Тема 4.
Web-сервіси

- Основні поняття технології Web-сервісів
- Простий протокол доступу до об’єктів (SOAP)
- Web Services Description Language (WSDL)
- Створення і розгортання простого Веб-сервісу та клієнта в Eclipse
- Створення клієнта на основі WSDL
- Приклад створення Веб-сервісу з використанням Axis2 і зверненням до бази даних
- Ознайомлення з REST-сервісами
- Приклад створення Веб-сервісу із використанням архітектури REST
- SOAP чи REST сервіси?

Основні поняття технології Web-сервісів

Існує декілька означень веб-сервісу. Веб-сервісом може бути будь-який додаток, який має доступ до Web, наприклад, веб-сторінка з динамічним вмістом. В більш вузькому розумінні, веб-сервіс – це додаток, що є відкритим інтерфейсом, і може бути використаний іншими додатками в Web. Концепція веб-сервісів реалізується за допомогою ряду технологій, які стандартизовані World Wide Web Consortium (W3C).

Взаємозв'язок цих технологій можна умовно представити наступним чином.

Технології	Роль технологій
Common Internet Protocols	Середовище обміну повідомленнями
Extensible Markup Language (XML)	Представлення даних
Simple Object Access Protocol (SOAP)	Обмін повідомленнями
Web Services Description Language (WSDL)	Опис можливих сервісів
Universal Description, Discovery, and Integration (UDDI)	Публікація сервісів, пошук сервісів

XML є фундаментом для створення більшості технологій, пов'язаних з веб-сервісами.

Для віддаленої взаємодії з веб-сервісами використовується *Simple Object Access Protocol (**SOAP**)*. SOAP забезпечує взаємодію розподілених систем, незалежно від об'єктної моделі, операційної системи або мови програмування. Дані передаються у вигляді особливих XML документів особливого формату.

Згідно з визначенням **W3C**, веб-сервіси - це додатки, які доступні за протоколами, які є стандартними для Інтернет. Немає вимоги, щоб веб-сервіси

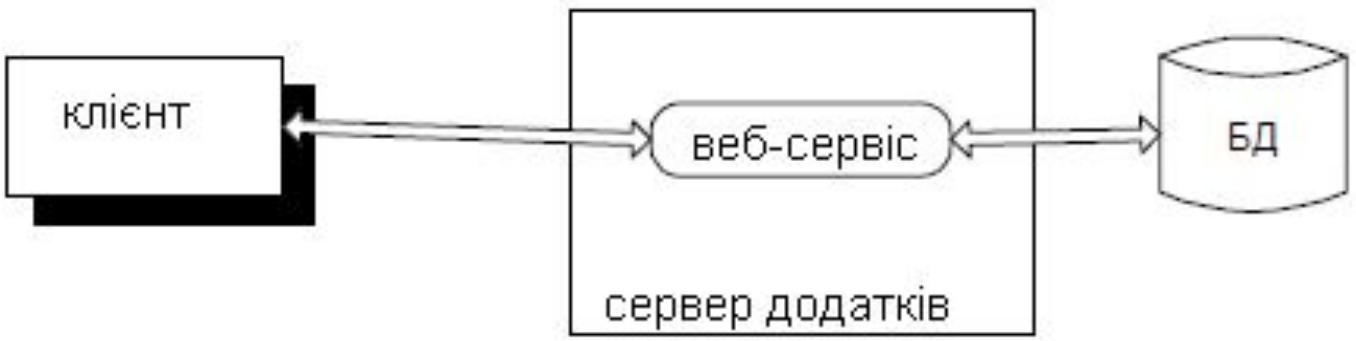
використовували якийсь певний транспортний протокол. Специфікація SOAP визначає, яким чином зв'язуються повідомлення SOAP і транспортний протокол. Найбільш часто реалізується передача SOAP повідомлень по протоколу HTTP. Також широко поширене використання в якості транспортного протоколу SMTP, FTP, TCP.

Згідно з визначенням W3C, «***WSDL*** - формат XML для опису мережевих сервісів як набору підсумкових операцій, які працюють за допомогою повідомлень, що містять документно-орієнтовану або процедурно-орієнтовану інформацію».

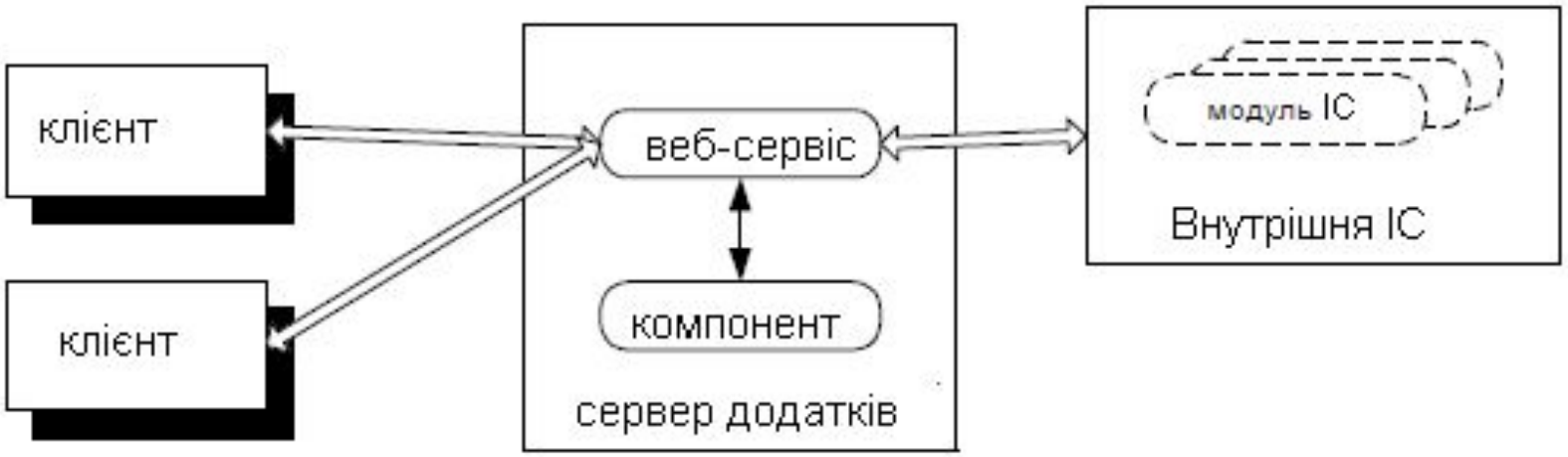
Технологія ***Universal Description, Discovery and Integration (UDDI)*** передбачає ведення реєстру веб-сервісів. Підключившись до цього реєстру, споживач зможе знайти веб-сервіси, які найкращим чином задовольняють його потреби. Технологія UDDI дає можливість пошуку та публікації потрібного сервісу як людиною, так і програмою-клієнтом. Пошук і публікація в реєстрі надається програмі-клієнтові як набір веб-сервісів реєстру UDDI.

Розробники концепції веб-сервісів пропонують наступні сценарії застосування веб-сервісів.

Веб-сервіси як реалізація логіки додатка (бізнес-логіки). Тобто створення нового додатка, бізнес-логіка якого реалізується у веб-сервісі.

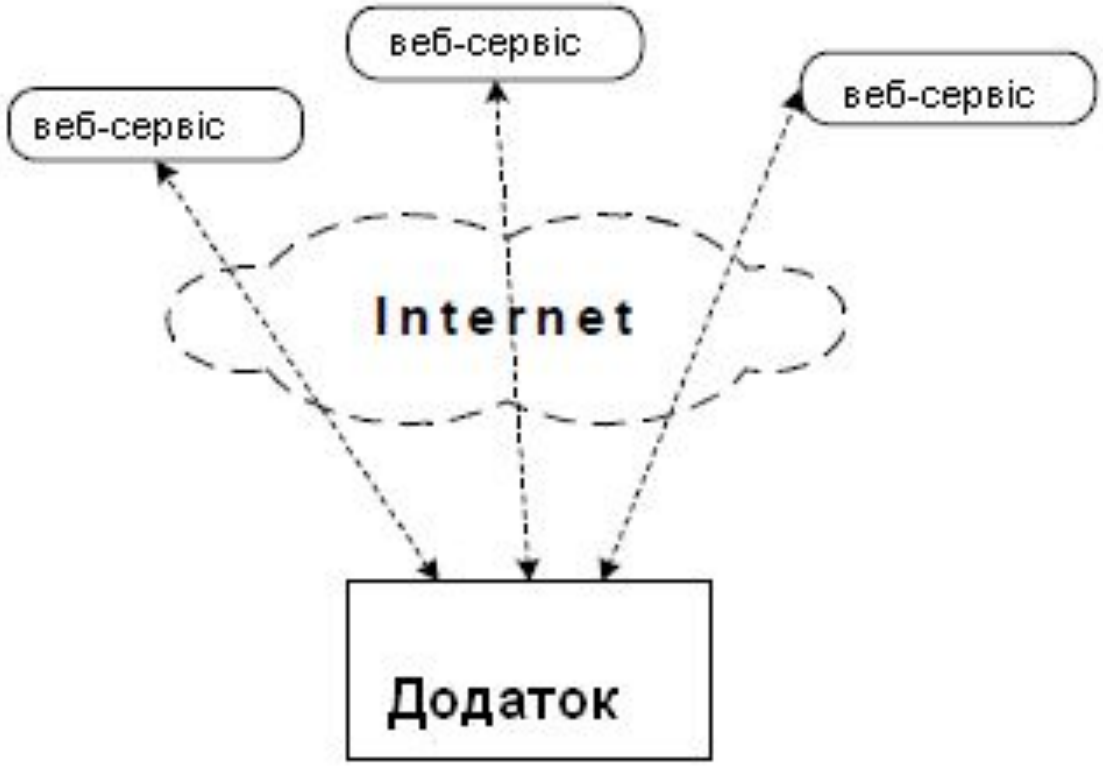


Веб-сервіси як засіб інтеграції. Тобто використання веб-сервісу як засобу доступу віддалених клієнтів до внутрішньої ІС компанії, або для організації взаємодії компонента (наприклад, EJB, COM-компонента) з різними віддаленими клієнтами.



Як було сказано вище, концепція веб-сервісів включає в себе можливість ведення реєстру веб-сервісів. Передбачається, що в реєстрі зберігається загальний опис веб-сервісу, інтерфейс сервісу і адреса, за якою він доступний.

Використання веб-сервісу як будівельного блоку при створенні додатка. Програма може використовувати веб-сервіси як віддалені компоненти, які надають певну функціональність. Існують різні сервіси, які надають якісне рішення таких завдань як аутентифікація, ведення календаря, відправлення повідомлень, пошук, переклад і т. п.



Простий протокол доступу до об'єктів (SOAP)

SOAP – це протокол, що базується на HTTP-XML. Він дозволяє додаткам взаємодіяти між собою через Інтернет, використовуючи для цього XML-документи, які називаються *повідомленнями SOAP.*

Повідомлення SOAP містить ***конверт***, який описує вмістиме, отримувача повідомлення і вимоги до обробки повідомлення. Необов' язковий елемент ***header*** *(заголовок)* повідомлення SOAP містить інструкції по обробці для додатків, які приймають повідомлення. Заголовок також може містити інформацію про маршрутизацію. З допомогою заголовка *header* поверх SOAP можуть надбудовуватись більш складні протоколи. Записи в заголовку можуть модульно розширювати повідомлення для таких задач, як аутентифікація, управління транзакціями і здійснення платежів. **Тіло** SOAP-повідомлення містить специфічні для додатку дані, призначені для отримувача повідомлення.

SOAP можна використовувати для здійснення віддаленого виклику процедур ***Remote Procedure Call (RPC)***, який представляє собою запит, що надсилається іншому комп'ютеру для виконання певної задачі. RPC використовує словник XML для задання методу, що викликається, параметрів, які йому передаються і універсального ідентифікатора ресурса (**URI**) цільового об'єкту.

Web Services Description Language (WSDL)

Для забезпечення незалежності від мови програмування кожен Веб-сервіс забезпечується описом на мові ***WSDL***, що є діалектом XML. Мова WSDL дозволяє описувати наступні об'єкти:

- сервіс (**service**) містить один або декілька портів (**port**), кожен з яких задається прив'язкою порта (**binding**);
- кожна прив'язка порта визначається типом порта (**port type**) з деяким набором параметрів, що використовуються при передачі даних;
- кожен тип порта містить опис одної або декількох операцій (**operation**);
- кожна операція полягає в передачі одного або двох повідомлень (**message**), причому, якщо повідомлень два, то одне повинно бути вхідним, а інше вихідним. Також в описі операцій можуть бути перераховані повідомлення про помилки;
- кожне повідомлення може складатися з одної або декількох частин (**part**);
- кожна частина характеризується деяким типом даних.

Наведемо приклад найпростішого Веб-сервісу:

```
// на сервері
import javax.jws.*;
@WebService(targetNamespace= "http://samples/hello")
public class Hello {
  public String getHello(String name) {
    return String.format("Hello, %s!", name);
                                          }
                                         }
```

Якщо деякий метод визначає операцію, то типи даних його параметрів, а також тип даних значення, що повертається, може бути:

- примітивним (int, double і т.д.);
- обгорткою (Integer, Double і т.д.);
- рядком (String);
- датою (Date) або календарем (Calendar);
- будь-яким класом, анотованим XmlRootElement;
- масивом, колекцією, списком або множиною елементів перерахованих вище типів (масив байтів обробляється особливим чином, колекції, списки і множини не можуть бути вкладеними, масиви можуть бути вкладеними, однак робити так не рекомендується).

Для Веб-сервісу автоматично створюється опис на мові WSDL, доступний за адресою http://localhost/hello?wsdl, він включає в даному випадку єдиний сервіс HelloService, який містить єдиний порт HelloPort, що в свою чергу містить єдину операцію getHello

Надати клієнтам доступ до Веб-сервісу можна наступним чином:

// на сервері

```
Endpoint.publish("http://localhost/hello",new Hello());
```

Щоб клієнт міг звернутися до Веб-сервісу, спочатку необхідно визначити інтерфейс, що представляє операції Веб-сервісу:

// на клієнті

```
@WebService(targetNamespace= "http://samples/hello")
public interface Hello {
   String getHello(String name);
}
```

Маючи інтерфейс, звернутися до Веб-сервіса можна так:

```
// на клієнті
Service helloService = Service.create(
  new URL("http://localhost/hello?wsdl"),
    new QName("http://samples/hello", "HelloService"));
Hello helloPort = helloService.getPort(Hello.class);
String hello = helloPort.getHello("World");
System.out.println(hello);
```

За замовчуванням повідомлення передаються по протоколу HTTP, причому при передачі вони упаковуються у конверти SOAP. Так, у прикладі вхідне повідомлення, представлене елементом getHello, в конверті виглядає так:

```
<!--від клієнта до сервера -->
<S:Envelope
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
 <S:Body>
  <ns2:getHello xmlns:ns2="http://samples/hello">
   <arg0>World</arg0>
  </ns2:getHello>
 </S:Body>
</S:Envelope>
```

Аналогічно, вихідне повідомлення, представлене елементом getHelloResponse, в конверті виглядає так:

```
<!--від сервера до клієнта -->
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header/>
  <S:Body>
   <ns2:getHelloResponse xmlns:ns2="http://samples/hello">
    <return>Hello, World!</return>
   </ns2:getHelloResponse>
  </S:Body>
</S:Envelope>
```

Є можливість явно задати імена елементів, які використовуються у вхідному та вихідному повідомленнях. Наприклад:

```
@RequestWrapper(localName = "hello-rq")
@ResponseWrapper(localName = "hello-rs")
@WebResult(name = "hello")
String getHello(@WebParam(name = "name") String name);
```

Із урахуванням перерахованих анотацій вхідне повідомлення в конверті буде мати вигляд:

```
<S:Envelope
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
 <S:Body>
  <ns2:hello-rq xmlns:ns2="http://samples/hello">
   <name>World</name>
  </ns2:hello-rq>
 </S:Body>
</S:Envelope>
```

А вихідне так:

```
<S:Envelope
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
 <S:Header/>
 <S:Body>
  <ns2:hello-rs xmlns:ns2="http://samples/hello">
   <hello>Hello, World!</hello>
  </ns2:hello-rs>
 </S:Body>
</S:Envelope>
```

Створення і розгортання простого Веб-сервісу та клієнта в Eclipse

Компоненти Веб-сервіса

З точки зору реалізації, Веб-сервіс можна представити у вигляді двох компонентів.

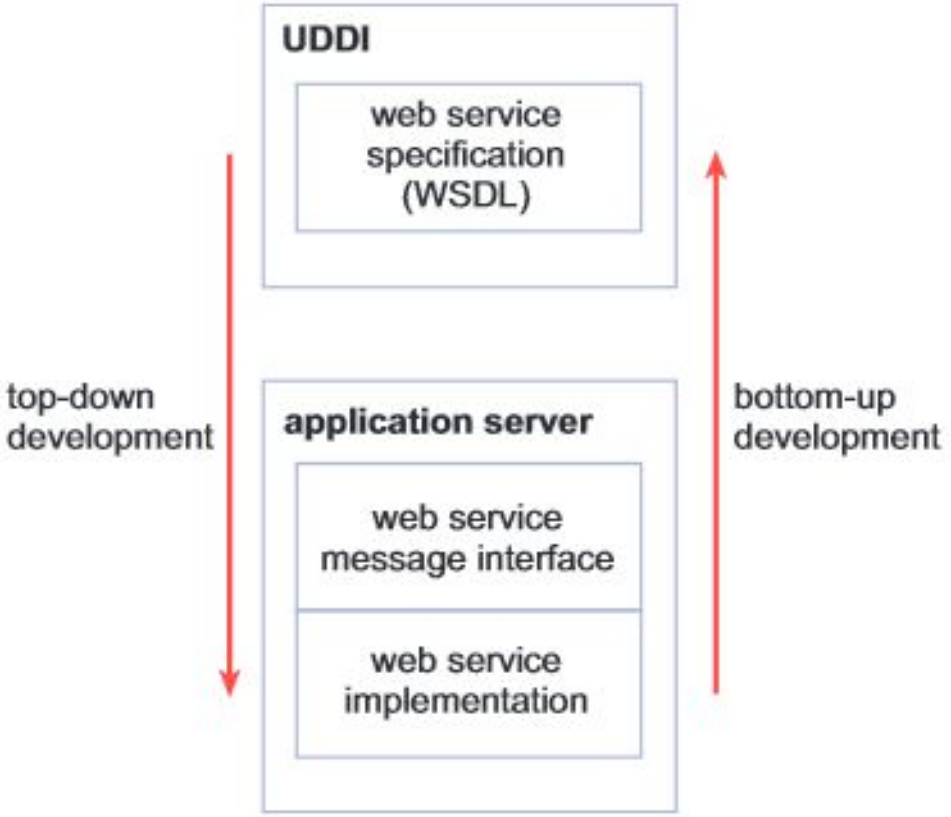
Деяка «логіка», яка забезпечує функціональність. Розглянемо, наприклад, простий сервіс для обчислення ПДВ, він буде містити деякий метод, що в якості вхідного параметру отримує певну суму в гривнях, це значення множиться на 20%, і повертається загальна сума разом із ПДВ.

Інтерфейс, який реалізується у форматі XML. В ньому визначено, як передається запит сервісу, і якою повинна бути очікувана відповідь або результат. Крім того, щоб використати реальний Веб-сервіс, потрібно реалізувати клієнт, який формує запит (на основі XML-інтерфейсу) і відображає результат, що повертається Веб-сервісом.

Методи розробки

Засоби автоматизації розробки Веб-сервісів поділяють на дві основні групи. При розробці знизу-вгору **(«bottom-up»)** спочатку пишуться імплементуючі класи, а з їх вихідного тексту генеруються WSDL-файли, що документують сервіс. Недоліком цього методу є властивість Java-класів часто змінюватись.

При підході зверху-вниз **(«top-down»)** спочатку готується WSDL, а з нього генерується скелет Java-класу, що імплементує сервіс.



Використання середовища Eclipse для написання Веб-сервісу

Плагіни Axis2

При роботі будемо використовувати два плагіни *Axis2,* насправді, явно тільки один з них: «Service Archive Generator Wizard».

Інший плагін «Code Generator Wizard» може бути використаний для реалізації розробки Веб-сервісу або зверху-вниз, або знизу-вгору. Він використовуватиметься у цьому додатку, але неявно. (Дистрибутиви можна скачати на сайті https://axis.apache.org/axis2/java/core/download.cgi)

Плагіни WTP

Web Tools Platform (*WTP*) є проектом Eclipse, який розширює діапазон інструментів для розробки, запуску і тестування Веб-сервісів. При розробці цього додатку будуть використані деякі можливості.

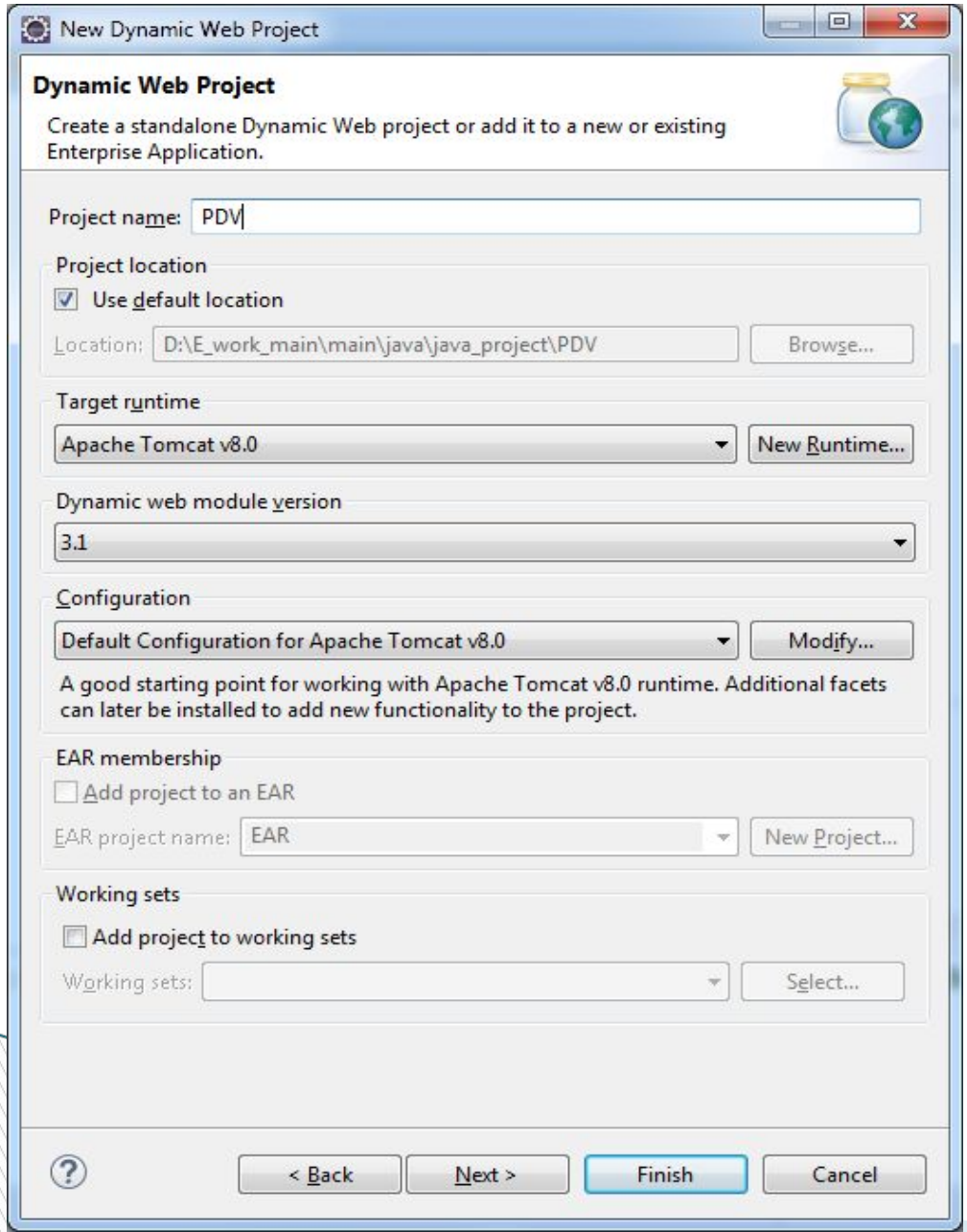
Плагін SoapUI

Цей плагін має ряд функцій, деякі з яких схожі на функціональності WTP. Тим не менш, плагін *SoapUI* використовують для перевірки Веб-сервісу на сумісність, яку WTP безпосередньо не підтримує.

Отже, щоб створити найпростіший Веб-сервіс, потрібно здійснити таку послідовність дій.

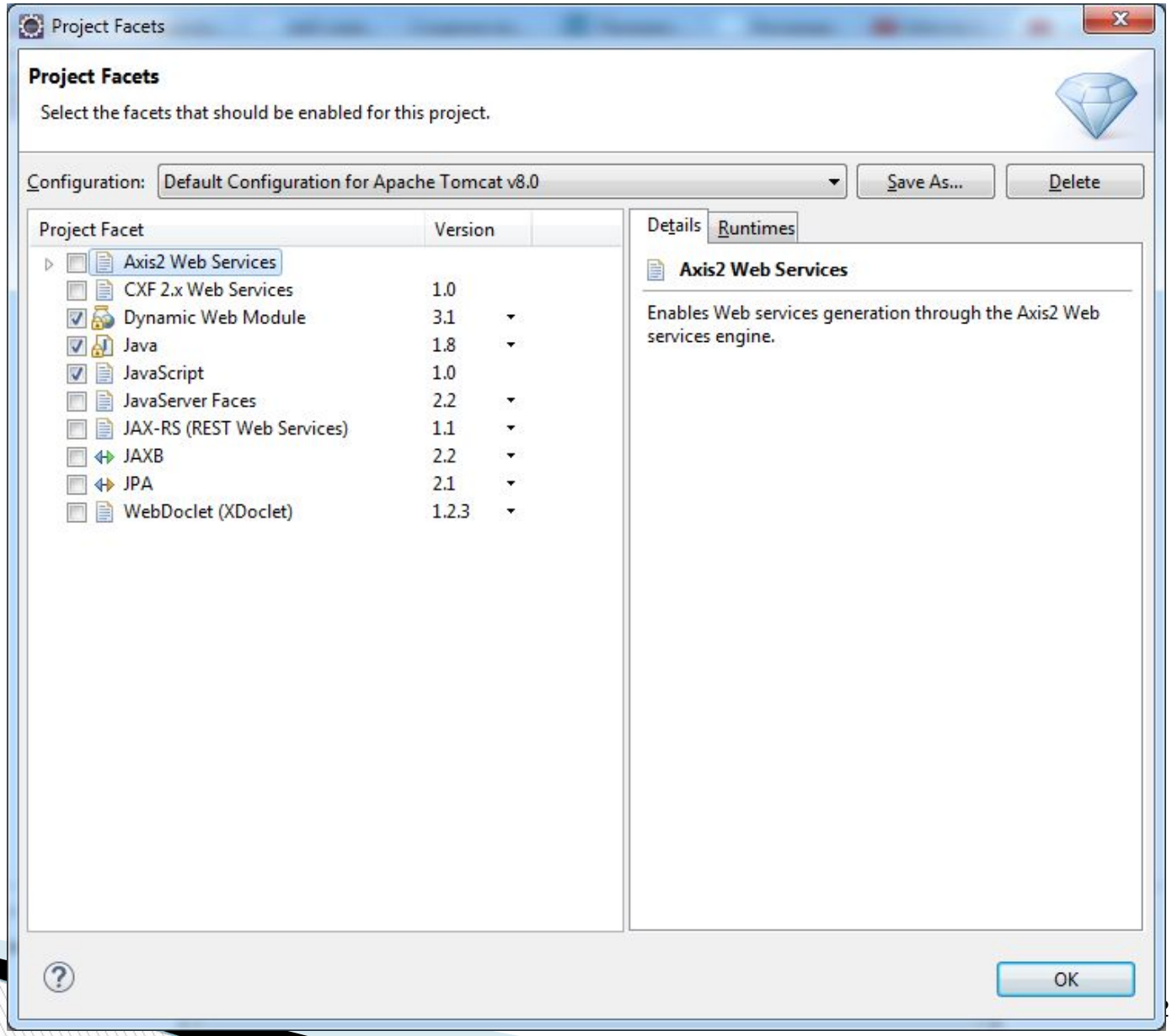
1. Створіть проект в Eclipse типу "Dynamic Web Project", де буде реалізовано ваш Веб-сервіс.
2. Напишіть Java-код ("бізнес-логіку"), який реалізує функціональність вашого Веб-сервісу.
3. Використайте Eclipse для автоматичного створення компонентів (WSDL і т.д.), щоб
перетворити Java-код у Веб-сервіс і потім запустити його.
4. Створіть ще один проект типу "Dynamic Web Project", який підтримуватиме клієнтський додаток, що буде використовуватись для доступу і тестування Веб-сервісу.
5. Використовуючи Eclipse, автоматично згенеруйте набір веб-сторінок, які функціонуватимуть в якості клієнтського інтерфейсу для виклику Веб-сервісу.
6. Використайте клієнтські веб-сторінки, щоб відправити запит на веб-сервіс і отримати відповідь.

Створення проекту



Першим етапом буде створення проекту для програмної реалізації Веб-сервісу. Виберіть File / New / Project / Dynamic Web Project, потім натисніть «Далі». Це дозволить запустити майстер створення веб-проекту. Введіть назву вашого проекту в наступному діалоговому вікні. В нашому випадку «PDV». Після цього можна натиснути кнопку "Finish" і завершити створення проекту, або натиснути кнопку "Next", щоб продовжити і подивитися, які варіанти доступні в налаштуванні проекту.

Якщо натиснути кнопку “Modify…”, відкриється діалогове вікно для вибору характеристик проекту



Написання програмної частини сервісу

Реалізуємо тепер "бізнес-логіку" або код обробки, який виконує завдання Веб-сервісу. Для цього створюємо новий пакет в «Java Resources / src» під назвою **chnu.cv.web**, а в ньому Java-клас **Pdv**, в якому реалізовано єдиний метод **pdv_result**. Java-код цього методу дуже простий.

```
package chnu.cv.web;

public class Pdv {
 public double pdv_result(float sum){
      return  sum+0.2*sum;
 }
}
```

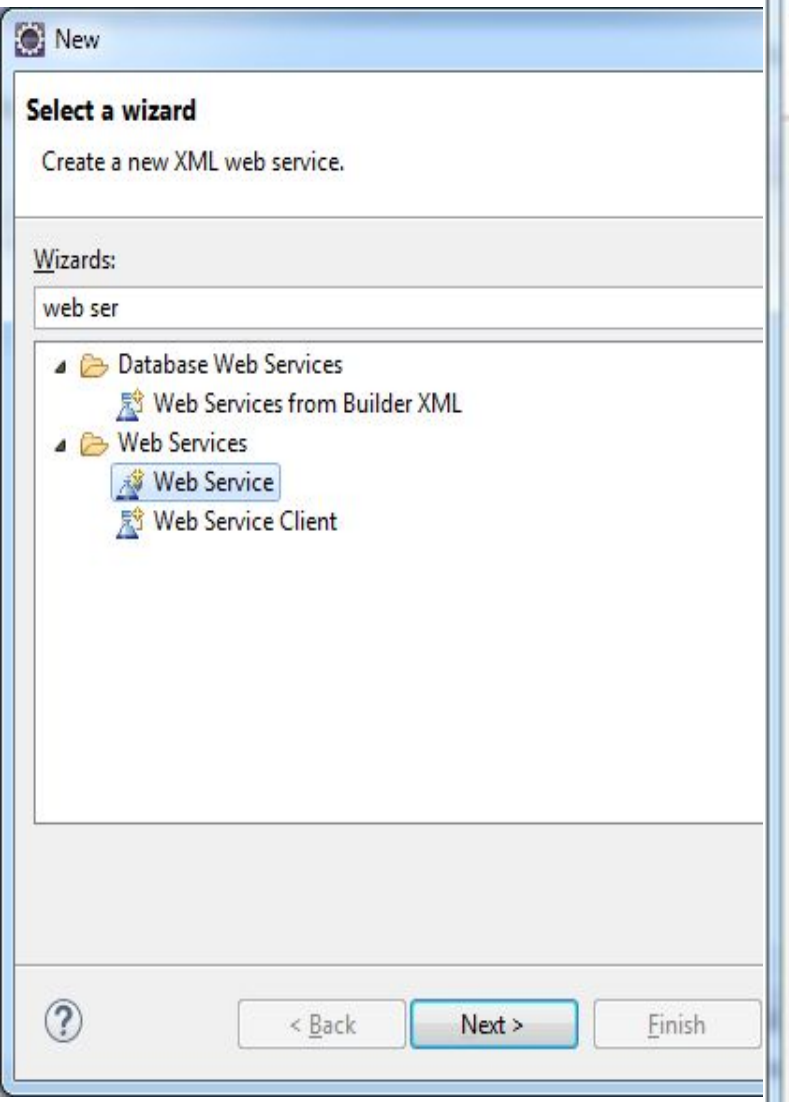
Слід зазначити, що пакет може містити довільну кількість класів, кожен з яких може мати декілька методів.

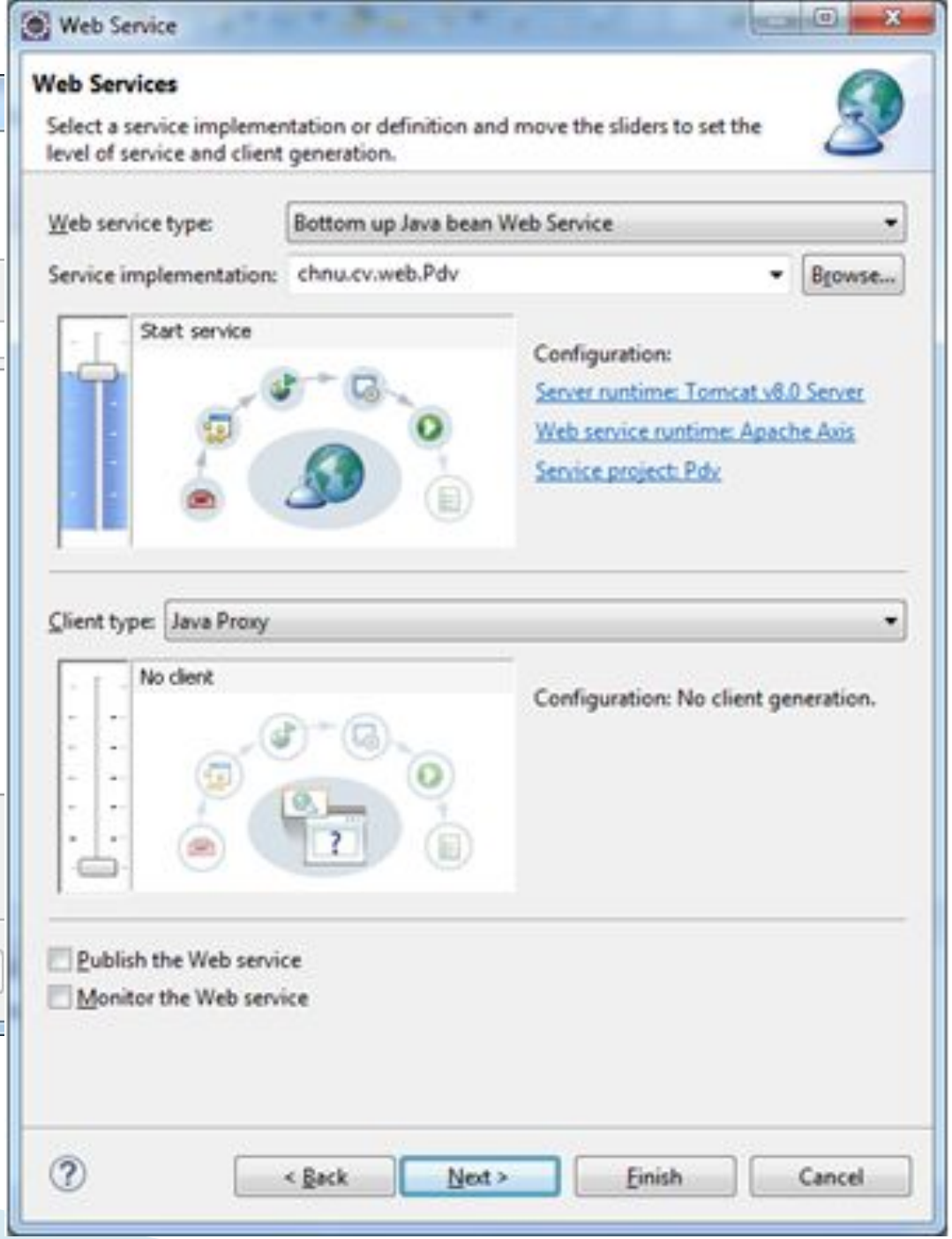
Створення Веб-сервісу і клієнта

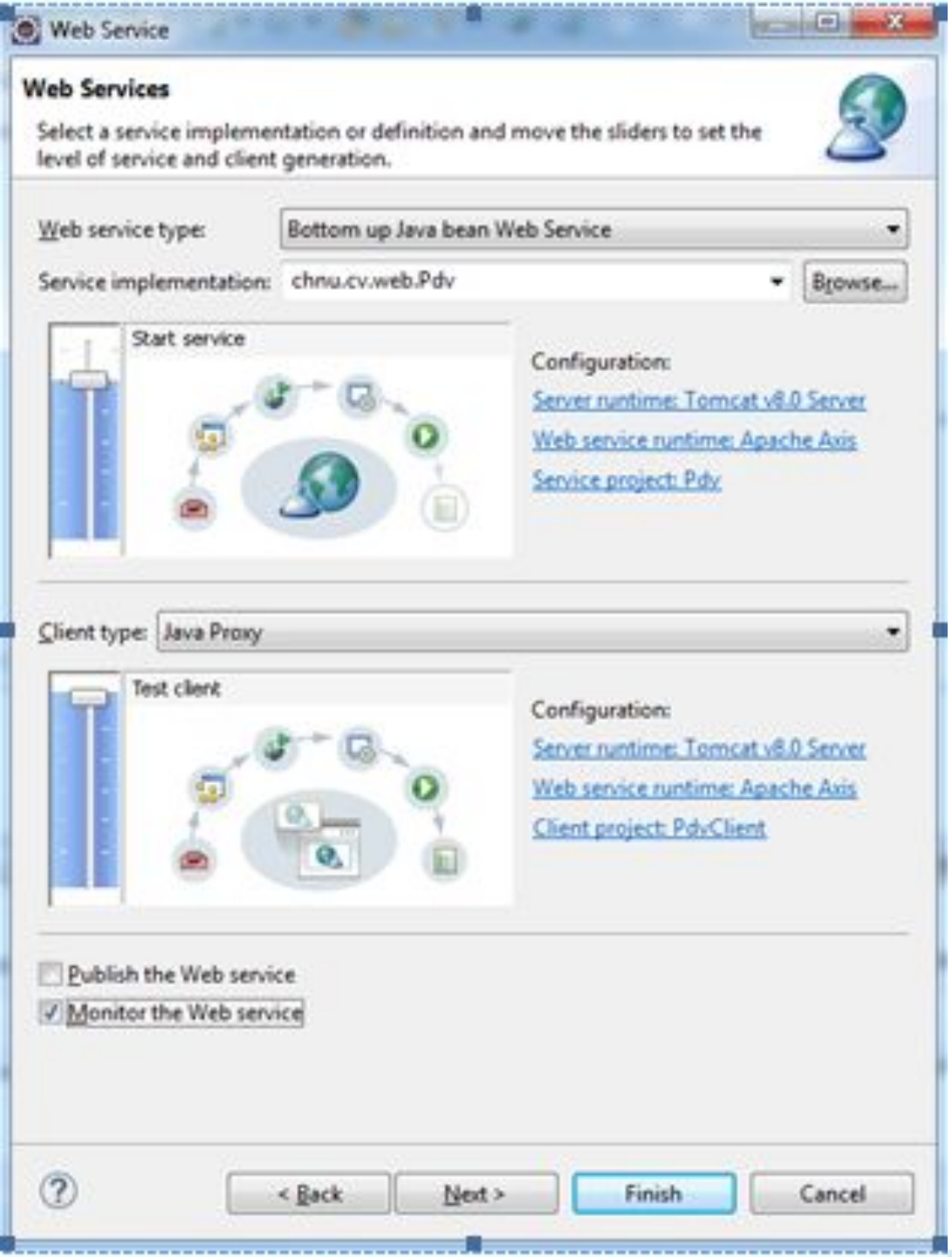
Тепер на основі створеного динамічного веб-проекту реалізуємо Веб-сервіс. Більшу частину дій Eclipse здійснює автоматично, а саме:

- упакування Веб-сервісу з інтерфейсом XML;
- розгортання Веб-сервісу на сервері згідно налаштувань ;
- запуск сервера додатків, якщо потрібно;
- формування набору сторінок, які можуть бути використані в якості простого клієнта, щоб перевірити роботу веб-сервіса.

Виберіть клас “Pdv.java” в Project Explorer і далі пункт меню File / New / Other...; потім відкрийте папку "Web Services" і виберіть "Web Service“. Потім натисніть «Далі». Наступне діалогове вікно має три секції


New
Select a wizard
Create a new XML web service.
Wizards:
web ser
Database Web Services
Web Services from Builder XML
Web Services
Web Service
Web Service Client
< Back
Next >
Finish


Web Service
Web Services
Select a service implementation or definition and move the sliders to set the level of service and client generation.
Web service type:
Bottom up Java bean Web Service
Service implementation:
chnu.cv.web.Pdv
Browse...
Start service
Configuration:
Server runtime: Tomcat v8.0 Server
Web service runtime: Apache Axis
Service project: Pdv
Client type:
Java Proxy
No client
Configuration: No client generation.
Publish the Web service
Monitor the Web service
< Back
Next >
Finish
Cancel

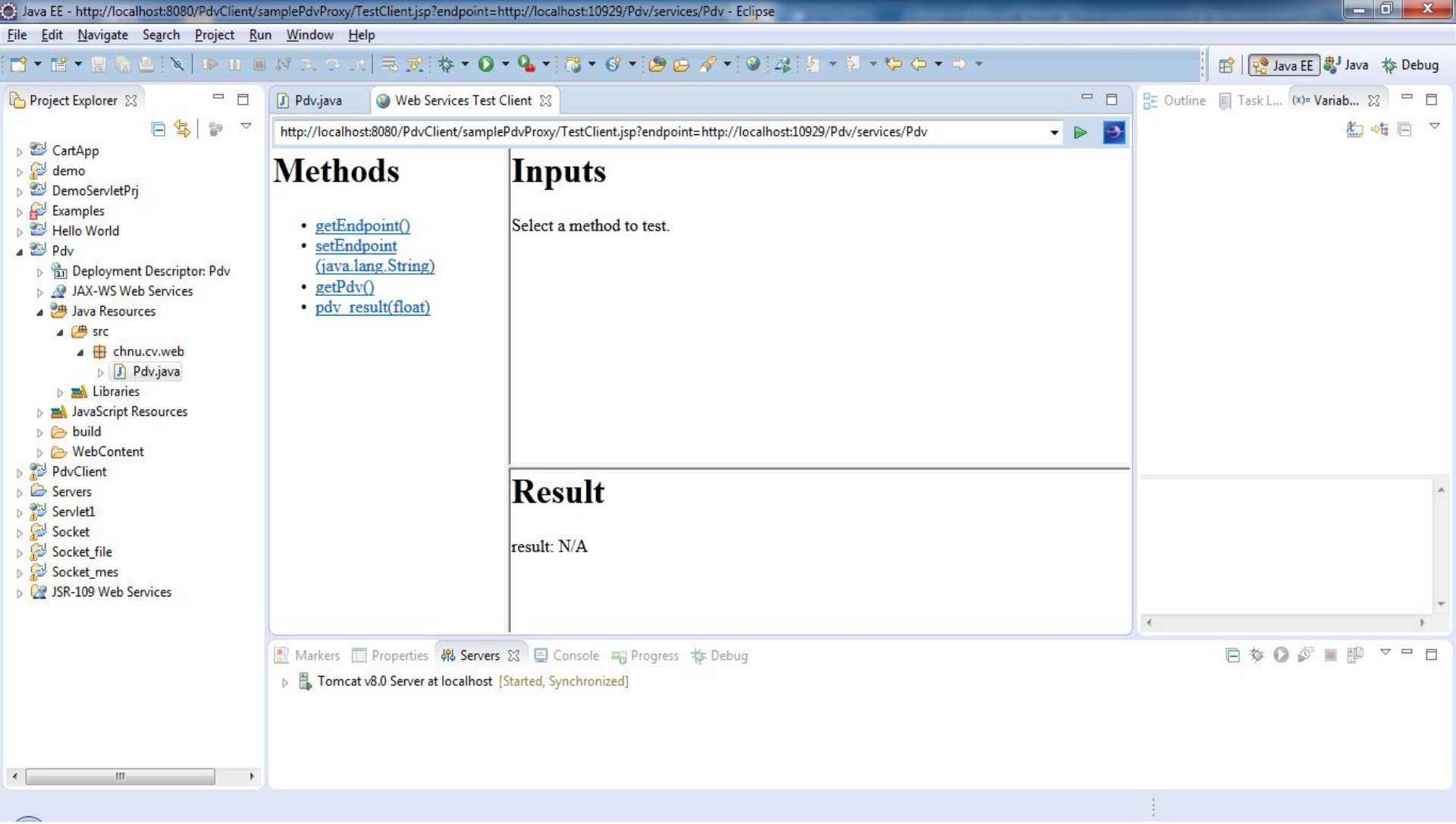


Повзунок для клієнта повинен бути встановлений на "Test client" рівні, таким чином, Eclipse згенерує клієнта, за допомогою якого можна протестувати сервіс. Зверніть увагу, що "Тип клієнта" вказаний як "Java Proxy". Java-проксі – це клас Java, який генерується в якості додатку для реалізації функціональних можливостей клієнта тестування. Це відбувається автоматично.

Перед тим як натиснути кнопку "Finish", відзначте прапорцем "Monitor the Web service". Це буде інструктувати Eclipse, що потрібно запустити TCP/IP Monitor, який буде відображати повідомлення між Eclipse (через проксі-сервер) і веб-сервісом. При тестуванні сервісу Ви зможете переглянути ці XML-повідомлення.

Коли ви натиснете кнопку «Finish», Eclipse почне створення і розгортання веб-сервісу і створить клієнт для тестування сервісу. Ці дії можуть зайняти деякий час.

Після того, як Eclipse закінчить створення веб-сервісу і клієнта, діалогове вікно конфігурації закриється, і ви побачите, що до списку в Project Explorer були додані два нові елементи



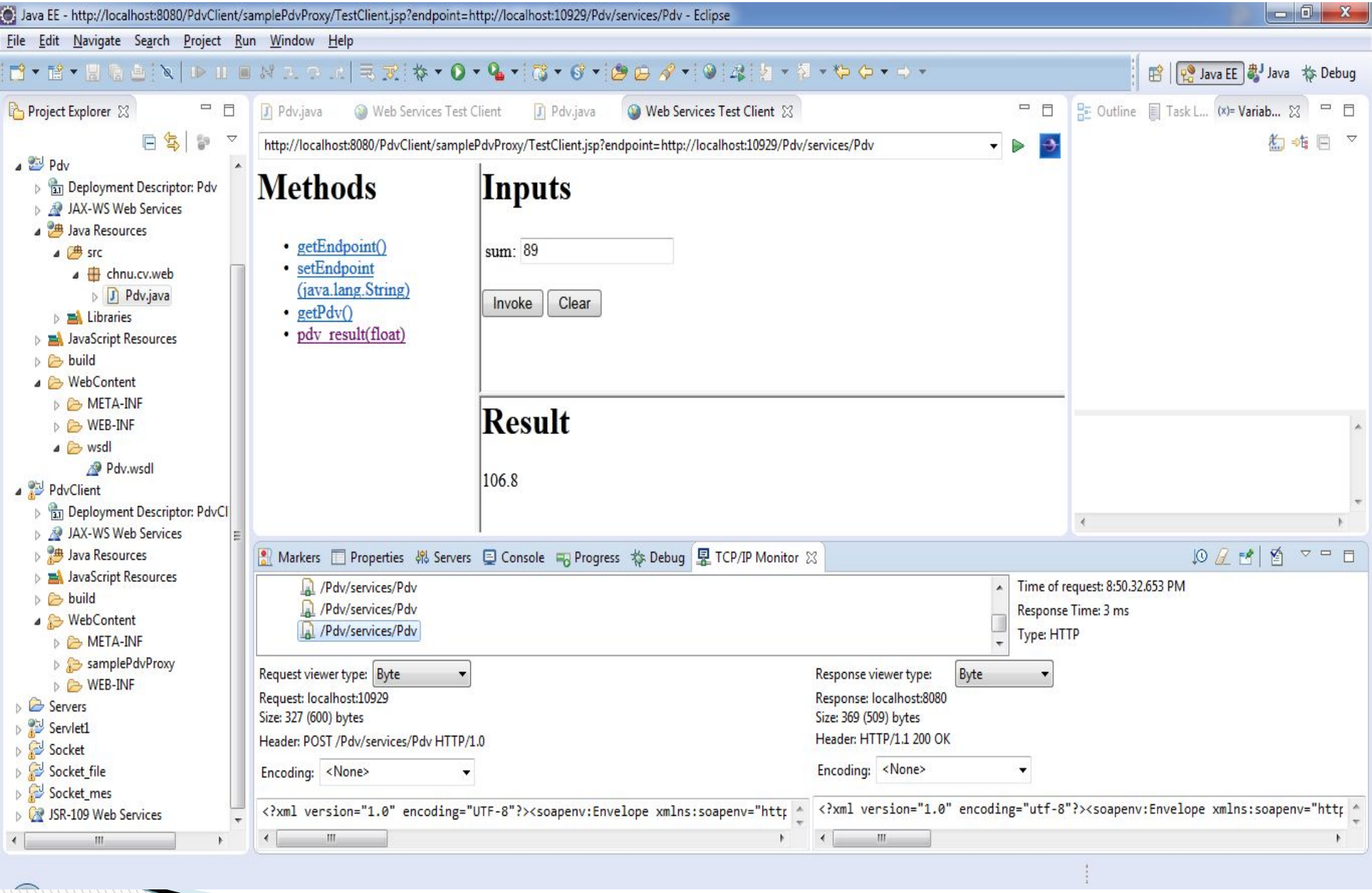
Перший із них – це Java-проект "PdvClient", який Eclipse створює для зберігання коду для згенерованого клієнта. Другий – нова папка "JSR-109 Web Services", що призначена для будь-яких створених файлів, які підтримують Java Specification Request (JSR) 109

Тестування Веб-сервісу
Використання клієнта

Тепер ви можете протестувати веб-сервіс. Панель Methods містить список дій, які може виконати веб-сервіс. Перші три в списку - «getEndpoint()», «setEndpoint (java.lang.String)» і «getPdv()» - методи, які Eclipse генерує за замовчуванням, так що деякими аспектами веб-сервісу можна керувати за допомогою клієнта тестування.

Кінцева точка Веб-сервісу в цьому випадку - це адреса (URL), за якою веб-сервіс може бути доступний в мережі. Ви також можете зустріти термін «SOAP endpoint». Знову ж таки, він стосується мережевої папки, в якій може бути доступним веб-сервіс, у даному випадку при відправленні SOAP-повідомлення до веб-сервісу.

Останній метод **«pdv_result(float)»** визначає Java-код веб-сервісу, який ви створили раніше.


Java EE - http://localhost:8080/PdvClient/samplePdvProxy/TestClient.jsp?endpoint=http://localhost:10929/Pdv/services/Pdv - Eclipse
File Edit Navigate Search Project Run Window Help
Java EE
Java
Debug
Project Explorer
Pdv
Deployment Descriptor: Pdv
JAX-WS Web Services
Java Resources
src
chnu.cv.web
Pdv.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
wsdl
Pdv.wsdl
PdvClient
Deployment Descriptor: PdvCl
JAX-WS Web Services
Java Resources
JavaScript Resources
build
WebContent
META-INF
samplePdvProxy
WEB-INF
Servers
Servlet1
Socket
Socket_file
Socket_mes
JSR-109 Web Services
Pdv.java
Web Services Test Client
Pdv.java
Web Services Test Client
http://localhost:8080/PdvClient/samplePdvProxy/TestClient.jsp?endpoint=http://localhost:10929/Pdv/services/Pdv
Methods
getEndpoint()
setEndpoint (java.lang.String)
getPdv()
pdv_result(float)
Inputs
sum: 89
Invoke
Clear
Result
106.8
Outline
Task L...
(x)= Variab...
Markers
Properties
Servers
Console
Progress
Debug
TCP/IP Monitor
/Pdv/services/Pdv
/Pdv/services/Pdv
/Pdv/services/Pdv
Time of request: 8:50.32.653 PM
Response Time: 3 ms
Type: HTTP
Request viewer type: Byte
Request: localhost:10929
Size: 327 (600) bytes
Header: POST /Pdv/services/Pdv HTTP/1.0
Encoding: <None>
Response viewer type: Byte
Response: localhost:8080
Size: 369 (509) bytes
Header: HTTP/1.1 200 OK
Encoding: <None>
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http
<?xml version="1.0" encoding="utf-8"?><soapenv:Envelope xmlns:soapenv="http

Моніторинг повідомлень веб-сервісу

У нижній частині вікна Eclipse ви побачите, що запит, відправлений Веб-сервісу, і відповідь від сервіса відображаються в TCP/IP Monitor. Eclipse виводить ці повідомлення в одному рядку. Запит виглядає наступним чином:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
      xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Body>
            <pdv_result xmlns="http://web.cv.chnu">
                  <sum>89.0</sum>
            </pdv_result>
      </soapenv:Body>
</soapenv:Envelope>
```

Як можна бачити, це SOAP конверт з досить простим тілом. Власний простір імен додатку побудований з використанням Java-пакету класу Java. Назва Java-методу «pdv_result» була використана як SOAP-тег, що включає інший тег, який є назвою змінної sum.

SOAP-відповідь дуже подібна, тільки дещо інші елементи тіла:

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd=http://www.w3.org/2001/XMLSchema
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
      <pdv_resultResponse xmlns="http://web.cv.chnu">
            <pdv_resultReturn>106.8</pdv_resultReturn>
      </pdv_resultResponse>
    </soapenv:Body>
 </soapenv:Envelope>
```

Опис сервісу

Кінцева точка Веб-сервісу

Ви пройшли основні етапи створення простого веб-сервісу. Розглянемо деякі важливі моменти. Раніше було зазначено, що панель Methods в Eclipse, крім методу pdv_result, містить й інші. Кінцева точка веб-сервісу – це URL-адреса, за якою знаходиться сервіс, і за якою він може бути доступним. Якщо ви натиснете на «getEndpoint()», а потім на кнопку «Invoke», URL-адреса, яка буде відображатися на панелі результатів матиме вигляд: **http://localhost:8080/PDV/services/Pdv.**

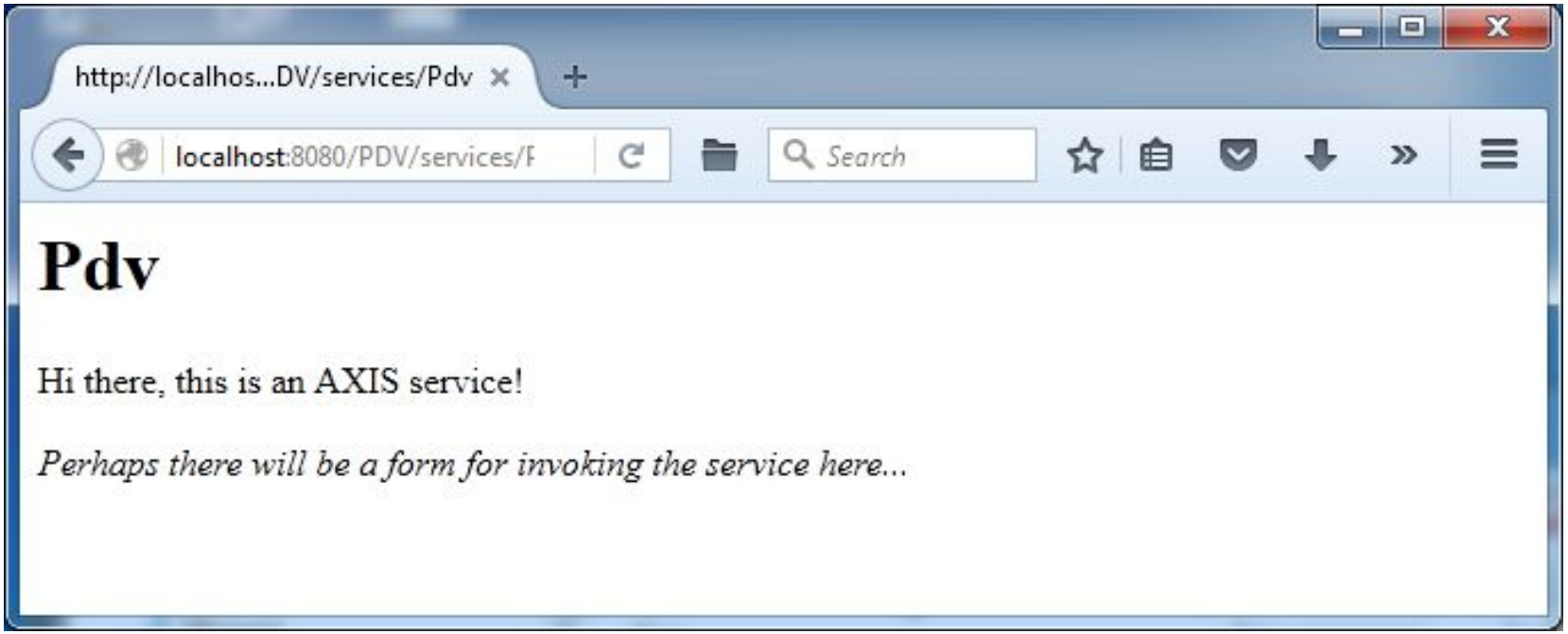
Хоча сервіс знаходиться за цією адресою, але інтерфейс, який ви використовуєте, не доступний за цією адресою. У верхній частині панелі Web Services Test Client можна побачити реальну URL-адресу, що використовується для відображення тестового клієнта. Ця URL-адреса буде також показана на панелі заголовку вікна Eclipse.

http://localhost:8080/PDVClient/samplePdvProxy/TestClient.jsp?endpoint= http://localhost:8080/PDV/services/Pdv.

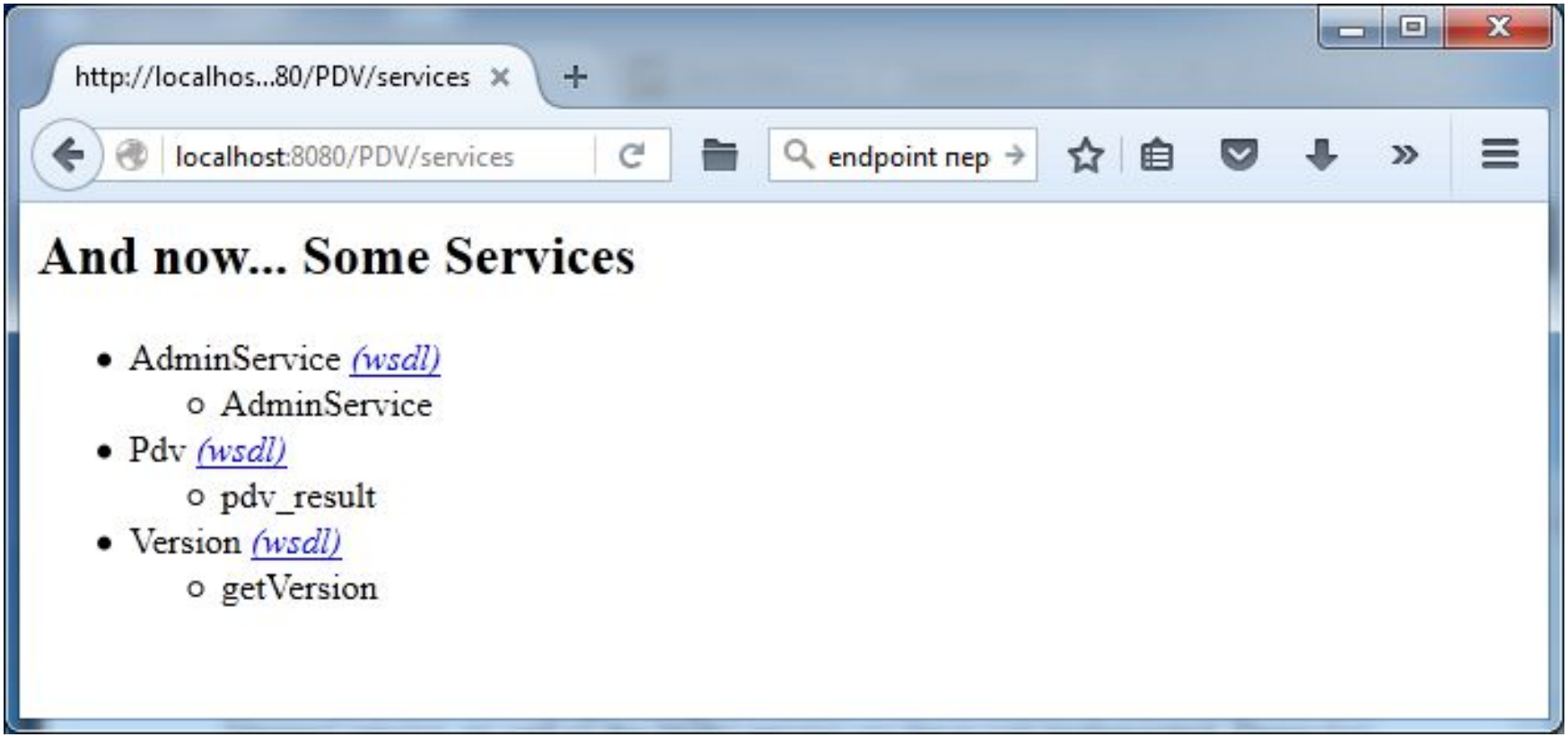
Завершальна частина URL **http://localhost:8080/PDV/services/Pdv** служить параметром, який передається цій JSP. Цей параметр є адресою кінцевої точки, яка використовується в JSP для доступу до веб-сервісу при створенні запиту.

Axis ma WSDL

При створенні Веб-сервісу в Eclipse використовуються загальнодоступна бібліотека **Axis**. Якщо ви задасте реальну URL-адресу веб-сервісу **(http://localhost:8080/PDV/services/Pdv)** у веб-браузері, ви побачите згенеровану веб-сторінку, яка вказує на те, що використовується бібліотека Axis, і що форма для доступу до сервісу може бути створена вами пізніше



Axis також генерує сторінку, де перераховані всі доступні веб-сервіси і пов’язані з ними документи WSDL. Наприклад, перелік сервісів надається за адресою: **http://localhost:8080/PDV/services**



Так виглядає WSDL для нашого Веб-сервісу Pdv:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://web.cv.chnu"
xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://web.cv.chnu"
xmlns:intf="http://web.cv.chnu" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
 <wsdl:types>
  <schema elementFormDefault="qualified" targetNamespace="http://web.cv.chnu"
xmlns="http://www.w3.org/2001/XMLSchema">
   <element name="pdv_result">
    <complexType>
     <sequence>
      <element name="sum" type="xsd:float"/>
     </sequence>
    </complexType>
   </element>
   <element name="pdv_resultResponse">
    <complexType>
     <sequence>
      <element name="pdv_resultReturn" type="xsd:double"/>
     </sequence>
    </complexType>
   </element>
  </schema>
 </wsdl:types>
```

```
    <wsdl:message name="pdv_resultRequest">
       <wsdl:part element="impl:pdv_result" name="parameters">
       </wsdl:part>
    </wsdl:message>
    <wsdl:message name="pdv_resultResponse">
       <wsdl:part element="impl:pdv_resultResponse" name="parameters">
       </wsdl:part>
    </wsdl:message>
    <wsdl:portType name="Pdv">
       <wsdl:operation name="pdv_result">
          <wsdl:input message="impl:pdv_resultRequest" name="pdv_resultRequest">
        </wsdl:input>
          <wsdl:output message="impl:pdv_resultResponse" name="pdv_resultResponse">
        </wsdl:output>
       </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="PdvSoapBinding" type="impl:Pdv">
      <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
       <wsdl:operation name="pdv_result">
          <wsdlsoap:operation soapAction=""/>
          <wsdl:input name="pdv_resultRequest">
             <wsdlsoap:body use="literal"/>
          </wsdl:input>
          <wsdl:output name="pdv_resultResponse">
             <wsdlsoap:body use="literal"/>
          </wsdl:output>
       </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="PdvService">
       <wsdl:port binding="impl:PdvSoapBinding" name="Pdv">
          <wsdlsoap:address location="http://localhost:8080/PDV/services/Pdv"/>
       </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Eclipse та Axis забезпечують дуже хороший рівень підтримки для створення і опису веб-сервісів, а саме:

створення логіки Веб-сервісу й розгортання її на сервері додатків;

створення простого клієнта і використання його для тестування сервісу;

забезпечення обміну повідомленнями SOAP;

створення WSDL, які могли б бути використані іншими клієнтами для доступу до створеного Веб-сервісу.

Зверніть увагу, що файл WSDL у вашому проекті міститься у папці WebContent, він буде потрібний при опублікуванні Веб-сервісу в службі каталогів (UDDI).

Створення клієнта на основі WSDL

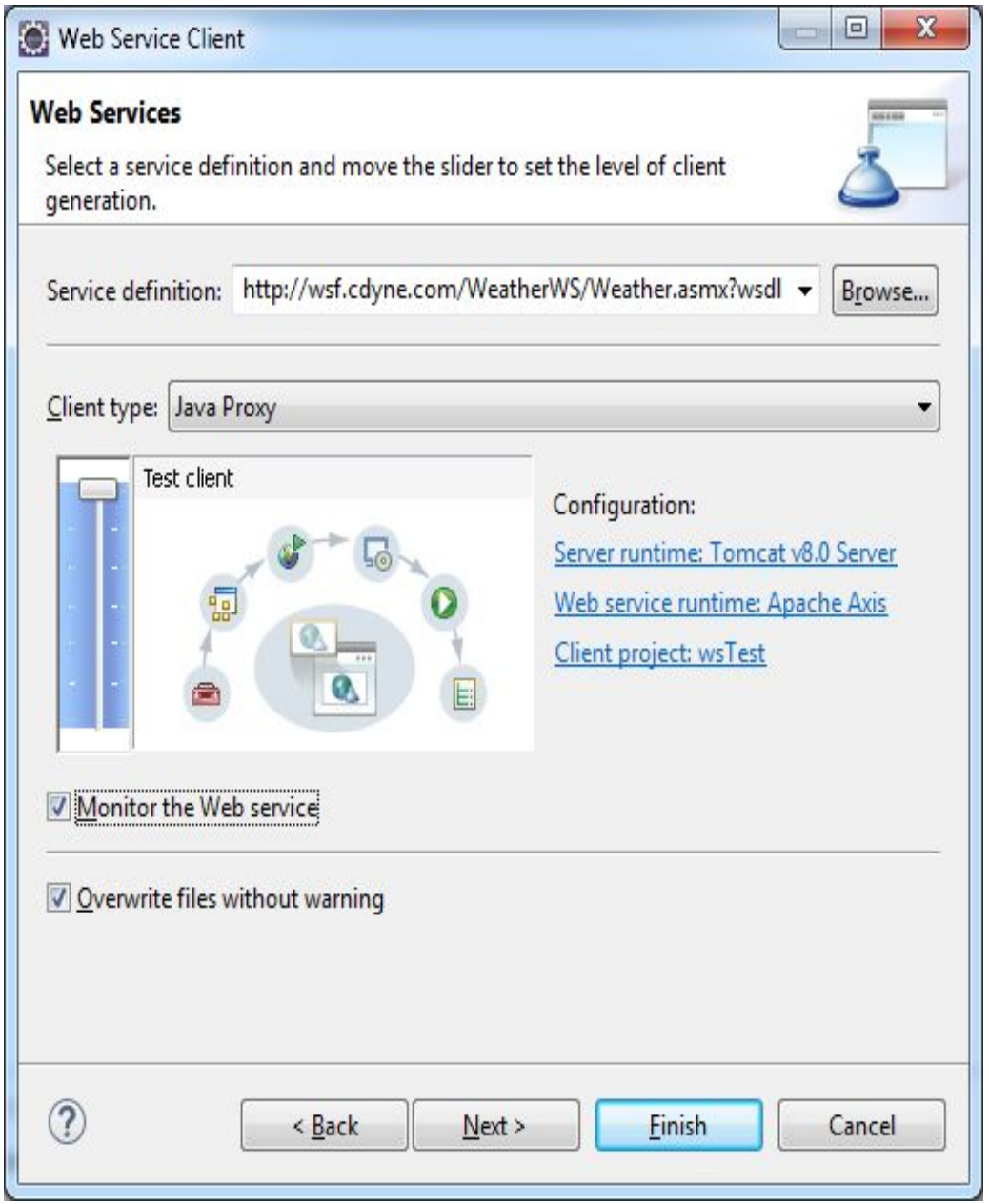
Зауважимо, що WSDL для створення клієнта може бути розташований локально на вашому комп'ютері або ж на іншій машині й доступний через мережу Інтернет.

Eclipse включає бібліотеку Web Tools Platform (WTP), яка забезпечує підтримку створення клієнта на основі вмісту WSDL.

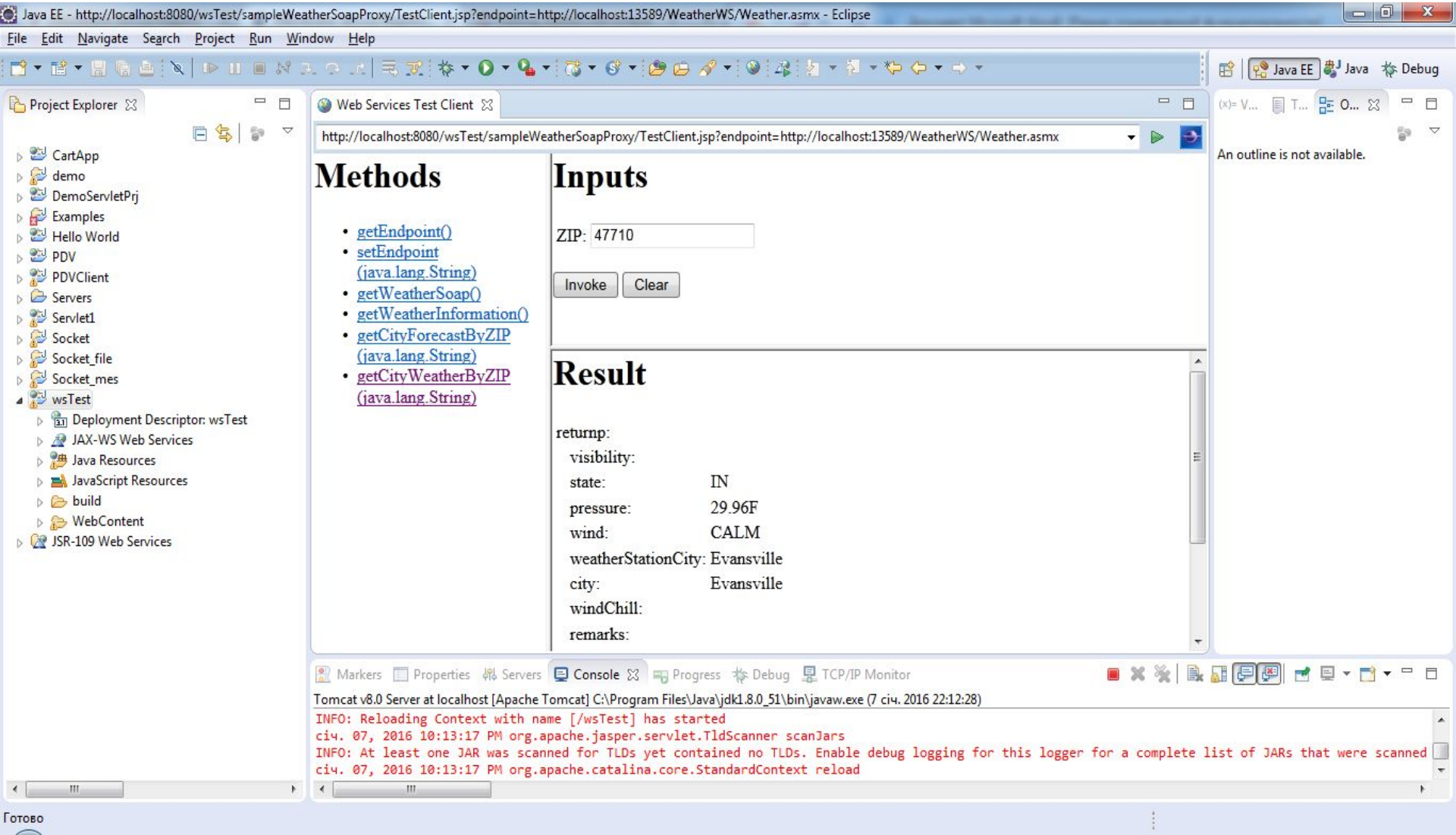
В мережі Інтернет є багато сайтів, які надають вільний доступ до користування Веб-сервісами, й крім того, дозволяють реєструвати власні сервіси. Розглянемо один із таких сайтів-репозиторіїв, а саме: http://service-repository.com. Скористаємось сервісом *Weather*. Його опис міститься у WSDL-файлі (http://wsf.cdyne.com/WeatherWS/Weather.asmx?WSDL). Цей сервіс дозволяє дізнатися про поточні погодні умови та прогноз погоди у містах США за ZIP-кодом.

Отже, щоб створити клієнт на основі цього WSDL, потрібно здійснити наступні кроки:

- Запустити Eclipse і створити новий проект wsTest (File / New / Other / Dynamic Web Project).
- Створити новий клієнт Веб-сервісу (File / New / Other / Web Services / Web Service Client). В цьому випадку ми використовуємо доступний Веб-сервіс погоди, про який говорилось вище. В рядку Service definition вказуємо URL для WSDL.

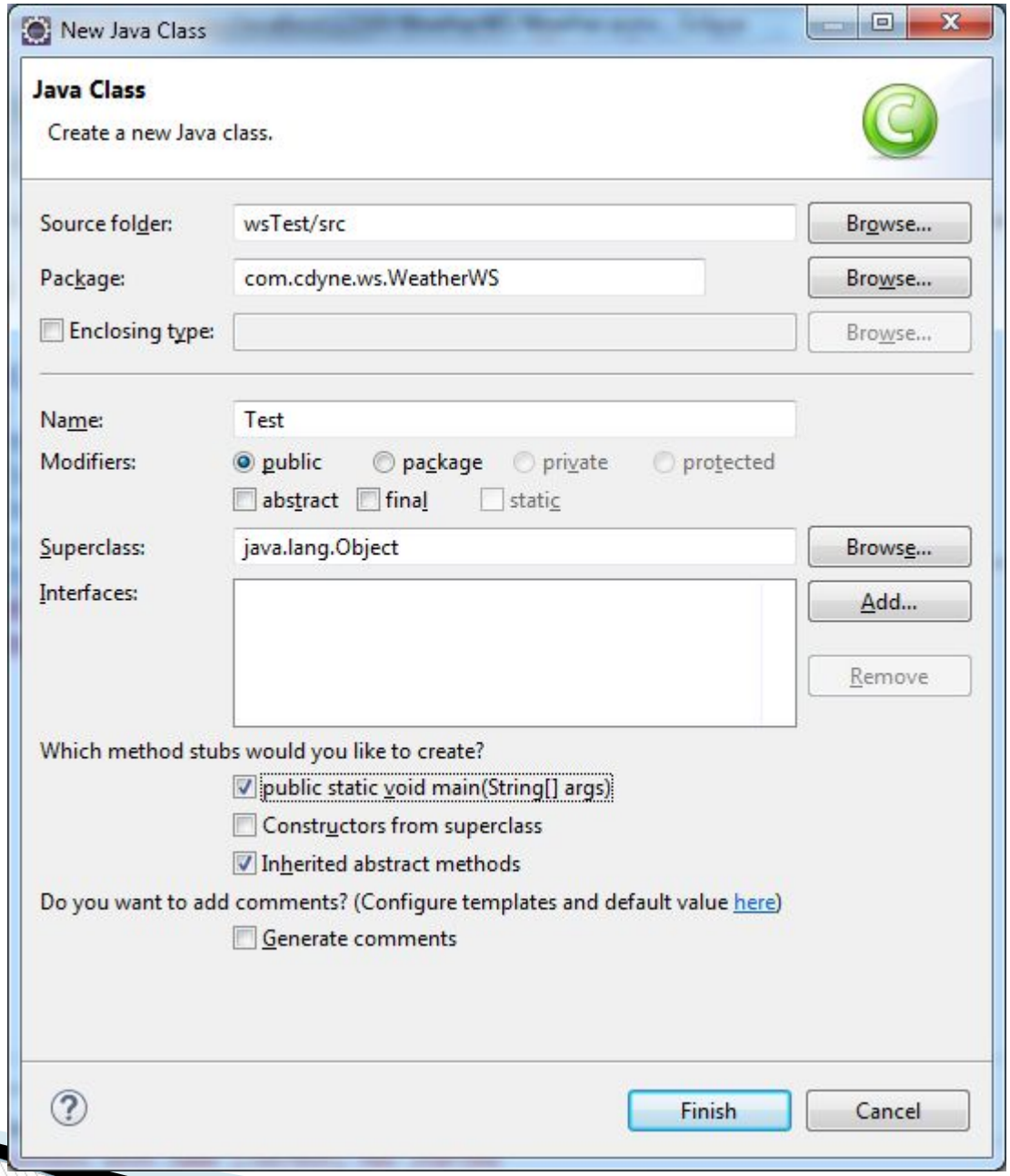


3. Натиснути кнопку «Next», запустити сервер і на ньому клієнт для тестування Веб-сервісу.
4. Вибрати, наприклад, метод **GetCityWeatherByZIP**(). Отримаємо наступні результати:



5. Створимо власний клас **Test.java**, щоб продемонструвати, як отримати доступ до потрібних даних (File / New / Class). Цей клас буде містити наступний код:

```
import com.cdyne.ws.WeatherWS.*;
import java.net.URL;
 public class Test
{
 public static void main(String[] arg)
 {
  try
  {
   WeatherSoapStub service = (WeatherSoapStub)new
WeatherLocator().getWeatherSoap(new
URL("http://wsf.cdyne.com/WeatherWS/Weather.asmx"));
   WeatherReturn weather = service.getCityWeatherByZIP("47710");
   System.out.println("Location: " + weather.getCity() + ", " +
weather.getState());
   System.out.println("Description: " + weather.getDescription());
   System.out.println("Temperature: " + weather.getTemperature() + " degrees");
  }
  catch (Exception e)
  {
   e.printStackTrace();
  }
 }
}
```


New Java Class
Java Class
Create a new Java class.
Source folder:
wsTest/src
Browse...
Package:
com.cdyne.ws.WeatherWS
Browse...
Enclosing type:
Browse...
Name:
Test
Modifiers:
public
package
private
protected
abstract
final
static
Superclass:
java.lang.Object
Browse...
Interfaces:
Add...
Remove
Which method stubs would you like to create?
public static void main(String[] args)
Constructors from superclass
Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
Generate comments
Finish
Cancel

Зверніть увагу, що в рядку 11 ми прописуємо URL-адресу кінцевої точки. Вона, як правило, така ж, як URL WSDL, але не містить рядковий параметр запиту **?WSDL**.

6. Виконати код (натисніть Run / Run). Ви повинні побачити результат, подібний до цього:

Location: Evansville, IN
Description: Sunny
Temperature: 79 degrees

Приклад створення Веб-сервісу з використанням Axis2 і зверненням до бази даних

Постановка задачі

Необхідно розробити веб-сервіс, що надає користувачам інформацію про погодні умови в містах світу. Інформація про погоду зберігається в БД і включає в себе три характеристики: температура, хмарність, опади. Користувачеві-клієнту надається веб-інтерфейс де він має можливість вибрати місто і отримати інформацію про погоду в цьому місті.

- База даних (Derby DB).
- Веб-сервіс (Tomcat, Axis2).
- Веб-додаток, що складається з JSP-сторінок і сервлетів.
- Робоче місце клієнта (браузер).

Для вирішення поставленого завдання необхідно виконати наступні кроки:

1. Створити новий проект для веб-сервісу.
3. Створити таблицю ***weather*** і заповнити її даними про міста і погодні умови.
4. Створити Java-клас ***WeatherInfo*** для представлення даних про погоду.
5. Розробити веб-сервіс ***WeatherWS***, що містить метод ***getWeatherInfo***(), який приймає як параметр назву міста і повертає дані у вигляді об'єкта ***WeatherInfo***. Також сервіс повинен містити метод ***getCities***() для отримання списку міст.
6. Розгорнути веб-сервіс на сервері додатків.
7. Створити новий проект для веб-клієнта.
8. Використовуючи WSDL-опис розгорнутого веб-сервісу згенерувати класи-заглушки для доступу до веб-сервісу.
9. Розробити клас-фільтр, що забезпечує отримання від веб-сервісу списку міст і передачу його JSP-сторінці.

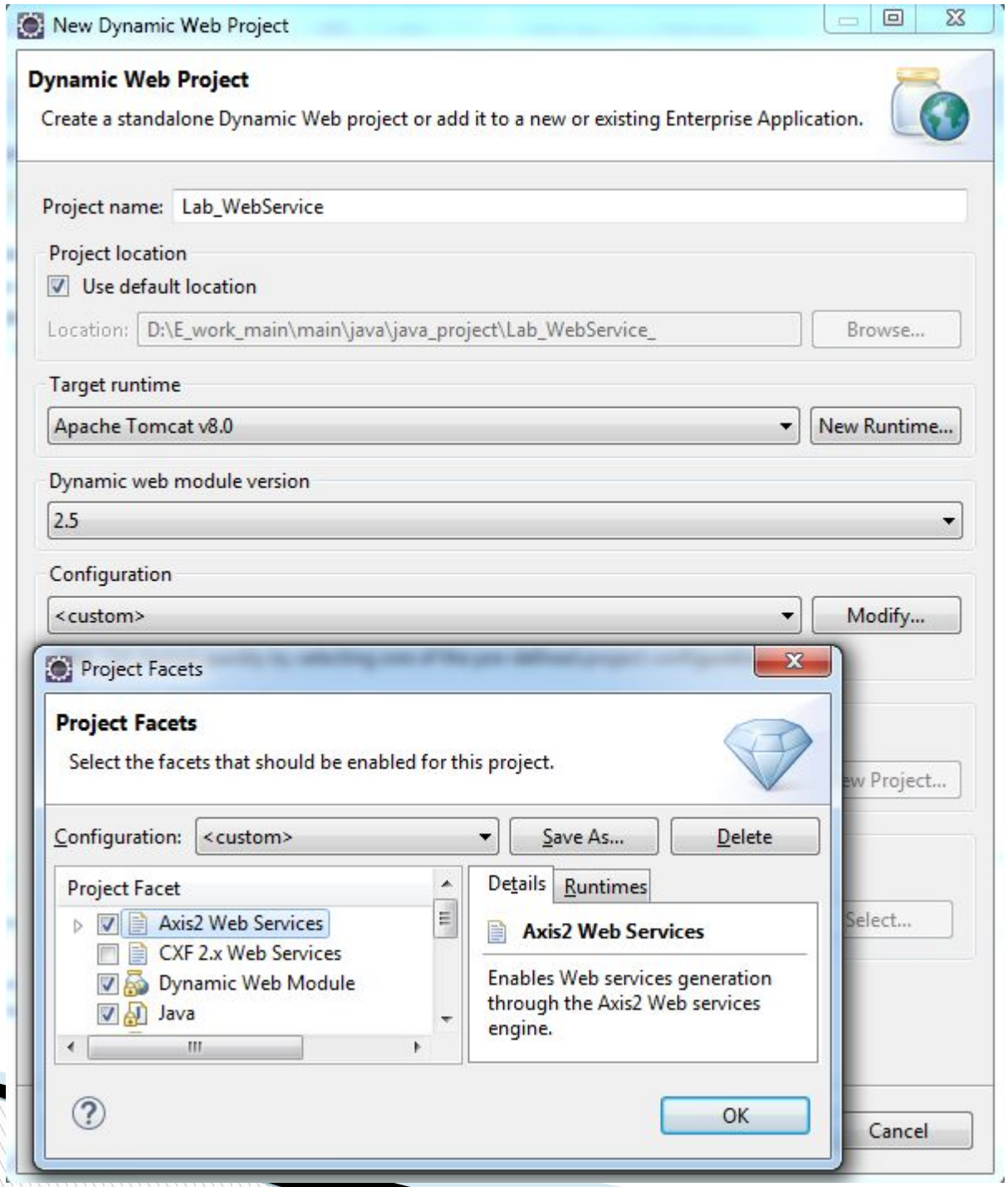
- Розробити JSP сторінку для введення даних і відображення результатів.

2. Розробити сервлет, що забезпечує виклик веб-сервісу для отримання інформації про погоду по обраному місту.
3. Запакувати веб-додаток і розгорнути на сервері.
4. Протестувати роботу програми в браузері.

Створення нового проекту для веб-сервісу

1) Виберіть пункт меню File / New / Other…, у вікні вибору типу проекту вкажіть Web / Dynamic Web Project і натисніть Next.

2) Вкажіть ім'я проекту ***Lab_WebService***. У полях Target Runtime і Configurations повинна бути обрана конфігурація сервера Tomcat v8.0. При натисненні кнопкиModify, з’явиться вікно Project Facets, в якому, крім налаштувань по замовчуванню, потрібно вибрати Axis2 Web Services. Перед натисненням кнопки Finish, відзначити автоматичну генерацію файлу web.xml.


New Dynamic Web Project
Dynamic Web Project
Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.
Project name: Lab_WebService
Project location
Use default location
Location: D:\E_work_main\main\java\java_project\Lab_WebService_
Browse...
Target runtime
Apache Tomcat v8.0
New Runtime...
Dynamic web module version
2.5
Configuration
<custom>
Modify...
Project Facets
Project Facets
Select the facets that should be enabled for this project.
Configuration: <custom>
Save As...
Delete
Project Facet
Axis2 Web Services
CXF 2.x Web Services
Dynamic Web Module
Java
Details
Runtimes
Axis2 Web Services
Enables Web services generation through the Axis2 Web services engine.
OK
Cancel

Створення таблиці weather і вставка тестових даних

1) Підключіться до БД Derby і запустіть сервер БД (аналогічно до того, як це було зроблено раніше).

2) Для зберігання SQL-скриптів створимо новий файл ***weather.sql*** у папці C:\Apache\db-derby-10.10.2.0-bin\bin з наступними командами:

```
-- підключення
connect
'jdbc:derby://localhost:1527/myDB;create=true;user=me;password=min
e';
-- розкоментуйте наступний рядок, якщо треба перестворити таблицю
--drop table weather;
-- створення таблиці
create table weather(city varchar(20), temperature integer,
cloudiness varchar(20), precipitations varchar(20));
-- вставка тестових даних
insert into weather values ('Kyiv', -5, 'heavy', 'snow');
insert into weather values ('London', 0, 'average', 'rain');
insert into weather values ('Tokyo', 10, 'heavy', 'rain');
insert into weather values ('Paris', 5, 'light', 'no');
-- вибрати все із таблиці для перевірки
select * from weather;
-- відключення і вихід
disconnect;
exit;
```

3) Збережіть файл.
4) Виконайте команду: **ij>run ‘weather.sql’;.**
5) У випадку вдалого виконання скрипта буде виведено таблицю зданими погоди по містах.

CITY	TEMPERATURE	CLOUDINESS	PRECIPITATIONS
Kyiv	-5	heavy	snow
London	0	average	rain
Tokyo	10	heavy	rain
Paris	5	light	no

Створення Java-класу WeatherInfo для представлення та даних про погоду

1) Клацніть правою кнопкою миші на значок проекту Lab_WebService і виберіть New / Class. Як ім'я класу (Name) вкажіть *WeatherInfo*, а в якості імені пакета (Package) задайте *chnu.javaeelabs*.

2) Клас *WeatherInfo* повинен містити чотири поля, відповідні стовпцям таблиці *weather*. Так як складний об'єкт *WeatherInfo* призначений для трансляції в SOAP-повідомлення для подальшої передачі по мережі, то він повинен відповідати таким основним вимогам:

- містити конструктор за умовчанням;
- містити поля простих типів, або складних типів, що відповідають правилам;
- відповідати вимогам специфікації JavaBeans, тобто реалізовувати методи get / set для доступу до полів;
- не містити статичних полів і методів.

Повний код класу наведено нижче:

```java
package chnu.javaeelabs;
public class WeatherInfo {
  private String city;
  private int temperature;
  private String cloudiness;
  private String precipitations;
        // Конструктор за замовчуванням
  public WeatherInfo() {}

  public WeatherInfo(String city, int temperature, String cloudiness, String precipitations) {
        this.city = city;
        this.temperature = temperature;
        this.cloudiness = cloudiness;
        this.precipitations = precipitations;
  }
  public String getCity() {
        return city;
  }
  public void setCity(String city) {
        this.city = city;
  }
  public int getTemperature() {
        return temperature;
  }
  public void setTemperature(int temperature) {
        this.temperature = temperature;
  }
  public String getCloudiness() {
        return cloudiness;
  }
  public void setCloudiness(String cloudiness) {
        this.cloudiness = cloudiness;
  }
  public String getPrecipitations() {
        return precipitations;
  }
  public void setPrecipitations(String precipitations) {
        this.precipitations = precipitations;
  }
}
```

Розробка веб-сервісу WeatherWS

1) Клацніть правою кнопкою миші на пакет ***chnu.javaeelabs*** і виберіть New / Class. Створіть новий клас з ім'ям ***WeatherWS***.

2) Код класу ***WeatherWS*** містить реалізацію методу ***getWeatherInfo***(), що приймає як параметр назву міста і повертає дані у вигляді об'єкта ***WeatherInfo***; даний метод буде зареєстрований у WSDL-файлі і є методом веб-сервісу, доступним для виклику віддаленими клієнтами.

Повний код класу ***WeatherWS*** наведено нижче:

```
package chnu.javaeelabs;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;

public class WeatherWS {
	// Допоміжний метод отримання з'єднання
	private Connection getConnection() throws Exception {
		// Завантаження драйвера БД Derby
		Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance();
		// Отримання з'єднання з БД
		return
DriverManager.getConnection("jdbc:derby://localhost:1527/myDB;create=true;user=me;password=mine");
	}
	/**
	 * Повертає список міст, по яких
	 * є інформація про погоду
	 * @return список міст
	 */
```

```java
/**
 * Повертає список міст, по яких
 * є інформація про погоду
 * @return список міст
 */
public List<String> getCities() {
	System.out.println("GetCities start");
	try {
		List<String> cities = new ArrayList<String>();
		// Отримання з'єднання з БД
		Connection con = getConnection();
		// Виконання SQL-запиту
		ResultSet rs = con.createStatement().executeQuery(
				"Select DISTINCT city From weather");
		while (rs.next()) {
			cities.add(rs.getString(1));
		}
		// Закриваєм вибірку і з'єднання з БД
		rs.close();
		con.close();
		return cities;
	} catch (Exception ex) {
		ex.printStackTrace();
		return null;
	}
}
```

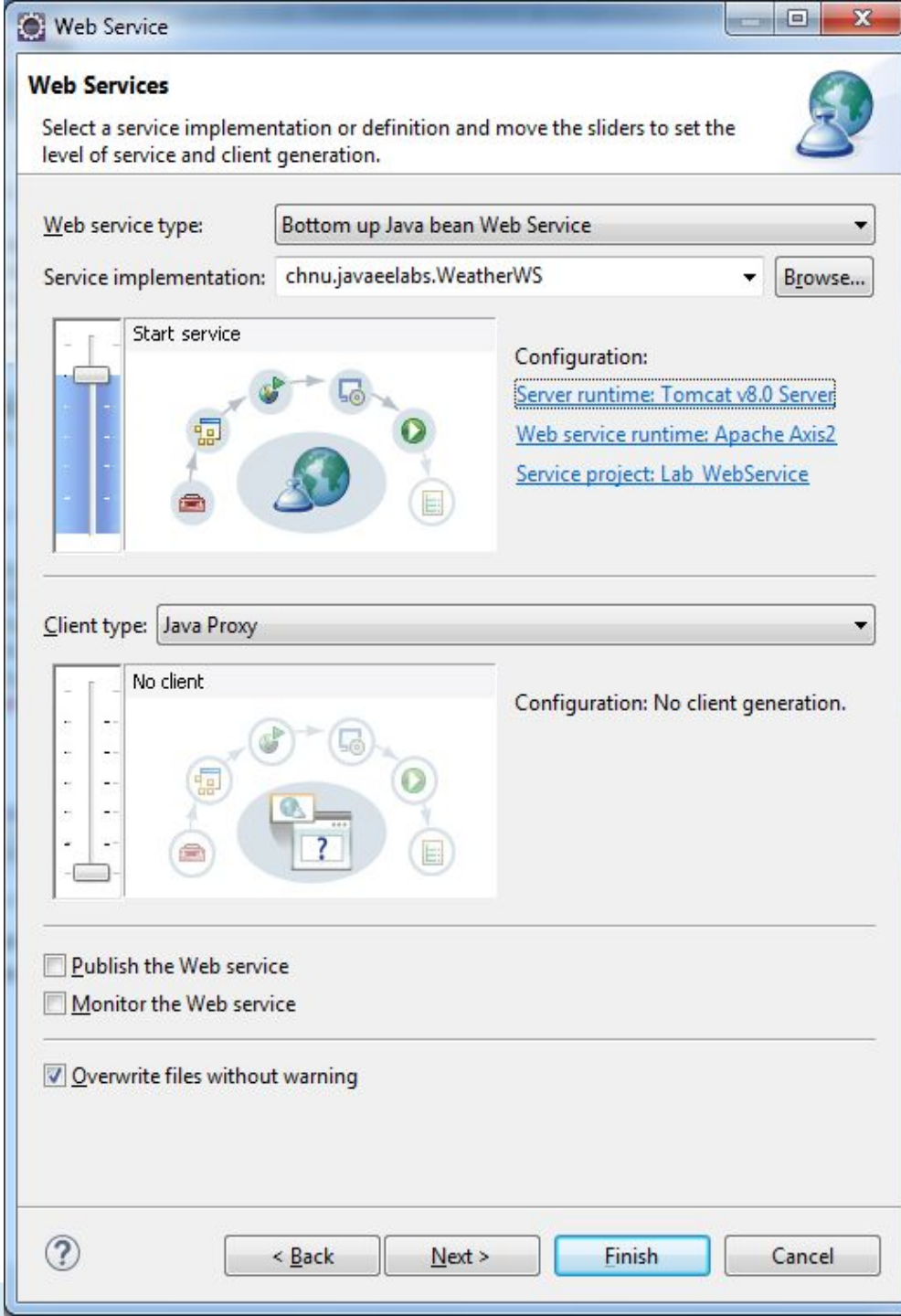
```
    * Повертає інформацію про погоду у вказаному місті
    * @param city назава міста
    * @return інформація про погоду
    */
   public WeatherInfo getWeatherInfo(String city) {
          System.out.println("GetWeatherInfo start");
          try {
                WeatherInfo weatherInfo = null;
                // Отримання з'єднання з БД
                Connection con = getConnection();
                // Виконання SQL-запиту
                ResultSet rs = con.createStatement().executeQuery(
                      "Select city, temperature, " +
                                  "cloudiness, precipitations " +
                                  "from weather " +
                                  "where city like '" + city + "'");
                if (rs.next()) {
                      // Формування нового об'єкту класа WeatherInfo
                      // на основі даних вибраного запису
                      weatherInfo = new WeatherInfo(
                      rs.getString(1),
                      rs.getInt(2),
                      rs.getString(3),
                      rs.getString(4));
                }
                // Закриваєм вибірку і з'єднання з БД
                rs.close();
                con.close();

                return weatherInfo;
          } catch (Exception ex) {
                ex.printStackTrace();
                return null;
          }
    }
}
```

Створення та розгортання Веб-сервісу

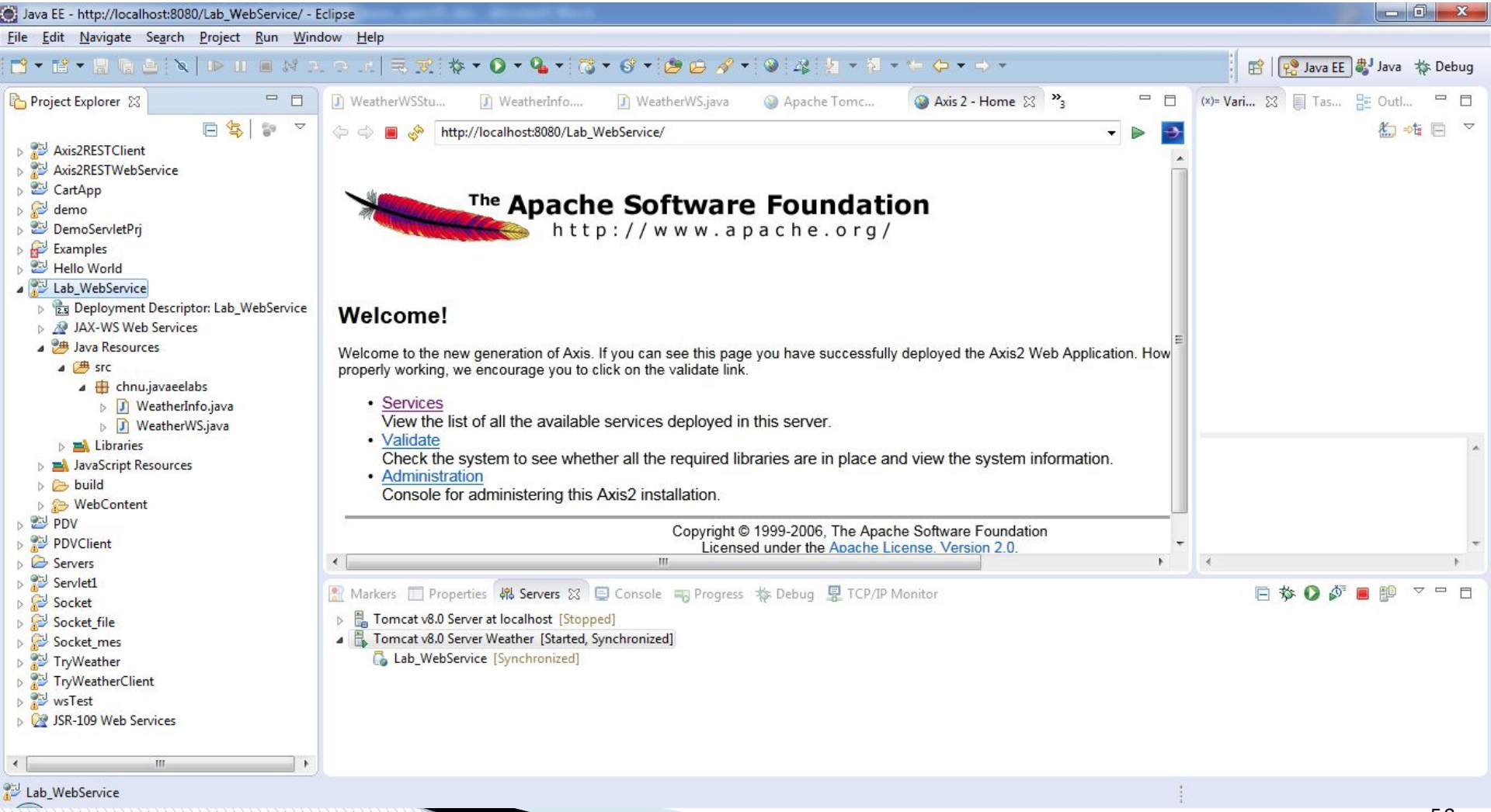
Веб-сервіси, реалізовані у вигляді звичайного java-класу (POJO - plain old java object), упаковуються у веб-додаток і розгортаються у веб-контейнері сервера додатків.

Будемо створювати Веб-сервіс методом, описаним вище. Для цього потрібно:

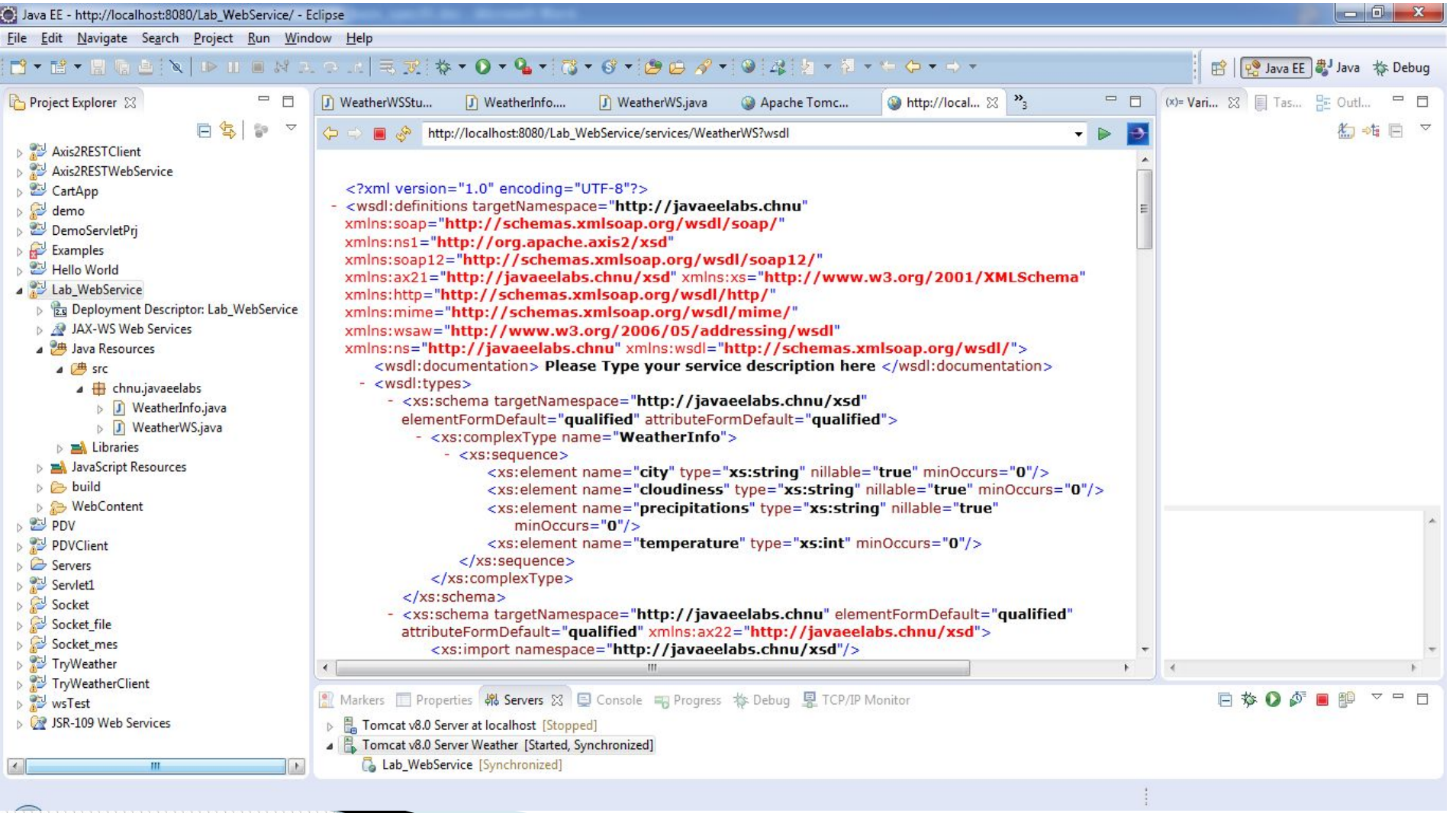
1) Виділити клас ***WeatherWS***, клацнути на ньому правою кнопкою миші, вибрати New / Other…/ Web Services / Web service. У вікні, що з'явилося, налаштувати потрібну конфігурацію (в якості Web service runtime обов'язково вказати Apache Axis2).



Решту налаштувань залиште без змін. Після натиснення кнопки Finish буде запущено веб-сервіс, з'явиться наступна сторінка (в подальшому для запуску Веб-сервісу потрібно буде викликати контекстне меню на проекті Lab_WebService і виконати команду Run As / Run on Server).



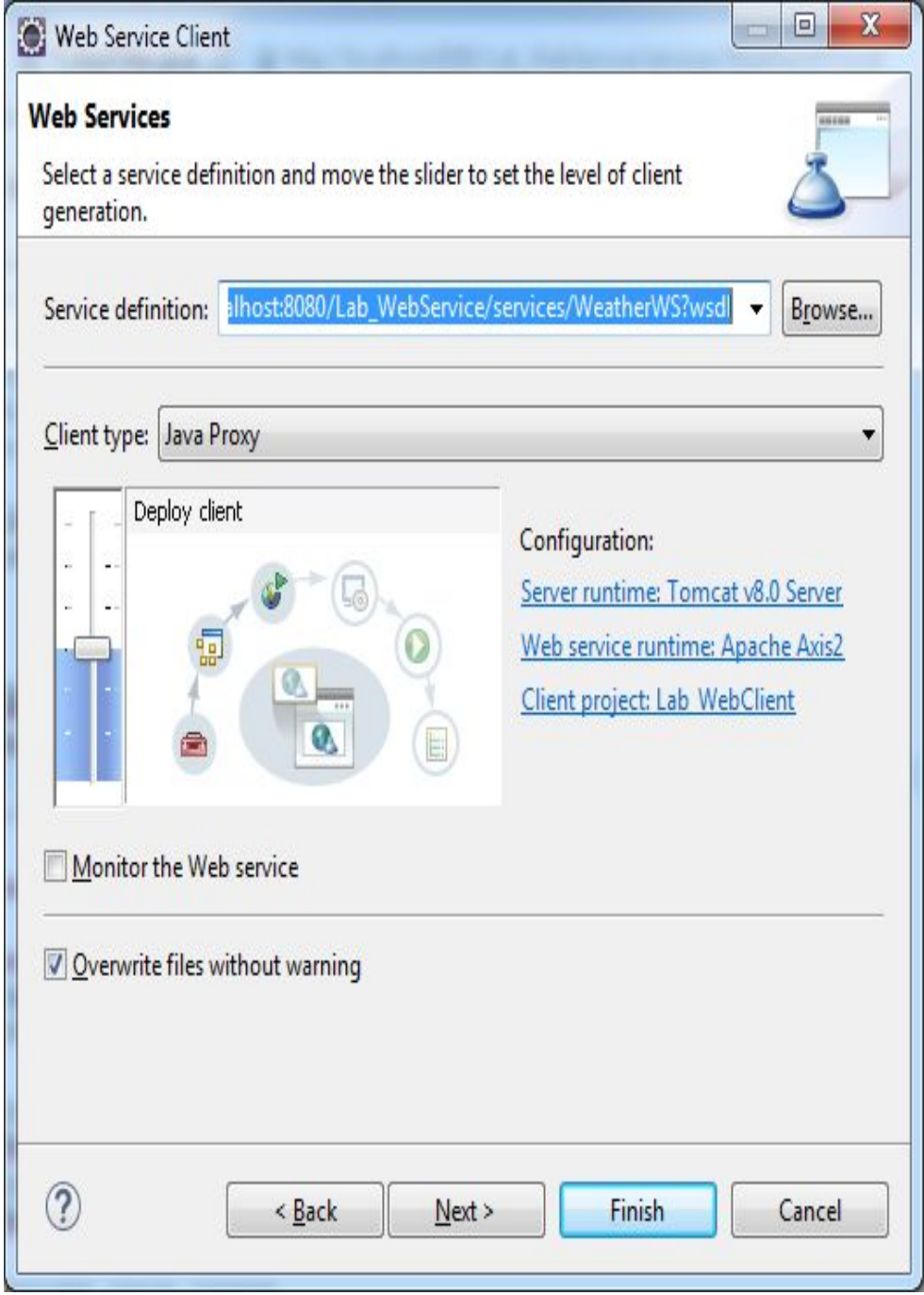
Якщо перейти за посиланням Services / WeatherWS, можна переглянути код автоматично згенерованого wsdl-файлу, що знаходться за адресою: http://localhost:8080/Lab_WebService/services/WeatherWSService?wsdl



Створення нового проекту для веб-клієнта і генерація заглушок (stubs) для доступу до веб-сервісу

1) Створіть новий проект типу Dynamic Web Project. Назвіть проект Lab_WebClient. Класи-заглушки, необхідні для звернення до віддаленого веб-сервісу, можуть бути згенеровані на основі WSDL-файлу веб-сервісу.

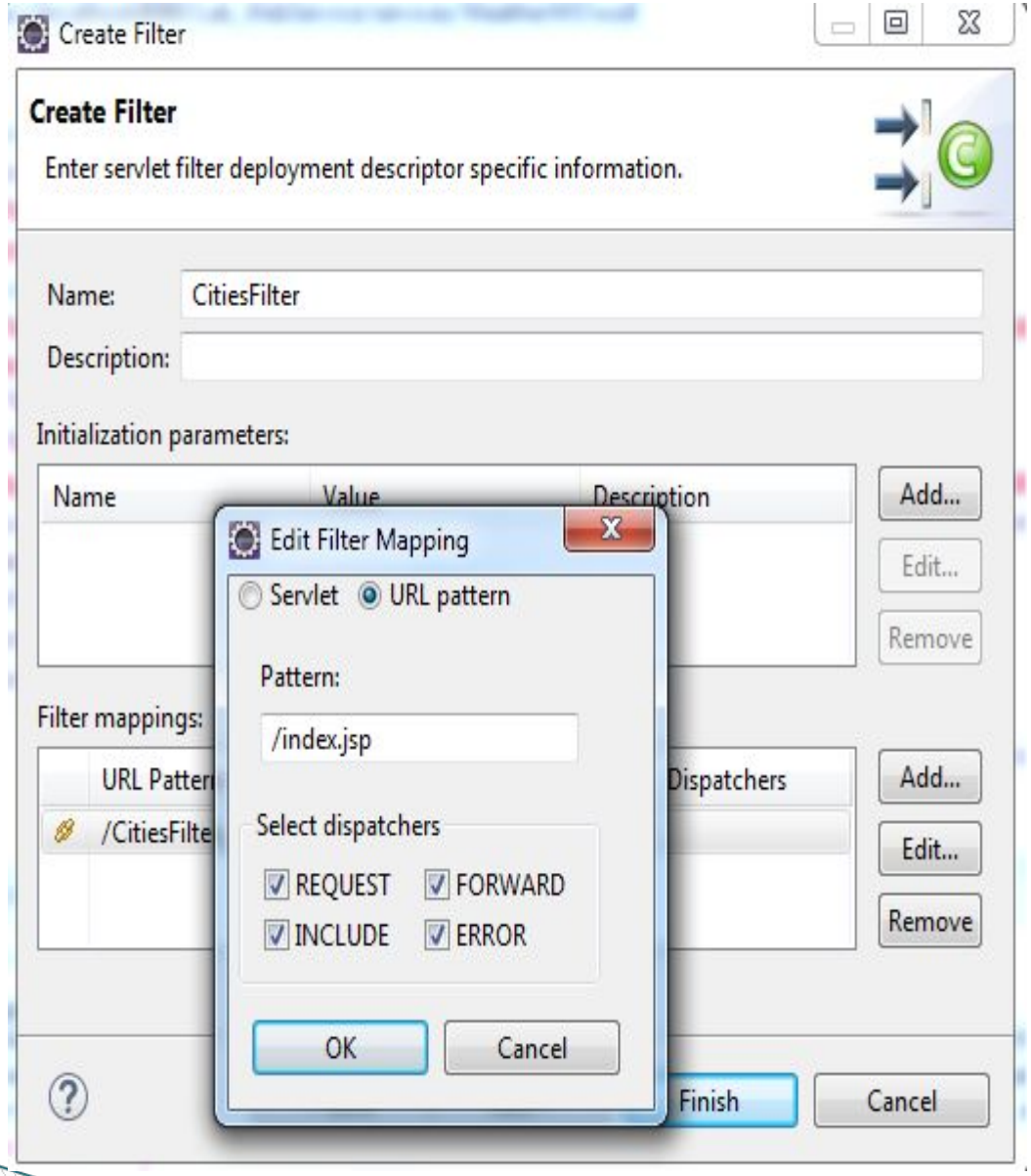
2) В контекстному меню даного проекту виберіть New / Other…/ Web Services / Web Service Client. З'явиться вікно, в якому потрібно здійнити налаштування для клієнта веб-сервіса. В якості Service definition необхідно вказати адресу wsdl-файла сервісу **WeatherWS**



Розробка веб-фільтру CityFilter для передачі jsp-сторінці списку міст

Фільтри використовуються для перехоплення запитів до сторінки і обробки параметрів цього запиту. У нашому випадку в запит до сторінки index.jsp додаватиметься список міст, отриманий за допомогою методу веб-сервісу getCitites().

1) Створіть в пакеті *chnu.javaeelabs.client* новий класс-фільтр *CitiesFilter*. Для цього на відповідному пакеті через контекстне меню викличте команду New / Filter. В якості Class name вкажіть *CitiesFilter*, натисніть Next. Відредагуйте Filter mappings, як показано на Рис.81. Необхідно пов'язати фільтр зі сторінкою index.jsp таким чином, щоб він спрацьовував при виконанні запиту до цієї сторінки. Ці налаштування також автоматично вказуються у файлі web.xml. Для параметра Dispatchers виберіть всі чотири дії (FORWARD, INCLUDE, REQUEST і ERROR), при яких буде спрацьовувати фільтр. При створенні списку Interfaces вкажіть, що клас реалізує інтерфейс javax.servlet.Filter. Натисніть Finish.



У методі **doFilter**() необхідно встановити зв'язок з віддаленим веб-сервісом, за допомогою методу **getCities**() отримати список міст і помістити цей список в параметри об'єкта-запиту (request) під ім'ям cities.

```java
package chnu.labsjavaee.client;
import java.io.IOException;
import java.util.Arrays;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import chnu.javaeelabs.WeatherWSStub;
import chnu.javaeelabs.WeatherWSStub.GetCities;

public class CitiesFilter implements Filter {
	public void destroy() {}
	public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
			throws IOException, ServletException {

		WeatherWSStub stub = new
WeatherWSStub("http://localhost:8080/Lab_WebService/services/WeatherWS?wsdl");
		GetCities gs = new GetCities();
		String[] cities = stub.getCities(gs).get_return();
		for (Object citym : cities) {
			System.out.println(citym);}
		// Поміщаємо список міст в параметри запиту
		request.setAttribute("cities", Arrays.asList(cities));
		// Напаравляємо запит далі по ланцюжку
		chain.doFilter(request, response);
	}
    public void init(FilterConfig arg0) throws ServletException {}
}
```

Створення сервлета WeatherServlet для отримання інформації про погоду

1) Створіть новий сервлет з ім'ям ***WeatherServlet*** в пакеті chnu.javaeelabs.client.

2) Метод doGet() сервлета зчитує параметр city, переданий користувачем, звертається до веб-сервісу і за допомогою методу getWeatherInfo() зчитує дані про погоду в зазначеному місті. Результат - об'єкт WeatherInfo - поміщається в запит у вигляді параметра з ім'ям weatherInfo і запит перенаправляється назад сторінці index.jsp. Програмний код сервлета наведений нижче:

```
package chnu.labsjavaee.client;

import java.io.IOException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import chnu.javaeelabs.WeatherWSStub;
import chnu.javaeelabs.WeatherWSStub.GetCities;
import chnu.javaeelabs.WeatherWSStub.GetWeatherInfo;
import chnu.javaeelabs.WeatherWSStub.GetWeatherInfoResponse;
import chnu.javaeelabs.WeatherWSStub.WeatherInfo;
/**
 * Servlet implementation class WeatherServlet
 */
```

```java
public class WeatherServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
     /**
     * @see HttpServlet#HttpServlet()
     */
    public WeatherServlet() {
        super();
        // TODO Auto-generated constructor stub
    }
    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");

      // Отримуємо із запиту значення параметра city
      String city = request.getParameter("city");
      // Створюємо об'єкт для доступу до веб-сервісу
      WeatherWSStub stub = new
    WeatherWSStub("http://localhost:8080/Lab_WebService/services/WeatherWS?wsdl");

      GetWeatherInfo getWeatherInfo2 = new GetWeatherInfo();
      getWeatherInfo2.setCity(city);
      // Отримуємо погоду в місті
      WeatherInfo weatherInfo = stub.getWeatherInfo(getWeatherInfo2).get_return();
      // Поміщаємо в параметри запиту
      request.setAttribute("weatherInfo", weatherInfo);
      // Перенаправляємо виклик назад сторінці index.jsp
      RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/index.jsp");
      dispatcher.forward(request, response);
    }
}
```

Розробка JSP-сторінки index.jsp

Наш проект буде містити одну сторінку **index.jsp**, яка буде використовуватися для вибору міста в випадаючому списку і для відображення інформації про погоду по цьому місту.

1) Створіть нову JSP-сторінку з ім'ям index.jsp.

2) Реалізація JSP-сторінки включає в себе наступні основні кроки:

- зміна кодування сторінки на UTF-8 для коректного відображення символів кирилиці;
- зміна заголовка сторінки на «Інформація про погоду»;
- створення форми, що включає в себе список (<select>) з ім'ям city для відображення міст, отриманих в параметрі cities. При виборі елементу списку форма передає значення параметра city сервлету WeatherServlet;
- зчитування параметра weatherInfo. Якщо він не порожній, відобразити на сторінці дані про погоду.

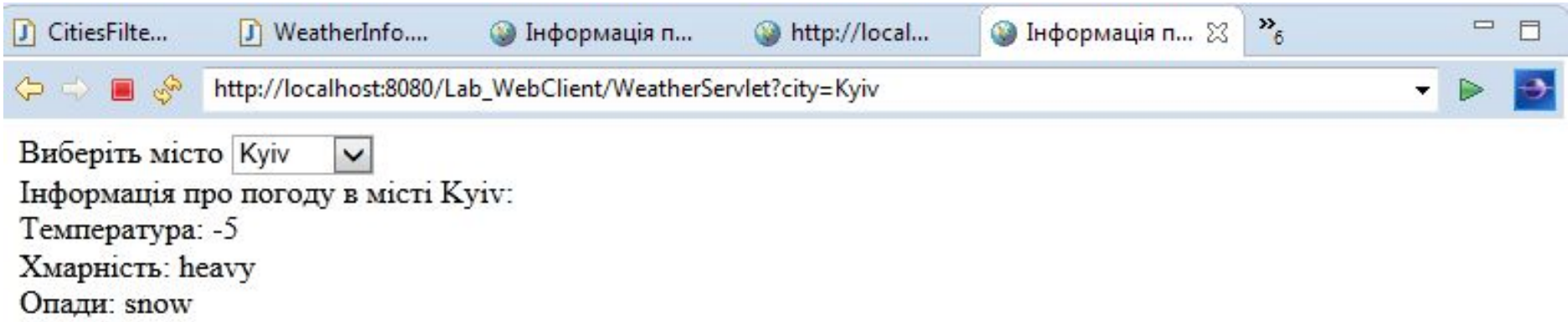
Нижче наведений код JSP-сторінки:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<%@page import="java.util.List"%>
<%@page import="chnu.javaeelabs.WeatherWSStub.GetWeatherInfo"%>
<%@page import="chnu.javaeelabs.WeatherWSStub.GetWeatherInfoResponse"%>
<%@page import="chnu.javaeelabs.WeatherWSStub.WeatherInfo"%>

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Інформація про погоду</title>
</head>
<body>
<form action="WeatherServlet" method = "GET">
Виберіть місто
<select name="city" onChange="submit();">
<!-- порожній елемент списку -->
<option>
<%
// Зчитуємо дані про погоду
WeatherInfo weatherInfo = (WeatherInfo)request.getAttribute("weatherInfo");
//Зчитуємо список міст
List cities  = (List)request.getAttribute("cities");
//Відображення списку міст
if (cities != null) {
      for (int i=0; i<cities.size(); i++) {
            String sel = "";
            // Якщо місто вже вказане, вибираємо його в списку
            if (weatherInfo!=null && cities.get(i).equals(weatherInfo.getCity()))
              sel = " selected = \"selected\"";
              out.print("<option" + sel + ">" + cities.get(i) + "</option>");
      }
}
%>
```

```
%>
</select>
</form>
<%
// Відображення інформації про погоду
if (weatherInfo != null) {
        out.print("Інформація про погоду в місті " + weatherInfo.getCity() + ":");
        out.print("<br>");
        out.print("Температура: " + weatherInfo.getTemperature());
        out.print("<br>");
        out.print("Хмарність: " + weatherInfo.getCloudiness());
         out.print("<br>");
         out.print("Опади: " + weatherInfo.getPrecipitations());
}
%>
</body>
</html>
```

Тестування веб-клієнта

Щоб запустити веб-клієнт і протестувати його, викличте контекстне меню на проекті Lab_WebClient, виберіть пункт Run As / Run on Server. Якщо всі попередні кроки виконані вірно, з'явиться сторінка, на якій можна вибрати місто і перевірити роботу додатку, переглянувши дані про погоду.



Ознайомлення з REST-сервісами

REST (Representational State Transfer, передача репрезентативного стану) - архітектурний стиль побудови розподіленої системи, в основі якого лежить поняття ресурсу та його стану. Ресурс має свій універсальний ідентифікатор (URI, unified resource identifier), використовуючи який над даним ресурсом можна здійснювати різні дії, наприклад **CRUD** (create, read, update, delete).

Далі представлена таблиця основних, використовуваних при реалізації REST-архітектури методів HTTP, з їх коротким цільовим призначенням.

Метод	Опис
OPTIONS	Використовується для запиту про підтримувані методи, адреси, а також додаткову довідкову інформацію
GET	Використовується для отримання стану ресурсу
HEAD	Використовується для отримання метаданих стану ресурсу
PUT	Створює або замінює вміст ресурсу
POST	Додає вміст в існуючий ресурс
DELETE	Використовується для видалення ресурсу

Приклад створення Веб-сервісу із використанням архітектури REST

Платформа Axis2 підтримує HTTP-протокол на основі реалізації зв'язування WSDL HTTP Binding, що дає можливість викликати Веб-сервіс за допомогою HTTP-запитів GET і POST.

Будь-який Веб-сервіс, розгорнутий на платформі Axis2, стає доступним для GET-запиту, що складається з URL-адреси кінцевої точки, імені операції Веб-сервісу і параметрів операції, що викликається, Веб-сервісу а також для POST-запиту, який може містити XML-дані.

Для включення підтримки REST на стороні клієнта для Options-об'єкта необхідно встановити відповідну властивість:

Для включення підтримки REST на стороні клієнта для Options-об'єкта необхідно встановити відповідну властивість:

```
Options options = new Options();
options.setProperty(Constants.Configuration.ENABLE_REST,
Constants.VALUE_TRUE);
```

Для RESTful Веб-серверів і клієнтів RESTful Веб-сервісів платформа Axis2 забезпечує формат повідомлень JSON. Для підключення підтримки JSON-формату необхідно в конфігураційний файл axis2.xml включити наступні елементи:

- формат повідомлень application / json:

```
<messageFormatters>
     <messageFormatter contentType="application/json"
          class="org.apache.axis2.json.JSONMessageFormatter"/>
     . . .
</messageFormatters>
<messageBuilders>
     <messageBuilder contentType="application/json"
          class="org.apache.axis2.json.JSONOMBuilder"/>
     . . .
</messageBuilders>
```

- формат повідомлень text/javascript:

```
<messageFormatters>
     <messageFormatter contentType="text/javascript"
          class="org.apache.axis2.json.JSONBadgerfishMessageFormatter"/>
     . . .
</messageFormatters>
<messageBuilders>
     <messageBuilder contentType="text/javascript"
          class="org.apache.axis2.json.JSONBadgerfishOMBuilder"/>
     . . .
</messageBuilders>
```

Також потрібно додати до шляху додатку бібліотеки jettison.jar платформи Axis2, що забезпечує серіалізацію/десеріалізацію JSON-повідомлень.

Розглянемо приклад RESTful Веб-сервісу і клієнта RESTful Веб-сервісу платформи Axis2.

1. Для створення RESTful Веб-сервісу відкриємо середовище Eclipse, в меню File виберемо пункти New / Dynamic Web Project і введемо ім'я проекту **Axis2REST-WebService.** Натиснувши кнопку New Runtime, виберемо Apache Tomcat v8.0, у списку Dynamic web module version виберемо значення 2.5 і натиснемо кнопку Finish.

2. В меню Window середовища Eclipse виберемо пункти Preferences / Web Services / Axis2 Preferences. На вкладці Axis2 Runtime, натиснувши кнопку Browse, визначимо каталог попередньо інстальованого середовища виконання Apache Axis2 (Binary Distribution, http. На вкладці Axis2 Runtime, натиснувши кнопку Browse, визначимо каталог попередньо інстальованого середовища виконання Apache Axis2 (Binary Distribution, http://. На вкладці Axis2 Runtime, натиснувши кнопку Browse, визначимо каталог попередньо інстальованого середовища виконання Apache Axis2 (Binary Distribution, http://axis. На вкладці Axis2 Runtime, натиснувши кнопку Browse, визначимо каталог попередньо інстальованого середовища виконання Apache Axis2 (Binary Distribution, http://axis.. На вкладці Axis2 Runtime, натиснувши кнопку Browse, визначимо каталог попередньо інстальованого середовища виконання Apache Axis2 (Binary Distribution, http://axis.apache. На вкладці Axis2 Runtime, натиснувши кнопку Browse, визначимо каталог попередньо інстальованого середовища виконання Apache Axis2 (Binary Distribution, http://axis.apache.. На вкладці Axis2 Runtime,

5. У вікні Project Explorer клацнемо правою кнопкою миші на вузлі **Axis2RESTWebService** і виберемо пункти New / Class, введемо ім'я Веб-сервісу **RESTService** й ім'я пакету **restservice** і натиснемо кнопку Finish.
6. Доповнимо код класу **RESTService**, як показано нижче.
7. Для розгортання Web-сервісу RESTService на платформі Axis2 сервера Tomcat у вікні Project Explorer клацнемо правою кнопкою миші на вузлі Axis2RESTWebService і виберемо пункти New / Other / Web Services / Web Service. Натиснемо кнопку Next, у рядку Service implementation введемо **restservice.RESTService** і натиснемо кнопку Finish.
У результаті в папку WebContent\WEB-INF\services проекту Axis2RESTWebService буде додана папка RESTService з клас-файлом Веб-сервісу і його конфігураційним файлом services.xml.
Код класу **RESTService**:

```
package restservice;
import org.apache.axiom.om.*;

public class RESTService {
	public OMElement getHello(OMElement root) throws Exception {
		org.apache.axis2.context.MessageContext mcIn =
	org.apache.axis2.context.MessageContext.getCurrentMessageContext().getOperationContext().get
MessageContext(org.apache.axis2.wsdl.WSDLConstants.MESSAGE_LABEL_IN_VALUE);
		Object object =
mcIn.getProperty(org.apache.axis2.Constants.Configuration.MESSAGE_TYPE);
	String messageType = (String) object;
	System.out.println(messageType);
	System.out.println(mcIn.getEnvelope().toString());
	OMElement element = root.getFirstElement();
	String name=element.getText();
	System.out.println("Name"+" "+name);
	if (!messageType.equals("application/json")) {
		OMFactory factory = OMAbstractFactory.getOMFactory();
		OMNamespace ns1 = factory.createOMNamespace("http://restservice", "ns1");
		OMNamespace s1 = factory.createOMNamespace("http://www.w3.org/2001/XMLSchema-
instance", "s1");
		OMNamespace s2 = factory.createOMNamespace("http://www.w3.org/2001/XMLSchema", "s2");
```

```
        OMElement rootresponse = factory.createOMElement("getHelloResponse", ns1);
        OMElement returnresponse = factory.createOMElement("return", ns1);
        returnresponse.declareNamespace(s1);
        returnresponse.declareNamespace(s2);
        OMAttribute type = factory.createOMAttribute("type",s1,"s2:string");
        returnresponse.addAttribute(type);
        returnresponse.setText("Hello"+" "+name);
        rootresponse.addChild(returnresponse);
        System.out.println(rootresponse.toString());
        return rootresponse;}
    if (messageType.equals("application/json")) {
        OMFactory factory = OMAbstractFactory.getOMFactory();
        OMNamespace ns = factory.createOMNamespace("", "");
        OMElement resJson = factory.createOMElement("getHelloResponse", ns);
        OMElement returnJson = factory.createOMElement("return", ns);
        returnJson.setText("Hello "+name);
        resJson.addChild(returnJson);
        System.out.println(resJson.toString());
        return resJson; }
    return null;
}}
```

Так як конфігураційний файл **axis2.xml** папки WebContent \ WEB-INF \ conf проекту **Axis2RESTWebService** визначає Handler-обробник RawXMLINOutMessageReceiver на глобальному рівні, файл **services.xml** прийме наступний вигляд:

```
<service name="RESTService" >
   <parameter name="ServiceClass" locked="false">
    restservice.RESTService</parameter>
</service>
```

Для включення підтримки JSON-формату додамо в папку WebContent \ WEB-INF \ lib проекту Axis2RESTWebService бібліотечний файл jettison.jar платформи Axis2 і змінимо файл **axis2.xml:**

```
<messageFormatters>
     <messageFormatter contentType="application/x-www-form-urlencoded"
       class="org.apache.axis2.transport.http.XFormURLEncodedFormatter"/>
     <messageFormatter contentType="multipart/form-data"
       class="org.apache.axis2.transport.http.MultipartFormDataFormatter"/>
     <messageFormatter contentType="application/xml"
       class="org.apache.axis2.transport.http.ApplicationXMLFormatter"/>
     <messageFormatter contentType="text/xml"
       class="org.apache.axis2.transport.http.SOAPMessageFormatter"/>
     <messageFormatter contentType="application/soap+xml"
       class="org.apache.axis2.transport.http.SOAPMessageFormatter"/>
     <messageFormatter contentType="application/json"
       class="org.apache.axis2.json.JSONMessageFormatter"/>
</messageFormatters>
<messageBuilders>
     <messageBuilder contentType="application/xml"
       class="org.apache.axis2.builder.ApplicationXMLBuilder"/>
     <messageBuilder contentType="application/x-www-form-urlencoded"
       class="org.apache.axis2.builder.XFormURLEncodedBuilder"/>
     <messageBuilder contentType="multipart/form-data"
       class="org.apache.axis2.builder.MultipartFormDataBuilder"/>
     <messageBuilder contentType="application/json"
       class="org.apache.axis2.json.JSONOMBuilder"/>
</messageBuilders>
```

Наберемо в рядку браузера адресу **http://localhost:8080/Axis2RESTWebService/services/RESTService/getHello? Name = User** і у відповідь побачимо повідомлення Веб-сервісу:

```
<ns1:getHelloResponse xmlns:ns1="http://restservice">
<ns1:return xmlns:s2="http://www.w3.org/2001/XMLSchema"
  xmlns:s1="http://www.w3.org/2001/XMLSchema-instance"
  s1:type="s2:string">Hello User</ns1:return>
</ns1:getHelloResponse>
```

GET-запит буде конвертований середовищем виконання Axis2 в SOAP-повідомлення:

```
<?xml version='1.0' encoding='utf-8'?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
    <soapenv:Body>
        <axis2ns4:getHello xmlns:axis2ns4="http://restservice">
            <name>User</name>
        </axis2ns4:getHello>
    </soapenv:Body>
</soapenv:Envelope>
```

Повідомлення-відповідь не є SOAP-повідомленням, а буде являти собою чисті XML-дані

1. Для створення клієнта Веб-сервісу в перспективі Java середовища Eclipse в меню File виберемо пункти New / Java Project, введемо ім'я проекту **Axis2RESTClient** і натиснемо кнопку Finish.
2. У вікні Project Explorer клацнемо правою кнопкою миші на вузлі Axis2RESTClient і виберемо пункт Properties. У діалоговому вікні виберемо Project Facets, вкажемо Dynamic Web Module 2.5, Axis2 Web Services і натиснемо кнопку OK.
3. У вікні Project Explorer клацнемо правою кнопкою миші на вузлі Axis2RESTClient і виберемо пункти New / Class, введемо ім'я класу **RESTClient** і ім'я пакету restclient, вкажемо опцію public static void main (String [] args) і натиснемо кнопку Finish.
4. Змінимо код класу **RESTClient**, як показано нижче:

```
package restclient;
import org.apache.axis2.client.*;
import org.apache.axiom.om.*;

public class RESTClient {
	public static void main(String[] args) throws Exception {
		java.io.File configFile = new java.io.File(
				"WebContent/WEB-INF/conf/axis2.xml");
		System.out.println(configFile.exists());
		org.apache.axis2.context.ConfigurationContext configurationContext =
org.apache.axis2.context.ConfigurationContextFactory
				.createConfigurationContextFromFileSystem(

	"D:\\E_work_main\\main\\java\\java_project\\Axis2RESTClient\\WebContent\\WEB-INF\\",
					configFile.getAbsolutePath());
		ServiceClient sc = new ServiceClient(configurationContext, null);
```

```
OMFactory factory = OMAbstractFactory.getOMFactory();
OMNamespace ns1 = factory.createOMNamespace("http://restservice", "ns1");
OMElement root = factory.createOMElement("getHello", ns1);
OMElement name = factory.createOMElement("name", ns1);
name.setText("User");
root.addChild(name);
Options op = new Options();
op.setTo(new org.apache.axis2.addressing.EndpointReference(
            "http://127.0.0.1:8080/Axis2RESTWebService/services/RESTService"));
op.setProperty(org.apache.axis2.Constants.Configuration.ENABLE_REST,
            Boolean.TRUE);
// Soap
op.setProperty(org.apache.axis2.Constants.Configuration.MESSAGE_TYPE,
            "text/xml");
op.setAction("urn:getHello");
sc.setOptions(op);
OMElement resSoap = sc.sendReceive(root);
```

```
        System.out.println("SOAP " + resSoap.toString());
        // XML
        op.setProperty(org.apache.axis2.Constants.Configuration.MESSAGE_TYPE,
                       "application/xml");
        op.setAction("");
        sc.setOptions(op);
        OMElement resXml = sc.sendReceive(root);
        System.out.println("XML" + resXml.toString());
        // JSON
        op.setProperty(org.apache.axis2.Constants.Configuration.MESSAGE_TYPE,
                       "application/json");
        sc.setOptions(op);
        OMNamespace ns = factory.createOMNamespace("", "");
        OMElement rootJson = factory.createOMElement("getHello", ns);
        OMElement nameJson = factory.createOMElement("name", ns);
        nameJson.setText("User");
        rootJson.addChild(nameJson);
        OMElement resJson = sc.sendReceive(rootJson);
        System.out.println("JSON " + resJson.toString());
    }
}
```

Для включення підтримки JSON-формату конфігураційний файл **axis2.xml** папки WebContent\WEB-INF\conf проекту **Axis2RESTClient** змінений аналогічно файлу axis2.xml проекту Axis2RESTWebService і в шлях додатку Axis2RESTClient додана бібліотека jettison.jar платформи Axis2 за допомогою вибору опцій Properties / Java Build Path / Libraries / Add External JARs.

У методі main() класу RESTClient підтримка REST включається за допомогою визначення властивості Options-об'єкта:

op.setProperty(org.apache.axis2.Constants.Configuration.ENABLE_REST, Boolean.TRUE);

Інтерфейс OMAPI використовується для створення двох типів вихідних повідомлень. Перший тип вихідних повідомлень містить XML-дані з оголошенням простору імен і призначений для значень HTTP-заголовка Content-Type, рівних text/xml і application/xml. XML-дані другого типу вихідних повідомлень не містять оголошення простору імен і призначені для JSON-формату. Тип вихідних повідомлень встановлюється визначенням відповідної властивості Options-об'єкта.

Після запуску класу **RESTClient** як Java-додатку в консолі середовища Eclipse можна побачити вхідні повідомлення клієнта Web-сервісу, а у вікні TCP-монітора вхідні і вихідні повідомлення Web-сервісу. Видно, що вхідні і вихідні повідомлення Веб-сервісу типу text/xml відповідають SOAP-формату, повідомлення типу application/xml представляють собою чисті XML-дані, а повідомлення типу application/json мають JSON-формат.

SOAP чи REST сервіси?

1. SOAP активно використовує XML для кодування запитів і відповідей, а також строгу типізацію даних, що гарантує їх цілісність при передачі між клієнтом і сервером. З іншого боку, запити і відповіді в REST можуть передаватися в ASCII, XML, JSON або будь-яких інших форматах, які розпізнаються одночасно і клієнтом, і сервером. Крім того, в моделі REST відсутні вбудовані вимоги до типізації даних. У результаті пакети запитів і відповідей у REST мають набагато менші розміри, ніж відповідні їм пакети SOAP.
2. У моделі SOAP рівень передачі даних протоколу HTTP є «пасивним спостерігачем», і його роль обмежується передачею запитів SOAP від клієнта серверу з використанням методу POST. Деталі сервісного запиту, такі як ім'я віддаленої процедури і вхідні аргументи, кодуються в тілі запиту. Архітектура REST, навпаки, розглядає рівень передачі даних HTTP як активного учасника взаємодії, використовуючи існуючі методи HTTP, такі як GET, POST, PUT і DELETE, для позначення типу запитуваного сервісу.

Отже, з точки зору розробника, запити REST в загальному випадку більш прості для формулювання і розуміння, так як вони використовують існуючі і добре зрозумілі інтерфейси HTTP.

3. Модель SOAP підтримує певний степінь інтроспекції, дозволяючи розробникам сервісу описувати його API у файлі формату WSDL. Використовуючи їх, клієнти SOAP можуть автоматично отримувати докладну інформацію про імена і сигнатури методів, типи вхідних та вихідних даних і значення, що повертаються. Модель REST не має WSDL, проте багато в чому її інтерфейс може бути більш зрозумілий і навіть інтуїтивний, оскільки базується на стандартних методах HTTP.
4. В основі REST лежить концепція ресурсів, у той час як SOAP використовує інтерфейси, засновані на об'єктах і методах. Інтерфейс SOAP може містити практично необмежену кількість методів; інтерфейс REST, навпаки, обмежений чотирма можливими операціями, що відповідають чотирьом методам HTTP.

SOAP-запит:

```
<?xml version="1.0"  encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
                   xmlns:ns1="http://rpc.geocoder.us/Geo/Coder/US/"
                   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                   xmlns:xsl="http://www.w3.org/2001/XMLSchema-instance"
                   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
                   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <ns1:geocode>
      <location xsl:type="xsd:string">1600 Pennsylvania Av, Washington, DC</location>
    </ns1:geocode>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP-відповідь:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
   xmlns:xsl="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <geocodeREspose xmlns="http://rpc.geocoder.us/Geo/Coder/US/">
        <geo:results soapenc:arrayType="geo:GeocoderAddressResult[1]"
                xsl:type="soapenc:Array"
                xmlns:geo="http://rpc.geocoder.us/Geo/Coder/US/">
           <geo:item xsl:type="geo:GeocoderAddressResult"
xmlns:geo="http://rpc.geocoder.us/Geo/Coder/US/">
              <geo:number xsl:type="xsd:int">1600</geo:number>
              <geo:lat xsl:type="xsd:float">38.898748</geo:lat>
              <geo:street xsl:type="xsd:string">Pensylvania</geo:street>
              <geo:state xsl:type="xsd:string">DC</geo:state>
              <geo:city xsl:type="xsd:string">Washington</geo:city>
              <geo:zip xsl:type="xsd:int">20502</geo:zip>
              <geo:suffix xsl:type="xsd:string">NW</geo:suffix>
              <geo:long xsl:type="xsd:float">-77.037684</geo:long>
              <geo:type xsl:type="xsd:string">Ave</geo:type>
              <geo:prefix xsl:type="xsd:string">
           </geo:item>
        </geo:results>
      </geocodeResponse>
   </soap:Body>
</soap:Envelope>
```

REST-запит:

GET http://geocoder.us/service/rest/geocode?address=1600+Pennsylvania+Ave,+Washington+DC

REST-відповідь:

```
<?xml version="1.0"?>
<rdf:RDF
   xmls:dc="http://purl.org/dc/elements/1.1/"
   xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   <geo:Point rdf:nodeID="aid47091944">
      <dc:description>1600 Pensylvaia Ave NW, Washington DC 20502</dc:description>
      <geo:long>-77.037684</geo:long>
      <geo:lat>38.898748</geo:lat>
   </geo:point>
</rdf:RDF>
```

Наприклад, якщо мова йде про розробку системи корпоративного рівня, особливо з можливими потребами інтеграції з іншими системами, то, швидше за все, слід зробити вибір SOAP-сервісів. По-перше, це буде виправдано, тому що SOAP-сервіси мають потужний потенціал до інтеграції. По-друге, як правило, кількість платформ, з використанням яких побудована корпоративна система, невелика, і, отже, немає вимоги підтримати можливість взаємодії з безліччю інших платформ.

Протилежна ситуація спостерігається у випадку, коли необхідно розробити сервіс, орієнтований на велику кількість користувачів, і в даному випадку вигідніше використовувати REST. По-перше, це пов'язано з широкою підтримкою протоколу HTTP сучасними платформами і мовами програмування. Звичайно, протокол SOAP теж підтримується, але часто зустрічаються випадки, коли проста розробка клієнта на платформі, відмінній від тієї, на якій розроблений сервіс, просто неможлива. Наприклад, не вдається по WSDL згенерувати артефакти, або відбуваються помилки під час маршаллінг-анмаршаллінга об'єктів. По-друге, з точки зору продуктивності вигідніший REST. Можливо, саме тому великі соціальні мережі, такі як Facebook, Twitter пропонують саме REST API, а не SOAP API.

Завдання до лабораторних робіт

Частина 1

Згідно з індивідуальним варіантом завдання, отриманим від викладача, потрібно:

• визначити інтерфейс, що реалізується SOAP-сервісом;
• реалізувати сервіс мовою програмування Java;
• реалізувати клієнтський додаток (для тестування сервісу).

1. Необхідно розробити веб-сервіс, що надає інформацію про курси валют у різних банках (наприклад, співвідношення UAH, USD та EUR). Інформація про курси валют зберігається в БД. Користувач за допомогою веб-інтерфейсу вибирає два види валюти й отримує інформацію про співвідношення курсу по всіх банках.
2. Необхідно розробити веб-сервіс для банку XYZ, що надає можливість конвертації зазначеної суми з одного виду валюти в інший (наприклад, UAH <-> USD <-> EUR). Інформація про курси валют зберігається в БД. Користувач за допомогою веб-інтерфейсу вказує суму, вихідну і результуючу валюту й отримує інформацію про суму, отриману в результаті конвертації з вихідної валюти в результуючу.
3. Напишіть серверний клас, що містить методи, які можуть здійснювати сортування заданих чисел. Напишіть клієнтську програму, яка може здійснювати віддалений виклик одного з методів, відправляючи при цьому множину невідсортованих
значень. Користувач при цьому вказує, яким способом потрібно відсортувати цю множину (швидке сортування, «бульбашка» і т.п.). Відобразіть на клієнті результати сортування.
4. Розробіть веб-сервіс для відділу кадрів підприємства, що надає можливість знайти працівників, стаж роботи яких перевищує *n* років, а також вивести дані про пенсіонерів цього підприємства. Інформація про працівників зберігається в БД. Користувач за допомогою веб-інтерфейсу вказує значення *n*, а також стать пенсіонерів, інформацію про яких він хоче отримати.
5. Розробіть веб-сервіс для перегляду дерева каталогів на віддаленому комп'ютері. Користувач за допомогою веб-інтерфейсу має можливість вказати диск, вмістиме якого його цікавить.

6. Розробіть веб-сервіс, який реалізує відгадування «задуманого» віддаленим комп'ютером цілого числа. Користувач за допомогою веб-інтерфейсу вказує проміжок, в якому буде знаходитись задумане число.
7. Розробіть веб-сервіс для деканату, що надає можливість знайти тих студентів, які склали останню сесію на «відмінно», і тих, які будуть рекомендовані на відрахування. Інформація про студентів знаходиться в БД. Користувач за допомогою веб-інтерфейсу має можливість задати курс (перший, другий, …, п'ятий), студенти якого його цікавлять.
8. Розробіть веб-сервіс для бібліотеки, що надає можливість знайти потрібну книгу. Інформація про книги знаходиться в БД. Користувач за допомогою веб-інтерфейсу вказує назву потрібної книги. На клієнті повинні відобразитися всі видання даної книги і окремо – найновіше з них. Якщо якогось немає в наявності, вказати через скільки днів воно буде доступне (в БД має бути поле «дата повернення»).
9. Розробіть веб-сервіс, що надає можливість надрукувати прізвища студентів в порядку спадання середнього бала за навчання. Інформація про студентів знаходиться в БД. Користувач за допомогою веб-інтерфейсу має можливість вказати номер групи.
10. Розробіть веб-сервіс, що надає можливість надрукувати в алфавітному порядку прізвища студентів, що навчаються на 4,5 та їх процент. Інформація про студентів знаходиться в БД. Користувач за допомогою веб-інтерфейсу має можливість вказати номер групи.
11. Розробіть веб-сервіс, що надає можливість надрукувати в алфавітному порядку прізвища власників автомобілів, які мешкають у вказаному місті. Інформація про автомобілі знаходиться в БД. Користувач за допомогою веб-інтерфейсу має можливість вказати назву міста.
12. Розробіть веб-сервіс, що надає можливість надрукувати пару точок з максимальною відстанню між ними. Інформація про точки у тривимірному просторі задана в БД. Для опису точки використати запис: M=(A – ім'я, X,Y,Z – координати, m – маса). Користувач за допомогою веб-інтерфейсу має можливість вказати, яку кількість перших точок із БД потрібно розглянути.
13. Розробіть веб-сервіс для бібліотеки, що надає можливість знайти потрібні книги. Інформація про книги знаходиться в БД. Користувач за допомогою веб-інтерфейсу вказує, зі скількох слів повинна складатися назва книги.
14. Розробіть веб-сервіс для бібліотеки, що надає можливість знайти потрібні книги і надрукувати їх за хронологічним порядком років видання. Інформація про книги знаходиться в БД. Користувач за допомогою веб-інтерфейсу вказує автора.
15. Розробіть веб-сервіс для бібліотеки, що надає можливість знайти потрібні книги і надрукувати їх в алфавітному порядку прізвищ авторів. Інформація про книги знаходиться в БД. Користувач за допомогою веб-інтерфейсу вказує рік видання.

Частина 2

Скориставшись даними будь-якого сайту-репозиторію Веб-сервісів (наприклад, service-repository.com або free-web-services.com) згенерувати клієнт, використавши посилання на WSDL-документ, який описує певний сервіс.

Створити власний клас, щоб продемонструвати, як отримати доступ до потрібних даних.

Вибір Веб-сервісу та методи, які будуть реалізовані у власному класі, узгодити з викладачем.