Clients Web services REST avec Java (JAXB)

Hyacinthe MENIET

3 août 2019

Attention ! Depuis peu, Yahoo a arrêté de fournir ses web services **Yahoo ! Maps Web Services** et **Yahoo ! Search Web Services**. Toutefois, j'ai choisi de conserver ce tutoriel pour 3 raisons. Premièrement, je pense que l'arrêt de ces web services ne supprime pas l'intérêt didactique de ce tutoriel. Ensuite, le web service **Flickr Photos Web Services** est toujours disponible. Enfin, si vous étiez intéressé par l'intérêt opérationnel de ce tutoriel alors je vous conseillerais de vous tourner vers des solutions équivalentes que sont **Google Maps Geocoding API** et **Google Custom Search API**. Vous trouverez de nombreux Web services REST sur l'Internet. Bon nombre d'entre eux sont accessibles exclusivement en lecture seule. C'est à dire que vous ne pourrez dialoguez avec eux que via la méthode GET (et parfois POST), dans le but d'en consulter les données. En voici quelques exemples :

Web service	Description
Yahoo ! Maps Web Services	Donne la latitude et la longitude d'une adresse donnée (fonctionne au mieux avec les adresses des USA).
Yahoo ! Search Web Services	Extrait les expressions ou les mots significatifs d'un texte.
Flickr Photos Web Services	Permet de rechercher des photos sur Flickr à partir du photographe, du sujet ou d'autres critères.
eBay Shopping Web Services	Permet de rechercher dans les articles, les produits et les commentaires d'eBay
Amazon E-Commerce Service	Permet de rechercher dans le catalogue produit d'Amazon.

Dans ce document je présente les trois premiers.

1. Pré-requis

— Vous êtes familier de Java 8 et de sa syntaxe.
— Vous êtes familier des Web services REST.
— Vous disposez du JDK 8 minimum.
— Vous êtes familier de Maven et disposez de sa version 3.3 minimum

2. Vue d'ensemble

2.1 Vue d'ensemble de REST

Je présente brièvement les fondamentaux de REST dans Web services REST avec Java (JAXB).

2.2 Vue d'ensemble de l'article

Dans la pratique la plupart des Web services n'utilisent que les méthodes GET et POST ceci parce qu'une part importante des navigateurs ne supportent que ces deux méthodes. Toujours dans la pratique, pour consommer un Web service vous aurez rarement des URI de la forme : http://localhost:8080/weatherws/cities/307 ce sera plutôt http://localhost:8080/weatherws/cities?id=307. Ce document sera donc dédié à la consommation de Web service réels (e.g. Yahoo! Maps Web Services, Yahoo! Search Web Services et Flickr Photos Web Services). Pour la sérialisation et désérialisation du java en XML, j'utiliserai JAXB et DOM.

3. Le client pour Yahoo ! Maps Web Services

Yahoo ! propose un Web service de géolocalisation assez facile à consommer : le Yahoo! Maps Web Services . Visitez-le pour avoir la liste exhaustive des paramètres qu'il accepte et ainsi l'utiliser de manière optimale. Yahoo ! Maps Web Services associe à une adresse donnée sa longitude et sa latitude. Pour suivre les gros consommateurs de son service, Yahoo ! exige de chacun un identifiant. Vous pouvez vous en procurer un à cette adresse sinon utiliser celui du didacticiel (e.g **YahooDemo**). Notez que chaque adresse IP est soumise à un quota de 50 000 requêtes par jour.
Un exemple sera plus parlant. Interrogez l'application depuis votre navigateur avec l'adresse du Metropolitan Museum of Art à New-York.
Pour consommer ce Web service vous aurez besoin de trois choses :

- Un groupe de classes qui prendront en charge la désérialisation du XML en Java. C'est dans cette optique que Yahoo ! indique dans le flux XML l'URI vers le schéma XML associé (e.g. http://api.local.yahoo.com/MapsService/V1/GeocodeResponse.xsd)
- Un wrapper du Yahoo ! Maps Web Services qui fournira une couche d'abstraction du Web service pour le client.
- Une classe java munie d'un **main** qui servira de client java pour le Yahoo ! Maps Web Services.

Créez un nouveau projet de type jar avec Maven (commande à taper à la racine de votre workspace) :

```
$ mvn archetype:generate -DgroupId=net.dotmyself.restclient.
    yahoomaps -DartifactId=restclient-yahoomaps -Dversion=1.0 -
    Dpackage=net.dotmyself.restclient.yahoomaps -
    DarchetypeArtifactId=maven-archetype-quickstart -
    DinteractiveMode=false
```

Ecrasez le pom.xml généré, dans le dossier restclient-yahoomaps, par ce pom.xml.
Générez les classes utiles à la désérialisation via la commande :

```
$ xjc -d src/main/java -p 'net.dotmyself.restclient.yahoomaps'
    http://api.local.yahoo.com/MapsService/V1/GeocodeResponse.xsd
parsing a schema...
compiling a schema...
src/main/java/net/dotmyself/restclient/yahoomaps/ObjectFactory.
    java
src/main/java/net/dotmyself/restclient/yahoomaps/ResultSet.java
```

```
src/main/java/net/dotmyself/restclient/yahoomaps/ResultType.java
src/main/java/net/dotmyself/restclient/yahoomaps/package-info.
    java
```

Quelques explications :

- L'option -d permet d'indiquer le dossier dans lequel seront générées les classes. Cette commande est à taper à la racine du projet Maven, c'est pourquoi j'indique comme destination le dossier des sources java.
- L'option -p permet d'indiquer le package dans lequel disposer les classes Java (e.g. `net.dotmyself.res`

Créez le wrapper (GeocodeMapper.java) :

```
package net.dotmyself.restclient.yahoomaps;
import java.io.UnsupportedEncodingException;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLEncoder;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
/**
* A class designed to wrap the Yahoo! Geocode service.
*
* @author Hyacinthe MENIET
*/
public class GeocodeMapper {
private static final String BASE_URL = "http://api.local.yahoo.
   com/MapsService/V1/geocode";
private static final String APP_ID = "YahooDemo"; //replace with
   your own code
private static final String URL_ENCODING = "UTF-8";
/**
* Retrieves the location of an address.
*
* @param street The street address you are searching for.
* @param city The city for the location you are searching for.
* @param state The US state (if applicable) you are searching for
   .
* @param zip The US ZIP code (if applicable) you are searching
   for.
* @return list of {@link ResultType}
*/
public static List<ResultType> getResultSet(String street, String
    city, String state, String zip) {
List<ResultType> results = null;
try {
```

```
StringBuilder sb = new StringBuilder(BASE_URL);
sb.append("?appid=");
sb.append(URLEncoder.encode(APP_ID, URL_ENCODING));
if (street != null && street.length() > 1) {
sb.append("&street=");
sb.append(URLEncoder.encode(street, URL_ENCODING));
}
if (city != null && city.length() > 1) {
sb.append("&city=");
sb.append(URLEncoder.encode(city, URL_ENCODING));
}
if (state != null && state.length() > 1) {
sb.append("&state=");
sb.append(URLEncoder.encode(state, URL_ENCODING));
}
if (zip != null && zip.length() > 1) {
sb.append("&zip=");
sb.append(URLEncoder.encode(zip, URL_ENCODING));
}
ResultSet rs = extractLocation(new URL(sb.toString()));
results = (rs != null) ? rs.getResult() : null;
} catch (UnsupportedEncodingException | MalformedURLException ex)
    {
Logger.getLogger(GeocodeMapper.class.getName()).log(Level.SEVERE,
    null, ex);
}
return results;
}
/**
* Retrieves the location of an address.
*
* @param location A free form field of address information.
* @return list of {@link ResultType}
*/
public static List<ResultType> getResultSet(String location) {
List<ResultType> results = null;
try {
StringBuilder sb = new StringBuilder(BASE_URL);
sb.append("?appid=");
sb.append(URLEncoder.encode(APP_ID, URL_ENCODING));
if (location != null && location.length() > 1) {
sb.append("&location=");
sb.append(URLEncoder.encode(location, URL_ENCODING));
}
ResultSet rs = extractLocation(new URL(sb.toString()));
results = (rs != null) ? rs.getResult() : null;
} catch (UnsupportedEncodingException | MalformedURLException ex)
    {
Logger.getLogger(GeocodeMapper.class.getName()).log(Level.SEVERE,
```

```
    null, ex);
}
return results;
}
/**
* Uses the JAXB classes to process the returned XML and returns
    the
* corresponding {@link ResultSet}
*
* @param url The URL for the query.
*/
private static ResultSet extractLocation(URL url) {
ResultSet result = null;
try {
JAXBContext context = JAXBContext.newInstance("net.dotmyself.
    restclient.yahoomaps");
Unmarshaller unmarshaller = context.createUnmarshaller();
result = (ResultSet) unmarshaller.unmarshal(url);
} catch (JAXBException ex) {
Logger.getLogger(GeocodeMapper.class.getName()).log(Level.SEVERE,
    null, ex);
}
return result;
}
}
```

Créez le client Web service (GeocodeWSClient.java) :

```
package net.dotmyself.restclient.yahoomaps;
import java.io.StringReader;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
/**
*
* Web services Client which calls the Yahoo's Geocoding service.
*
* @author Hyacinthe MENIET
*/
public class GeocodeWSClient {
/**
* The main method.
*
* @param args The XML representation of the address which you
    want to
* retrieve longitude and latitude.
```

```
*/
public static void main(String[] args) {
try {
if (args == null || args.length < 1) {
throw new IllegalArgumentException("You must indicate the address
    you want to geocode.");
}
JAXBContext context = JAXBContext.newInstance("net.dotmyself.
   restclient.yahoomaps");
Unmarshaller unmarshaller = context.createUnmarshaller();
// processes the given XML into a ResultType
ResultType geocode = ((ResultSet) unmarshaller.unmarshal(
new StringReader(args[0]))).getResult().get(0);
// calls the web service
List<ResultType> listOfResultType = GeocodeMapper.getResultSet(
   geocode.getAddress(),
geocode.getCity(), geocode.getState(), geocode.getZip());
// Displays results
for (ResultType resultType : listOfResultType) {
System.out.println("precision=" + resultType.getPrecision() + ","
+ "Latitude=" + resultType.getLatitude() + ","
+ "Longitude=" + resultType.getLongitude() + ","
+ "Country=" + resultType.getCountry());
}
} catch (JAXBException ex) {
Logger.getLogger(GeocodeWSClient.class.getName()).log(Level.
   SEVERE, null, ex);
}
}
}
```

Après génération du jar, le résultat de l'exécution :

```
java -cp restclient-yahoomaps-1.0.jar net.dotmyself.restclient.
   yahoomaps.GeocodeWSClient "<?xml version=\"1.0\" encoding=\"
   UTF-8\"?><ResultSet xmlns=\"urn:yahoo:maps\"><Result><Address
   >1000 Fifth Avenue</Address><City>New York</City><State>NY</
   State></Result></ResultSet>"
precision=address,Latitude=40.778504,Longitude=-73.962653,Country
   =US
```

Le comment je récupère les paramètres pour les transmettre au Web service, n'est pas décisif. J'ai choisi de re-utiliser le schéma XML de Yahoo! parce les schémas d'entrée et de sortie sont très proches. De plus en procédant ainsi, je maximise l'utilisation du code généré. Mais rien ne vous empêche de diverger.

4. Le client pour Yahoo ! Search Web Services

Yahoo ! propose un deuxième Web service : Yahoo! Search Web Services. Visitez-le pour avoir la liste exhaustive des paramètres qu'il accepte et ainsi l'utiliser de manière optimale. Ce service permet d'extraire les phrases et les mots importants d'un texte. Il peut par exemple servir à catégoriser les articles d'un blog. Pour suivre les gros consomateurs de son service, Yahoo ! exige de chacun un identifiant. Vous pouvez vous en procurer un à cette adresse sinon utiliser celui du didacticiel (e.g. **YahooDemo**). Notez que chaque adresse IP est soumise à un quota de 50 000 requêtes par jour.
Pour s'affranchir de la limite des 2K des requêtes GET inhérente au navigateur MS Internet Explorer, ce Web service peut être appelé via la méthode POST. C'est d'ailleurs la voie que j'emprunterai dans le client Java. Pour varier les plaisirs.
Un exemple sera plus parlant. Interrogez l'application depuis votre navigateur avec cet extrait d'un billet de dotmyself.
Pour consommer ce Web service vous aurez besoin de trois choses :

- Un groupe de classes qui prendront en charge la désérialisation du XML en Java. C'est dans cette optique que Yahoo ! indique dans le flux XML l'URI vers le schéma XML associé (e.g. http://api.search.yahoo.com/ContentAnalysisService/V1/TermExtractionResponse.xsd)
- Un wrapper du Yahoo ! Search Web Services qui fournira une couche d'abstraction du Web service pour le client.
- Une classe java munie d'un **main** qui servira de client java pour le Yahoo ! Search Web Services.

Créez un nouveau projet de type jar avec Maven (commande à taper à la racine de votre workspace) :

```
$ mvn archetype:generate -DgroupId=net.dotmyself.restclient.
    yahoosearch -DartifactId=restclient-yahoosearch -Dversion=1.0
    -Dpackage=net.dotmyself.restclient.yahoosearch -
    DarchetypeArtifactId=maven-archetype-quickstart -
    DinteractiveMode=false
```

Ecrasez le pom.xml généré, dans le dossier restclient-yahoosearch, par ce pom.xml.
Générez les classes utiles à la désérialisation via la commande :

```
$ xjc -d src/main/java -p 'net.dotmyself.restclient.yahoosearch'
    http://api.search.yahoo.com/ContentAnalysisService/V1/
    TermExtractionResponse.xsd
parsing a schema...
compiling a schema...
src/main/java/net/dotmyself/restclient/yahoosearch/ObjectFactory.
    java
src/main/java/net/dotmyself/restclient/yahoosearch/ResultSet.java
src/main/java/net/dotmyself/restclient/yahoosearch/package-info.
    java
```

Quelques explications :

— L'option -d permet d'indiquer le dossier dans lequel seront générées les classes. Cette commande est à taper à la racine du projet Maven, c'est pourquoi j'indique comme destination le dossier des sources java.

— L'option -p permet d'indiquer le package dans lequel disposer les classes Java (e.g. `net.dotmyself.res`

Créez le wrapper (TermExtractionMapper.java) :

```
package net.dotmyself.restclient.yahoosearch;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.UnsupportedEncodingException;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.net.URLEncoder;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
/**
*
* A class designed to wrap the Yahoo! Term Extraction service.
*
* @author Hyacinthe MENIET
*/
public class TermExtractionMapper {
private static final String BASE_URL = "http://api.search.yahoo.
   com/ContentAnalysisService/V1/termExtraction";
private static final String APP_ID = "YahooDemo"; //replace with
   your own code
private static final String URL_ENCODING = "UTF-8";
/**
* Retrieves the important words and phrases in a block of text.
*
* @param context The block of text that the terms will be
   extracted from.
* @param query A query to help identify the topic of the context.
* @return List of extracted String or <code>null</code> if no
   String could
* be found.
*/
public static List<String> getTerms(String context, String query)
    {
List<String> terms = null;
try {
// builds de query string
StringBuilder sb = new StringBuilder("appid=");
sb.append(URLEncoder.encode(APP_ID, URL_ENCODING));
```

```
if (context != null && context.length() > 1) {
sb.append("&context=");
sb.append(URLEncoder.encode(context, URL_ENCODING));
}
if (query != null && query.length() > 1) {
sb.append("&query=");
sb.append(URLEncoder.encode(query, URL_ENCODING));
}
// perform a POST connection
URL url = new URL(BASE_URL);
URLConnection conn = url.openConnection();
conn.setDoOutput(true);
OutputStreamWriter writer = new OutputStreamWriter(conn.
    getOutputStream());
writer.write(sb.toString());
writer.flush();
// reads the response
JAXBContext jbctx = JAXBContext.newInstance("net.dotmyself.
    restclient.yahoosearch");
Unmarshaller unmarshaller = jbctx.createUnmarshaller();
ResultSet rs = (ResultSet) unmarshaller.unmarshal(conn.
    getInputStream());
terms = (rs != null) ? rs.getResult() : null;
} catch (UnsupportedEncodingException | MalformedURLException |
    JAXBException ex) {
Logger.getLogger(TermExtractionMapper.class.getName()).log(Level.
    SEVERE, null, ex);
} catch (IOException ex) {
Logger.getLogger(TermExtractionMapper.class.getName()).log(Level.
    SEVERE, null, ex);
}
return terms;
}
}
```

Créez le client Web service (TermExtractionWSClient.java) :

```
package net.dotmyself.restclient.yahoosearch;
import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
```

```
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
/**
* Web services Client which calls the Yahoo! Term Extraction
    service.
*
* @author Hyacinthe MENIET
*/
public class TermExtractionWSClient {
public static void main(String[] args) {
try {
if (args == null || args.length < 1) {
throw new IllegalArgumentException("You must indicate the text
    you want to extract important terms.");
}
// Reads and parse the given XML with DOM
DocumentBuilderFactory factory = DocumentBuilderFactory.
    newInstance();
DocumentBuilder constructor = factory.newDocumentBuilder();
ByteArrayInputStream str = new ByteArrayInputStream(args[0].
    getBytes("UTF-8"));
Document document = constructor.parse(str);
Element root = document.getDocumentElement();
String context = getTagValue("context", root);
String query = getTagValue("query", root);
// Displays results
List<String> terms = TermExtractionMapper.getTerms(context, query
    );
for (String term : terms) {
System.out.println(term);
}
} catch (ParserConfigurationException | SAXException |
    UnsupportedEncodingException ex) {
Logger.getLogger(TermExtractionWSClient.class.getName()).log(
    Level.SEVERE, null, ex);
} catch (IOException ex) {
Logger.getLogger(TermExtractionWSClient.class.getName()).log(
    Level.SEVERE, null, ex);
}
}
/**
* Returns the text content of the given tag.
*
* @param tag The tag.
* @param root The root Element.
* @return The String or <code>null</code> if no String could be
    found.
*/
```

```
private static String getTagValue(String tag, Element root) {
NodeList listOfTags = root.getElementsByTagName(tag);
if (listOfTags != null && listOfTags.getLength() > 0) {
Node node = listOfTags.item(0);
return node.getTextContent();
}
return null;
}
}
```

Après génération du jar, le résultat de l'exécution :

```
java -cp restclient-yahoosearch-1.0.jar net.dotmyself.restclient.
    yahoosearch.TermExtractionWSClient "<?xml version=\"1.0\"
    encoding=\"UTF-8\"?><request><context>Je pense qu'Internet
    represente une formidable opportunite pour les proprietaires
    des droits sur la musique. En effet, il permet d'augmenter la
    visibilite d'une oeuvre et indirectement celle de son auteur.
    Mecaniquement cela devrait produire plus de coup de coeur,
    voire d'achat</context><query>music</query></request>"
coup de coeur
formidable
```

Cette fois j'utilise en entrée du XML que je lis à la volée grâce à DOM dans TermExtractionWS-Client. Rien ne vous oblige à suivre cette voie.

5. Le client pour Flickr Photos Web Services

Terminons par Flickr qui fournit un service de recherche de photos, par auteur, par tag, par groupe et bien d'autres critères. Consultez le Flickr Photos Web Services pour avoir la liste des API. Comme Yahoo !, Flickr exige une clé pour utiliser son service. Je vous suggère d'en générer une depuis cette adresse.
Un exemple sera plus parlant. Interrogez l'application depuis votre navigateur avec une recherche sur `http://www.flickr.com/services/rest/?api_key={votre_clé}&method=flickr`
A partir des informations ci-dessus, vous pouvez inférer l'Url vers une photo. Les URL sont de la forme :

- http ://farm{farm}.static.flickr.com/{server}/{id}_{secret}.jpg (500 px)
- http ://farm{farm}.static.flickr.com/{server}/{id}_{secret}_m.jpg (240 px)
- http ://farm{farm}.static.flickr.com/{server}/{id}_{secret}_s.jpg (75 px)
- http ://farm{farm}.static.flickr.com/{server}/{id}_{secret}_t.jpg (La thumbnail - 100 px)
- http ://farm{farm}.static.flickr.com/{server}/{id}_{secret}_b.jpg (La large 1024 px)
- http ://farm{farm}.static.flickr.com/{server}/{id}_{secret}_o.jpg (la photo originale)

Pour consommer ce Web service vous aurez besoin de trois choses :

- Un groupe de classes qui prendront en charge la désérialisation du XML en Java. Flickr ne fournissant pas de schéma XML, je vais produire les classes java qui vont bien en me basant sur le résultat ci-dessus.

— Un wrapper du Flickr Photos Web Services qui fournira une couche d'abstraction du Web service pour le client.
— Une classe java munie d'un **main** qui servira de client java pour le Flickr Photos Web Services.

Créez un nouveau projet de type jar avec Maven (commande à taper à la racine de votre workspace) :

```
$ mvn archetype:generate -DgroupId=net.dotmyself.restclient.
    flickrphotos -DartifactId=restclient-flickrphotos -Dversion
    =1.0 -Dpackage=net.dotmyself.restclient.flickrphotos -
    DarchetypeArtifactId=maven-archetype-quickstart -
    DinteractiveMode=false
```

Ecrasez le pom.xml généré, dans le dossier restclient-flickrphotos, par ce pom.xml.
Créez la classe Photo qui est la structure de données associée à l'item **<photo>** (Photo.java) :

```
package net.dotmyself.restclient.flickrphotos;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;
/**
* @author Hyacinthe MENIET
*/
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "")
@XmlRootElement(name = "photo")
public class Photo {
@XmlAttribute
protected Integer farm;
@XmlAttribute
protected Integer id;
@XmlAttribute
protected Integer isfamily;
@XmlAttribute
protected Integer isfriend;
@XmlAttribute
protected Integer ispublic;
@XmlAttribute
protected String owner;
@XmlAttribute
protected String secret;
@XmlAttribute
protected Integer server;
@XmlAttribute
protected String title;
/**
* Gets the value of the farm property.
```

```
*
* @return possible object is {@link Integer }
*
*/
public Integer getFarm() {
return farm;
}
/**
* Sets the value of the farm property.
*
* @param value allowed object is {@link Integer }
*
*/
public void setFarm(Integer value) {
this.farm = value;
}
/**
* Gets the value of the id property.
*
* @return possible object is {@link Integer }
*
*/
public Integer getId() {
return id;
}
/**
* Sets the value of the id property.
*
* @param value allowed object is {@link Integer }
*
*/
public void setId(Integer value) {
this.id = value;
}
/**
* Gets the value of the isfamily property.
*
* @return possible object is {@link Integer }
*
*/
public Integer getIsfamily() {
return isfamily;
}
/**
* Sets the value of the isfamily property.
*
* @param value allowed object is {@link Integer }
*
*/
```

```
public void setIsfamily(Integer value) {
this.isfamily = value;
}
/**
* Gets the value of the isfriend property.
*
* @return possible object is {@link Integer }
*
*/
public Integer getIsfriend() {
return isfriend;
}
/**
* Sets the value of the isfriend property.
*
* @param value allowed object is {@link Integer }
*
*/
public void setIsfriend(Integer value) {
this.isfriend = value;
}
/**
* Gets the value of the ispublic property.
*
* @return possible object is {@link Integer }
*
*/
public Integer getIspublic() {
return ispublic;
}
/**
* Sets the value of the ispublic property.
*
* @param value allowed object is {@link Integer }
*
*/
public void setIspublic(Integer value) {
this.ispublic = value;
}
/**
* Gets the value of the owner property.
*
* @return possible object is {@link String }
*
*/
public String getOwner() {
return owner;
}
/**
```

```
* Sets the value of the owner property.
*
* @param value allowed object is {@link String }
*
*/
public void setOwner(String value) {
this.owner = value;
}
/**
* Gets the value of the secret property.
*
* @return possible object is {@link String }
*
*/
public String getSecret() {
return secret;
}
/**
* Sets the value of the secret property.
*
* @param value allowed object is {@link String }
*
*/
public void setSecret(String value) {
this.secret = value;
}
/**
* Gets the value of the server property.
*
* @return possible object is {@link Integer }
*
*/
public Integer getServer() {
return server;
}
/**
* Sets the value of the server property.
*
* @param value allowed object is {@link Integer }
*
*/
public void setServer(Integer value) {
this.server = value;
}
/**
* Gets the value of the title property.
*
* @return possible object is {@link String }
*
```

```
*/
public String getTitle() {
return title;
}
/**
* Sets the value of the title property.
*
* @param value allowed object is {@link String }
*
*/
public void setTitle(String value) {
this.title = value;
}
}
```

Créez la classe Photos qui est la structure de données associée à l'item **<photos>** (Photos.java) :

```
package net.dotmyself.restclient.flickrphotos;
import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;
/**
* @author Hyacinthe MENIET
*/
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {"photo"})
@XmlRootElement(name = "photos")
public class Photos {
@XmlElement(required = true)
protected List<Photo> photo;
@XmlAttribute
protected Integer page;
@XmlAttribute
protected Integer pages;
@XmlAttribute
protected Integer perpage;
@XmlAttribute
protected Integer total;
/**
* Gets the value of the photo property.
*
* This accessor method returns a reference to the live list, not
   a
* snapshot. Therefore any modification you make to the returned
```

```
    list will
* be present inside the JAXB object. This is why there is not a
* <CODE>set</CODE> method for the photo property.
*
* For example, to add a new item, do as follows: getPhoto().add(
    newItem);
*
*
* Objects of the following type(s) are allowed in the list {@link
     Photo }
*
*
* @return {@link List}
*/
public List<Photo> getPhoto() {
if (photo == null) {
photo = new ArrayList<>();
}
return this.photo;
}
/**
* Gets the value of the page property.
*
* @return possible object is {@link Integer }
*
*/
public Integer getPage() {
return page;
}
/**
* Sets the value of the page property.
*
* @param value allowed object is {@link Integer }
*
*/
public void setPage(Integer value) {
this.page = value;
}
/**
* Gets the value of the pages property.
*
* @return possible object is {@link Integer }
*
*/
public Integer getPages() {
return pages;
}
/**
* Sets the value of the pages property.
```

```
*
* @param value allowed object is {@link Integer }
*
*/
public void setPages(Integer value) {
this.pages = value;
}
/**
* Gets the value of the perpage property.
*
* @return possible object is {@link Integer }
*
*/
public Integer getPerpage() {
return perpage;
}
/**
* Sets the value of the perpage property.
*
* @param value allowed object is {@link Integer }
*
*/
public void setPerpage(Integer value) {
this.perpage = value;
}
/**
* Gets the value of the total property.
*
* @return possible object is {@link Integer }
*
*/
public Integer getTotal() {
return total;
}
/**
* Sets the value of the total property.
*
* @param value allowed object is {@link Integer }
*
*/
public void setTotal(Integer value) {
this.total = value;
}
}
```

Créez la classe Rsp qui est la structure de données associée à l'item **<rsp>** (Rsp.java) :

```
package net.dotmyself.restclient.flickrphotos;
import javax.xml.bind.annotation.XmlAccessType;
```

```
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;
/**
* @author Hyacinthe MENIET
*/
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {"photos"})
@XmlRootElement(name = "rsp")
public class Rsp {
@XmlElement
protected Photos photos;
@XmlAttribute
protected String stat;
/**
* Gets the value of the photos property.
*
* @return possible object is {@link Photos }
*
*/
public Photos getPhotos() {
return photos;
}
/**
* Sets the value of the photos property.
*
* @param value allowed object is {@link Photos }
*
*/
public void setPhotos(Photos value) {
this.photos = value;
}
/**
* Gets the value of the stat property.
*
* @return possible object is {@link String }
*
*/
public String getStat() {
return stat;
}
/**
* Sets the value of the stat property.
*
* @param value allowed object is {@link String }
*
*/
```

```
public void setStat(String value) {
this.stat = value;
}
}
```

Créez le wrapper (PhotoMapper.java) :

```
package net.dotmyself.restclient.flickrphotos;
import java.io.UnsupportedEncodingException;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLEncoder;
import java.text.MessageFormat;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
/**
* A class designed to wrap the Flickr Photos Web Services.
*
* @author Hyacinthe MENIET
*/
public class PhotoMapper {
private static final String BASE_URL = "http://www.flickr.com/
   services/rest/";
private static final String APP_ID = "your_own_code"; //replace
   with your own code
private static final String METHOD = "flickr.photos.search";
private static final String URL_ENCODING = "ISO-8859-1";
/**
* Retrieves photographs by tags.
*
* @param tags Small blocks of text attached to a photograph to
   identify its
* content.
* @return list of {@link Photo} or <code>null</code> if no {@link
    Photo}
* could be found.
*/
public static List<Photo> getPhotos(String tags) {
List<Photo> photoList = null;
try {
StringBuilder sb = new StringBuilder(BASE_URL);
sb.append("?api_key=");
sb.append(URLEncoder.encode(APP_ID, URL_ENCODING));
sb.append("&method=");
sb.append(URLEncoder.encode(METHOD, URL_ENCODING));
```

```
if (tags != null && tags.length() > 1) {
sb.append("&tags=");
sb.append(URLEncoder.encode(tags, URL_ENCODING));
}
Photos ps = extractPhotos(new URL(sb.toString()));
photoList = (ps != null) ? ps.getPhoto() : null;
} catch (UnsupportedEncodingException | MalformedURLException ex)
    {
Logger.getLogger(PhotoMapper.class.getName()).log(Level.SEVERE,
   null, ex);
}
return photoList;
}
/**
* Uses the JAXB classes to process the returned XML and returns
   the
* corresponding {@link Photos}
*
* @param url The URL for the query.
* @return {@link Photos} or <code>null</code> if no {@link Photos
   } could be
* found.
*/
private static Photos extractPhotos(URL url) {
Photos photos = null;
try {
JAXBContext context = JAXBContext.newInstance(Rsp.class);
Unmarshaller unmarshaller = context.createUnmarshaller();
Rsp rsp = (Rsp) unmarshaller.unmarshal(url);
photos = (rsp != null) ? rsp.getPhotos() : null;
} catch (JAXBException ex) {
Logger.getLogger(PhotoMapper.class.getName()).log(Level.SEVERE,
   null, ex);
}
return photos;
}
/**
* Creates the photographs thumbnail URL.
*
* @param photo The {@link Photo}
* @return The thumbnail URL.
*/
public static String buildThumbnailUrl(Photo photo) {
String url = null;
try {
if (photo == null) {
throw new NullPointerException("You should provide a valid Photo
   in order to ask to thumbnail URL");
}
```

```
url = MessageFormat.format("http://farm{0}.static.flickr.com
    /{1}/{2}_{3}_t.jpg",
new Object[]{
URLEncoder.encode(photo.getFarm().toString(), URL_ENCODING),
URLEncoder.encode(photo.getServer().toString(), URL_ENCODING),
URLEncoder.encode(photo.getId().toString(), URL_ENCODING),
URLEncoder.encode(photo.getSecret(), URL_ENCODING)
});
} catch (UnsupportedEncodingException ex) {
Logger.getLogger(PhotoMapper.class.getName()).log(Level.SEVERE,
    null, ex);
}
return url;
}
}
```

Créez le client Web service (PhotoWSClient.java) :

```
package net.dotmyself.restclient.flickrphotos;
import java.util.List;
/**
* Web services Client which calls the Flickr Photos Web Services.
*
* @author Hyacinthe MENIET
*/
public class PhotoWSClient {
/**
* The main method.
*
* @param args The tags.
*/
public static void main(String[] args) {
if (args == null || args.length < 1) {
throw new IllegalArgumentException("You must indicate the text
    you want to extract important terms.");
}
List<Photo> photos = PhotoMapper.getPhotos(args[0]);
for (Photo photo : photos) {
System.out.println(PhotoMapper.buildThumbnailUrl(photo));
}
}
}
```

Après génération du jar, le résultat de l'exécution :

```
java -cp restclient-flickrphotos-1.0.jar net.dotmyself.restclient
    .flickrphotos.PhotoWSClient travel
http://farm2.static.flickr.com/1081/1241102488_1cb8b1444b_t.jpg
http://farm2.static.flickr.com/1038/1241108964_262bc03ca2_t.jpg
```

```
[...]
http://farm2.static.flickr.com/1409/1240885274_648c2d3e43_t.jpg
```

Visualisez l'image de votre choix depuis votre navigateur pour valider que les Urls générées sont correctes.