

Experiment No.: 3 Creating Web service

Learning Objective: Student should be able to understand creating web services using and also different web services components.

Theory:

A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response. As all communication is in XML, web services are not tied to any one operating system or programming language—Java can talk with Perl; Windows applications can talk with Unix applications.

Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction. These systems can include programs, objects, messages, or documents. The architecture of web service interacts among three roles: service provider, service requester, and service registry. The interaction involves the three operations: publish, find, and bind. These operations and roles act upon the web services artifacts. The web service artifacts are the web service software module and its description. Web Service Architecture

There are three roles in web service architecture:

- i. **Service Provider:** From an architectural perspective, it is the platform that hosts the services.
- ii. **Service Requestor:** Service requestor is the application that is looking for and invoking or initiating an interaction with a service. The browser plays the requester role, driven by a consumer or a program without a user interface.
- iii. **Service Registry:** Service requestors find service and obtain binding information for services during development.

Different process in web service Architecture:

- Publication of service descriptions (Publish)
- Finding of services descriptions (Find)
- Invoking of service based on service descriptions (Bind)

Publish: In the publish operation, a service description must be published so that a service requester can find the service.

Find: In the find operation, the service requestor retrieves the service description directly. It can be involved in two different lifecycle phases for the service requestor:

- At design, time to retrieve the service's interface description for program development.
- And, at the runtime to retrieve the service's binding and location description for invocation.

Bind: In the bind operation, the service requestor invokes or initiates an interaction with the service at runtime using the binding details in the service description to locate, contact, and invoke the service.

Artifacts of the web service

There are two artifacts of web services:

- Service
- Service Registry

Service: A service is an interface described by a service description. The service description is the implementation of the service. A service is a software module deployed on network-accessible platforms provided by the service provider. It interacts with a service requestor. Sometimes it also functions as a requestor, using other Web Services in its implementation.

Service Description: The service description comprises the details of the interface and implementation of the service. It includes its data types, operations, binding information, and network location. It can also categorize other metadata to enable discovery and utilize by service requestors. It can be published to a service requestor or a service registry.

Web Service Implementation:

Requirements Phase: The objective of the requirements phase is to understand the business requirement and translate them into the web services requirement. The requirement analyst should do requirement elicitation (it is the practice of researching and discovering the requirements of the system from the user, customer, and other stakeholders). The analyst should interpret, consolidate, and communicate these requirements to the development team. The requirements should be grouped in a centralized repository where they can be viewed, prioritized, and mined for interactive features.

Analysis Phase: The purpose of the analysis phase is to refine and translate the web service into conceptual models by which the technical development team can understand. It also defines the high-level structure and identifies the web service interface contracts.

Design Phase: In this phase, the detailed design of web services is done. The designers define web service interface contract that has been identified in the analysis phase. The defined web service interface contract identifies the elements and the corresponding data types as well as mode of interaction between web services and client.

Coding Phase: Coding and debugging phase is quite similar to other software component-based coding and debugging phase. The main difference lies in the creation of additional web service interface wrappers, generation of WSDL, and client stubs.

Test Phase: In this phase, the tester performs interoperability testing between the platform and the client's program. Testing to be conducted is to ensure that web services can bear the maximum load and stress. Other tasks like profiling of the web service application and inspection of the SOAP message should also perform in the test phase.

Deployment Phase: The purpose of the deployment phase is to ensure that the web service is properly deployed in the distributed system. It executes after the testing phase. The primary task of deployer is to ensure that the web service has been properly configured and managed. Other optional tasks like specifying and registering the web service with a UDDI registry also done in this phase.

Web Service Protocol Stack: A second option for viewing the web service architecture is to examine the emerging web service protocol stack. The stack is still evolving, but currently has four main layers.

Service Transport

This layer is responsible for transporting messages between applications. Currently, this layer includes Hyper Text Transport Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP), and newer protocols such as Blocks Extensible Exchange Protocol (BEEP).

XML Messaging

This layer is responsible for encoding messages in a common XML format so that messages can be understood at either end. Currently, this layer includes XML-RPC and SOAP.

Service Description

This layer is responsible for describing the public interface to a specific web service. Currently, service description is handled via the Web Service Description Language (WSDL).

Service Discovery

This layer is responsible for centralizing services into a common registry and providing easy publish/find functionality. Currently, service discovery is handled via Universal Description, Discovery, and Integration (UDDI).

Service transport is responsible for actually transporting XML messages between two computers.

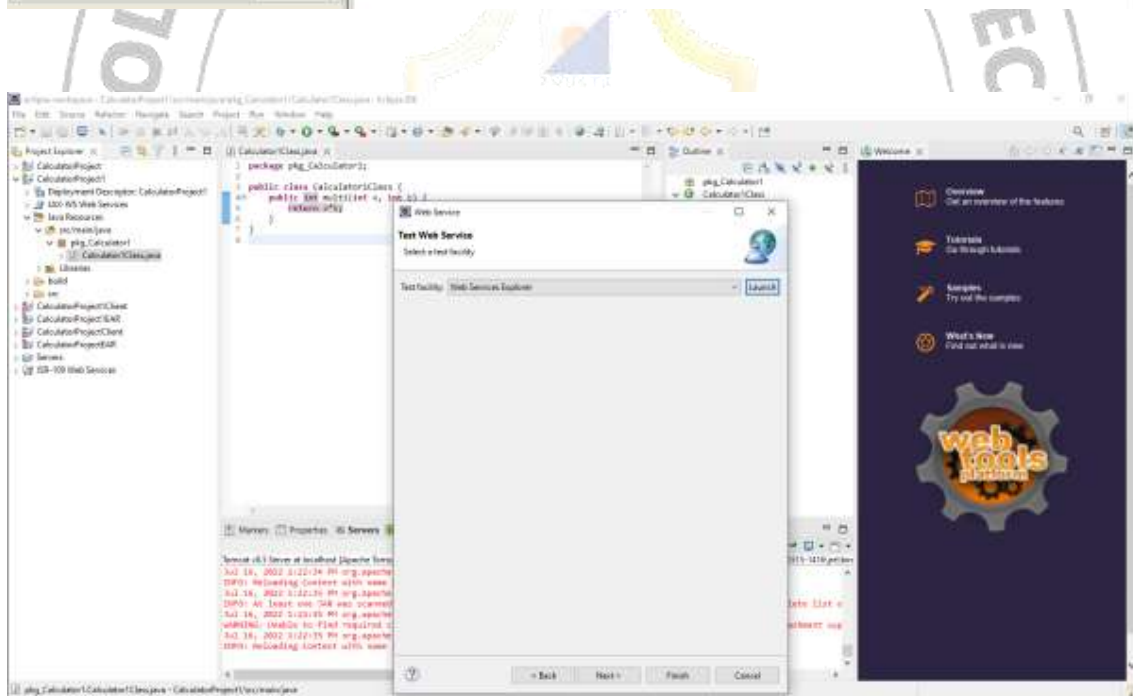
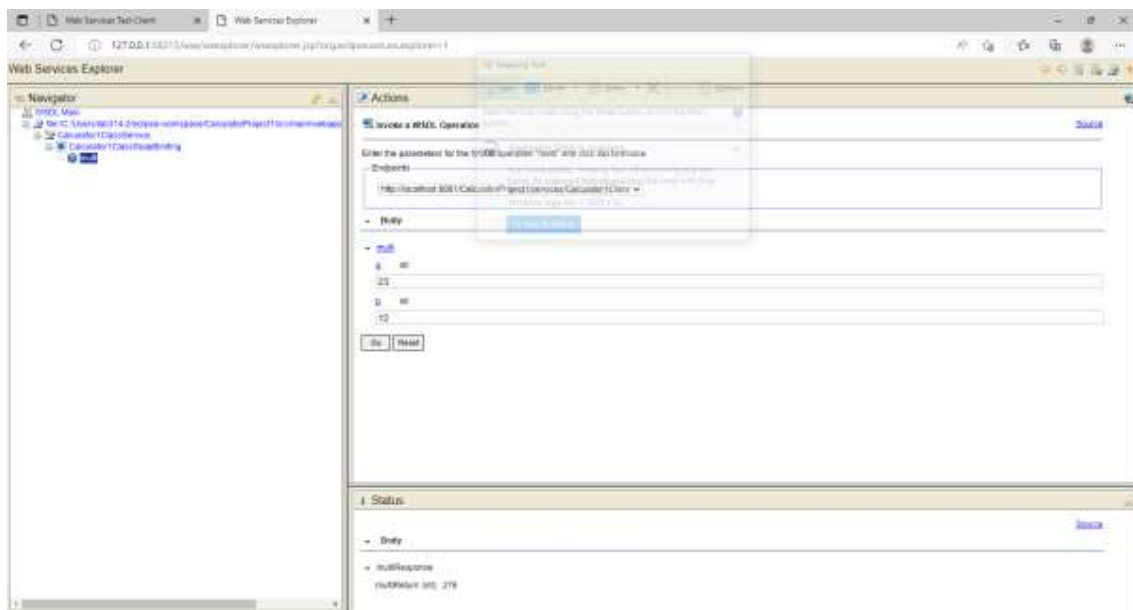
Hyper Text Transfer Protocol (HTTP)

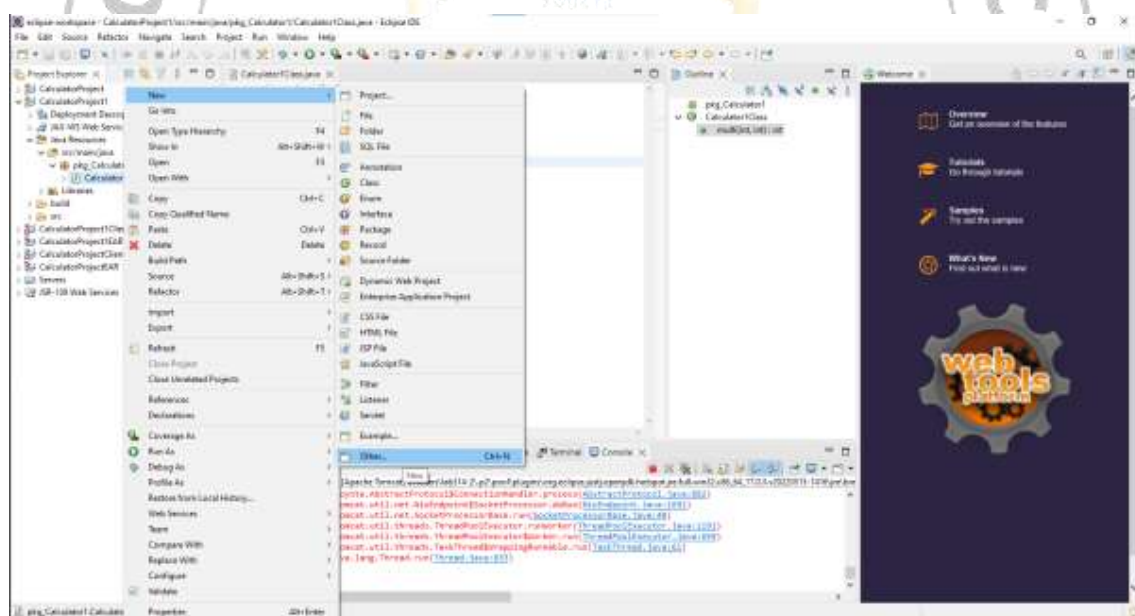
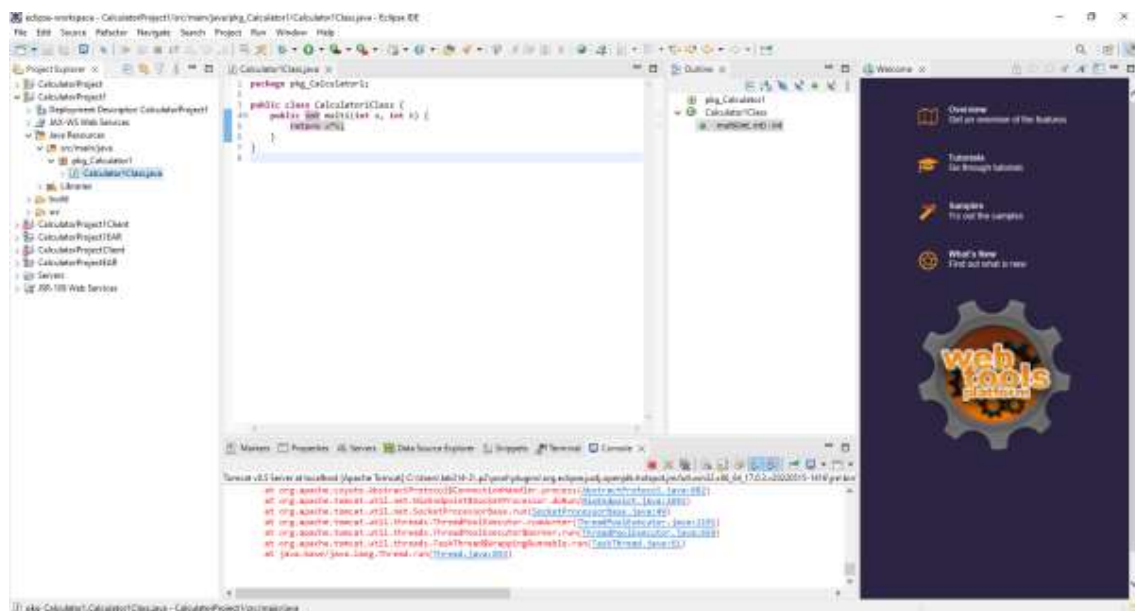
Currently, HTTP is the most popular option for service transport. HTTP is simple, stable, and widely deployed. Furthermore, most firewalls allow HTTP traffic. This allows XMLRPC or SOAP messages to masquerade as HTTP messages. This is good if you want to integrate remote applications, but it does raise a number of security concerns.

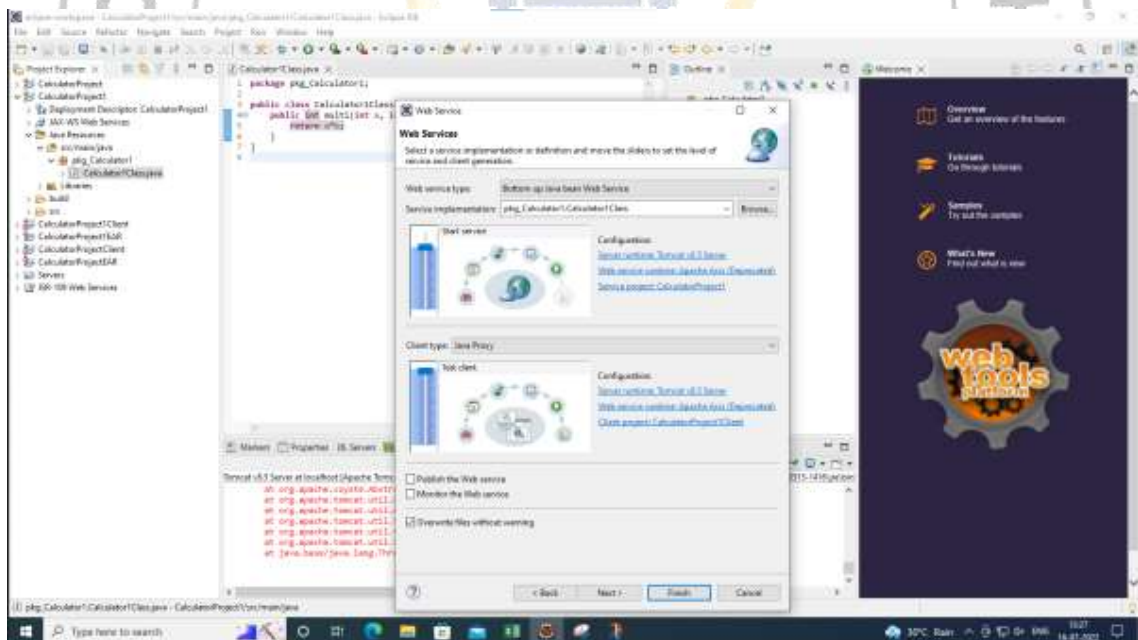
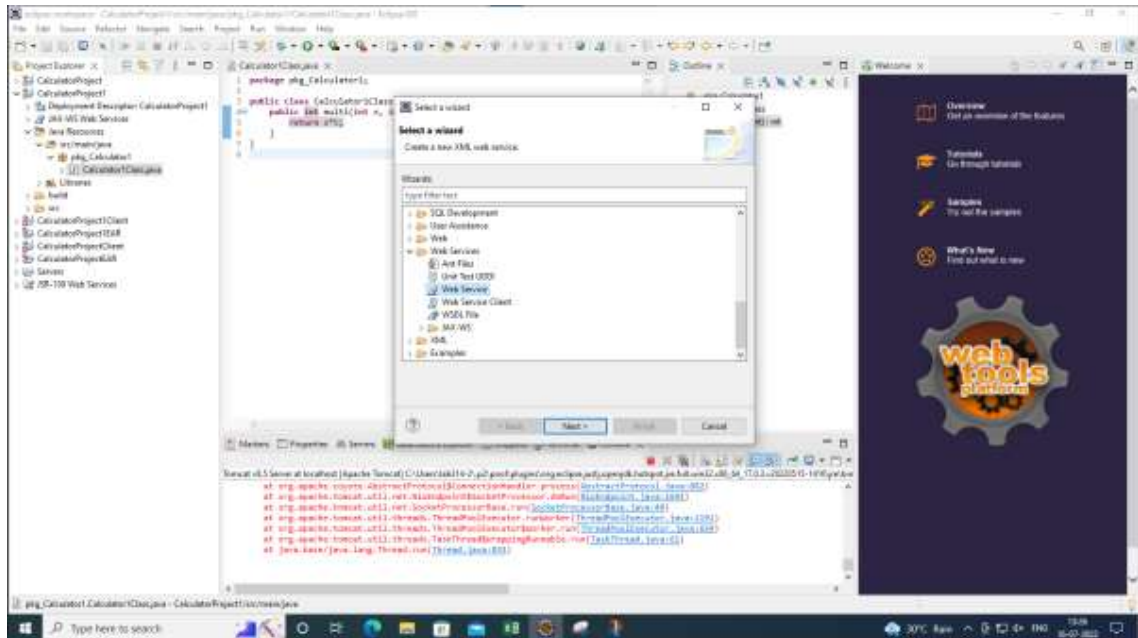
Blocks Extensible Exchange Protocol (BEEP)

This is a promising alternative to HTTP. BEEP is a new Internet Engineering Task Force (IETF) framework for building new protocols. BEEP is layered directly on TCP and includes a number of built-in features, including an initial handshake protocol, authentication, security, and error handling. Using BEEP, one can create new protocols for a variety of applications, including instant messaging, file transfer, content syndication, and network management. SOAP is not tied to any specific transport protocol. In fact, you can use SOAP via HTTP, SMTP, or FTP. One promising idea is therefore to use SOAP over BEEP.

Output:







Result and Discussion:

1. A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system.
2. Phases of Web Services
 - a. Requirements
 - b. Analysis
 - c. Design
 - d. Coding
 - e. Test
 - f. Deployment
3. A second option for viewing the web service architecture is to examine the emerging web service protocol stack.

Learning Outcomes: Students should have the ability to

LO1: Define Web services.

LO2: Identify different phases of web services.

LO3: Explain web service protocol.

Course Outcomes: Upon completion of the course students will be able to know about web services and its implementation.

Conclusion: In this Experiment, we have understood Web services, its artifacts and different phases.

Viva Questions:

1. Define web services.
2. Explain artifacts of web services.
3. Explain different phases in web service Implementation.
4. Explain Service Transport in web services.

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				

Experiment No.: 4 Integrate Software Components using middleware

Learning Objective: Student should be able to integrate software components using middleware

Tool:- Java RMI Theory:

Software system integration is essential where communication between different applications running on different platform is needed. Suppose a system designed for payroll running with Human Resource System. In that case employees' data need to be inserted in both systems. The system integration benefits a lot in these cases where data and services needed to be shared.

Web services are becoming very popular to share data between systems over the network and over the internet as well. In software industry the software integration carried same steps as software development and hence demands same kind of development procedures and testing.

This ensures the meaningful and clear communication between the systems. Systems integration becomes inevitable in Enterprise Systems where the whole organization needed to share data and services and give the feel to user as one system. The core purpose of integration is to make the systems communicate and also to make the whole system flexible and expandable.

The integration of different softwares written in different language and based on different platforms can be tricky. In that situation a middleware is necessary to enable the communication between different softwares. The middleware enables the software system not only to share data but also share the services.

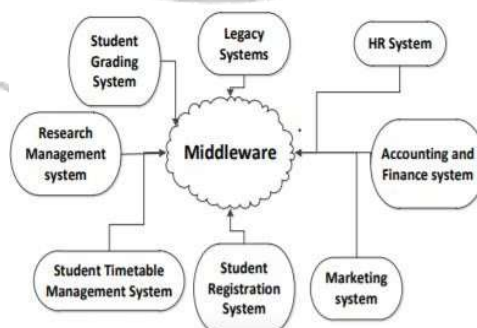


Figure 1: Middleware

A . Middleware

Independently written software systems need to be integrated in large system for example industry, institution, etc. These systems need to agree on common method for integration. To get them communicate there is need to have something in between them. The middle thing is termed commonly as middleware.

B .Service Oriented Architecture

Service Oriented Architecture (SOA) is the architectural design and pattern which is used to provide services to different applications. Its goal is to achieve loose coupling between interacting components of applications. Web Services, Corba, Jini, etc are the technologies used to implement SOA.

Main benefit of SOA is that it can provide the means of communication between completely different applications (built in different technologies). The services are also completely independent and reusable and the nature of reusability provide the less time to market. All the services technologies needs to be implemented using the SOA design pattern and need to be designed on the basis of SOA to get maximum benefit.

C .Web Services

Web service is the SOA technology with additional requirements of using internet protocols (HTTP, FTP,SMTP, etc.) and using XML (Extensible Markup Language) for message transmission. [7] These are application components which communicate between different applications using open protocol. Open protocol is the web protocol for querying and updating information. These components can be used by different kinds of applications to exchange information. HTML (HyperText Markup Language) and XML are the basics of web service implementation.

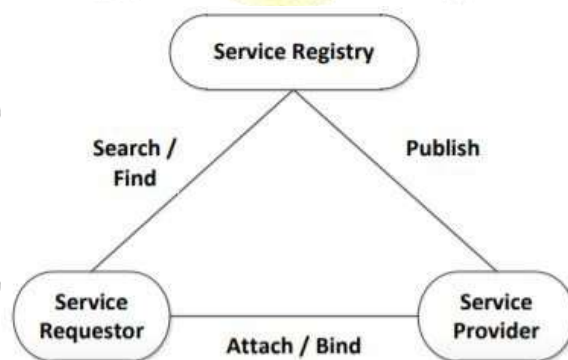


Figure 2: Web-Service Architecture

D. WSDL (Web Service Descriptive Language)

WSDL is a language to describe web services and providing the link to access these web services. It is acting as a publisher in web service architecture. It is the XML document which is recommended by W3C (World Wide Web Consortium) in 2007. In

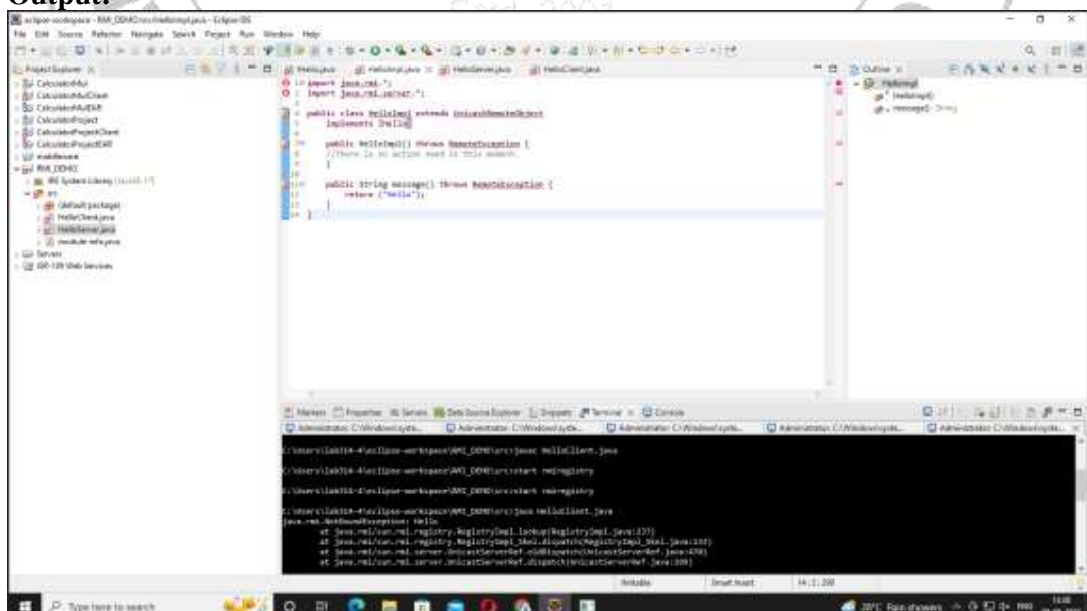
E. UDDI (Universal Description, Discovery and Integration)

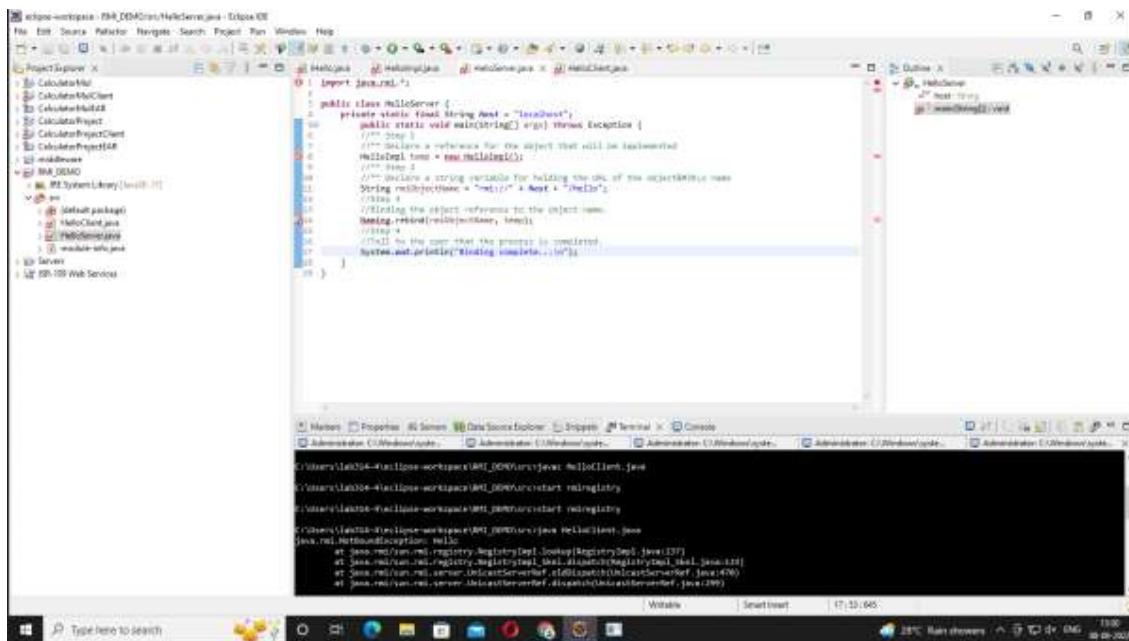
F. SOAP (Simple Object Access Protocol)

G. SOA - A solution to spaghetti architecture

In a fairly medium scale to large software architecture where there is need to integrate or communicate between different kinds of applications the introduction of links can make the architecture messy and it is called spaghetti architecture [9]. Figure 4 shows that problem in detail. It makes the system less flexible and expandability is the nightmare. Service Oriented Architecture (SOA) makes the architecture flexible and expandable.

Output:



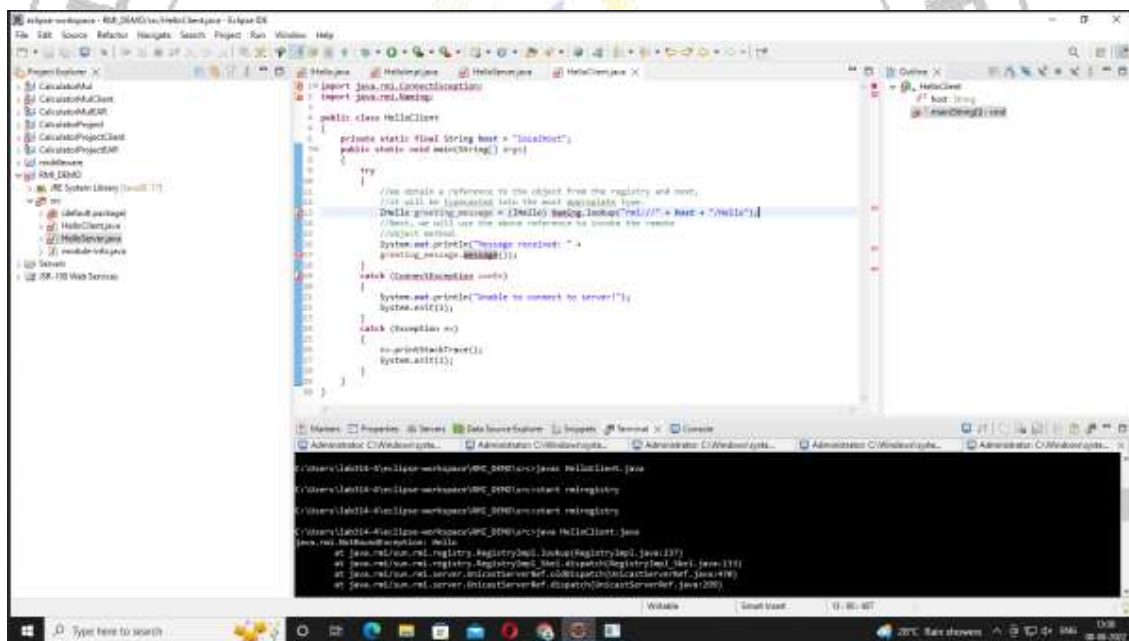


```

import java.rmi.*;

public class HelloServer {
    private static final String host = "localhost";

    public static void main(String[] args) throws Exception {
        // Step 1
        // Create a reference for the object that will be implemented
        HelloImpl h = new HelloImpl();
        // Step 2
        // Create a string variable for holding the url of the object's name
        String url = "rmi://" + host + "/Hello";
        // Step 3
        // Bind the object reference to the object name
        Naming.rebind(url, h);
        // Step 4
        // Tell the user that the process is completed
        System.out.println("Binding complete.");
    }
}
    
```



```

import java.rmi.*;
import java.rmi.Naming;

public class HelloClient {
    private static final String host = "localhost";

    public static void main(String[] args) {
        try {
            // Create a reference to the object from the registry and print.
            // (it will be implemented into the next application)
            Object greetingMessage = Naming.lookup("rmi://" + host + "/Hello");
            // Next, we will use the above reference to invoke the remote
            // object method.
            System.out.println("Message received: " +
                greetingMessage.toString());
        } catch (RemoteException ex) {
            System.out.println("Unable to connect to server!");
            System.exit(1);
        } catch (Exception ex) {
            ex.printStackTrace();
            System.exit(1);
        }
    }
}
    
```

Result and Discussion:

Learning Outcomes: Students should have be able to

LO1: Define Middleware.

LO2: Identify different components in middleware.

LO3: Explain Software Components using middleware.

Course Outcomes: Upon completion of the course students will be able to understandmiddleware and its components.

Conclusion:

Viva Questions:

1. Define Middleware.
2. Explain Service Oriented Architecture.
3. Explain Web Services.
4. Explain Universal Description, Discovery and Integration.

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				

Experiment No.: 5 Use middleware to implement connectors

Learning Objective: Student should be able to understand use of middleware to implement connectors.

Theory:

What is Middleware ?

Middleware is a more effective program that acts as bridge in between various applications and other databases otherwise tools. It is placed in between operating system and other applications which run on it. Middleware allows making better communication, application services, messaging, authentication, API management and management of data between different kinds of applications which help to exchange data.

The connectors sit between the two APIs or you can say and the ends of the connectors are APIs. The connectors receive data from one app/solution and process it to make it understandable and accessible in the other app/solution, regardless of whether any direct form of integration was available in the two apps.

Role of Middleware is :-

Middleware is a potentially useful tool when building software connectors. First, it can be used to bridge thread, process and network boundaries. Second, it can provide pre-built protocols for exchanging data among software components or connectors. Finally, some middleware packages include features of software connectors such as filtering, routing, and broadcast of messages or other data.

A signal interaction is a one-way interaction between an initiating object, called a client, and a responding object, called a server. An operation interaction is an interaction between a client object and server object that is either an interrogation or an announcement. An interrogation is composed of two one-way interactions: a request and a response. An announcement is a one-way request from a client object to a server object in which the client object expects no response, and the server object does not respond. A flow interaction is an ordered set of one or more one-way communications from a producer object to a consumer object. These interactions are a generalized

metamodel for describing communication styles between objects that can be implemented using a variety of middleware such as Remote Procedure Call (RPC) and Remote Method Invocation (RMI) and message queues (as selected and specified in the technology view).

can be implemented using a variety of middleware such as Remote Procedure Call (RPC) and Remote Method Invocation (RMI) and message queues.

Connectors as a primary vehicle for interprocess communication. A single conceptual connector can be “broken up” vertically (a) or horizontally (b) for this purpose.

Vertical Connectors :

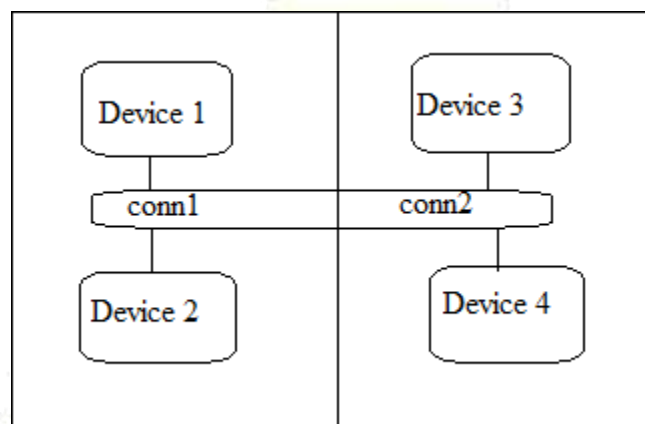


Figure (a)

Horizontal Connectors :

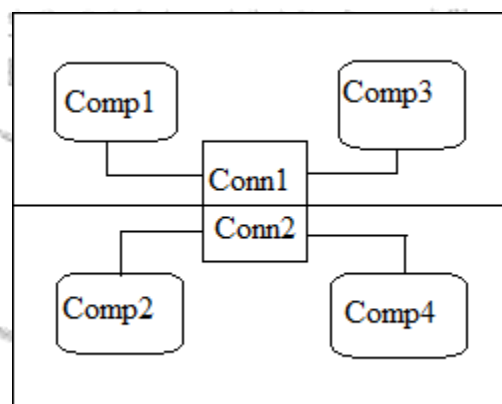


Figure (b)

Linking Ports Across Process Boundaries :

The ports can call methods on each other, sending messages as method parameters. Our intent was to simply use the middleware to exchange port references across process boundaries and use the existing technique for message passing.

the middleware technology would be entirely encapsulated within the port entity and would not be visible to architects or developers. The single process implementation of a C2 connector links two ports together by having each port contain a reference to the other one.

Linking Connectors Across Process Boundaries :

Sharing communication ports across process boundaries gave us fine-grained control over implementing an architecture as a multi-process application. However, it required additional functionality in the C2 implementation framework and did not isolate the change to the appropriate abstraction: the connector. In order to remedy this, we devised two connector-based approaches. Both of these approaches consist of implementing a single conceptual software connector using two or more actual connectors that are linked across process or network boundaries. Each actual connector thus becomes a segment of a single “virtual connector.” All access to the underlying middleware technology is encapsulated entirely within the abstraction of a connector, meaning that it is unseen by both architects and developers. We call the first approach “lateral welding,” depicted in Fig. 2a. Messages sent to any segment of the multi-process connector are broadcast to all other segments. Upon receiving a message, each segment has the responsibility of filtering and forwarding it to components in its process as appropriate. Only messages are sent across process boundaries. While the lateral welding approach allowed us to “vertically slice” a C2 application, we also developed an approach to “horizontally slice” an application, as shown in Fig. 2b. This approach is similar to the idea of lateral welding: a conceptual connector is broken up into top and bottom segments, each of which exhibits the same properties as a single-process connector to the components attached above and below it, respectively. However, the segments themselves are joined using the appropriate middleware. When used with a middleware technology that supports dynamic change at run-time, all of these approaches, both using ports and connectors, can be used to build applications where processes can join and leave a running application.

Using Middleware Technologies :

To explore the use of OTS middleware with software connectors, we chose four representative technologies from the field of available middleware packages. These were Q, an RPC system, Polyolith, a message bus, RMI, a connection mechanism for Java objects, and ILU, a distributed objects package. A description of one of our efforts involving integrating two middleware technologies simultaneously in the same application is given here. With each middleware package, we were able to encapsulate all the middleware functionality within the connectors

themselves. This means that architects and developers can use the middleware-enhanced connectors thus created just as they would use normal, in-process C2 connectors.

Simultaneous Use of Multiple Middleware Packages :

Each middleware technology we evaluated has unique benefits. By combining multiple such technologies in a single application, the application can potentially obtain the benefits of all of them. For instance, a middleware technology that supports multiple platforms but only a single language, such as RMI, could be combined with one that supports multiple languages but a single platform, such as Q, to create an application that supports both multiple languages and multiple platforms. The advantages of combining multiple middleware technologies within software connectors are manifold. In the absence of a single panacea solution that supports all required platforms, languages, and network protocols, the ability to leverage the capabilities of several different middleware technologies significantly widens the range of applications that can be implemented within an architectural style such as C2. We combined our implementations of ILU-C2 and RMI-C2 connectors in a version of the KLAX application, a real time video game application built as an experimental platform for work on the C2 architecture. We were able to do so with no modification to the middleware-enhanced C2 framework or the connectors themselves by combining the lateral welding technique shown in Fig. 2a with the horizontal slicing technique shown in Fig. 2b. This approach works for any combination of OTS connectors that use the lateral welding technique. An alternative approach would have been to create a single connector that supported both ILU and RMI, but this would have required changes to the framework.

OUTPUTS:

Estd. 2001

ISO 9001 : 2015 Certified
NBA and NAAC Accredited

eclipse-workspace - Middleware/src/ClientClass.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- CalculatorProject
 - CalculatorProject1
 - CalculatorProject1Client
 - CalculatorProject1EAR
 - CalculatorProjectClient
 - CalculatorProjectEAR
- Middleware
 - IRE System Library [JavaSE-17]
 - src
 - (default package)
 - ClientClass.java
 - InterfaceClass.java
 - ServerClass.java
 - Middleware2
 - IRE System Library [JavaSE-17]
 - src
 - (default package)
 - ClientClass.java
 - InterfaceClass.java
 - ServerClass.java
 - Servers

ServerClass.java

```

1
2 public class ClientClass {
3
4
5 public static void main(String[] args) {
6     // 1000 Auto-generated method stub
7     InterfaceClass c = new ServerClass();
8     System.out.println(c.add(11,9));
9
10 }
11
12 }
13
  
```

InterfaceClass.java

```

1
2
3
4
5
6
7
8
9
10
11
12
13
  
```

ClientClass.java

```

1
2 public class ClientClass {
3
4
5 public static void main(String[] args) {
6     // 1000 Auto-generated method stub
7     InterfaceClass c = new ServerClass();
8     System.out.println(c.add(11,9));
9
10 }
11
12 }
13
  
```

ClientClass.java

```

1
2 public class ClientClass {
3
4
5 public static void main(String[] args) {
6     // 1000 Auto-generated method stub
7     InterfaceClass c = new ServerClass();
8     System.out.println(c.add(11,9));
9
10 }
11
12 }
13
  
```

Outline

- ClientClass
 - main(String[]): void

Problems Javadoc Declaration Console

<terminated> ClientClass [Java Application] C:\Users\lab314-2\p2\poo\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\jav

eclipse-workspace - Middleware/src/ClientClass.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- CalculatorProject
 - CalculatorProject1
 - CalculatorProject1Client
 - CalculatorProject1EAR
 - CalculatorProjectClient
 - CalculatorProjectEAR
- Middleware
 - IRE System Library [JavaSE-17]
 - src
 - (default package)
 - ClientClass.java
 - InterfaceClass.java
 - ServerClass.java
 - Middleware2
 - IRE System Library [JavaSE-17]
 - src
 - (default package)
 - ClientClass.java
 - InterfaceClass.java
 - ServerClass.java
 - Servers

ServerClass.java

```

1
2 public class ClientClass {
3
4
5 public static void main(String[] args) {
6     // 1000 Auto-generated method stub
7     InterfaceClass c = new ServerClass();
8     System.out.println(c.add(11,9));
9
10 }
11
12 }
13
  
```

InterfaceClass.java

```

1
2
3
4
5
6
7
8
9
10
11
12
13
  
```

ClientClass.java

```

1
2 public class ClientClass {
3
4
5 public static void main(String[] args) {
6     // 1000 Auto-generated method stub
7     InterfaceClass c = new ServerClass();
8     System.out.println(c.add(11,9));
9
10 }
11
12 }
13
  
```

ClientClass.java

```

1
2 public class ClientClass {
3
4
5 public static void main(String[] args) {
6     // 1000 Auto-generated method stub
7     InterfaceClass c = new ServerClass();
8     System.out.println(c.add(11,9));
9
10 }
11
12 }
13
  
```

Outline

- ClientClass
 - main(String[]): void

Problems Javadoc Declaration Console

<terminated> ClientClass [Java Application] C:\Users\lab314-2\p2\poo\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416\jre\bin\jav

.eclipse-workspace - Middleware/src/ClientClass.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- CalculatorProject
 - CalculatorProjectClient
 - CalculatorProjectIAR
 - CalculatorProjectClient
 - CalculatorProjectIAR
- Middleware
 - IRE System Library [JavaSE-17]
 - src
 - (default package)
 - ClientClass.java
 - InterfaceClass.java
 - ServerClass.java
 - Middleware2
 - IRE System Library [JavaSE-17]
 - src
 - (default package)
 - ClientClass.java
 - InterfaceClass.java
 - ServerClass.java
 - Servers



Estd. 2001

ISO 9001 : 2015 Certified
 NBA and NAAC Accredited

TECHNOLOGY

eclipse-workspace - Middleware/src/ServerClass.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer: CalculatorProject, CalculatorProjectClient, CalculatorProjectTEAR, CalculatorProjectClient, CalculatorProjectTEAR, Middleware, JRE System Library [Javac11], src (default package), ClientClass.java, InterfaceClass.java, ServerClass.java, Servers, JSR-100 Web Services

```

1 public class ServerClass implements InterfaceClass {
2
3
4 //public static void main(String[] args) {
5 // TODO Auto-generated method stub
6
7 //}
8
9 public int add(int a,int b) {
10     return a+b;
11 }
12 public int sub(int a, int b) {
13     return a - b;
14 }
15 public int mul(int a, int b) {
16     return a * b;
17 }
18 public float div(int a, int b) {
19     return a / b;
20 }
21
22 }
23
24
  
```

Outline: ServerClass, add(int, int): int, sub(int, int): int, mul(int, int): int, div(int, int): float

Markers Properties JLS Servers Data Source Explorer Snippets Terminal Console

<terminated> ClientClass [Java Application] C:\Users\lab314\2\p2\poo\plugins\org.eclipse.jdt.openjdk.hotspot.jre.full.win32.x86_64_17.0.3\j20220515-1418\jre\bin\java.exe
 20

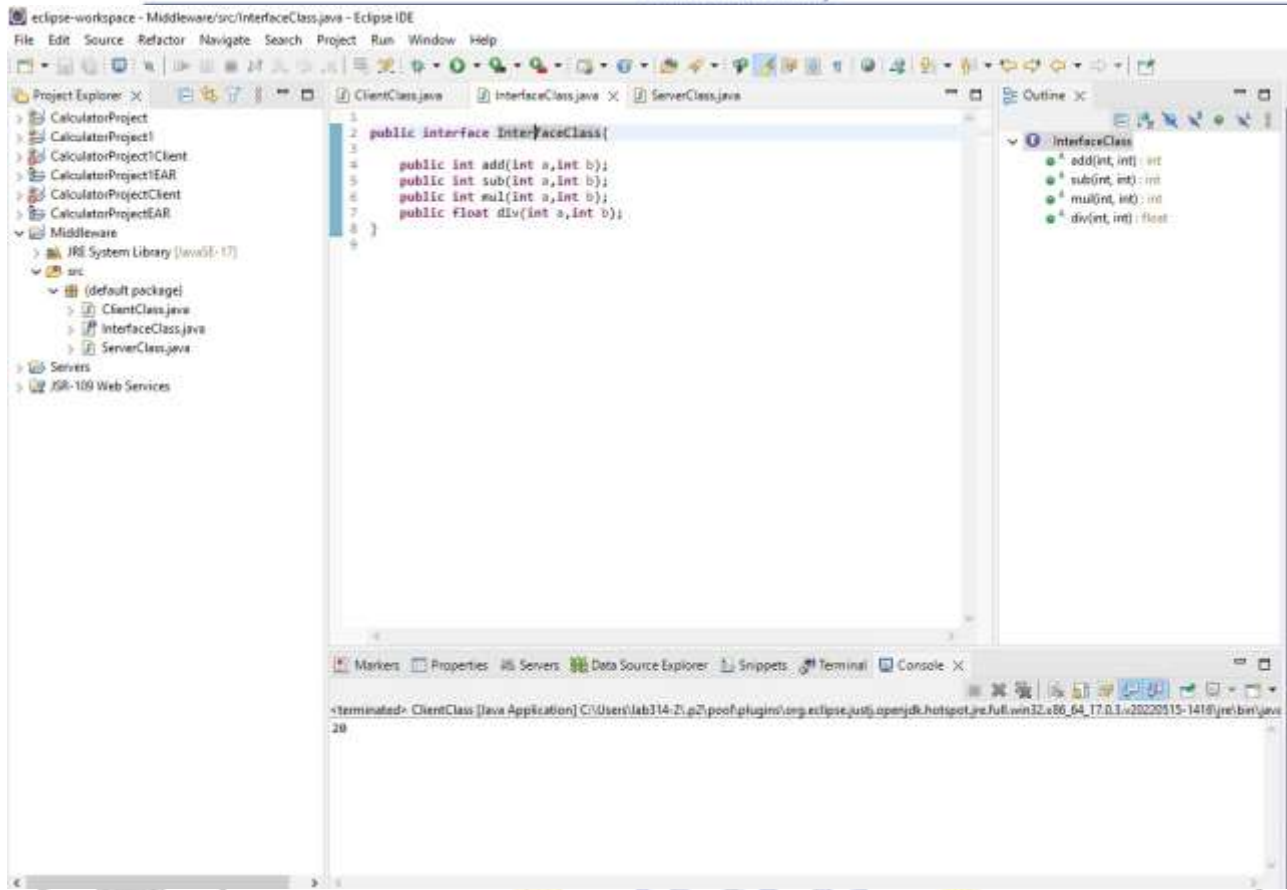
Writable Smart Insert 4:7:6



Estd. 2001

ISO 9001 : 2015 Certified
 NBA and NAAC Accredited

TECHNOLOGY



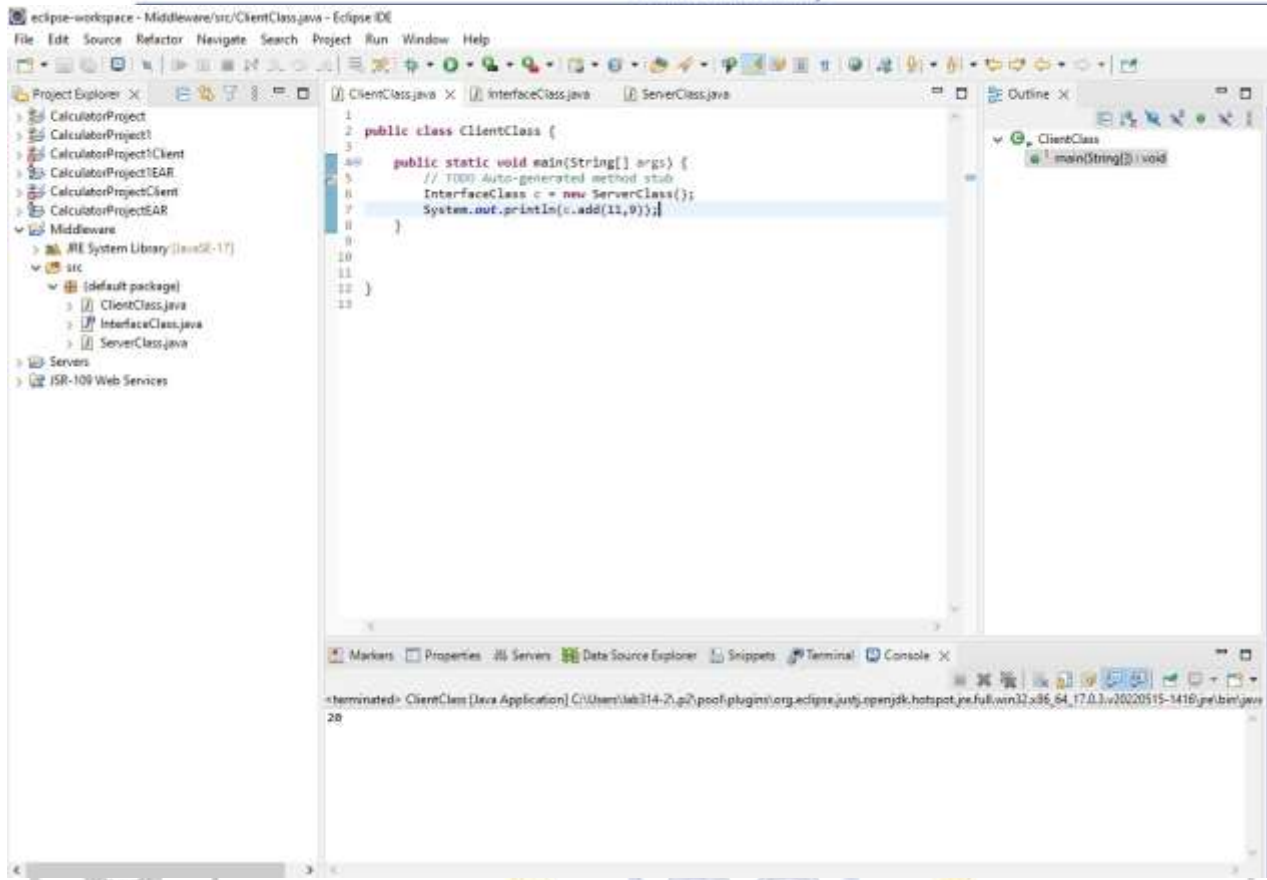
THAKKUR



Estd. 2001

ISO 9001 : 2015 Certified
 NBA and NAAC Accredited

TECHNOLOGY



Result and Discussion:

Learning Outcomes: Students should have be able to

- LO1: Define Middleware.
- LO2: Identify different connectors in middleware.
- LO3: Explain middleware implements in connectors.

Course Outcomes: Upon completion of the course students will be able to understand middleware and its connectors.

Conclusion:

Viva Questions:

1. Define Middleware.
2. Explain use of middleware.
3. Explain implementation of connectors.

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				



Estd. 2001

ISO 9001 : 2015 Certified
 NBA and NAAC Accredited

Experiment No.: 7

Identifying Design requirements for an Architecture for any specific domain.

Learning Objective: Student should be able to understand Design requirements for an Architecture for any specific domain.

Theory:

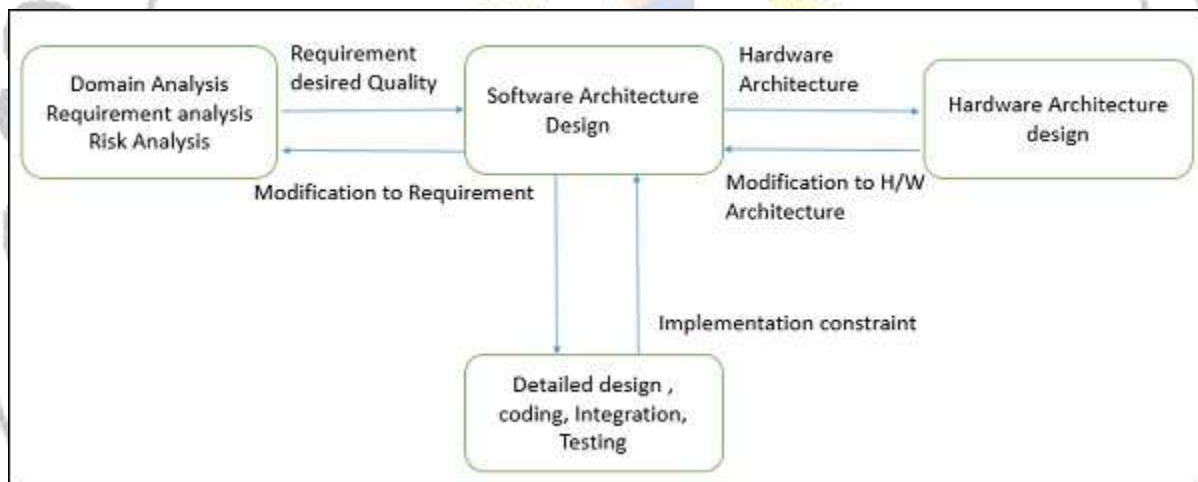
Software design:

Software design provides a design plan that describes the elements of a system, how they fit, and work together to fulfill the requirement of the system. The objectives of having a design plan are as follows –

To negotiate system requirements, and to set expectations with customers, marketing, and management personnel.

Act as a blueprint during the development process.

Guide the implementation tasks, including detailed design, coding, integration, and testing.



It comes before the detailed design, coding, integration, and testing and after the domain analysis, requirements analysis, and risk analysis.

Software design is responsible for the code level design such as, what each module is doing, the classes scope, and the functions purposes, etc. When used strategically, they can make a programmer significantly more efficient by allowing them to avoid reinventing the wheel, instead using methods refined by others already. They also provide a useful common language to conceptualize repeated problems and solutions when discussing with others or managing code in larger teams.

Major artifacts of the software design process include:

- **Software requirements specification.** This document describes the expected behavior of the system in the form of functional and non-functional requirements. These requirements should be clear, actionable, measurable, and traceable to business requirements. Requirements should also define how the software should interact with humans, hardware, and other systems.
- **High-level design.** The high-level design breaks the system's architectural design into a less-abstracted view of sub-systems and modules and depicts their interaction with each other. This high-level design perspective focuses on how the system, along with all its components, implements in the form of modules. It recognizes the modular structure of each sub-system and their interaction among one another.
- **Detailed design.** Detailed design involves the implementation of what is visible as a system and its sub-systems in a high-level design. This activity is more detailed towards modules and their implementations. It defines a logical structure of each module and their interfaces to communicate with other modules.

The purpose of an architecture schema or design record is to serve as a vehicle for software understanding by functioning as a collection point for knowledge about the components that make up a DSSA. In particular, the design record organizes

- domain- specific knowledge about components or design alternatives and
- Implementation in knowledge about alternate implementations,

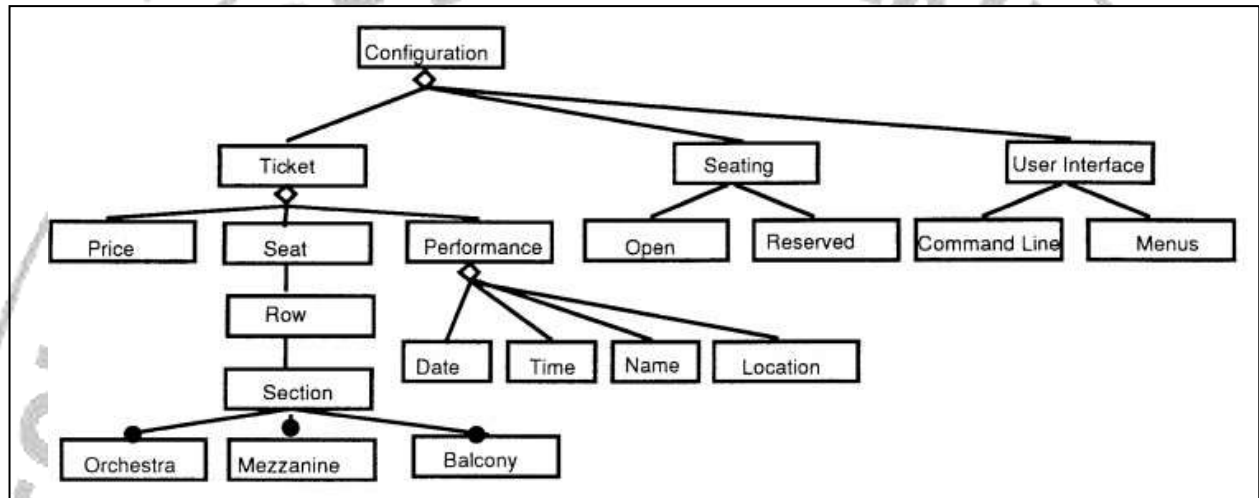
The primary goal of a design record is to adequately describe the components in a reference architecture such that the application engineer can make design decisions and component selections without looking at implementations. The secondary goal of a design record is to provide information that the tools in the supporting environment can use.

The design record data elements used by Loral Federal Systems

phases in the software life cycle, include:

1. Name/type
2. Description
3. reference requirements satisfied,
4. design structure (data flow and control flow diagrams),
5. design rationale,
6. interface and architecture specifications and dependencies,
7. P D L (program Design Language) text,
8. implementation,

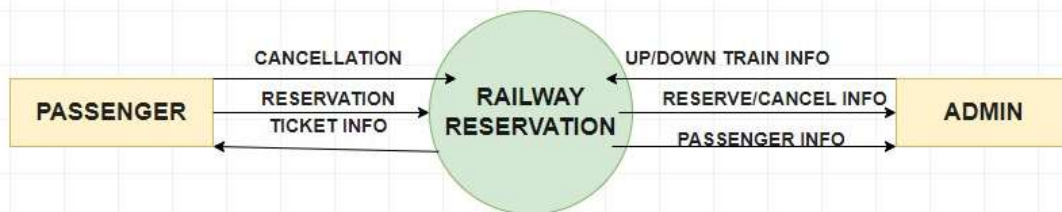
10. test cases.
11. metric data,
12. access rights,
13. search points,
14. catalog information,
15. library and DSSA links, and hypertext paths



Many different programming formats incorporate the same essential elements. In all cases, the design programming fits within a larger context of planning efforts which can also be programmed. For design programming for a building, we propose a six-step process as follows:

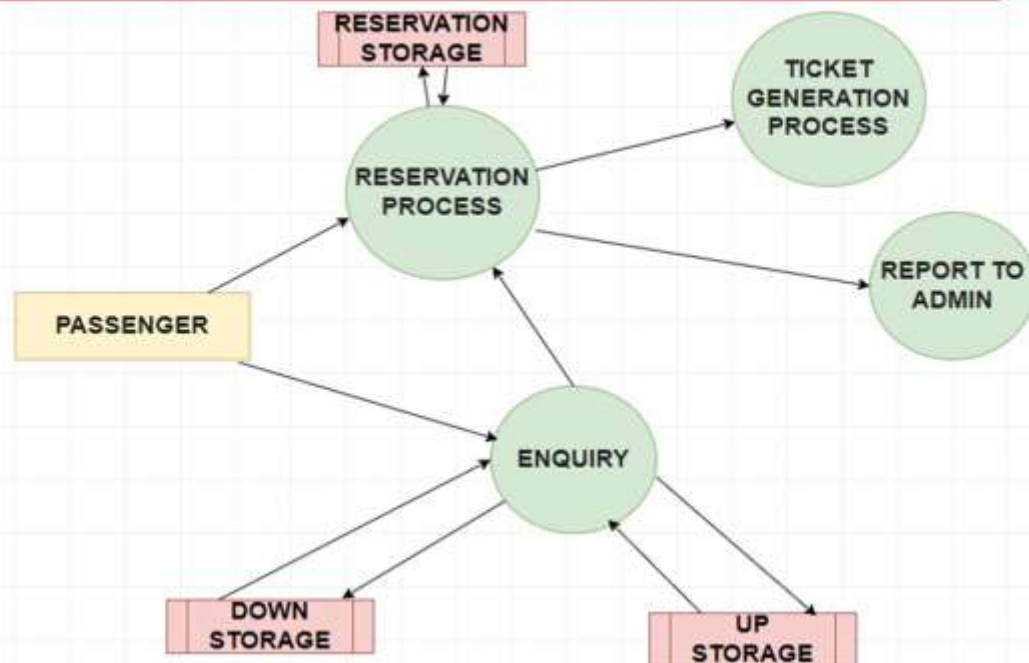
1. Research the project type
2. Establish goals and objectives
3. Gather relevant information
4. Identify strategies
5. Determine quantitative requirements
6. Summarize the program

RAILWAY RESERVATION SYSTEM 0-LEVEL DFD



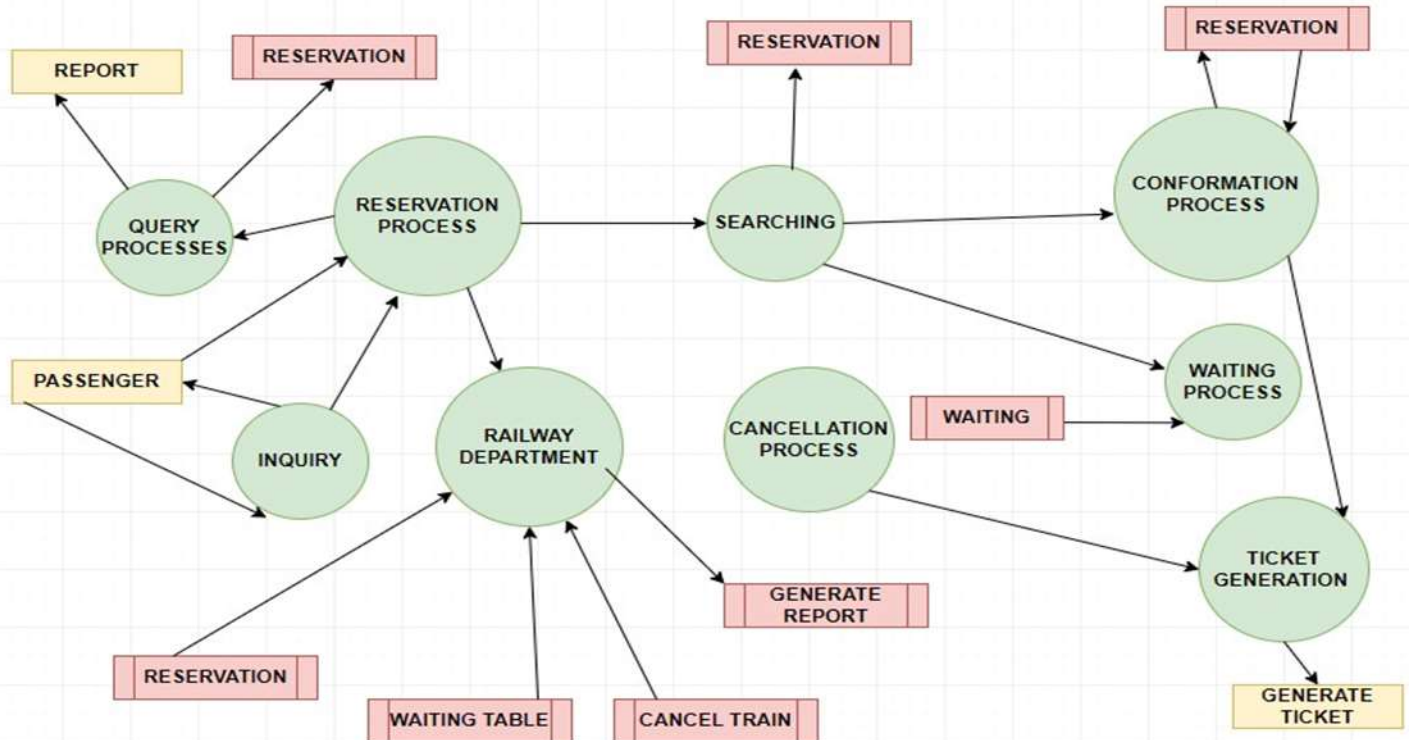
Context Diagram / Level – 0 DFD

RAILWAY RESERVATION SYSTEM 1-LEVEL DFD



1st Level

RAILWAY RESERVATION SYSTEM 2-LEVEL DFD



Result and Discussion:

Estd. 2001

ISO 9001 : 2015 Certified
NBA and NAAC Accredited

Learning Outcomes: Students should have been able to understand

LO1: Define software design.

LO2: Identify different design requirements for software.

LO3: Explain implementation of design requirements of software.

Course Outcomes: Upon completion of the course students will be able to understand design requirement of software Architecture.

Conclusion:

Viva Questions:

1. Define design requirement in software.
2. Explain different design requirements in software Architecture.
3. Explain the phases of programming design.
4. Explain any three design requirements.

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	Certified and NBA Accredited
Marks Obtained				