TP n°3 : Développement de Web Services avec l'API JAX-RS

Le but de ce troisième TP est d'apprendre à manipuler l'API JAX-RS pour le développement de Services Web REST à partir de la plateforme de développement Java. Le TP insiste sur les des développements serveur et client d'un Service Web REST.

But pédagogique : Transformation d'une classe Java en Service Web REST, manipulation des annotations JAX-RS, génération de fichier WADL, utilisation de SOAP-UI pour invoquer un service REST à partir de WADL, utilisation de l'implémentation JERSEY, utilisation de l'API cliente de JERSEY.

Prérequis techniques :

- Java Development Kit 6 (jdk1.6uXX) téléchargeable sur à l'adresse URL suivante: http://www.oracle.com/technetwork/java/javase/downloads/index.html
- Eclipse IDE for Java EE developers v3.X téléchargeable à l'adresse URL suivante : http://www.eclipse.org/downloads/
- Apache Tomcat 6.XX téléchargeable à l'adresse URL suivante :
- http://tomcat.apache.org/download-60.cgi
- L'API JAX-RS JERSEY 1.10 téléchargeable à l'adresse URL suivante : http://jax-ws.java.net/2.2.5/
- L'application SOAP-UI téléchargeable à l'adresse URL suivante : http://www.soapui.org/soapUI-News/soapui-401-the-bug-fix-release.html

Exercice n°1 : Développer un Service Web REST : Interrogation et réservation de train

But : Développer un service web REST à partir d'une classe Java, utilisation d'un Sub-Resource Locator, mise en place d'un CRUD complet, déployer sur Tomcat et tester avec SOAP-UI.

Description : Le service Web REST de cet exercice consiste à créer un système CRUD pour l'interrogation et la réservation de trains. Les ressources manipulées par les services sont donc un train et une réservation. Le service Web REST doit pouvoir lister l'ensemble des trains, lister les trains qui satisfont un critère de recherche (ville de départ, ville d'arrivée, jour de départ et un intervalle de temps), de créer, de modifier, de supprimer et de lister une réservation pour un client donné. Nous intéresserons sur l'accessibilité des services et non sur le code métier.

Étapes à suivre :

1. Créer un nouveau projet « *File →New →Dynamic Web Project* » et le nommer « *BookTrainWebServiceRestExercice*1».
2. Créer un package nommé « *soa.jaxrslabs.booktrainwebservicerestexercice1* ».
3. Dans ce package, créer la classe « *Train.java* » qui modélise le concept de Train et qui contient un attribut « *String numTrain* » (identifiant du train), un attribut « *String*

villeDepart » (la ville de départ du train), un attribut « *String villeArrivee* » (la ville d'arrivée du train) et un attribut « *int heureDepart* » (l'heure de départ du train formaté 1230 = 12h30). Ajouter les getters/setters sur ces attributs.

4. Ajouter au-dessus de la déclaration de la classe, l'annotation suivante «*@XmlRootElement(name = "train")* ».
5. Créer une classe « *BookTrainBD .java* » qui sert à persister toutes les informations concernant le service Web de cet exercice. L'implémenter comme ci-dessous :

```
public class BookTrainBD {

	private static List<Train> trains = new ArrayList<Train>();

	static {
		trains.add(new Train("TR123", "Poitiers", "Paris", 1250));
		trains.add(new Train("TR127", "Poitiers", "Paris", 1420));
		trains.add(new Train("TR129", "Poitiers", "Paris", 1710));
	}

	public List<Train> getTrains() {
		return trains;
	}
}
```

6. Toujours dans le même package, créer une classe « *TrainRessource.java* » permettant l'accès aux services Web REST de la ressource Train. Définir comme chemin de ressource racine la valeur « *trains* » (via l'utilisation de l'annotation « *@Path* ») puis ajouter trois méthodes qui permettent respectivement de retourner la liste des trains, un train à partir de son identifiant et une recherche de trains par critères passés en paramètre de la requête (ville de départ, ville d'arrivée et heure de départ). Pour cette dernière méthode, le sous-chemin associé est « *search* ». Noter que le format de retour des services est du XML. Voici le squelette de cette classe sur laquelle il faut ajouter les annotations adéquates :

```java
@Path("trains")
public class TrainRessource {

	// TODO ajouter les annotations adéquates
	public List<Train> getTrains() {
		System.out.println("getTrains");
		return BookTrainBD.getTrains();
	}

	// TODO ajouter les annotations adéquates
	// Utiliser l'annotation @PathParam pour les paramètres
	public Train getTrain(String numTrain) {
		System.out.println("getTrain");
		for (Train currentTrain : BookTrainBD.getTrains()) {
			if (currentTrain.getNumTrain().equals(numTrain)) {
				return currentTrain;
			}
		}
		return null;
	}

	// TODO ajouter les annotations adéquates
	// Utiliser l'annotation @QueryParam pour les paramètres
	public List<Train> searchTrainsByCriteria(String departure, String arrival,
			int departureHour) {
		System.out.println("searchTrainsByCriteria");
		List<Train> foundTrains = new ArrayList<Train>();
		for (Train currentTrain : BookTrainBD.getTrains()) {
			if (currentTrain.getVilleDepart().equals(departure)
					&& currentTrain.getVilleArrivee().equals(arrival)
					&& currentTrain.getHeureDepart() == departureHour) {
				foundTrains.add(currentTrain);
			}
		}
		return foundTrains;
	}
}
```

7. Éditer le fichier « *web.xml* » et copier/coller le code suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
      id="WebApp_ID" version="2.5">
      <display-name>de.vogella.jersey.first</display-name>
      <servlet>
            <servlet-name>Jersey REST Service</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>soa.jaxrslabs.booktrainwebservicerestexercice1</param-value>
            </init-param>
            <load-on-startup>1</load-on-startup>
      </servlet>
      <servlet-mapping>
            <servlet-name>Jersey REST Service</servlet-name>
            <url-pattern>/*</url-pattern>
      </servlet-mapping>
</web-app>
```

8. Exécuter votre projet. Observer les résultats en copiant/collant les liens suivants :
   - http://localhost:8083/BookTrainWebServiceRestExercice1/trains
   - http://localhost:8083/BookTrainWebServiceRestExercice1/trains/TR123
   - http://localhost:8083/BookTrainWebServiceRestExercice1/trains/search?departure=Poitiers&arrival=Paris&departureHour=1710
9. Créer une classe « *BookTrain.java* » qui modélise le concept de réservation de Train et qui contient un attribut « *String numBook* » (identifiant fonctionnel de la réservation d'un train), un attribut « *Train currentTrain* » (association sur le train de la réservation) et un attribut « *int numberPlaces* » (nombre de places réservées). Ajouter les getters/setters sur ces attributs.
10. Ajouter au-dessus de la déclaration de la classe, l'annotation suivante «*@XmlRootElement(name = "booktrain")* ».
11. Compléter la classe « *BookTrainBD.java* » de façon à persister les informations concernant les services liés à la réservation de train. Voici le code complété :

```java
public class BookTrainBD {

	private static List<Train> trains = new ArrayList<Train>();

	private static List<BookTrain> bookTrains = new ArrayList<BookTrain>();

	static {
		trains.add(new Train("TR123", "Poitiers", "Paris", 1250));
		trains.add(new Train("TR127", "Poitiers", "Paris", 1420));
		trains.add(new Train("TR129", "Poitiers", "Paris", 1710));
	}

	public static List<Train> getTrains() {
		return trains;
	}

	public static List<BookTrain> getBookTrains(){
		return bookTrains;
	}
}
```

12. Créer la classe « *BookTrainResource.java* » permettant l'accès aux services Web REST de la ressource « *Réservation* ». Quatre méthodes sont à définir. La première « createBookTrain » est invoquée pour la création d'une ressource Réservation (méthode POST). La deuxième « getBookTrains » (méthode GET) est utilisée pour lister l'ensemble des réservations. La troisième « getBookTrain » (méthode GET) permet de retourner les informations d'une réservation à partir d'un numéro de réservation. Finalement « removeBookTrain » (méthode DELETE) permet de supprimer une réservation. Voici le code du squelette de cette classe dont il faudra ajouter les annotations :

```java
public class BookTrainRessource {

	// TODO Ajouter les annotations adéquates
	public String createBookTrain(String numTrain, int numberPlaces) {
		Train currentTrain = null;
		for (Train current : BookTrainBD.getTrains()) {
			if (current.getNumTrain().equals(numTrain)) {
				currentTrain = current;
			}
		}
```

```
            if (currentTrain == null) {
                  return "";
            }

            BookTrain newBookTrain = new BookTrain();
            newBookTrain.setNumberPlaces(numberPlaces);
            newBookTrain.setCurrentTrain(currentTrain);
            newBookTrain.setNumBook(Long.toString(System.currentTimeMillis()));
            BookTrainBD.getBookTrains().add(newBookTrain);

            return newBookTrain.getNumBook();
      }

      // TODO Ajouter les annotations adéquates
      public List<BookTrain> getBookTrains() {
            System.out.println("getBookTrains");
            return BookTrainBD.getBookTrains();
      }

      // TODO Ajouter les annotations adéquates
      // Compléter le chemin de façon à intégrer un template parameter (id)
      @Path("{id}")
      public BookTrain getBookTrain(@PathParam("id") String bookNumber) {
            List<BookTrain> bookTrains = BookTrainBD.getBookTrains();

            for (BookTrain current : bookTrains) {
                  if (current.getNumBook().equals(bookNumber)) {
                        return current;
                  }
            }
            return null;
      }

      // TODO Ajouter les annotations adéquates
      // Compléter le chemin de façon à intégrer un template parameter (id)
      public void removeBookTrain(@PathParam("id") String bookNumber) {
            BookTrain currentBookTrain = null;
            for (BookTrain current : BookTrainBD.getBookTrains()) {
                  if (current.getNumBook().equals(bookNumber)) {
```

```
                        currentBookTrain = current;
                    }
                }
                BookTrainBD.getBookTrains().remove(currentBookTrain);
        }
}
```

13. L'accès à la ressource Réservation (via la classe « *BookTrainRessource.java* ») est obtenu via l'utilisation d'un « *sub-resource locator* ». Compléter la classe « *TrainRessource.java* » en ajout la méthode suivante et en lui ajoutant l'annotation adéquate :

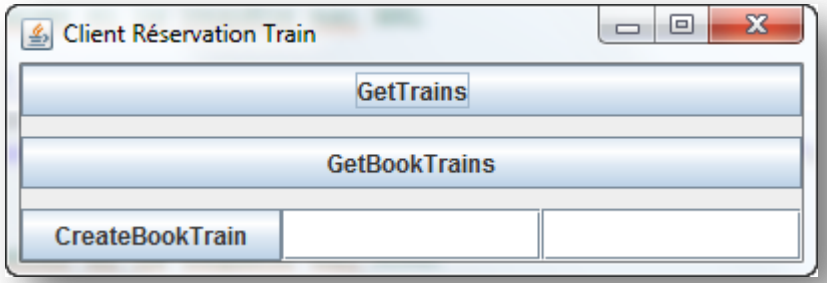
```
        // TODO Ajouter l'annotation adéquate
        // de telle sorte à définir le chemin suivant "/trains/booktrains"
        // pour invoquer cette méthode
        public BookTrainRessource getBookTrainRessource() {
                return new BookTrainRessource();
        }
```

14. Tester les différentes opérations avec le logiciel SOAP-UI dont il faudra spécifier le chemin du fichier WADL généré, qui est :
    - http://localhost:8083/BookTrainWebServiceRestExercice1/application.wadl

Exercice n°2 : Développer un client de Service Web REST : Interrogation et réservation de train.

But : Développer un client de service web REST à partir de l'API JERSEY, création des requêtes via le patron Builder.

Description : Ce troisième exercice se propose de fournir un client pour l'accès au service Web REST défini dans l'exercice n°1. Une interface graphique en Java/Swing permet de contrôler les appels aux différents services.



Étapes à suivre :

1. Créer un projet java client lourd « *File→New→Other→Java project* » et le nommer « *BookTrainClientWebServiceRestExercice3* ».
2. Créer un package nommé « *soa.jaxrslabs.booktrainclientwebservicerestexercice3* ».
3. Dans ce package, copier/coller les classes « Train.java » et « BookTrain.java » à partir du projet de l'exercice n°1.
4. Ajouter les bibliothèques Jersey au projet (penser à les définir dans le « *buildpath* » du projet).
5. Créer une classe « *BookTrainClientMain.java* ». La compléter, à partir du code source suivant :

```
public class BookTrainClientMain extends JFrame {


   private static final long serialVersionUID = -6298739485988198268L;

   private WebResource service;
```

```
public BookTrainClientMain() {
	super("Client Réservation Train");

	this.initializeService();
	this.setLayout(new GridLayout(3, 1, 10, 10));
	JPanel panelTrains = new JPanel();
	panelTrains.setLayout(new GridLayout(1, 1));

	JButton getTrains = new JButton("GetTrains");
	panelTrains.add(getTrains);
	getTrains.addActionListener(new ActionListener() {
		public void actionPerformed(ActionEvent e) {
			callGetTrains();
		}
	});

	JPanel panelBookTrains = new JPanel();
	panelBookTrains.setLayout(new GridLayout(1, 1));

	JButton getBookTrains = new JButton("GetBookTrains");
	panelBookTrains.add(getBookTrains);
	getBookTrains.addActionListener(new ActionListener() {
		public void actionPerformed(ActionEvent e) {
			callGetBookTrains();
		}
	});
	JPanel createBookTrainFormPanel = new JPanel();
	createBookTrainFormPanel.setLayout(new GridLayout(1, 3));
	final JTextField numTrain = new JTextField();
	final JTextField numberPlaces = new JTextField();
	JButton createBookTrains = new JButton("CreateBookTrain");
	createBookTrains.addActionListener(new ActionListener() {
		public void actionPerformed(ActionEvent e) {
			createBookTrains(numTrain.getText(), numberPlaces.getText());
		}
	});
	createBookTrainFormPanel.add(createBookTrains);
	createBookTrainFormPanel.add(numTrain);
	createBookTrainFormPanel.add(numberPlaces);
```

```
        this.add(panelTrains);
        this.add(panelBookTrains);
        this.add(createBookTrainFormPanel);

        this.pack();
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.setVisible(true);
    }

    private void initializeService() {
        // TODO à compléter
    }

    private void callGetTrains() {
        // TODO à compléter
    }

    private void callGetBookTrains() {
        // TODO à compléter
    }

    private void createBookTrains(String text, String text2) {
        // TODO à compléter
    }

    public static void main(String[] args) {
        new BookTrainClientMain();
    }
}
```

6. La méthode « *initializeService()* » permet d'initialiser l'accès au service Web REST. C'est à l'intérieure de celle-ci que l'attribut « *service* » devra être initialisé pour être utilisé dans les prochaines méthodes.
7. La méthode « *callGetTrains()* » permet de lister l'ensemble des trains (l'affichage de la liste ne se fera qu'en console via l'utilisation de System.out.println()).
8. La méthode « *createBookTrains(String numTrain, String numberPlaces)* » permet de créer une réservation d'un train. Dans cette méthode, il faudra afficher dans la console le numéro de réservation retourné par le service de création.
9. La méthode « *callGetBookTrains()* » permet de récupérer l'ensemble des réservations qui ont été créées. Il faudra aussi les afficher dans la console.

10. Exécuter le programme en vous assurant que le service Web REST défini dans l'exercice n°1 est déployé et opérationnel.