Atelier Framework Spring-05
Service Web REST - Partie02

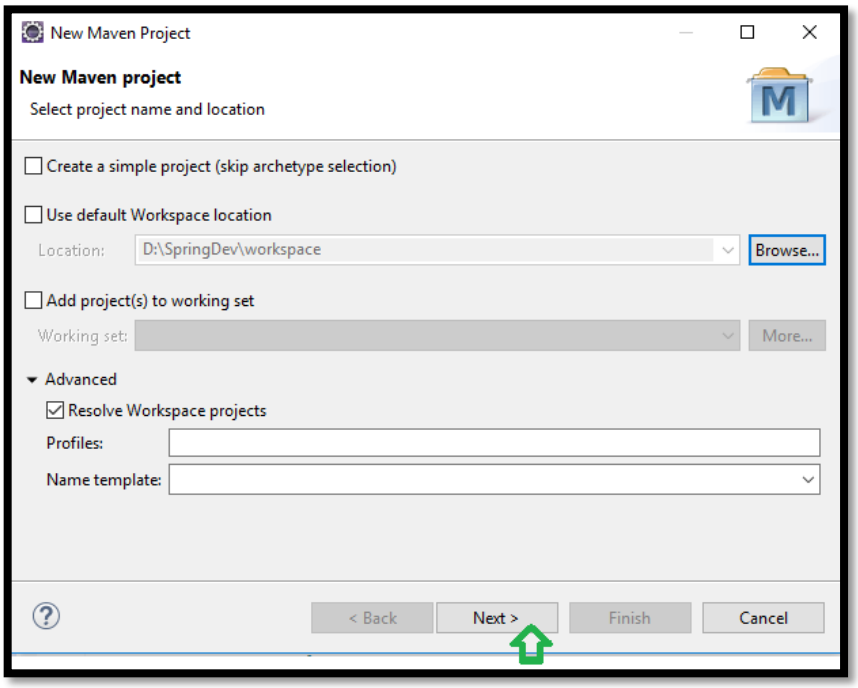
Objectifs

Consommer un Service Web REST avec un client JAVA

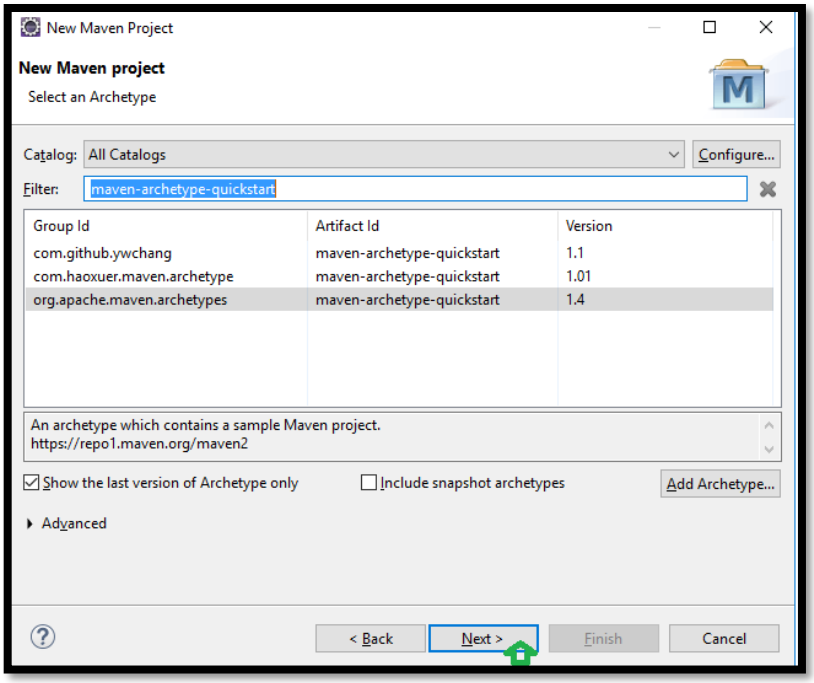
1. **Déclarer les dépendances nécessaires pour utiliser JERSEY**
2. **Utiliser la classe WebResource**
   - Appeler des méthodes (GET, POST, PUT, DELETE)
   - Configurer les paramètres de la requête
3. **Utiliser l'API « Gson » de Google**
   - Convertir une représentation JSON en objet JAVA
   - Convertir une représentation JSON en un tableau d'objets JAVA

A. Créer le projet client JAVA avec Maven

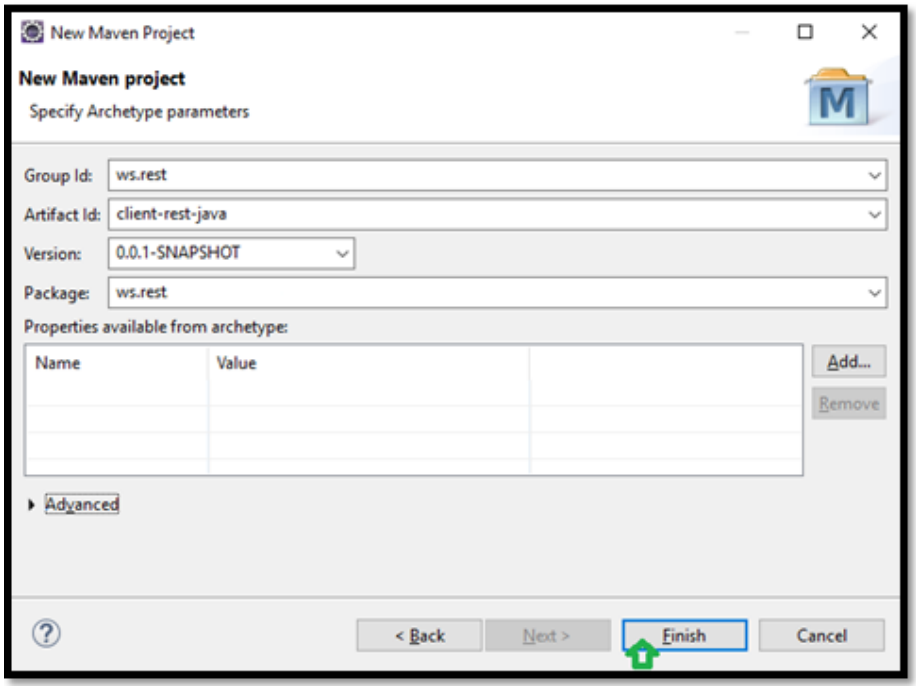
1. Créer un projet **MAVEN**



2. Choisir le modèle **maven-archetype-quickstart :**



3. Spécifier les caractéristiques suivantes pour créer un projet client REST JAVA :
   - groupId : **ws.rest**
   - artifactId : **client-rest-java**
   - version : **1.0-SNAPSHOT**
   - package **:ws.rest**



4. Editer le fichier «**pom.xml**» pour déclarer les dépendances MAVEN suivantes pour l'utilisation de l'implémentation JERSEY (implémentation de l'API **JAX-RS** de JAVA) :

```
  <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.1.0</version>
  <scope>provided</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/asm/asm -->
<dependency>
  <groupId>asm</groupId>
  <artifactId>asm</artifactId>
  <version>3.3.1</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.sun.jersey/jersey-bundle -->
<dependency>
  <groupId>com.sun.jersey</groupId>
  <artifactId>jersey-bundle</artifactId>
  <version>1.19.2</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.json/json-->
<dependency>
  <groupId>org.json</groupId>
  <artifactId>json</artifactId>
  <version>20160810</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.sun.jersey/jersey-server -->
<dependency>
  <groupId>com.sun.jersey</groupId>
  <artifactId>jersey-server</artifactId>
  <version>1.19.2</version>
</dependency>
```

```xml
<!-- https://mvnrepository.com/artifact/com.sun.jersey/jersey-core -->
<dependency>
  <groupId>com.sun.jersey</groupId>
  <artifactId>jersey-core</artifactId>
  <version>1.19.2</version>
</dependency>
```

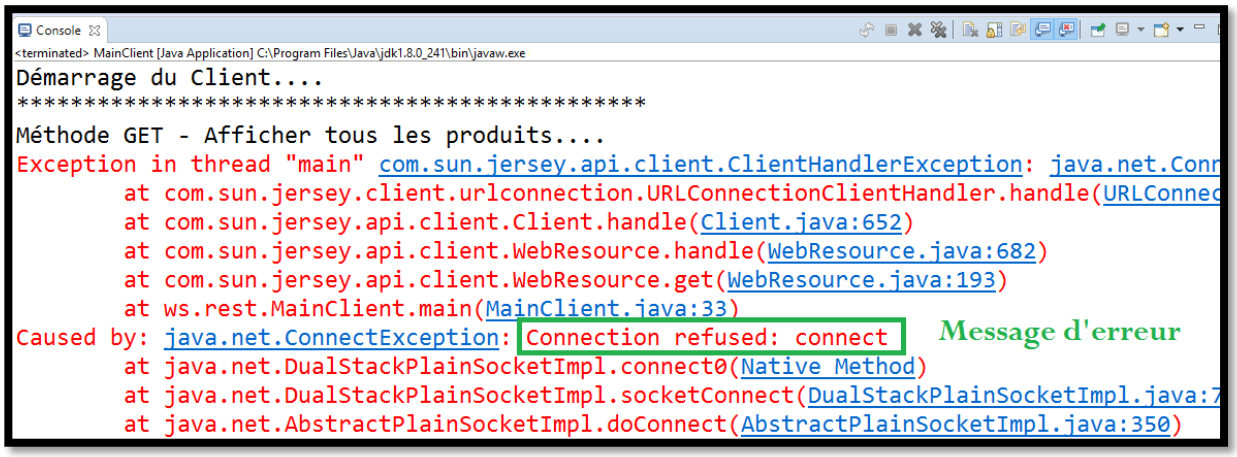
B. Invoquer les méthodes d'un service web REST avec la classe « WebResource »

5. Créer, sous le dossier « **main** », une classe «**MainClient**» ayant le code suivant :

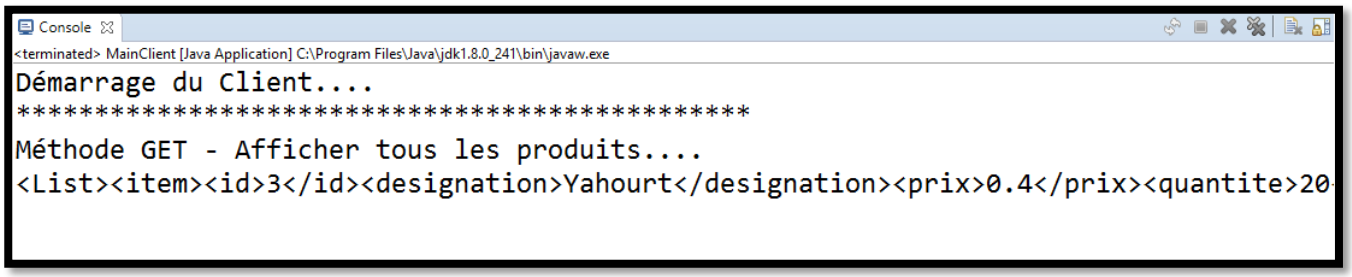
```java
package ws.rest;

import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.config.ClientConfig;
import com.sun.jersey.api.client.config.DefaultClientConfig;
import javax.ws.rs.core.UriBuilder;
import java.net.URI;

public class MainClient {

    public static void main( String[] args )
    {
        System.out.println("Démarrage du Client....");
        // Objet de configuration
        ClientConfig config = new DefaultClientConfig();
        //objet client
        Client client = Client.create(config);
        //créer l'uri
        URI uri =
UriBuilder.fromUri("http://localhost:8080/produits").build();
    //obtenir une resource correspondante à l'uri du service web
        WebResource service = client.resource(uri);
    //Requête GET
System.out.println( "*********************************************" );
System.out.println("Méthode GET - Afficher tous les produits....");
```

```
//référencer la méthode "getAllProduits"
WebResource resource= service.path("/");
//passer la méthode "get"
String reponseGetAllProduits= resource.get(String.class);
//Afficher la réponse textuelle
System.out.println(reponseGetAllProduits);
    }
}
```

6. Lancer l'exécution de la classe «**MainClient**» et remarquer le déclenchement de l'erreur suivante :



7. Avant de lancer l'exécution du projet client, il est nécessaire de publier le service web REST «**produits**» du projet «**jpa-spring-boot-ServiceWeb-REST**» avec le serveur «**Tomcat**» sur le port «**8080**». Puis lancer l'exécution du projet client pour consommer le service web et afficher la liste des produits comme suit :



8. Pour ajouter un nouveau produit, utiliser la méthode « **post** » de la classe « **WebResource** ». Ajouter le code suivant à la fin du corps de la méthode « main »:

```
//Requête POST
System.out.println( "***********************************************" );
System.out.println( "Méthode POST - Ajouter un nouveau produit...." );
//référencer la méthode "saveProduit"
WebResource resourceSave= service.path("/") ;
Produit nouveauProduit =new  Produit("Biscuit", 0.800, 15);
//passer la méthode « post »
Produit reponseSaveProduit=
          resourceSave
          .type(MediaType.APPLICATION_JSON)
          .post(Produit.class, nouveauProduit) ;
// Afficher la réponse textuelle de l'opération d'ajout
System.out.println(reponseSaveProduit) ;
```

- Remarquer que la classe «**Produit**» n'est pas reconnue.
- Ceci nécessite la déclaration d'une dépendance du projet «**jpa-spring-boot-ServiceWeb-REST**» dans le fichier « **pom.xml** » du projet client «**client-rest-java**» (prendre les mêmes paramètres indiqués dans le fichier pom.xml du projet «**jpa-spring-boot-ServiceWeb-REST**») :

```
<dependency>
 <groupId>spring.jpa</groupId>
 <artifactId>jpa-spring-boot-Thymeleaf</artifactId>
 <version>0.0.1-SNAPSHOT</version>
</dependency>
```

- Ainsi la classe « **Produit** » est reconnue (ajouter l'instruction «**import**» correspondante). Passer maintenant à l'exécution de la classe « **MainClient** ». Remarquer la génération de l'erreur suivante :

```
A message body writer for Java type, class spring.jpa.model.Produit, and MIME media type, application/json, was not found
```

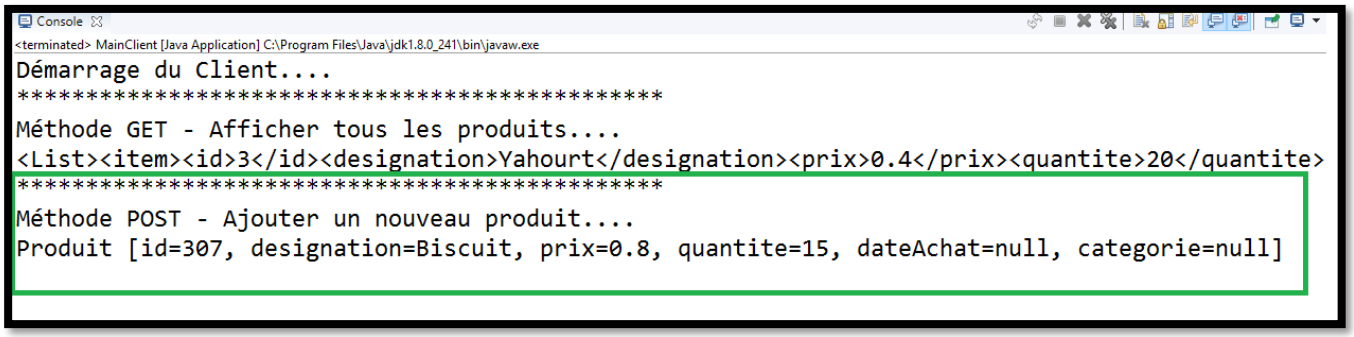
- Ceci est dû au fait le produit envoyé pour l'insertion est représenté au format JSON alors qu'au niveau service web, la méthode concernée (**saveProduit**) n'est pas configurée pour recevoir des données JSON. Modifier, donc, la méthode «**saveProduit**» au niveau du service web comme suit :

```
//  ajouter un produit avec la méthode "POST"
//  http://localhost:8080/produits/   (POST)
@PostMapping(
   // spécifier le path de la méthode
   value = "/"  ,
  //spécifier le format de retour
  produces = { MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE  }
  ,consumes= { MediaType.APPLICATION_JSON_VALUE }

  )
public Produit saveProduit(@RequestBody Produit p) {
return produitRepos.save(p);
}
```

- Ajouter la dépendance «jackson-jaxrs-json-provider» au niveau du projet client pour convertir le produit à insérer au format JSON avant de l'envoyer au serveur :

```
<dependency>
  <groupId>com.fasterxml.jackson.jaxrs</groupId>
  <artifactId>jackson-jaxrs-json-provider</artifactId>
  <version>2.3.0</version>
</dependency>
```

- Lancer l'exécution pour avoir le résultat suivant :



- Utiliser la méthode « **accept** » de l'objet « **WebResource** » pour recevoir les données en mode JSON.

C. Utiliser l'API « Gson » de Google pour convertir un objet Java dans sa représentation JSON et vice versa

9. Ajouter la dépendance suivante :

```
<!-- Google JSON API-->
    <dependency>
      <groupId>com.google.code.gson</groupId>
      <artifactId>gson</artifactId>
      <version>2.2.4</version>
    </dependency>
```

10. Ajouter le code suivant à la fin de la méthode « **main** » de la classe « **MainClient** » pour récupérer des objets « **Produit** » en utilisant « **Gson** » (ajouter les les instructions « **import** » correspondantes à « **Gson** » :

```
// Récupérer des objets "Produit" en utilisant l'API gson de Google
        Gson gson = new GsonBuilder().create();
        JsonArray jo = new
JsonParser().parse(reponseGetAllProduits).getAsJsonArray();
       //  jsonArray = jo.getAsJsonArray("produit");
        Produit[] listeP = gson.fromJson(jo,Produit[].class);
        System.out.println(
"*********************************************" );
        System.out.println( "Liste des produits (API gson)....");
        for (Produit p: listeP)
        {
            System.out.println(p);
        }
```

11. Lancer maintenant l'exécution et remarquer la génération de l'erreur suivante :

```
Caused by: java.text.ParseException: Unparseable date:
```

12. Modifier la manière de création d'un objet « **Gson** » en utilisant l'instruction suivante (spécifier le pattern pour les dates) et relancer l'exécution:

```
Gson gson = new GsonBuilder().setDateFormat("yyyy-MM-dd").create();
```