Module 626-1

ARCHITECTURE DU SYSTÈME D'INFORMATION ET DESIGN PATTERNS

Dr. Stefan Behfar

session 3

Overview

- Resource patterns
- Service composition
  - SOA and microservices
  - System design patterns
- Client-service communication
- Service containers (Docker)

Quality Attributes with REST

Feature for performance : statelessness

- Statelessness enables the tactic: “Introduce concurrency”
  - Since the requests can be processed in isolation, independently from previous requests, they can be distributed on several machines without impact on the outcome of the processing.

Quality Attributes with REST

Features for understandability: statelessness, uniform interface, adressability

- Statelessness
  - Allows each request to be understood in isolation. Indeed all the context to interpret a request is in the request
- Uniform interface & adressability
  - Allows protocol visibility and simplicity
    - ✓ Each request is transparent (easily interpretable)

4. Statelessness I

Reminder: state = internal configuration of a system that specifies the response to the event it receives.

There are 2 kinds of states:

- **Resource state**: configuration of a resource on the server that is available to all clients. This determines what representation one could get from the resource.
- **Client state**: configuration of the client that determines what query it could issue next (following some user input for example). This is also called the ***application*** *state.*

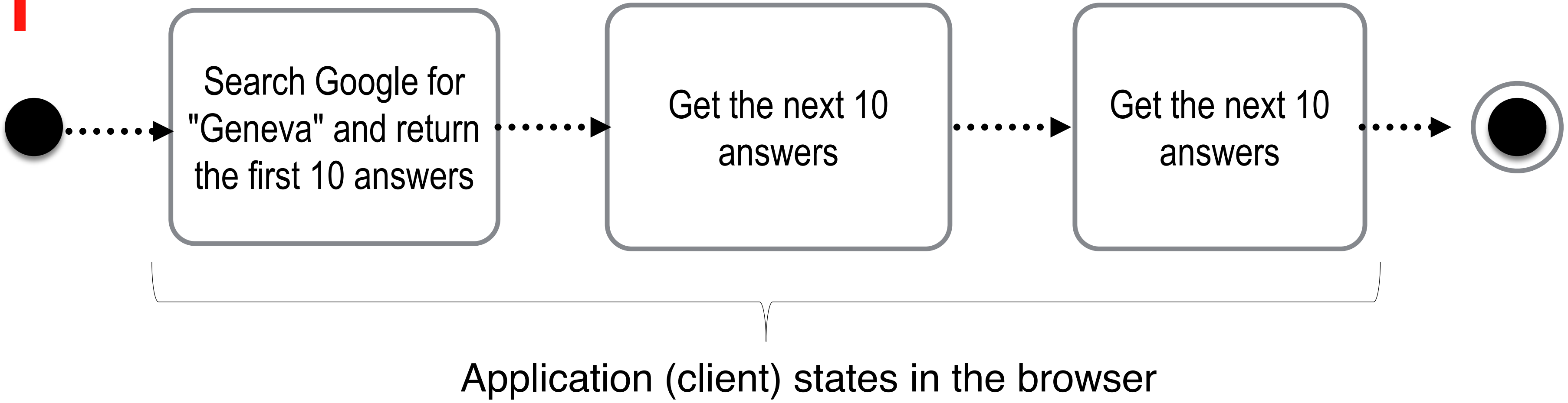
Statelessness II

All queries should be independent from each other from the point of view of the server

- The server does not record the interaction with a particular client.
- Each client "navigates" among the resources and records the last (client) state he reached.

- Therefore, there are 2 locations for the states:
  - ✓ Client states (client side, specific to the client).
  - ✓ Resource states (server side, available to all clients)

Examples of client states


Search Google for "Geneva" and return the first 10 answers
Get the next 10 answers
Get the next 10 answers
Application (client) states in the browser

Transitions
- www.google.com/search?q=geneva
- www.google.com/search?q=geneva & start=10

Client's state communicated to the server

Statelessness, what does it mean?

- It means that there are **no user sessions** on the server.
  - The server must not record the state of the conversation with some user
  - But this does not mean that it should not record states in a database, provided that it is **resource** state.

- Tricky example: shopping cart
  - Application or resource state ?
    - ✓ Specific to client, since other customer do not know about it ?
    - ✓ But if the client crashes, it could retrieve it, so it is not on the client ?

Buying on internet is part of a business process the customer must go through to complete the transaction. The steps are required to buy the goods. Therefore the server must know about it : resource states (the steps of the process). But if the user only browses products without buying, this is client state.

Changement de paradigme en conception d'application

1. Penser application distribuée
2. Penser clients nombreux et simultanés
3. Penser montée en charge

Resource Patterns

Basic patterns to interact with the resources

Request / Acknowledge

Problem: how to handle long lasting services i.e. services that cannot reply immediately?

Solutions:

- Request/acknowledge/poll
- Request/acknowledge/callback

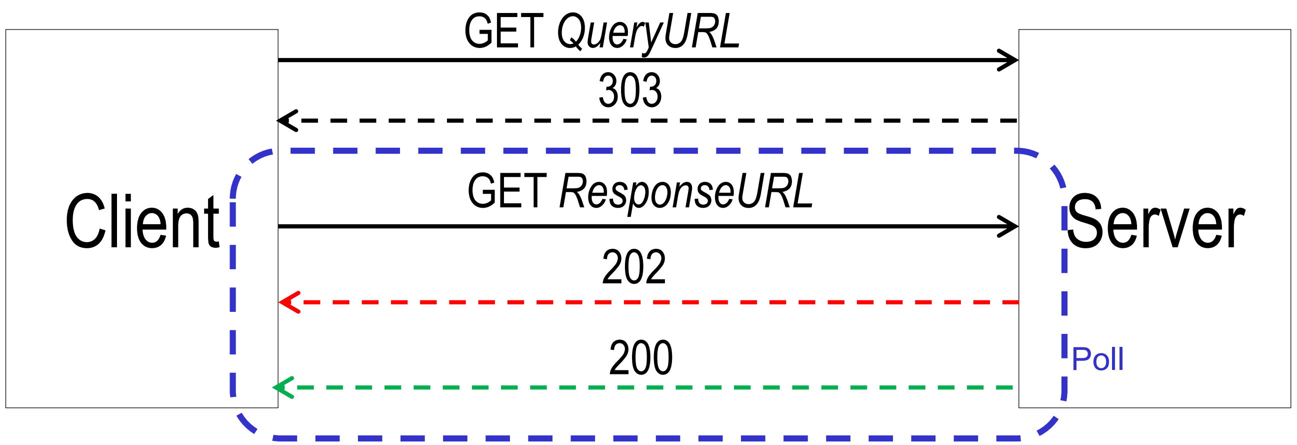
Request/acknowledge/poll

The query, once submitted is returned immediately

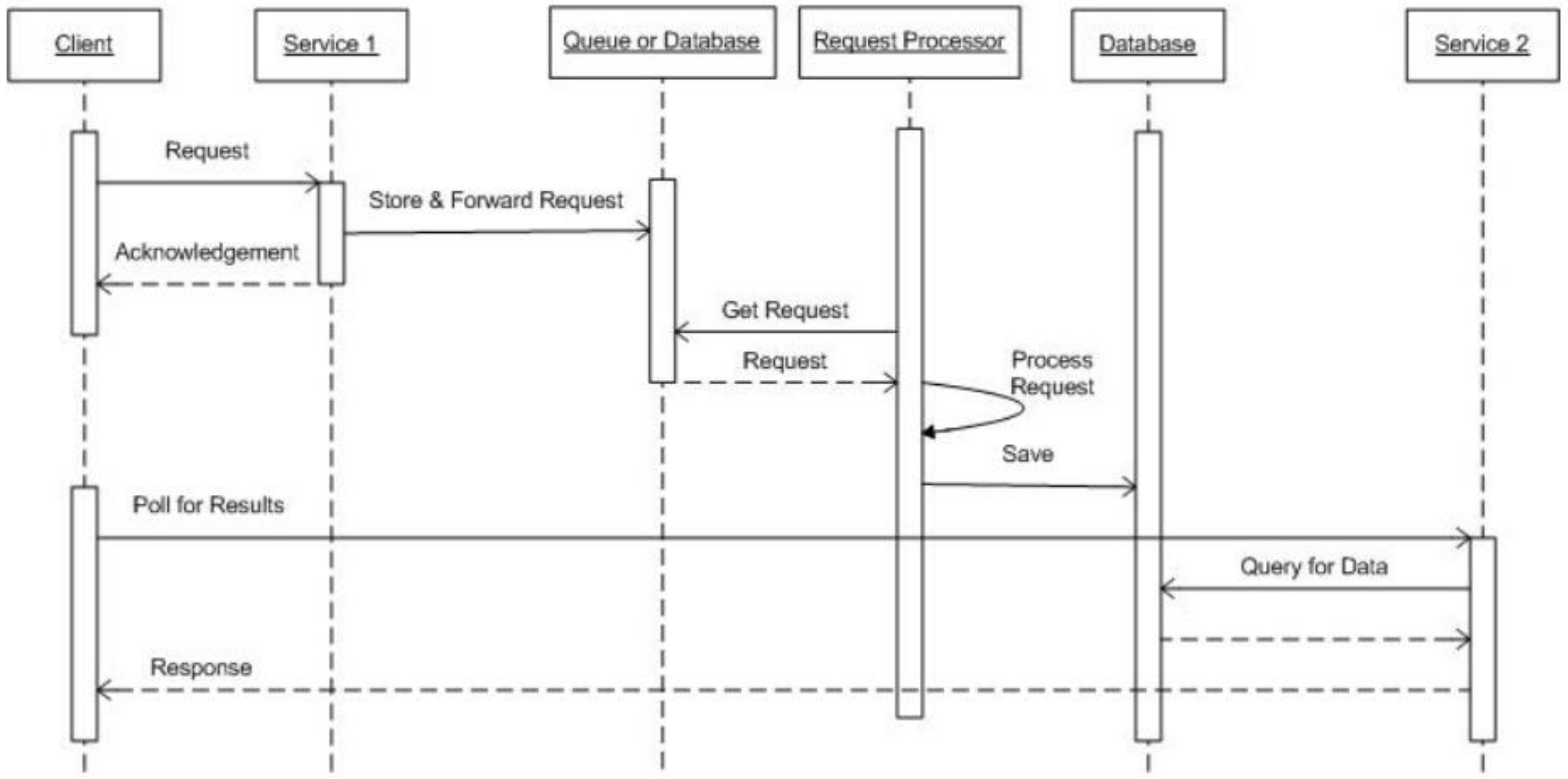
- Return code: 303* (See other) & provision of a new URL (**Location**).
- The client will poll the provided new URL. Return codes:
  - 202 Accepted, as long as not ready
  - 200 OK, when ready

Since the response is provided through another URL there is no overlap

* Depending on the client, it may issue another query to the alternative URL automatically. So you won't see 303



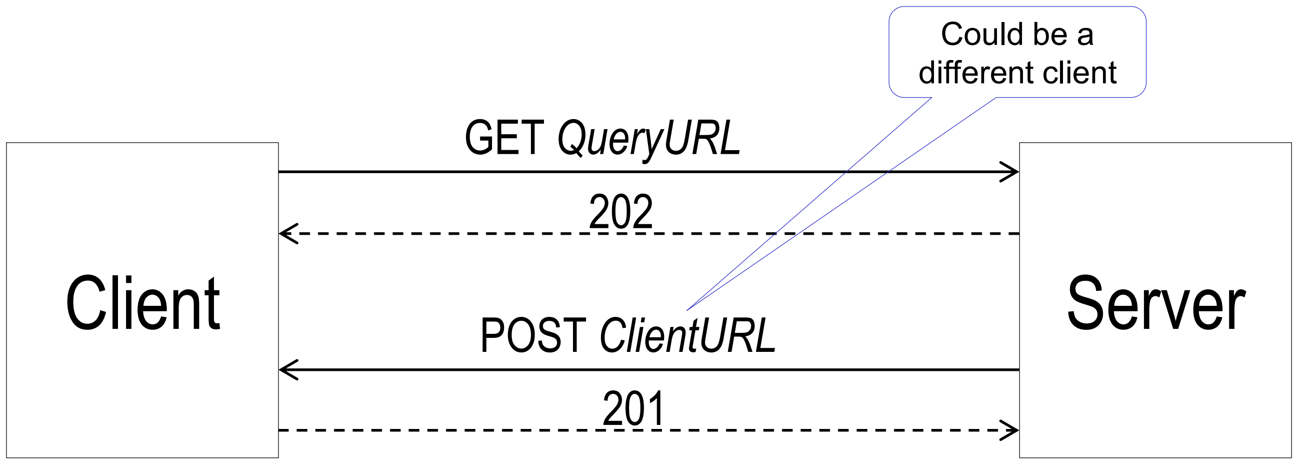
Interactions



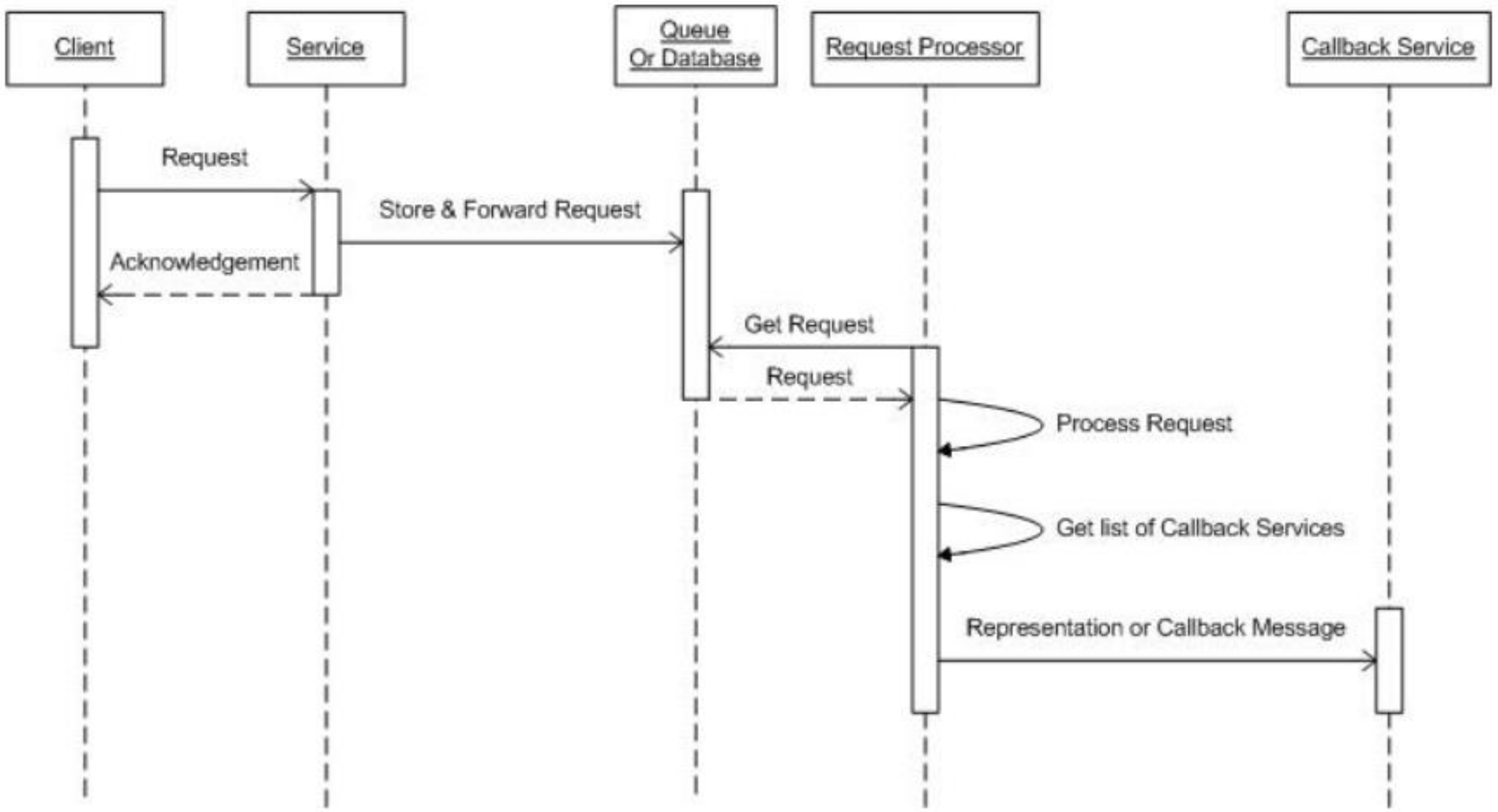
Source: http://servicedesignpatterns.com/clientserviceinteractions/requestacknowledge, accessed 2017.05.07

Request/acknowledge/callback

- The GET query contains the URL to which the response must be sent. The query is returned immediately with 202 (Accepted)
- When the response is ready, the server sends it to the recipients through a POST (PUT) query to the identified resources (which return 201 (Created))
  - In the general case (several recipients), the *payload* includes the URL of the client resource to be updated when the processing completes
  - If the callback URL is unique, it can be passed in the location header



Interations



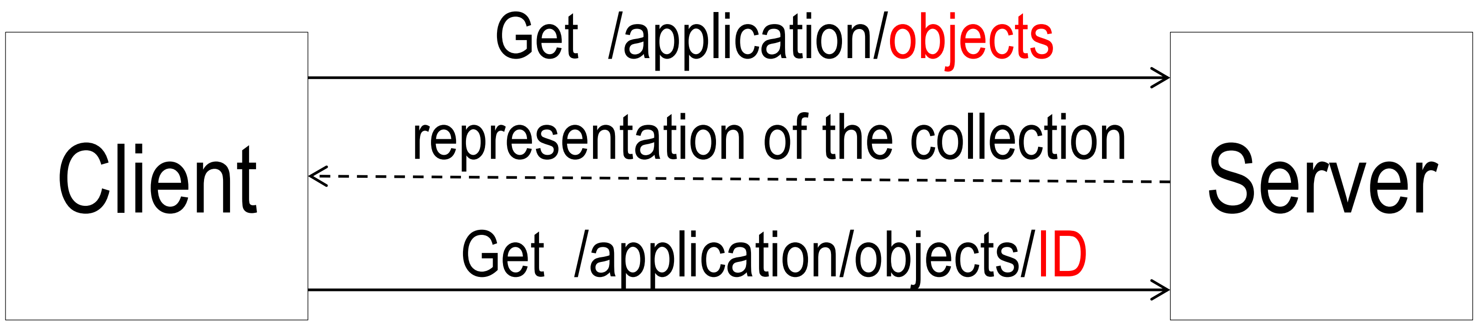
Source: http://servicedesignpatterns.com/clientserviceinteractions/requestacknowledge, accessed 2017.05.07

Collection Pattern

- **Problem**: how to select objects among a set of objects to pass them further to another service ?
  - For bandwidth reasons, one cannot pass ALL objects to the client for it to select one of them.

- **Solution**: create a collection of representations of objects for selection purpose. Contents:
  - Meaningful description for selection
  - Unique ID

Collection



- Example of the implementation of the Collection pattern: Atom
  - The Atom FEED structure transfer only the meaningful part of an ENTRY (Title & ID + alternate link) for further selection and retrieval

FEED

feed (<feed> </feed> tags)

- required elements
  - **id**: identifies the feed using a universally unique and permanent URI.
  - **title**: human readable title for the feed
  - **updated**: indicates the last time the feed was modified
- recommended elements
  - **author**: identifies the author of the feed. A feed must contain at least one author element unless all of the *entry* elements contain at least one author element
  - **link**: Identifies a related Web page.
- A feed should contain a link back to the feed itself (rel = “SELF”)

ENTRY

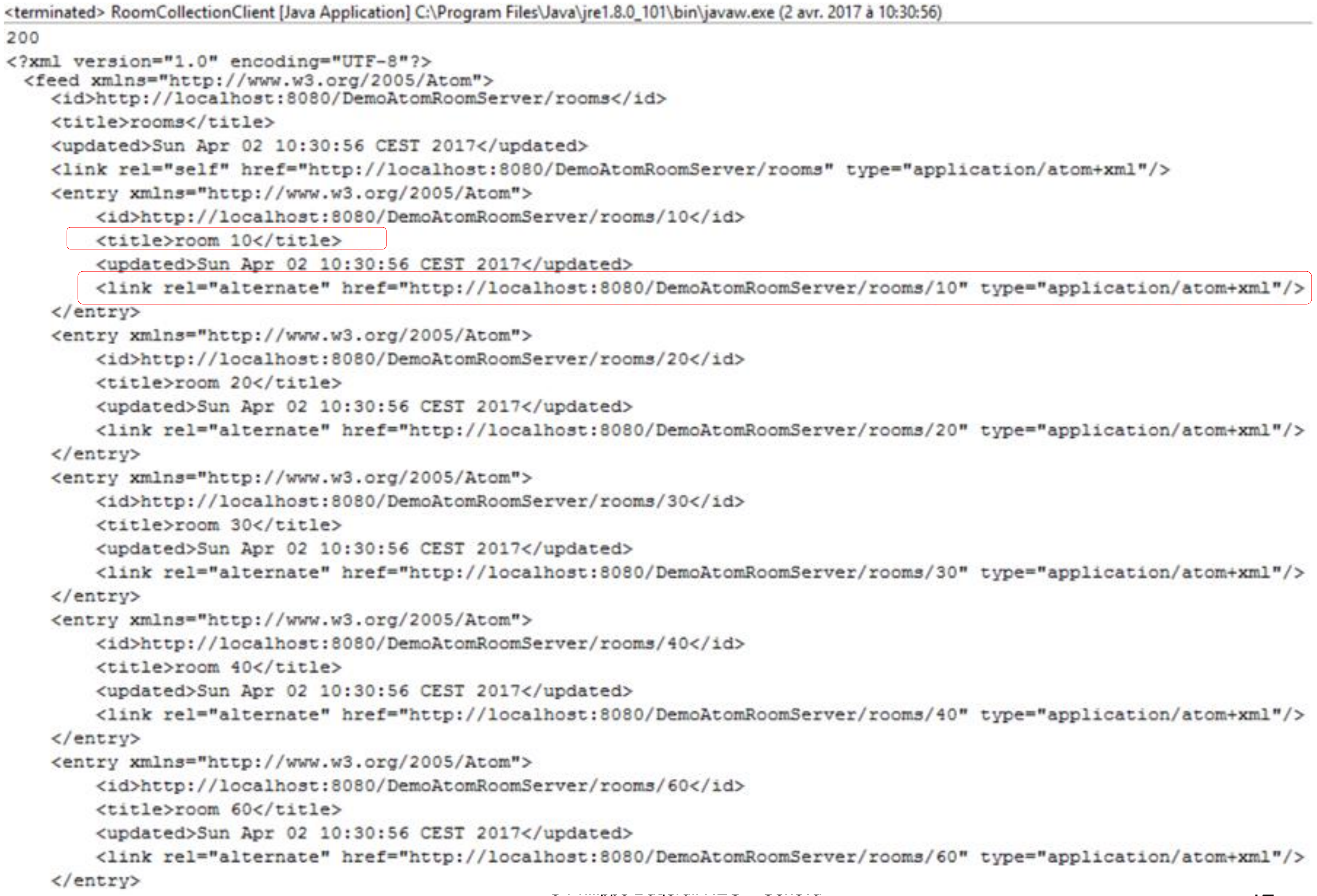
entry (<entry> </entry> tags)

- required elements
  - **id**: identifies the entry using a universally unique and permanent URI.
  - **title**: human readable title for the entry
  - **updated**: indicates the last time the entry was modified

- recommended elements
  - **author**: identifies the author of the feed. An entry must contain at least one author element unless there is an author element in the enclosing feed.
  - **content**: full contents or "alternate" link to the complete content of the entry. Content must be provided if there is no alternate link.
    - *An entry must contain an alternate link if there is no content element*..
  - **link**: Identifies a related Web page. The type of relation is defined by the rel attribute.

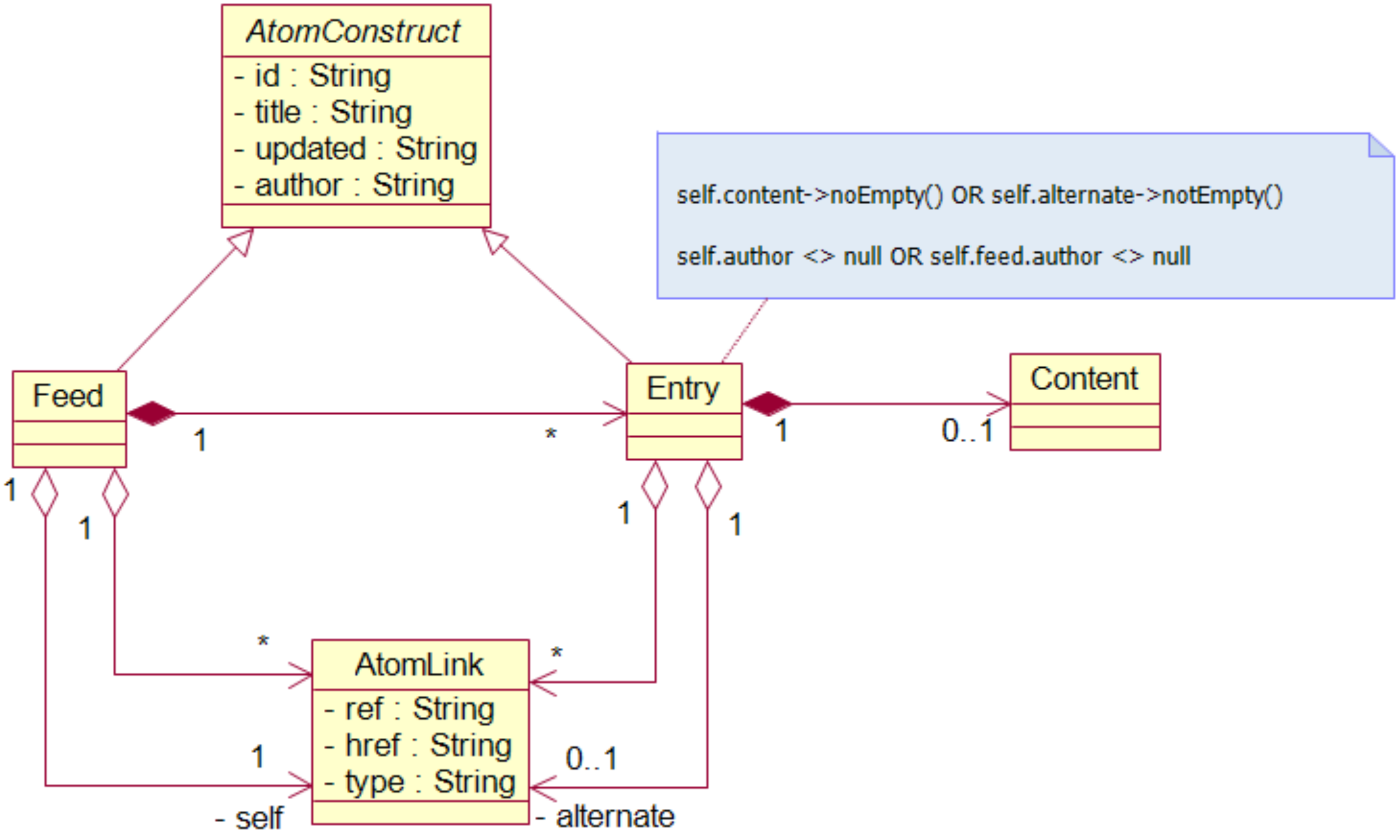
heg

Feed

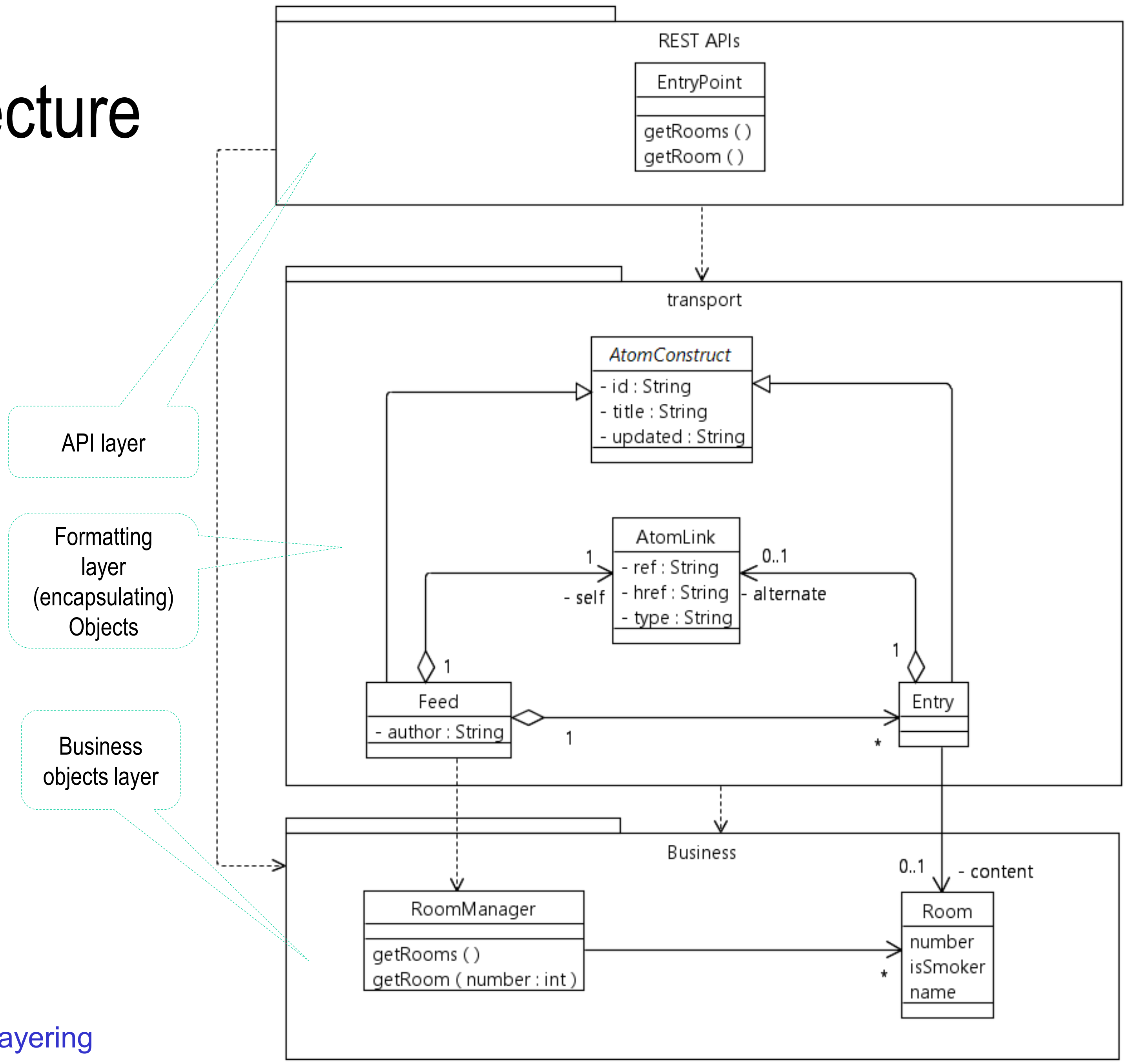


```
<terminated> RoomCollectionClient [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.exe (2 avr. 2017 à 10:30:56)
200
<?xml version="1.0" encoding="UTF-8"?>
  <feed xmlns="http://www.w3.org/2005/Atom">
    <id>http://localhost:8080/DemoAtomRoomServer/rooms</id>
    <title>rooms</title>
    <updated>Sun Apr 02 10:30:56 CEST 2017</updated>
    <link rel="self" href="http://localhost:8080/DemoAtomRoomServer/rooms" type="application/atom+xml"/>
    <entry xmlns="http://www.w3.org/2005/Atom">
        <id>http://localhost:8080/DemoAtomRoomServer/rooms/10</id>
        <title>room 10</title>
        <updated>Sun Apr 02 10:30:56 CEST 2017</updated>
        <link rel="alternate" href="http://localhost:8080/DemoAtomRoomServer/rooms/10" type="application/atom+xml"/>
    </entry>
    <entry xmlns="http://www.w3.org/2005/Atom">
        <id>http://localhost:8080/DemoAtomRoomServer/rooms/20</id>
        <title>room 20</title>
        <updated>Sun Apr 02 10:30:56 CEST 2017</updated>
        <link rel="alternate" href="http://localhost:8080/DemoAtomRoomServer/rooms/20" type="application/atom+xml"/>
    </entry>
    <entry xmlns="http://www.w3.org/2005/Atom">
        <id>http://localhost:8080/DemoAtomRoomServer/rooms/30</id>
        <title>room 30</title>
        <updated>Sun Apr 02 10:30:56 CEST 2017</updated>
        <link rel="alternate" href="http://localhost:8080/DemoAtomRoomServer/rooms/30" type="application/atom+xml"/>
    </entry>
    <entry xmlns="http://www.w3.org/2005/Atom">
        <id>http://localhost:8080/DemoAtomRoomServer/rooms/40</id>
        <title>room 40</title>
        <updated>Sun Apr 02 10:30:56 CEST 2017</updated>
        <link rel="alternate" href="http://localhost:8080/DemoAtomRoomServer/rooms/40" type="application/atom+xml"/>
    </entry>
    <entry xmlns="http://www.w3.org/2005/Atom">
        <id>http://localhost:8080/DemoAtomRoomServer/rooms/60</id>
        <title>room 60</title>
        <updated>Sun Apr 02 10:30:56 CEST 2017</updated>
        <link rel="alternate" href="http://localhost:8080/DemoAtomRoomServer/rooms/60" type="application/atom+xml"/>
    </entry>
```

Reminder ATOM: minimal UML model



Architecture



Reuse-based layering

Pattern: Data Transfer Objects

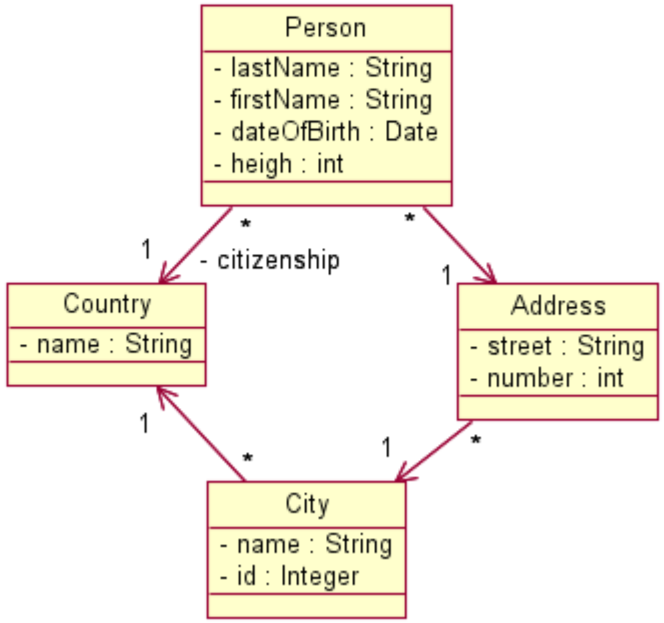
- Problem: to minimize network traffic, one does not want to send/receive all information of domain object, but only the meaningful part for the service
  - How to extract and the selected part of the domain object for network transfer.
  - Constraint: one does not want to implement transfer-specific code in domain objects
- When translating to XML using a standard library (JAXB), how to include / exclude information ?
  - Constraint (again): we do not want to add annotations in domain objects

h e g

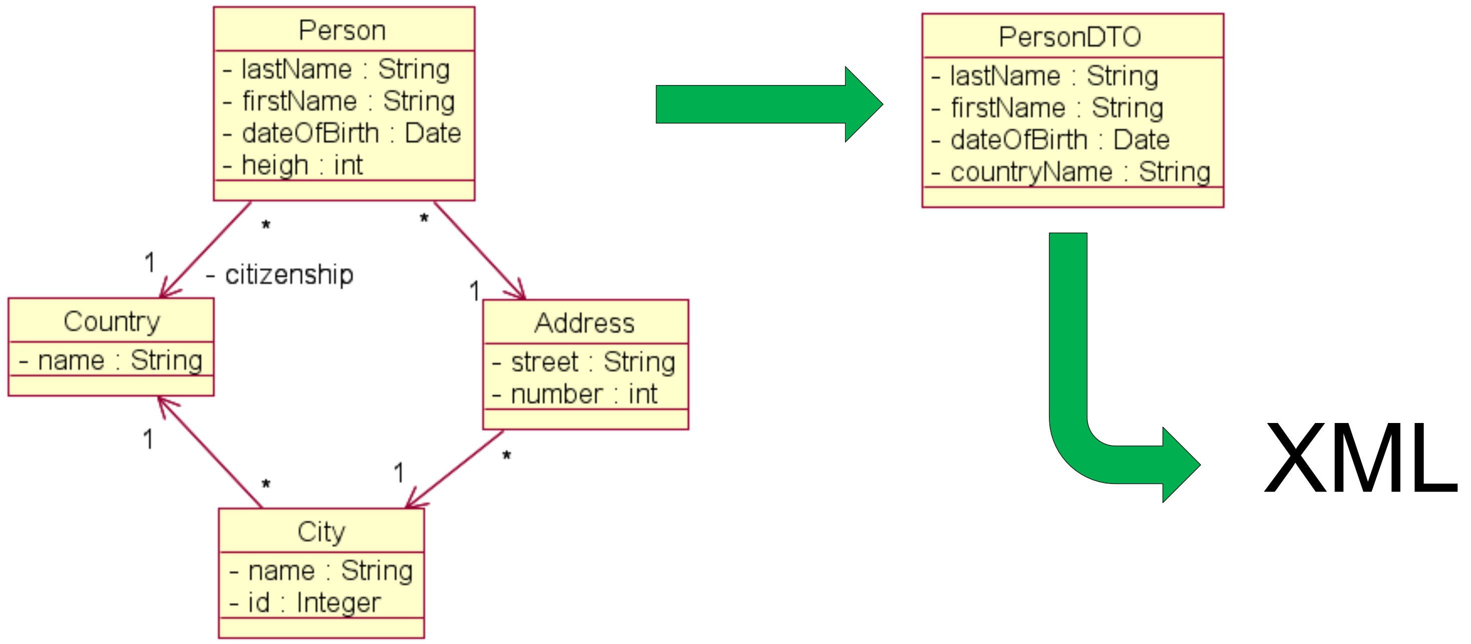
DTO

Solution: create a Data Transfer Object to contain only the part of the domain object we must transfer (or parts of several objects).

Example: when translating some Person data (name, birthdate, country) to XML using JAXB we do not want to include the full object tree.



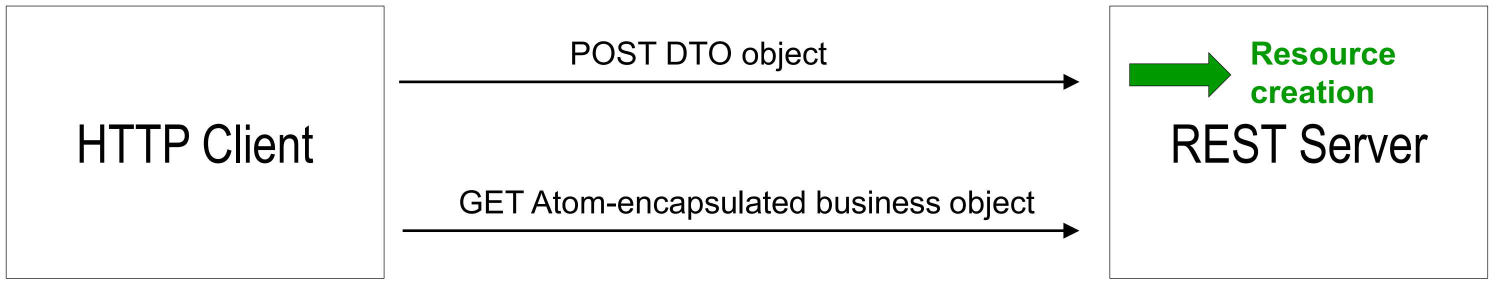
Solution DTO



- The DTO object can easily be translated to XML using JAXB, without impacting the domain object
- Not being a business object, the DTO can be annotated

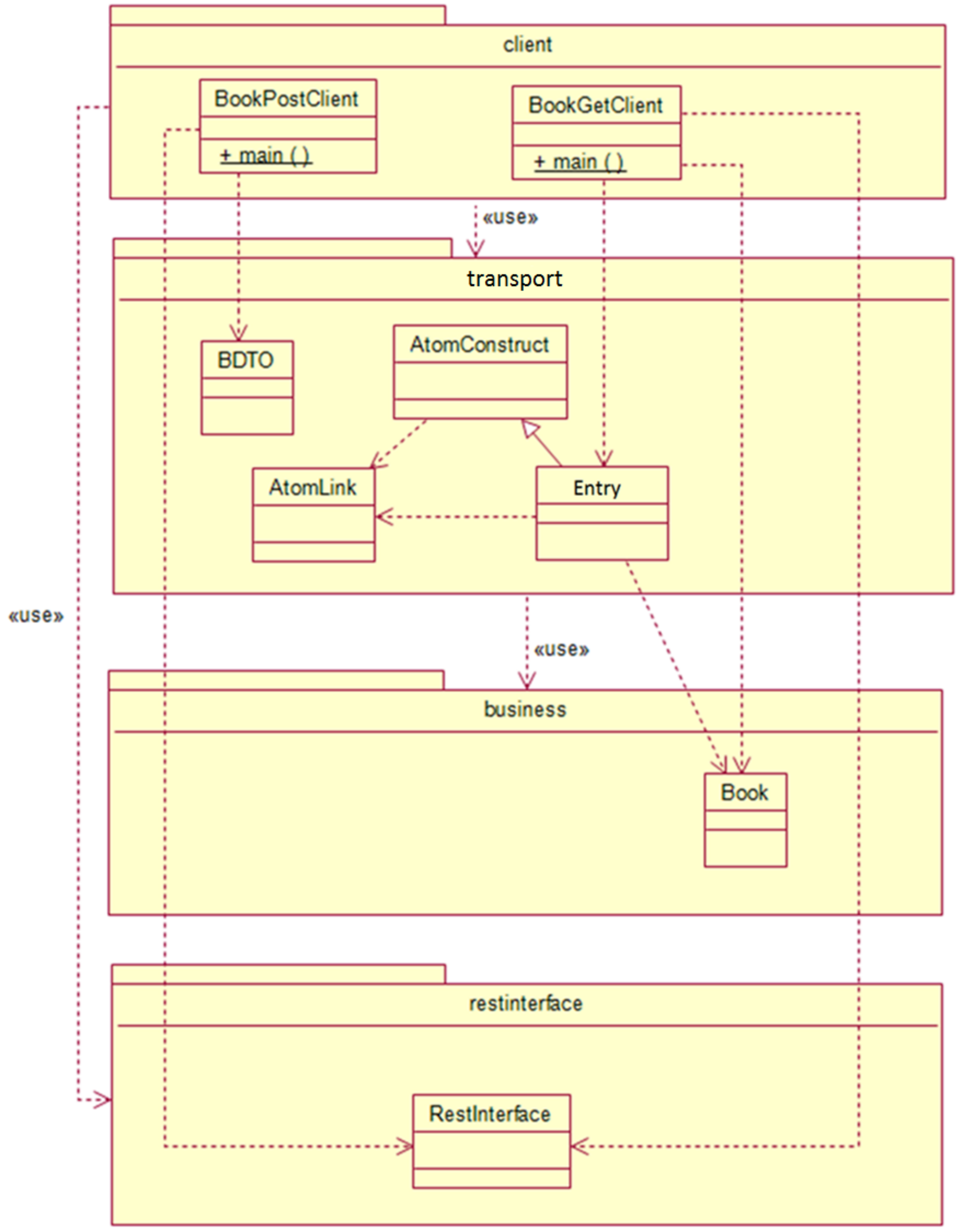
Example: POST

When POSTing representations to create objects on the server, the representation can be sent using a DTO



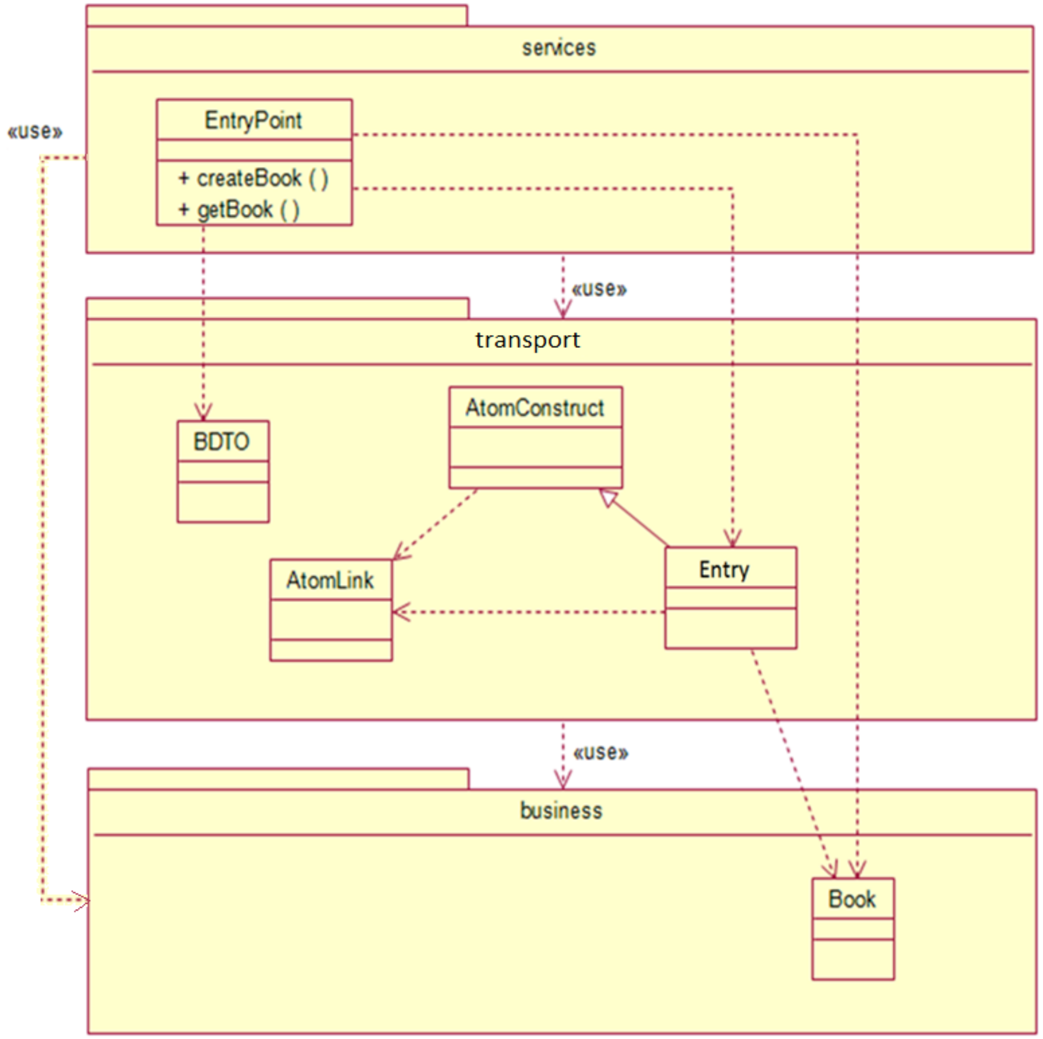
h e g

Creating books, client side

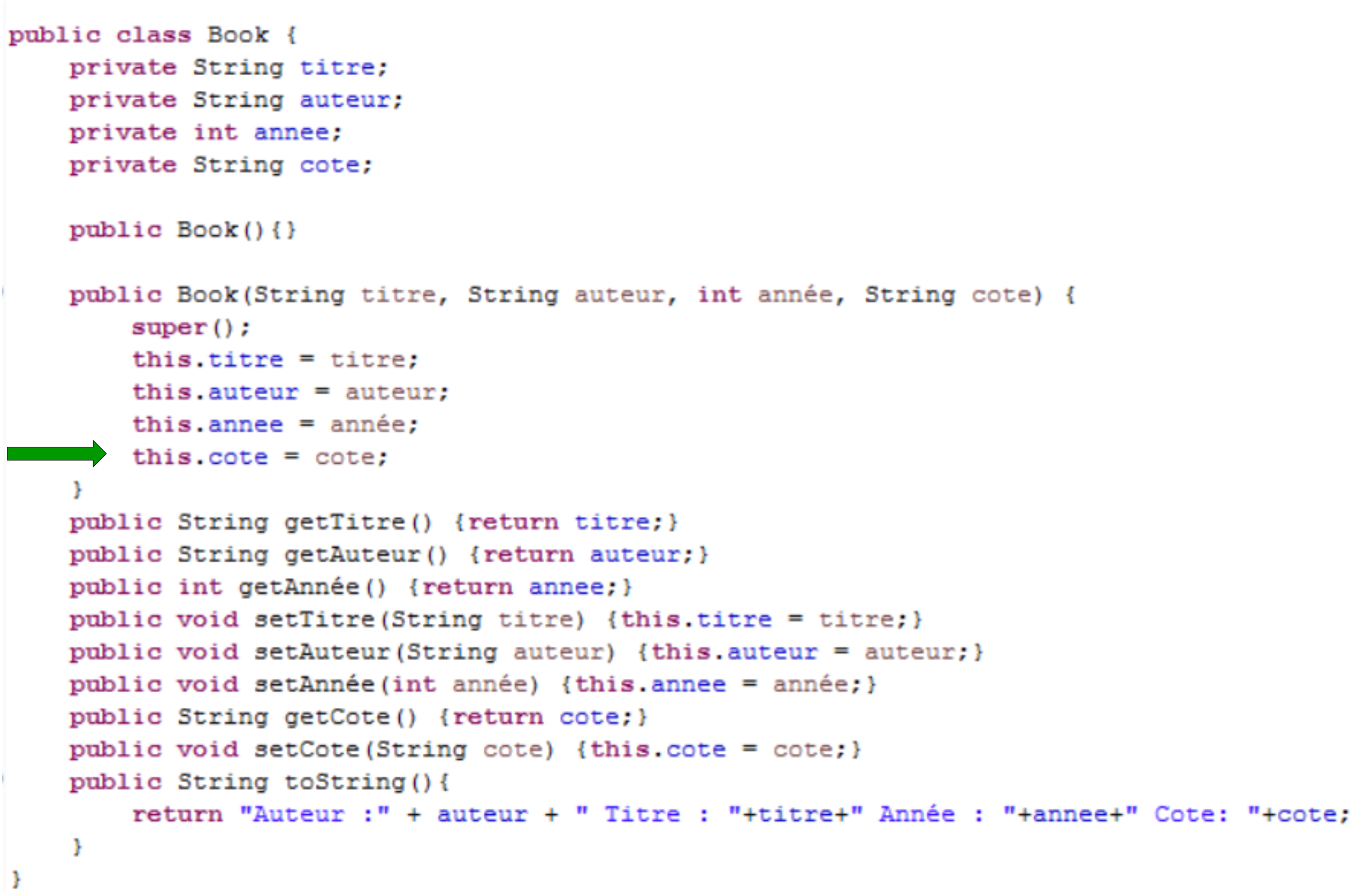


h e g

Example: creating books, server side



Books



```
public class Book {
    private String titre;
    private String auteur;
    private int annee;
    private String cote;

    public Book(){}

    public Book(String titre, String auteur, int année, String cote) {
        super();
        this.titre = titre;
        this.auteur = auteur;
        this.annee = année;
        this.cote = cote;
    }
    public String getTitre() {return titre;}
    public String getAuteur() {return auteur;}
    public int getAnnée() {return annee;}
    public void setTitre(String titre) {this.titre = titre;}
    public void setAuteur(String auteur) {this.auteur = auteur;}
    public void setAnnée(int année) {this.annee = année;}
    public String getCote() {return cote;}
    public void setCote(String cote) {this.cote = cote;}
    public String toString(){
        return "Auteur :" + auteur + " Titre : "+titre+" Année : "+annee+" Cote: "+cote;
    }
}
```

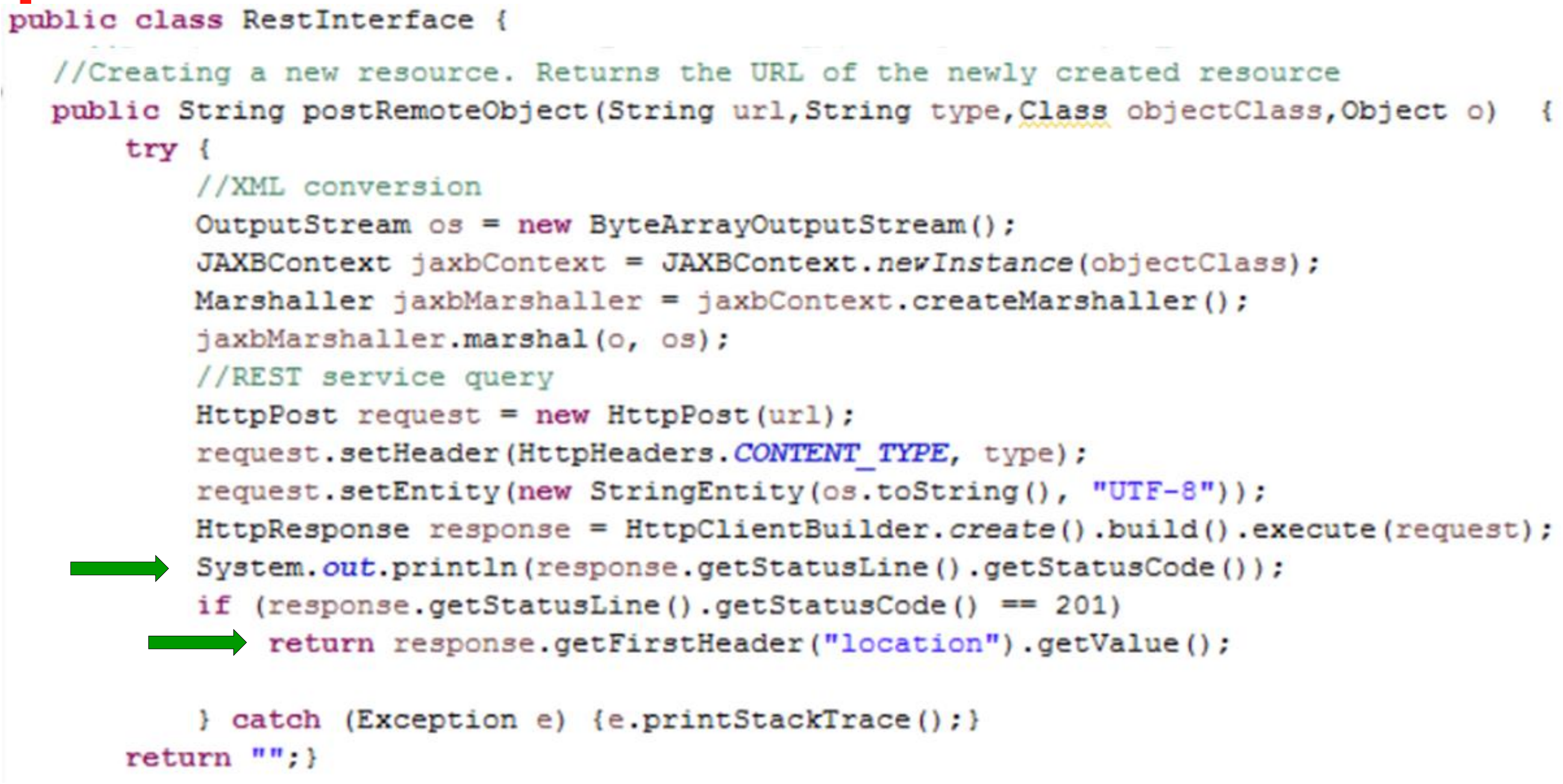
Book DTO

```
import javax.xml.bind.annotation.*;

@XmlRootElement
@XmlAccessorType( XmlAccessType.FIELD)
public class BDTO {
    private String title;
    private String auteur;
    private int annee;

    public BDTO(){}

    public BDTO(String title, String auteur, int annee) {
        super();
        this.title = title;
        this.auteur = auteur;
        this.annee = annee;
    }

    public String getTitle() {return title;}
    public String getAuteur() {return auteur;}
    public int getAnnee() {return annee;}
}
```

Client

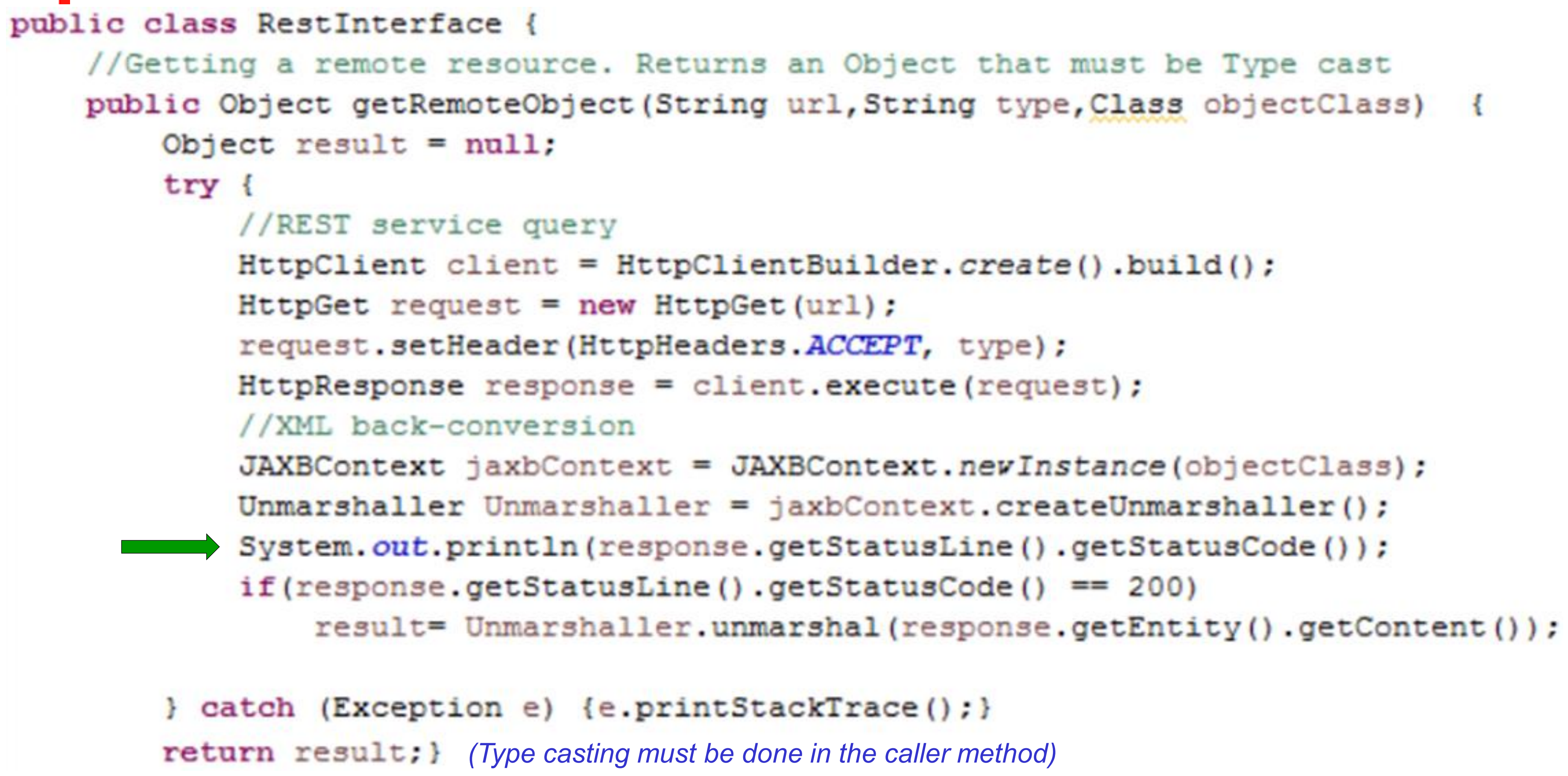
```java
public class BookPostClient {

    public static void main(String[] args) throws Exception {

        BDTO dto = new BDTO("Seigneur des anneaux","Tolkien",1954);
        String url = "http://localhost:8080/DemoPostServer/books/";

        String newUrl = new RestInterface()
                .postRemoteObject(url,MediaType.APPLICATION_XML, BDTO.class,dto);
        System.out.println(newUrl);
    }
}

public class BookGetClient {

    public static void main(String[] args) {

            String url = "http://localhost:8080/DemoPostServer/books/Tol1954";
            Entry<Book> o = (Entry<Book>)new RestInterface()
                    .getRemoteObject(url,MediaType.APPLICATION_ATOM_XML, Entry.class);
            Book b = o.getContents();
            System.out.println(b);
            }
}
```

REST interface: POST



```java
public class RestInterface {

   //Creating a new resource. Returns the URL of the newly created resource
   public String postRemoteObject(String url,String type,Class objectClass,Object o)  {
       try {
           //XML conversion
           OutputStream os = new ByteArrayOutputStream();
           JAXBContext jaxbContext = JAXBContext.newInstance(objectClass);
           Marshaller jaxbMarshaller = jaxbContext.createMarshaller();
           jaxbMarshaller.marshal(o, os);
           //REST service query
           HttpPost request = new HttpPost(url);
           request.setHeader(HttpHeaders.CONTENT_TYPE, type);
           request.setEntity(new StringEntity(os.toString(), "UTF-8"));
           HttpResponse response = HttpClientBuilder.create().build().execute(request);
           System.out.println(response.getStatusLine().getStatusCode());
           if (response.getStatusLine().getStatusCode() == 201)
               return response.getFirstHeader("location").getValue();

           } catch (Exception e) {e.printStackTrace();}
       return "";}
```

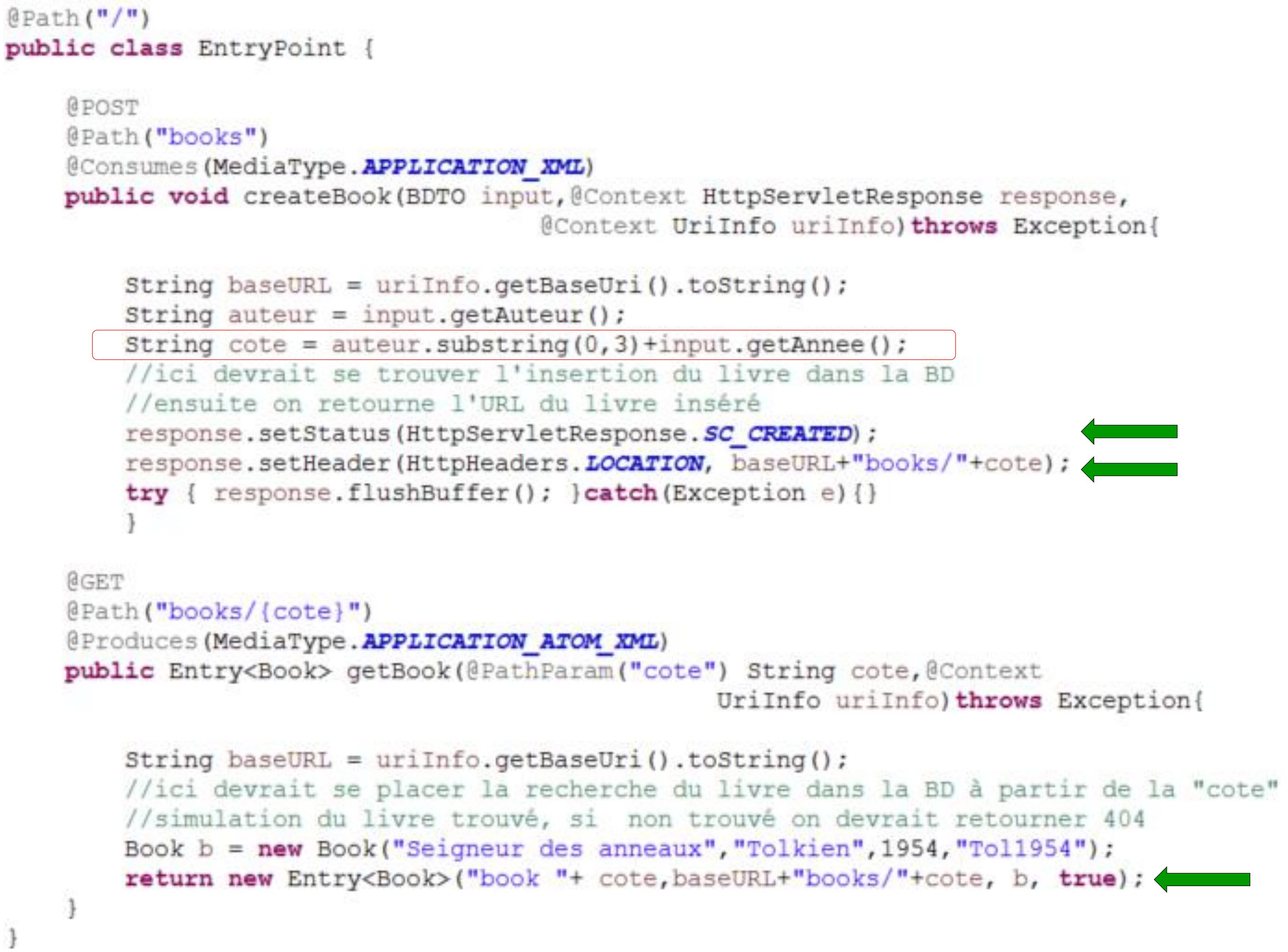
REST interface: GET



```
public class RestInterface {
    //Getting a remote resource. Returns an Object that must be Type cast
    public Object getRemoteObject(String url,String type,Class objectClass)  {
        Object result = null;
        try {
            //REST service query
            HttpClient client = HttpClientBuilder.create().build();
            HttpGet request = new HttpGet(url);
            request.setHeader(HttpHeaders.ACCEPT, type);
            HttpResponse response = client.execute(request);
            //XML back-conversion
            JAXBContext jaxbContext = JAXBContext.newInstance(objectClass);
            Unmarshaller Unmarshaller = jaxbContext.createUnmarshaller();
            System.out.println(response.getStatusLine().getStatusCode());
            if(response.getStatusLine().getStatusCode() == 200)
                result= Unmarshaller.unmarshal(response.getEntity().getContent());

        } catch (Exception e) {e.printStackTrace();}
        return result;}
```

(Type casting must be done in the caller method)

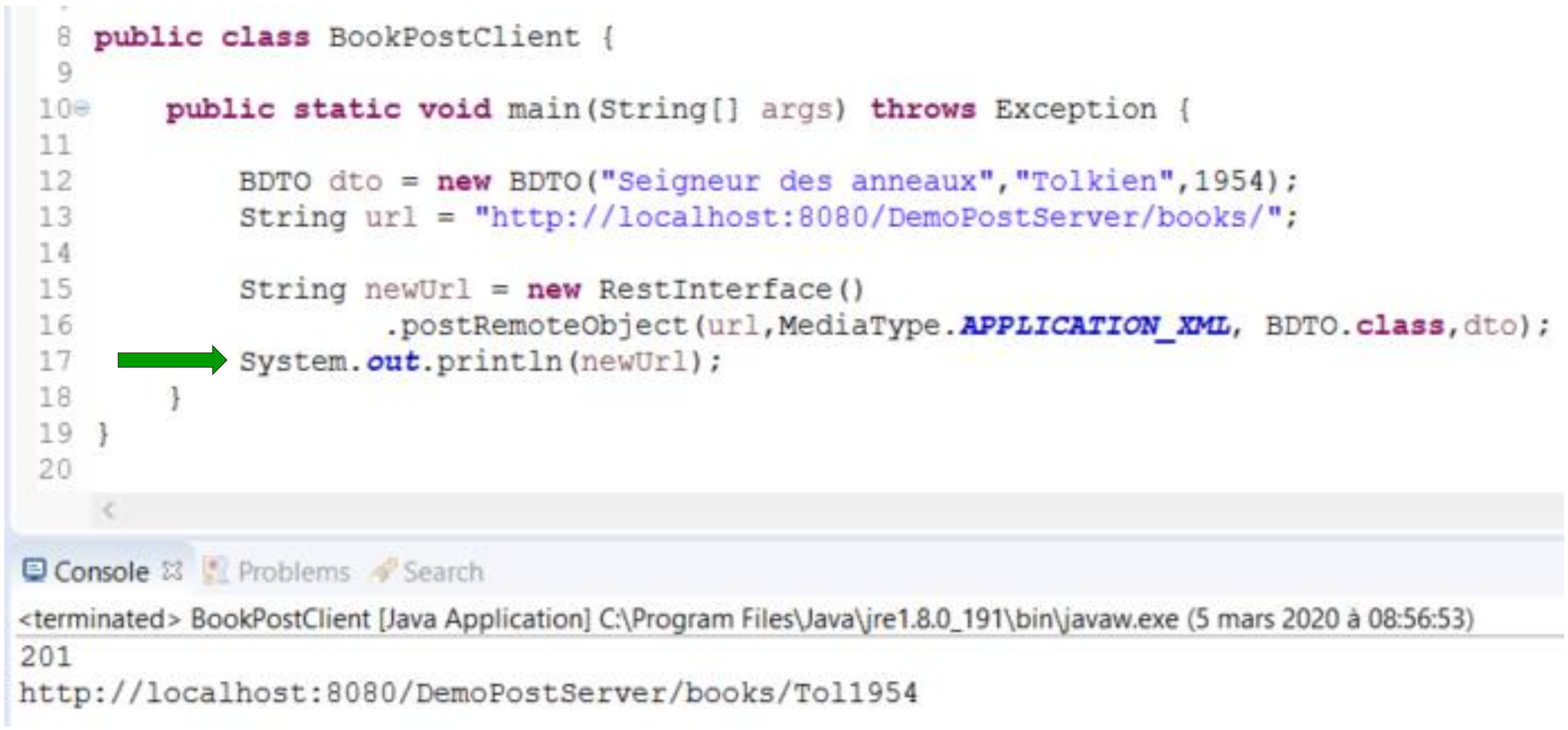
Services



```java
@Path("/")
public class EntryPoint {

    @POST
    @Path("books")
    @Consumes(MediaType.APPLICATION_XML)
    public void createBook(BDTO input,@Context HttpServletResponse response,
                                @Context UriInfo uriInfo)throws Exception{

        String baseURL = uriInfo.getBaseUri().toString();
        String auteur = input.getAuteur();
        String cote = auteur.substring(0,3)+input.getAnnee();
        //ici devrait se trouver l'insertion du livre dans la BD
        //ensuite on retourne l'URL du livre inséré
        response.setStatus(HttpServletResponse.SC_CREATED);
        response.setHeader(HttpHeaders.LOCATION, baseURL+"books/"+cote);
        try { response.flushBuffer(); }catch(Exception e){}
        }

    @GET
    @Path("books/{cote}")
    @Produces(MediaType.APPLICATION_ATOM_XML)
    public Entry<Book> getBook(@PathParam("cote") String cote,@Context
                                            UriInfo uriInfo)throws Exception{

        String baseURL = uriInfo.getBaseUri().toString();
        //ici devrait se placer la recherche du livre dans la BD à partir de la "cote"
        //simulation du livre trouvé, si  non trouvé on devrait retourner 404
        Book b = new Book("Seigneur des anneaux","Tolkien",1954,"Tol1954");
        return new Entry<Book>("book "+ cote,baseURL+"books/"+cote, b, true);
    }
}
```

h e g

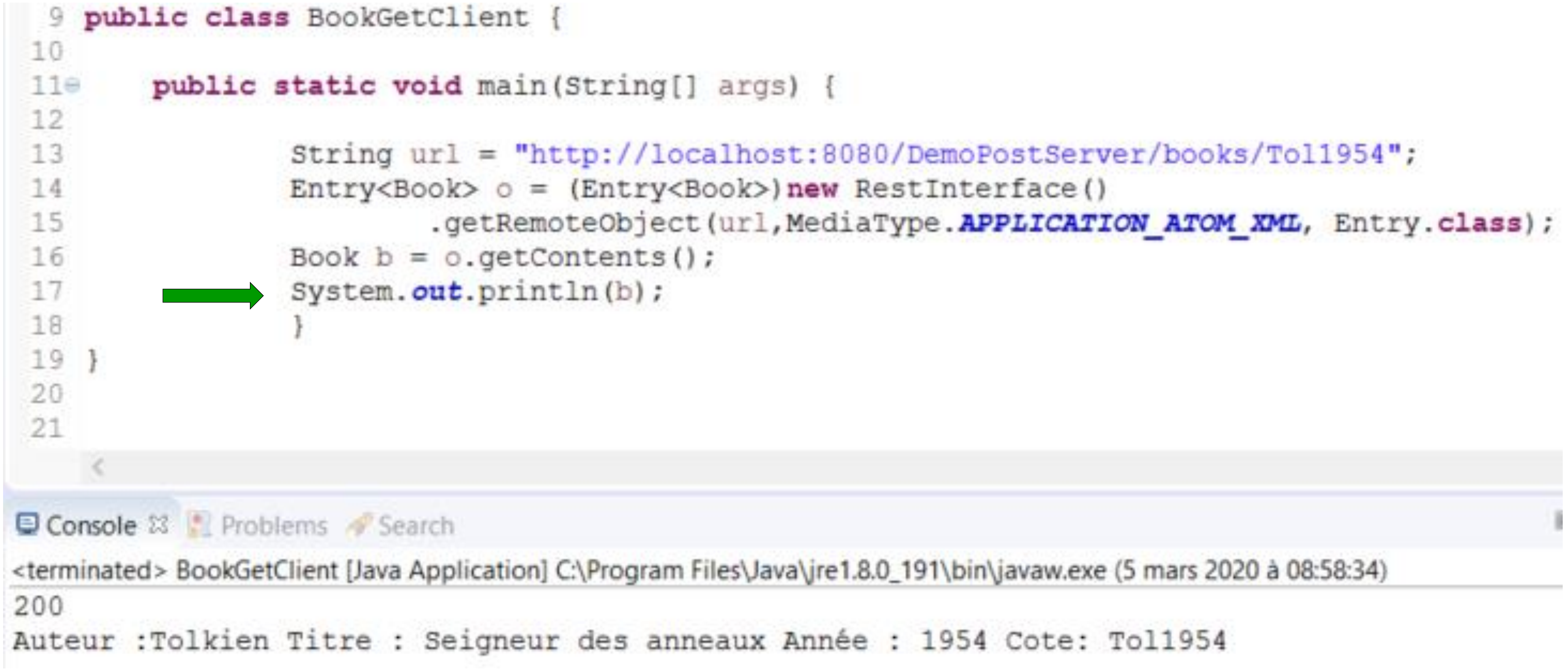
Test: resource creation


```java
public class BookPostClient {

    public static void main(String[] args) throws Exception {

        BDTO dto = new BDTO("Seigneur des anneaux","Tolkien",1954);
        String url = "http://localhost:8080/DemoPostServer/books/";

        String newUrl = new RestInterface()
                .postRemoteObject(url,MediaType.APPLICATION_XML, BDTO.class,dto);
        System.out.println(newUrl);
    }
}
```

Console Problems Search

```
<terminated> BookPostClient [Java Application] C:\Program Files\Java\jre1.8.0_191\bin\javaw.exe (5 mars 2020 à 08:56:53)
201
http://localhost:8080/DemoPostServer/books/Tol1954
```

Test: getting the resource



```
public class BookGetClient {

    public static void main(String[] args) {

            String url = "http://localhost:8080/DemoPostServer/books/Tol1954";
            Entry<Book> o = (Entry<Book>)new RestInterface()
                    .getRemoteObject(url,MediaType.APPLICATION_ATOM_XML, Entry.class);
            Book b = o.getContents();
            System.out.println(b);
            }
}
```

```
<terminated> BookGetClient [Java Application] C:\Program Files\Java\jre1.8.0_191\bin\javaw.exe (5 mars 2020 à 08:58:34)
200
Auteur :Tolkien Titre : Seigneur des anneaux Année : 1954 Cote: Tol1954
```

Demo

DemoPostServer

Communication “Meta patterns”

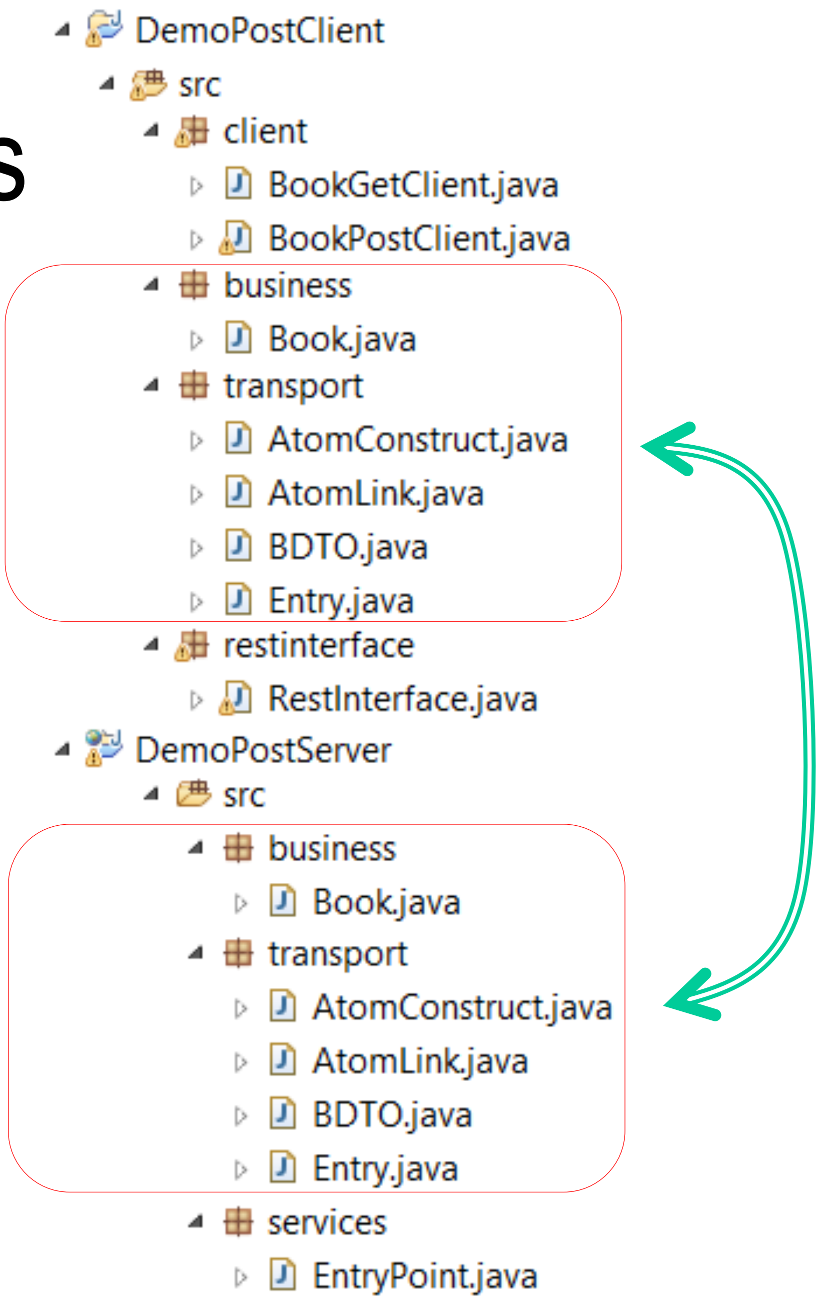
1. Payload format

- When communicating between client and server:
  - Business objects should be transported with annotated transport objects (Atom objects)
  - Business objects will not be annotated

2. Application communication layers

- When designing the client and servers application layers Duplicate the common layers : business object and transport in both the client and the server source code to make sure they will be the same.

Layers



```
DemoPostClient
  src
    client
      BookGetClient.java
      BookPostClient.java
    business
      Book.java
    transport
      AtomConstruct.java
      AtomLink.java
      BDTO.java
      Entry.java
    restinterface
      RestInterface.java
DemoPostServer
    src
      business
        Book.java
      transport
        AtomConstruct.java
        AtomLink.java
        BDTO.java
        Entry.java
      services
        EntryPoint.java
```