Module 2
REST Web Service

Outline

- What is web service
- Types of Web Services
- What is REST?
- REST Architecture/Framework
- REST Principles
- Building REST Application
  - JAX-RS (Java API for Restful web service)

Web Services

- **Definition:**
  - web service is a software system designed to support **interoperable machine-to-machine interaction** over a network
  - the web services offer a regular way for **interoperation between software applications** that are running on various types of platforms as well as frameworks. (W3C)
  - web service is **a standardized method for propagating messages between client and server applications**.
- Web services perform specific tasks
  - Usually, web services are searched for over the network as well as call upon accordingly.
  - As a web service is called, it would be capable of providing operation for the client that has invoked the web service.

- **Types of Web Services**
  - **SOAP** - Simple Object Access Protocol
  - **REST** – REpresentation State Transfer

REST

- **REST** is the abbreviation of Representational State Transfer, a phrase coined in the year 2000 by Mr. Roy Fielding.
- **REST** is an architectural style that defines a set of constraints to be used for creating web services.
- **REST API** is a way of accessing web services in a simple and flexible way without having any processing.
- REST web services **do not impose any rules** concerning how it needs to be applied in practice at a low level
- it only holds the **high-level design** guiding principles and leaves it to the developer to think about the implementation.
- Contrasting SOAP, which aims at actions, **REST deals majorly on the resources.**
- A web service that communicates / exchanges information between application using REST architecture / principle is called **RESTful web service.**

Principles of REST architecture

- Uniform Interface
- Stateless
- Cacheable
- Layered system

Principles of REST architecture (1) Uniform Interface

- Resource : everything is a resource
- URI : any resource can be accessed by a URI
- HTTP methods: makes explicit use of HTTP methods

Principles of REST architecture (1) Uniform Interface

- A resource can be defined as a vital element to be referenced within a client-server system.
- REST architecture treats all of its content as a resource, which includes Html Pages, Images, Text Files, Videos, etc.
- Access to resources is provided by the REST server where REST client is used for accessing as well as modification of resources.
- All of its resources get identified via URI

REST Response types

- Four types:
  - XML
  - JSON
  - HTML
  - Plain text

- JSON

- JSON stands for **J**ava**S**cript **O**bject **N**otation
- JSON is a lightweight format for storing and transporting data
- JSON is often used when data is sent from a server to a web page
- JSON is "self-describing" and easy to understand

```
{
"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]
}
```

XML format that shows the user's profile information and JSON format

```xml
<root>
   <Data>
      <UserGUID>4f1764fe-b3d3-ce7c-0f08-0ef09d834b7e</UserGUID>
      <FullName>Alex</FullName>
      <ProfilePic>http://localhost/9c0977f4-877a-8138-4fbb-
a524edf20437.jpg</ProfilePic>
   </Data>
   <Message>Success</Message>
   <ResponseCode>200</ResponseCode>
</root>
```

```json
{
  "ResponseCode": 200,
  "Message": "Success",
  "Data": {
    "UserGUID": "4f1764fe-b3d3-ce7c-0f08-0ef09d834b7e",
    "FullName": "Alex",
    "ProfilePic": "http:\/\/localhost\/9c0977f4-877a-8138-4fbb-
a524edf20437.jpg"
  }
}
```

Principles of REST architecture (1) REST methods

- RESTful web service makes use of HTTP for determining the action to be conceded out on the particular resources

Method	Description
GET	This method helps in offering read-only access for the resources.
POST	This method is implemented for creating a new resource.
DELETE	This method is implemented for removing a resource.
PUT	This method is implemented for updating an existing resource or creating a fresh one.

HTTP Method	URI	Operation
GET	http://localhost/appName/students/	For fetching information for all students.
GET	http://localhost/appName/students/10	For fetching the student having roll number 10.
DELETE	http://localhost/appName/student/10	For removing the student record having ID number 10
DELETE	http://localhost/appName/courses/4	To remove the course having course-id 4.
PUT	http://localhost/appName/student/10	For update the student record having ID number 10
PUT	http://localhost/appName/courses/4	For upgrading the course having course-id 4.
POST	http://localhost/appName/student	To create a new entry.

RESTful Web Application

- Producing/creating Restful web service
  - Demo in Netbeans (four response types)
- Consuming Restful web service
  - Java-based or Python-based

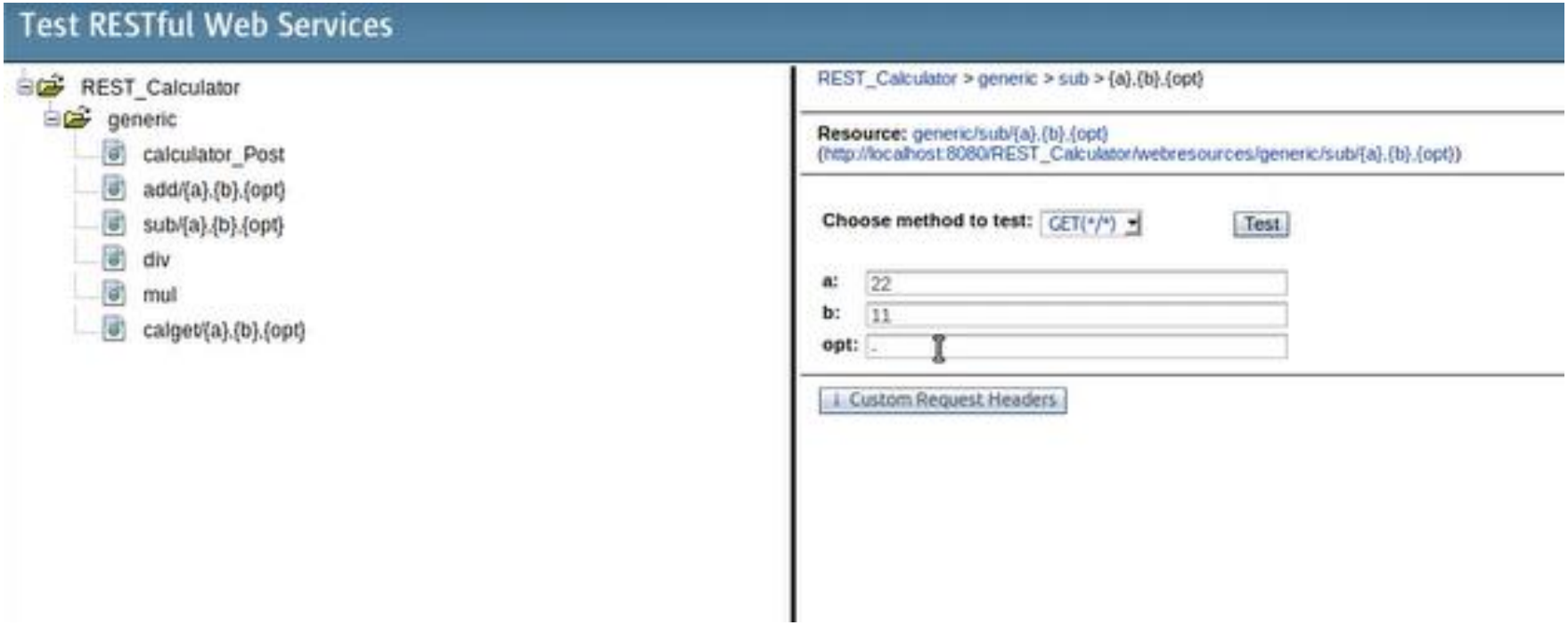
RESTful Web Application

```
@GET
@Path("/add/{a},{b},{opt}")
public int add(@PathParam("a") int a,@PathParam("b") int b,@PathParam("opt") String opt)
{
    if(opt.equals("+"))
        return a+b;
    else
            return 0;
}

@GET
@Path("/sub/{a},{b},{opt}")
public int sub(@PathParam("a") int a,@PathParam("b") int b,@PathParam("opt") String opt)
{
    if(opt.equals("-"))
        return a-b;
    else
            return 0;
}

@POST
@Path("/mul")
public int mul(@FormParam("a") int a,@FormParam("b") int b,@FormParam("opt") String opt)
{
    if(opt.equals("*"))
        return a*b;
    else
        return 0;
```

RESTful Web Application



RESTful Web Application

Test RESTful Web Services

- REST_Calculator
  - generic
    - calculator_Post
    - add/{a},{b},{opt}
    - sub/{a},{b},{opt}
    - div
    - mul
    - calget/{a},{b},{opt}

REST_Calculator > generic > calculator_Post

Resource: generic/calculator_Post
(http://localhost:8080/REST_Calculator/webresources/generic/calculator_Post)

Choose method to test: POST(application/x-www-form-urlencoded) Test

Content: a=5&b=10&opt=-

Custom Request Headers

Status: 200 (OK)

Response:

Tabular View | Raw View | Sub-Resource | Headers | Http Monitor

-5

Java code for consuming the service

```java
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MediaType;

import org.glassfish.jersey.client.ClientConfig;

public class HelloClient {

    public static void main(String[] args) {
        String uri = "http://localhost/HelloREST/rest/bonjour";
        ClientConfig config = new ClientConfig();
        Client client = ClientBuilder.newClient(config);
        WebTarget target = client.target(uri);

        String response = target.request()
                                .accept(MediaType.TEXT_HTML)
                                .get(String.class);

        System.out.println(response);

    }

}
```

Python code for consuming the service

```
#Python code to consume(get) Java RESTful web service

import requests
api_url = "http://localhost:8080/REST_Join/webresources/generic/"
response = requests.get(api_url)
#response.status_code
#response.headers["Content-Type"]
response.text

#Python code to consume(get) Java RESTful web service
import requests
api_url = "http://localhost:8080/REST_Join/webresources/generic/"
response = requests.put(api_url, "200")
response.text
```

Thank you