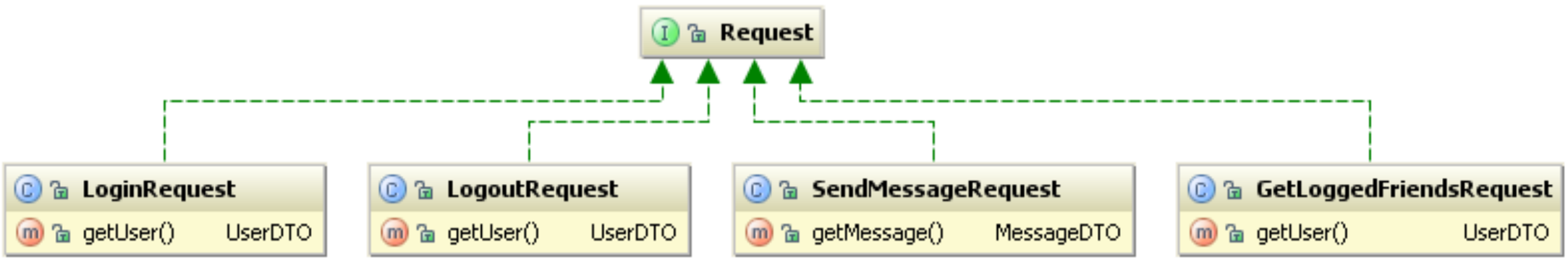
Medii de proiectare și programare

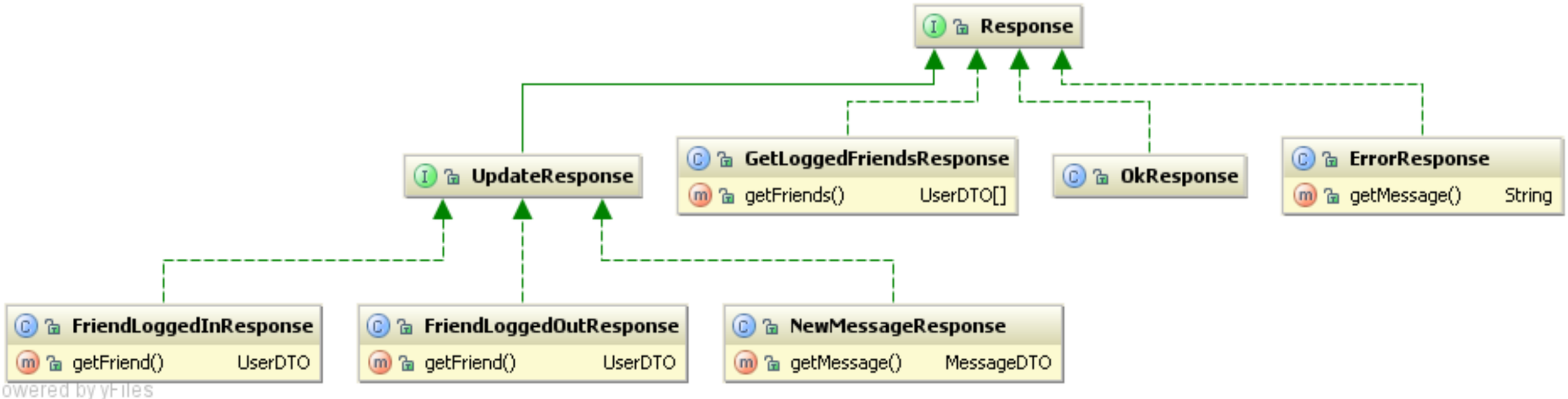
2019-2020
Curs 6

Conținut curs 6

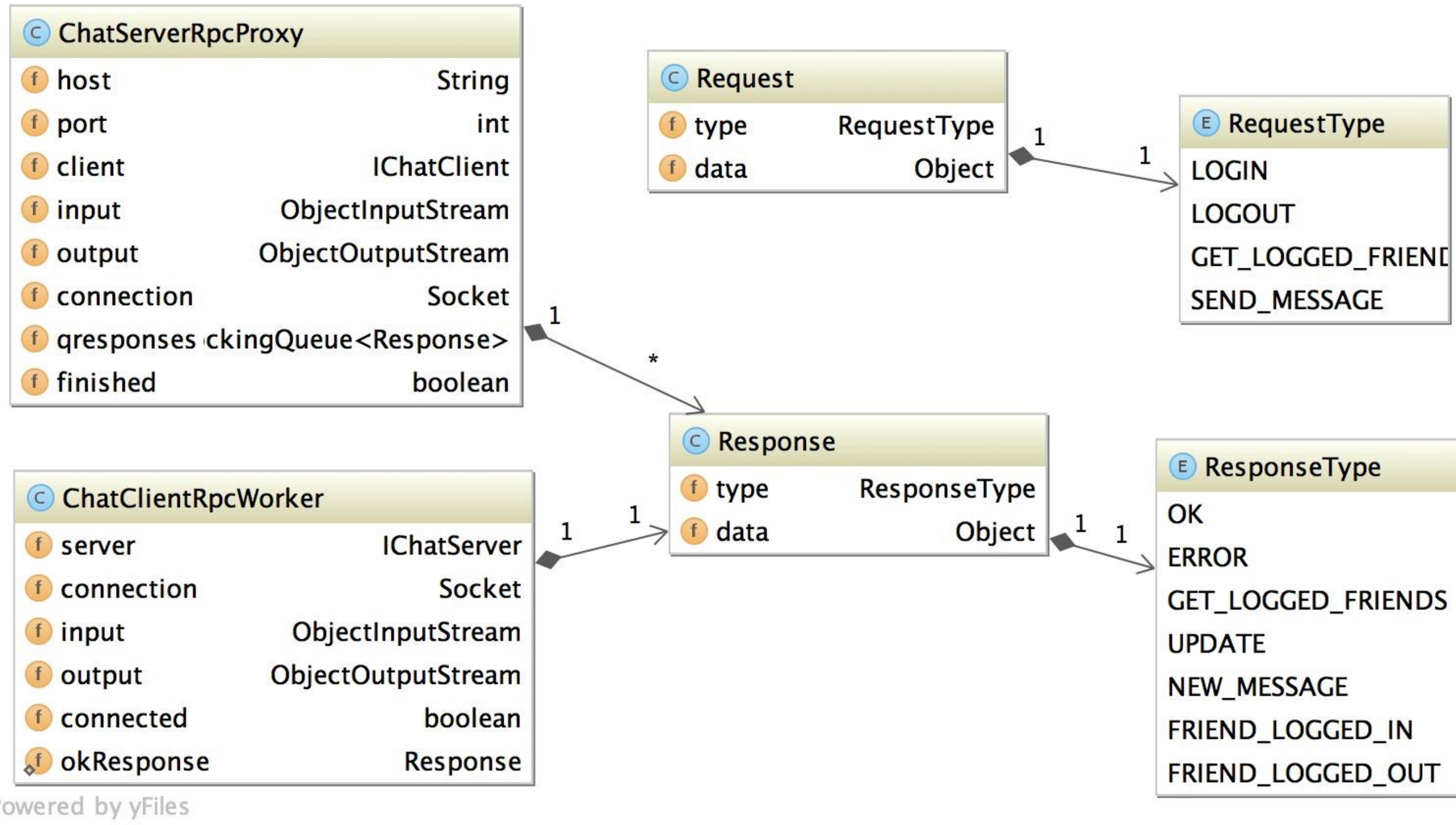
- Exemplu Mini-Chat (networking)
- Remote Procedure Call
- RMI
- Spring Remoting (cursul 7)
- .NET Remoting (cursul 7)

Mini-chat Object Protocol





Mini-chat Rpc Protocol



Remote Procedure Call

- Apelul procedurilor la distanță (eng. *Remote procedure call - RPC*) este o tehnologie de comunicare între procese care permite unei aplicații să inițieze execuția unei subrutine sau a unei proceduri în alt spațiu de adrese, fără ca programatorul să scrie explicit codul corespunzător interacțiunii dintre procese.
- Programatorul scrie aproximativ același cod indiferent dacă apelează o rutină locală (din același proces) sau una la distanță (din alt proces, în alt spațiu de adrese).
- Când se folosește paradigma orientată pe obiecte RPC - apeluri la distanță sau apelul metodelor la distanță.
- RPC este adesea folosit pentru implementarea aplicațiilor client-server.
- Un apel la distanță este inițiat de client prin trimiterea unei cereri către un server la distanță pentru execuția unei proceduri cu parametrii dați. Serverul trimite un răspuns clientului, iar aplicația își continuă execuția.
- Cât timp serverul procesează cererea clientului, execuția clientului este blocată (așteaptă până când serverul a terminat procesarea cererii).

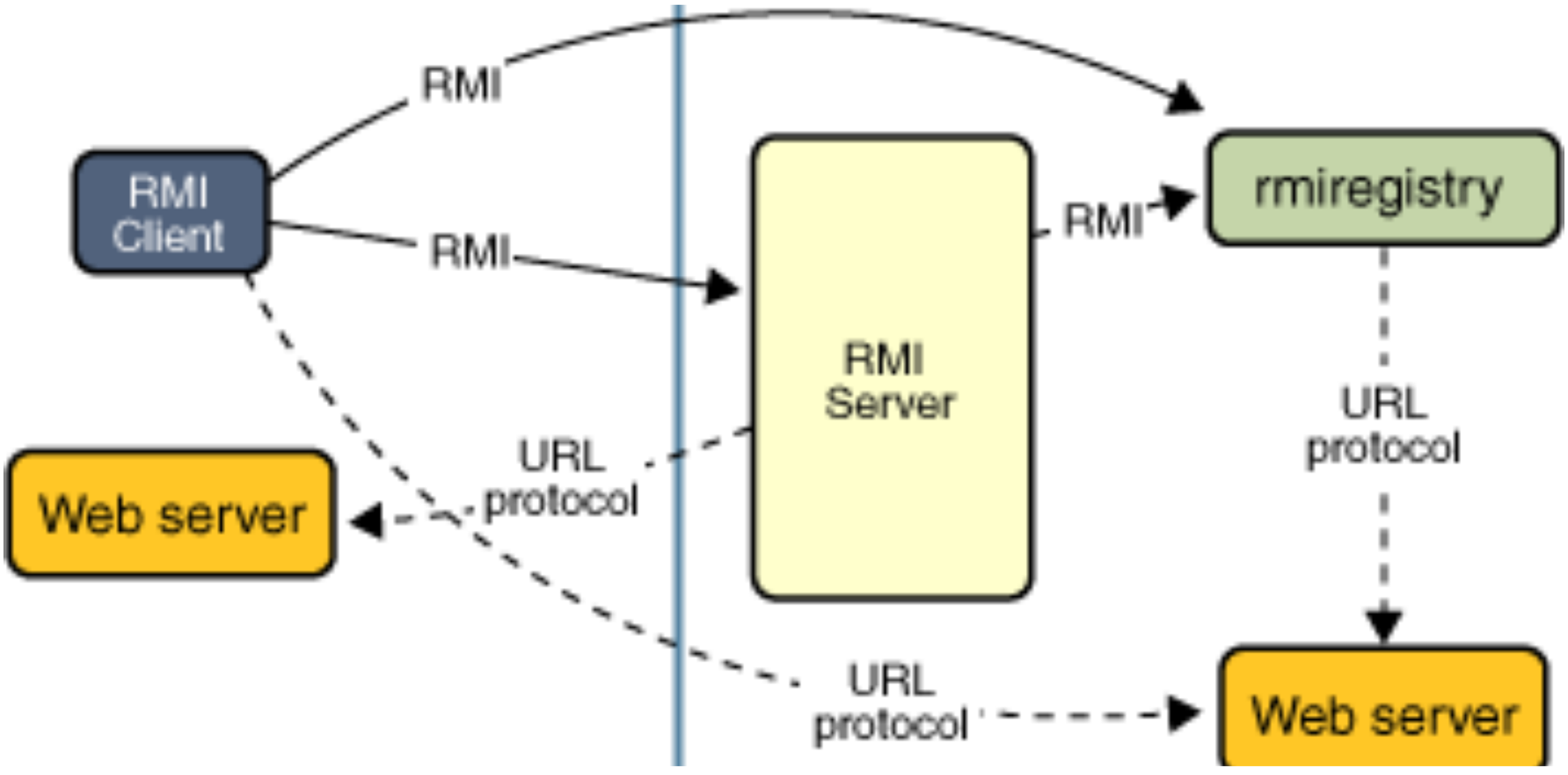
Java RMI

- Tehnologia Java Remote Method Invocation (RMI) permite unui obiect dintr-o mașină virtuală Java să apeleze metodele unui obiect dintr-o altă mașina virtuală Java.
- Aplicațiile RMI conțin cel puțin doua programe separate: *server* și *client*.
  - Programul server creează obiecte ce pot fi apelate la distanță, face referințele către aceste obiecte disponibile clienților și așteaptă ca clienții să apeleze metode asupra obiectelor.
  - Programul client obține o referință către unul sau mai multe obiecte la distanță, iar apoi apelează metode asupra acestuia (acestora).
- RMI furnizează mecanismul prin care serverul și clientul comunică și își transmit informații.

Java RMI

- *Localizarea obiectelor la distanță (remote)*. Mecanisme pentru obținerea referințelor către obiecte remote:
    - O aplicație înregistrează obiectele remote folosind *rmiregistry* - instrument oferit de RMI, bazat pe nume.
    - O aplicație poate transmite/primi referințe către obiecte remote, în urma altor apeluri la distanță.
- *Comunicarea cu obiecte remote.* Detaliile comunicării între obiecte remote și clienți sunt rezolvate de RMI.
    - Apelurile la distanță sunt asemănătoare cu apelurile de metode Java obișnuite.
- *Încărcarea claselor corespunzătoare obiectelor transmise ca și parametrii/ rezultat*. RMI permite transmiterea unor obiecte a căror clase nu erau disponbile la pornirea aplicației.

Java RMI


RMI
RMI Client
RMI
RMI
rmiregistry
RMI Server
URL protocol
URL protocol
Web server
URL protocol
Web server

RMI

- *Obiecte remote*: obiecte care conțin metode ce pot fi apelate din diferite instanțe de mașini virtuale. Un obiect devine remote dacă clasa corespunzătoare implementează o interfață *remote*, cu următoarele caracteristici:
  - O interfață remote moștenește interfața **`java.rmi.Remote`**.
  - Fiecare metodă din interfață specifică excepția java.rmi.RemoteException în signatura sa (poate să arunce și alte excepții).

```
import java.rmi.*;
public interface IServer extends Remote{
 public ResultObject method(Param1 p1, Param2 p2,…) throws
   RemoteException;
}
```

- *Obiecte locale* (obiecte non-remote): obiecte folosite în apelurile la distanță (parametrii sau rezultat).
  - Trebuie să fie serializabile.

Arhitectura unei aplicații bazate pe RMI

- *Biblioteca comună* (interfețe remote și clasele corespunzătoare parametrilor).
  - O interfață remote specifică metodele remote care pot fi apelate de clienți, parametrii acestora, tipurile parametrilor, rezultatul returnat, tipul rezultatului.
- *Server* (Implementarea interfețelor remote).
  - Obiectele remote trebuie să implementeze una sau mai multe interfețe remote.
  - Pot să implementeze alte interfețe a căror metode pot fi apelate doar local.
- *Client(Clienții)* (Implementarea clienților).
  - Clienții care folosesc obiecte remote pot fi implementați oricând după definirea interfețelor remote și “publicarea” acestora.

Pachetul `java.rmi`

- Interfața **`Remote`**: folosită pentru identificarea obiectelor a căror metode pot fi apelate dintr-o altă mașina virtuală. Orice obiect remote trebuie să implementeze direct sau indirect această interfață. Doar metodele specificate într-o interfață remote pot fi apelate la distanță.
- Clasa **`Naming`**: conține metode statice pentru păstrarea și obținerea referințelor către obiecte remote în/dintr-un registru specific.
  - Fiecare metodă a clasei **`Naming`** are un parametru de tip String cu următorul format:

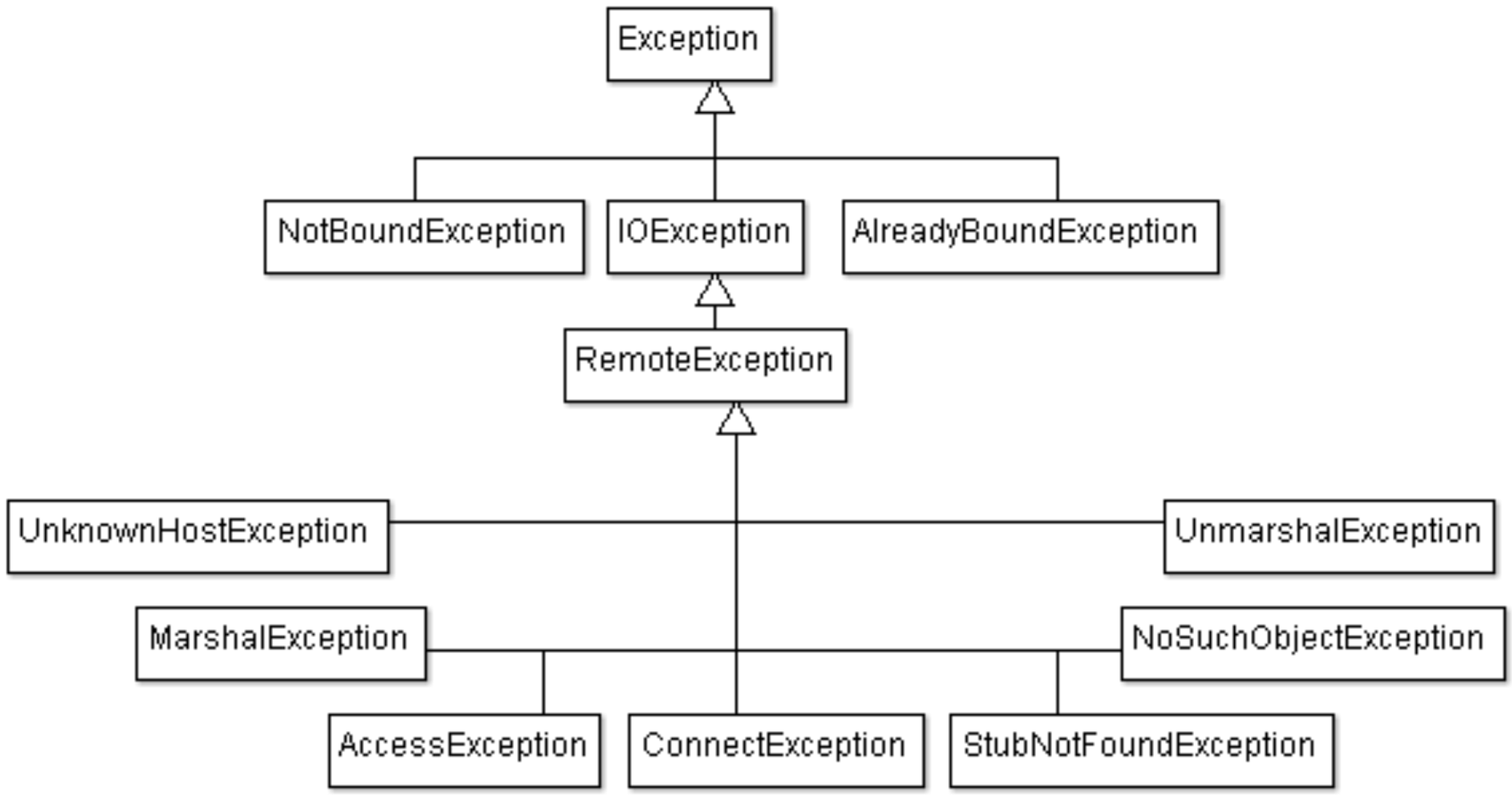
`//host:port/name`

- **`bind(String name, Remote obj)`** asociaza numele de un obiect remote.
- **`lookup(String name):Remote`** returnează referința/proxy/stub către un obiect remote asociat cu numele specificat.
- **`rebind(String name, Remote obj)`** reactualizează asocierea dintre numele specificat și obiectul remote.
- **`unbind(String name)`** distruge/șterge asocierea dintre numele specificat și obiectul remote corespunzător.

java.rmi package

- Clasa **`RMISecurityManager -`** subclasă a clasei **`SecurityManager`** folosita de aplicațiile RMI care descarcă cod. Class loader-ul RMI nu va descărca orice clase de la locații remote dacă nu a fost setat un obiect de tip SecurityManager.

```
System.setSecurityManager(new SecurityManager());
```



Pachetul java.rmi.registry

- Un registru este un obiect remote care păstrează asocieri între obiecte remote și un nume specific.
- Un server își înregistrează obiectele remote folosind registrul, iar clienții le vor căuta folosind același registru.
- Când un obiect (clientul) vrea să apeleze o metodă a unui obiect remote trebuie întâi să caute obiectul remote folosind numele. Registrul va returna clientului o referință către obiectul remote, iar clientul va folosi această referință pentru apelul metodei la distanță.
- Interfața **`Registry`**: este o interfață *remote* pentru un obiect remote de tip registru care conține metode pentru păstrarea și regăsirea referințelor către obiecte remote.
  - **`bind`**, **`lookup`**, **`rebind`**, **`unbind`**
- Clasa **`LocateRegistry`**
  - folosită pentru obținerea referinței către un registru aflat la o anumită adresă (inclusiv localhost)
  - folosită pentru crearea unui obiect de tip registru care acceptă cereri de la clienti la portul specificat.

Exemplu simplu RMI

- Definirea interfeței remote:

```
package rmi.services;

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


public interface ITransformer extends Remote {
    String transform(String text) throws RemoteException;
}
```

Exemplu RMI

- Implementarea interfeței remote:

```
package rmi.server;
import rmi.services.ITransformer;
import java.rmi.RemoteException;
import java.util.Date;

public class TransformerImpl implements ITransformer{
    public String transform(String text) throws RemoteException {
        System.out.println("Method called "+text);
        return text.toUpperCase()+(new Date());
    }
}

//for dynamically stub creation
//public class TransformerImpl extends UnicastRemoteObject
                        implements ITransformer
```

Exemplu RMI

- Crearea arhivei jar care conține clasele și interfețele comune serverului și clienților:

```
jar cvf services.jar rmi/services/ITransformer.class
```

- Compilarea clasei care implementează interfața remote folosind fișierul creat anterior.
- (Versiuni <1.5) Crearea server stub-ului folosind `rmic` :

```
rmic -classpath services.jar rmi.server.TransformerImpl
```

Se crea un fisier numit `rmi.server.TransformerImpl_stub.class`

- Crearea fișierului pentru permisiuni:

```
grant {
    permission java.security.AllPermission;
    permission java.net.SocketPermission "127.0.0.1:1024-",
   "connect,resolve";
};
```

Exemplu RMI

• Crearea clasei care pornește serverul:

```
public class StartServer {
    public static void main(String[] args) {
       //optional, setare SecurityManager
       if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }
        try {
            String name = "Transformer";
            ITransformer engine = new TransformerImpl();
             //daca TransformerImpl nu extinde UnicastRemoteObject
            ITransformer stub =(ITransformer)
   UnicastRemoteObject.exportObject(engine, 0);
            Registry registry = LocateRegistry.getRegistry();
            registry.rebind(name, stub);
            System.out.println("Transformer bound");
        } catch (Exception e) {
            System.err.println("Transformer exception:"+e);
        }
  }
}
```

Exemplu RMI

- Pornirea serverului:
  - Start RMI registry:
    - **`Start rmiregistry`** (Windows)
    - **`Rmiregistry`** (Linux, Mac OS)
  - Copierea claselor comune într-un loc ce poate fi accesat de clienți public:
    - http://scs.ubbcluj.ro/~xyz/rmi
    - c:/temp/rmi/transformer/server/
  - Pornirea serverului folosind comanda:

```
java -cp .;<path to>services.jar
-Djava.rmi.server.codebase=file:/c:/temp/rmi/transformer/server/
-Djava.rmi.server.hostname=localhost
-Djava.security.policy=server.policy
 StartServer
```

Exemplu RMI

- Pornirea clientului:

```
public class StartClient {
  public static void main(String args[]) {
    //optional, setare SecurityManager
    if (System.getSecurityManager() == null)
            System.setSecurityManager(new SecurityManager());
    try {
        String name = "Transformer";
        Registry registry = LocateRegistry.getRegistry("localhost");
        ITransformer comp = (ITransformer) registry.lookup(name);
        String text="Ana are mere.";
        String resp=comp.transform(text);
        System.out.println("Result: "+resp);
    } catch (RemoteException e) {…}
     catch (NotBoundException e) {…}
     }
}
```

Exemplu RMI

- Crearea fișierului de permisiuni pentru client:

```
grant {
    permission java.security.AllPermission;
    permission java.net.SocketPermission "127.0.0.1:1024-",
  "connect,resolve";
};
```

- Execuția clientului

```
java -cp .;<path to>services.jar
-Djava.security.policy=client.policy StartClient
```

Exemplu RMI

- Implementare MiniChat folosind RMI