



**MODULE-14**  
**WEB SERVICES AND PROJECT**

# Course Topics

## → Module 1

- » Introduction to Java

## → Module 2

- » Data Handling and Functions

## → Module 3

- » Object Oriented Programming in Java

## → Module 4

- » Packages and Multi-threading

## → Module 5

- » Collections

## → Module 6

- » XML

## → Module 7

- » JDBC

## → Module 8

- » Servlets

## → Module 9

- » JSP

## → Module 10

- » Hibernate

## → Module 11

- » Spring

## → Module 12

- » Spring, Ajax and Design Patterns

## → Module 13

- » SOA

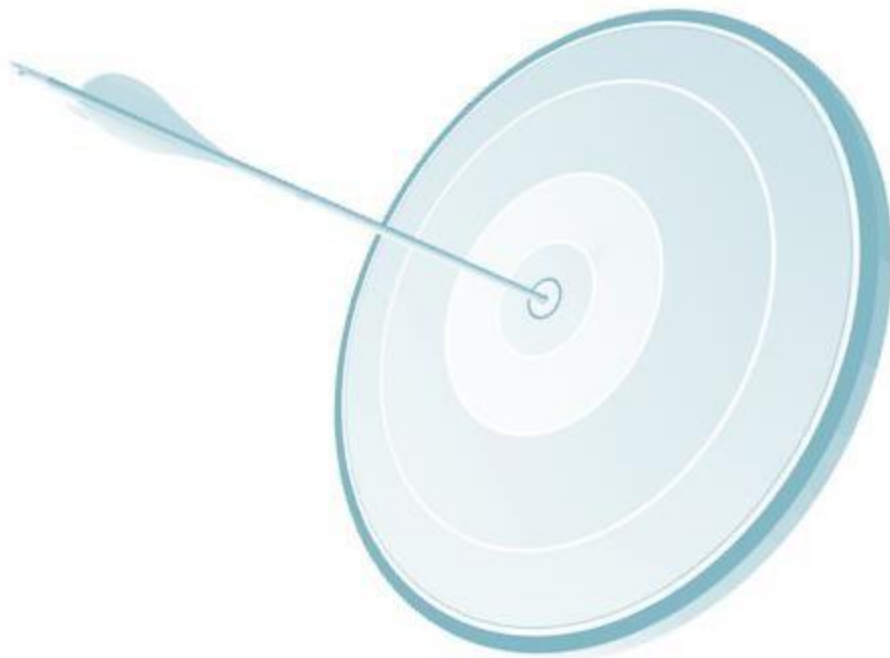
## → Module 14

- » **Web Services and Project**

# Objectives

At the end of this module, you will be able to

- Understand Web Services and its types
- Implement WSDL File
- Create a SOAP based web service
- Create a RESTful web service
- Get the glimpse of the final Project





# Web Services

# Web Services – Introduction

A web service is a piece of business logic, located in some location on internet, that is accessible through HTTP.

A web service could be as simple as visiting a web site or as complex as facilitating a multi organization operations.

## Characteristics of a Web Service:

- Data can be exchanged using XML format.
- Loosely coupled.
- Ability to be Asynchronous or Synchronous.

# Why Web Services

## From Business Point:

### Integration

- with in an organization
- between companies
- allows time/cost efficiencies
  - » Purchase orders.
  - » Answering enquiries.
  - » Processing shipment requests and many more

# Examples of Using Web Services

→ A stock quote service:

An application needs current value of the given stock then web service can return it.

→ A route finder for delivery of goods:

Given the initial and final location, find the most cost-effective delivery route.

→ A weather service:

An application wants to get a regular update of weather then a web service can be used which provides this service.

# Web Services

---

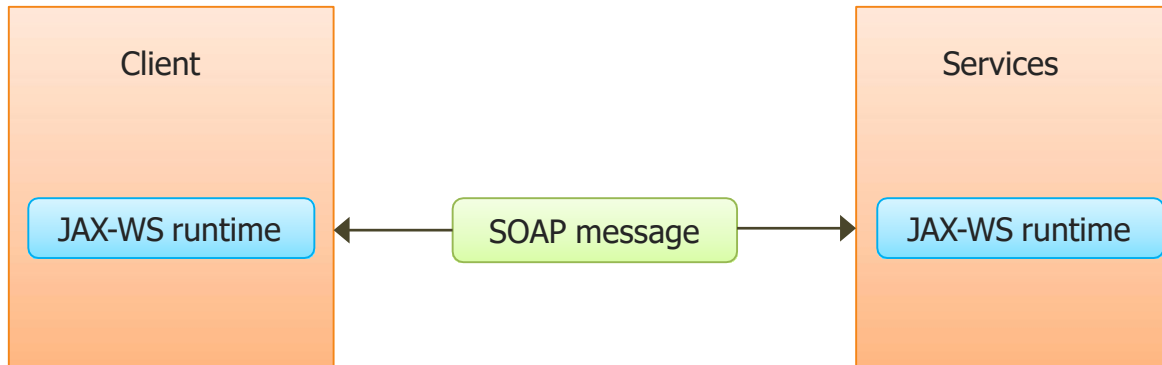
Web services can be implemented either using

- SOAP – Simple Object Access protocol. Uses XML for exchange of data.
- Restful Services – Uses HTTP protocol to access the web service.



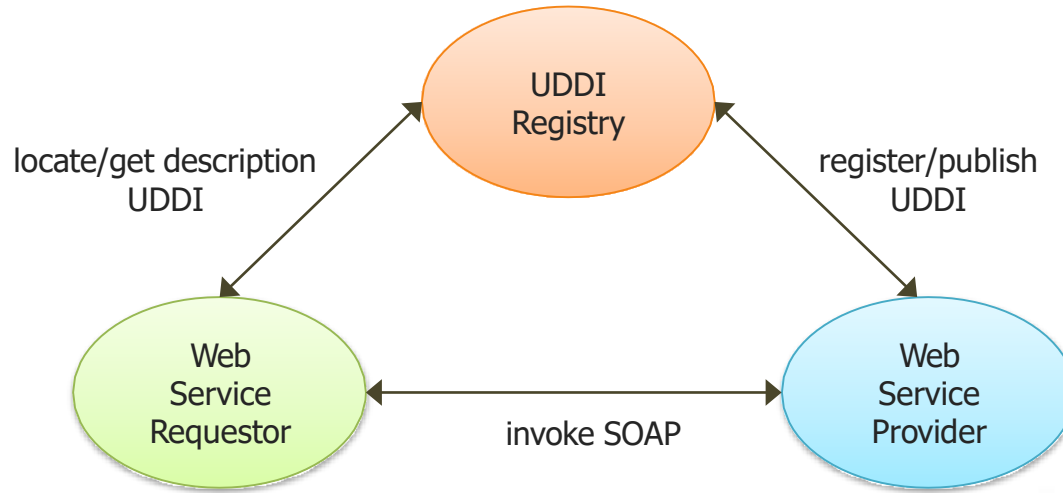
# Web Services - SOAP

Client interacts with the service using SOAP message. SOAP is written in XML

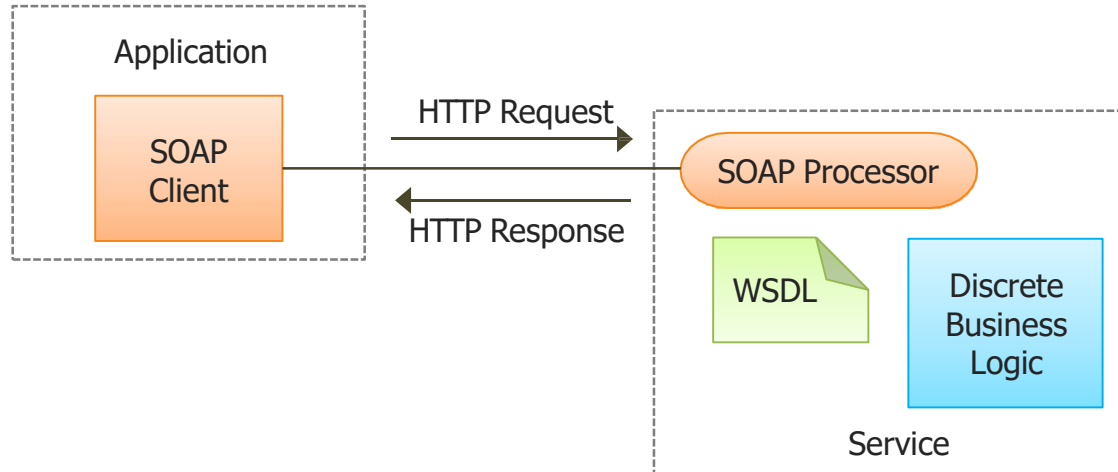


# Web Services - SOAP

- Web Service is published in UDDI registry
- UDDI stands for Universal Description, Discovery and Integration
- Client looks in the registry for the services and gets connected with the required service and interacts with it



# Web Services Interaction - SOAP



# Web Services - SOAP

- Client data is converted into XML using SOAP format. It is sent on internet.
- SOAP message is parsed by the service at the server side.
- Data and the required information is understood.
- Performs the client request and returns back the data to client in SOAP format.

# SOAP

---

- SOAP stands for **S**imple **O**bject **A**ccess **P**rotocol
- SOAP is used for communicating between two applications
- SOAP is a format for sending messages
- SOAP communicates via Internet using HTTP
- SOAP is platform independent
- SOAP is written using XML
- SOAP is a W3C recommendation

# SOAP Format

---

- **SOAP Envelope** – This the root element of a SOAP message.
- **SOAP Header** – It is optional and used to specify application specific information.
- **SOAP Body** – This portion contains the actual SOAP message which needs to be sent to the web service. It also includes the data which needs to be sent the server.

# Example of SOAP Request

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:s="http://www.stock_brokerage.org/stock">
    <s:GetStockPrice>
      <s:StockName>Oracle</s:StockName>
    </s:GetStockPrice>
  </soap:Body>

</soap:Envelope>
```

# Example of SOAP Response

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:s="http://www.Stock_brokerage.org/stock">
    <s:GetStockPriceResponse>
      <s:Price>100.50</s:Price>
    </s:GetStockPriceResponse>
  </soap:Body>

</soap:Envelope>
```



# WSDL

---

- WSDL is **W**eb **S**ervices **D**escription **L**anguage
- WSDL is written in XML and it is a XML document
- WSDL describes the Web services
- WSDL is a W3C recommendation

# WSDL

WSDL XML document has the following elements:

- `<types>` - This section is used to specify the data types used by the web services.
- `<messages>` - Messages for communication will be specified in this section. Input message and output message can be specified in this section.
- `<portType>` - This defines the web service and the operations / methods to be executed. Also the messages that are involved in this operation /method. Messages could be input message and output message.
- `<binding>` - This section defines the message bindings and transportation protocol.
- `<service>` - Location of the service – URL.

# Example of WSDL File

```
<definitions name="HelloService" targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <message name="SayHelloRequest">
    <part name="firstName" type="xsd:string"/>
  </message>
  <message name="SayHelloResponse">
    <part name="greeting" type="xsd:string"/>
  </message>
  <portType name="Hello_PortType">
    <operation name="sayHello">
      <input message="tns:SayHelloRequest"/>
      <output message="tns:SayHelloResponse"/>
    </operation>
  </portType>
  <binding name="Hello_Binding" type="tns:Hello_PortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="sayHello">
      <soap:operation soapAction="sayHello"/>
      <input>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:examples:helloservice" use="encoded"/>
      </input>
      <output>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:examples:helloservice" use="encoded"/>
      </output>
    </operation>
  </binding>
  <service name="Hello_Service">
    <documentation>WSDL File for HelloService</documentation>
    <port binding="tns:Hello_Binding" name="Hello_Port">
```

# Apache Axis

---

Apache Axis (Apache Extensible Interaction System) is an open-source, XML based Web service framework.

It consists of a Java and a C++ implementation of the SOAP server, and various utilities and APIs for generating and deploying Web service applications. Using Apache Axis, developers can create interoperable, distributed computing applications.

Apache Axis can be integrated with Eclipse.

You can download Apache Axis for Eclipse from this site:

<https://axis.apache.org/axis2/java/core/tools/eclipse/plugin-installation.html>

Apache Axis generates the WSDL code and the required client

# Web Services Development using SOAP

- Download Apache Axis2 and integrate with eclipse.
- Create a Dynamic Web Project from Eclipse.
- Create a class under Src Directory.
- Write a public function. It could be like as given below:

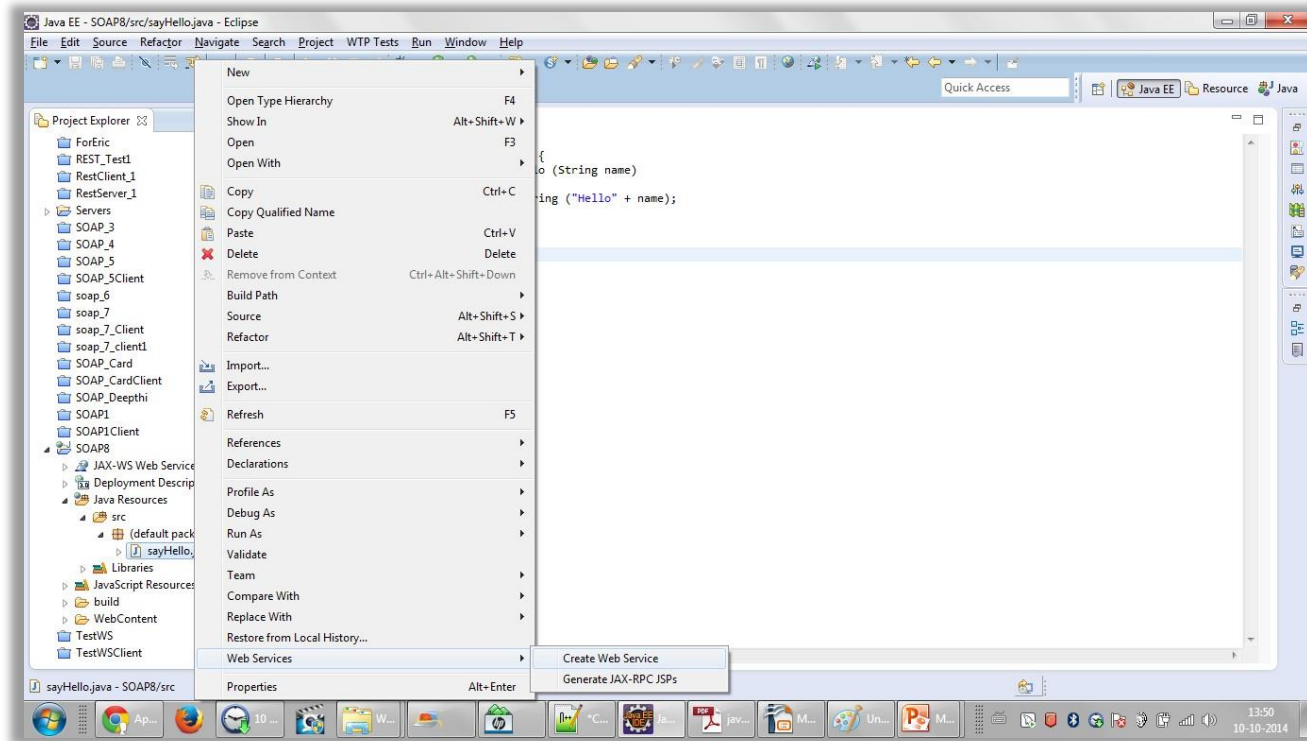
```
public class sayHello {  
    public String hello (String name)  
    {  
        return new String ("Hello" + name);  
    }  
}
```

# Web Services Implementation

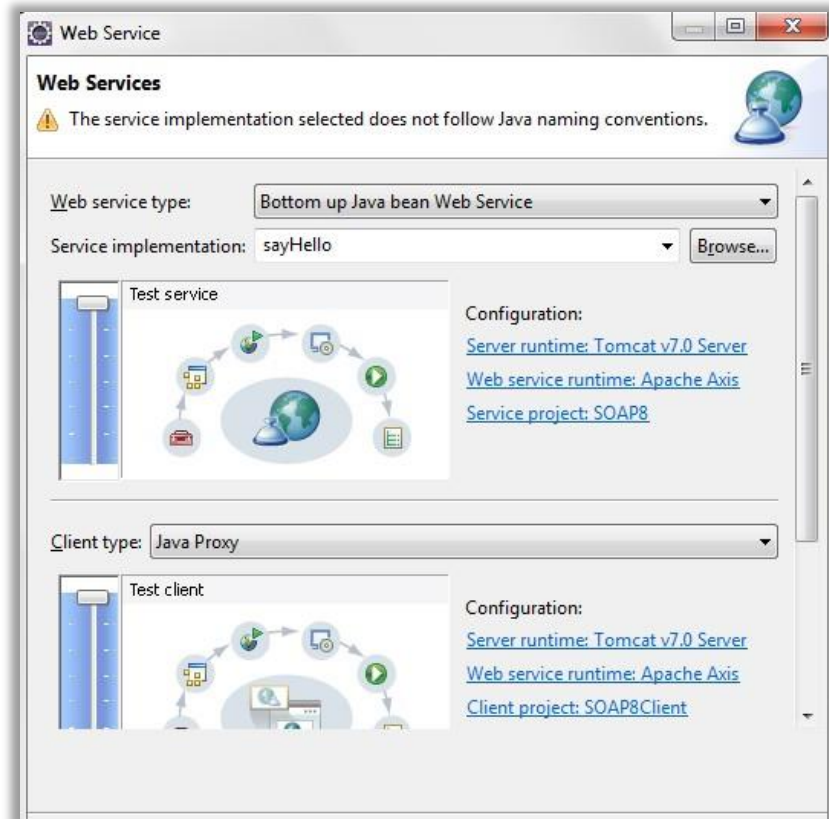
---

- Select the Java file from eclipse project explorer.
- Right click on it and select web services and select and click “create web service”.

# Web Services Implementation

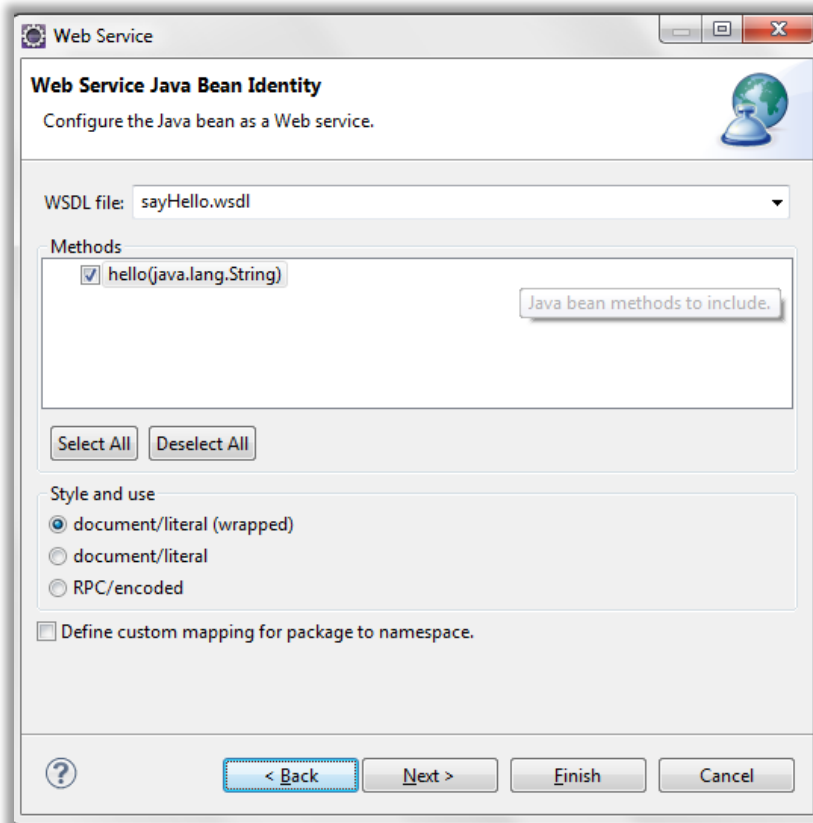


# Web Services Implementation

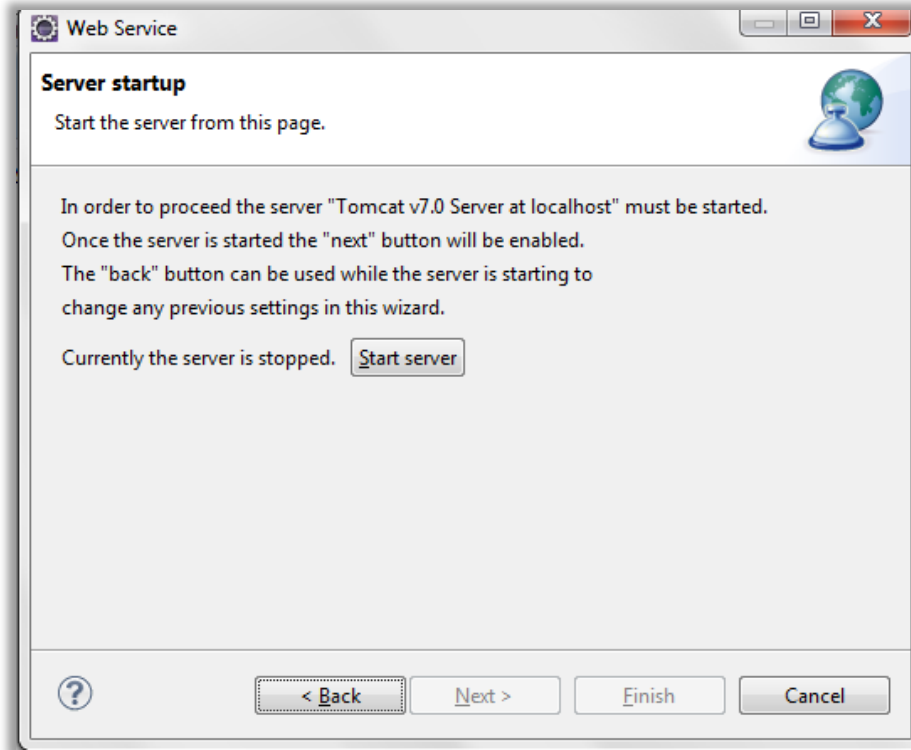




# Web Services Implementation



# Web Services Implementation

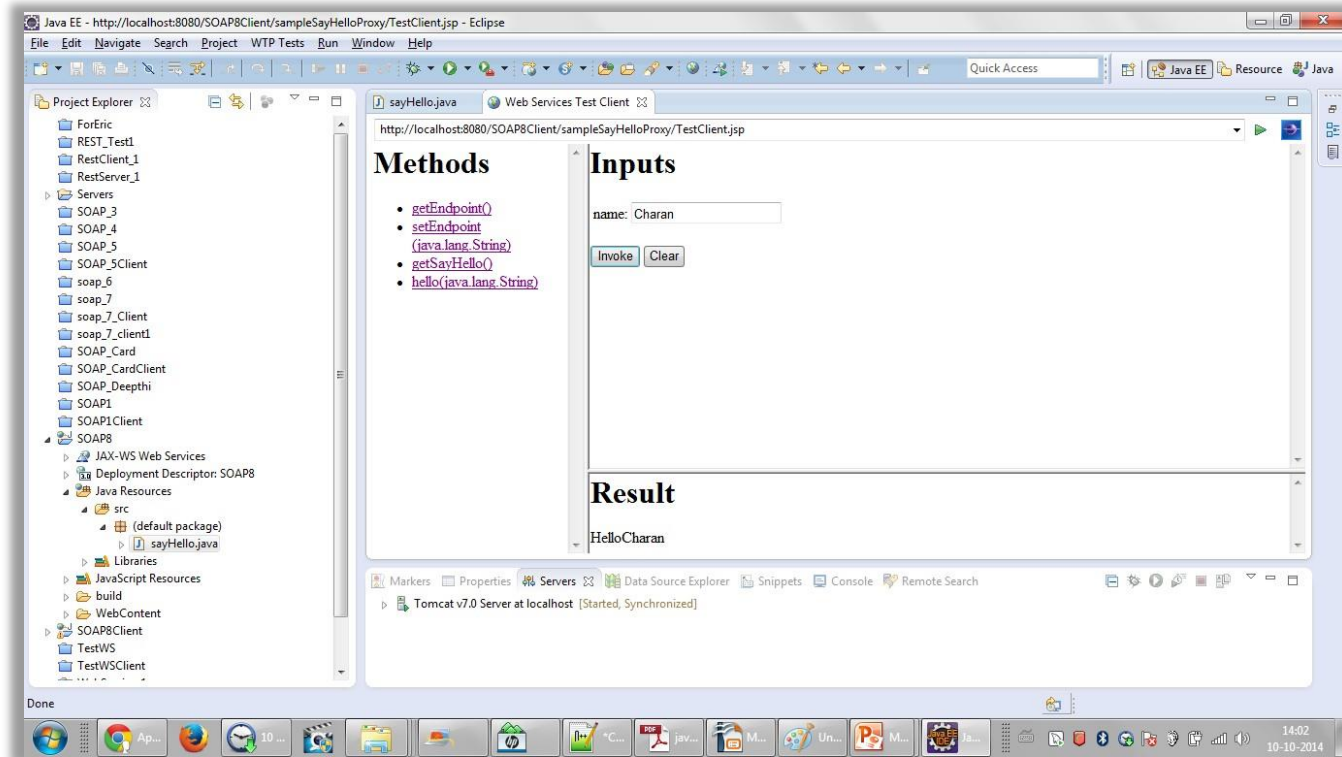


# Web Services Implementation

---

- Click on start server
- Click on Finish
- Click on the function developed

# Web Services Implementation



# REST Services

- REST stands for Representational State Transfer
- This service is developed using HTTP protocol
- In Rest based architecture, similar to SOAP based Architecture, service is developed in REST and REST client has to be written to use the REST Service
- Java defines REST support via the Java Specification Request (JSR) 311. This specification is called JAX-RS (The Java API for RESTful Web Services)

# JAX-RS

---

- Jersey is the reference implementation for JSR (Java Specification Request)-311
- Jersey provides the library for RESTful Web Services using Servlet container
- Download Jersey distribution from the site: <https://jersey.java.net/download.html>

# Restful Services

→ REST is an architecture. RESTful service is a service which follows the REST architecture.

→ Following Annotations are used:

- » **@Path (path)** – Sets the path to the base URL. The base URL is the application name, servlet and URL specified in the web.xml.
- » **@GET** – Indicates the following method will address HTTP GET request.
- » **@Produces** – This function will generate the given output. It could be TEXT/HTML/XML formats etc.

# Creating Restful Service

---

- Create a dynamic web project in eclipse.
- Add jersey jar files to the build path.
- Add the web.xml under web content/WEB-INF directory.
- Add a Java file and add the required code of the service.



# Restful Service – web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
id="WebApp_ID" version="2.5">
  <display-name>jax_Rest1</display-name>
  <servlet>
    <servlet-name>Jersey REST Service</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
    <init-param>
      <param-name>com.sun.jersey.config.property.packages</param-name>
      <param-value>myPackage</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey REST Service</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>
</web-app>
```

# RESTful Service – Java Code

```
@Path("/Hello")
public class Hello {
    // This method is called if TEXT_PLAIN is request
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String sayPlainTextHello() {
        return "Hello Jersey1";
    }
}
```

```
@GET
@Produces(MediaType.TEXT_XML)
public String sayXMLHello() {
    return "<?xml version='1.0'>" + "<hello> Hello Jersey2" + "</hello>";
}
```

# Creating Restful Service

- Hello should be added for the path of this service.
- Name of the class is Hello.
- **@GET** – The following method will respond to GET method of the client request.
- **@Produces** – This method returns the data specified in the Produces annotation.
- Method returns the data.
- Run the project so that Service is running on Apache Tomcat.

# RESTful – Client

```
public class Test_Client {  
    public static void main(String[] args) {  
        ClientConfig config = new DefaultClientConfig();  
        Client client = Client.create(config);  
        WebResource service = client.resource(getBaseURI());  
        // Get plain text  
        System.out.println(service.path("rest").path("Hello").accept(MediaType.TEXT_PLAIN).get(String.class));  
        // Get XML  
        System.out.println(service.path("rest").path("Hello").accept(MediaType.TEXT_XML).get(String.class));  
        // The HTML  
        System.out.println(service.path("rest").path("Hello").accept(MediaType.TEXT_HTML).get(String.class));  
    }  
  
    private static URI getBaseURI() {  
        //Try the following from the browser.  
        //http://localhost:8080/jax_rest2/rest/Hello?name=charan  
  
        return UriBuilder.fromUri("http://localhost:8080/jax_rest2").build();  
    }  
}
```

# RESTful – Client

- Base URL is obtained for the service.
- Methods of RESTful services are invoked by using the path and the media type.
- Apart from writing the client, service can be tested with the URL too

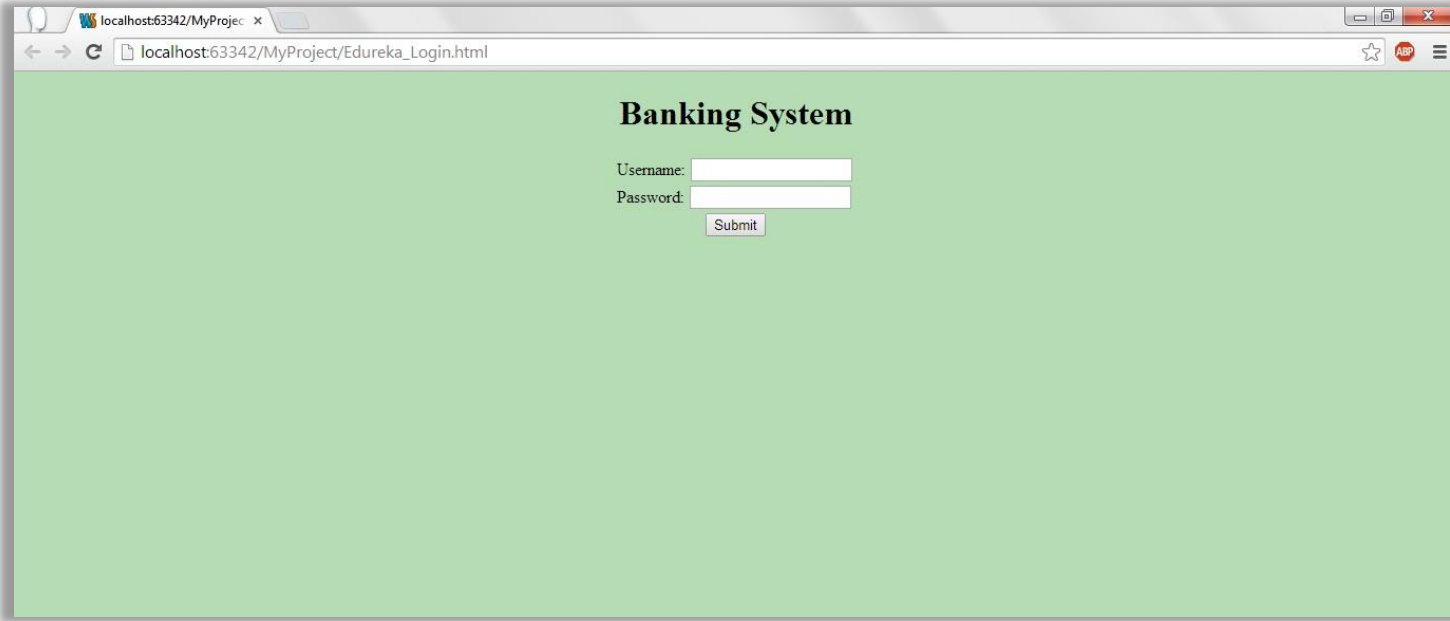
For example:

- » [http://localhost:8080/jax\\_rest2/rest/Hello?name=charan](http://localhost:8080/jax_rest2/rest/Hello?name=charan)
- » Here, service is running under 8080 port.  
Name of the application is jax\_rest2, /rest/Hello is the Path and name=charan is passed as parameter to this service.



# PROJECT

# User Login Screen



The screenshot shows a web browser window with the address bar displaying 'localhost:63342/MyProject/Edureka\_Login.html'. The page content is centered on a light green background. At the top, the text 'Banking System' is displayed in a bold, black, serif font. Below this, there are two input fields: 'Username:' followed by a white text box, and 'Password:' followed by a white text box. A small 'Submit' button is positioned below the password field. The browser's address bar also shows navigation icons (back, forward, refresh) and a star icon for bookmarks.

**Banking System**

Username:

Password:

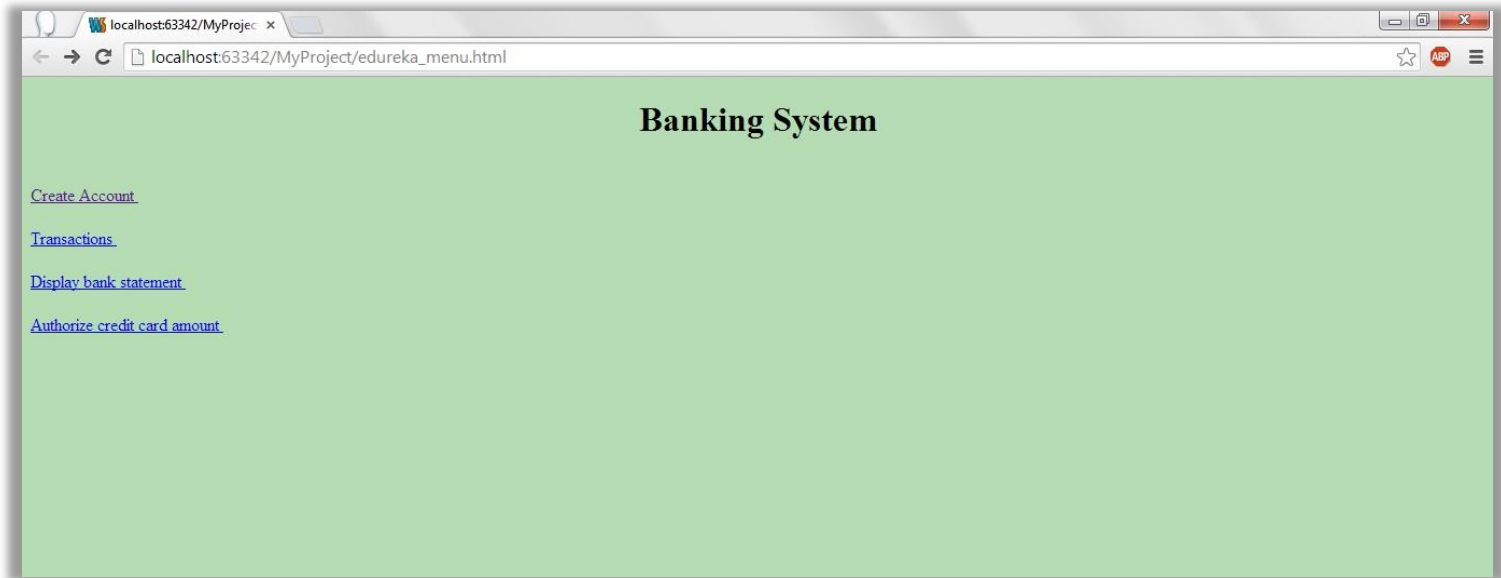
# Login Screen

---

- In this screen, user enters the user id and password.
- Entered user id and password has to be verified from the table user\_pass.
- If the entered user is right, account creation screen is displayed else the message invalid user id/password is displayed to the user.



# Main Menu Display

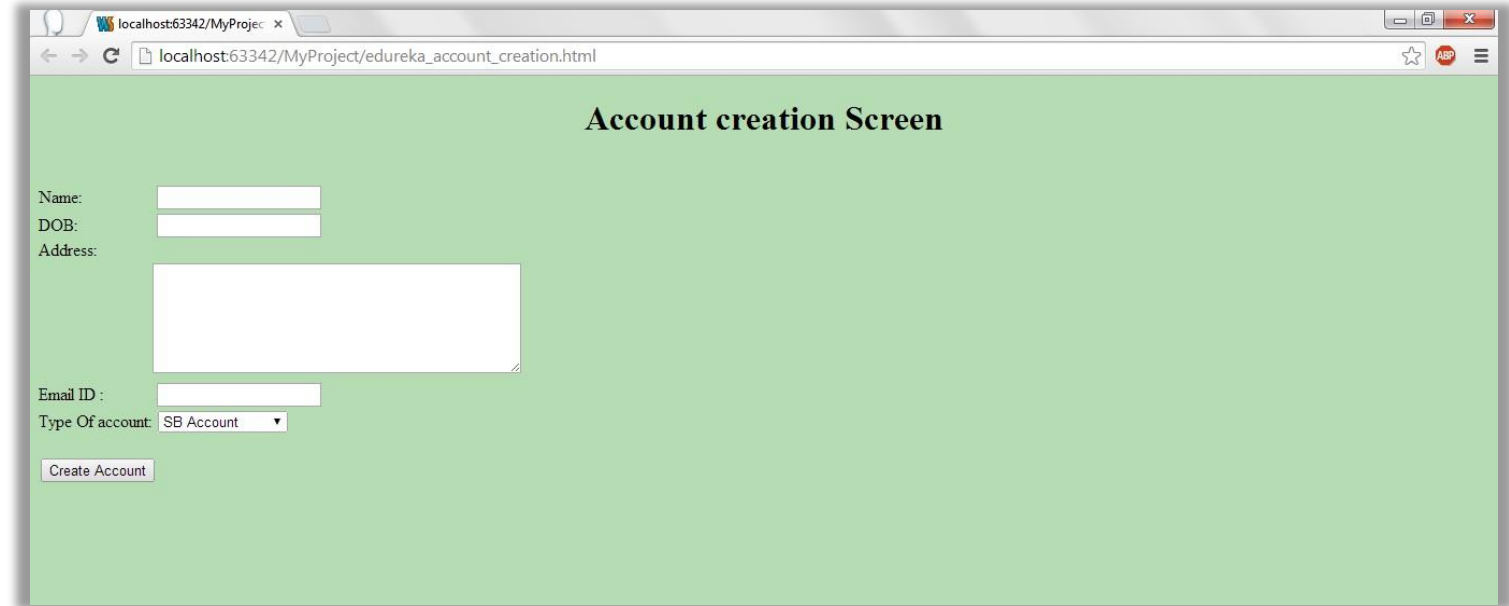


# Main Menu

---

- If the user enters a valid user id and password then this (Main Menu screen) will be displayed.
- Based on the user's choice corresponding action is taken.

# Account Creation Screen



The screenshot shows a web browser window with the title bar "localhost:63342/MyProjec" and the address bar "localhost:63342/MyProject/edureka\_account\_creation.html". The page has a light green background and is titled "Account creation Screen" in bold black text. The form contains the following fields and controls:

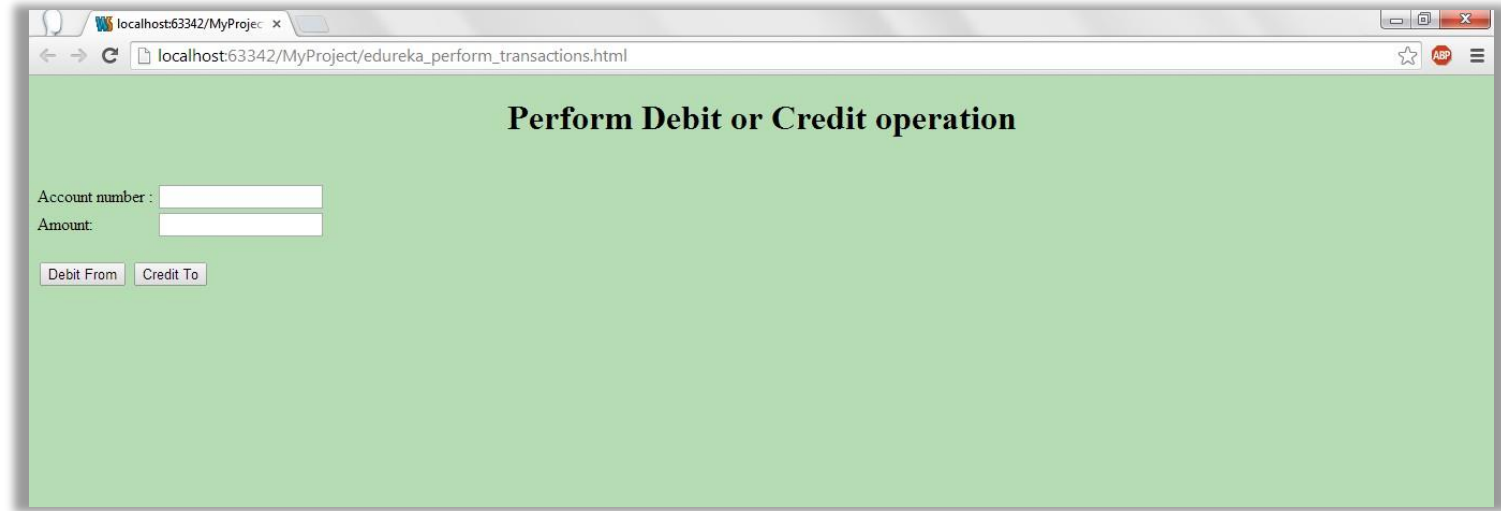
- Name:** A single-line text input field.
- DOB:** A single-line text input field.
- Address:** A multi-line text area.
- Email ID :** A single-line text input field.
- Type Of account:** A dropdown menu with "SB Account" selected.
- Create Account:** A button with a light purple gradient.

# Account Creation

---

- In the main menu screen, when the user selects Account Creation, this screen will be displayed.
- User will enter the data in this screen and clicks on the create account button.
- With this, entered data will be entered into the table account\_details.
- After performing this operation, user will be shown the appropriate message as "Account created successfully" or "User Account is not created".

# Transactions



A screenshot of a web browser window. The address bar shows the URL `localhost:63342/MyProject/edureka_perform_transactions.html`. The page has a light green background and a title **Perform Debit or Credit operation**. Below the title, there are two input fields: "Account number :" and "Amount:". Below these fields are two buttons: "Debit From" and "Credit To".

**Perform Debit or Credit operation**

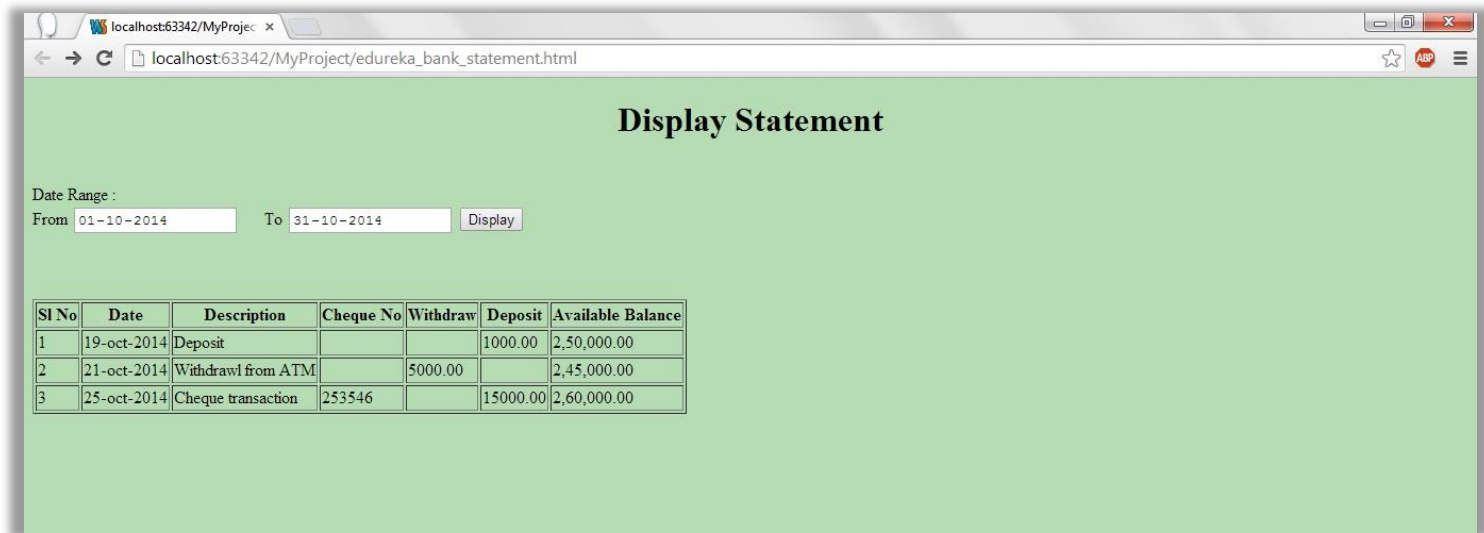
Account number :

Amount:

# Transactions

- From the main menu, when the user clicks on Transactions, this screen will be displayed.
- Account number and the amount to be transferred will be entered by the user.
- If the user clicks on Debit From, the given amount will be debited from the specified account to the logged in user's account. If the chosen option is **Credit To**, then from the logged in user's account specified amount will be transferred to the given account number.

# Display Statement



**Display Statement**

Date Range :

From  To

SI No	Date	Description	Cheque No	Withdraw	Deposit	Available Balance
1	19-oct-2014	Deposit			1000.00	2,50,000.00
2	21-oct-2014	Withdrawal from ATM		5000.00		2,45,000.00
3	25-oct-2014	Cheque transaction	253546		15000.00	2,60,000.00

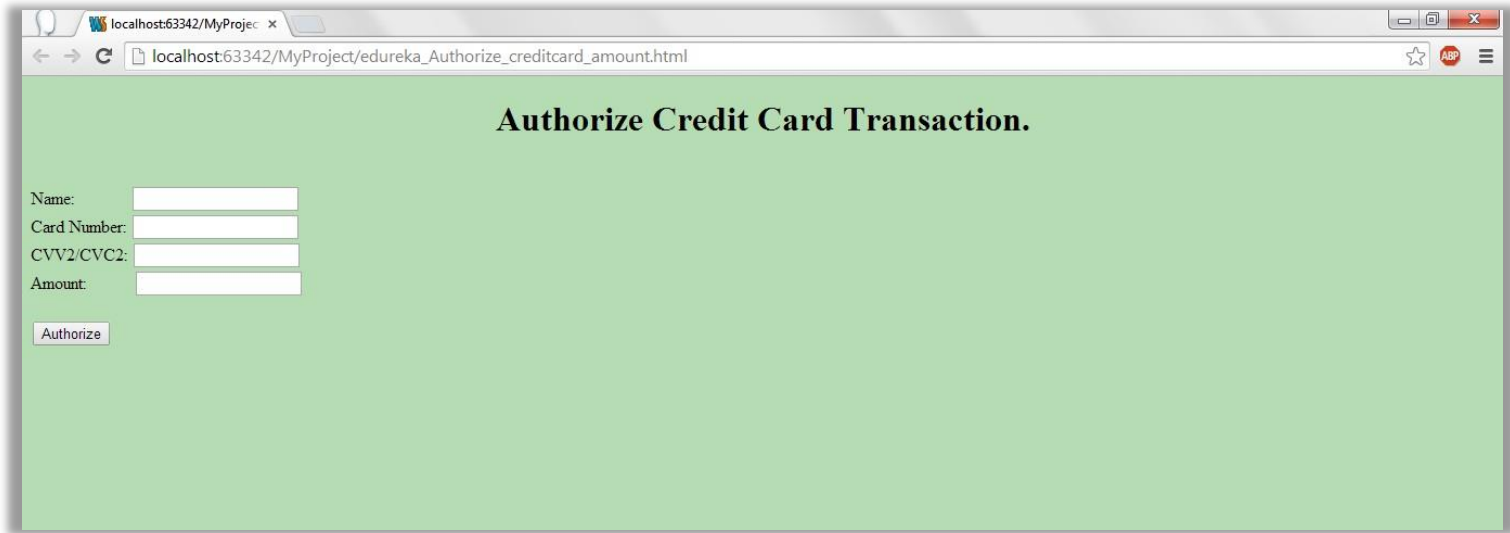
# Display Statement

---

- When the user selects Display statement from the main menu, this screen will be displayed.
- User enters the date range and clicks on display button.
- With the specified date range logged in, user's transactions are displayed in the given format.



# Authorize Credit Card Transactions



A screenshot of a web browser window displaying a form titled "Authorize Credit Card Transaction." The browser's address bar shows the URL "localhost:63342/MyProject/edureka\_Authorize\_creditcard\_amount.html". The form is set against a light green background and includes the following elements:

- Title:** "Authorize Credit Card Transaction." centered at the top.
- Input Fields:** Four white text boxes for "Name:", "Card Number:", "CVV2/CVC2:", and "Amount:", each preceded by its respective label.
- Submit Button:** A button labeled "Authorize" located below the input fields.

# Authorize Credit Card Transactions

- When the user clicks on authorize credit card amount on main menu then this screen will be displayed.
- User will enter all the details and will click on Authorize. Using web services this client will get to know whether card holder's credit amount is approved or not. After performing this operation, appropriate message should be displayed to the user as **Approved** or **Rejected**.

# Say Bye to John!!



That's all Folks!!

This is the end of our Java Journey. But your learning doesn't stop here. Do keep checking the LMS to learn more and keep practicing.

[Remember to attempt the Project.](#)

# QUESTIONS



# Assignment

Practice the SOAP and RESTful Web Services given as the example in the class.

Note: You will find the examples under sample programs in the LMS.



Thank you!

