|  |  |
| --- | --- |
| **Roll No.:** A059 | **Name:** Chinmay Parikh |
| **Prg/Yr/Sem:** B.Tech(I.T.)/4th /7 | **Batch:** A3 |
| **Date of Experiment:** 23/8/2014 | **Date of Submission:** 30/8/2014 |

**Aim**: Implementing Java Web Service JEE6 Application (Factorial) using JAX-WS API and accessing through a Web Client.

1. **Design Scenario –**

a. Creating a Java Web Service Factorial with operation fact with an integer parameter.

b. Creating a Client Web Service which accesses the factorial service and displays the result of factorial in a jsp page.

The Java API for XML Web Services (JAX-WS) is a Java programming language API for creating web services. JAX-WS is also used to build Web services and corresponding clients that communicate using XML to send messages or use remote procedure calls to exchange data between client and service provider.

JAX-WS represents remote procedure calls or messages using XML-based protocols such as

SOAP, but hides SOAP's innate complexity behind a Java-based API. Developers use this

API to define methods, and then code one or more classes to implement those methods and leave the communication details to the underlying JAX-WS API. Clients create a local proxy to represent a service, and then invoke methods on the proxy. The JAX-WS runtime system converts API calls and matching replies to and from SOAP messages.

1. **Detailed Steps for creating a web service & web service client in Net Beans 7.0**

*Steps –*

Part 1

1. Start Net Beans 7.0 by clicking on it and start a new project File->New Project. Select the New Project as Java Web -> Web Application.
2. Click on Next and specify the application name as FactorialWebService.
3. This displays the created application FactorialWebService on the Projects list with the default index.jsp page as Hello World.
4. Right Click on the Application FactorialWebService and select New->Web Service. Create the web service called factorial in package practfactorial.
5. Now Select the created web service factorial and add an operation to it. Right Click ->Add Operation.
6. Create the operation as fact with parameter x as int.
7. Now click on the source view Here make some modification in the business logic for
8. factorial calculation we can see the code of factorial.java which is our web service. It has @WebServices as factorial, @WebMethod as fact and @WebParam as x.
9. Add the code in the appropriate place which is highlighted in a red box. This is the logic to calculate factorial given the parameter x by the client service.
10. In the project window right click on the application FactorialWebService and Build it. After successfully build then deploy it. This will start the GlassFishServer 3.1.
11. After the application is up working now select the service factorial and right click on it and select Test Web Service.
12. It will run the web service on the server and displayed in the browser.
13. We can enter a fact number 5 to obtain the answer click on fact button, This invokes the fact operation and displays the SOAP request & response code. The WSDL file can be displayed by clicking on WSDL File in the factorial Web Service Tester page.
14. Copy the WSDL URL in a safe place as this is required when the client web service is created. http://localhost:8080/FactorialWebService/factorial?WSDL.

Part 2

1. Start a new project File->New Project. Select the New Project as Java Web -> Web Application. Specify the Project Name as FactorialClientWS.
2. After the client project application is created we have to add a web service to it by right clicking on the project FactorialClientWS selecting New->Web Service Client.
3. Add the WSDL URL which was displayed when creating the service and paste it in WSDL URL http://localhost:8080/FactorialWebService/factorial?WSDL. Specify the package as practfactorial.
4. After this the client service fact has to be added to the index.jsp page as output for the client application. For this select the fact operation and simply drag it into the index.jsp page content after the <h1>Hello World </h1> html code and rewrite the value of x as 5.
5. Now run the project by right clicking on FactorialClientWS and selecting Run. This will display a web page with the contents Hello World And the result of the factorial 5 which is 120. We can re run the project with different factorial values.
6. **Questions –**
7. **Distinguish earlier approaches and SOA architecture style.**

**Ans:**

From its main predecessors:-Distributed Component Model (DCM), a subset of CBSE, SOA carries the following features: composability,separation of concerns,loose coupling and abstraction. SOA combines benefits and challenges from COTS and CBD except that services are consumed from wherever they are. We note that SOA relaxes the restrictions on the features to make it unique from its predecessors.

1. **Explain characteristics and benefits of SOA.**

**Ans:** Characteristics –

There is no clear definition of SOA in the pertinent literature. However, it is possible to identify certain key features which are widely considered to apply to all service-oriented architectures.

1. Standardized service interfaces: one of the fundamental requirements is the need for standardized interfaces and specifications. The specifications must include how the service can be used, which data (types) are required and how certain guidelines can be applied.

2. Loose connection: it should be possible to link the services together loosely to form one process. This presupposes a modicum of interdependence between individual services. The principle is to limit the amount of dependency to the level required to still guarantee interoperability (and compatibility).

3. Functional abstraction: services should prescind outward functional specifics and implementation details in order to enable their loose connection. The only functions to be encapsulated are outward services and specifications.

4. Reusability: one basic idea is the reusability of services further down the process chain by other parties or even for other applications. This idea must be factored in at the development stage.

5. Service autonomy: a service should be able to function independently. Service autonomy is the term used to describe a service which is self-sufficient in terms of managing all the necessary logic, resources and environment and which is independent of external services.

6. Statelessness of service: the philosophy behind the performance of services is that a defined service is rendered. This can also involve data retention but only if expressly requested. Other services do not hold any master data and therefore do not have to perform any status management tasks between two service requests.

7. Findability of service: it must be possible to find a service to be able to use it. This is generally managed via service repositories. A repository is available to all consumers and providers and contains service interface and implementation specifications. It stores all the information required by a consumer in order to request a service.

8. Capacity of service for orchestration: orchestration is the name given to the process whereby individual services are combined in functional and technical terms to form larger units, the business processes. Since the objective of SOA is to map the technical business process, the capacity for orchestration is a further requirement on SOA services. Services should be able to perform individual tasks effectively in an overall process. One of the central requirements is independence from the complexity and extent of the process in any given case.

Benefits –

1. *Reuse:* The ability to create services that are reusable in multiple applications.
2. *Efficiency:* The ability to quickly and easily create new services and new applications using a combination of new and old services, along with the ability to focus on the data to be shared rather than the implementation underneath.
3. *Loose technology coupling:* The ability to model services independently of their execution environment and create messages that can be sent to any service.
4. *Division of responsibility:* The ability to more easily allow business people to concentrate on business issues, technical people to concentrate on technology issues, and for both groups to collaborate using the service contract.
5. **Explain the format of SOAP Request & Response.**

**Ans:** A SOAP request looks like this:

<?xml version="1.0" encoding="UTF-8"?>

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" >

<env:Body>

<GetRelevanceResult xmlns="http://[webreportshostname]:[webreportsport]

/webreports?wsdl">

<relevanceExpr>names of bes computers</relevanceExpr>

<username>user</username>

<password>password</password>

</GetRelevanceResult>

</env:Body>

</env:Envelope>

A SOAP response looks like this:

<?xml version="1.0" encoding="UTF-8"?>

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">

<env:Body>

<GetRelevanceResultResponse "xmlns=http://[webreportshostname]:[webreportsport]

/webreports?wsdl">

<a>Computer 1</a>

<a>Computer 2</a>

<a>Computer 3</a>

</GetRelevanceResultResponse>

</env:Body>

</env:Envelope>

*Envelope –*

The element name is "Envelope".

The element MUST be present in a SOAP message

The element MAY contain namespace declarations as well as additional attributes. If present, such additional attributes MUST be namespace-qualified. Similarly, the element MAY contain additional sub elements. If present these elements MUST be namespace-qualified and MUST follow the SOAP Body element.

*Header –*

The element name is "Header".

The element MAY be present in a SOAP message. If present, the element MUST be the first immediate child element of a SOAP Envelope element.

The element MAY contain a set of header entries each being an immediate child element of the SOAP Header element. All immediate child elements of the SOAP Header element MUST be namespace-qualified.

*Body –*

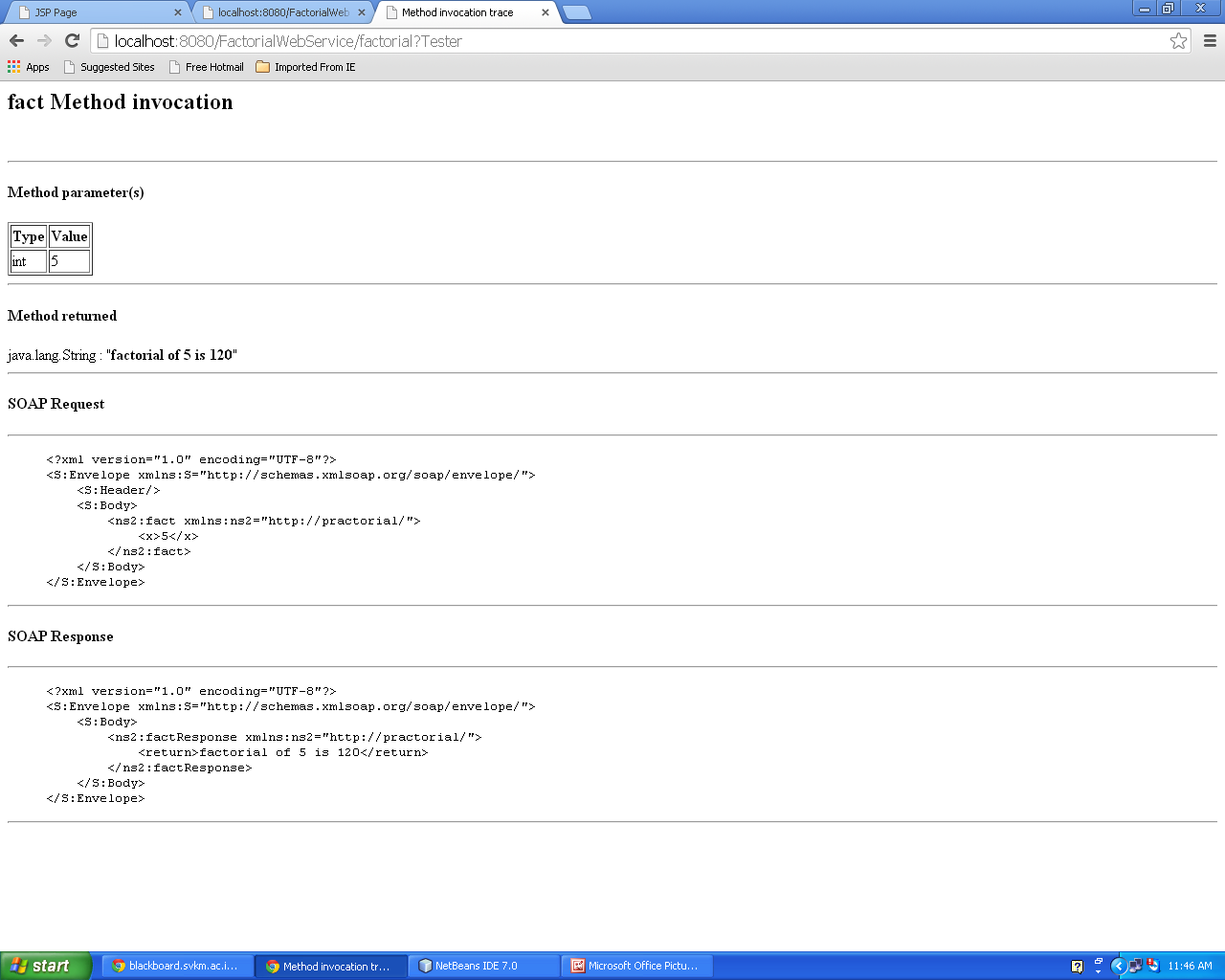
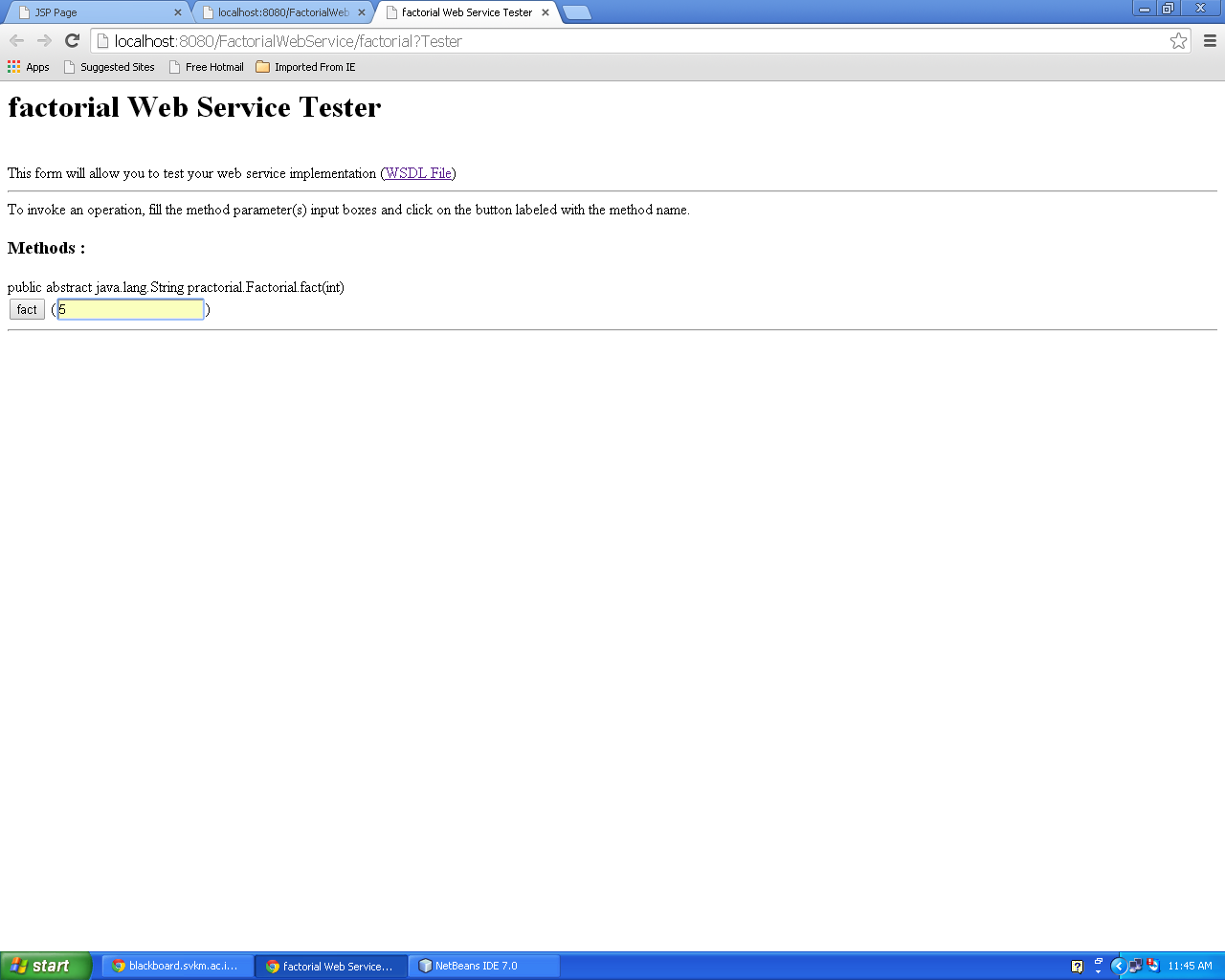
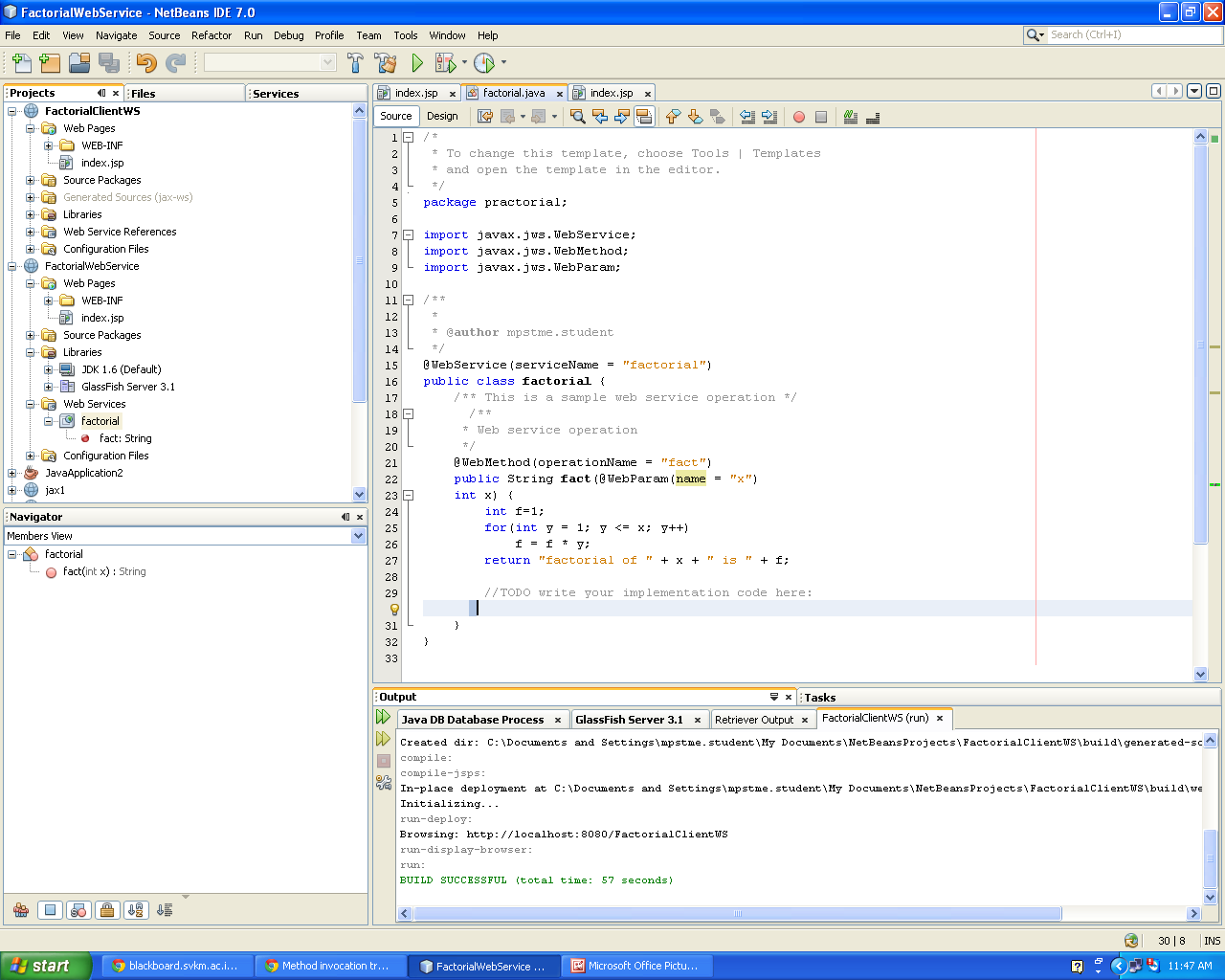
The element name is "Body".

The element MUST be present in a SOAP message and MUST be an immediate child element of a SOAP Envelope element. It MUST directly follow the SOAP Header element if present. Otherwise it MUST be the first immediate child element of the SOAP Envelope element.

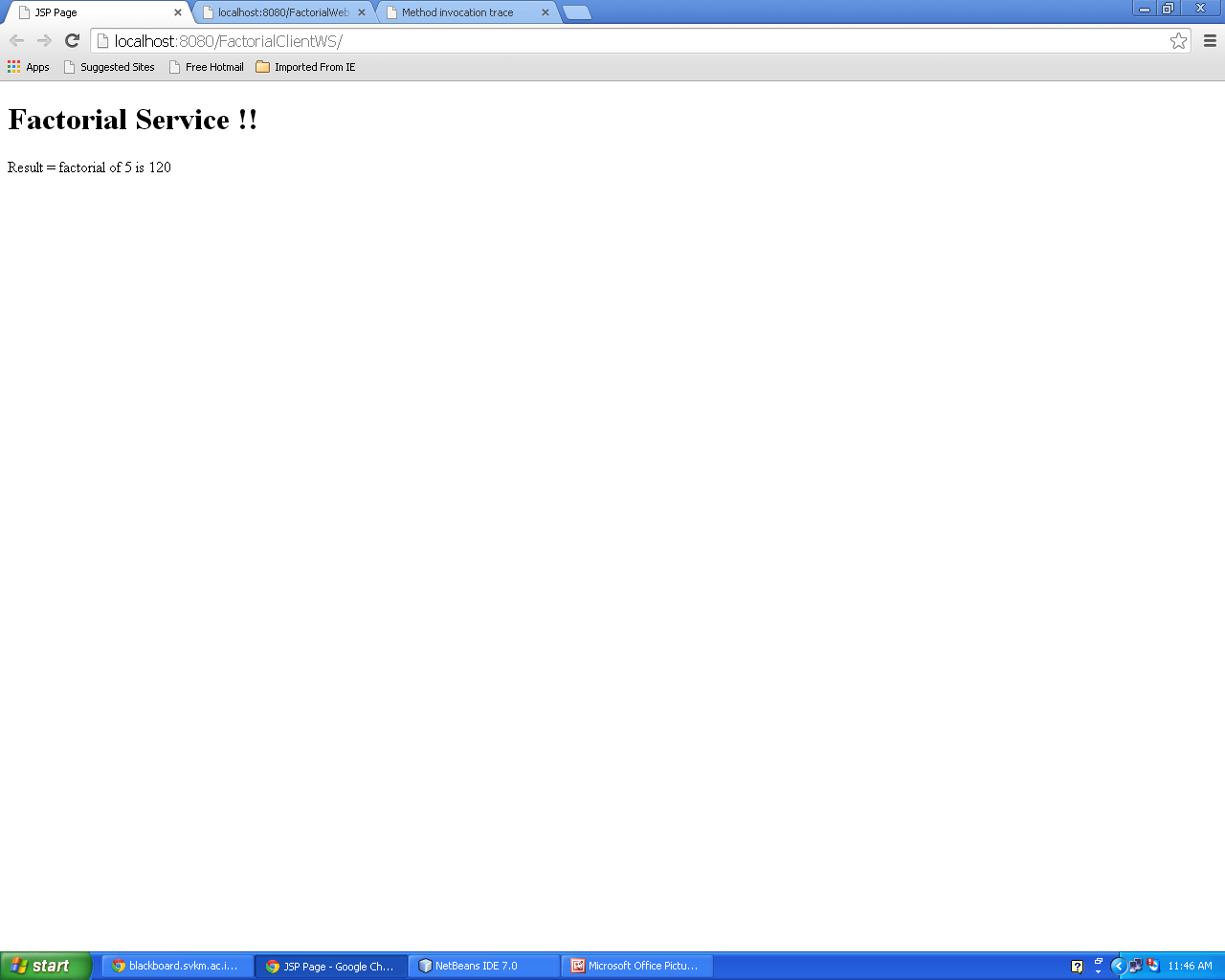
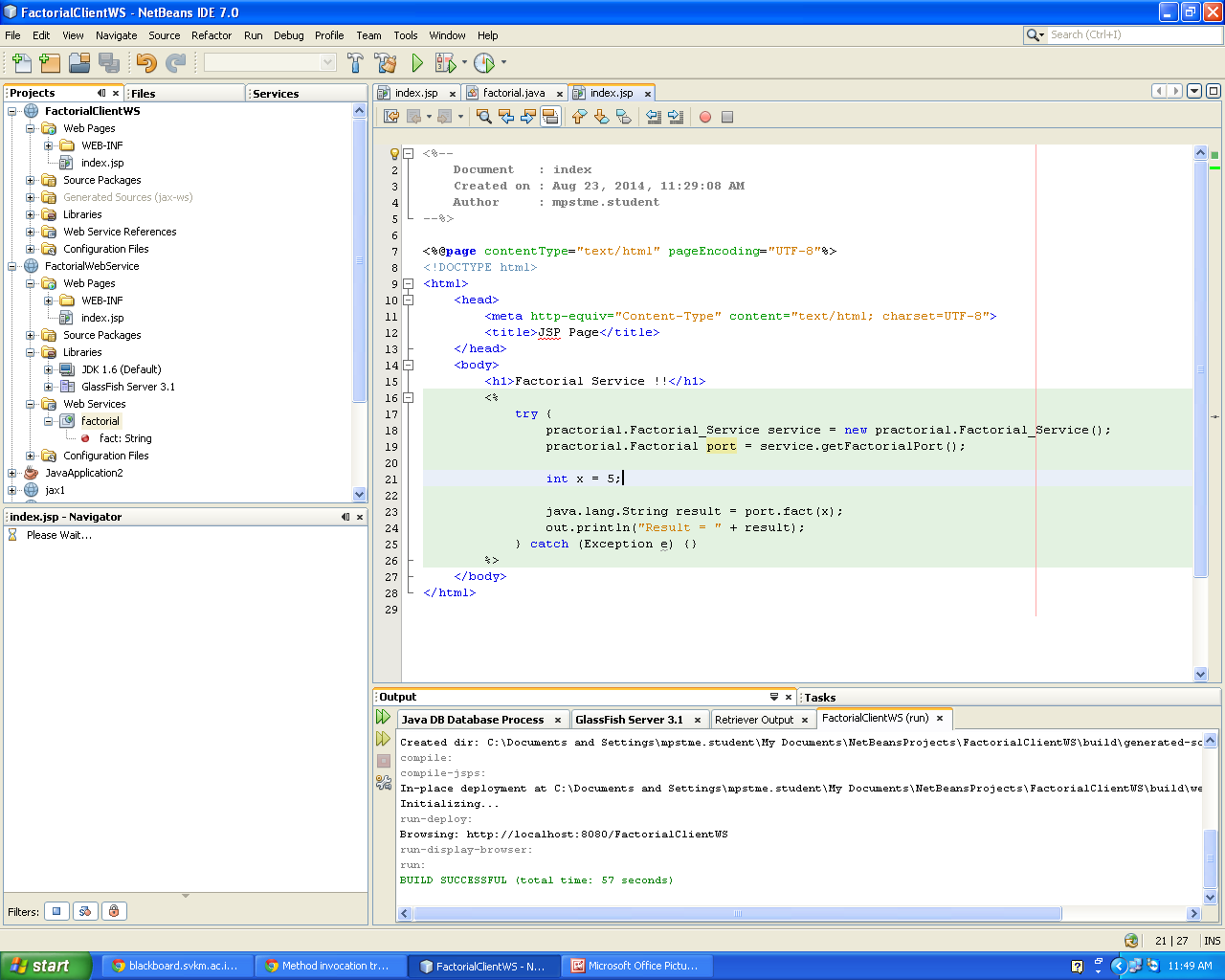
The element MAY contain a set of body entries each being an immediate child element of the SOAP Body element. Immediate child elements of the SOAP Body element MAY be namespace-qualified. SOAP defines the SOAP Fault element, which is used to indicate error messages.

1. **Printouts –**

Part 1 –



Part 2 –



1. **Conclusion –** Hence we have learnt about JAVA Web Services and implemented a factorial program. We implemented it in JAX-WS API and accessed it through a Web Client.