TP Création des web services

Outils utilisés

- JavaSE : http://www.oracle.com
- NetBeans, en version EE : https://netbeans.org/downloads/
- Glassfish, serveur d'applications et conteneur JAVA EE, fourni avec NetBeans en version Java EE

Optionnellement, les outils suivants peuvent également intervenir

- SQLite Browser, pour manipuler les bases de données SQLite : http://sqlitebrowser.sourceforge.net/
- Un environnement Apache/PHP, pour éventuellement tester les Web Services en PHP plutôt qu'en Java (WAMP)

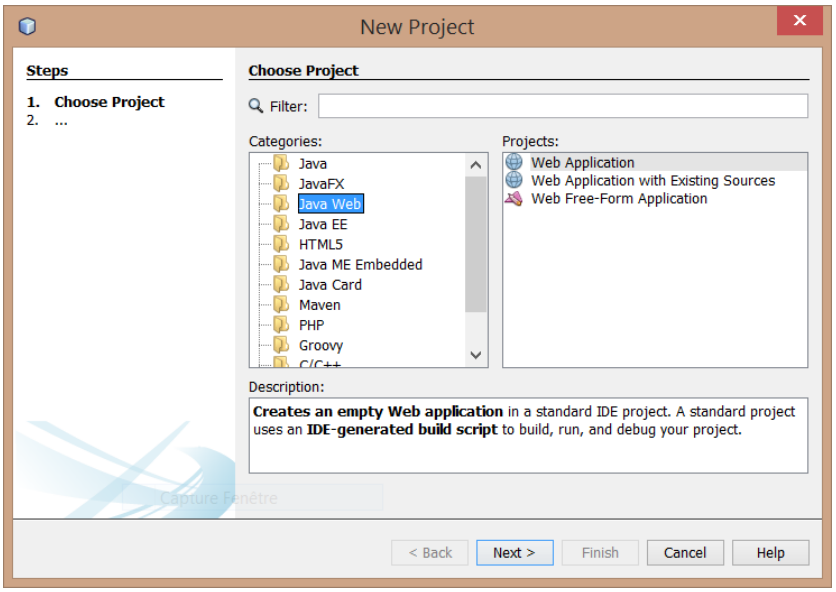
Projet Simple

Dans cette section, nous allons utiliser l'approche **Bottom-Up** pour créer un web service à partir d'une classe Java proposant quelques simples opérations mathématiques :

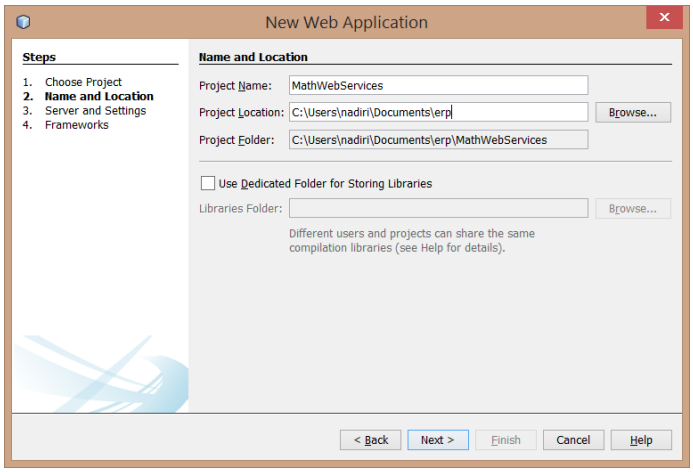
- Additionner deux nombres
- tester si un nombre est premier
- décomposer un nombre en facteurs premiers

Pour cela, créer sous NetBeans un nouveau projet de type **Web Application** :

Nouveau projet Web

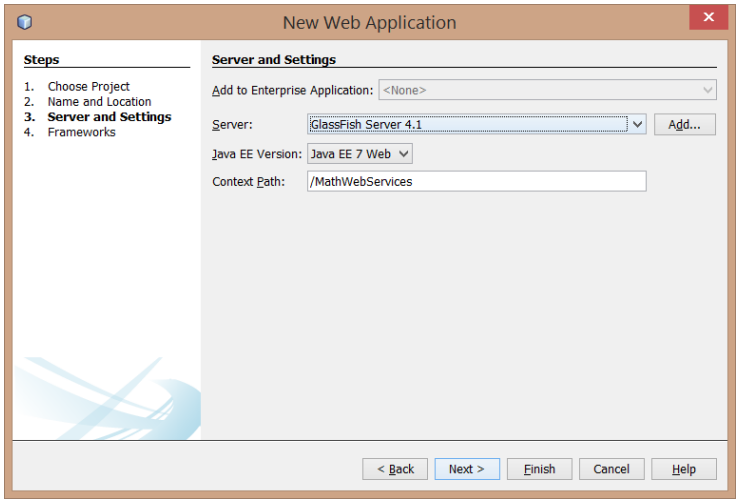


Nommage et emplacement



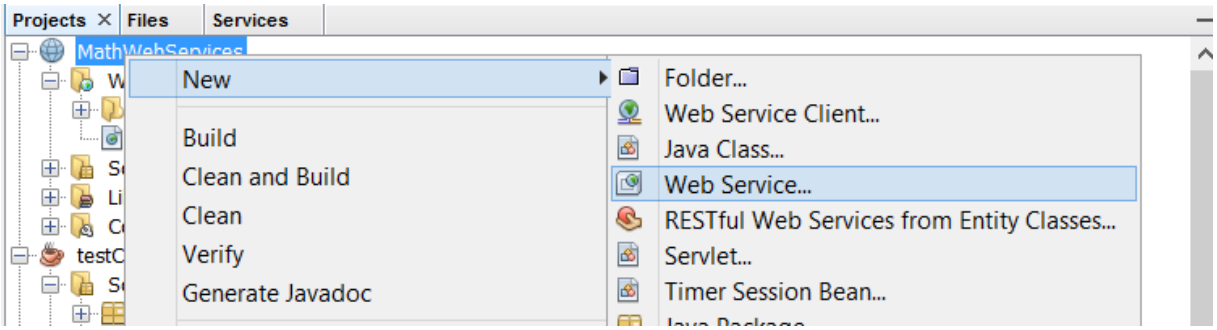
Nous utilisons ici Glassfish pour héberger notre code métier et exposer le service. D'autres conteneurs sont disponibles, tels que Tomcat, Geronimo, Websphere, Oracle Application Server, Wildfly. Chaque serveur peut utiliser des frameworks différents pour prendre en charge la génération du code, la sérialisation/désérialisation des messages, etc. Glassfish repose sur L'implémentation Metro qui prend en charge JAX-WS (Java API for XML Web Services) pour prendre en charge la traduction WSDL/Java et SOAP/Java.

Environnement d'exécution

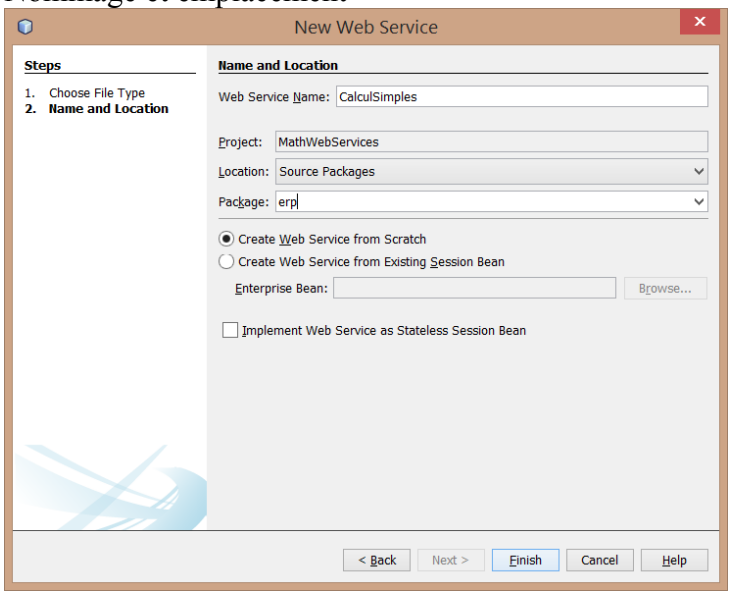


Nous allons ensuite créer un nouveau web service dans le projet :

Création d'un nouveau service



Nommage et emplacement



Une fois les paramètres saisis, Netbeans présente le code métier généré, avec une opération par défaut : `hello`. L'utilisation de l'API JAX-WS, intégrée à la plateforme depuis la version JAVA EE 5, permet d'utiliser des métadonnées directement dans le code source pour transformer une classe et certaines de ses méthodes en web service. Ces métadonnées sont appelés `Annotations`, et sont exploitées par l'environnement pour extraire des informations qu'il fallait auparavant définir dans des descripteurs XML annexes. Le développement d'un web service est ainsi grandement simplifié. Trois annotations sont utilisées ici

@WebService
: Définit la classe comme étant un web service, identifié par un nom

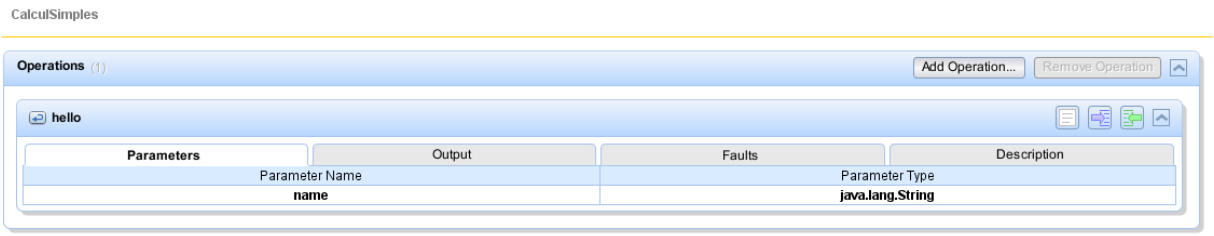
@WebMethod
: Expose une méthode de la classe en tant qu'opération proposée par le web service, sous un nom donné. Par défaut, toutes les méthodes publiques d'une classe sont exposées, le paramètre `(exclude=true)` permet d'annuler ce comportement

@WebParam
: Définit un paramètre de l'opération

```java
package erp;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

/**
 *
 * @author nadiri
 */
@WebService(serviceName = "CalculSimples")
public class CalculSimples {

    /**
     * This is a sample web service operation
     */
    @WebMethod(operationName = "hello")
    public String hello(@WebParam(name = "name") String txt) {
        return "Hello " + txt + " !";
    }
}
```

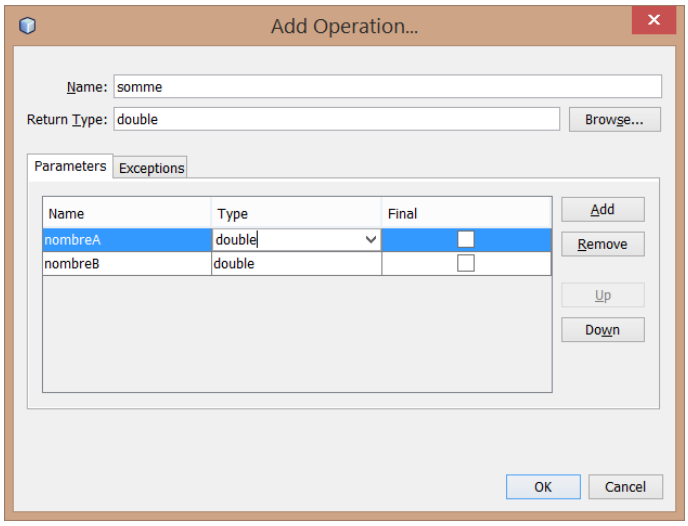
Code généré et annotations

Netbeans propose un assistant pour créer des opérations. Pour cela, passer de la vue *Source* à la vue *Design*. :



Vue graphique du service

Créer ensuite une nouvelle opération additionner avec les paramètres suivants :



Création d'une opération

En basculant à nouveau vers la vue *Source*, nous pouvons voir le code généré avec les informations :

```
/**
 * Web service operation
 */
@WebMethod(operationName = "somme")
public double somme(@WebParam(name = "nombreA") double nombreA, @WebParam(name = "nombreB") double nombreB) {
    //TODO write your implementation code here:
    return 0.0;
}
```

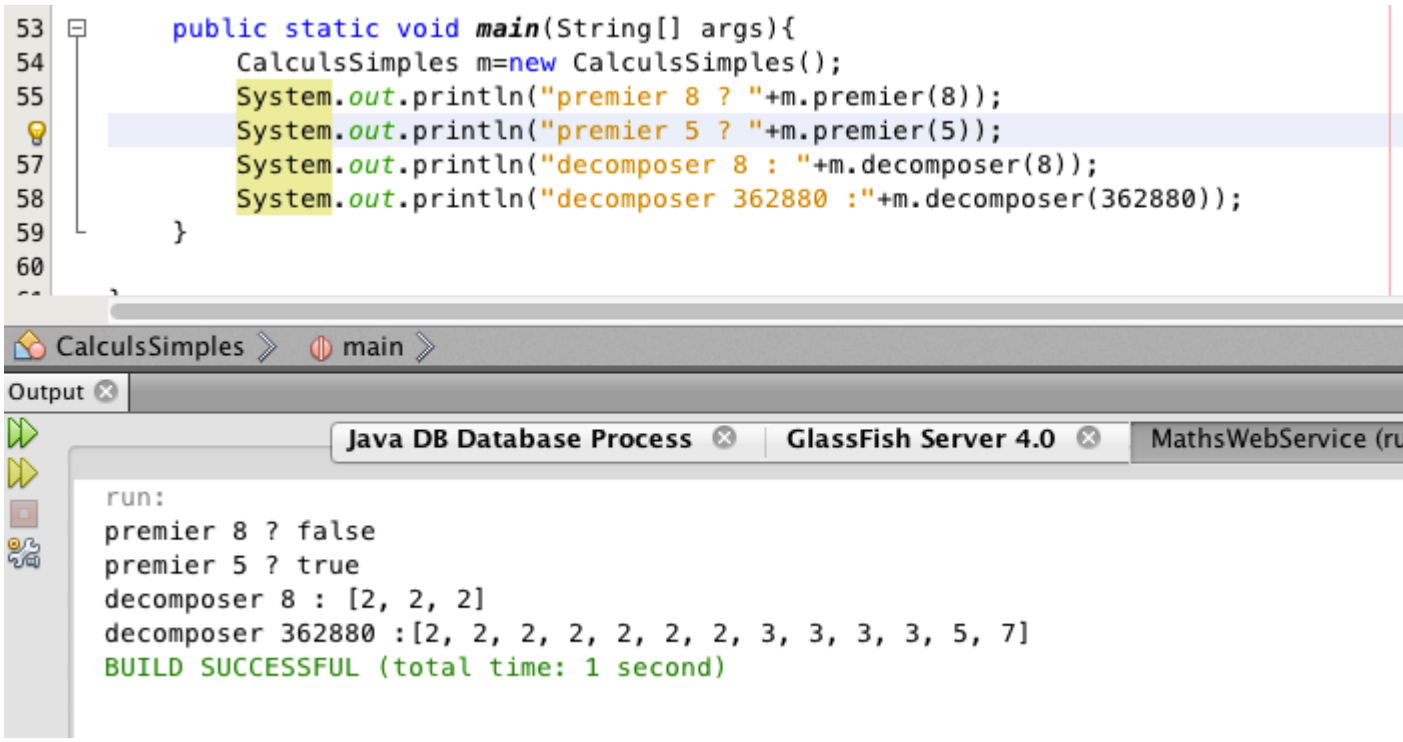
Code généré

Créer de la même manière deux autres opérations `decomposer` et `premier`. Vous pouvez également éditer le code sans passer par l'assistant.

Autres opérations

```
31
32      @WebMethod(operationName = "decomposer")
33      public List decomposer(@WebParam(name = "entier") int entier) {
34          List l = new ArrayList();
35          int i=2;
36          while(i<=entier){
37              if(entier%i==0){
38                  l.add(i);
39                  entier=entier/i;
40              }else
41                  i++;
42          }
43          return l;
44      }
45
46      @WebMethod(operationName = "premier")
47      public boolean premier(@WebParam(name="entier") int entier){
48          return decomposer(entier).size()==1;
49      }
50
51
52  }
```

La dernière étape consiste à tester si le code fonctionne correctement avant de passer à la phase suivante. Pour cela nous créons une méthode `main` pour exécuter chacune des fonctions et vérifier leur comportement. Pour exécuter la classe en tant que simple application, nous utilisons le menu `Run » Run File`



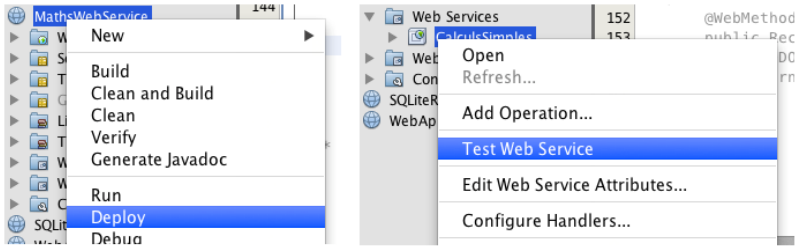
Test du code métier

Tester Invocation du web service par le navigateur

Le web service est maintenant prêt à être déployé sur un serveur. Rappelons que les web services ont vocation à être exécutés dans un environnement qui prend en charge les requêtes provenant des clients, la traduction des messages SOAP, l'appel du code métier et la transmission du résultat après retraduction en SOAP.

Dans Netbeans, sélectionner le projet, et dans le menu du bouton droit choisir l'action `Deploy`. Si le serveur Glassfish n'est pas lancé, il faudra patienter quelques secondes...

Une fois le serveur lancé et le projet déployé, nous pouvons tester le web service. Sélectionner le web service dans le projet, puis choisir l'action `Test Web Service` dans le menu du bouton droit.

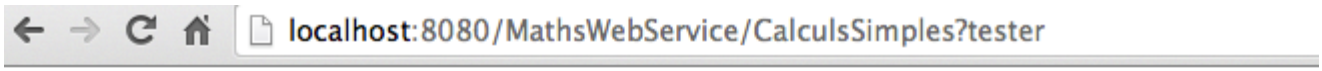


Déploiement et test

Glassfish/JAX-WS propose une page résumant les opérations exposées par le web service, et la possibilité de les tester en passant les paramètres éventuels.

Aperçu du web service

Nous pouvons noter que le testeur propose une vue des messages SOAP échangés durant les requêtes :



Method parameter(s)

Type	Value
double	123
double	456

Méthode renvoyée

double : "**579.0**"

Demande SOAP

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xml
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:additionner xmlns:ns2="http://emiage.org/">
            <nombreA>123.0</nombreA>
            <nombreB>456.0</nombreB>
        </ns2:additionner>
    </S:Body>
</S:Envelope>
```

Réponse SOAP

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xml
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:additionnerResponse xmlns:ns2="http://emiage.org/">
            <return>579.0</return>
        </ns2:additionnerResponse>
    </S:Body>
</S:Envelope>
```

Test de l'opération *additionner*

decomposer Appel de méthode

Method parameter(s)

Type	Value
int	362880

Méthode renvoyée

java.util.List : "**[2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 5, 7]**"

Demande SOAP

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:decomposer xmlns:ns2="http://emiage.org/">
            <entier>362880</entier>
        </ns2:decomposer>
    </S:Body>
</S:Envelope>
```

Test de l'opération *decomposer*

Notre web service est à présent fonctionnel, nous pouvons passer à la phase suivante : consommer le web service à partir d'une application.

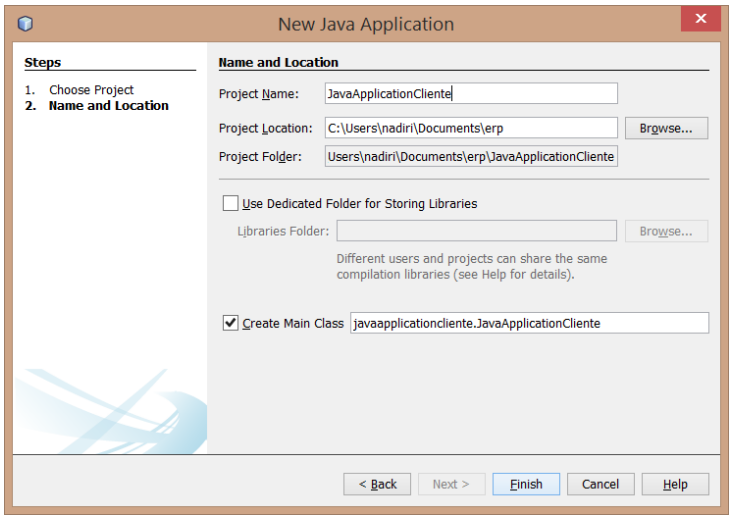
Consommer un web service

Dans cette section, nous voyons comment consommer un web service à partir d'une application java puis à partir d'un script PHP. On peut considérer un consommateur de web service comme une application cliente (au sens de l'architecture Client-Serveur), ou bien comme un *Mashup*, c'est-à-dire une application utilisant des services provenant d'une ou plusieurs sources pour proposer un contenu (ou un service).

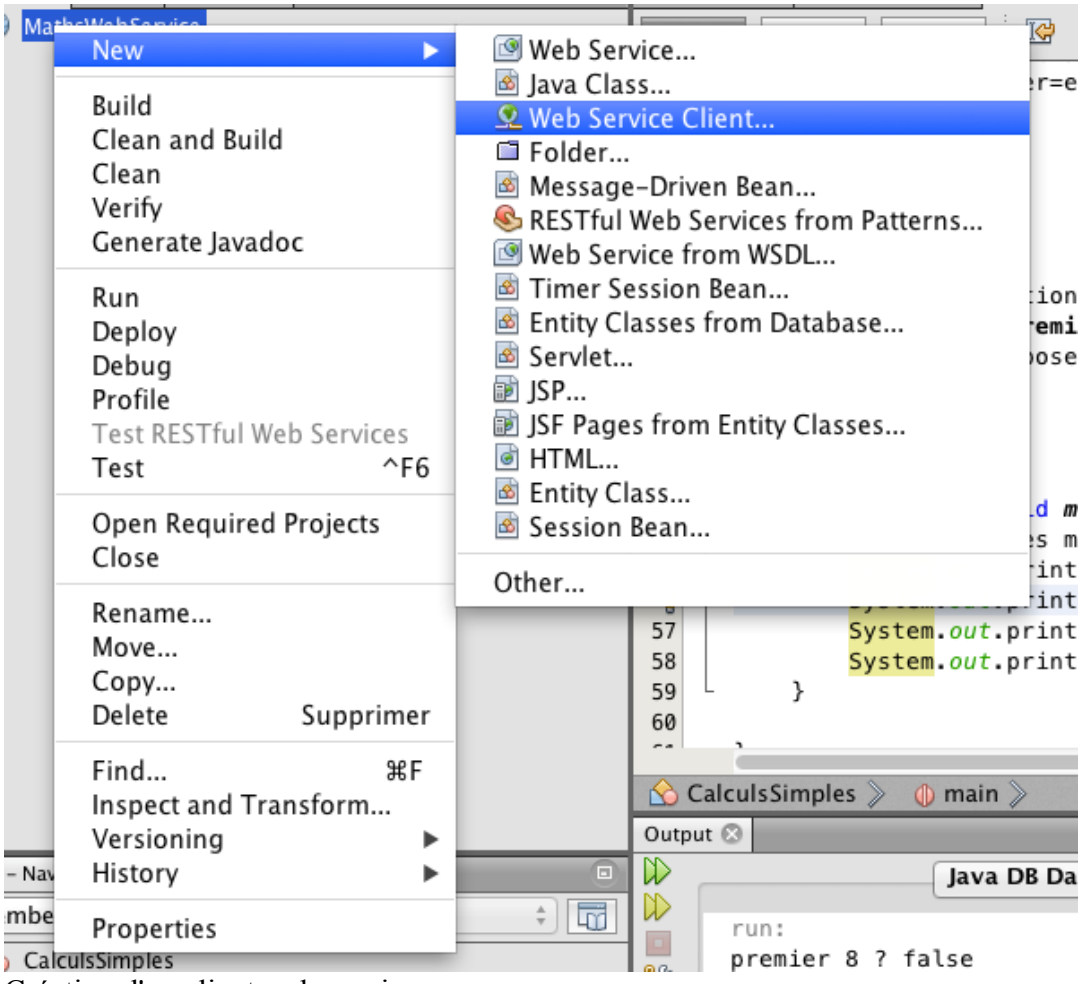
Projet

Commençons par créer un nouveau projet Java standard

Netbeans : création d'un projet Java

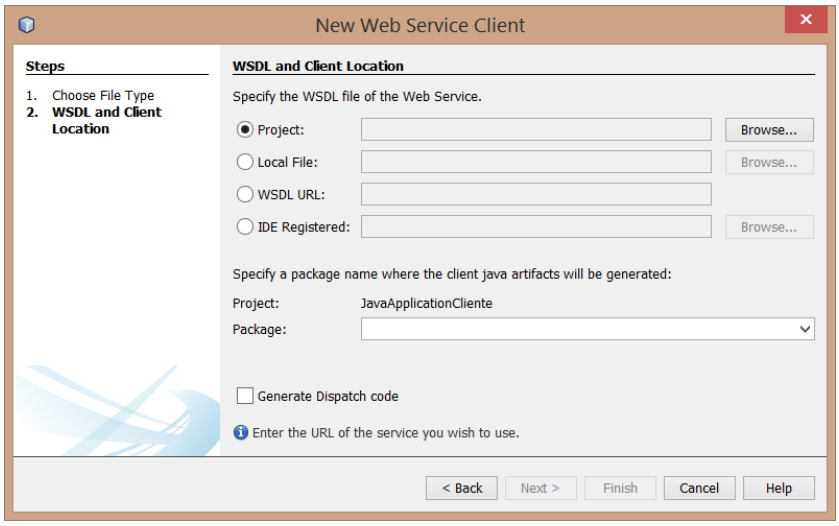


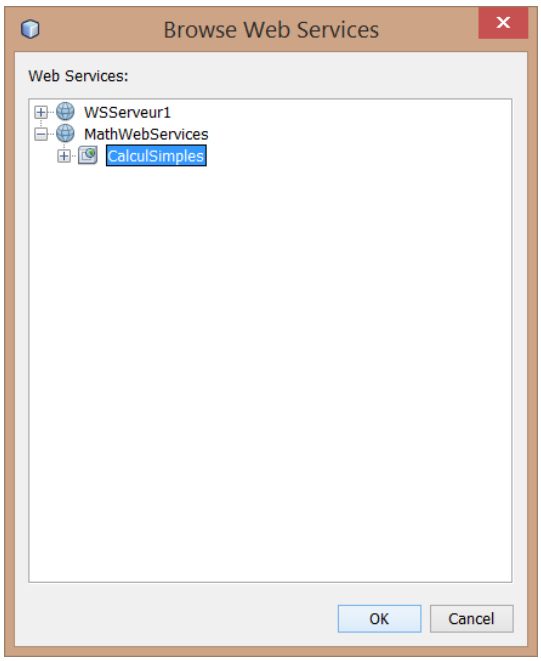
A l'intérieur du projet, nous créons un nouveau "web service client"



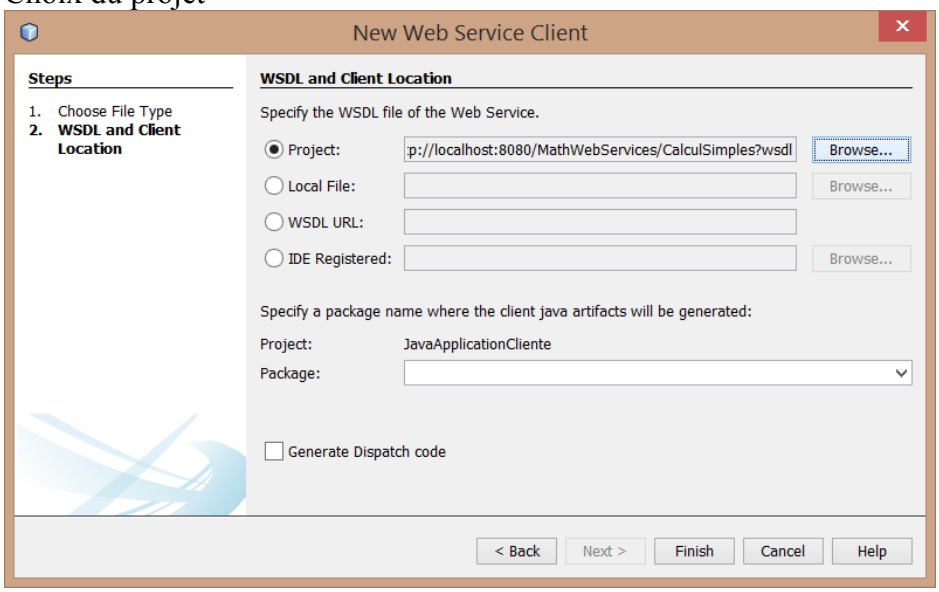
Création d'un client web-service

Nous sélectionnons ensuite un projet de type web-service afin d'importer sa définition (son WSDL). Il serait également possible de créer une application cliente à partir d'un WSDL accessible en ligne.





Choix du projet



Choix du WSDL

Ensuite, nous pouvons choisir quelles sont les opérations que nous souhaitons invoquer dans notre application. Pour cela, nous déplions le dossier `Web Service References` dans le projet, puis à l'aide d'un glisser/déposer, nous insérons une des opérations directement dans le code.

Importation d'une opération

Une fois la méthode (`static`) importée, il ne reste qu'à l'invoquer dans le code de la méthode `main`, le point d'entrée de notre application.

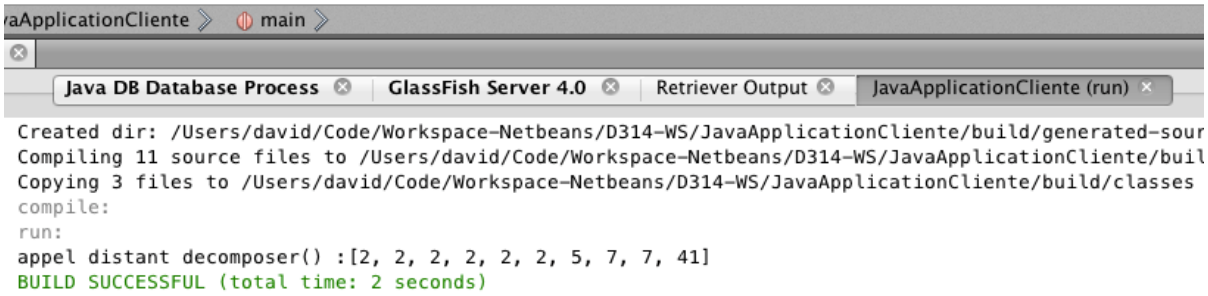
```
public static void main(String[] args) {
    System.out.println(premier (501));
  }
```

Invocation de l'opération dans la fonction main

Test du client

Après s'être assuré que le web service est toujours lancé, nous pouvons exécuter l'application cliente (Menu `Run`)

```java
public class JavaApplicationCliente {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        System.out.println("appel distant decomposer() :"+decomposer(642880));

    }

    private static java.util.List<java.lang.Object> decomposer(int entier) {
        org.emiage.CalculsSimples_Service service = new org.emiage.CalculsSimples_Service();
        org.emiage.CalculsSimples port = service.getCalculsSimplesPort();
        return port.decomposer(entier);
    }

}
```



Lancement du client et invocation du web service

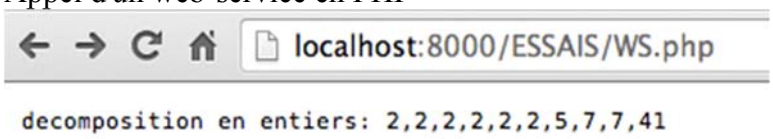
Consommer un web-service via PHP

PHP propose maintenant en standard les librairies permettant de consommer des web services à l'aide de la classe `SoapClient`. Une fois instanciée en indiquant l'URL du WSDL, la classe retournée représente l'objet distant qui propose le web service. Étant donnée la dynamicité du langage PHP, les opérations proposées sont directement mappées au sein de cette classe. À noter, les éventuels arguments sont passés à travers un tableau associatifs, dont les clés doivent correspondre exactement aux noms des paramètres spécifiés par le web service

En partant du principe d'un serveur Apache/PHP est installé sur la machine (http://easyphp.org, http://wampserver.com, http://www.mamp.info), nous pouvons tester le code suivant.

```php
<pre>
<?php
$mathservice = new SoapClient("http://localhost:8080/MathsWebService/CalculsSimples?wsdl");
$result = $mathservice->decomposer(array("entier"=>642880));
echo "decomposition en entiers: ". implode(",",$result->return);

```

Appel d'un web-service en PHP




```
decomposition en entiers: 2,2,2,2,2,2,5,7,7,41
```

Aperçu dans le navigateur

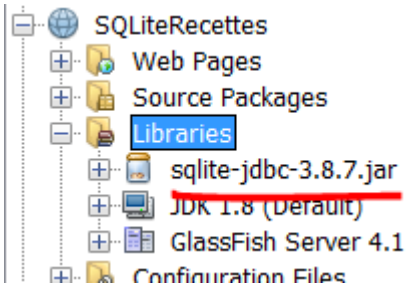
Exemple avec une BD SQLite

Embarquer une base de données SQLite

Bien souvent, un web-service met à disposition qu'il fournit des données qui lui sont propres à travers les opérations qu'il propose. Rappelons que l'environnement d'exécution est généralement un Container (cf précédemment : Tomcat, Geronimo, Websphere, Oracle Application Server, Glassfish). Dans cette partie, nous mettons en place une base de données embarquées (SQLite) et décrivons les étapes qui permettent de l'exploiter pendant la phase de développement puis de déploiement.

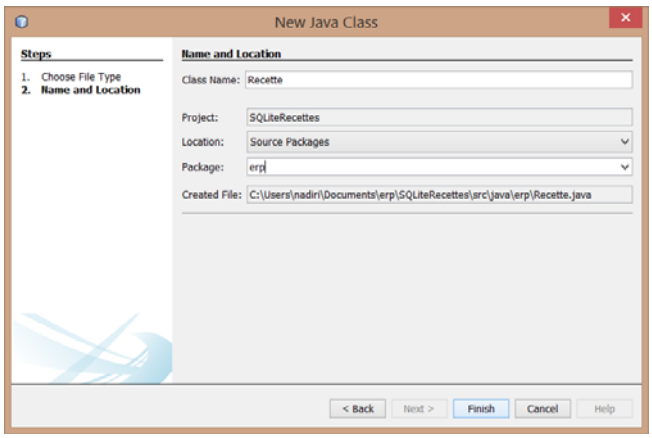
La première étape consiste à inclure la librairie SQLite (à télécharger : https://bitbucket.org/xerial/sqlite-jdbc).

Après avoir créé un projet Web-service nommé SQLiteRecettes, ajouter l'archive jar dans la section `librairies`.



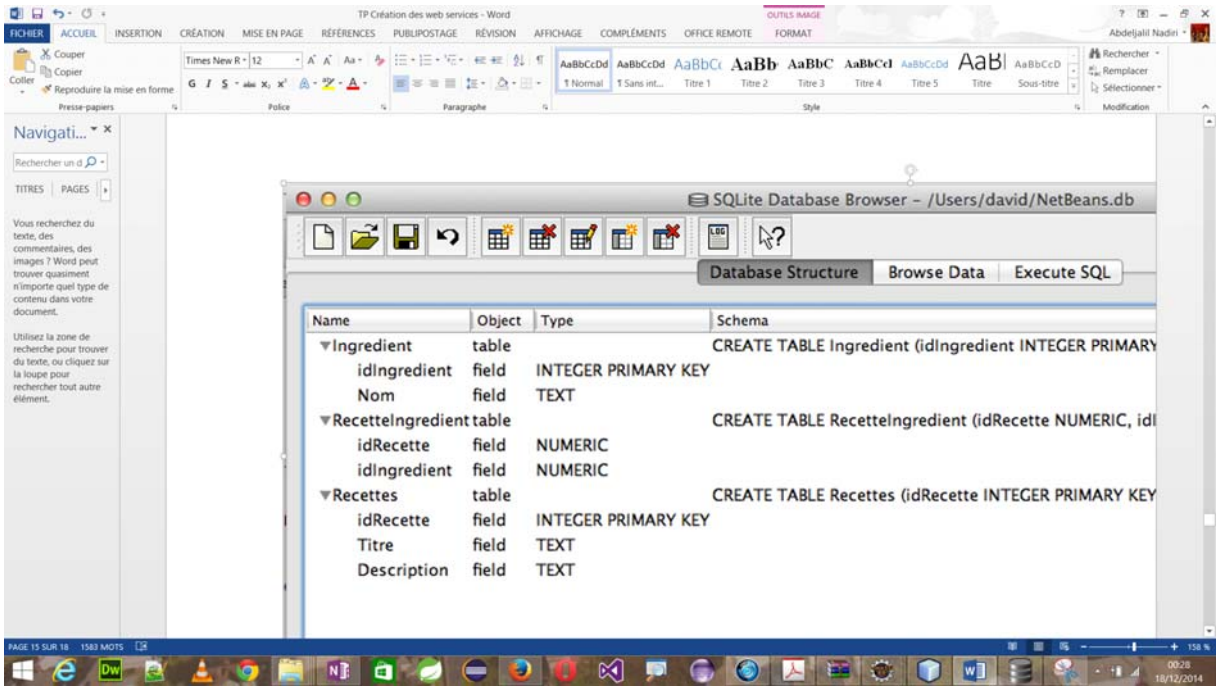
Librairie SQLite dans un projet

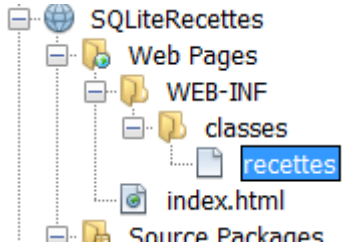
Nous créons ensuite une classe pour tester la librairie et utiliser une base de données test



Nouveau service de test

A l'aide d'un éditeur de base de données (ici : SQLite DataBase Browser, lien de téléchargement : http://sourceforge.net/projects/sqlitebrowser), nous créons 3 tables permettant de stocker des recettes de cuisine, puis enregistrons la base dans le dossier WEB-INF/classes de notre projet (le dossier classes doit être créé s'il n'existe pas, dans la catégorie other sélectionner folder)





Tables SQlite

Chargement de la base

Nous implémentons ensuite la classe du web-service. Pour que le driver soit chargé au démarrage de la classe, nous plaçons un bloc de code `static` de la manière suivante

```java
package erp;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

/**
 *
 * @author nadiri
 */
@WebService(serviceName = "TestSqlite")
public class TestSqlite {

   // Charger le pilote en même temps que la classe
static {
    try{
        Class.forName("org.sqlite.JDBC");
    }
    catch (ClassNotFoundException e)
    {

    }

        }
```

Instanciation du driver JDBC

Nous implémentons ensuite une classe qui va permettre de contenir les données d'une recette dans une structure propre. Il est bien-entendu possible d'utiliser des tableaux, listes et chaînes de caractères comme résultat d'un web service, même si l'on préférera des classes pour des types de données plus complexes.

Le mécanisme qui permet à Java de traduire des classes vers ou à partir des types XML est JAX-B. Lorsque les retours d'opérations utilisent explicitement un type de données complexes, JAX-B se charge automatiquement de faire la traduction.

```java
package erp;

import java.io.Serializable;

/**
 *
 * @author nadiri
 */
public class Recette implements Serializable{
   private String titre;
   private String description;

    public String getTitre() {
        return titre;
    }

    public void setTitre(String titre) {
        this.titre = titre;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }


}
```

Classe Recette : structure de données

Nous implémentons ensuite l'opération qui permet de chercher les recettes qui contiennent un ingrédient donné.

Note

Au moment du déploiement, le web service est embarqué dans un serveur d'applications, qui possède sa propre structure de fichiers. Il est donc fortement déconseillé d'utiliser des chemins absolus pour charger la base de données.

Nous utilisons le `ClassLoader` pour aller charger une ressource, ici notre fichier de base de données. Lors du déploiement, les fichiers, dossiers et jars situés dans le répertoire WEB-INF/classes du projet seront copiés tels quels dans le dossier de l'application sur le serveur.

L'opération suivante renvoie une `List` contenant des objets de types recettes.

```
     */
    @WebMethod(operationName = "rechercheParIngredient")
    public List<Recette> rechercheParIngredient(@WebParam(name = "ingrdient") String ingredient) {
        URL ch = getClass().getResource("/recette");
        String chemin = ch.getPath();
        try {

            Connection cnx = DriverManager.getConnection("jdbc:sqlite:" + chemin);
            Statement st = cnx.createStatement();

            ResultSet rs = st.executeQuery("Select titre, description from recettes"
                    + "inner join RecetteIngredient on RecetteIngredient.idRecette=Recettes.idRecette "
                    + "inner join Ingredient on ingredient.idIngredient=recetteIngredient.idIngrdient "
                    + "where nom like '%" + ingredient + "'"
            );

            List<Recette> resultat = new ArrayList<>();
            while (rs.next()) {
                String titre = rs.getString("Titre");
                String description = rs.getString("Description");
                Recette r = new Recette();
                r.setDescription(description);
                r.setTitre(titre);
                resultat.add(r);
            }
            return resultat;
        } catch (SQLException e) {
            return null;
        }

    }
}
```

Service de recherche de recettes par ingrédient

JAX-B Recette<->XML

Un simple client (ici en PHP) nous renvoie deux résultats, sous la forme d'un classe possédant les deux attributs déclarés dans notre classe recettes.

Test SQLITE

```
array(2) {
  [0]=>
  object(stdClass)#3 (2) {
    ["description"]=>
    string(77) "Poulet Indien cuit au four avec sauce crémeuse et servie avec du riz basmati"
    ["titre"]=>
    string(15) "Poulet au Curry"
  }
  [1]=>
  object(stdClass)#4 (2) {
    ["description"]=>
    string(52) "Poulet cuit avec poivrons/tomates, servi avec du riz"
    ["titre"]=>
    string(16) "Poulet Basquaise"
  }
}
```

Consommation du service en PHP