.lock.lock();
      result.setGrade(this.grade);
      this.lock.unlock();
      return result;
   }
}
```

```
public class Sophomore extends Student{
   private static final long serialVersionUID = 1L;

   public Sophomore(String fname, String lname,
         String street, int no, int id) {
      super(fname, lname, street, no, id);
   }

   public String toString(){
      return "Sophomore - "+super.toString();
   }

   public Student copy(){
      Sophomore result=
            new Sophomore(this.fname,this.lname,this.streetAddress,
            this.addressNo,this.studentId);
      this.lock.lock();
      result.setGrade(this.grade);
      this.lock.unlock();
      return result;
   }
}
```

```
public class StudentManagerMain {

    public static int PORT=2000;

    public static void main(String[] args) {

        try {
            //create manager and add data
            PisaStudentManager manager= new PisaStudentManager();
            manager.addStudent(new Freshman("Robert","Brown","Dawson",12,0));
            manager.addStudent(new Freshman("Michael","Reds","Pearse",40,1));
            manager.addStudent(new Sophomore("Joanna","Moore","Collins",62,2));
            manager.addStudent(new Sophomore("Ann","Brown","Buffallo",132,3));

            //register to RMI registry
            Registry registry= LocateRegistry.createRegistry(PORT);
            registry.rebind(StudentManager.REMOTE_OBJECT_NAME, manager);


            System.out.println("Finished server setup."+manager.getRef());
        } catch (RemoteException e) {
            System.out.println("Error setting up RMI server: "+e.getMessage());
        }
    }

}
```

Server

```
Alinas-MacBook-Air:RMIServerWeek10 Alina$ ls
Freshman.class                  Sophomore.class                 StudentManager.class
Freshman.java                   Sophomore.java                  StudentManager.java
PisaStudentManager.class        Student.class                   StudentManagerMain.class
PisaStudentManager.java         Student.java                    StudentManagerMain.java
```

Cliente

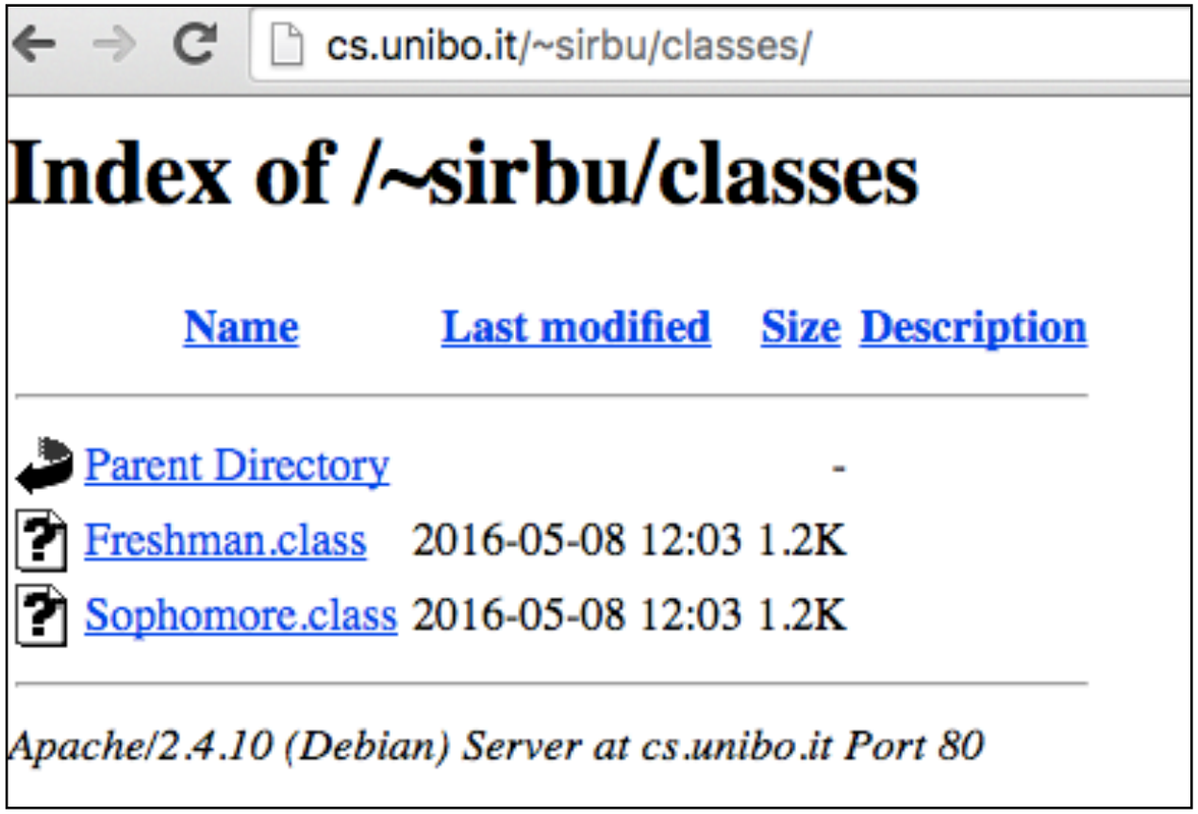
```
Alinas-MacBook-Air:RMIClientWeek10 Alina$ ls
Student.class               StudentClient.class     StudentManager.class
Student.java                StudentClient.java      StudentManager.java
Alinas-MacBook-Air:RMIClientWeek10 Alina$
```

Lancio cliente con `-Djava.security.policy=permission`:

```
Error in client: error unmarshalling return; nested exception is:
    java.lang.ClassNotFoundException: Freshman
java.rmi.UnmarshalException: error unmarshalling return; nested exception is:
    java.lang.ClassNotFoundException: Freshman
    at sun.rmi.server.UnicastRef.invoke(UnicastRef.java:198)
    at
java.rmi.server.RemoteObjectInvocationHandler.invokeRemoteMethod(RemoteObjectIn
vocationHandler.java:227)
```

Dynamic class loading - polimorfismo

- Mettere a disposizione le classi addizionali.



Dynamic class loading - polimorfismo

- Informare il cliente da dove può scaricare le classi - due possibilità:
  - Lanciare il cliente indicando il *codebase* - come prima per lo *stub*
  - Includere il *codebase* nel *server* -
    - durante la serializzazione gli oggetti vengono annotati con l'URL del *codebase* del *server*
    - il cliente scarica automaticamente le classi dall'URL ricevuto con gli oggetti
    - il cliente deve avere l'opzione `java.rmi.server.useCodebaseOnly=false` (altrimenti usa solo il codebase locale e non quello inviato dal *server*)
- La seconda opzione offre più flessibilità: la *codebase* può cambiare sul *server* senza dover informare i clienti

Dynamic class loading - polimorfismo

- I possibilità
  - lancio *server* senza parametri
  - lancio cliente con

```
-Djava.security.policy=permission
-Djava.rmi.server.codebase= http://cs.unibo.it/
~sirbu/classes/
```

```
Proxy[StudentManager,RemoteObjectInvocationHandler[UnicastRef [liveRef: [endpoint:
[192.168.1.73:63818](remote),objID:[-98ef99:1548fdb609a:-7fff, 805403467835634289 2]]]]]
Received students:
Freshman - Robert Brown living at 12 Dawson street. Student id 0. Current grade -1.0
Sophomore - Ann Brown living at 132 Buffallo street. Student id 3. Current grade -1.0
Received students:
Freshman - Robert Brown living at 12 Dawson street. Student id 0. Current grade 30.0
Sophomore - Ann Brown living at 132 Buffallo street. Student id 3. Current grade -1.0
```

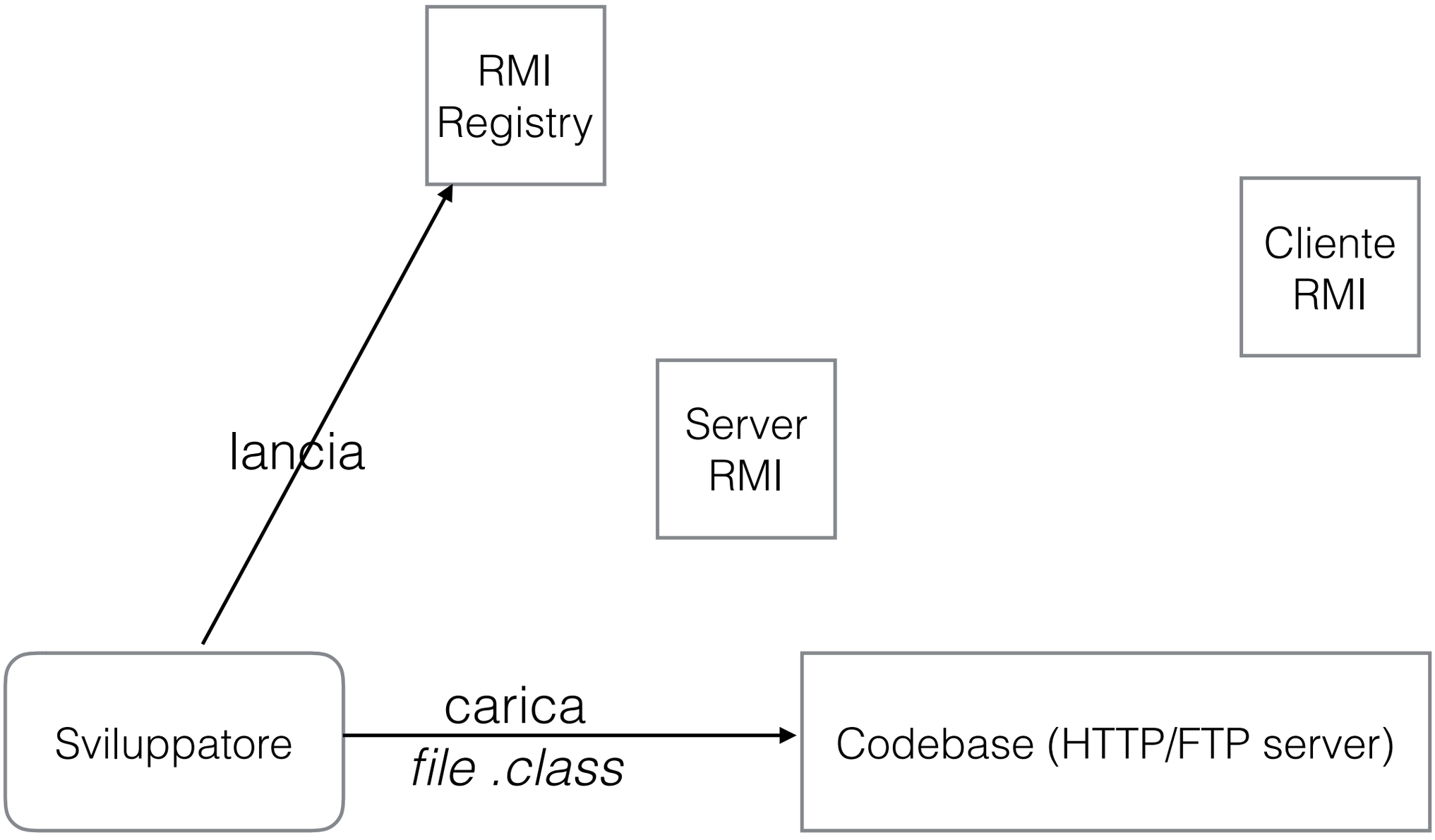
Dynamic class loading - polimorfismo

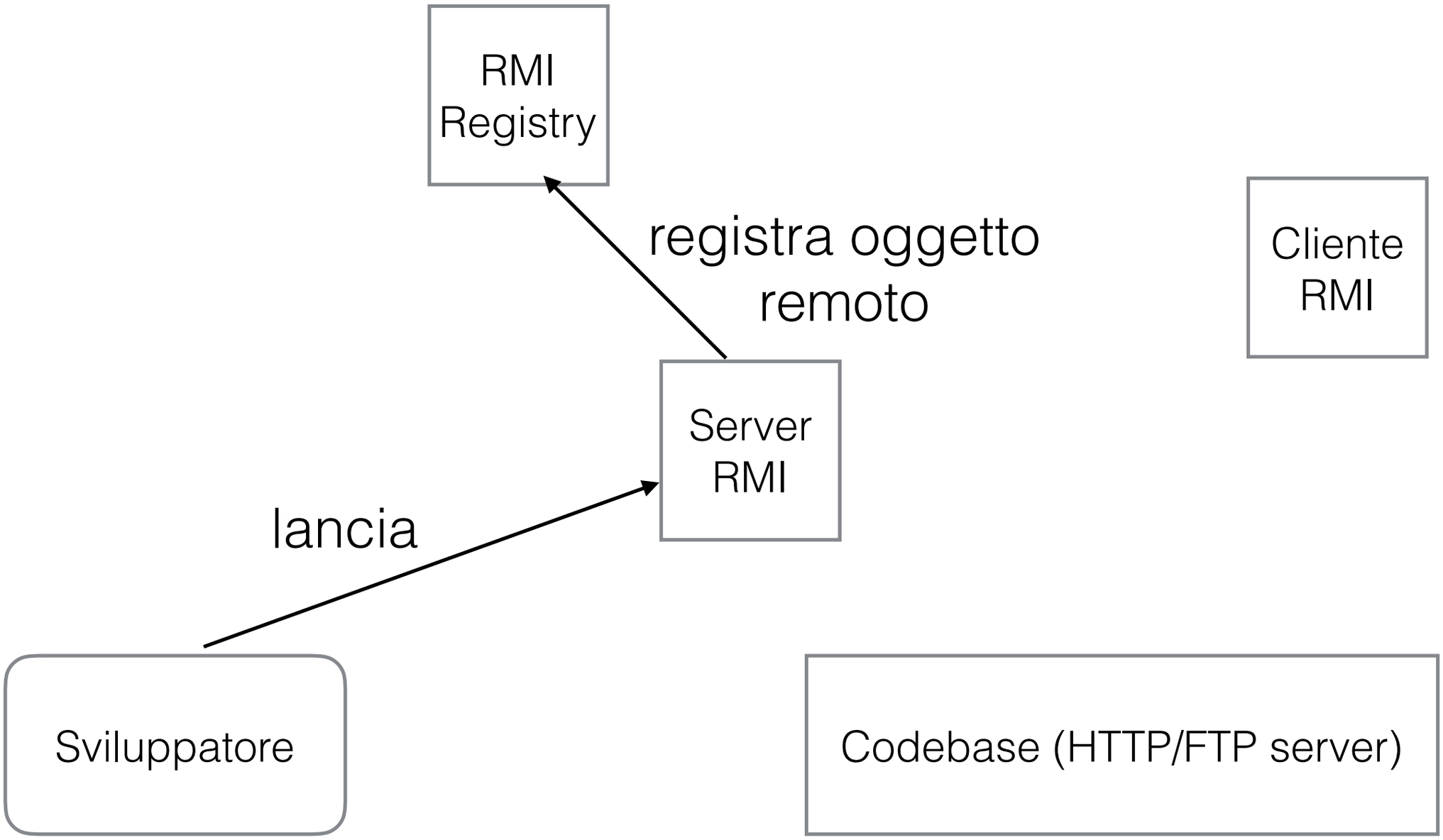
- II possibilità

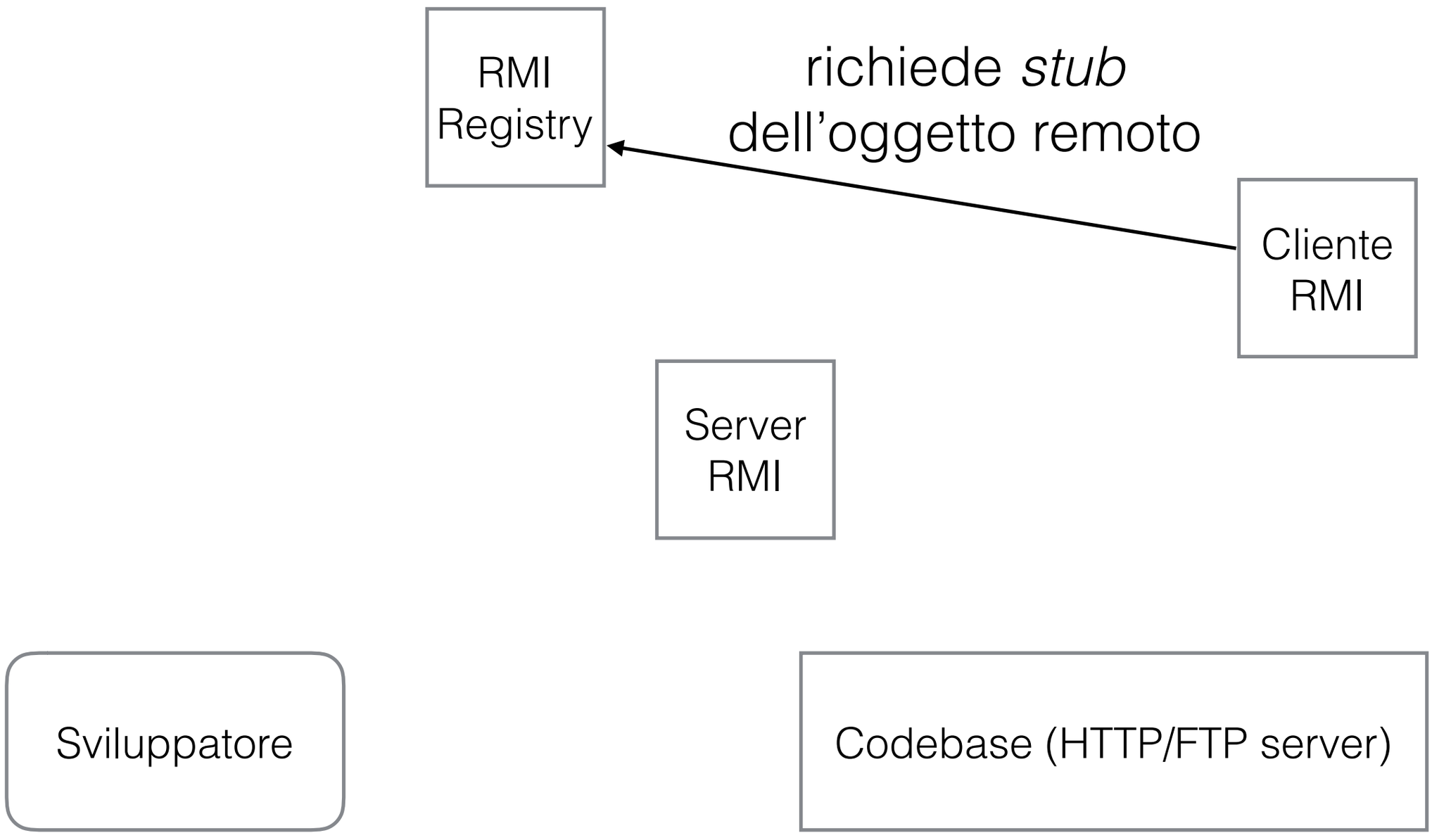
  - lancio *server* con `-Djava.rmi.server.codebase=http://cs.unibo.it/~sirbu/classes/`

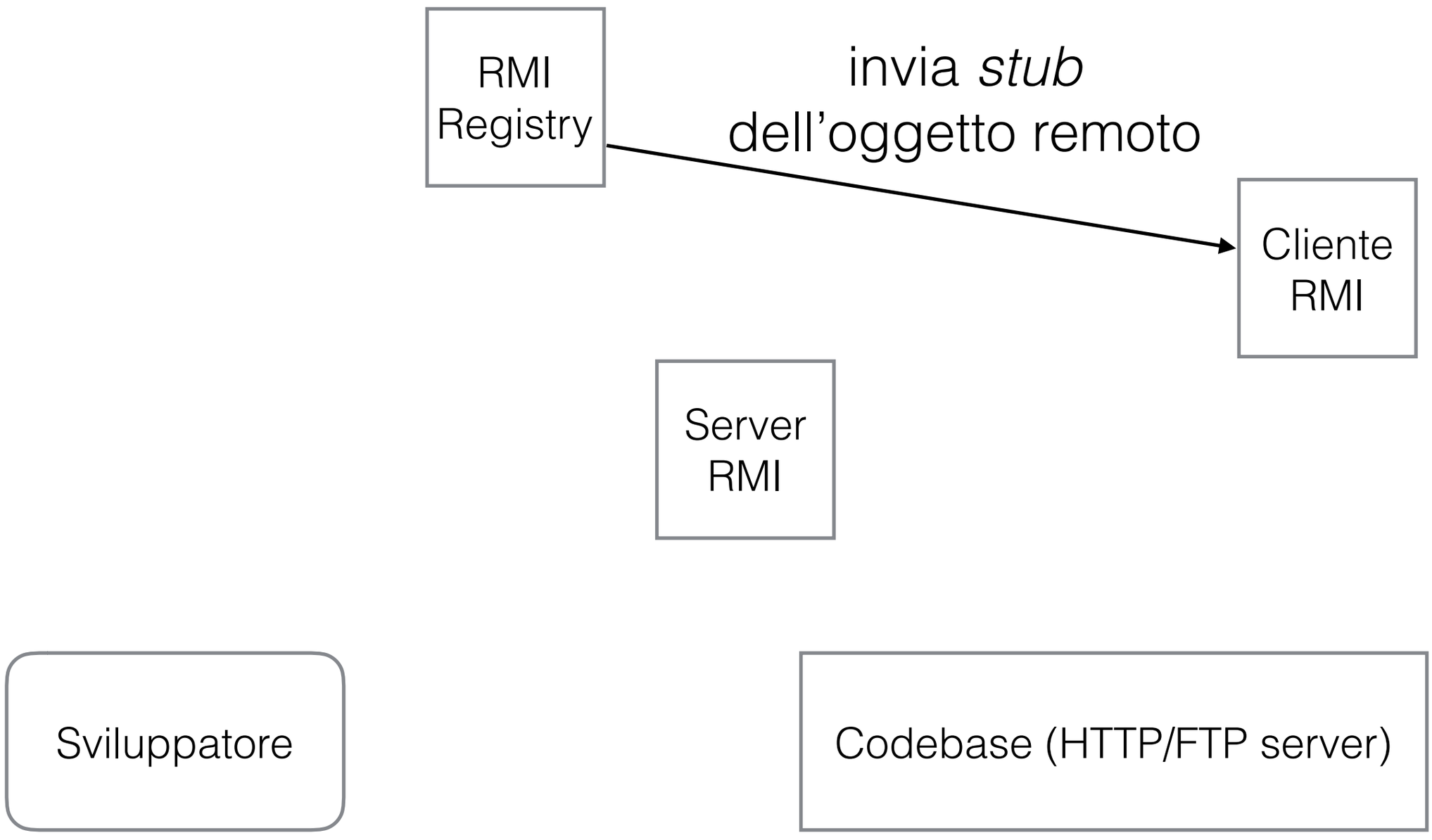
    - lancio cliente con `-Djava.security.policy=permission -D java.rmi.server.useCodebaseOnly=false`

```
Proxy[StudentManager,RemoteObjectInvocationHandler[UnicastRef [liveRef: [endpoint:
[192.168.1.73:63818](remote),objID:[-98ef99:1548fdb609a:-7fff, 805403467835634289 2]]]]]
Received students:
Freshman - Robert Brown living at 12 Dawson street. Student id 0. Current grade -1.0
Sophomore - Ann Brown living at 132 Buffallo street. Student id 3. Current grade -1.0
Received students:
Freshman - Robert Brown living at 12 Dawson street. Student id 0. Current grade 30.0
Sophomore - Ann Brown living at 132 Buffallo street. Student id 3. Current grade -1.0
```


RMI
Registry
Server
RMI
Cliente
RMI
lancia
Sviluppatore
carica
file .class
Codebase (HTTP/FTP server)


RMI
Registry
registra oggetto
remoto
Cliente
RMI
Server
RMI
lancia
Sviluppatore
Codebase (HTTP/FTP server)


RMI
Registry
richiede *stub*
dell'oggetto remoto
Cliente
RMI
Server
RMI
Sviluppatore
Codebase (HTTP/FTP server)


RMI
Registry
invia *stub*
dell'oggetto remoto
Cliente
RMI
Server
RMI
Sviluppatore
Codebase (HTTP/FTP server)

RMI Registry

Cliente RMI

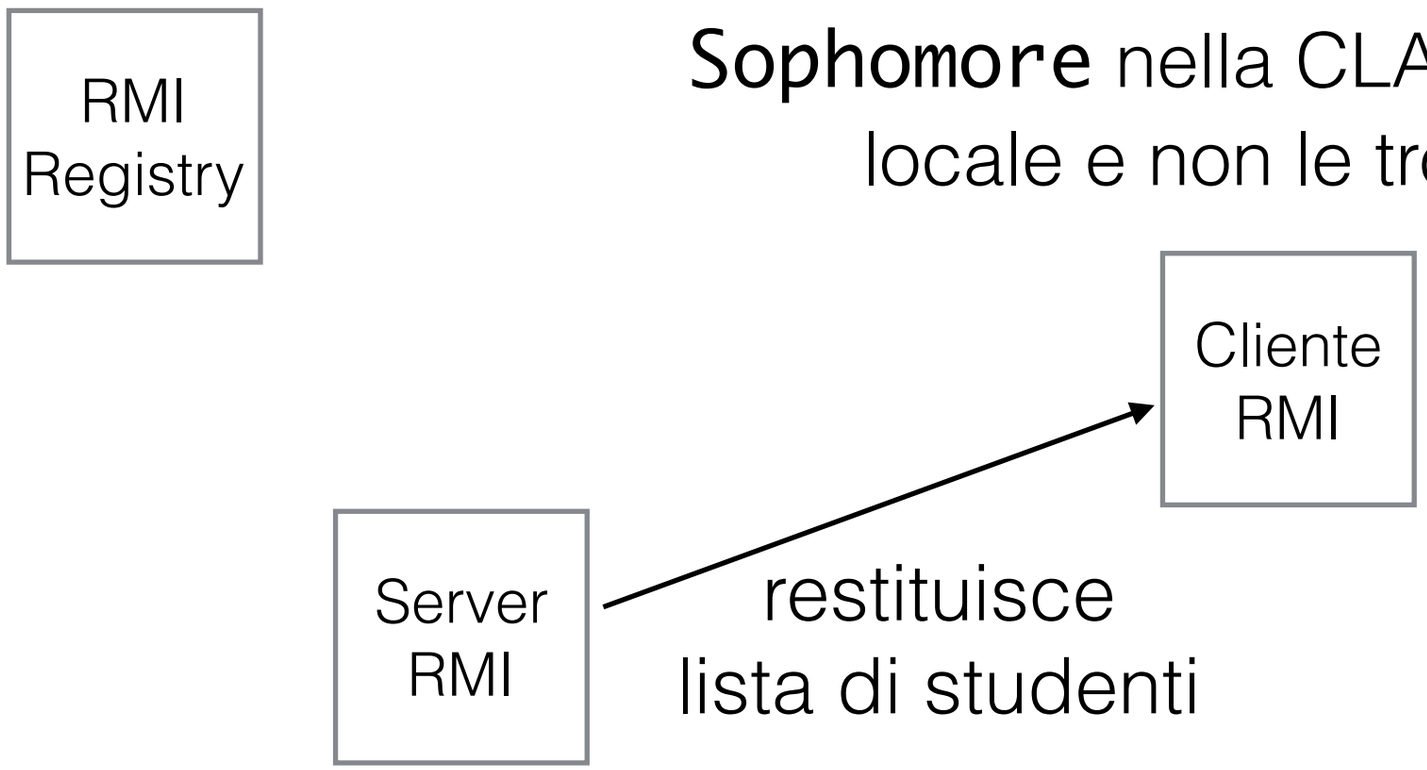
Server RMI

richiama metodo `searchByLastName()`

Sviluppatore

Codebase (HTTP/FTP server)

Cliente riceve lista di studenti, cerca definizioni di `Freshman` e `Sophomore` nella CLASSPATH locale e non le trova



Sviluppatore

Codebase (HTTP/FTP server)

RMI Registry

Server RMI

Cliente RMI

Invia richiesta per `Freshman.class` e `Sophomore.class`

Sviluppatore

Codebase (HTTP/FTP server)

Cliente può istanziare gli oggetti di tipo `Freshman` e `Sophomore`