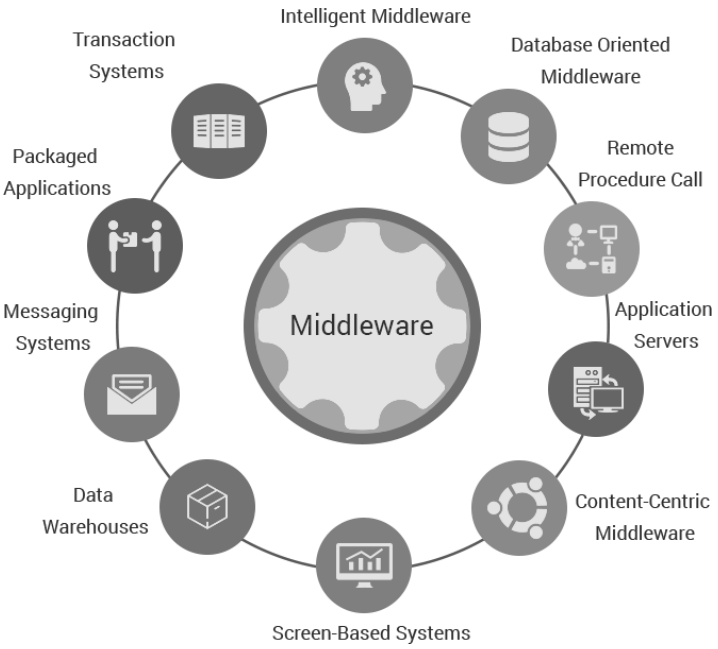
INFORMATION TECHNOLOGY DEPARTMENT

MIDDLEWARE TECHNOLOGIES LAB MANUAL



1. Institution Vision

To be part of universal human quest for development and progress by contributing high calibre, ethical and socially responsible engineers who meet the global challenge of building modern society in harmony with nature.

2. Institution Mission

- To attain excellence in imparting technical education from the undergraduate through doctorate levels by adopting coherent and judiciously coordinated curricular and co-curricular programs
- To foster partnership with industry and government agencies through collaborative research and consultancy
- To nurture and strengthen auxiliary soft skills for overall development and improved employability in a multi-cultural work space
- To develop scientific temper and spirit of enquiry in order to harness the latent innovative talents
- To develop constructive attitude in students towards the task of nation building and empower them to become future leaders
- To nourish the entrepreneurial instincts of the students and hone their business acumen.
- To involve the students and the faculty in solving local community problems through economical and sustainable solutions.

3. Department vision

Fostering a bright technological future by enabling the students to function as leaders in software industry and serve as means of transformation to empower society through ITeS.

4. Department Mission

To create an ambience of academic excellence through state of art infrastructure and learner-centric pedagogy leading to employability in multi-disciplinary fields.

5. Program Educational Objectives

The Program Educational Objectives of Information Technology Program are as follows:

1. Graduates will demonstrate technical competence and leadership in their chosen fields of employment by identifying, formulating, analyzing and creating efficient IT solutions.
2. Graduates will communicate effectively as individuals or team members and be successful in varied working environment.
3. Graduates will demonstrate lifelong learning through continuing education and professional development.
4. Graduates will be successful in providing viable and sustainable solutions within societal, professional, environmental and ethical context.

6. Program Outcomes

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO 12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

7. Program Specific Outcomes

PSO1: Work as Software Engineers for providing solutions to real world problems using Structured, Object Oriented Programming languages and open source software.

PSO2: Function as Systems Engineer, Software Analyst and Tester for IT and ITeS.

8. Introduction TO MWT Laboratory

MWT: Middle-ware is computer software that connects software components or applications. The software consists of a set of services that allows multiple processes running on one or more machines to interact. This technology evolved to provide for interoperability in support of the move to coherent distributed architectures, which are used most often to support and simplify complex, distributed applications. It includes web servers, application servers, and similar tools that support application development and delivery. Middle-ware is especially integral to modern information technology based on XML, SOAP, Web services, and service-oriented architecture. Middle-ware sits "in the middle" between application software that may be working on different operating systems. It is similar to the middle layer of three-tier single system architecture, except that it is stretched across multiple systems or applications. Examples include EAI software, telecommunications software, transaction monitors, and messaging and- queuing software.

Middle-ware sits "in the middle" between application software that may be working on different operating systems. It is similar to the middle layer of three-tier single system architecture, except that it is stretched across multiple systems or applications. Examples include EAI software, telecommunications software, transaction monitors, and messaging and- queuing software.

Definitions

It's Software that provides a link between separate software applications. Middleware is sometimes called plumbing because it connects two applications and passes data between them. Middle-ware allows data contained in one database to be accessed through another. This definition would fit enterprise application integration and data integration
software.

ObjectWeb defines Middle-ware as: "The software layer that lies between the operating system and applications on each side of a distributed computing system in a network."

Origins

Middle-ware is a relatively new addition to the computing landscape. It gained popularity in the 1980s as a solution to the problem of how to link newer applications to older legacy systems, although the term had been in use since 1968. It also facilitated distributed processing, the connection of multiple applications to create a larger application, usually over a network.

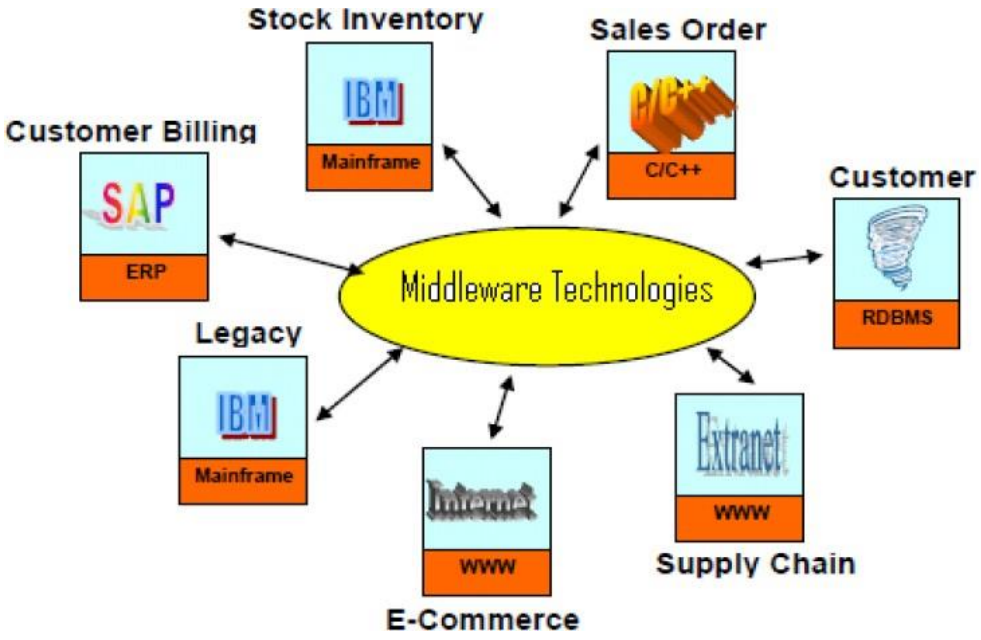
Organizations

IBM, Red Hat, and Oracle Corporation are major vendors providing Middle-ware software. Vendors such as Axway, SAP, TIBCO, Informatica, Pervasive and web Methods were specifically founded to provide Web-oriented Middle-ware tools. Groups such as the Apache Software Foundation and the Object Web Consortium encourage the development of open source Middle-ware. Microsoft .NET "Framework" architecture is essentially "Middle-ware" with typical Middle-ware functions distributed between the various products, with most inter-computer interaction by industry standards, open API's or RAND software license.

Use of Middle-ware

Middle-ware services provide a more functional set of application programming interfaces to allow an application to:

- Locate transparently across the network, thus providing interaction with another service or application.
- Be independent from network services.
- Be reliable and always available.



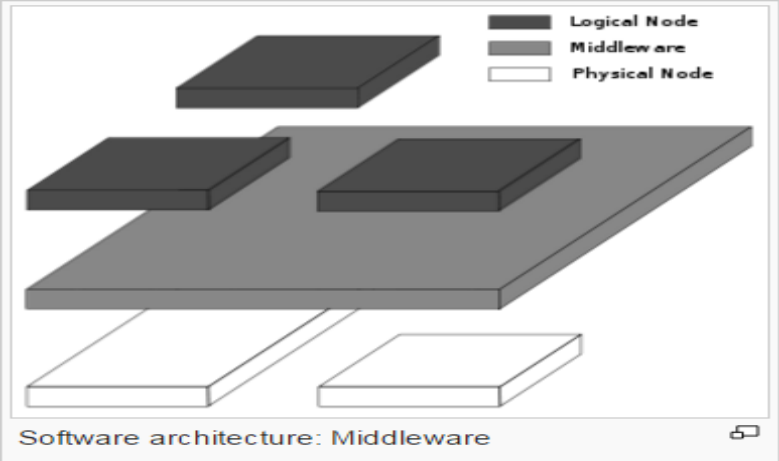
Laboratory Objective

Upon successful completion of this Lab the student will be able to:

- Understand the basic structure of distributed systems.
- Understand the motivation of using middleware.
- Understand the basic concepts underlying the ASP.net and C#.net.
- Learn to make judgment in choosing a suitable middleware for application problems.
- Understand the basic concepts of Web Services and EJB.

Overview of MWT

Middleware is computer software that provides services to software applications beyond those available from the operating system. It can be described as "software glue". Middleware makes it easier for software developers to implement communication and input/output, so they can focus on the specific purpose of their application.



Middleware in distributed applications

The term is most commonly used for software that enables communication and management of data in distributed applications. An IETF workshop in 2000 defined middleware as "those services found above the transport (i.e., over TCP/IP) layer set of services but below the application environment (i.e., below application-level APIs). In this more specific sense middleware can be described as the dash ("-") in client-server or the -to- in peer-to-peer. Middleware includes web servers, application servers, content management systems, and similar tools that support application development and delivery.

Object Web defines middleware as: "The software layer that lies between the operating system and applications on each side of a distributed computing system in a network." Services that can be regarded as middleware include enterprise application integration, data integration, message oriented middleware (MOM), object request brokers (ORBs), and the enterprise service bus (ESB).

Database access services are often characterized as middleware. Some of them are language specific implementations and support heterogeneous features and other related communication features. Examples of database-oriented middleware include ODBC, JDBC and transaction processing monitors.

Distributed computing system middleware can loosely be divided into two categories – those that provide human-time services (such as web request servicing) and those that perform in machine-time. This latter middleware is somewhat standardized through the Service Availability Forum and is commonly used in complex, embedded systems within telecom, defense and aerospace industries.

System Configuration

Hardware Configuration

Processor	Pentium IV
RAM	1GB

Software Configuration

Languages	:	Asp .Net, C#.Net and Java.
Database	:	MS-SQL Server 2005/2008, MSACCESS
IDE	:	Visual Studio 2010, Net Beans

Program 1
REMOTE METHOD INVOCATION

Problem Definition
Create a distributed name server (like DNS) RMI.

Problem Description

Overview
Java Remote Method Invocation (RMI) allows writing distributed objects using Java. RMI provides a simple and direct model for distributed computation with Java objects. Because RMI is centered on Java; it brings the power of Java safety and portability to distributed computing. RMI is Java's remote procedure call (RPC) mechanism.

RMI connects to existing and legacy systems using the standard Java native method interface JNI. RMI can also connect to existing relational database using the standard JDBC. The RMI/JNI and RMI/JDBC combinations let you use RMI to communicate today with existing servers in non-Java languages.

Advantages
RMI features:
· Object Oriented.
· Mobile Behavior.
· Design Patterns.
· Safe and Secure.
· Easy to Write/Easy to Use.
· Connects to Existing/Legacy Systems.
· Write Once, Run Anywhere.
· Distributed Garbage Collection.
· Parallel Computing.
· The Java Distributed Computing Solution.

Architecture
The architecture is designed to allow for future expansion of server and reference types so that RMI can add features in a coherent way.

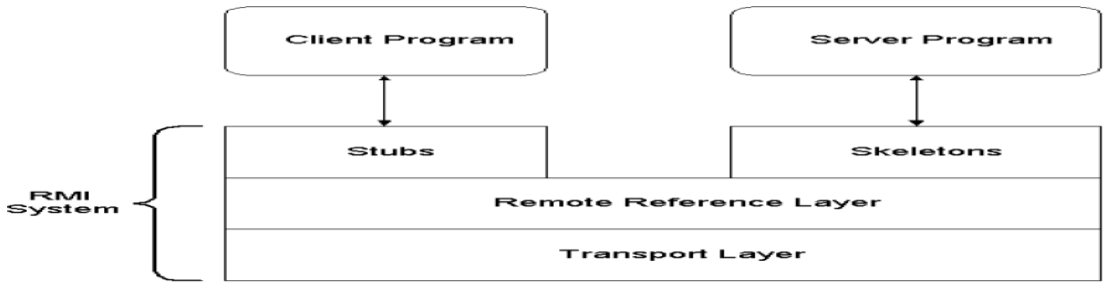


Figure: RMI Architectural Layers

When a client receives a reference to a server, RMI downloads a stub that translates calls on that reference into remote calls to the server. The stub marshals the arguments to the method using object serialization, and sends the marshalled Invocation across the wire to the server.

On the server side the call is received by the RMI system and connected to a skeleton, which is responsible for un-marshalling the arguments and invoking the server's implementation of the method. When the server's implementation completes, either by returning a value or by throwing an exception, the skeleton marshals the result and sends a reply to the client's stub.

The stub un-marshals the reply and either returns the value or throws the exception as appropriate. Stubs and skeletons are generated from the server implementation, usually using the program rmic. Stubs use references to talk to the skeleton. This architecture allows the reference to define the behavior of communication.

Implementation

Steps for Developing an RMI System

1. Define the remote interface
2. Develop the remote object by implementing the remote interface.
3. Develop the client program.
4. Compile the Java source files.
5. Generate the client stubs and server skeletons i.e. rmic implementation_filename.
6. Start the RMI registry using i.e. start rmiregistry.
7. Start the remote server objects i.e. java server_filename.
8. Run the client i.e. java client_filename.

Program

Hello.java

```
import java.rmi.*;
public interface dnsinter extends Remote
{       public String showip(String ws) throws Exception;
  }
```

dnsimp.java

```
import java.rmi.*;
import java.rmi.server.*;
import java.sql.*;
public class dnsimp extends UnicastRemoteObject implements dnsinter
{
      Connection con;
      PreparedStatement pt;
      ResultSet rs;
         String ip;

      public dnsimp() throws Exception
      {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            con=DriverManager.getConnection("Jdbc:Odbc:Bank");
```

```
	}

	public String showip(String ws) throws Exception
	{
		pt=con.prepareStatement("select ip from dns where ws=?");
		pt.setString(1,ws);
			rs=pt.executeQuery();
			while(rs.next())
		{
			ip =rs.getString("ip");
		}
		return ip;
	}
}
```

dnsServer.java

```
import java.rmi.*;
public class dnsServer
{
	public static void main(String arg[]) throws Exception
	{
		dnsimp bi=new dnsimp();
		Naming.rebind("bankobj",bi);
	}
}
```

dnsClient.java

```
import java.io.*;
import java.rmi.*;
public class dnsClient
{
	public static void main(String arg[]) throws Exception
	{

		Remote r=Naming.lookup("rmi://127.0.0.1/bankobj");
			dnsinter m=(dnsinter)r;
			String ip=m.showip("www.google.com");
			System.out.println("ip address is" +ip);
		    }
}
```

Problem Validation

Compiling and Running:

```
C:\Documents and Settings\student>cd Desktop
C:\Documents and Settings\student\Desktop>cd dns
C:\Documents and Settings\student\Desktop\dns>javac *.java
C:\Documents and Settings\student\Desktop\dns>rmic DnsImpl
C:\Documents and Settings\student\Desktop\dns>start rmiregistry
RMI OBJECT BOUND TO REGISTRY
C:\Documents and Settings\student\Desktop\dns>start java DnsServer
C:\Documents and Settings\student\Desktop\dns>java DnsClient
ip address is 192.1.8.6
C:\Documents and Settings\student\Desktop\dns>
```

Input: None
Output: ip address is 192.1.8.6
Problem Definition
Implementing for Adding 2 Numbers Service using RMI

Program 2
BEAN DEVELOPMENT KIT

Problem Definition

Create a Java Bean to draw various graphical shapes and display it using or without using BDK

Problem Description

Overview

JavaBeans™ is a portable, platform-independent component model written in the Java programming language. The JavaBeans architecture was built through a collaborative industry effort and enables developers to write reusable components in the Java programming language. Java Bean components are known as beans. Beans are dynamic in that they can be changed or customized. Through the design mode of a builder tool, you use the property sheet or bean customizer to customize the bean and then save (persist) your customized beans.

Advantages

Beans features:

· GUI (graphical user interface).
· Non-visual beans, such as a spelling checker.
· Animation applet.
· Spreadsheet application.

Architecture

The JavaBeans™ architecture is based on a component model which enables developers to create software units called components. Components are self-contained,
reusable software units that can be visually assembled into composite components, applets, applications, and servlets using visual application builder tools.

Beans are dynamic in that they can be changed or customized. Through the design mode of a builder tool you can use the Properties window of the bean to customize the bean and then save (persist) your beans using visual manipulation. You can select a bean from the toolbox, drop it into a form, modify its appearance and behavior, define its interaction with other beans, and combine it and other beans into an applet, application, or a new bean.

The following list briefly describes key bean concepts.

· Introspection.
· Properties.
· Customization
· Events
· Persistence
· Methods.

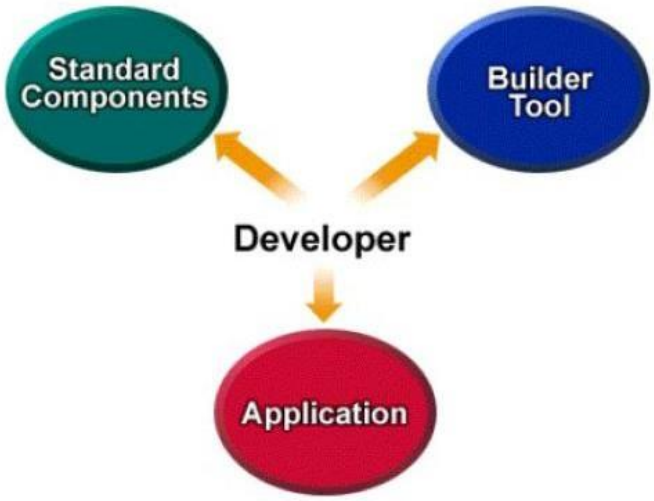


Figure: Integration of Beans with Applications using Builder tools

Implementation

Writing a Simple Bean

1. Creating a simple bean.
2. Compiling the bean.
3. Generating a Java Archive (JAR) file.
4. Loading the bean into the GUI Builder of the NetBeans IDE.
5. Inspecting the bean's properties and events.

Program:

```
package newpackage;
import javax.swing.JButton;
import java.awt.*;
import java.io.Serializable;
public class grbuttonx extends JButton implements Serializable
{
private Image img;
public grbuttonx()
{ img=null;
}
public void setImage(Image i)
{ img=i;
}
@Override
public void paint(Graphics g)
{ if (
img != null)
g.drawImage(img, 0, 0,null);
else
{
```

```
g.setColor(Color.red);
Dimension size = getSize();
g.fillOval(0, 0, size.width, size.height);
}
}
}
```

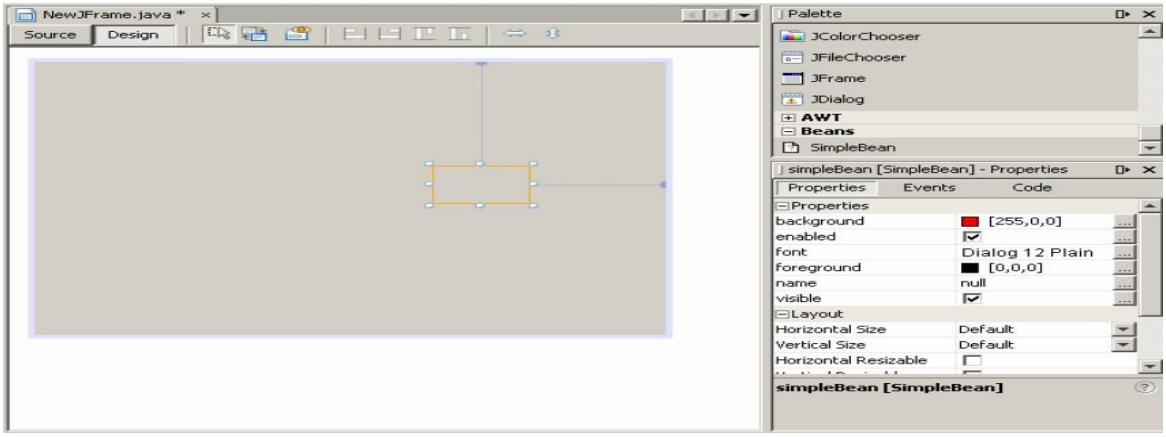
Place The Following Line In build.xml
<manifest> <attribute name="Main-Class" value="${main.class}"/></manifest>

Note - Simple Bean extends the javax.swing.JLabel graphic component and inherits its properties, which makes the SimpleBean a visual component. **SimpleBean** also implements the java.io.Serializable interface. Your bean may implement either the **Serializable** or the **Externalizable interface**.

Problem Validation:
Compiling and Running:
1. Compiling the bean.
2. Generating a Java Archive (JAR) file.
3. Loading the bean into the GUI Builder of the NetBeans IDE.
4. Inspecting the bean's properties and events.

Input:
None

Output:
Sample Output:



Problem Definition

Program to create java bean without using BDK

Problem Description

Overview

JavaBeans™ is a portable, platform-independent component model written in the Java programming language. The JavaBeans architecture was built through a collaborative industry effort and enables developers to write reusable components in the Java programming language. Java Bean components are known as beans. Beans are dynamic in that they can be changed or customized. Through the design mode of a builder tool, you use the property sheet or bean customizer to customize the bean and then save (persist) your customized beans.

Advantages

Beans features:

· GUI (graphical user interface).
· Non-visual beans, such as a spelling checker.
· Animation applet.
· Spreadsheet application.

Architecture

The JavaBeans™ architecture is based on a component model which enables developers to create software units called components. Components are self-contained,
reusable software units that can be visually assembled into composite components, applets, applications, and servlets using visual application builder tools.

Beans are dynamic in that they can be changed or customized. Through the design mode of a builder tool you can use the Properties window of the bean to customize the bean and then save (persist) your beans using visual manipulation. You can select a bean from the toolbox, drop it into a form, modify its appearance and behavior, define its interaction with other beans, and combine it and other beans into an applet, application, or a new bean.

The following list briefly describes key bean concepts.

· Introspection.
· Properties.
· Customization
· Events
· Persistence
· Methods.

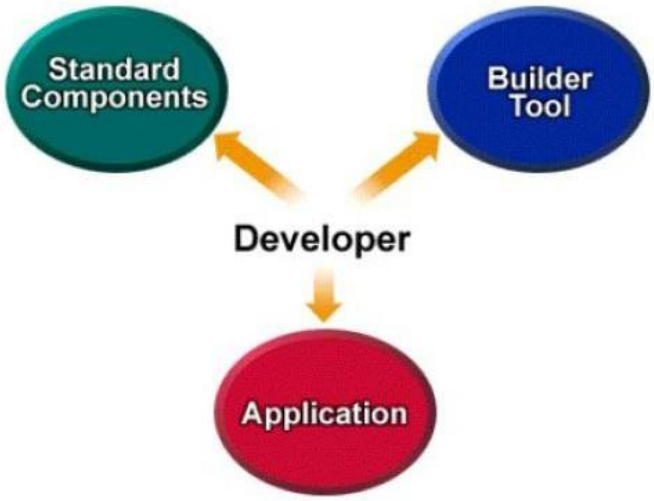


Figure: Integration of Beans with Applications using Builder tools

Implementation

Writing a Simple Bean

1. Creating a simple bean.
2. Compiling the bean.
3. Generating a Java Archive (JAR) file.
4. Loading the bean into the GUI Builder of the NetBeans IDE.
5. Inspecting the bean's properties and events.

Program:

PersonBean.java

```
public class PersonBean implements java.io.Serializable {
    private String name;
   private boolean deceased;
    public String getName() {        return this.name;    }
    public void setName(final String name) {        this.name = name;    }
    public boolean isDeceased() {        return this.deceased;    }
    public void setDeceased(final boolean deceased) {        this.deceased = deceased;    }
}
```

TestPersonBean.java

```
        public class TestPersonBean
          {
        public static void main(String[] args)
         {
       PersonBean person = new PersonBean();
       person.setName("Bob");
       person.setDeceased(false);
```

```
    System.out.print(person.getName());
     System.out.println(person.isDeceased() ? " [deceased]" : " [alive]");
    }
       }
```

Problem Validation:

Compiling and Running:

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\student>cd Desktop
C:\Documents and Settings\student\Desktop>cd "javabean with or without bdk"
C:\Documents and Settings\student\Desktop\javabean with or without bdk>javac PersonBean.java
C:\Documents and Settings\student\Desktop\javabean with or without bdk>javacTes tPersonBean.java
C:\Documents and Settings\student\Desktop\javabean with or without bdk>java TestPersonBean
Bob [alive]

Input: None.
Output: Bob [alive]

Program 3
ENTERPRISE JAVA BEANS

Problem Definition

Develop an Enterprise Java Bean for student Information System.

Problem Description

Overview

Enterprise beans are Java EE components that implement Enterprise JavaBeans(EJB) technology. Enterprise beans run in the EJB container, a runtime environment within the Application Server. Although transparent to the application developer, the EJB container provides system-level services such as transactions and security to its enterprise beans. These services enable you to quickly build and deploy enterprise beans, which form the core of transactional Java EE applications.

Written in the Java programming language, an enterprise bean is a server-side component that encapsulates the business logic of an application. The business logic is the code that fulfills the purpose of the application. In an inventory control application, for example, the enterprise beans might implement the business logic in methods called Check Inventory Level and order Product. By invoking these methods, clients can access the inventory services provided by the application.

Advantages

EJB features:

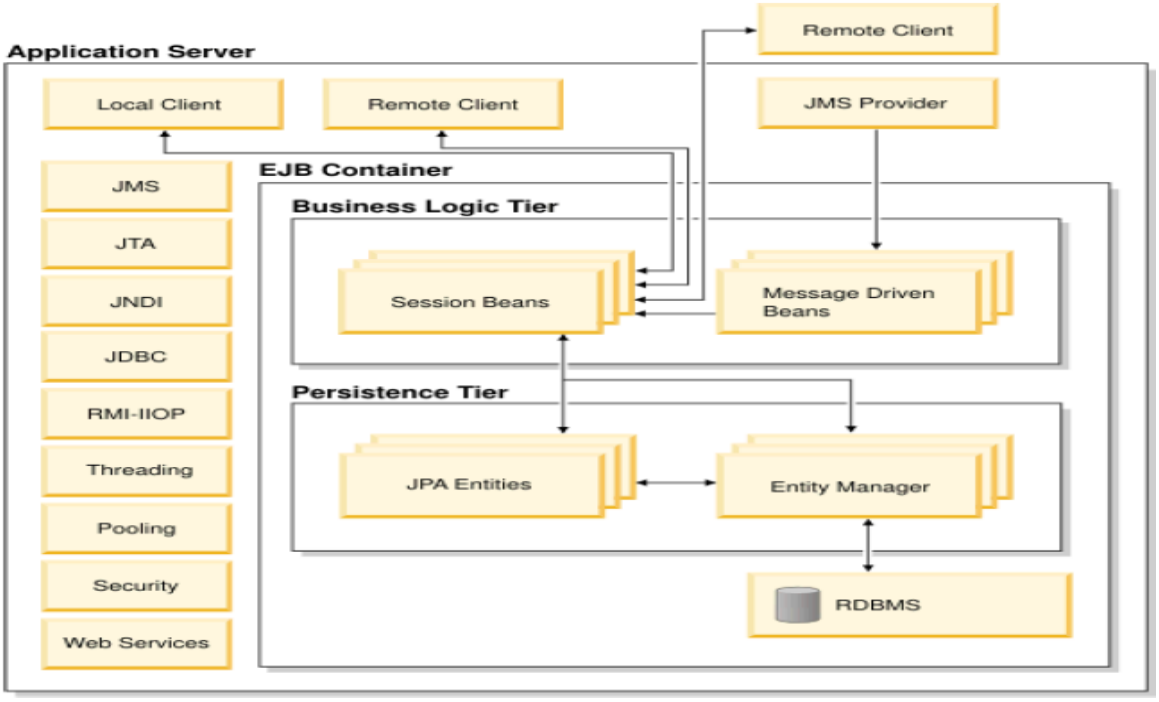
· Enterprise beans simplify the development of large, distributed applications.
· The EJB container provides system-level services to enterprise beans.
· The beans rather than the clients contain the application's business logic.
· The clients are thinner.
· Enterprise beans are portable components.

Types of Enterprise Beans

Session: Performs a task for a client; optionally may implement a web service
Message-Driven: Acts as a listener for a particular messaging type, such as the Java Message Service API

Note - Entity beans have been replaced by Java Persistence API entities

Architecture



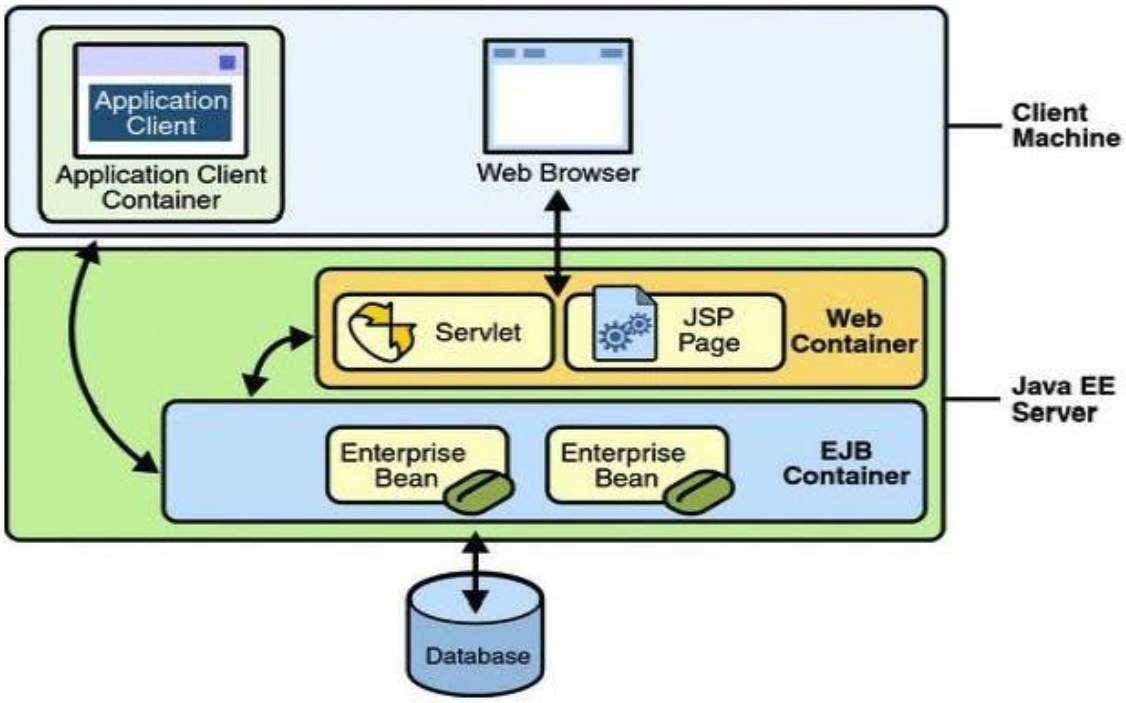


Figure :EJB Container inside Java EE Server

The Life Cycles of Enterprise Beans

An enterprise bean goes through various stages during its lifetime, or life cycle. Each type of enterprise bean (stateful session, stateless session, or message-driven) has a different life cycle. The descriptions that follow refer to methods that are explained along with the code examples in the next two chapters. If you are new to enterprise beans, you should skip this section and run the code examples first.

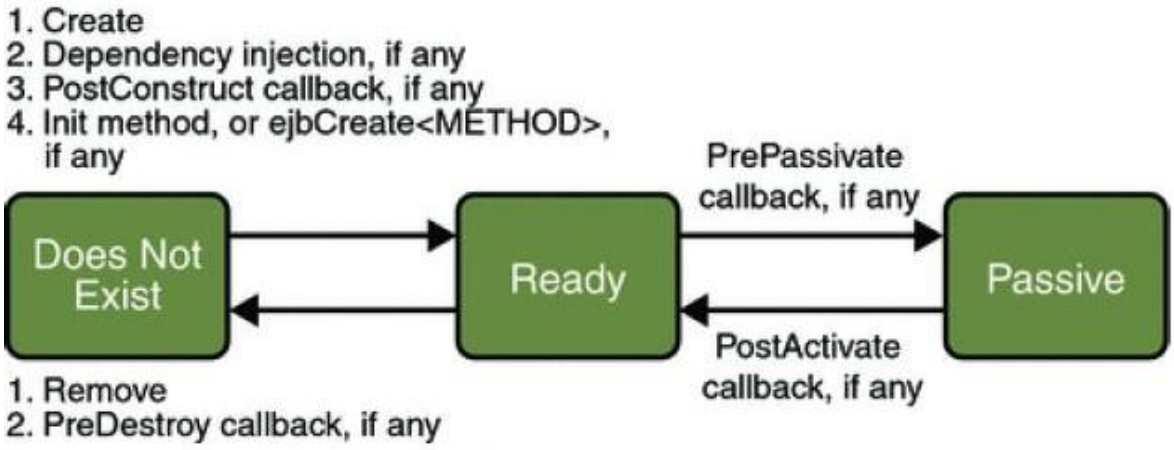


Figure: Life Cycle of a Stateful Session Bean

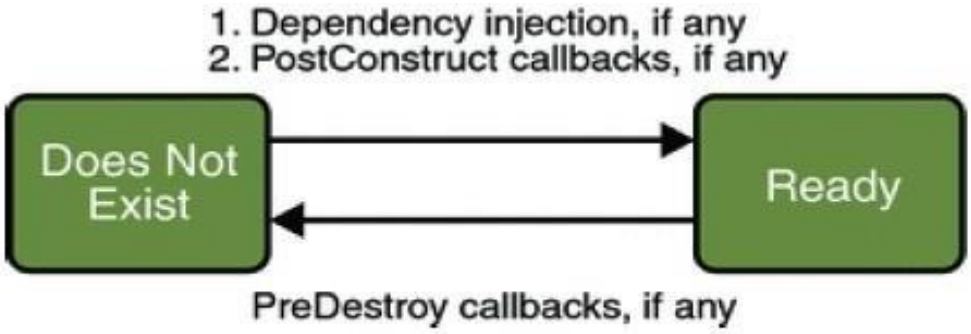


Figure: Life Cycle of a Stateless Session Bean

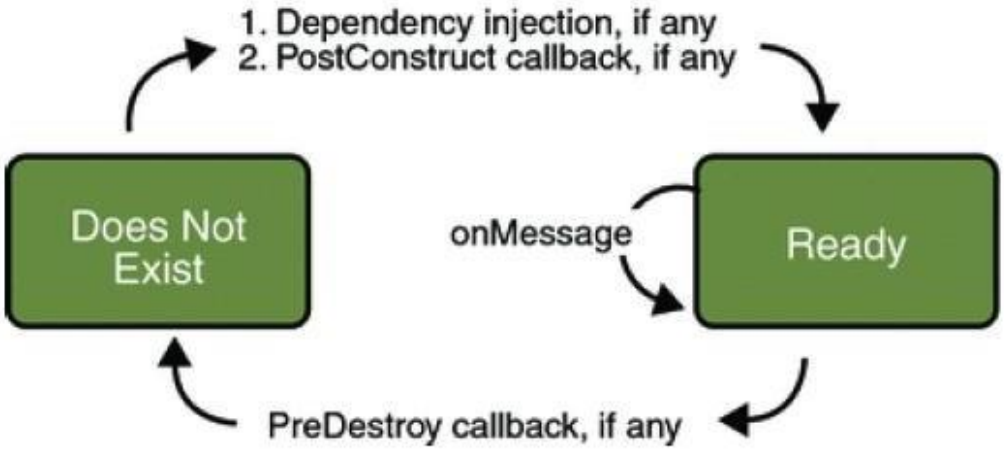


Figure: Life Cycle of a Message-Driven Bean

Program:

NewEntity.java

```
package newpackage;
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;
@Entity
public class NewEntity implements Serializable {
    //private static final long serialVersionUID = 1L;
    //@GeneratedValue(strategy = GenerationType.AUTO)
    @Id
    private Long id;
    private String name;
    public NewEntity() {     }
    public String getName() {         return name;     }
    public void setName(String name) {         this.name = name;     }
    public Long getId() {         return id;     }

    public void setId(Long id) {         this.id = id;     }
    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }
    @Override
    public boolean equals(Object object) {
```

```
 if (!(object instanceof NewEntity)) {            return false;         }
        NewEntity other = (NewEntity) object;
        if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "newpackage.NewEntity[id=" + id + "]";
    }
}
```

NewEntityFacade.java

```
package newpackage;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
@Stateless
public class NewEntityFacade implements NewEntityFacadeRemote {
    @PersistenceContext(unitName="EnterpriseApplication2-ejbPU")
    private EntityManager em;
    public void create(Long id,String name) {

        NewEntity newEntity =new NewEntity();
        newEntity.setId(id);
        newEntity.setName(name);
        em.persist(newEntity);
    }
    public void edit(NewEntity newEntity) {
        em.merge(newEntity);
    }
    public void remove(NewEntity newEntity) {
        em.remove(em.merge(newEntity));
    }
    public NewEntity find(Object id) {

  return em.find(NewEntity.class, id);
    }
    public List<NewEntity> findAll() {
        return em.createQuery("select object(o) from NewEntity as o").getResultList();
    }
}
```

NewEntityFacadeRemote.java

```
package newpackage;
import java.util.List;
import javax.ejb.Remote;

@Remote
public interface NewEntityFacadeRemote {
    void create(Long id,String name);
    void edit(NewEntity newEntity);
    void remove(NewEntity newEntity);
    NewEntity find(Object id);
    List<NewEntity> findAll();
}
```

NewServlet.java

```
package newpackage;
import java.io.IOException;
import java.io.PrintWriter;
import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class NewServlet extends HttpServlet {
@EJB
private NewEntityFacadeRemote newEntityFacade;
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {
        Long id=null;
        String name=null;
        id=Long.parseLong(request.getParameter("id"));
        name=request.getParameter("name");
        newEntityFacade.create(id,name);
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            // TODO output your page here
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet NewServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet NewServlet at " + request.getContextPath () + "</h1>");
            out.println("</body>");
            out.println("</html>");
        } finally {
```

```
            out.close();
        }
    }
    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on the
left to edit the code.">
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }
    @Override
    public String getServletInfo() {
        return "Short description";
    }// </editor-fold>
}
```

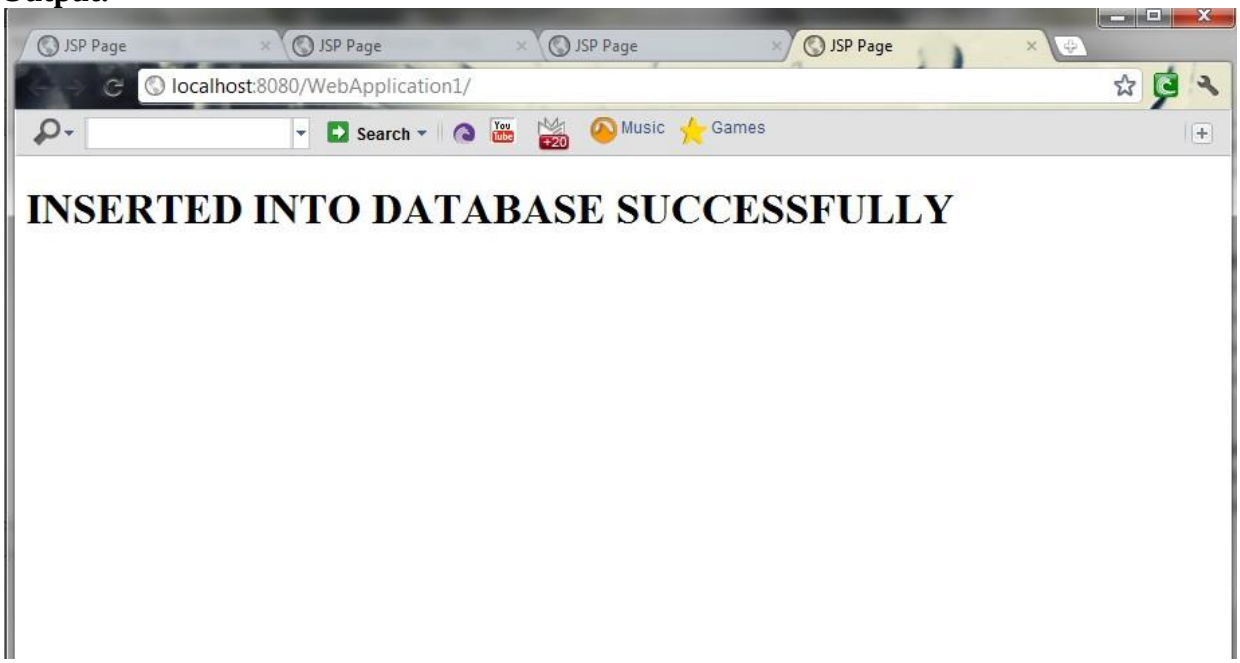
index.jsp

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello !</h1>
        <form action="NewServlet" method="GET">
            <input type="text" name="id" value="" />
            <input type="text" name="name" value="" />
            <input type="submit" value="submit" />
        </form>
    </body>
</html>
```

Problem Validation:
Compiling and Running: Press F5 to compile and Run.

Input:

Output:



Program 4
ENTERPRISE JAVA BEANS

Problem Definition

Develop an Enterprise Java Bean for Library operations.

Problem Description

Overview

Enterprise beans are Java EE components that implement Enterprise JavaBeans (EJB) technology. Enterprise beans run in the EJB container, a runtime environment within the Application Server. Although transparent to the application developer, the EJB container provides system-level services such as transactions and security to its enterprise beans. These services enable you to quickly build and deploy enterprise beans, which form the core of transactional Java EE applications.

Written in the Java programming language, an enterprise bean is a server-side component that encapsulates the business logic of an application. The business logic is the code that fulfills the purpose of the application. In an inventory control application, for example, the enterprise beans might implement the business logic in methods called Check Inventory Level and order Product. By invoking these methods, clients can access the inventory services provided by the application.

Advantages

EJB features:

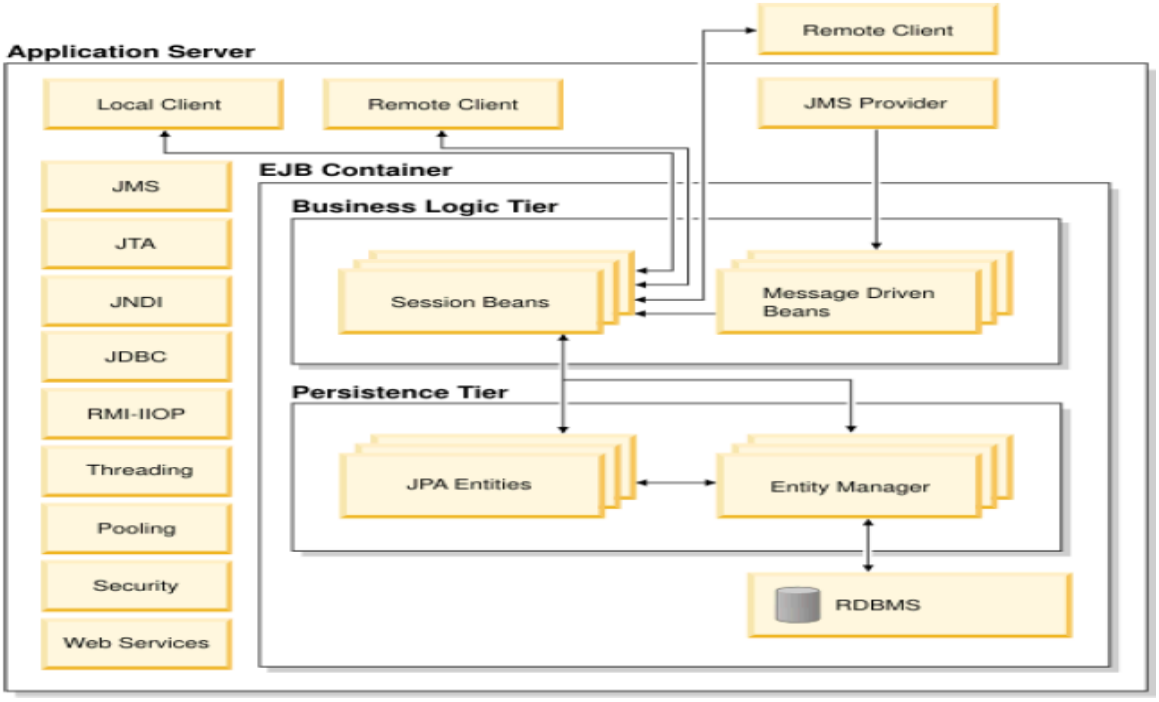
· Enterprise beans simplify the development of large, distributed applications.
· The EJB container provides system-level services to enterprise beans.
· The beans rather than the clients contain the application's business logic.
· The clients are thinner.
· Enterprise beans are portable components.

Types of Enterprise Beans

Session: Performs a task for a client; optionally may implement a web service
Message-Driven: Acts as a listener for a particular messaging type, such as the Java Message Service API

Note - Entity beans have been replaced by Java Persistence API entities

Architecture



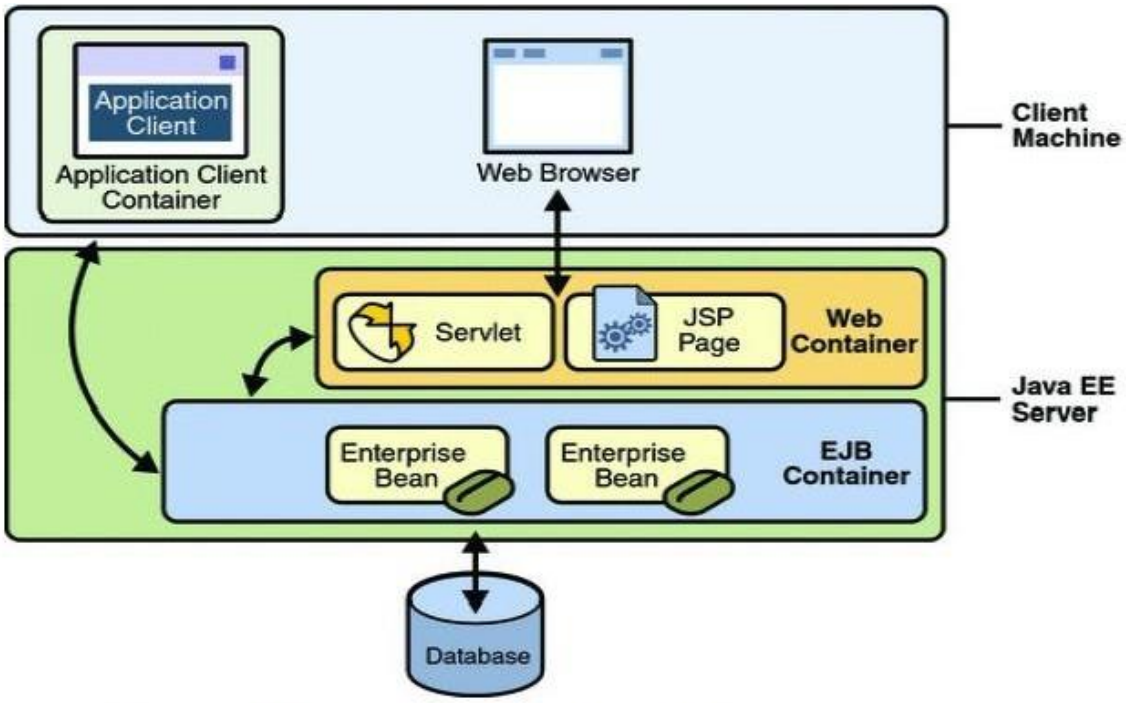


Figure :EJB Container inside Java EE Server

The Life Cycles of Enterprise Beans

An enterprise bean goes through various stages during its lifetime, or life cycle. Each type of enterprise bean (stateful session, stateless session, or message-driven) has a different life cycle. The descriptions that follow refer to methods that are explained along with the code examples in the next two chapters. If you are new to enterprise beans, you should skip this section and run the code examples first.

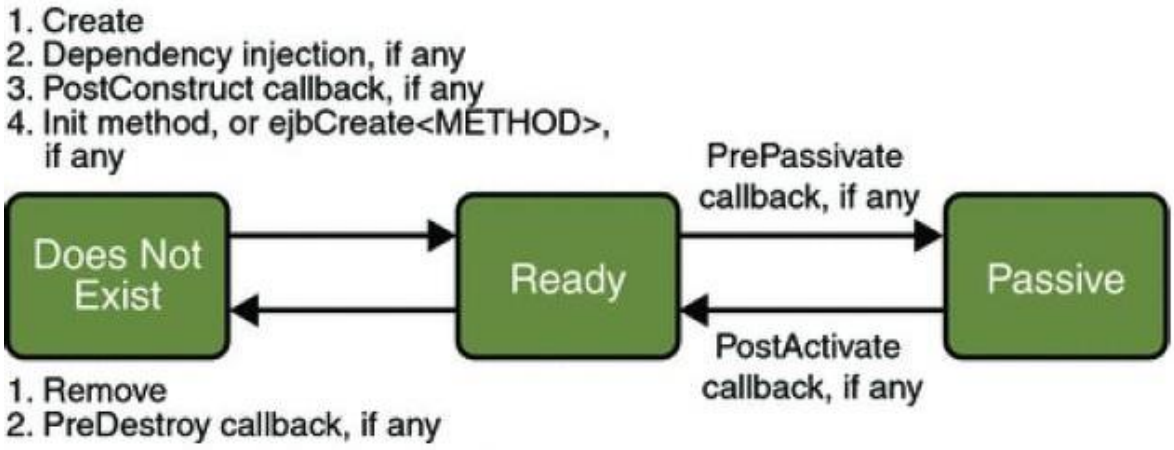


Figure: Life Cycle of a Stateful Session Bean

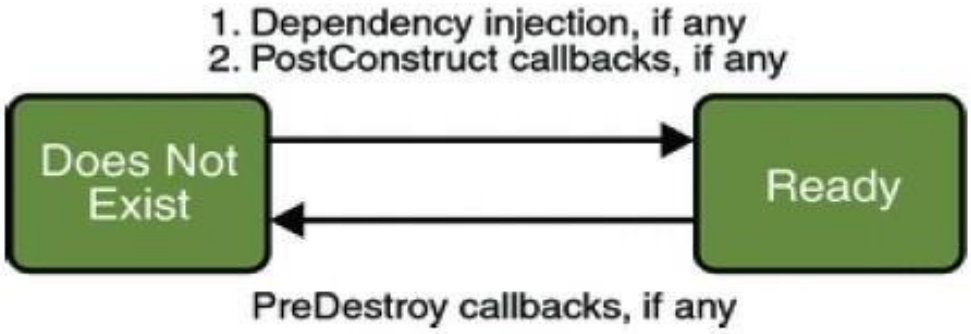


Figure: Life Cycle of a Stateless Session Bean

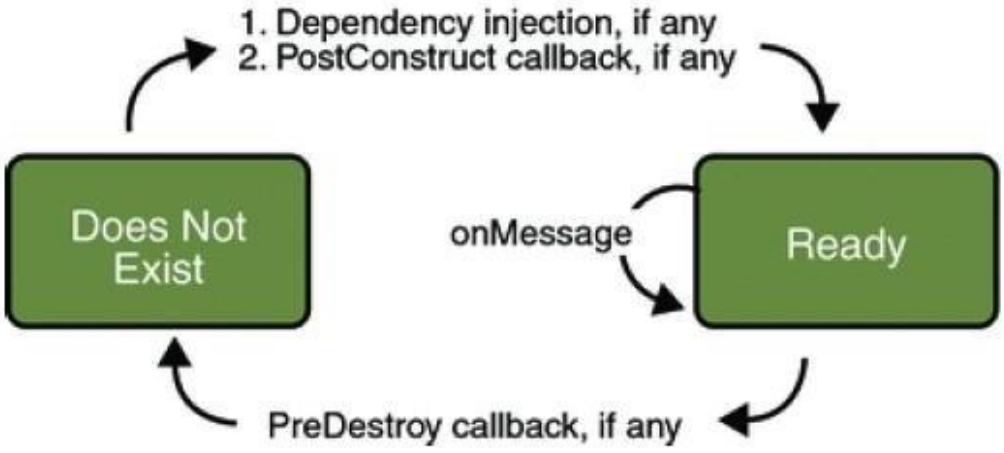


Figure: Life Cycle of a Message-Driven Bean

Program:

NewEntity.java

```
package newpack beage;
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
@Entity
public class NewEntity implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String stu_name;
    private String stu_cls;
    private Long no_of_buks_issued;

    public NewEntity() {    }

    public Long getNo_of_buks_issued() {
        return no_of_buks_issued;
    }

    public void setNo_of_buks_issued(Long no_of_buks_issued) {
        this.no_of_buks_issued = no_of_buks_issued;
    }
public String getStu_cls() {
```

```
        return stu_cls;
    }

    public void setStu_cls(String stu_cls) {
        this.stu_cls = stu_cls;
    }

    public String getStu_name() {
        return stu_name;
    }

    public void setStu_name(String stu_name) {
        this.stu_name = stu_name;
    }

public Long getId() {
return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id fields are not set
        if (!(object instanceof NewEntity)) {
            return false;
        }
        NewEntity other = (NewEntity) object;
        if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }
    @Override
    public String toString() {
        return "newpackage.NewEntity[ id=" + id + " ]";
    }
```

```
}
```

NewEntityFacade.java

```
package newpackage;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
@Stateless
public class NewEntityFacade extends AbstractFacade<NewEntity> {
    @PersistenceContext(unitName = "EnterpriseApplication4-ejbPU")
    private EntityManager em;
public void create(Long id,String stu_name,String stu_cls,Long no_of_buks_issued) {
    NewEntity n=new NewEntity();
    n.setId(id);
    n.setStu_name(stu_name);
    n.setStu_cls(stu_cls);
    n.setNo_of_buks_issued(no_of_buks_issued);
    em.persist(n);
}
    public void edit(NewEntity newEntity) {        em.merge(newEntity);    }

    public void remove(NewEntity newEntity) {        em.remove(em.merge(newEntity));    }

    public NewEntity find(Object id) {
  return em.find(NewEntity.class, id);
    }

    public List<NewEntity> findAll() {
        return em.createQuery("select object(o) from NewEntity as o").getResultList();
    }
 }
```

NewEntityFacadeRemote.java

```
package newpackage;
import java.util.List;
import javax.ejb.Remote;
@Remote
public interface NewEntityFacadeRemote {
    void create(Long id,String name);
    void edit(NewEntity newEntity);
    void remove(NewEntity newEntity);
    NewEntity find(Object id);
    List<NewEntity> findAll();
}
```

NewServlet.java

```
package newpackage;

import java.io.IOException;
import java.io.PrintWriter;
import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "NewServlet", urlPatterns = {"/NewServlet"})
public class NewServlet extends HttpServlet {
   @EJB
   private NewEntityFacade newEntityFacade;
   protected void processRequest(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {
        Long id=Long.parseLong(request.getParameter("id"));
       String stu_name=request.getParameter("stu_name");
       String stu_cls=request.getParameter("stu_cls");
Long no_of_buks_issued=Long.parseLong(request.getParameter("no_of_buks_issued"));
       newEntityFacade.create(id,stu_name,stu_cls,no_of_buks_issued);
       response.setContentType("text/html;charset=UTF-8");
       PrintWriter out = response.getWriter();
       try {

           out.println("<html>");
           out.println("<head>");
           out.println("<title>Servlet NewServlet</title>");
           out.println("</head>");
           out.println("<body>");
           out.println("<h1>name is " + stu_name + "</h1>");

           out.println("<h1>id is " + id + "</h1>");
             out.println("<h1>class is is " + stu_cls + "</h1>");
              out.println("<h1>no of buks issued are" + no_of_buks_issued + "</h1>");
           out.println("</body>");
           out.println("</html>");
     } finally {
           out.close();
       }
   }
```

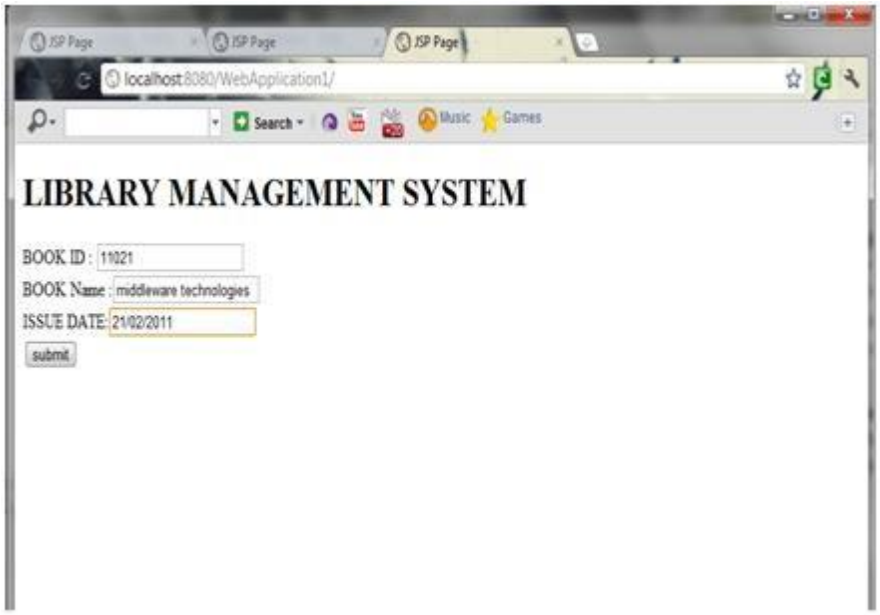
index.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
```

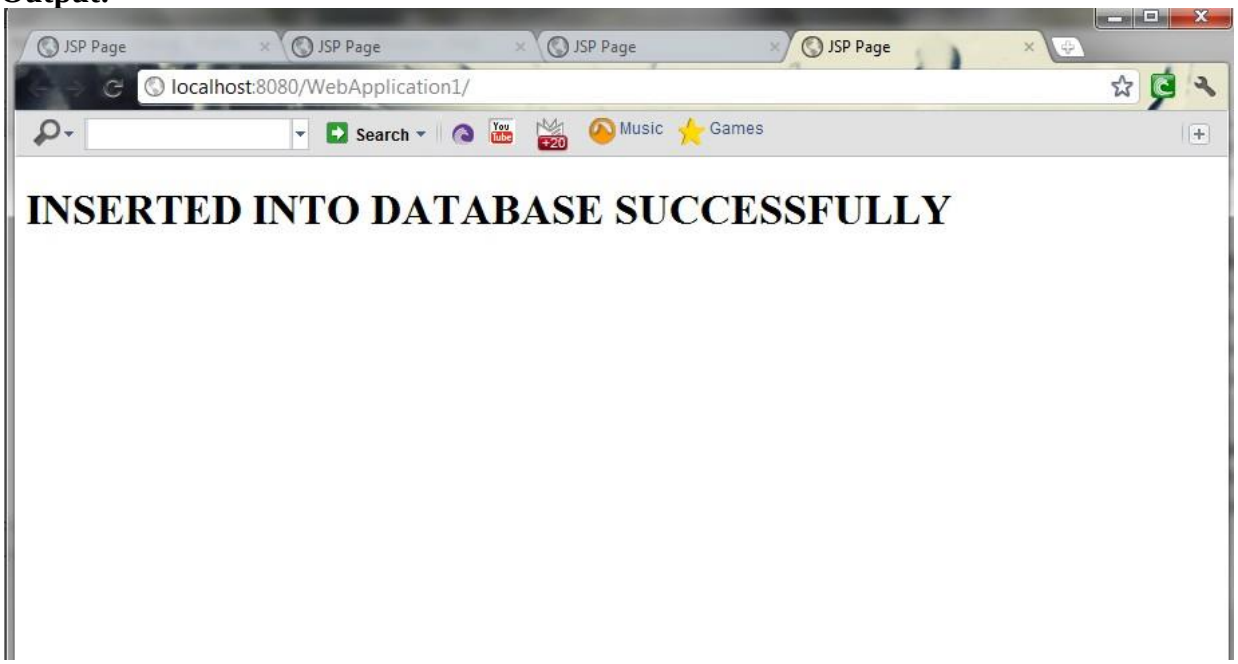
```
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>JSP Page</title>
   </head>
   <body>
      <h1>Hello World!</h1
      ><form method="get" action="NewServlet"/>
      <p> ur name is.. </p>
      <input type="text" name="stu_name" value="enter name" />
      <p> ur id is.. </p>
      <input type="text" name="id" value="enter id" />
      <p> ur cls is.. </p>
      <input type="text" name="stu_cls" value="enter cls" />
      <p> buks isued are.. </p>
      <input type="text" name="no_of_buks_issued" value="enter no of buks" />
      <input type="submit" value="submit"/>

   </body>
</html>
```

Problem Validation:
Compiling and Running:
Steps for developing EJB
1. Create the Remote interface.
2. Create the Bean class.
3. Create the client.
4. Deploy the bean.
5. Run the bean.

Input:


JSP Page
JSP Page
JSP Page
localhost:8080/WebApplication1/
Search
Music
Games
LIBRARY MANAGEMENT SYSTEM
BOOK ID : 11021
BOOK Name : middleware technologies
ISSUE DATE: 21/02/2011
submit

Output:


JSP Page
JSP Page
JSP Page
JSP Page
localhost:8080/WebApplication1/
Search
Music
Games
INSERTED INTO DATABASE SUCCESSFULLY

Program 5
Creating Active-X controls

Problem Definition
Create an Active-X control for Timetable

Problem Description
Here we use active x control to load data from a file and display it on a window.

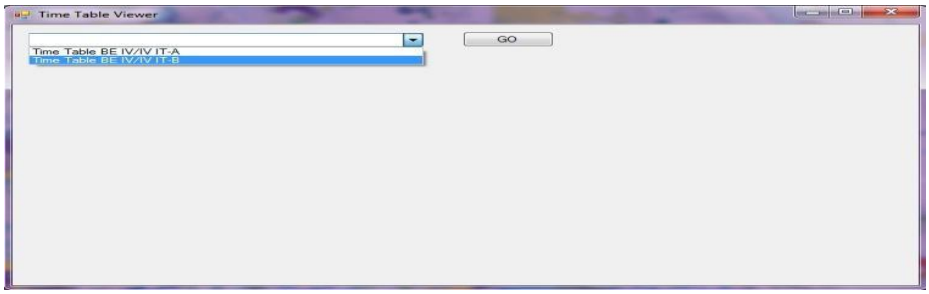
Program

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace Timetable
{
	public partial class Form1 : Form
	{
		public Form1()
		{
			InitializeComponent();
		}

		private void button1_Click(object sender, EventArgs e)
		{
			switch (comboBox1.SelectedIndex)
			{
			case 0: pictureBox1.Image =
		Image.FromFile(@"C:\Users\Farhaan\Documents\Visual Studio
		2008\Projects\Timetable\Timetable\IT-A.jpg");
				break;
				case 1: pictureBox1.Image =
			Image.FromFile(@"C:\Users\Farhaan\Documents\Visual Studio
			2008\Projects\Timetable\Timetable\IT-B.jpg");
				break;
			}
		}
	}
}
```

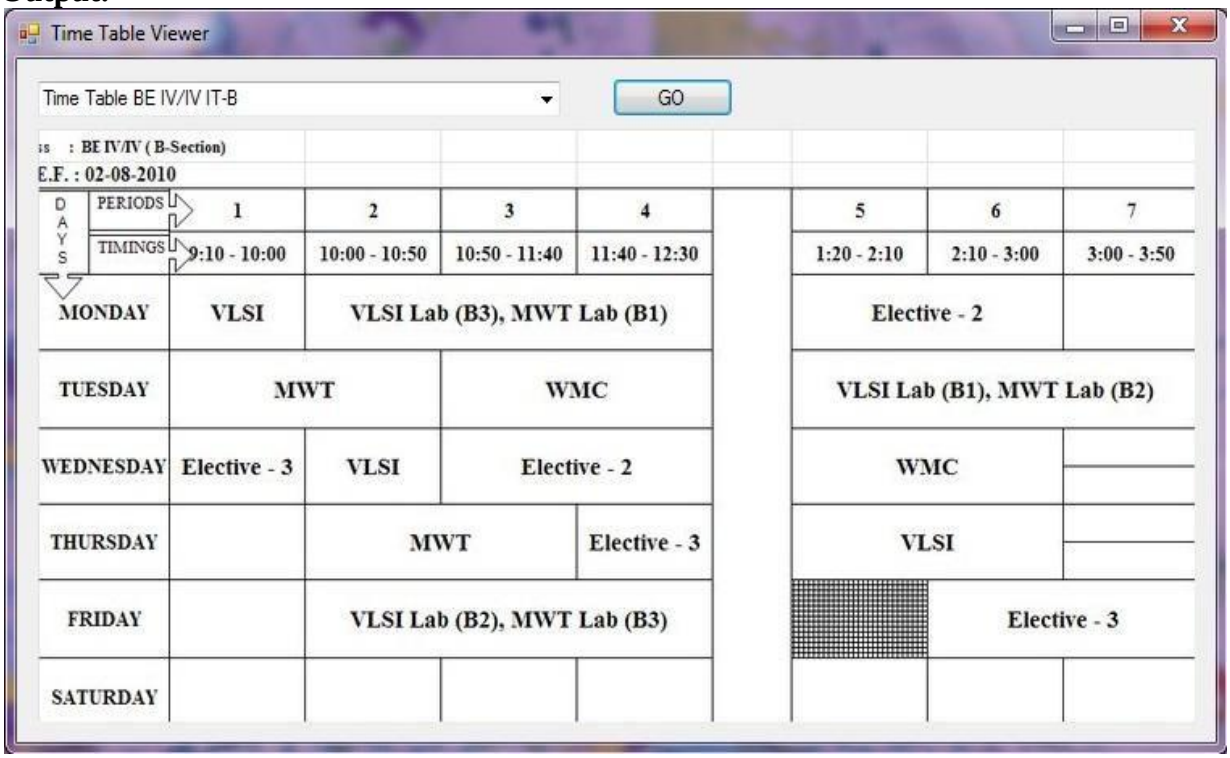
Problem Validation:

Compiling and Running:
Press F5 Button for compiling and Running.

Input:



Output:



Program 6

Develop a component for converting the currency values using COM / .NET

Problem Definition

To implement Currency Convertor.

Problem Description

Overview

ActiveX is a framework for defining reusable software components (known as controls) that perform a particular function or a set of functions in Microsoft Windows in a way that is independent of the programming language used to implement them. A software application can then be composed from one or more of these components in order to provide its functionality.

It was introduced in 1996 by Microsoft as a development of its Component Object Model (COM) and Object Linking and Embedding (OLE) technologies and it is commonly used in its Windows operating system, although the technology itself is not tied to it.

Component Object Model (COM) is a binary-interface standard for software componentry introduced by Microsoft in 1993. It is used to enable inter-process communication and dynamic object creation in a large range of programming languages. The term COM is often used in the Microsoft software development industry as an umbrella term that encompasses the OLE, OLE Automation, ActiveX, COM+ and DCOM technologies.

Distributed Component Object Model (DCOM) is a proprietary Microsoft technology for communication among software components distributed across networked computers. DCOM, which originally was called "Network OLE", extends Microsoft's COM, and provides the communication substrate under Microsoft's COM+ application server infrastructure. It has been deprecated in favor of the Microsoft .NET Framework. Not to be confused with dcom as shorthand for data communications.

Advantages

ActiveX controls feature:

Small program building blocks to create distributed applications that work over the Internet through web browsers.

Examples include

· Customized applications for gathering data.

· Viewing certain kinds of files.

· Displaying animation.

One can compare ActiveX controls in some sense to Java applets: programmers designed both of these mechanisms so that web browsers could download and execute them.

Programmers can write ActiveX controls in any of the following languages/environments:

· MFC

· ATL

· C++

· C#RPC
· Borland Delphi
· Visual Basic

The addition of the "D" to COM was due to extensive use of DCE/RPC (Distributed Computing Environment/RPC) – more specifically Microsoft's enhanced version, known as MSRPC.

In terms of the extensions it added to COM, DCOM had to solve the problems of Marshalling – serializing and de-serializing the arguments and return values of method calls "over the wire". Distributed garbage collection – ensuring that references held by clients of interfaces are released when, for example, the client process crashed, or the network connection was lost.

Introduction to Microsoft's .NET Platform

The **Microsoft .NET Framework** is a software framework that can be installed on computers running Microsoft Windows operating systems. It includes a large library of coded solutions to common programming problems and a virtual machine that manages the execution of programs written specifically for the framework. The .NET Framework is a Microsoft offering and is intended to be used by most new applications created for the Windows platform. The framework's Base Class Library provides a large range of features including user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications. The class library is used by programmers, who combine it with their own code to produce applications. Programs written for the .NET Framework execute in a software environment that manages the program's runtime requirements.

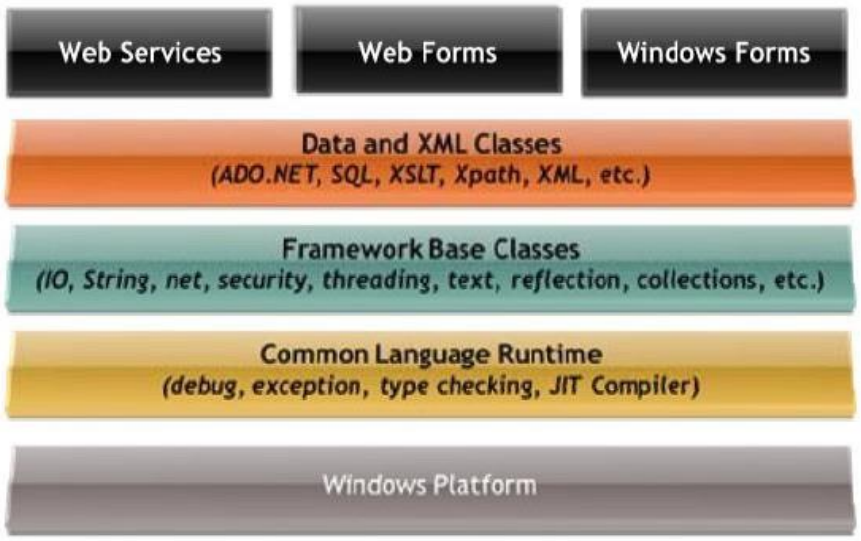
Also part of the .NET Framework, this runtime environment is known as the Common Language Runtime (CLR). The CLR provides the appearance of an application virtual machine so that programmers need not consider the capabilities of the specific CPU that will execute the program. The CLR also provides other important services such as security, memory management, and exception handling. The class library and the CLR together constitute the .NET Framework. The .NET Framework family also includes two versions for mobile or embedded device use. A reduced version of the framework, the .NET Compact Framework, is available on Windows CE platforms, including Windows Mobile devices such as smart phones. Additionally, the .NET Micro Framework is targeted at severely resource constrained devices.

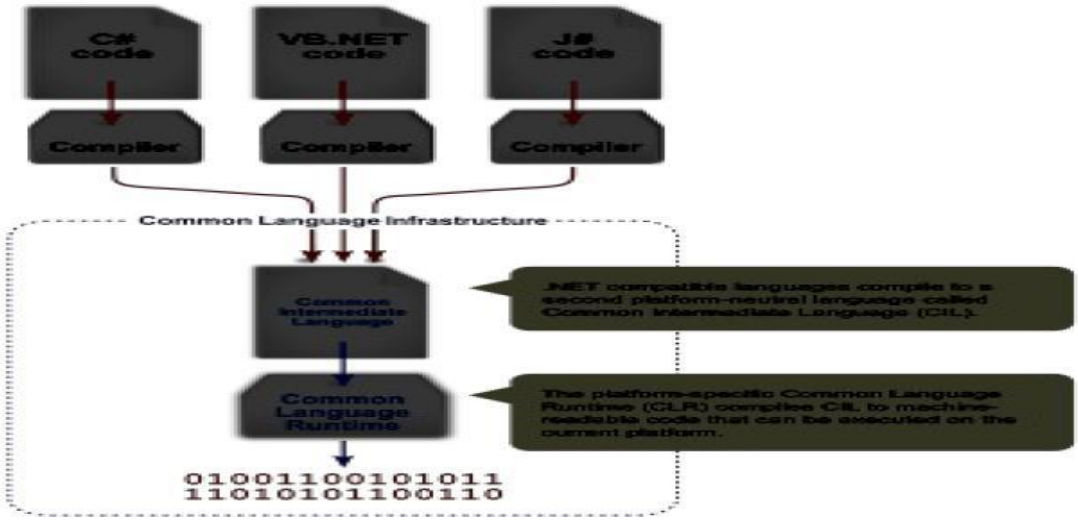
Microsoft .NET features

- Interoperability
- Common Runtime Engine
- Language Independence
- Base Class Library
- Simplified Deployment
- Security
- Portability

4.3 Architecture

Figure: .NET Framework Architecture





Program

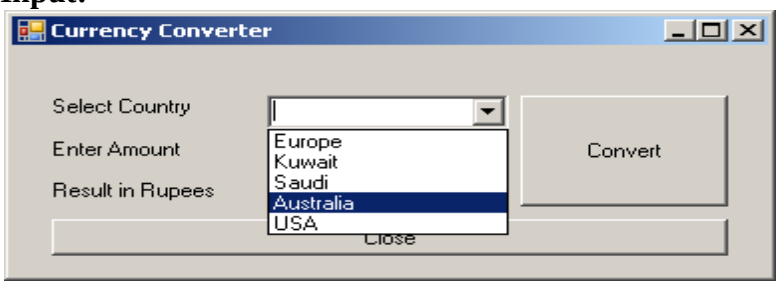
//Currency converter

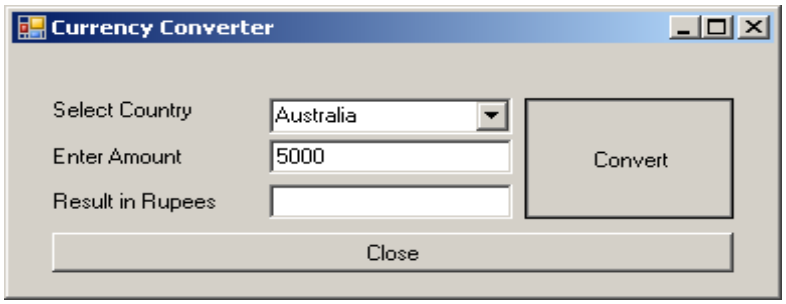
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{    public partial class Form1 : Form
   {       public Form1()
      {           InitializeComponent();       }
      private void button2_Click(object sender, EventArgs e)
      {           this.Close();       }
      private void Form1_FormClosing(object sender, FormClosingEventArgs e)
      {
         DialogResult d = MessageBox.Show("are you sure you wnat to exit", "Currency Converter",
MessageBoxButtons.YesNo, MessageBoxIcon.Question, MessageBoxDefaultButton.Button2);
         if (d == DialogResult.No)
            e.Cancel = true;       }
      private void button1_Click(object sender, EventArgs e)
      {
         if (comboBox1.SelectedIndex < 0)
         {              MessageBox.Show("please select a country", "currency converter", MessageBoxButtons.OK,
MessageBoxIcon.Information);
            comboBox1.Focus();
            return;          }
         else if (textBox1.Text.Length == 0)
         {              MessageBox.Show("please enter amount", "currency converter", MessageBoxButtons.OK,
MessageBoxIcon.Information);
            textBox1.Focus();
            return;          }
         double currencyValue = 0;
         switch (comboBox1.SelectedIndex)
         {
            case 0: currencyValue = 75;
               break;
            case 1: currencyValue = 170;
               break;
            case 2: currencyValue = 13;
               break;
            case 3: currencyValue = 43;
               break;
            case 4: currencyValue = 48;
               break;
         }
              double amount = double.Parse(textBox1.Text);
               textBox2.Text = (currencyValue * amount).ToString();
 }
}
}
```

Problem Validation:

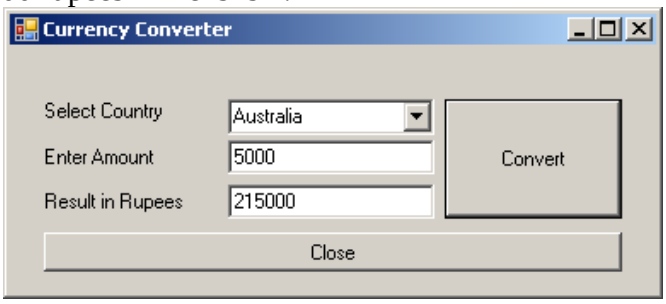
Compiling and Running:
Press F5 Button for compiling and Running.

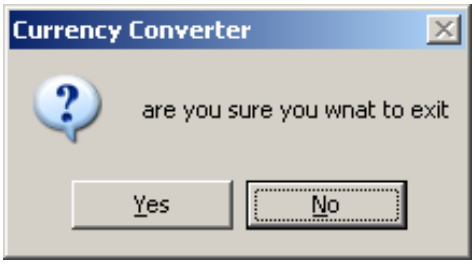
Input:





Output:
The currency value 36.00 rupees in Yens is 1.





Program 7

Develop a component for browsing CD catalogue using COM / .NET

Problem Definition

Create a component for CD/DVD browser using C#

Problem Description

Here a CD/DVD browser window was developed to browse the contents of the system or computer.

Program:

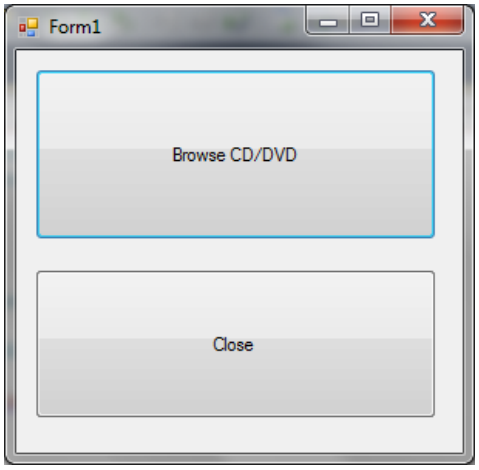
Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace Cdbrowserapp1
{
public partial class Form1 : Form
{
public Form1()
{
InitializeComponent();
}
private void button2_Click(object sender, EventArgs e)
{ this.Close();
}
private void button1_Click(object sender, EventArgs e)
{
using (OpenFileDialog open = new OpenFileDialog())
{
open.Title = "CD / DVD Browsing Application";
open.ShowDialog();
}}}}
```

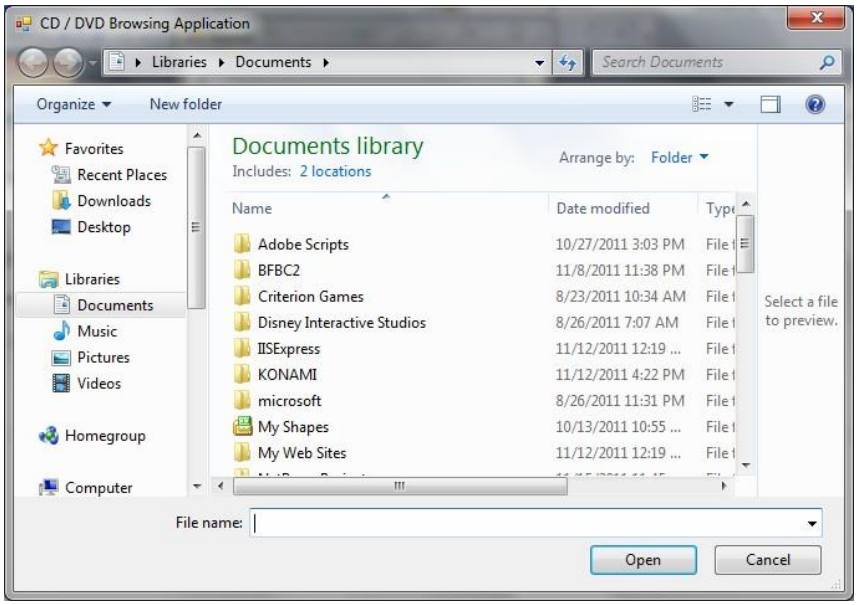
Problem Validation:

Compiling and Running:
Press F5 Button for compiling and Running.

Input: None



Output:



Program 8

Develop a component for retrieving information from message box Using DCOM/.NET

Problem Definition

Create a Component for Message Box Message Retrieval using C#.

Problem Description

Here we create a message box which inputs a text from user and performs some action depends on user action.

Program

//**Message Box**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace InformationFromMessageBox
{
    public partial class frmRetrieve : Form
    {
        public frmRetrieve()
        {
            InitializeComponent();
        }

        private void btnDisplay_Click(object sender, EventArgs e)
        {
            DialogResult d = MessageBox.Show("Click any of the button and see the
            information retrieved and displayed below!","Information To Retrieve",
            MessageBoxButtons.YesNoCancel, MessageBoxIcon.Information);

            if (d == DialogResult.Yes)
                lblInformation.Text = "You have clicked Yes button!";
            else if (d == DialogResult.No)
                lblInformation.Text = "You have clicked No button!";
            else if (d == DialogResult.Cancel)
                lblInformation.Text = "You have clicked Cancel button!";
            lblInformation.Visible = true;
        }

        private void btnClose_Click(object sender, EventArgs e)
```

```
{
        this.Close();
}

private void frmRetrieve_FormClosing(object sender, FormClosingEventArgs e)
{
        DialogResult d = MessageBox.Show("Are you sure you want to close this
        application?","Information To Retrieve",MessageBoxButtons.YesNo,
        MessageBoxIcon.Question,MessageBoxDefaultButton.Button2);
        if (d == DialogResult.No)
                e.Cancel = true;
}}}
```

Problem Validation:

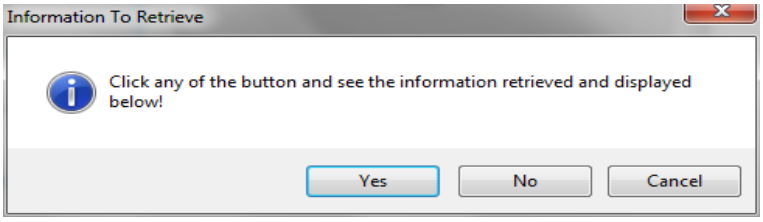
Compiling and Running:

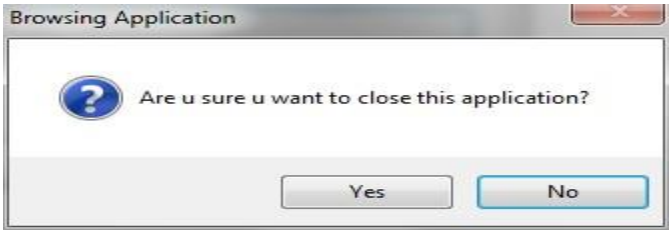
Press F5 Button for compiling and Running.

Input:

Output:

The currency value 36.00 rupees in Yens is 1.





Program 9
DEVELOPING MIDDLE-WARE COMPONENTS USING CORBA

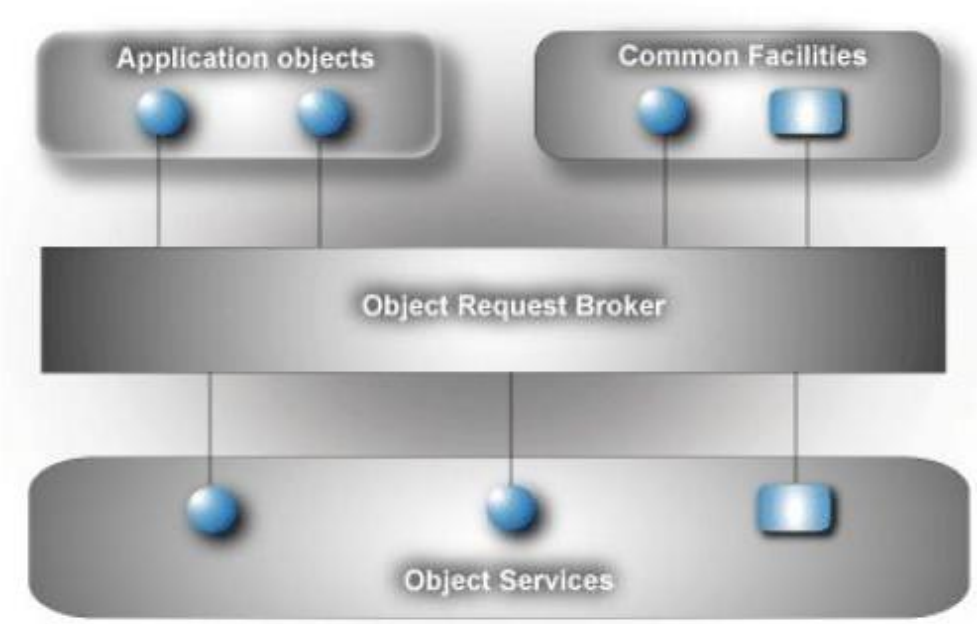
Problem Definition
Develop a middleware component for retrieving Stock Market Exchange information using CORBA

Problem Description

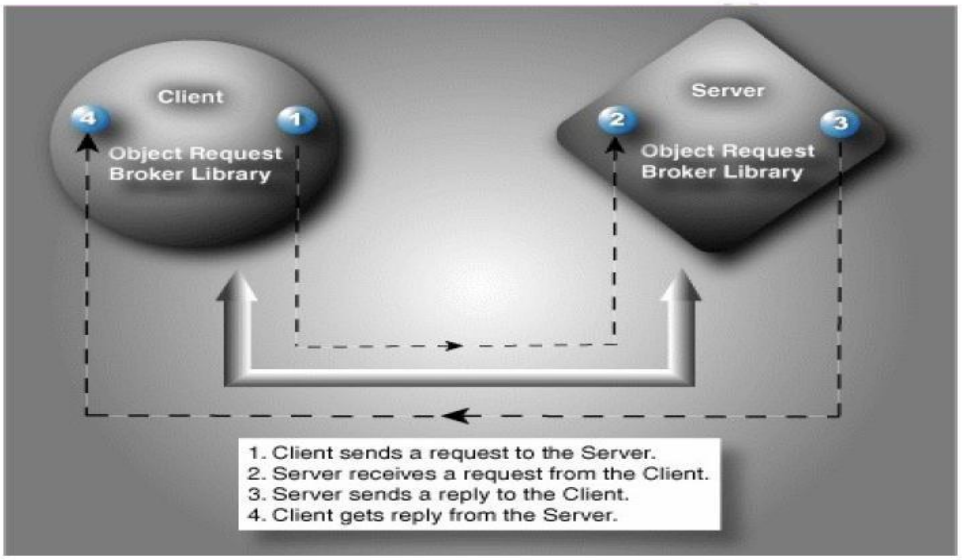
What is CORBA?
The Common Object Request Broker Architecture (CORBA) is a standard developed by the Object Management Group (OMG) to provide interoperability among distributed objects. CORBA is the world's leading middleware solution enabling the exchange of information, independent of hardware platforms, programming languages, and operating systems. CORBA is essentially a design specification for an Object Request Broker (ORB), where an ORB provides the mechanism required for distributed objects to communicate with one another, whether locally or on remote devices, written in different languages, or at different locations on a network.

The CORBA Interface Definition Language, or IDL, allows the development of language and location-independent interfaces to distributed objects. Using CORBA, application components can communicate with one another no matter where they are located, or who has designed them. CORBA provides the location transparency to be able to execute these applications.

CORBA is often described as a "software bus" because it is a software-based communications interface through which objects are located and accessed. The illustration below identifies the primary components seen within a CORBA implementation.



Data communication from client to server is accomplished through a well-defined object-oriented interface. The Object Request Broker (ORB) determines the location of the target object, sends a request to that object, and returns any response back to the caller. Through this object-oriented technology, developers can take advantage of features such as inheritance, encapsulation, polymorphism, and runtime dynamic binding. These features allow applications to be changed, modified and re-used with minimal changes to the parent interface. The illustration below identifies how a client sends a request to a server through the ORB:



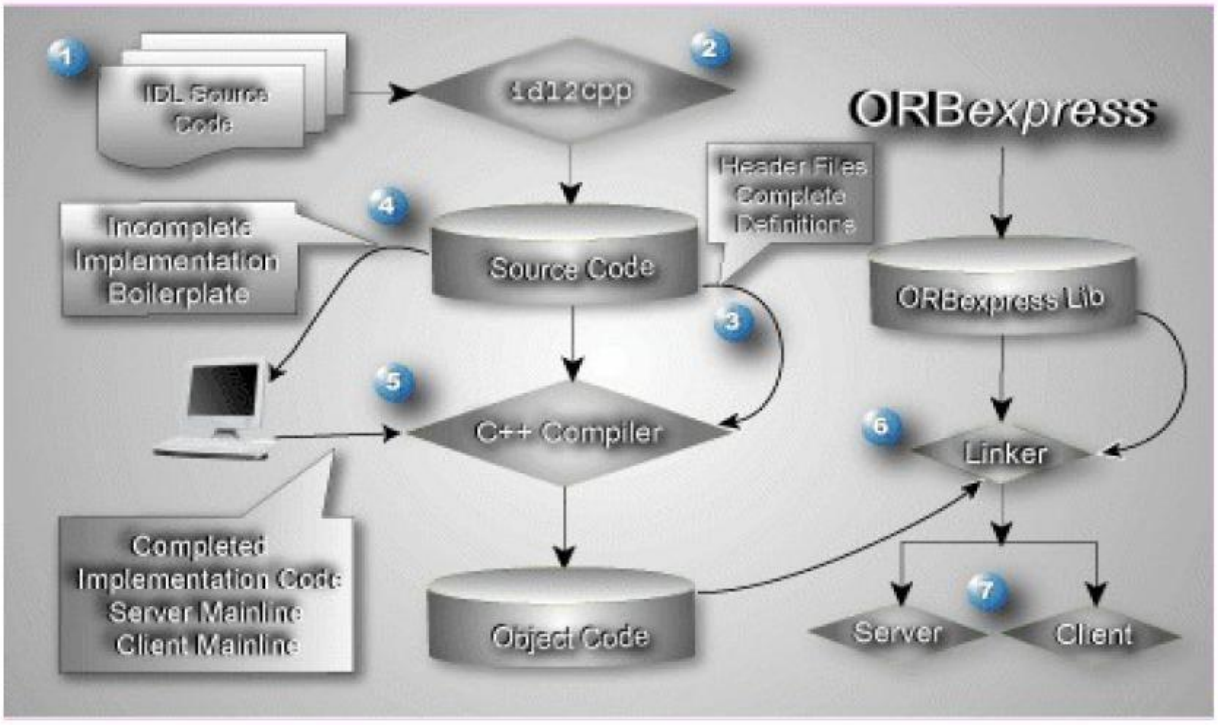
Interface Definition Language (IDL)

A cornerstone of the CORBA standards is the Interface Definition Language. IDL is the OMG standard for defining language-neutral APIs and provides the platform-independent delineation of the interfaces of distributed objects. The ability of the CORBA environments to provide consistency between clients and servers in heterogeneous environments begins with a standardized definition of the data and operations constituting the client/server interface. This standardization mechanism is the IDL, and is used by CORBA to describe the interfaces of objects.

IDL defines the modules, interfaces and operations for the applications and is not considered a programming language. The various programming languages, such as Ada, C++, or Java, supply the implementation of the interface via standardized IDL mappings.

Application Development Using ORB*express*

The basic steps for CORBA development can be seen in the illustration below. This illustration provides an overview of how the IDL is translated to the corresponding language (in this example, C++), mapped to the source code, compiled, and then linked with the ORB library, resulting in the client and server implementation.



The basic steps for CORBA development include:

1. Create the IDL to Define the Application Interfaces

The IDL provides the operating system and programming language independent interfaces to all services and components that are linked to the ORB. The IDL specifies a description of any services a server component exposes to the client. The term "IDL Compiler" is often used, but the IDL is actually translated into a programming language.

2. Translate the IDL

An IDL translator typically generates two cooperative parts for the client and server implementation, stub code and skeleton code. The stub code generated for the interface classes is associated with a client application and provides the user with a well-defined Application Programming Interface (API). In this example, the IDL is translated into C++.

3. Compile the Interface Files

Once the IDL is translated into the appropriate language, C++ in this example, these interface files are compiled and prepared for the object implementation.

4. Complete the Implementation

If the implementation classes are incomplete, the spec and header files and complete bodies and definitions need to be modified before passing through to be compiled. The output is a complete client/server implementation.

5. Compile the Implementation
Once the implementation class is complete, the client interfaces are ready to be used in the client application and can be immediately incorporated into the client process. This client process is responsible for obtaining an object reference to a specific object, allowing the client to make requests to that object in the form of a method call on its generated API.

6. Link the Application
Once all the object code from steps three and five have been compiled, the object implementation classes need to be linked to the C++ linker. Once linked to the ORB library, in this example, ORB*express*, two executable operations are created, one for the client and one for the server.

7. Run the Client and Server
The development process is now complete and the client will now communicate with the server. The server uses the object implementation classes allowing it to communicate with the objects created by the client requests.

In its simplest form, the server must perform the following:
• Create the required objects.
• Notify the CORBA environment that it is ready to receive client requests.
• Process client requests by dispatching the appropriate servant.

Overview
The **Common Object Request Broker Architecture** (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together, i.e. it supports multiple platforms.

Advantages
CORBA features:

- Language Independence
- OS Independence
- Freedom from Technologies
- Strong Data Typing
- High Tune-ability
- Freedom From Data Transfer Details
- Compression

Strengths of CORBA:

- Cross-platform and multi-vendor. Very strong support in Unix and mainframe systems.
- Is an industry standard.
- Some really excellent implementations are available for free.
- Many free versions are Open-Source
- A wider range of programming language bindings.
- Simpler programming interface.
- ALL objects/interfaces can be called dynamically at run time through a data-driven
- Interface: CORBA DII (Dynamic Invocation Interface.).
- Multiple inheritance in interfaces. (COM has single inheritance between interfaces,
- But discourages its use, favoring multiple interfaces instead.)

5.3 Architecture

CORBA is a mechanism in software for normalizing the method-call semantics between application objects that reside either in the same address space (application) or remote address space (same host, or remote host on a network). Version 1.0 was released in October 1991. CORBA uses an interface definition language (IDL) to specify the interfaces that objects will present to the outside world. CORBA then specifies a *mapping* from IDL to a specific implementation language like C++ or Java. Standard mappings exist for Ada, C, C++, Lisp, Ruby, Smalltalk, Java, COBOL, PL/I and Python. There are also non-standard mappings for Perl, Visual Basic, Erlang, and Tcl implemented by object request brokers (ORBs) written for those languages.

The CORBA specification dictates that there shall be an ORB through which the application interacts with other objects. In practice, the application simply initializes the ORB, and accesses an internal *Object Adapter* which maintains such issues as reference counting, object (and reference) instantiation policies, object lifetime policies, etc. The Object Adapter' is used to register instances of the *generated code classes*. Generated code classes are the result of compiling the user IDL code, which translates the high-level interface definition into an OS- and language-specific class base for use by the user application. This step is necessary in order to enforce the CORBA semantics and provide a clean user process for interfacing with the CORBA infrastructure.

Some IDL language mappings are "more hostile" than others. For example, due to the very nature of Java, the IDL-Java Mapping is rather straightforward and makes usage of CORBA very simple in a Java application. The C++ mapping is not trivial, but accounts for all the features of CORBA (e.g. exception handling). The C mapping is even stranger (since C is not an object-oriented language), but it does make sense and properly handles the RPC semantics.

A language mapping requires the developer ("user" in this case) to create some IDL code that represents the interfaces to his objects. Typically, a CORBA implementation comes with a tool called an IDL compiler which converts the user's IDL code into some language specific generated code. A traditional compiler then compiles the generated code to create the linkable-object files for the application. This diagram illustrates how the generated code is used within the CORBA infrastructure:

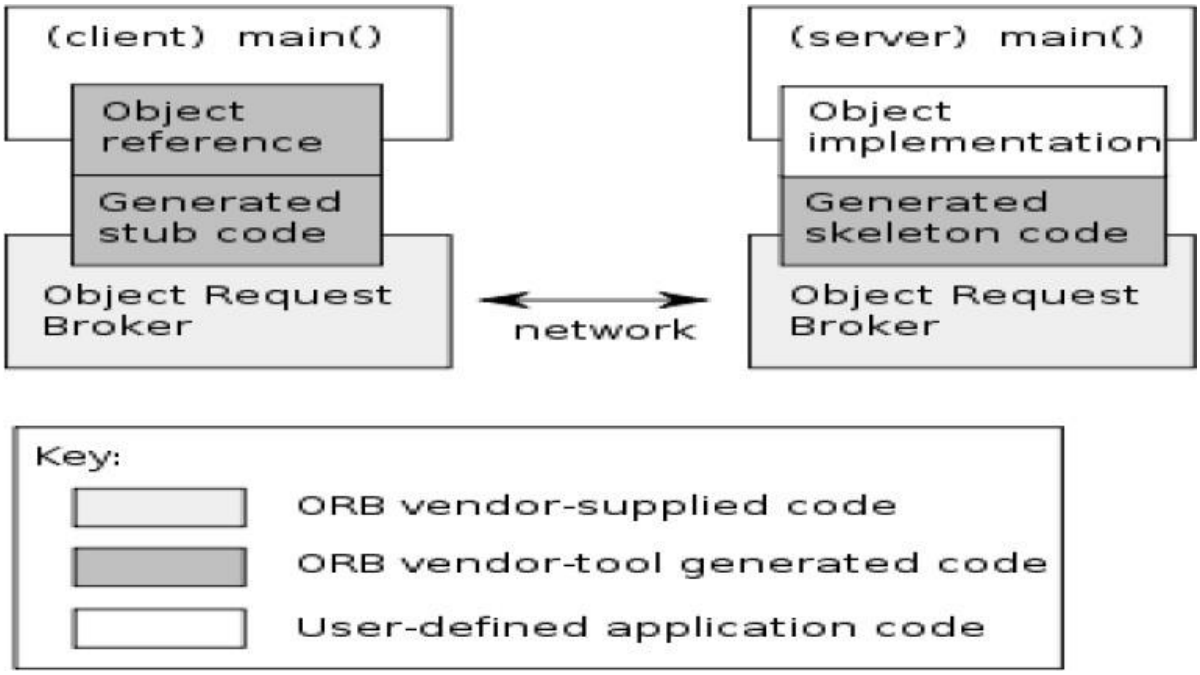


Figure: Illustration of the auto-generation of the infrastructure code from an interface defined using the CORBA IDL.

Program

Stock.idl

```
module StockApp
{
    interface Stock
    {
      long show(in string a);
    };
};
```

StockServer.java

```
import StockApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import java.sql.*;
import java.util.Properties;
```

```
class StockImpl extends StockPOA {
            private ORB orb;
                    public void setORB(ORB orb_val) {
              orb = orb_val;
            }

            public int show(String a)
              {
                    Connection con;
                PreparedStatement pt;
                ResultSet rs;
                String name = a;
                int var1=0;
           try
           {
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
             con=DriverManager.getConnection("Jdbc:Odbc:stock");
                pt = con.prepareStatement("select qty from stockDB where prod=?");
                  pt.setString(1, name);
                rs = pt.executeQuery();
                            while (rs.next()) {
                  var1 = rs.getInt("qty");

                    }
           }
           catch(ClassNotFoundException e){}
           catch(SQLException p){}

return var1;
        }
           }

           public class StockServer {
             public static void main(String args[]) {
               try{
                 ORB orb = ORB.init(args, null);
                 POA rootpoa =POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
                 rootpoa.the_POAManager().activate();
                 StockImpl stockImpl = new StockImpl();
                 stockImpl.setORB(orb);
                 org.omg.CORBA.Object ref =rootpoa.servant_to_reference(stockImpl);
                 Stock href = StockHelper.narrow(ref);
                 org.omg.CORBA.Object objRef =
                    orb.resolve_initial_references("NameService");
                    NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
                 String name = "Stock";
```

```
            NameComponent path[] = ncRef.to_name( name );
            ncRef.rebind(path, href);
            System.out.println("StockServer ready and waiting ...");
            orb.run();
           }
            catch (Exception e) {
              System.err.println("ERROR: " + e);
              e.printStackTrace(System.out);
            }

            System.out.println("StockServer Exiting ...");
         }
        }
```

// StockClient.java

```
import StockApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class StockClient
{
  static Stock stockImpl;

  public static void main(String args[])
    {
      try{
           ORB orb = ORB.init(args, null);

   org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
              NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
         String name = "Stock";
        stockImpl = StockHelper.narrow(ncRef.resolve_str(name));
        System.out.println("Obtained a handle on server object: " + stockImpl);
        String prod="abc";
        System.out.println("value is: "+stockImpl.show(prod));
           } catch (Exception e) {
          System.out.println("ERROR : " + e) ;
             e.printStackTrace(System.out);
            }
    }
}
```

Problem Validation:

Compiling and Running:

C:\Documents and Settings\User\Desktop>cd stock

C:\Documents and Settings\User\Desktop\stock>idlj -fall Stock.idl

C:\Documents and Settings\User\Desktop\stock>javac StockServer.java StockApp/*.java
C:\Documents and Settings\User\Desktop\stock>javac StockClient.java StockApp/*.java
C:\Documents and Settings\User\Desktop\stock>start orbd -ORBInitialPort 1050 -ORBInitialHost localhost
C:\Documents and Settings\User\Desktop\stock>start java StockServer -ORBInitialPort 1050 -ORBInitialHost localhost
C:\Documents and Settings\User\Desktop\stock> java StockClient -ORBInitialPort 1050 -ORBInitialHost localhost
Obtained a handle on server object:
IOR:000000000000001749444c3a53746f636b4170702f53746f636b3a312e30000000000001000
000000000008200010200000000a3132372e302e302e3100044500000031afabcb000000002098
d59eb400000001000000000000000100000008526f6f74504f4100000000080000000100000001
400000000000002000000010000002000000000000100010000000205010001000100200001010
900000001000101000000026000000020002
value is: 5000
C:\Documents and Settings\User\Desktop\stock>

Input: None

Output: 5000

Program 10

Develop a middleware component for retrieving Bank Balance using CORBA.

Problem Definition

Develop a middleware component for retrieving Bank Balance using CORBA. Exchange information using CORBA

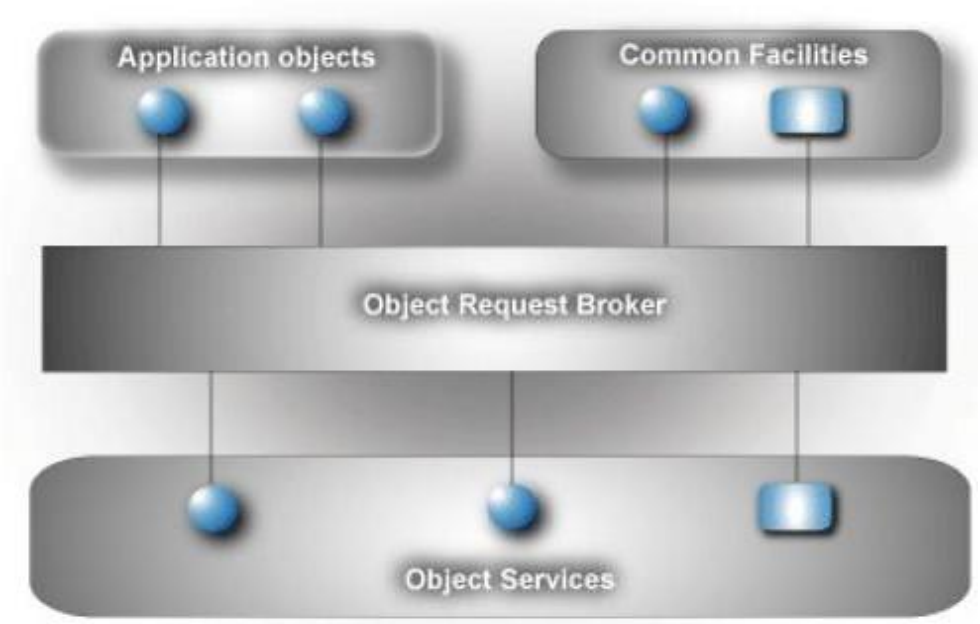
Problem Description

What is CORBA?

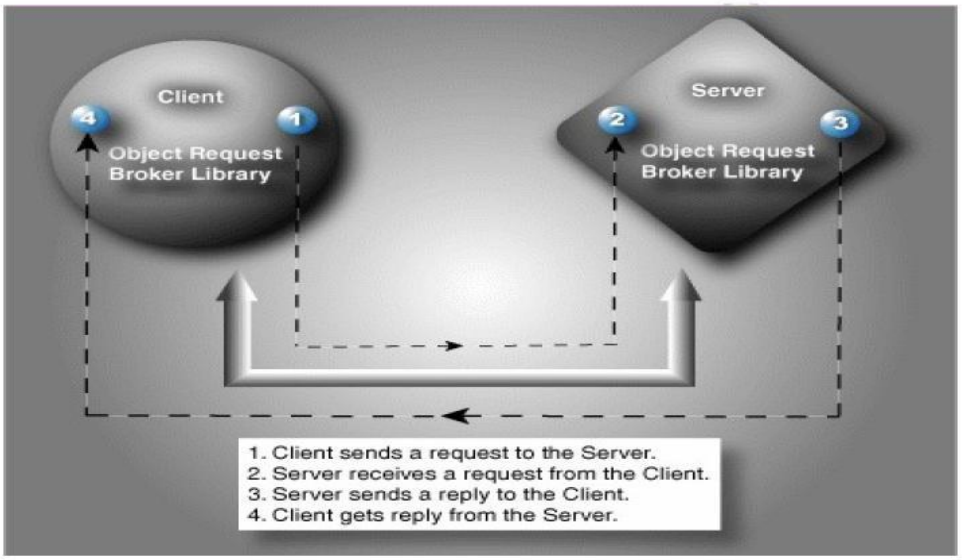
The Common Object Request Broker Architecture (CORBA) is a standard developed by the Object Management Group (OMG) to provide interoperability among distributed objects. CORBA is the world's leading middleware solution enabling the exchange of information, independent of hardware platforms, programming languages, and operating systems. CORBA is essentially a design specification for an Object Request Broker (ORB), where an ORB provides the mechanism required for distributed objects to communicate with one another, whether locally or on remote devices, written in different languages, or at different locations on a network.

The CORBA Interface Definition Language, or IDL, allows the development of language and location-independent interfaces to distributed objects. Using CORBA, application components can communicate with one another no matter where they are located, or who has designed them. CORBA provides the location transparency to be able to execute these applications.

CORBA is often described as a "software bus" because it is a software-based communications interface through which objects are located and accessed. The illustration below identifies the primary components seen within a CORBA implementation.



Data communication from client to server is accomplished through a well-defined object oriented interface. The Object Request Broker (ORB) determines the location of the target object, sends a request to that object, and returns any response back to the caller. Through this object-oriented technology, developers can take advantage of features such as inheritance, encapsulation, polymorphism, and runtime dynamic binding. These features allow applications to be changed, modified and re-used with minimal changes to the parent interface. The illustration below identifies how a client sends a request to a server through the ORB:



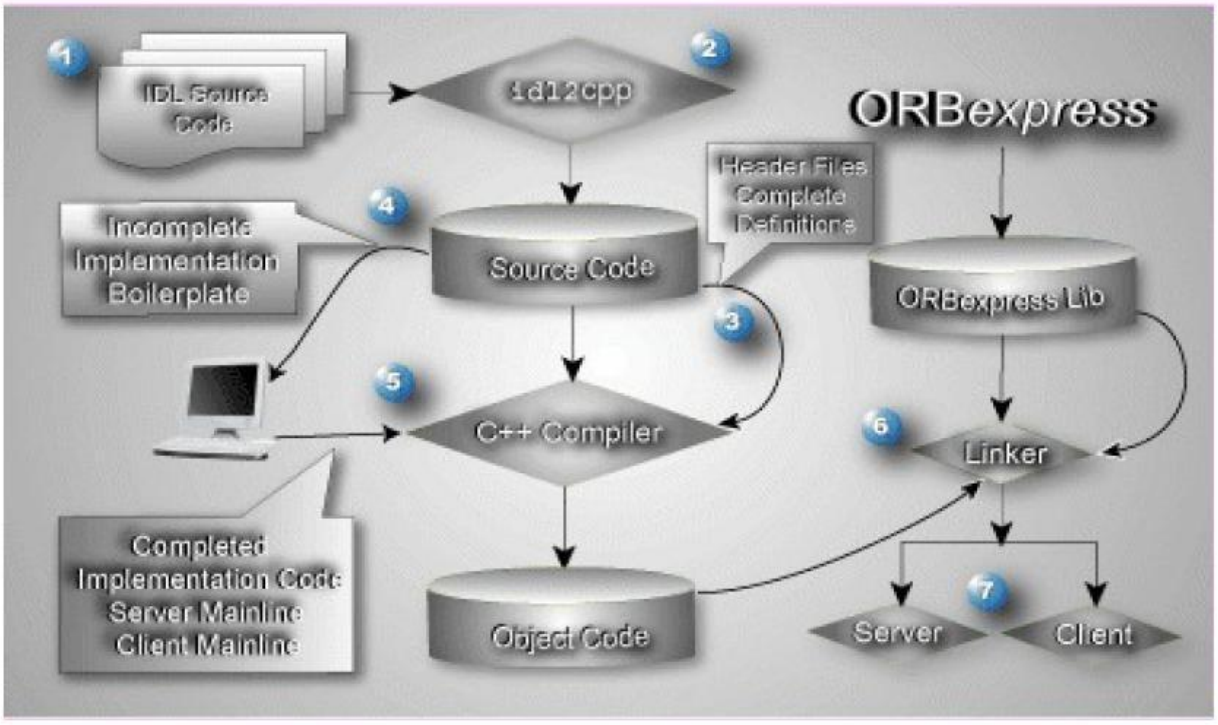
Interface Definition Language (IDL)

A cornerstone of the CORBA standards is the Interface Definition Language. IDL is the OMG standard for defining language-neutral APIs and provides the platform-independent delineation of the interfaces of distributed objects. The ability of the CORBA environments to provide consistency between clients and servers in heterogeneous environments begins with a standardized definition of the data and operations constituting the client/server interface. This standardization mechanism is the IDL, and is used by CORBA to describe the interfaces of objects.

IDL defines the modules, interfaces and operations for the applications and is not considered a programming language. The various programming languages, such as Ada, C++, or Java, supply the implementation of the interface via standardized IDL mappings.

Application Development Using ORBexpress

The basic steps for CORBA development can be seen in the illustration below. This illustration provides an overview of how the IDL is translated to the corresponding language (in this example, C++), mapped to the source code, compiled, and then linked with the ORB library, resulting in the client and server implementation.



The basic steps for CORBA development include:

1. Create the IDL to Define the Application Interfaces

The IDL provides the operating system and programming language independent interfaces to all services and components that are linked to the ORB. The IDL specifies a description of any services a server component exposes to the client. The term "IDL Compiler" is often used, but the IDL is actually translated into a programming language.

2. Translate the IDL

An IDL translator typically generates two cooperative parts for the client and server implementation, stub code and skeleton code. The stub code generated for the interface classes is associated with a client application and provides the user with a well-defined Application Programming Interface (API). In this example, the IDL is translated into C++.

3. Compile the Interface Files

Once the IDL is translated into the appropriate language, C++ in this example, these interface files are compiled and prepared for the object implementation.

4. Complete the Implementation

If the implementation classes are incomplete, the spec and header files and complete bodies and definitions need to be modified before passing through to be compiled. The output is a complete client/server implementation.

5. Compile the Implementation
Once the implementation class is complete, the client interfaces are ready to be used in the client application and can be immediately incorporated into the client process. This client process is responsible for obtaining an object reference to a specific object, allowing the client to make requests to that object in the form of a method call on its generated API.

6. Link the Application
Once all the object code from steps three and five have been compiled, the object implementation classes need to be linked to the C++ linker. Once linked to the ORB library, in this example, ORBexpress, two executable operations are created, one for the client and one for the server.

7. Run the Client and Server
The development process is now complete and the client will now communicate with the server. The server uses the object implementation classes allowing it to communicate with the objects created by the client requests.

In its simplest form, the server must perform the following:
• Create the required objects.
• Notify the CORBA environment that it is ready to receive client requests.
• Process client requests by dispatching the appropriate servant.

Overview
The **Common Object Request Broker Architecture** (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together, i.e. it supports multiple platforms.

Advantages
CORBA features:

- Language Independence
- OS Independence
- Freedom from Technologies
- Strong Data Typing
- High Tune-ability
- Freedom From Data Transfer Details
- Compression

Strengths of CORBA:

- Cross-platform and multi-vendor. Very strong support in Unix and mainframe systems.
- Is an industry standard.
- Some really excellent implementations are available for free.
- Many free versions are Open-Source
- A wider range of programming language bindings.
- Simpler programming interface.
- ALL objects/interfaces can be called dynamically at run time through a data-driven interface: CORBA DII (Dynamic Invocation Interface).
- Multiple inheritance in interfaces. (COM has single inheritance between interfaces, but discourages its use, favoring multiple interfaces instead.)

5.3 Architecture

CORBA is a mechanism in software for normalizing the method-call semantics between application objects that reside either in the same address space (application) or remote address space (same host, or remote host on a network). Version 1.0 was released in October 1991. CORBA uses an interface definition language (IDL) to specify the interfaces that objects will present to the outside world. CORBA then specifies a *mapping* from IDL to a specific implementation language like C++ or Java. Standard mappings exist for Ada, C, C++, Lisp, Ruby, Smalltalk, Java, COBOL, PL/I and Python. There are also non-standard mappings for Perl, Visual Basic, Erlang, and Tcl implemented by object request brokers (ORBs) written for those languages.

The CORBA specification dictates that there shall be an ORB through which the application interacts with other objects. In practice, the application simply initializes the ORB, and accesses an internal *Object Adapter* which maintains such issues as reference counting, object (and reference) instantiation policies, object lifetime policies, etc. The Object Adapter' is used to register instances of the *generated code classes*. Generated code classes are the result of compiling the user IDL code, which translates the high-level interface definition into an OS- and language-specific class base for use by the user application. This step is necessary in order to enforce the CORBA semantics and provide a clean user process for interfacing with the CORBA infrastructure.

Some IDL language mappings are "more hostile" than others. For example, due to the very nature of Java, the IDL-Java Mapping is rather straightforward and makes usage of CORBA very simple in a Java application. The C++ mapping is not trivial, but accounts for all the features of CORBA (e.g. exception handling). The C mapping is even stranger (since C is not an object-oriented language), but it does make sense and properly handles the RPC semantics.

A language mapping requires the developer ("user" in this case) to create some IDL code that represents the interfaces to his objects. Typically, a CORBA implementation comes with a tool called an IDL compiler which converts the user's IDL code into some language specific generated code. A traditional compiler then compiles the generated code to create the linkable-object files for the application. This diagram illustrates how the generated code is used within the CORBA infrastructure:

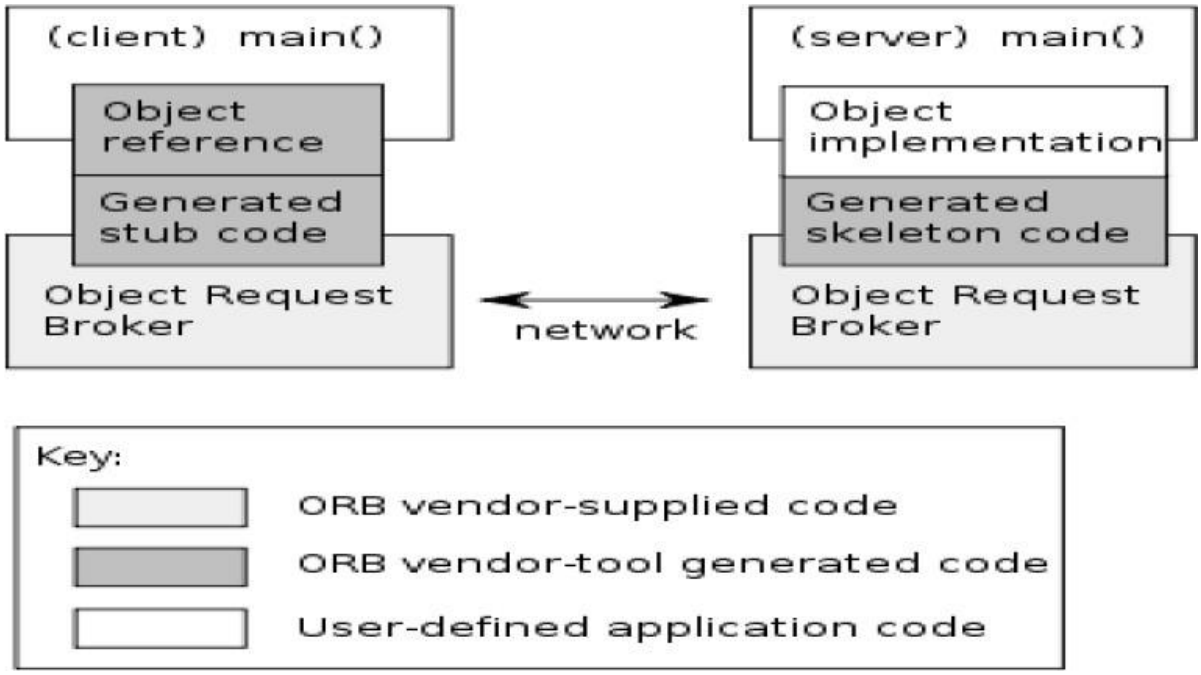


Figure: Illustration of the auto-generation of the infrastructure code from an interface defined using the CORBA IDL.

Program

Bank.idl

```
module BankApp
{
    interface Bank
    {
      long bal(in long a);
    };
};
```

BankServer.java

```
import BankApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import java.sql.*;
import java.util.Properties;

class BankImpl extends BankPOA {
private ORB orb;
```

```
  public void setORB(ORB orb_val) {
    orb = orb_val;
   }
     public int bal(int a)
     {
           Connection con;
      PreparedStatement pt;
      ResultSet rs;
     int acc = a;
      int var1=0;
try
{
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  con=DriverManager.getConnection("Jdbc:Odbc:bank");
    pt = con.prepareStatement("select bal from bankDB where acc=?");
       pt.setInt(1, acc);
      rs = pt.executeQuery();
          while (rs.next()) {
        var1 = rs.getInt("bal");
          }
}
catch(ClassNotFoundException e){}
catch(SQLException p){}
return var1;

}
}

public class BankServer {

  public static void main(String args[]) {
    try{
        ORB orb = ORB.init(args, null);
      POA rootpoa =POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      rootpoa.the_POAManager().activate();
      BankImpl bankImpl = new BankImpl();
      bankImpl.setORB(orb);
      org.omg.CORBA.Object ref =rootpoa.servant_to_reference(bankImpl);
      Bank href = BankHelper.narrow(ref);
      org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
         NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
      String name = "Bank";
      NameComponent path[] = ncRef.to_name( name );
      ncRef.rebind(path, href);
      System.out.println("BankServer ready and waiting ...");
      orb.run();
```

```
            }
              catch (Exception e) {
                System.err.println("ERROR: " + e);
                e.printStackTrace(System.out);
              }
              System.out.println("BankServer Exiting ...");
                    }
          }
```

BankClient.java

```
import BankApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class BankClient
{
  static Bank bankImpl;
  public static void main(String args[])
    {
     try{
          ORB orb = ORB.init(args, null);
       org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
       NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
       String name = "Bank";
       bankImpl = BankHelper.narrow(ncRef.resolve_str(name));

System.out.println("Obtained a handle on server object: " + bankImpl);
          int acc=123;
       System.out.println("balance is: "+bankImpl.bal(acc));
       } catch (Exception e) {
         System.out.println("ERROR : " + e) ;
            e.printStackTrace(System.out);
            }
    }
}
```

Problem Validation:

Compiling and Running:

C:\Documents and Settings\User\Desktop\Bank>idlj -fall Bank.idl

C:\Documents and Settings\User\Desktop\Bank>javac BankServer.java BankApp/*.java

C:\Documents and Settings\User\Desktop\Bank>javac BankClient.java BankApp/*.java

C:\Documents and Settings\User\Desktop\Bank>start orbd -ORBInitialPort 1050 -ORBInitialHost localhost

C:\Documents and Settings\User\Desktop\Bank>start java BankServer -ORBInitialPort 1050 -ORBInitialHost localhost

C:\Documents and Settings\User\Desktop\Bank> java BankClient -ORBInitialPort 1050 -ORBInitialHost localhost
Obtained a handle on server object: IOR:000000000000001549444c3a42616e6b4170702f
42616e6b3a312e3000000000000000010000000000000820001020000000000a3132372e302e30
3100045700000031afabcb000000002098dabfb10000000100000000000000010000000008526f6f7
504f41000000000800000001000000001400000000000002000000010000002000000000000100
0000000205010001000100200001010900000001000101000000002600000000200002
balance is: 100000

C:\Documents and Settings\User\Desktop\Bank>

Input: None

Output: 100000

Annexure – I

List of programs according to O.U. curriculum

Code: BIT432 **MWT LAB**

Instruction	3	Periods per week
Duration of University Examination	3	Hours
University Examination	50	Marks
Sessional	25	Marks

1. Create a distributed name server (like DNS) RMI
2. Create a Java Bean to draw various graphical shapes and display it using or without using BDK
3. Develop an Enterprise Java Bean for student Information System.
4. Develop an Enterprise Java Bean for Library operations.
5. Create an Active-X control for Timetable.
6. Develop a component for converting the currency values using COM / .NET
7. Develop a component for browsing CD catalogue using COM / .NET
8. Develop a component for retrieving information from message box using DCOM/.NET
9. Develop a middleware component for retrieving Stock Market Exchange information using CORBA.
10. Develop a middleware component for retrieving Bank Balance using CORBA.

Suggested Reading:

1. Robert Orfali, Dan Harkey and Jeri Edwards, “The Essential Client/ server Survival Guide”, Galgotia publications Pvt. Ltd., 2002.
2. Tom Valesky, “Enterprise Java Beans”, Pearson Education, 2002.
3. Jason Pritchard. “COM and CORBA side by side”, Addison Wesley, 2000.
4. Jesse Liberty, “Programming C#”, 2nd Edition, O’Reilly press, 2002.
5. Struts: the complete reference By James Holmes Edition: 2, illustrated Published by McGraw-Hill Professional, 2006(added).
6. Java Servlet Programming By Jason Hunter, William Crawford Edition: 2, illustrated Published by O’Reilly, 2001 (added)
7. Arno Puder, kay Romere and Frank pilhofer. Distributed Systems Architecture, Morgan Kaufman 2006
8. Mowbray, “ Inside CORBA”, Pearson Education, 2002.
9. Jeremy Rosenberger, “Teach yourself CORBA in 14 days”, Tec media, 2000.