JAX-RS

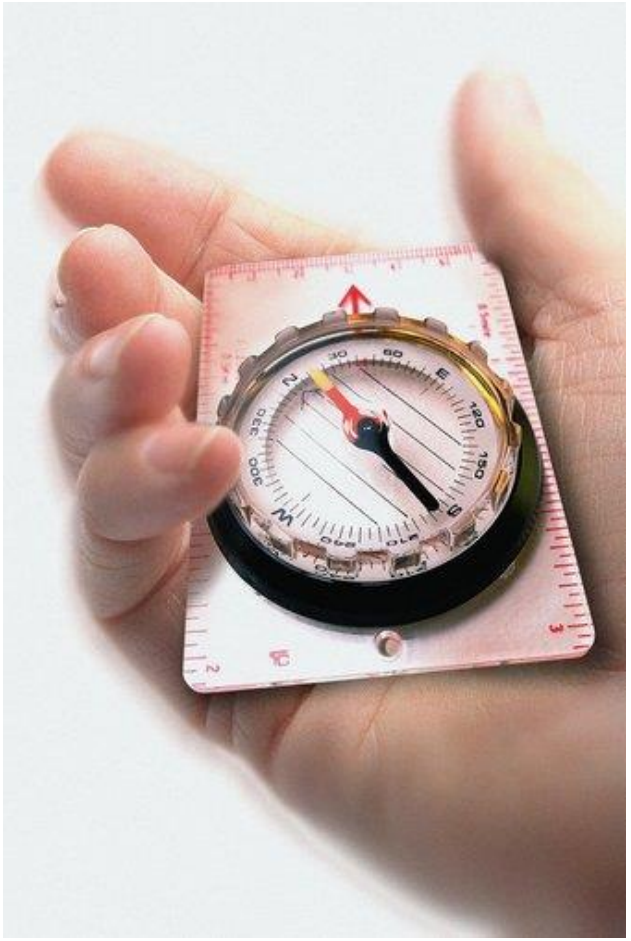
RESTful Web Services

Objectifs du cours

En complétant ce cours, vous serez en mesure de:

- Concevoir des services Web RESTful
- Utiliser l'API JAX-RS
- Convertissez facilement des JavaBeans en XML ou JSON

Plan de cours



- Rappels RESTful
- L'API JAX-RS
- JAX-RS avec JAXB

JAX-RS

RAPPELS RESTFUL

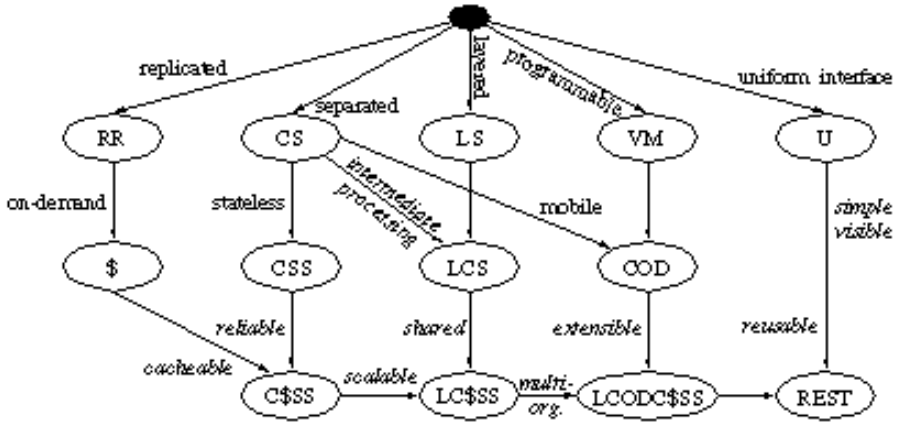


Figure 5-9. REST Derivation by Style Constraints

Définir le protocole HTTP

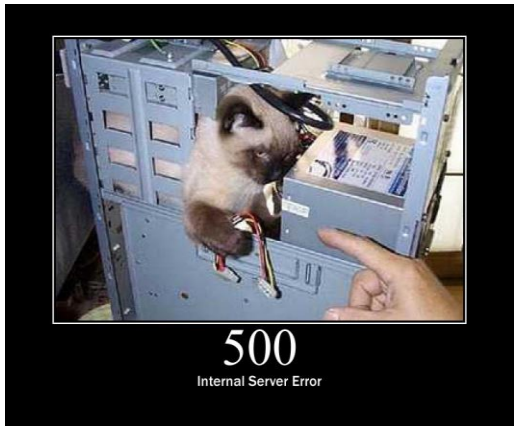
- **H**yper**T**ext **T**ransfer **P**rotocol
- Protocole de communication développé pour le Web
- Protocole de requête/réponse
- Stateless

Méthodes de requête HTTP

- HTTP définit neuf méthodes (ou verbes) :
  - **GET** : demande une représentation de la ressource
  - **POST** : soumet les données à traiter à la ressource identifiée
  - **PUT** : Uploads une représentation de la ressource spécifiée
  - **DELETE** : supprime la ressource spécifiée
  - …

Statuts HTTP

- Les statuts HTTP sont séparés en cinq catégories :
  - 1xx : Informationnel
  - 2xx : Succès
  - 3xx : Redirections
  - 4xx : erreur client
  - 5xx : Erreur de serveur



Contraintes REST

- Un système RESTful a les contraintes suivantes :
  - Il doit s'agir d'un système client-serveur
  - Il doit être stateless
  - Il doit prendre en charge un système de mise en cache
  - Il doit être uniformément accessible
  - Il doit être en couches (prise en charge de l'évolutivité)
  - Il peut être capable de transférer du code exécutable

Pour quoi est conçu REST ?

- Une ressource RESTful est tout ce qui est adressable sur le Web

- Des exemples de ressources :
  - La température à Paris à 20h00
  - Un article de blog
  - Une liste de rapports de bogues d'un BTS
  - Un résultat de recherche dans Google

Représentation

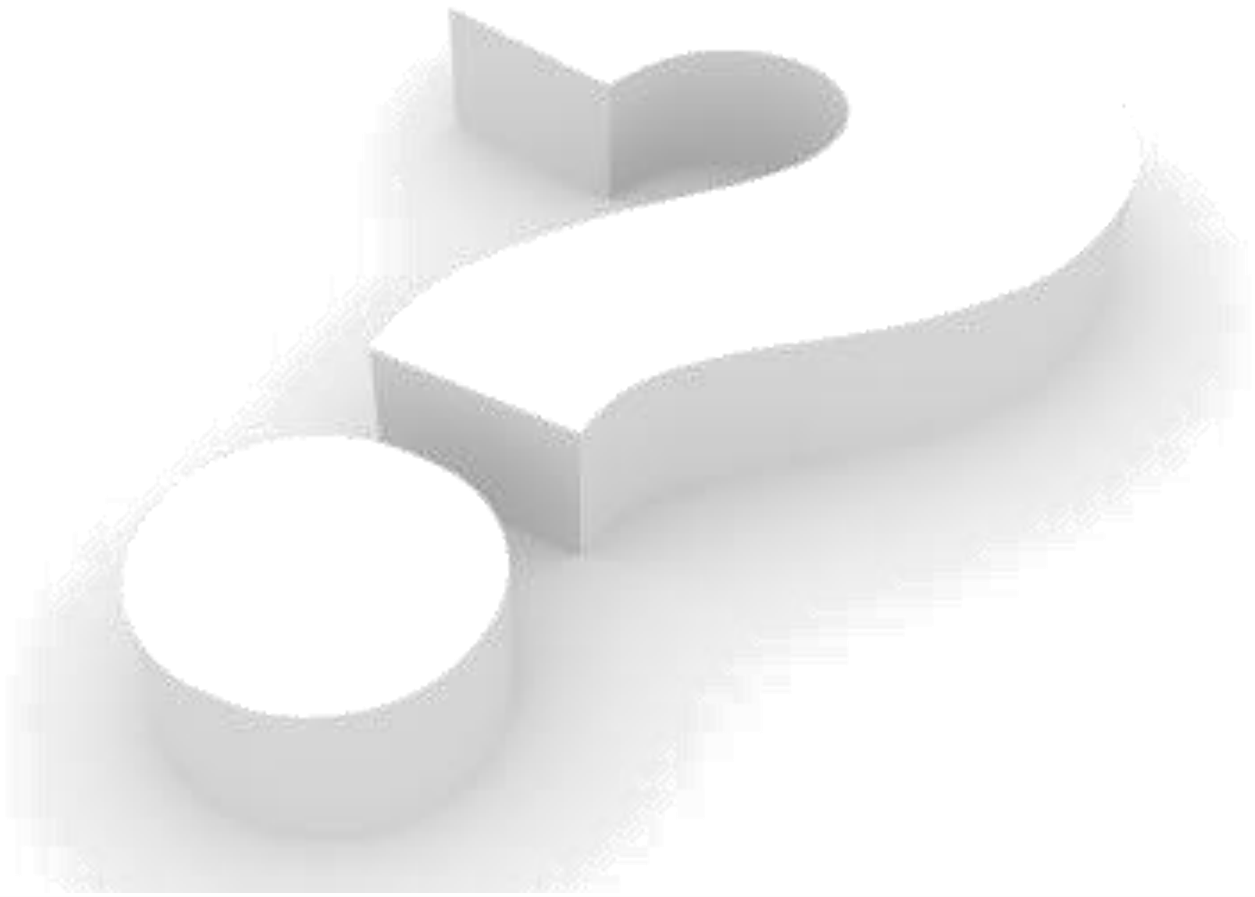
- Peut prendre diverses formes telles que:
  - HTML Document
  - Plain Text
  - XML Stream
  - JSON Stream
  - …
- Une ressource peut avoir plusieurs représentations
  - Mais avec le même URI

Verbes HTTP avec REST

- Le protocole HTTP fournit des méthodes sur lesquelles nous pouvons mapper les opérations CRUD
  - Et donc appliquer différentes actions avec la même URI !

Data Action	HTTP protocol equivalent
CREATE	POST
READ	GET
UPDATE	PUT
DELETE	DELETE

Questions ?



JAX-RS

JAX-RS API

Présentation

- JAX-RS est l'une des spécifications Java EE
  - Une API dédiée pour concevoir des Web Services REST !
- Comme toutes les spécifications Java EE, plusieurs implémentations sont disponibles :
  - Apache CXF, une bibliothèque de services Web open source
  - Jersey, implémentation de référence d'Oracle
  - RESTEasy, l'implémentation de JBoss

Présentation

- Pour ce cours, nous utiliserons Jersey !
  - Les librairies sont disponibles ici : https://eclipse-ee4j.github.io/jersey/download.html

- L'objectif de l'API JAX-RS est de faciliter le développement de services Web RESTful :
  - Juste besoin de développer des POJO…

Avantages

- Pas besoin de gérer la requête HTTP à bas niveau
- Concentration uniquement sur les règles métier !

- Nous allons voir les principales annotations fournies par JAX-RS…

Installation

- Pour utiliser Jersey, vous devez :
  - Inclure les bibliothèques dans votre projet
  - Définissez le servlet Jersey dans le descripteur de déploiement :
- Déclaration du mappage de servlet :

```
<servlet-mapping>
    <servlet-name>Jersey Web Application</servlet-name>
    <url-pattern>/resources/*</url-pattern>
</servlet-mapping>
```

Installation

- Déclaration de servlet :

```
<servlet>
    <servlet-name>Jersey Web Application</servlet-name>
    <servlet-class>
      com.sun.jersey.spi.container.servlet.ServletContainer
    </servlet-class>
    <init-param>
        <param-name>
          com.sun.jersey.config.property.packages
        </param-name>
        <param-value>com.cci.restful.rest</param-value>
    </init-param>
</servlet>
```

URI de la ressource

- Une ressource JAX-RS est une classe Java avec l'annotation @Path pour définir son URI

```
@Path("/students")
public class StudentsResource {
    ...
}
```

- Ici, votre ressource serrai à l'URI suivante :
  - http://your-website.com/resources/students

URI de la ressource

- Cette annotation peut également être utilisée pour définir des variables dans les URI :

```
@Path("/students/{idBooster}")
public class StudentsResource {
    ...
}
```

- Nous verrons plus tard comment récupérer la valeur de ces variables

Méthodes HTTP

- Pour déterminer quelle méthode Java appeler en fonction de la demande de méthode HTTP, JAX-RS fournit quatre annotations :
  - @GET
  - @POST
  - @PUT
  - @DELETE

Méthodes HTTP

- Annotations example:

```
@Path("/students")
public class StudentsResource {

    @GET
    public String handleGetRequest() { ... }

    @POST
    public String handlePostRequest(String payload) { ... }
}
```

Méthodes HTTP

- Le gestionnaire de méthodes POST et PUT peut avoir un paramètre de charge utile (payload) représentant les données du corps de la requête

- Les charges utiles peuvent avoir plusieurs types en fonction de l'en-tête de requête Content-Type
  - Par exemple, vous ne pouvez pas envoyer un fichier avec une chaîne de requête GET

Méthodes HTTP

- Types de contenu pris en charge :

Java Type	Content Type Supported
java.lang.String	*/*
byte[]	*/*
java.io.InputStream	*/*
java.io.Reader	*/*
javax.ws.rs.core. MultivaluedMap<String, String>	application/x-www-form-urlencoded
JAXB classes	text/xml, application/xml, application/json

Chemins relatifs dans les méthodes

- Parfois, il peut être utile de définir deux méthodes, chacune associée à une URI différente dans une même classe
  - Par exemple, une requête de recherche sur un objet

- Vous pouvez le faire en déclarant l'annotation @Path sur une méthode !

Chemins relatifs dans les méthodes

```
@Path("/students")
public class StudentsResource {

    @GET
    public String getStudentList() { ... }

    @GET @Path("/search/{query}")
    public String searchStudent() { ... }
}
```

- Dans ce cas, une requête GET sur **/students/search/Jack** exécutera la méthode **searchStudent()**

Variables d'URI

- Nous avons vu comment définir des variables avec l'annotation @Path dans la diapositive précédente…
  - … mais pas comment les récupérer dans une méthode Java

- Pour ce faire, JAX-RS définit une annotation @PathParam
  - L'annotation est placée entre parenthèses de fonction

Variables d'URI

- Vous pouvez le faire comme suit :

```
@GET @Path("/{idBooster}")
public String getStudent(@PathParam("idBooster") Long id) {
    ...
}

@GET @Path("/search/{query}")
public String searchStudent(@PathParam("query") String q) {
    ...
}
```

Formats d'entrée

- Appel la méthode en fonction de l'entête **HTTP Content-Type** de la requête client avec :
  - L'annotation *@Consume*

```
@Path("/students")
public class StudentResource {

    @POST @Consumes(MediaType.APPLICATION_XML)
    public String addXmlStudent(String payload) { ... }

    @POST @Consumes(MediaType.APPLICATION_JSON)
    public String addJsonStudent(String payload) { ... }
}
```

Formats de sortie

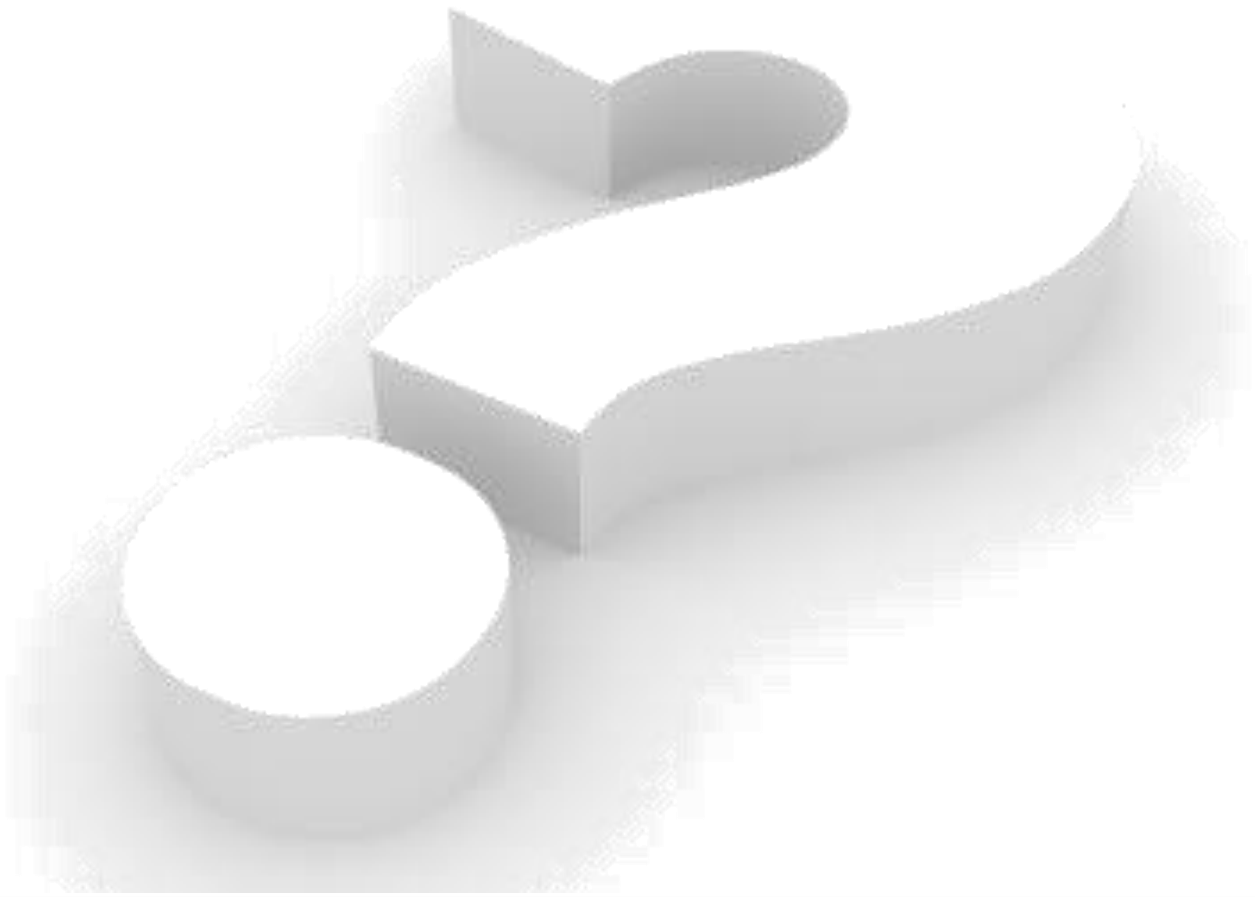
- Appel la méthode en fonction de l'en-tête **HTTP** **Accept** de la requête client grâce à:
  - *L'annotation @Produces*

```
@Path("/students")
public class StudentResource {

    @GET @Produces(MediaType.APPLICATION_XML)
    public String getStudentAsXml() { ... }

    @GET @Produces(MediaType.APPLICATION_JSON)
    public String getStudentAsJson() { ... }
}
```

Questions ?



Exercice (1/3)

- Créez un package **com.cci.supcommerce.rest**
- Créez une classe **ProductResource** avec les méthodes suivantes:
  - getAllProductsInXml()
  - getAllProductsInJson()
  - getProductInXml(Long productId)
  - getProductInJson(Long productId)
  - removeProduct(Long productId)

Exercice (2/3)

- Ajoutez les annotations correctes pour rendre ces méthodes RESTful Web Services en utilisant JAX-RS !
  - La représentation XML doit être comme ça :

```
<products>
    <product>
        <id>12</id>
        <name>Product1</name>
        <description>Description1</description>
        <price>123.00</price>
    </product>
    ...
</products>
```

Exercice (3/3)

- Ajoutez les annotations correctes pour rendre ces méthodes RESTful Web Services en utilisant JAX-RS !
  - La représentation JSON doit être comme ça:

```
{"products":[
    {
        "id":"12",
        "name":"Product1",
        "description":"Description1",
        "price":"123.00"
    }, {...}
]}
```

JAX-RS

JAX-RS AVEC JAX-B

Présentation

- JAXB pour **J**ava **A**rchitecture for **X**ML **B**inding
- Permet le mappage des classes Java aux représentations XML
- Deux caractéristiques principales :
  - **Marshal** objets java en XML
  - **Unmarshal** XML en objets Java

Présentation

- Très utile pour les Web Services !

- Ce cours ne concerne pas toutes les fonctionnalités de JAXB
  - Nous n'en verrons qu'une partie afin de faciliter le développement des Web Services REST !

Prise en charge de JAX-RS et JAXB

- Jersey fournit une intégration très utile de JAXB incluant les fonctionnalités suivantes :
  - Sérialiser les objets Java en tant que document JSON
  - Désérialiser le document JSON en objets Java

Prise en charge de JAX-RS et JAXB

- Jersey fournit une intégration très utile de JAXB incluant les fonctionnalités suivantes :
  - Désérialisation automatique des payload HTTP
    - Plus besoin de définir une méthode par type de consommation !
  - Sérialisation automatique des résultats de la méthode de ressource
    - Plus besoin de définir une méthode par type de produit !

Prise en charge de JAX-RS et JAXB

```
@Path("/students")
public class StudentResource {
    @GET @Path("/{idBooster}")
    public Student getStudent(
            @PathParam("idBooster") Long id) {
        Student student = ... //Retrieve Student in DB
        return student;
    }

    @POST
    public Response addStudent(Student student) {
        //Add the student in DB
        String studentUri = "/"+student.getIdBooster();
        return
           Response.created(
              URI.create(studentUri)).build();
    }
}
```

Annotations

- Depuis JAXB2, vous pouvez mapper une classe Java juste avec des annotations ! (ouais, encore des annotations…)
- Nous allons voir les principales :
  - @XmlRootElement
  - @XmlElement
  - @XmlAttribute

XMLRootElement

- Mappe une classe ou un type *enum* à un élément XML de niveau supérieur
- Par défaut, le nom de l'élément XML est le même que le nom de la classe
  - Modifiable avec l'attribut *name* de l'annotation

```
@XmlRootElement(name="doc")
public class Document {

}
```

XMLElement

- Mappe une propriété JavaBean à un élément XML dérivé du nom de la propriété
  - Cette annotation est facultative
  - Trois attributs d'annotation sont disponibles :
    - *name* : le nom de l'élément XML
    - *required* : spécifiez si l'élément est optionnel ou nullable
    - *defaultValue* : valeur par défaut de cet élément

```
@XmlElement(name = "first-name", required = true)
private String firstName;
```

XMLAttribute

- Mappe une propriété JavaBean à un attribut XML
  - Deux attributs d'annotation sont disponibles :
    - *name* : le nom de l'élément XML
    - *required* : spécifiez si l'élément est optionnel ou nullable

```
@XmlAttribute
private int score;
```

Exemple 1

```
@XmlRootElement
public class User {

    private String username;
    private String firstName;
    private String lastName;

    // Getters and Setters
}
```

Sera transformé dans le XML suivant :

Example 1

- JAX-RS avec JAX-B (Schema XML (XSD))

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="...">
  <xs:element name="user" type="user"/>
  <xs:complexType name="user">
    <xs:sequence>
      <xs:element name="firstName" type="xs:string"
            minOccurs="0"/>
      <xs:element name="lastName" type="xs:string"
            minOccurs="0"/>
      <xs:element name="username" type="xs:string"
            minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Exemple 2

```
@XmlRootElement
public class User {
    ... // Attributes
    @XmlAttribute(name="id-booster")
    public Long getIdBooster() { return idBooster; }

    @XmlElement(name="first-name")
    public String getFirstName() { return firstName; }

    @XmlElement(name="last-name")
    public String getLastName() { return lastName; }
    ... // Setters
}
```

Sera transformé dans le XML suivant :

Example 2

- JAX-RS avec JAX-B (Schema XML (XSD))

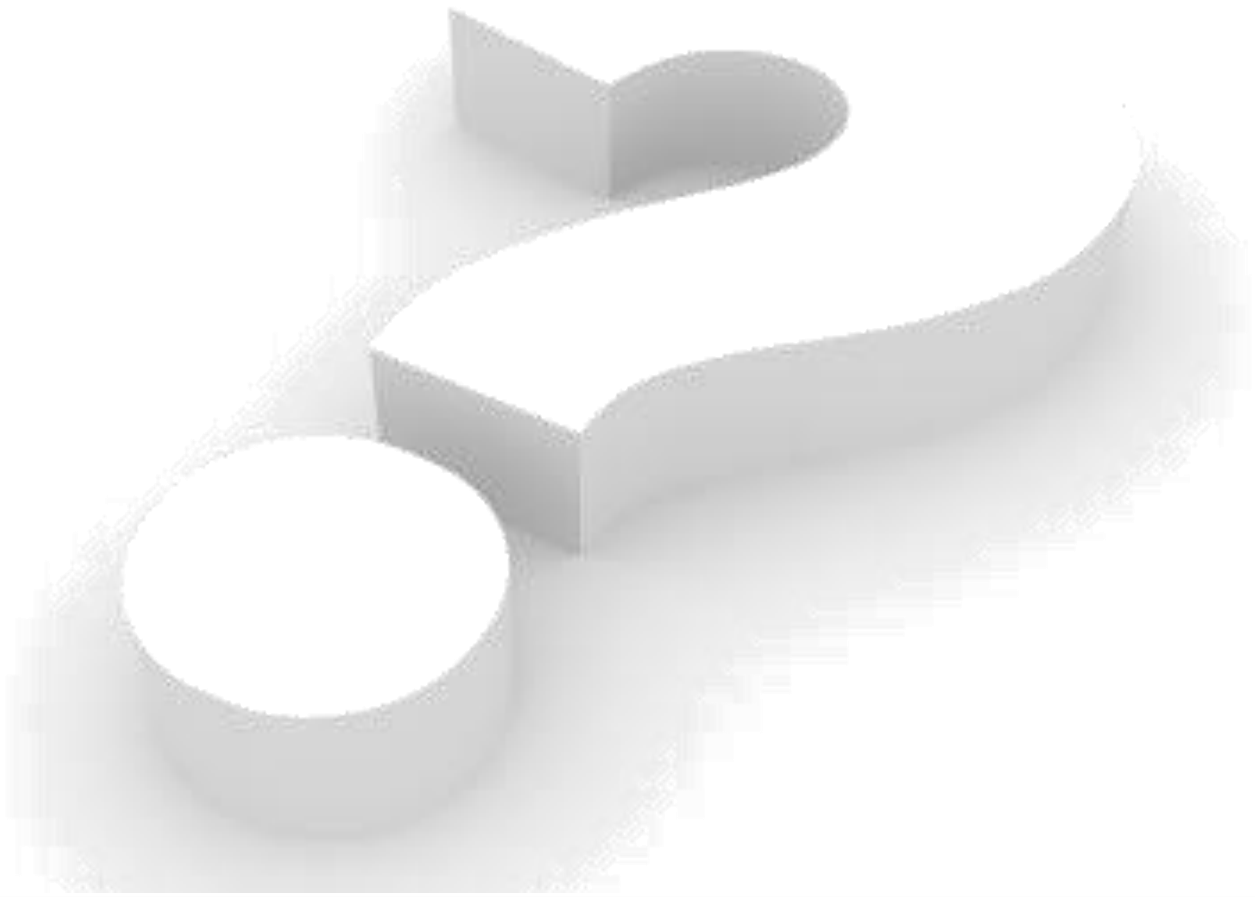
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="...">
  <xs:element name="user" type="user"/>
  <xs:complexType name="user">
    <xs:sequence>
      <xs:element name="first-name" type="xs:string"
          minOccurs="0"/>
      <xs:element name="last-name" type="xs:string"
          minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="id-booster" type="xs:long"/>
  </xs:complexType>
</xs:schema>
```

Example 2

- JAX-RS avec JAX-B
  - Document XML conforme à ce schéma :

```
<user id-booster="12345">
    <first-name>John</first-name>
    <last-name>Doe</last-name>
</user>
```

Questions ?



Exercice (1/2)

- Ajoutez les bibliothèques JAXB à votre application
- Mettez à jour vos entités avec les annotations JAXB dont vous avez besoin
- Refactorisez vos méthodes de service Web pour utiliser JAXB !

Exercice (2/2)

- Créez une nouvelle classe **CategoryResource** avec les méthodes suivantes :
  - addCategory(Category category)
  - getCategory (Long categoryId)
    - Doit retourner la catégorie avec tous ses produits
  - updateCategory(Category category)
- Ajoutez les annotations correctes pour rendre ces méthodes RESTful Web Services utilisant JAX-RS !

Fin

Merci de votre attention