Medii de proiectare și programare

2017-2018
Curs 4

Conținut curs 4

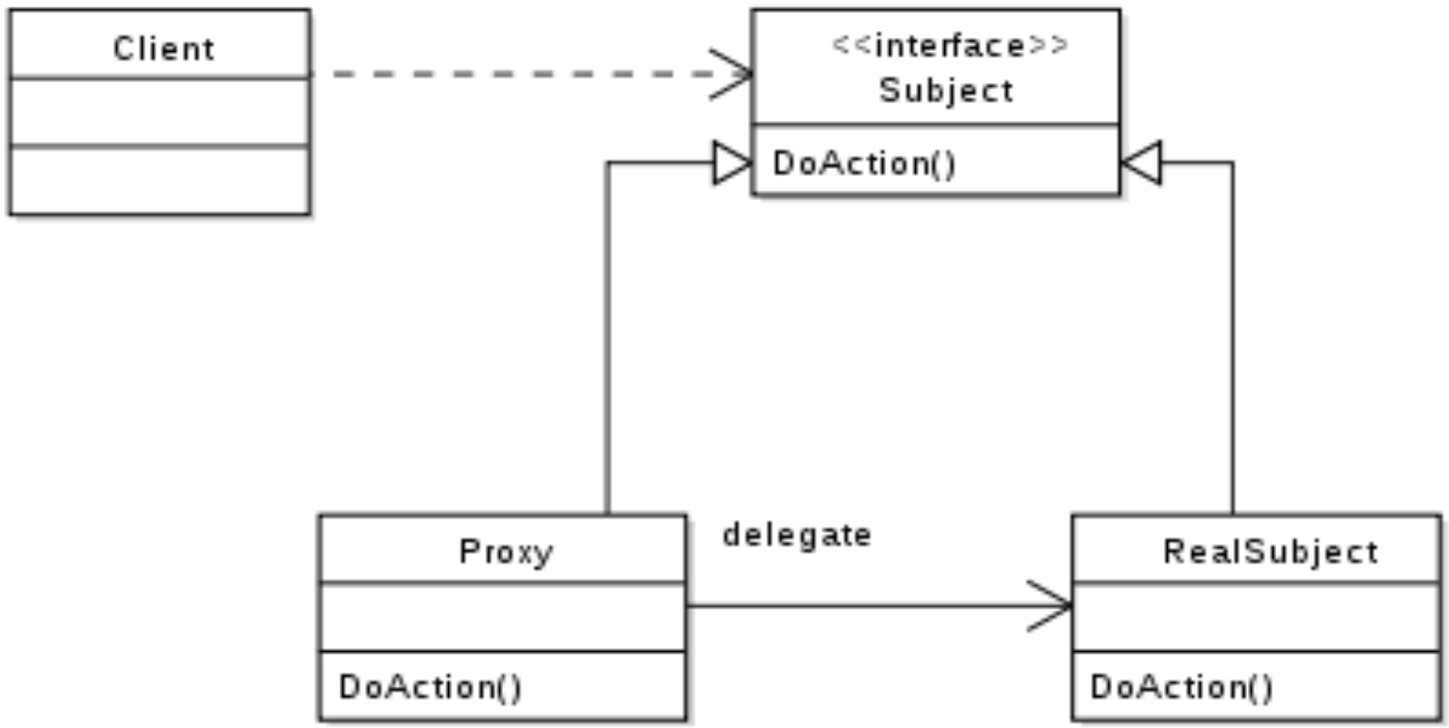
- Proiectarea unei aplicații client-server
- Șablonul Proxy
- Șablonul Data Transfer Object
    - Networking și threading în Java

Mini-chat

- Proiectați și implementați o aplicație client-server pentru un mini-chat având următoarele funcționalități:
  - *Login.* După autentificarea cu succes, o nouă fereastră se deschide în care sunt afișați toți prietenii *online* ai utilizatorului și o listă cu mesajele trimise/primite de utilizator. De asemenea, toți prietenii online văd în lista lor că utilizatorul este *online*.
  - *Trimiterea unui mesaj*. Un utilizator poate trimite un mesaj text unui prieten care este online. După trimiterea mesajului, prietenul vede automat mesajul în fereastra lui.
  - *Logout.* Toți prietenii online ai utilizatorului văd în lista lor că utilizatorul nu mai este *online*.

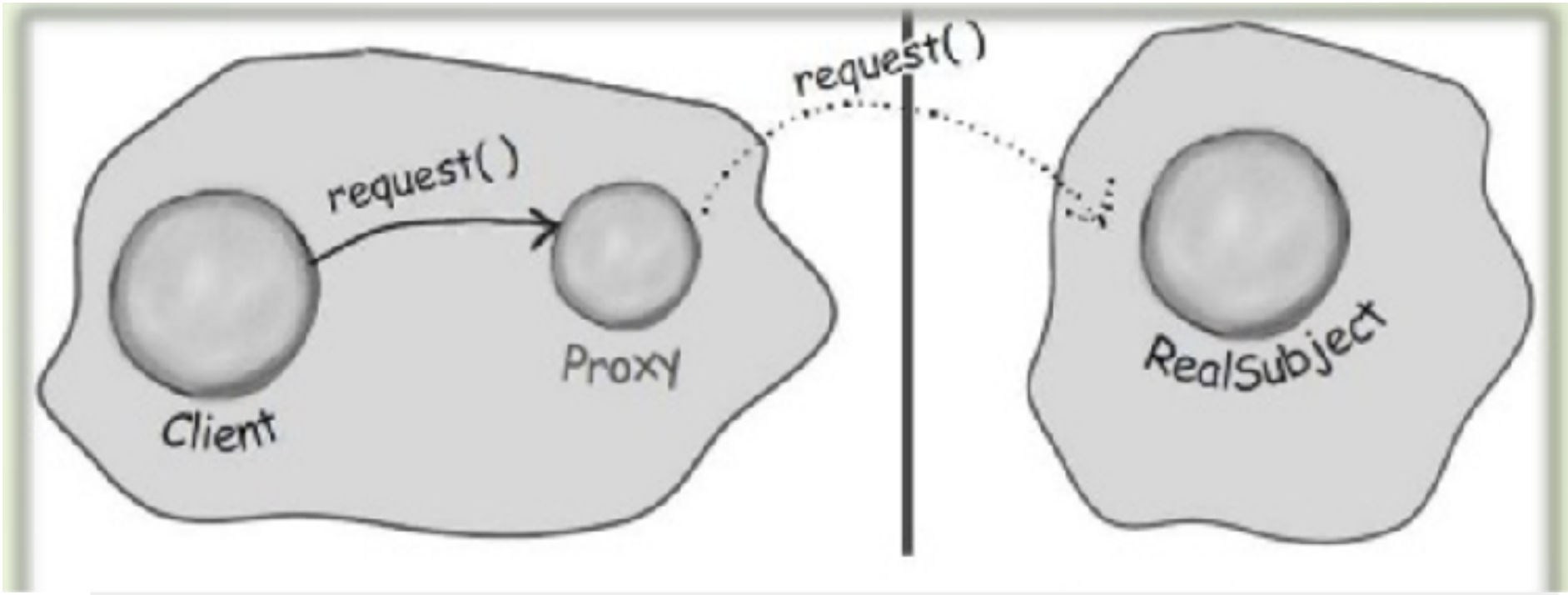
Șablonul Proxy

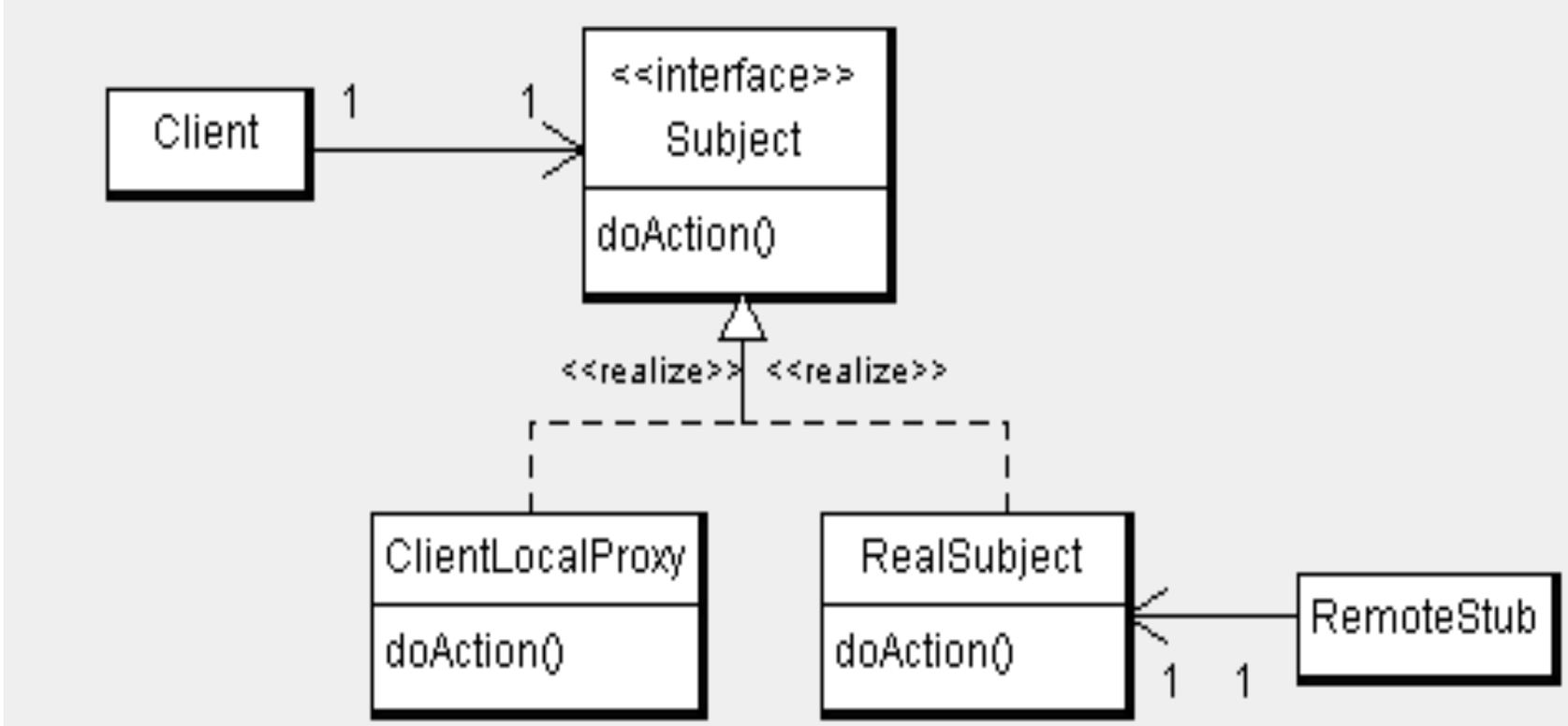
- Asigură pentru un obiect existent, un surogat sau un înlocuitor în scopul controlării accesului la acesta.
- Înlocuitorul poate fi:
  - Proxy la distanță (eng. *remote proxy*) - obiect în alt spatiu de adresă,
  - Proxy virtual (eng. *virtual proxy*) - un obiect mare din memorie,
  - Proxy de protecție - controlează accesul la obiectul original,
  - etc.



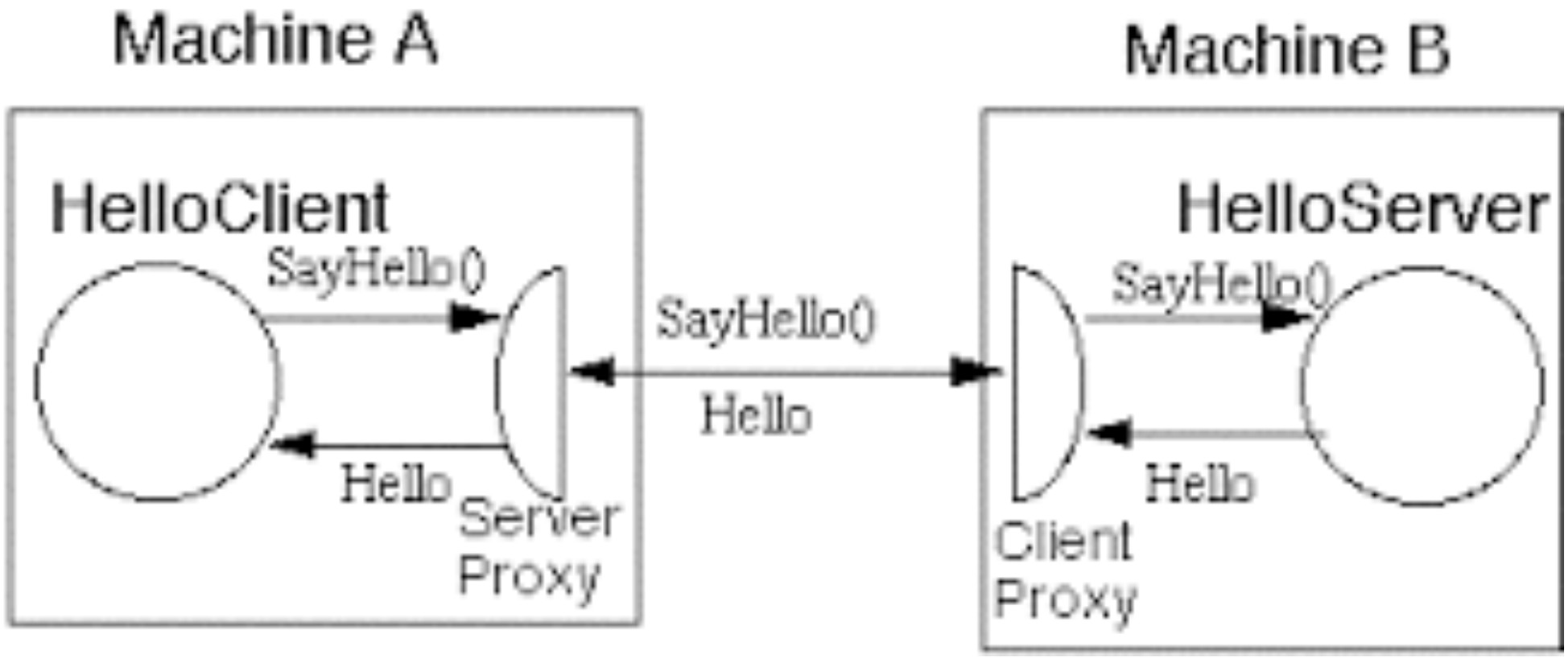
Șablonul *Remote Proxy*

- Remote Proxy oferă un înlocuitor local pentru un obiect aflat în alt spațiu de adresă/memorie.
- Înlocuitorul este responsabil cu codificarea unei cereri, a parametrilor și trimiterea lor către obiectul real aflat într-un spațiu de adresă diferit.
- Clientul cererii crede că comunică cu obiectul real, dar este un proxy între ei.
- Proxy-ul transformă cererile clientului în cereri la distanță, obține rezultatul cererii și îl transmite clientului.

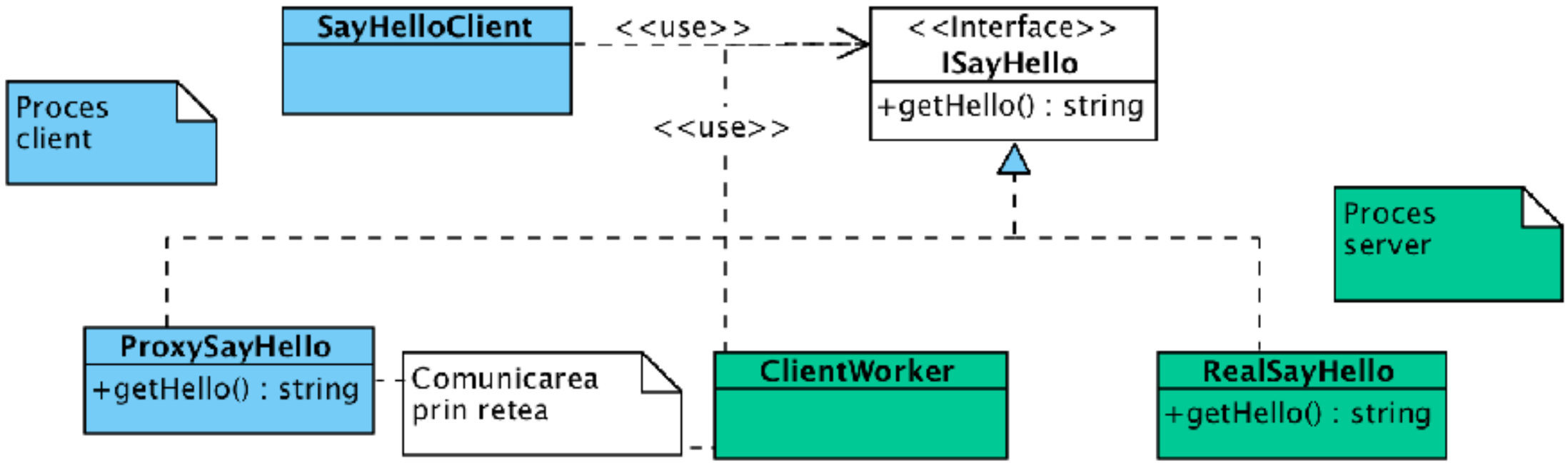

request()
request()
Client
Proxy
RealSubject


Client
1
1
<<interface>>
Subject
doAction()
<<realize>>
<<realize>>
ClientLocalProxy
doAction()
RealSubject
doAction()
RemoteStub
1
1

Șablonul Remote Proxy



Șablonul Remote Proxy

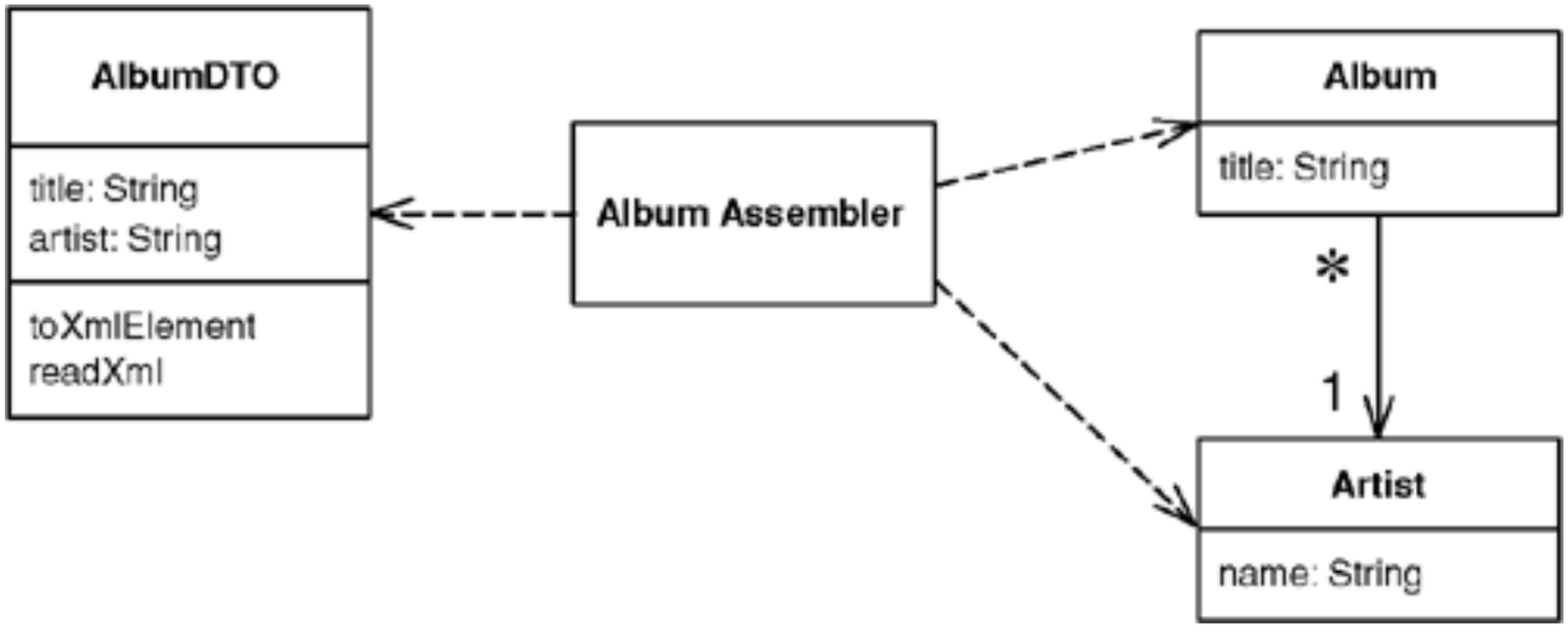


Șablonul Data Transfer Object

- Un DTO este un obiect care conține informația ce trebuie transmisă între unul sau mai multe procese pentru a reduce numărul de apeluri (dintre procese).
- Fiecare apel de metodă remote este costisitor, de aceea numărul de apeluri ar trebui redus și mai multă informație ar trebui transmisă la un apel.
- O soluție posibilă este de a folosi mai mulți parametri:
  - dificil de programat
  - în unele cazuri nu este posibil (ex., în Java o metodă poate returna o singură valoare).
- Soluția: crearea unui DTO (*Data Transfer Object*) care păstrează toată informația necesară unui apel. De obicei obiectul este serializabil (binar, XML, etc.) pentru a putea fi transmis prin rețea.
- Un alt obiect este responsabil cu conversia datelor din model într-un DTO și invers.

Șablonul Data Transfer Object

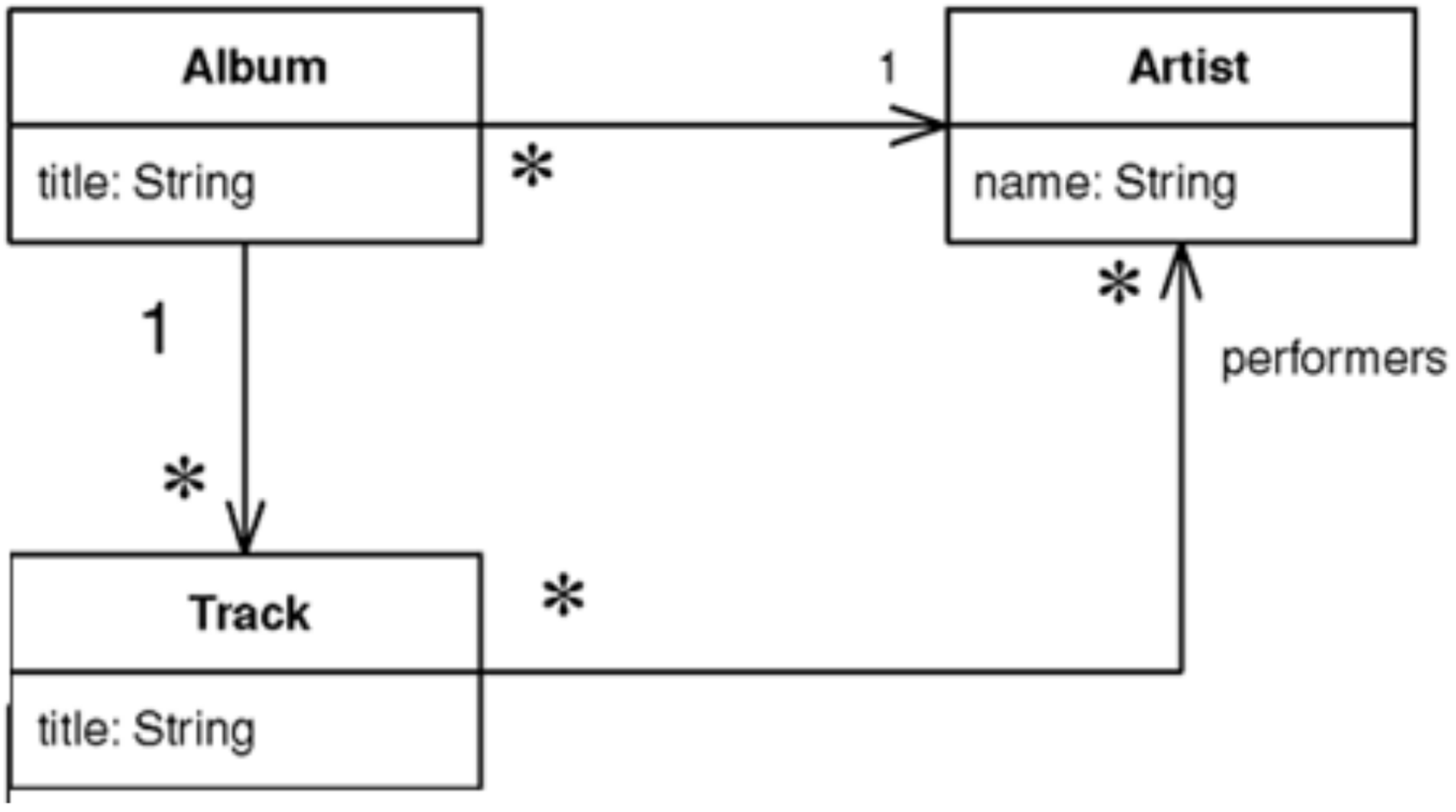
- Un DTO conține, de obicei, multe atribute și metode de tip get/set pentru acestea.
- Când un obiect remote are nevoie de date, cere DTO-ul corespunzător. DTO poate să conțină mai multă informație decât este necesar la acel apel, dar ar trebui să conțină toată informația de care va avea nevoie obiectul remote o perioadă.
- Un DTO conține de obicei informație provenind de la mai multe obiecte din model.

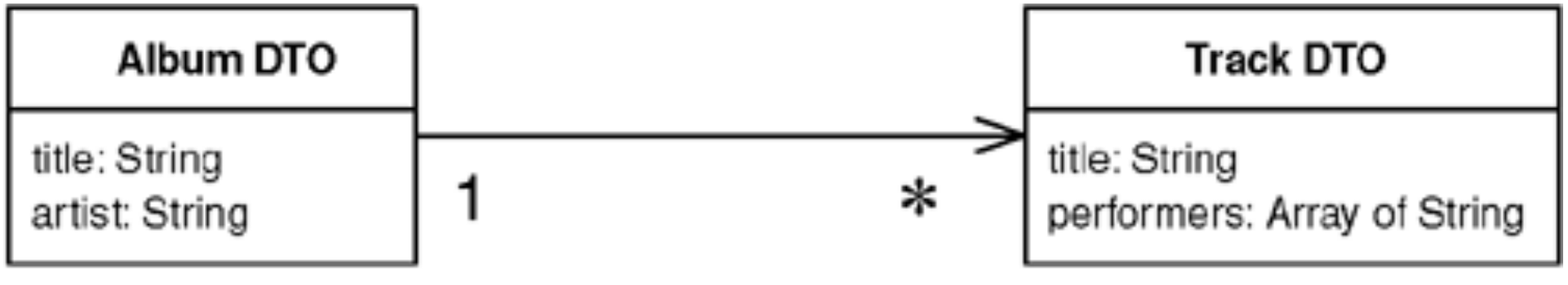


Șablonul Data Transfer Object

- Un DTO ar trebui folosit ori de câte ori este necesară transmiterea mai multor date între două procese într-un singur apel de metodă.
- Alternative:
  - De a folosi metode de tip set/get cu mai mulți parametrii transmiși prin referință.
    - Multe limbaje (ex. Java) permit returnarea unei singure valori.
    - Alternativa poate fi folosită pentru actualizări (metode de tip set), dar nu poate fi folosită pentru a obține date (metode de tip get).
  - Folosirea unei reprezentări sub formă de string.
    - Totul va fi cuplat cu reprezentarea sub formă de string (poate fi costisitoare).

Data Transfer Object - Exemplu





Data Transfer Object - Exemplu

```
class AlbumAssembler{
   public AlbumDTO writeDTO(Album subject) {
      AlbumDTO result = new AlbumDTO();
      result.setTitle(subject.getTitle());
      result.setArtist(subject.getArtist().getName());
      writeTracks(result, subject);
      return result;
   }
   private void writeTracks(AlbumDTO result, Album subject) {
      List<TrackDTO> newTracks = new ArrayList<TrackDTO>();
       for(Track track: subject.getTracks()){
            TrackDTO newDTO = new TrackDTO();
            newDTO.setTitle(track.getTitle());
            writePerformers(newDTO, track);
            newTracks.add(newDTO);
       }
result.setTracks(newTracks.toArray(new TrackDTO[newTracks.size()]));
   }
```

Data Transfer Object - Exemplu

```
private void writePerformers(TrackDTO dto, Track subject) {
      List<String> result = new ArrayList<String>();
       for(Artist artist: subject.getPerformers()){
            result.add(artist.getName());
      }
      dto.setPerformers(result.toArray(new String[result.size()]));
}
```

Networking în Java

- **`java.net`** - pachetul conține clase pentru comunicarea TCP/UDP prin rețea.
- TCP: **`Socket`** și **`ServerSocket`**.
- UDP: **`DatagramPacket`**, **`DatagramSocket`** și **`MulticastSocket`**.
- Clasa **`InetAddress`** reprezintă o adresă IP:
  - **`Inet4Address`**: pentru adrese IPv4 (32 bits).
  - **`Inet6Address`**: pentru adrese IPv6 (128 bits).

```
InetAddress localHost=InetAddress.getLocalHost();
InetAddress googAdr=InetAddress.getByName("www.google.com");
```

- **`InetSocketAddress`** asociere între o adresă IP și un port:

```
InetSocketAddress(InetAddress addr, int port) ;
InetSocketAddress(String hostname, int port);
```

Networking in Java

- **`ServerSocket`** reprezintă clasa corespunzătoare serverului care așteaptă conexiuni TCP.

- Constructori/Metode:

```
public ServerSocket(int port) throws BindException, IOException
public ServerSocket( ) throws IOException //not bind yet, since Java 1.4

//binds a server to a port
public void bind(SocketAddress endpoint) throws IOException

//blocks and waits for clients
public Socket accept( ) throws IOException

//closes the server
public void close() throws IOException
```

Networking în Java

```
ServerSocket server=null;
try{
    server=new ServerSocket(5555);
    while(keepProcessing){
         Socket client=server.accept();
         //processing code
    }
}catch(IOException ex){
    //…
}finally{
     if(server!=null){
        try{
           server.close();
        }catch(IOException ex){…}
    }
}
```

Networking în Java

- **`java.net.Socket`** deschide o conexiune TCP din partea clientului.

```
public Socket(String host, int port) throws UnknownHostException,
                                            IOException
public Socket(InetAddress host, int port) throws IOException
```

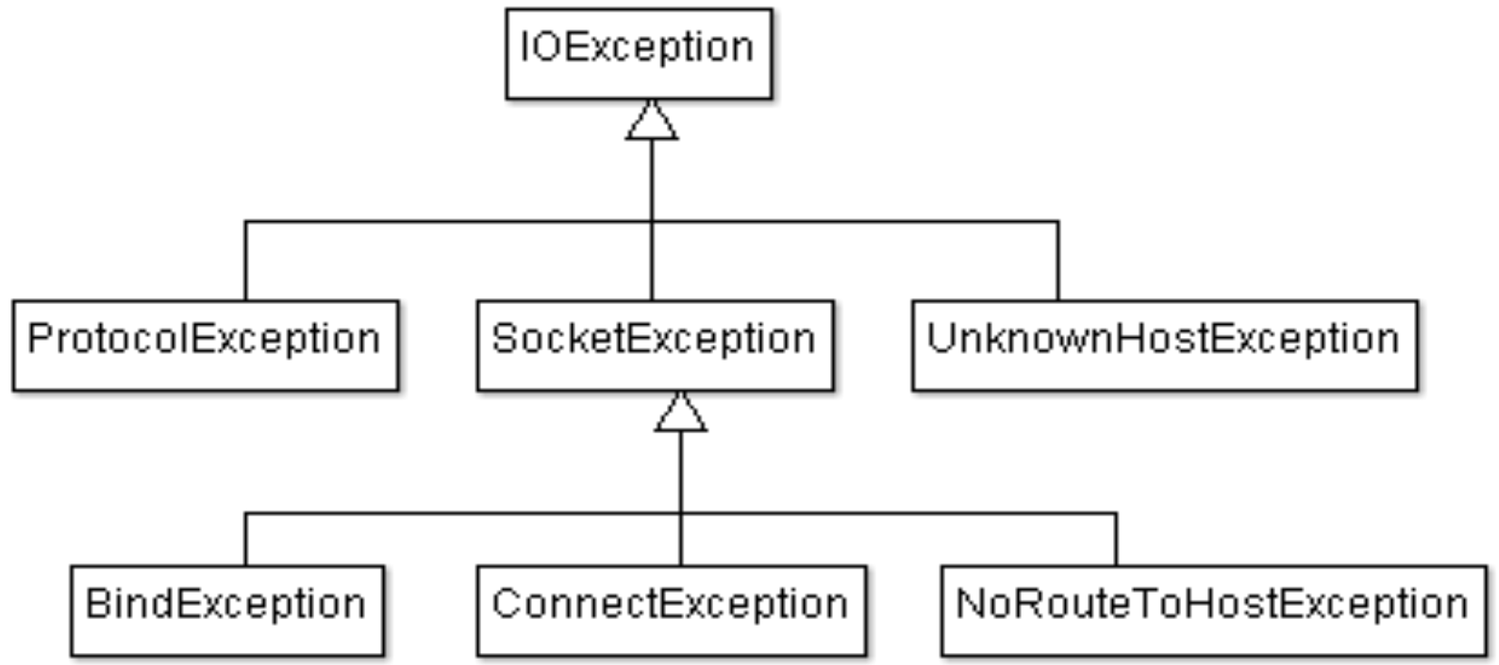
- Metode:

```
public int getPort()
public InputStream getInputStream() throws IOException
public OutputStream getOutputStream() throws IOException
public void close() throws IOException
```

Networking în Java

```
try (Socket connection=new Socket(“172.30.106.5”, 5555)){
    //processing code
}catch(UnknownHostException e){
   //…
}catch(IOException  e){
   //…
}
```

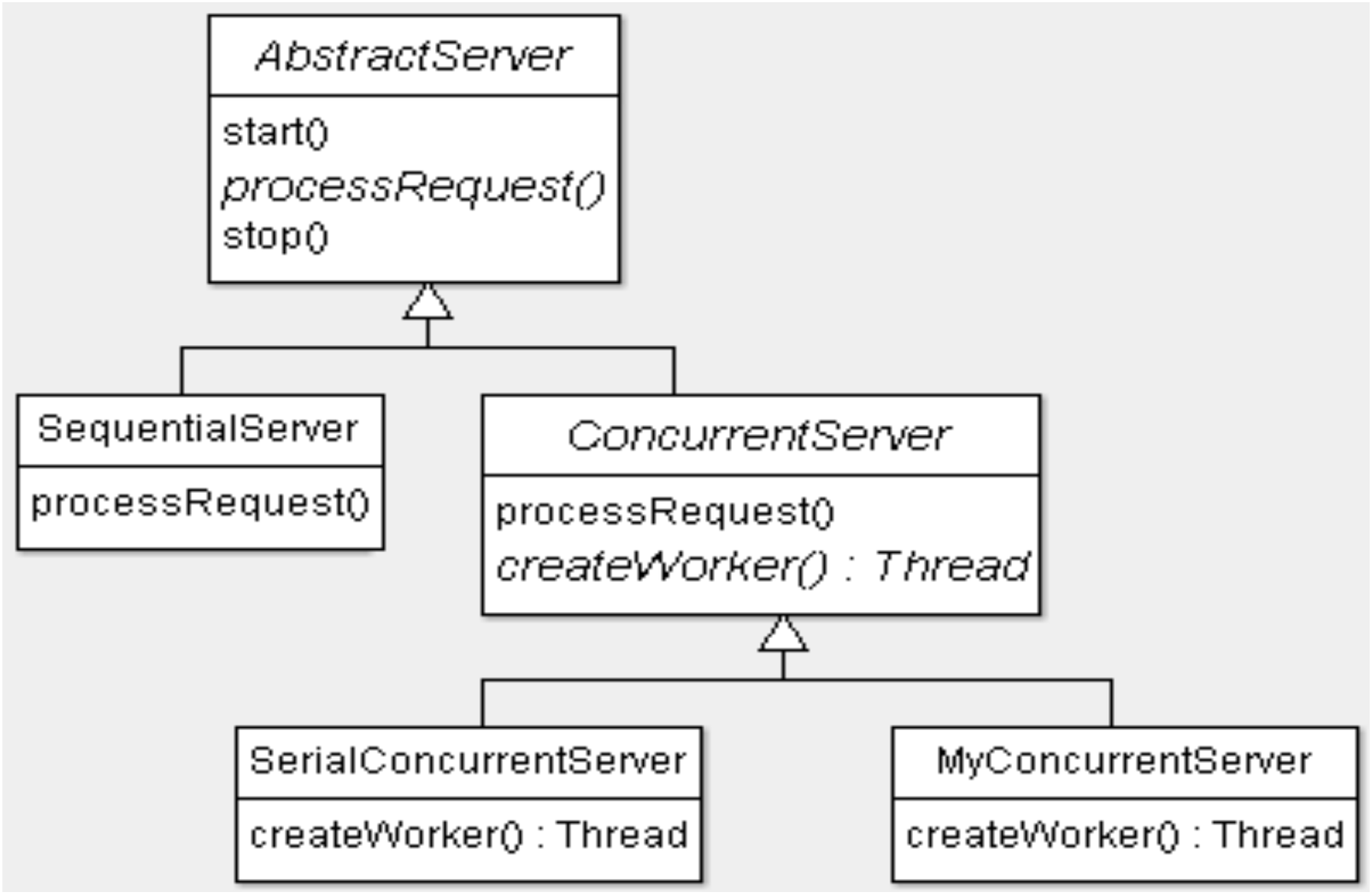
Excepții


IOException
ProtocolException
SocketException
UnknownHostException
BindException
ConnectException
NoRouteToHostException

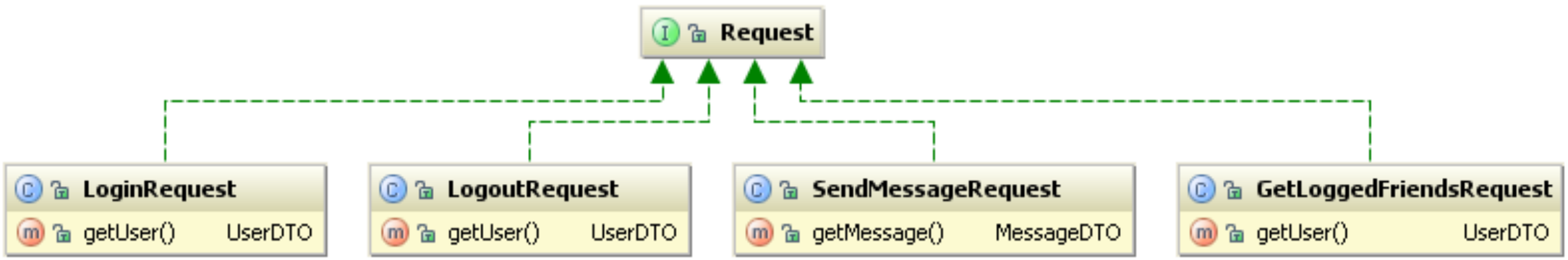
Exemplu Java

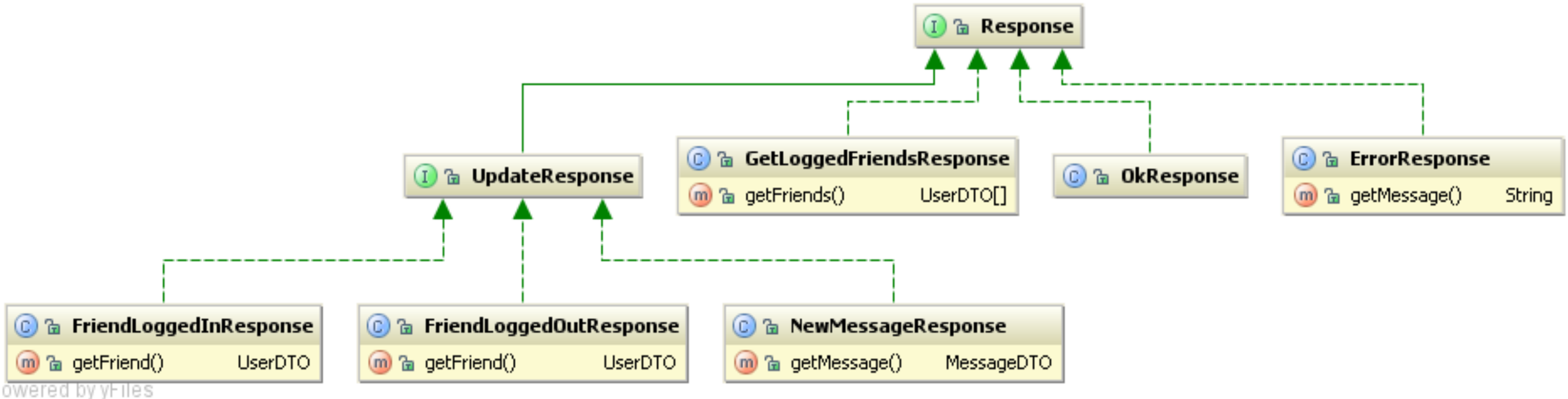
- O aplicație simplă client/server:
  - Serverul așteaptă conexiuni.
  - Clientul se conectează la server si îi trimite un text.
  - Serverul returnează textul scris cu litere mari, la care adaugă data și ora la care a fost primit textul.

Server Template



Mini-chat Object Protocol





Mini-chat Rpc Protocol