

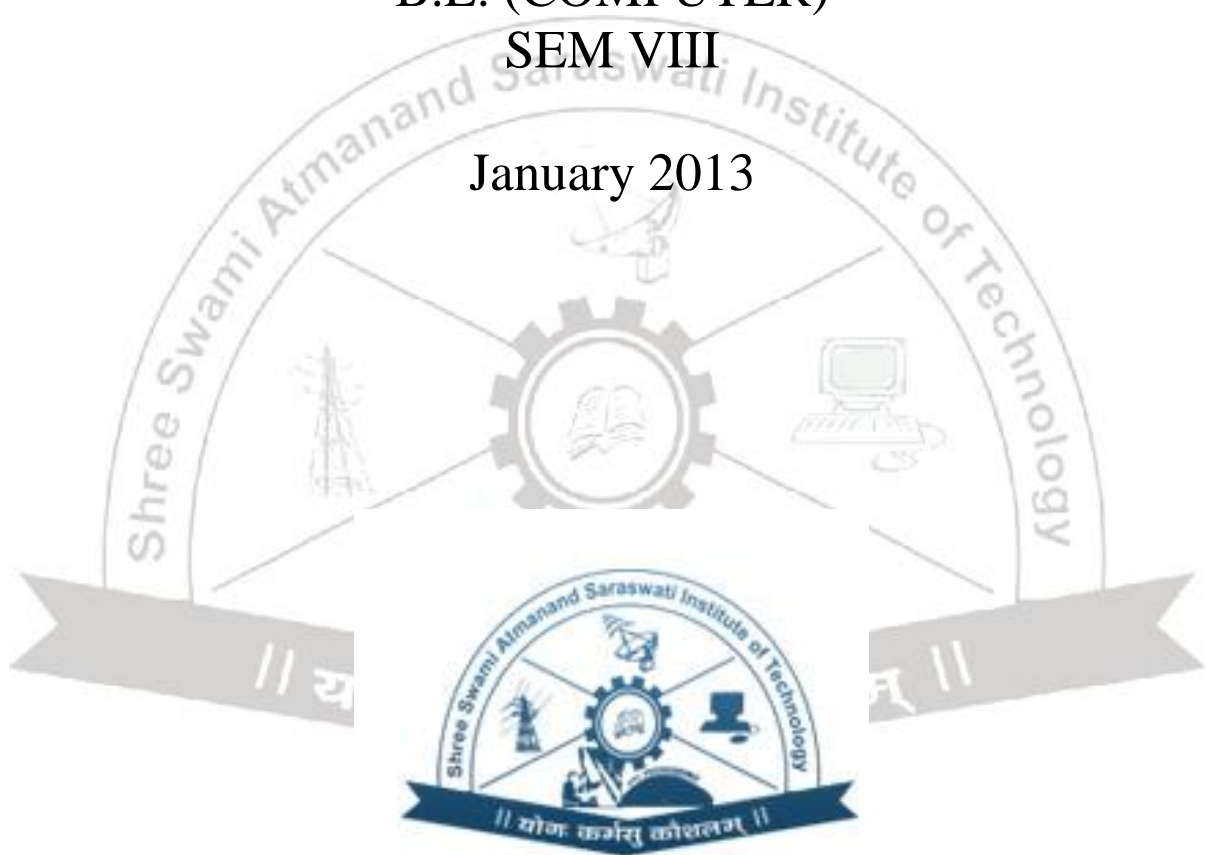
Laboratory Manual

For

DISTRIBUTED SYSTEMS (180701)

**B.E. (COMPUTER)
SEM VIII**

January 2013



Shree Swami Atmanand Saraswati Institute of Technology

Shree Swami Atmanand Vidya Sankul,

Kapodra, Varachha Road,

Surat – 395006

www.ssasit.org

Index

Sr. No.	Topic	Page No.
1.	RPC Programming <ol style="list-style-type: none"> 1. Implement PI calculation Service 2. Implement Calculator Service using SUN RPC. 3. Implement RPC programming on windows using DCOM. 	3
2.	RMI Programming <ol style="list-style-type: none"> 1. Implementation of “Hello Word” Service using JAVA RMI 2. Implementation of “Calculator” Service using JAVA RMI 3. Implement RMI –IIOP Programming. 	12
3.	Thread Programming in Java <ol style="list-style-type: none"> 1. Write an application that executes two threads. One thread display “HELLO WOLD” every 1000 milliseconds and another thread display “How Are You” every 2000 milliseconds. Create the threads by implementing Runnable interface. 2. Implement Multithreaded Echo server using Socket. 3. Implement producer consumer example. 	14
4.	Mobile Agent (IBM’s Aglet) Programming	19
5.	Implement CORBA File	29
6.	Incrementing a counter in shared memory. Tools/ Apparatus: Unix/Linux C Programming Environment	34
7.	Implement Network File System (NFS)	35
8.	Creation of A BPEL (Business Process Execution Language) Module and a Composite Application	36
9.	Web Service Programming	47

EXPERIMENT – 1**AIM:****RPC Programming**

4. Implement PI calculation Service
5. Implement Calculator Service using SUN RPC.
6. Implement RPC programming on windows using DCOM.

(1) Implement PI calculation Service

Create a pi.x file

Code of pi.x

```
program PIPROG {  
    version CALCU_PIVERS {  
        double CALCU_PI() = 1;  
    } = 1;  
} = 0x39876543;
```

Compile a pi.x file using,

rpcgen -a pi.x

where: option -a tells rpcgen to generate all of the supporting files

It will generate following files.

- pi_clnt.c -- the client stub
- pi_svc.c -- the server stub
- pi.h -- the header file that contains all of the XDR types generated from the specification
- makefile.pi -- makefile for compiling all of the client and server code
- pi_client.c -- client skeleton, need to be modified
- pi_server.c -- server skeleton, need to be modified

pi_server.c

```
#include <rpc/rpc.h>  
#include "pi.h"  
  
double *calcu_pi_1_svc(void *argp, struct svc_req *rqstp){  
    static double pi;  
    double sum = 0;  
    int i;  
    int sign;  
    for (i=1; i<10000000; i++){
```

```

        sign = (i+1) % 2;
        if ( sign == 0 )
            sign = 1;
        else
            sign = -1;
        sum += 1 / (2*(double)i - 1) * (double)sign;
    }
    pi = 4 * sum;
    return (&pi);
}

```

File Name: pi_client.c

```

#include <stdio.h>
#include <rpc/rpc.h>
#include "pi.h"

main(int argc, char *argv[]){
    CLIENT * clnt;
    double *result_1;
    char *host;
    char * calcu_pi_1_arg;

    if (argc < 2){
        printf("usage: %s server_host\n", argv[0]);
        exit(1);
    }

    host = argv[1]; /* server host name */
    clnt = clnt_create(host, PIPROG, CALCU_PIVERS, "tcp");
    if (clnt == (CLIENT *) NULL){
        /* Couldn't establish connection with server. Print error message and die. */
        clnt_pcreateerror(host);
        exit(1);
    }

    /* call remote procedure on the server side */

    result_1 = calcu_pi_1((void *)&calcu_pi_1_arg, clnt);
    if (result_1 == (double *) NULL){
        /* An error occurred while calling the server. Print error message and die.*/

        clnt_perror (clnt, "call failed");
        exit (1);
    }

    printf("PI value is= %f\n" , *result_1);
    clnt_destroy(clnt);
}

```

```
    exit(0);  
}
```

Compile client and server file :

```
$gcc pi_client.c pi_clnt.c -o pi_client -lnsl
```

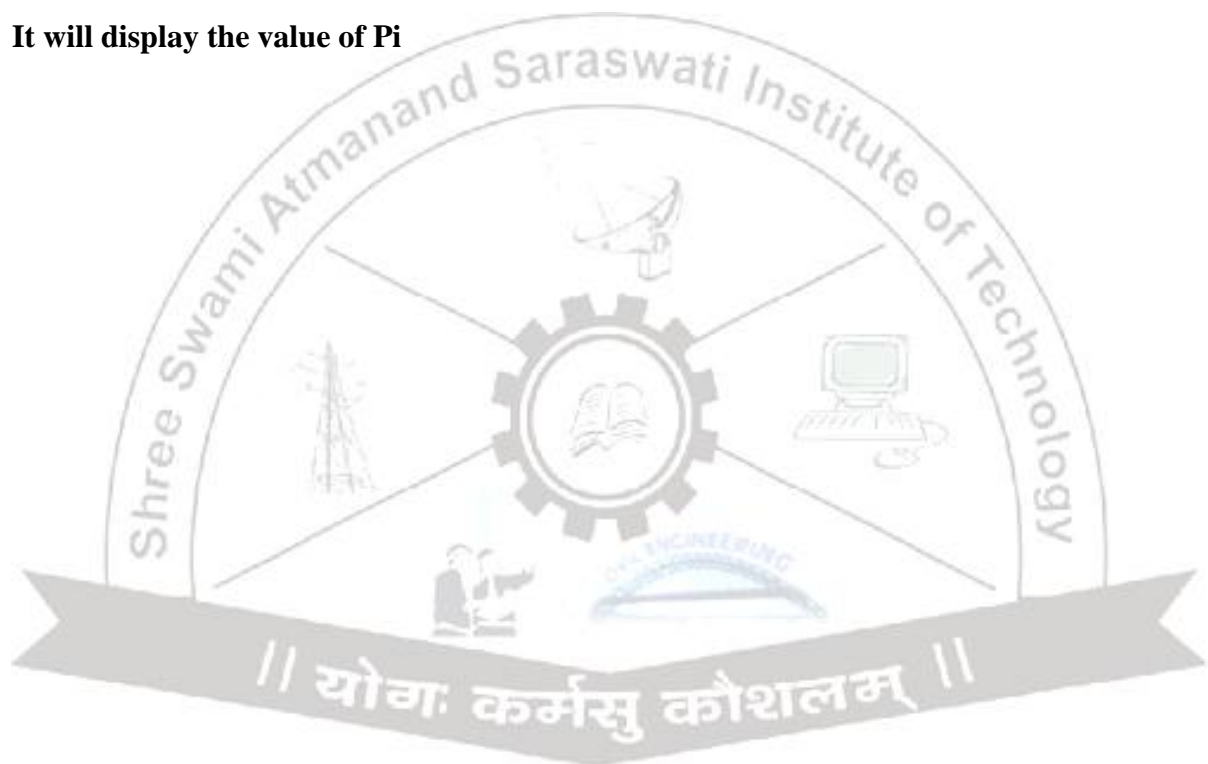
```
$gcc pi_server.c pi_svc.c -o pi_server -lnsl
```

Start server:

```
$/pi_server &
```

```
$/pi_client 10.10.0.30
```

It will display the value of Pi



(2) Implement calculator Service using SUN RPC.

File name: Calc.x

```
struct operands{
    int x;
    int y;
};

program Calc_PROG {
    version Calc_VERSION {
        int ADD(operands) = 1;
        int SUB(operands) = 2;
        int MULT(operands) = 3;
        int DIVI(operands) = 4;
    } = 1;
} = 0x25978443;
```

Server Programm:

Calc_server.c

```
#include "Calc.h"
#include <rpc/rpc.h>

int * add_1_svc(operands *argp, struct svc_req *rqstp){
    static int result;
    result = argp->x + argp->y;
    return (&result);
}

int * sub_1_svc(operands *argp, struct svc_req *rqstp){
    static int result;
    result = argp->x - argp->y;
    return (&result);
}

int * mult_1_svc(operands *argp, struct svc_req *rqstp){
    static int result;
    result = argp->x * argp->y;
    return (&result);
}

int * divi_1_svc(operands *argp, struct svc_req *rqstp){
    static int result;
    result = argp->x / argp->y;
    return (&result);
}
```

Client program:**Calc_client.c**

```

#include <rpc/rpc.h>
#include <stdio.h>
#include "Calc.h"

int main( int argc, char *argv[]){
    CLIENT *clnt;
    char o;
    Operands ops;
    int *result;

    if (argc!=5){
        fprintf(stderr,"Usage: %s hostname num1 num\n",argv[0]);
        exit(0);
    }

    clnt = clnt_create(argv[1], Calc_PROG, Calc_VERSION, "udp");

    if (clnt == (CLIENT *) NULL){
        clnt_pcreateerror(argv[1]);
        exit(1);
    }

    ops.x = atoi(argv[2]);
    ops.y = atoi(argv[3]);
    o=argv[4];

    If(o=='+'){
        result = add_1(&ops,clnt);
    }else if(o=='-'){
        result = sub_1(&ops,clnt);
    }else if(o=='*'){
        result = mult_1(&ops,clnt);
    }else if(o=='/'){
        result = divi_1(&ops,clnt);
    }

    if (result==NULL) {
        fprintf(stderr,"Trouble calling remote procedure\n");
        exit(0);
    }

    printf("%d  % c  %d = %d\n", op.x , o, op.y, *result);
    return(0);
}

```


(3) Implement RPC programming on windows using DCOM

Step 1: create .idl file

Name of file: DoRPC.idl

```
[uuid(F0C37BD0-0D1B-4513-8C51-EC0D699740C0), version(1.0),implicit_handle(handle_t
hDoRPCBinding) ]
```

/* Arguments:-

- (1) A unique identifier that distinguishes this interface from other interfaces.
- (2) This is version 1.0 of this interface.
- (3) This interface will use an implicit binding handle named hDoRPCBinding.

***/**

```
interface DoRPC // The interface is named DoRPC {
    void Show( [in, string] const char* szMsg); //name of function
}
```

Note: - As a companion to this IDL file we also need to write an Application Configuration File (.acf) with the following text and save it as DoRPC.acf in the same directory as the IDL file.

Step 2: create .acf file

Name of file: DoRPC.acf

```
[implicit_handle(handle_t hDoRPCBinding)]
```

//This interface will use an implicit binding handle named hDoRPCBinding.

```
interface DoRPC // The interface is named DoRPC{
}
```

Step 3: Compile the file

Use MIDL compiler to generate the source code, to use in your client/server application. To generate the files invoke the tool MIDL.exe as follows:

MIDL /app_config /no_cpp DoRPC.idl

This will generate three files for you: DoRPC.h, DoRPC_c.c, and DoRPC_s.c. The files with .c extensions are the source code files generated for the client and server side implementation of the interface; they can be distinguished by the _c and the _s appended to filename.

Step 4: server file: DoRPC_Server.cpp

```
#include <stdio.h>
#include "..\RPC1_IDL\DoRPC.h"
int main(){
    RPC_STATUS status;
```



```
Status=RpcServerUseProtseqEp((unsignedchar*)"ncacn_ip_tcp"),RPC_C_PROTSEQ_MAX_REQS_DEFAULT,(unsigned char*)"9191",NULL);
```

/* Arguments:-

- (1) Use TCP/IP protocol,
- (2) Backlog q length for TCP/IP,
- (3) TCP/IP port to use, No security. */

```
if(status){
    exit(status);
}
Status = RpcServerRegisterIf(DoRPC_v1_0_s_ifspec, NULL,NULL);
// Registers the DoRPC interface.
```

/*Arguments:-

- (1)Interface to register.
- (2)Use the MIDL generated entry-point vector.
- (3) Use the MIDL generated entry-point vector. */

```
if (status)
    exit(status);

// Start to listen for remote procedure calls for all registered interfaces.
// This call will not return until RpcMgmtStopServerListening is called.

status = RpcServerListen( 1, RPC_C_LISTEN_MAX_CALLS_DEFAULT, FALSE);
```

/*Arguments:-

- (1) Recommended minimum number of threads.
- (2) Recommended maximum number of threads.(3) Start listening now. */

```
if (status){
    exit(status);
}
return 0;
}

void* __RPC_USER midl_user_allocate(size_t size){// Memory allocation function for RPC.
    return malloc(size);
}

void __RPC_USER midl_user_free(void* p){ // Memory deallocation function for RPC.
    free(p);
}

void Show(const unsigned char* szMsg){ // Now we implement our server function.
    printf("%s\n",szMsg);
}
```

Client File Name: DoRPC_Client.cpp

```
#include <stdio.h>
#include "..\RPC1_IDL\DoRPC.h"
```

```
int main(){
    RPC_STATUS status;
    unsigned char* szStringBinding = NULL;
```

// Creates a string binding handle. This function formats the past values in a predefined format for use by RPC. Just like printf

```
status = RpcStringBindingCompose(NULL,(unsigned char*)"ncacn_ip_tcp"),(unsigned
char*)"localhost", (unsigned char*)"9191", NULL, &szStringBinding);
```

/*Arguments:-

- (1) UUID to bind to.
- (2) Use TCP/IP protocol.
- (3) TCP/IP network // the same machine as server
- (4) TCP/IP port to use.
- (5) Protocol dependent network options to use.(6) String binding output.

***/**

```
if (status)
    exit(status);
```

// Validates the format of the string binding handle and converts it to a binding handle. Connection is not done here either.

```
status = RpcBindingFromStringBinding(szStringBinding, &hDoRPCBinding);
```

/* Arguments:-

- (1) The string binding to validate.
- (2) Put the result in the implicit binding handle defined in the IDL file.

***/**

```
if(status){
    exit(status);
}
```

RpcTryExcept{

// Calls the RPC function. The hDoRPCBinding binding handle is used implicitly. Connection is done here.

```
const unsigned char szMsg[] = "Hello World!";
Show(szMsg);
```

```
}
```

RpcExcept(1){

```
printf("Runtime exception occurred: %d\n",RpcExceptionCode());
```

```
}
```

RpcEndExcept

```
status = RpcStringFree(&szStringBinding);
// Free the memory allocated by a string. String to be freed.

if(status){
    exit(status);
}

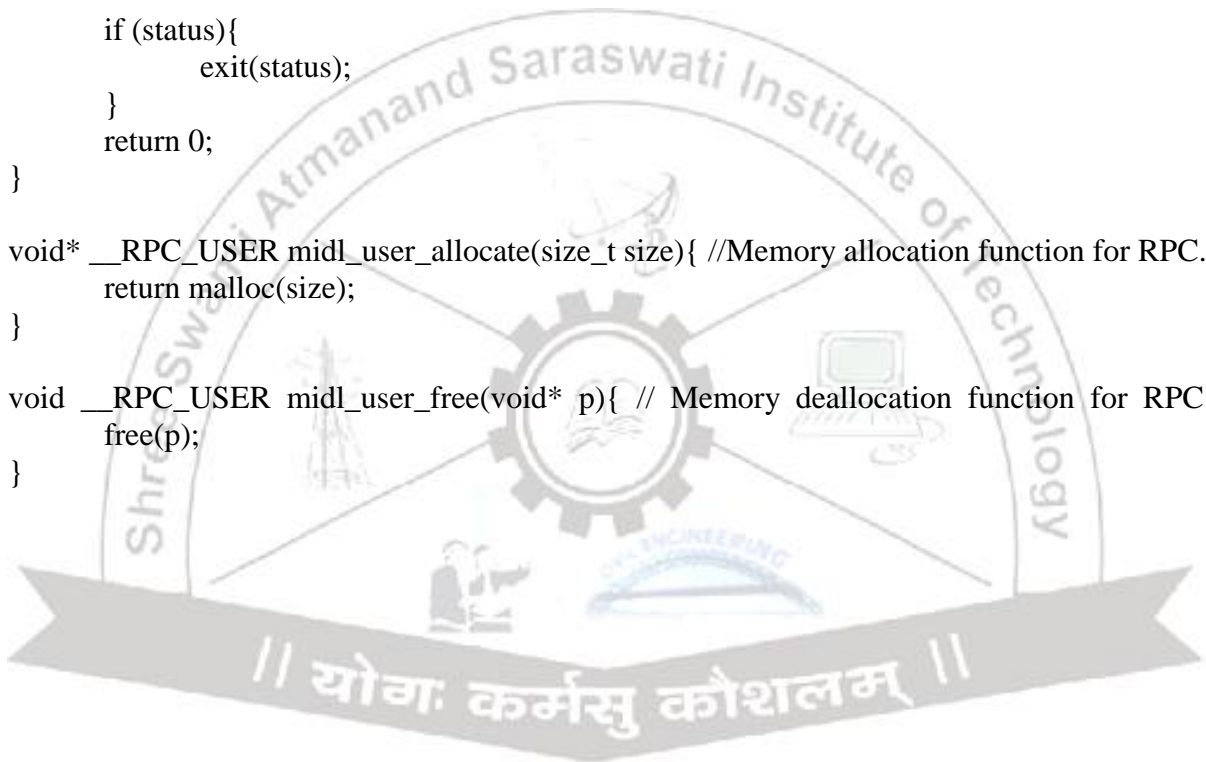
// Releases binding handle resources and disconnects from the server.

status = RpcBindingFree(&hDoRPCBinding);
// Frees the implicit binding handle defined in the IDL file.

if (status){
    exit(status);
}
return 0;
}

void* __RPC_USER midl_user_allocate(size_t size){ //Memory allocation function for RPC.
    return malloc(size);
}

void __RPC_USER midl_user_free(void* p){ // Memory deallocation function for RPC.
    free(p);
}
```



EXPERIMENT – 2**AIM: RMI Programming**

1. Implementation of “Hello Word” Service using JAVA RMI
2. Implementation of “Calculator” Service using JAVA RMI
3. Implement RMI –IIOP Programming.

(1) Implementation of “Hello Word” service using JAVA RMI

- **Step of developing an RMI System:**

1. Define the remote interface
Here **SampleServer** is interface
2. Develop the remote object by implementing the remote interface.
Implement **SampleServerImpl** class
3. Develop the client program.
Create **SampleClient** class, default **RMI port 1099**
4. Compile the Java source files.
Compile **SampleServer.java**, **SampleServerImpl.java**, **SampleClient.java** files
Using **javac** command
5. Generate the client stubs and server skeletons.
Using **rmic** (RMI Compiler) compile interface class **SampleServerImpl**,
which generates **SampleServerImpl_stub.class**
6. Start the RMI registry.
Use **start rmiregistry**
7. Start the remote server objects.
Use **java -Djava.security.policy=policy.all SampleServerImpl**
8. Run the client
Use **java -Djava.security.policy=policy.all SampleClient**

(2) Implementation of “Calculator” service using JAVA RMI**(3) Implementation RMI-IIOP Programming**

- **Step of developing an RMI-IIOP System:**

1. Define the remote interface
Here **HelloInterface** is a remote interface
2. Develop the remote object by implementing the remote interface.
Implement **HelloImpl** class that implements **HelloInterface**
3. Develop the client program.
Create **HelloClient** class
4. Develop the server program.
Create **HelloServer** class
5. Compile the Java source files.
Compile **HelloInterface.java** **HelloServer.java** **HelloClient.java**
HelloImpl.java Files using **javac** command

6. Generate the client stubs and server skeletons.

rmic -iiop HelloImpl

_HelloInterface_Stub.class - the client stub

_HelloImpl_Tie.class - the server skeleton

7. Start the orbd tool.

start orbd -ORBInitialPort 1050

8. Start the server objects.

java -classpath

-Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory

-Djava.naming.provider.url=iiop://localhost:1050 HelloServer



EXPERIMENT – 3**AIM: Thread Programming in Java**

4. Write an application that executes two threads. One thread display “HELLO WOLD” every 1000 milliseconds and another thread display “How Are You” every 2000 milliseconds. Create the threads by implementing Runnable interface.
5. Implement Multithreaded Echo server using Socket.
6. Implement producer consumer example.

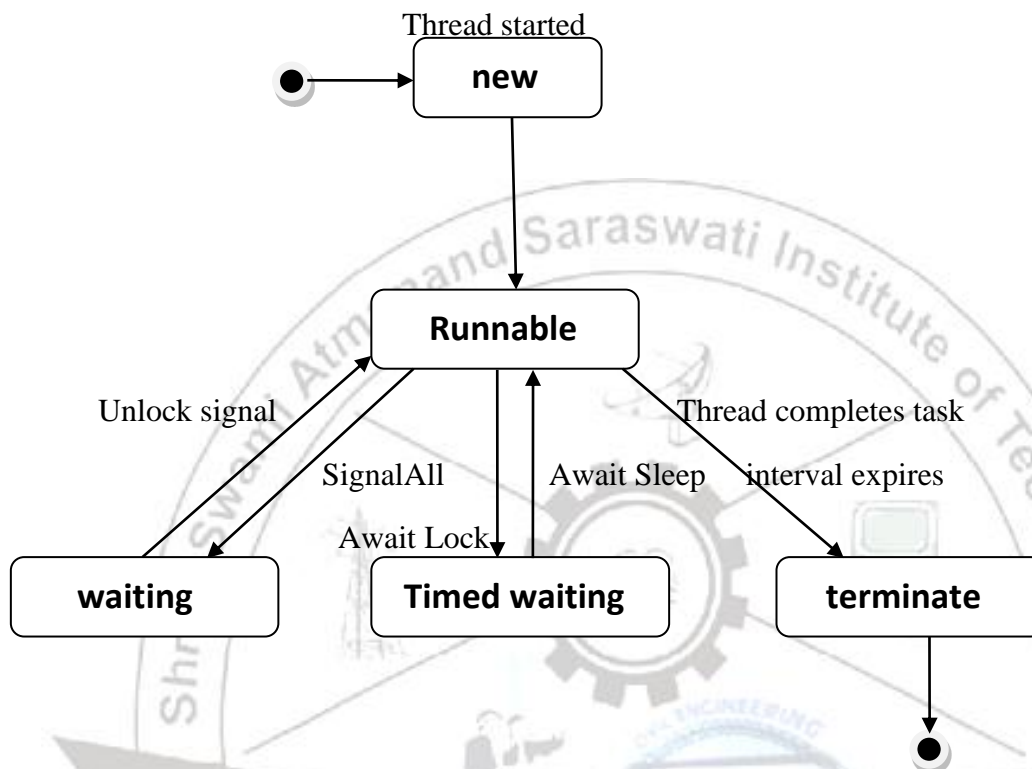
Exercise

(1) Differentiate Thread and Process.

Thread	Process
Threads (Light weight Processes) share the address space of the process that created it	processes have their own address.
Threads have direct access to the data segment of its process	processes have their own copy of the data segment of the parent process
Threads can directly communicate with other threads of its process	processes must use inter process communication to communicate with sibling processes
Threads have almost no overhead	processes have considerable overhead.
New threads are easily created	new processes require duplication of the parent process.
Threads can exercise control over threads of the same process	processes can only exercise control over child processes
Changes to the main thread (cancellation, priority change, etc.) may affect the behavior of the other threads of the process	changes to the parent process do not affect child processes

(2) Discuss the life cycle of thread.

Thread LifeCycle:



Thread has Five different states:

- **New:** A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.
- **Runnable:** After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting:** Sometimes a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.

- **Timed waiting:** A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated:** A runnable thread enters the terminated state when it completes its task or otherwise terminates

(3) Write the different way to create thread using java.

- By Implementing Runnable interface

To create thread using Runnable interface, a class first need to instantiate an object of type Thread from within that class. Thread defines several constructors. The one that we will use is shown here:

Thread(Runnable threadOb, String threadName);

Here threadOb is an instance of a class that implements the Runnable interface and the name of the new thread is specified by threadName.

After the new thread is created, we need to start the execution of thread. It is done using its **start()** method, which is declared within Thread.

void start();

we can define the code that constitutes the new thread inside **run()** method. **run()** method can call other methods, use other classes, and declare variables, just like the main thread can.

public void run()

- By extending the Thread class

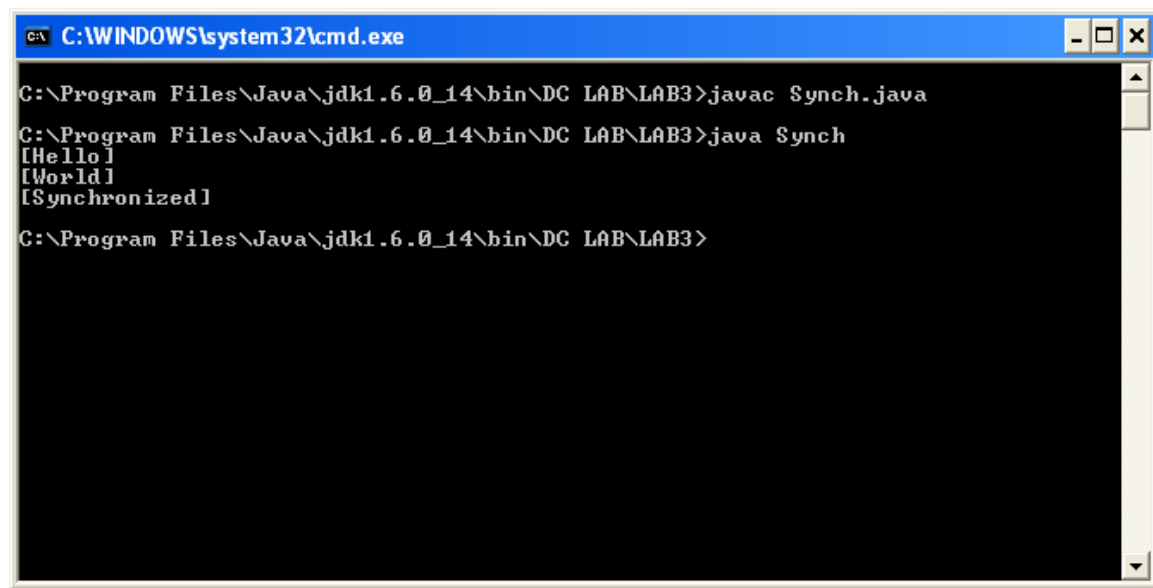
To create a new thread first need to extend **Thread** Super class and create an instance of that class. The newly created(extended) class must override the **run()** method, which is the entry point for the new thread. It must also call **start()** to begin execution of the new thread. Same as Runnable Interface

(4) Define Thread Synchronization with example.

```
class Callme
{
    void call(String msg) {
        System.out.print("[ " + msg);
        try {
            Thread.sleep(1000);
        }
        catch (InterruptedException e) {
            System.out.println("Interrupted");
        }
        System.out.println("]");
    }
}

class Caller implements Runnable {
    String msg;
    Callme target;
    Thread t;
    public Caller(Callme targ, String s) {
        target = targ;
        msg = s;
        t = new Thread(this);
        t.start();
    }
    public void run() {
        synchronized(target) {
            target.call(msg);
        }
    }
}

class Synch {
    public static void main(String args[]) {
        Callme target = new Callme();
        Caller ob1 = new Caller(target, "Hello");
        Caller ob2 = new Caller(target, "Synchronized");
        Caller ob3 = new Caller(target, "World");
        try {
            ob1.t.join();
            ob2.t.join();
            ob3.t.join();
        }
        catch(InterruptedException e) {
            System.out.println("Interrupted");
        }
    }
}
```

OUTPUT

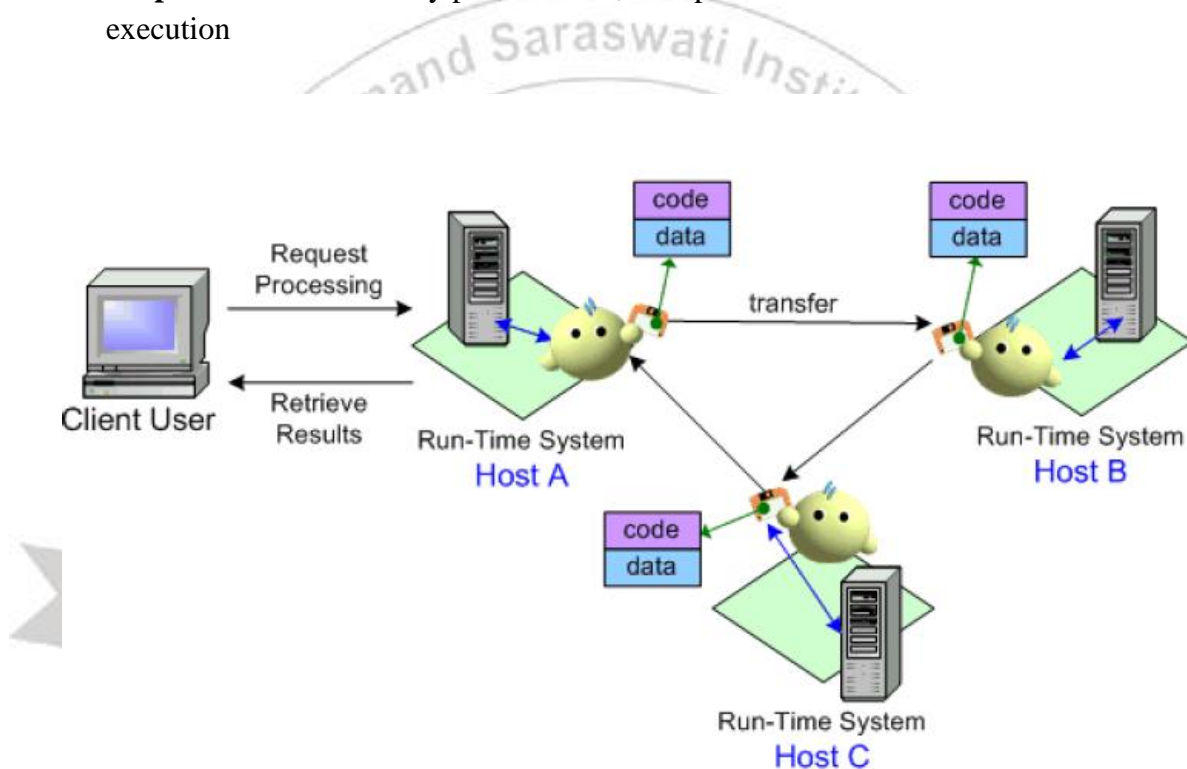
```
C:\WINDOWS\system32\cmd.exe

C:\Program Files\Java\jdk1.6.0_14\bin\DC LAB\LAB3>javac Synch.java
C:\Program Files\Java\jdk1.6.0_14\bin\DC LAB\LAB3>java Synch
[Hello]
[World]
[Synchronized]
C:\Program Files\Java\jdk1.6.0_14\bin\DC LAB\LAB3>
```



EXPERIMENT-4**AIM: Mobile Agent (IBM's Aglet) Programming****LOGIC:****(1) Mobile agent (IBM's Aglet) Programming****What is Mobile Agent?**

- Mobile agents are a **distributed computing paradigm**
- A mobile agent **migrates** from one machine to another **under its own control**
- Decides which locations to visit and what commands to execute
- Continuous interaction with the original source is not required
- **Suspend execution** at any point in time, transport itself to a new machine and resume execution

**Mobile Agent Toolkits**

- Provide the infrastructure for mobile agents ...
 - to interact with a local computer system; this is known as a “**context**” for mobile agents to reside and execute
 - to **move** from host to host
 - to **communicate** with other agents and hosts through **message passing**
 - to maintain privacy and integrity (of agents as well as hosts)
- These toolkits are normally Java-based e.g.
 - Aglets
 - Concordia
 - JADE

- OAA
- TACOMA (C++)

✓ **What is Aglets ?**

- Java based mobile agent toolkit developed by IBM
 - The name originates from Aglet = Agent + Applet
- Aglet properties:
 - **Mobile Java object** that can visit Aglet enabled hosts
 - ♦ Executes in a **Context**; there may be many contexts on a node
 - ♦ An **Aglet server** hosts N contexts
 - Aglet = Aglet state (variable values) + Aglet code (Java class)
 - Autonomous
 - ♦ Runs in a **separate thread** when it arrives at a host
 - Reactive
 - ♦ Responds to **messages** that are sent to it. A message is an object exchanged by aglets
 - ♦ Message passing is **synchronous and asynchronous**
 - Is unique
 - ♦ A **globally unique identifier** is bound to each aglet
 - Mobility level = **weak mobility**

Steps to Install Mobile Agent

Step1 :

First download and install Required Software from :

- Java 1.1.8_010/_16 JDK - <http://java.sun.com/products/archive/index.html>
- Aglet SDK- <http://www.trl.ibm.com/aglets/idoagree103.htm>

(<http://www.trl.ibm.co.jp/aglets>)

Step2 :

Install both.

For installing Aglet SDK:

- download - [Aglets1_0_3.class](#)
- Aglet SDK Page and install as following :



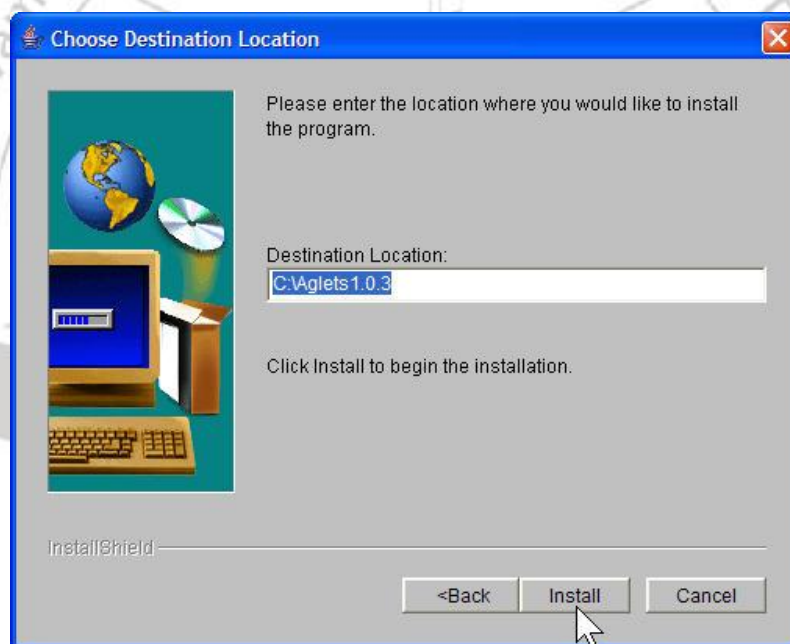
- This will open setup like this



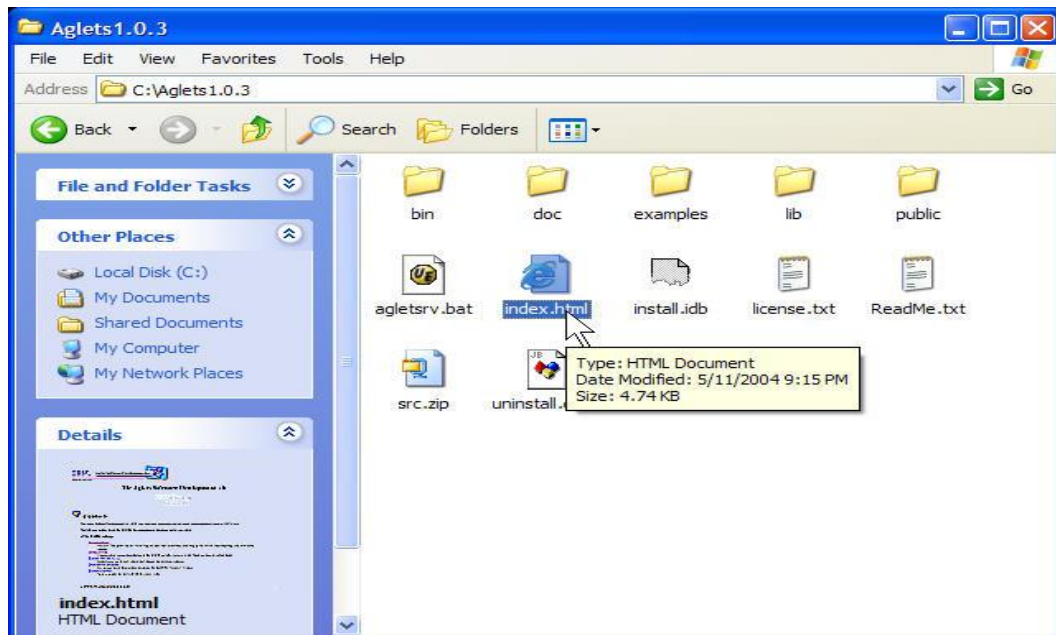
- Click next to continue



- Then click I accept and click next



- Then click install
- After finishing this setup you will get installed aglets in C:\Aglets 1.0.3 folder

**Step 3:-**

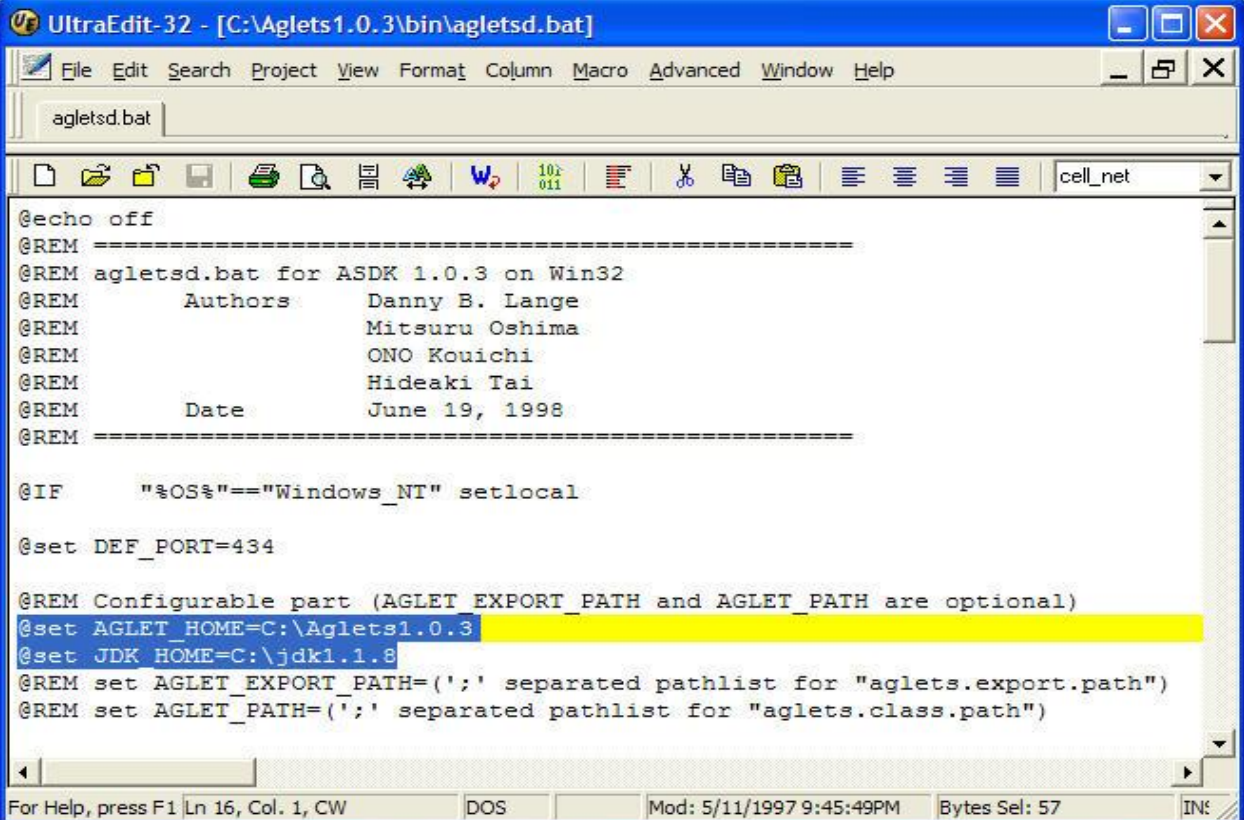
Now set following path before starting it as following way

A screenshot of a Windows command prompt window titled "C:\WINDOWS\System32\cmd.exe". The window shows the following commands and their outputs:

```
C:\>set AGLET_HOME=C:\Aglets1.0.3
C:\>set PATH=%AGLET_HOME%\bin;%PATH%
C:\>set HOME=c:\
C:\>set CLASSPATH=%AGLET_HOME%\lib\aglets.jar;%CLASSPATH%
C:\>set AGLET_PATH=C:\ASDK\public
C:\>set AGLET_EXPORT_PATH=%AGLET_PATH%
C:\>
```

Step 4:-

Now edit agletsd.bat file in following place



```

UltraEdit-32 - [C:\Aglets1.0.3\bin\agletsd.bat]
File Edit Search Project View Format Column Macro Advanced Window Help
agletsd.bat
@echo off
@REM =====
@REM agletsd.bat for ASDK 1.0.3 on Win32
@REM      Authors      Danny B. Lange
@REM                      Mitsuru Oshima
@REM                      ONO Kouichi
@REM                      Hideaki Tai
@REM      Date        June 19, 1998
@REM =====
@if "%OS%"=="Windows_NT" setlocal

@set DEF_PORT=434

@REM Configurable part (AGLET_EXPORT_PATH and AGLET_PATH are optional)
@set AGLET_HOME=C:\Aglets1.0.3
@set JDK_HOME=C:\jdk1.1.8
@REM set AGLET_EXPORT_PATH=(';' separated pathlist for "aglets.export.path")
@REM set AGLET_PATH=(';' separated pathlist for "aglets.class.path")

For Help, press F1 Ln 16, Col. 1, CW DOS Mod: 5/11/1997 9:45:49PM Bytes Sel: 57 IN:

```

Step 5 :-

- Make sure that the

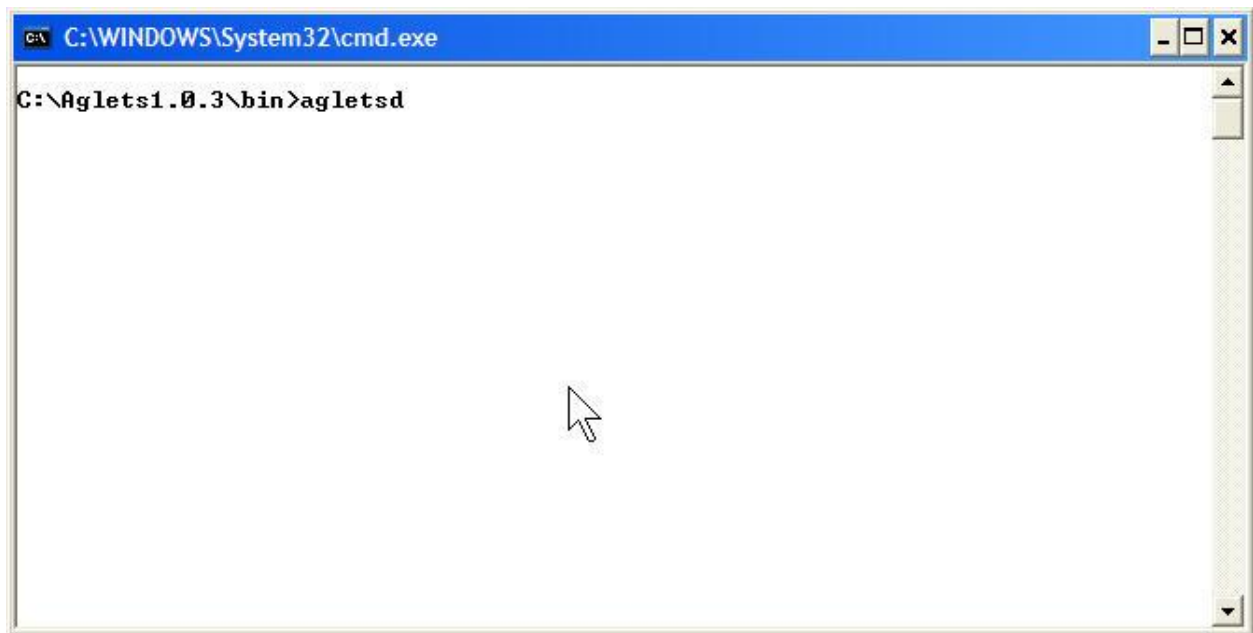
@set AGLET_HOME=C:\Aglets1.0.3

@set JDK_HOME= your installed jdk home derectory

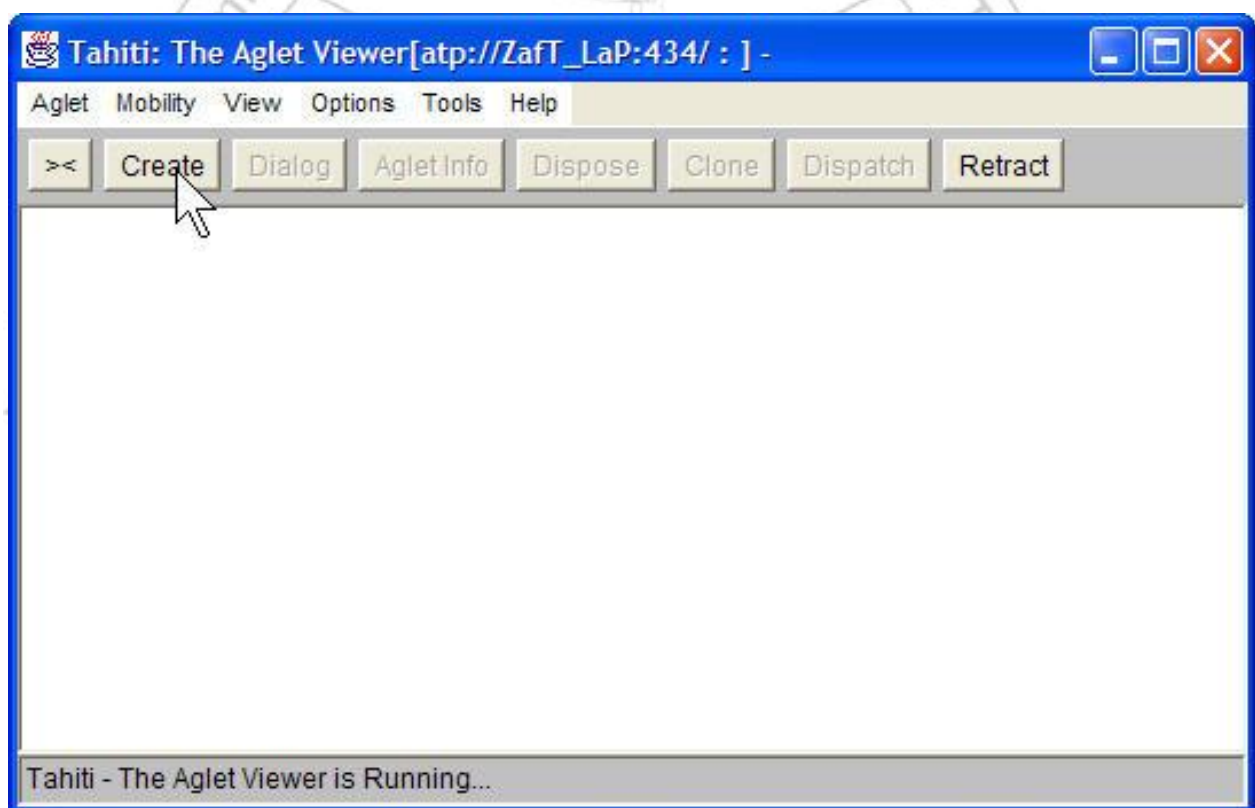
Is perfectly set if not set perfectly then make it perfect else it won't work now
save agletsd.bat file

Step 6:-

- Now to run **Tahiti**



- This will create :



- Now for running your aglet program click create :



- Select example.hello.HelloAglet and click create this will create window like below



- Now type what ever you want and click GO button this will create window like below

**Step to install & to run The Tahiti Server:**

- Ready with Jdk-1_6 Package and Aglets package (Aglets-2.0.2.Zip).
- install JDK and JRE in C:\ drive
- unzip the Aglets-2.0.2.Zip in C drive.
- Once U unzip the pack u will find Sub Folder like bin, Cnf, public.
- Set the Environment Variables as stated below.
 - ✓ Variable Name:- JAVA_HOME
 - ✓ Value :- C:\Java\jdk1.6
 - ✓ Variable Name:- ANT_HOME
 - ✓ Value :- C:\aglets-2.0.2
- now go to command prompt,
- Discover the path :- C:\aglets-2.0.2\bin
- Run ant
- To Run ant, at given path simply type ant. bat(e.g. C:\aglets-2.0.2\bin\ant.bat)
- Once ant run successfully, type ant install-home.
- once this is finished successfully , type “Agletsd “

- It starts running the aglets server called Tahiti Server.
- Tahiti server will ask u to enter the login name and password. (In login name type anonymous and in password field type: aglets and Tahiti Gui is appeared on the screen.
- Start working with the aglets lifecycle for example creating, dispatching, etc.
- Get the file of agent (java compiled file) from public sub directory).

Simple HelloWorld Application:

```
import com.ibm.aglet.*;  
import com.ibm.aglet.event.*;
```

```
public class HelloWorld extends Aglet{  
    public void onCreate(Object init) {  
        setText("Hello World.");  
    }  
}
```

Steps:-

- Create simple application (as shown above).
- Save this file and compile and put in aglets-2.0.2\public\example directory
- Start Tahiti server.
- Select given file from the create new option
- Run the file.

Problems:

EXPERIMENT – 5**AIM: Implement CORBA File****LOGIC:**

(1) Implement CORBA file

Counter.idl

```
interface Counter{
    readonly attribute long value;
    void inc();
    void dec();
};
```

CounterImpl.java

```
public class CounterImpl extends CounterPOA {
    private int count;
    public CounterImpl() {
        count = 0;
    }
    public void inc(){
        count++;
    }
    public void dec(){
        count --;
    }
    public int value(){
        return count;
    }
}
```

Server.java

```
import java.io.*;
import java.util.Properties;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import static java.lang.System.*;

public class Server{
    public static void main(String[] args){
        try {
            Properties props = getProperties();
            ORB orb = ORB.init(args, props);
            org.omg.CORBA.Object obj = null;
            POA rootPOA = null;
            try{
```



```

        obj = orb.resolve_initial_references("RootPOA");
        rootPOA = POAHelper.narrow(obj);
    }catch (org.omg.CORBA.ORBPackage.InvalidName e) { }

    CounterImpl c_impl = new CounterImpl();
    Counter c = c_impl._this(orb);

    try{
        FileOutputStream file = new FileOutputStream("Counter.ref");
        PrintWriter writer = new PrintWriter(file);
        String ref = orb.object_to_string(c);
        writer.println(ref);
        writer.flush();
        file.close();
        out.println("Server started." + " Stop: Ctrl-C");
    }catch (IOException ex){
        out.println("File error: " + ex.getMessage());
        exit(2);
    }

    rootPOA.the_POAManager().activate();
    orb.run();
}catch(Exception ex) {
    out.println("Exception: " + ex.getMessage());
    exit(1);
}
}
}

```

Client.java

```

import java.io.*;
import java.util.*;
import org.omg.CORBA.*;
import static java.lang.System.*;

public class Client{
    public static void main(String[] args) {
        try {
            Properties props = getProperties();
            ORB orb = ORB.init(args, props);
            String ref = null;
            org.omg.CORBA.Object obj = null;
            try {
                Scanner reader = new Scanner(new File("Counter.ref"));
                ref = reader.nextLine();
            }catch (IOException ex) {
                out.println("File error: " + ex.getMessage());
                exit(2);
            }
        }
    }
}

```

```

    }
    obj = orb.string_to_object(ref);
    if (obj == null) {
        out.println("Invalid IOR");
        exit(4);
    }
    Counter c = null;
    try {
        c = CounterHelper.narrow(obj);
    } catch (BAD_PARAM ex) {
        out.println("Narrowing failed");
        exit(3);
    }

    int inp = -1;
    do {
        out.print("Counter value: " + c.value() + "\nAction (+/-/e)? ");
        out.flush();
        do {
            try {
                inp = in.read();
            } catch (IOException ioe) {
            }
        } while (inp != '+' && inp != '-' && inp != 'e');

        if (inp == '+')
            c.inc();
        else if (inp == '-')
            c.dec();
        } while (inp != 'e');
    } catch (Exception ex) {
        out.println("Exception: " + ex.getMessage());
        exit(1);
    }
}
}

```

OUTPUT

Compile the Remote Interface

```
idlj -fall Counter.idl
```

Compile Clinet & Server file

```
javac Server.java
```

```
javac Client.java
```

When the parameter setting –fall is chosen, the JDK’s IDL to Java compiler generates the following files:

CounterPOA.java

_CounterStub.java

CounterHolder.java

CounterHelper.java

Counter.java

CounterOperations.java

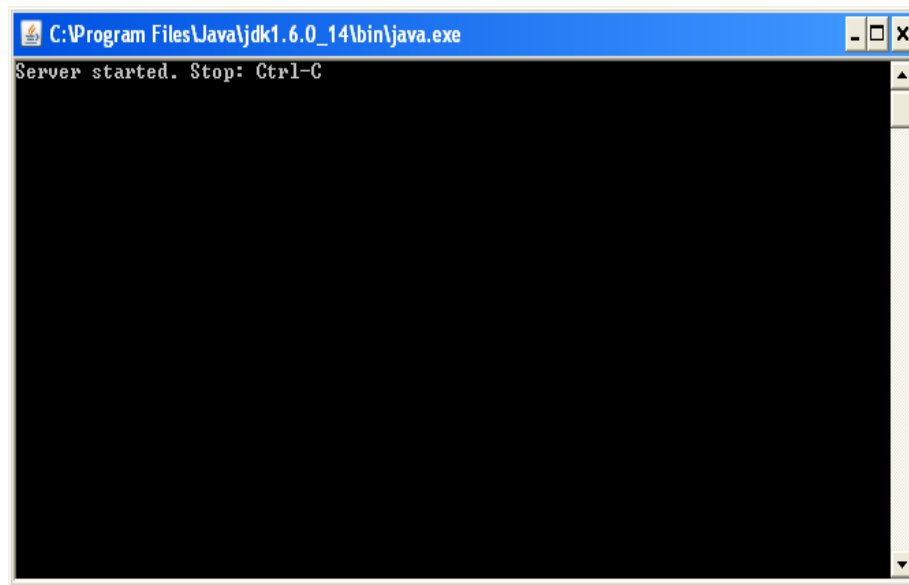
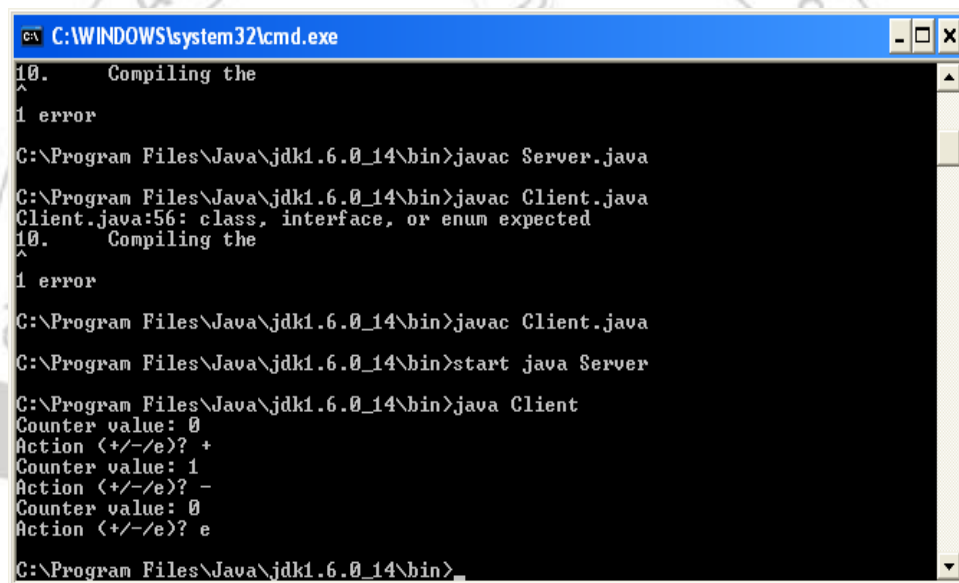
All these files are needed to implement the server application. The client application needs all files except CounterPOA.java.

Run Client and Server file

Start java Server

Java Client



Server.java**Client.java****Problem Statement:**

EXPERIMENT-6: Shared Memory, Semaphore

Aim: Incrementing a counter in shared memory.

Tools/ Apparatus: Unix/Linux C Programming Environment

Procedure:

- 1) Write a server C Program using Shared memory and semaphore (server increments counter between sem_wait and sem_post). Create shared memory using mmap.
- 2) Write a client C Program that reads counter value between sem_wait and sem_post. Access shared memory using open.



EXPERIMENT-7**AIM: Implement Network File System (NFS)****LOGIC:** Implementation of Clustering using MPI_CH2.**Steps:**

- Set up Network File System (NFS)
- Set up Secure Shell (SSH)
- Set up Message Passing Interface (MPI)

Requirement: 2 machine running Linux

- **Set up Network File System**

Consider we have two host machine with ip 10.10.3.4 and 10.10.3.3;
Now I want to make 10.10.3.4 as server and rest as client then to implement NFS file system between this two follow the following steps:

Step 1: Host with ip 10.10.3.4 edit following file as given
in /etc/exports put entry /home 10.10.3.3(rw, no_root_squash) //we want share home directory to client.

in /etc/host.deny put entry portmap:ALL

in /etc/hosts.allow put entry

Portmap: 10.10.3.3
lockd: : 10.10.3.3
rquotad : 10.10.3.3
mound: 10.10.3.3
statd : 10.10.3.3

Step 2: Execute following set of command from root on both machine to start the daemons.

rpc.portmap
rpc.mountd, rpc.nfsd
rpc.statd, rpc.lockd
rpc.rquotad

Step 3: Client machine execute following command to mount server directory on client machine

(If possible make firewall off).

mount 10.10.3.4:/home /mnt/newhome

where 10.10.3.4->server host

/mnt/newhome----directory on client to which server directory /home will be mounted

EXPERIMENT-8

AIM: Creation of A BPEL (Business Process Execution Language) Module and a Composite Application

LOGIC:

Create a SOA project with BPEL Module to compose a web service.

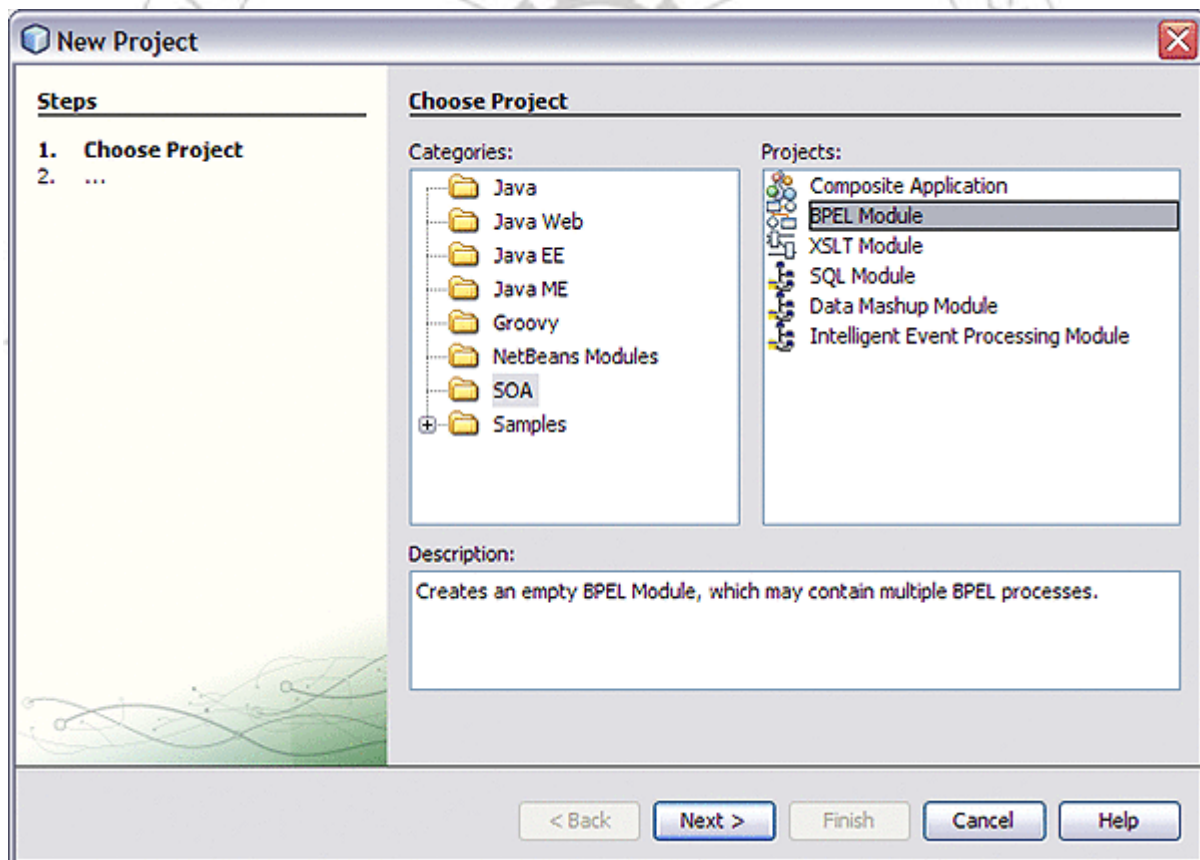
STEPS TO CREATE A BPEL MODULE

Creating a new BPEL Module Project

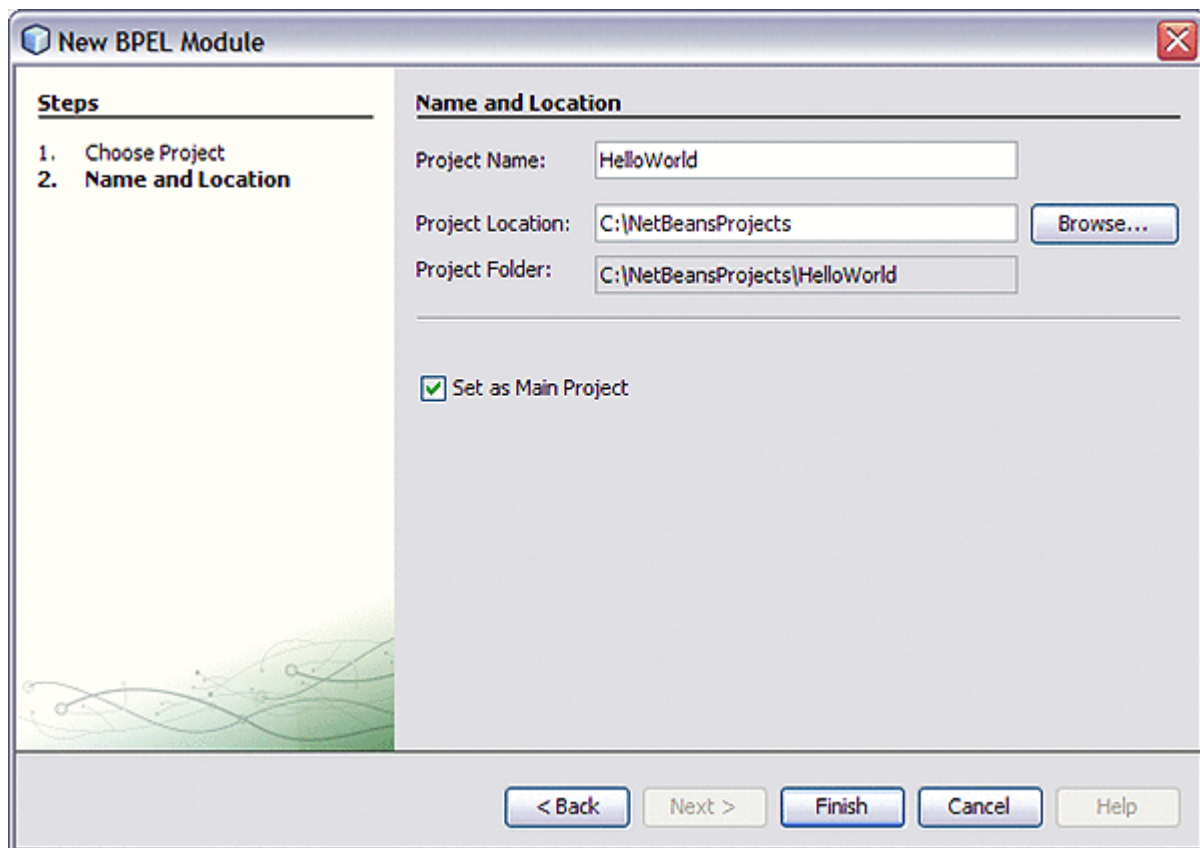
The following sections use the Synchronous sample project, and provide step-by-step directions for creating a simple BPEL module project, using the Synchronous sample project.

To Create a BPEL Module Project

1. In the NetBeans IDE, choose File > New Project.
The New Projects wizard appears
2. Under Categories, select Service Oriented Architecture.
3. Under Projects, select BPEL Module and click Next.

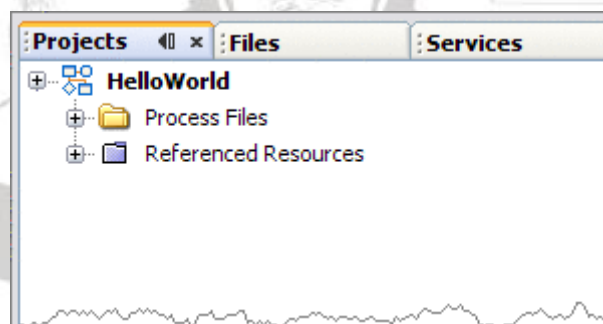


4. In the Name and Location page, enter the project name (for this example HelloWorld) and specify the project location or accept the defaults.



5. Click Finish.

The Projects window now contains a project node for the BPEL Module project.

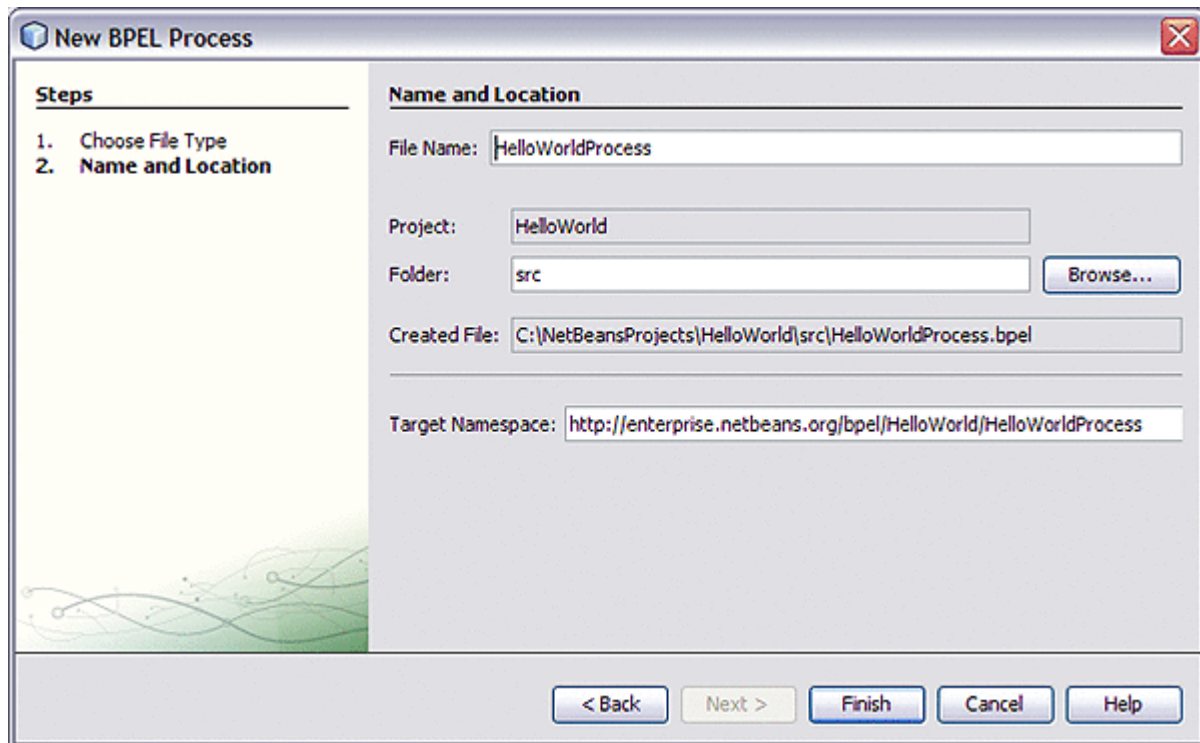


Creating a BPEL Process Using the BPEL Designer

Now that you have created your XML Schema and WSDL Document, you can create your BPEL process.

To Create the BPEL Process

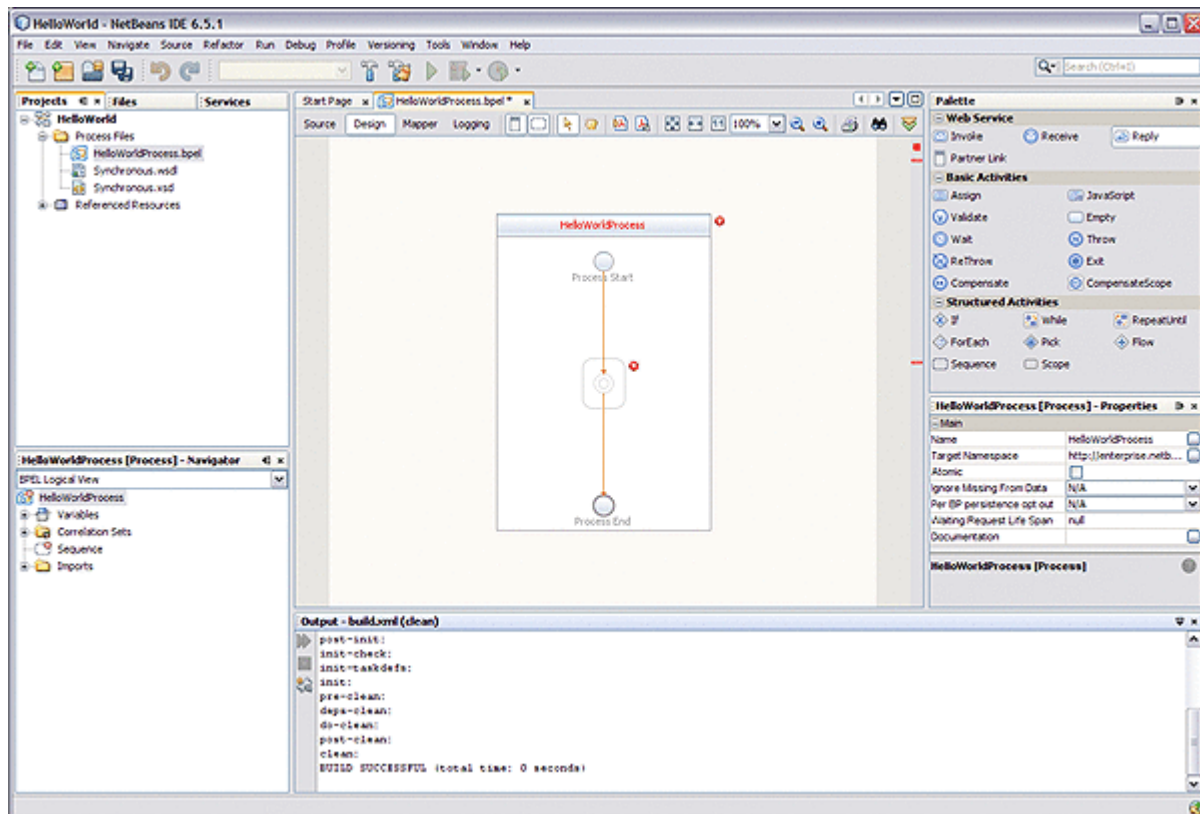
1. In the Projects window, expand your BPEL module project node, right-click the Process Files node, and choose New -> BPEL Process.
The New BPEL Process dialog box appears.



2. Enter a name for the process file name (HelloWorldProcess for this example), and click Finish.

The new BPEL file opens in the Design view of the BPEL Designer.

If the Palette and Properties windows are not displayed in your current view, click Windows -> Reset Windows on the NetBeans menu.

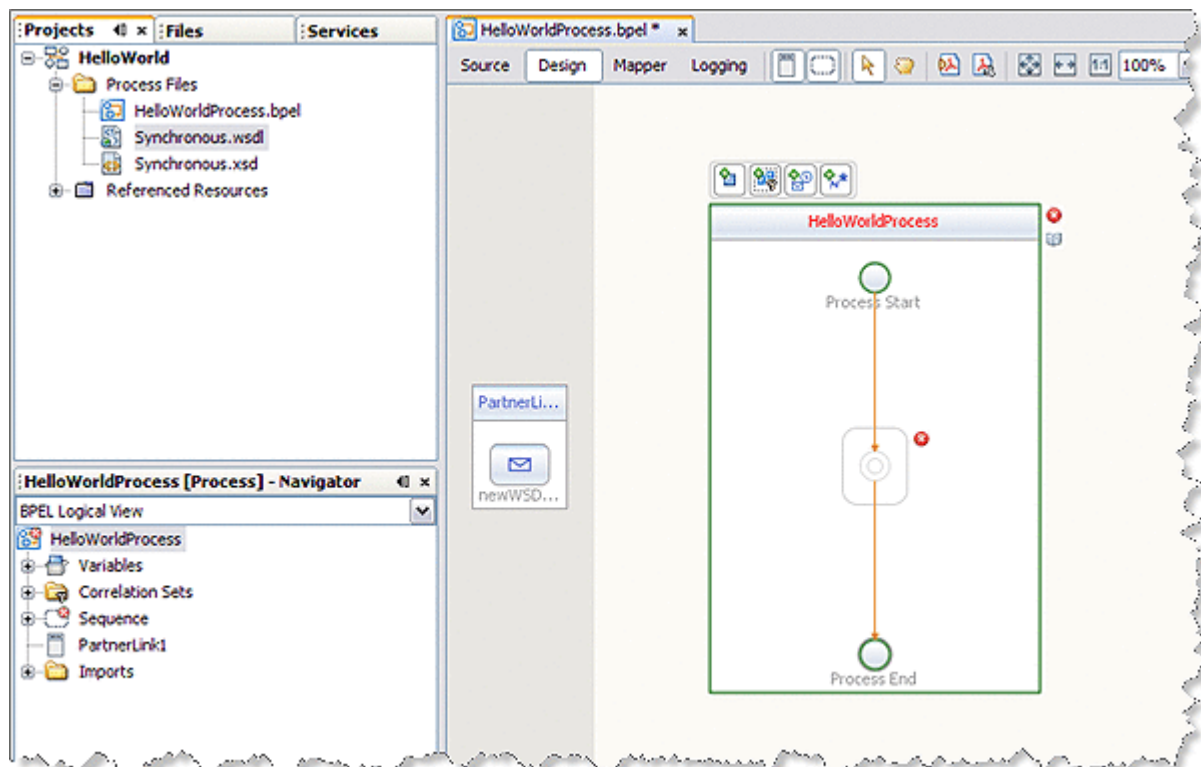


3. Add a partner link to the BPEL Process.

- In the Projects window, expand your project's Process Files node and select the .wsdl file (Synchronous.wsdl for this example).
- Drag the WSDL file from the Projects window to the left side of the Design view canvas.

The IDE provides visual prompts to show you where you can drop the selection.

The BPEL Editor adds a partner link to the canvas.



4. Add a Receive activity to the BPEL Process.

a. From the Web Service section of the Palette window, select the Receive activity.

b. Drag the Receive activity to the HelloWorldProcess process box in the Design view canvas, between the Process Start and the Process End activities.

The Receive1 activity is added to the process box.

c. Click the Receive1 activity's Edit icon.

The Receive1 Property Editor appears.

d. On the Main tab, change the value in the Name field to **start**.

e. From the Partner Link drop-down list, choose **PartnerLink1**.

The IDE populates the Operation field with NewWSDLOperation.

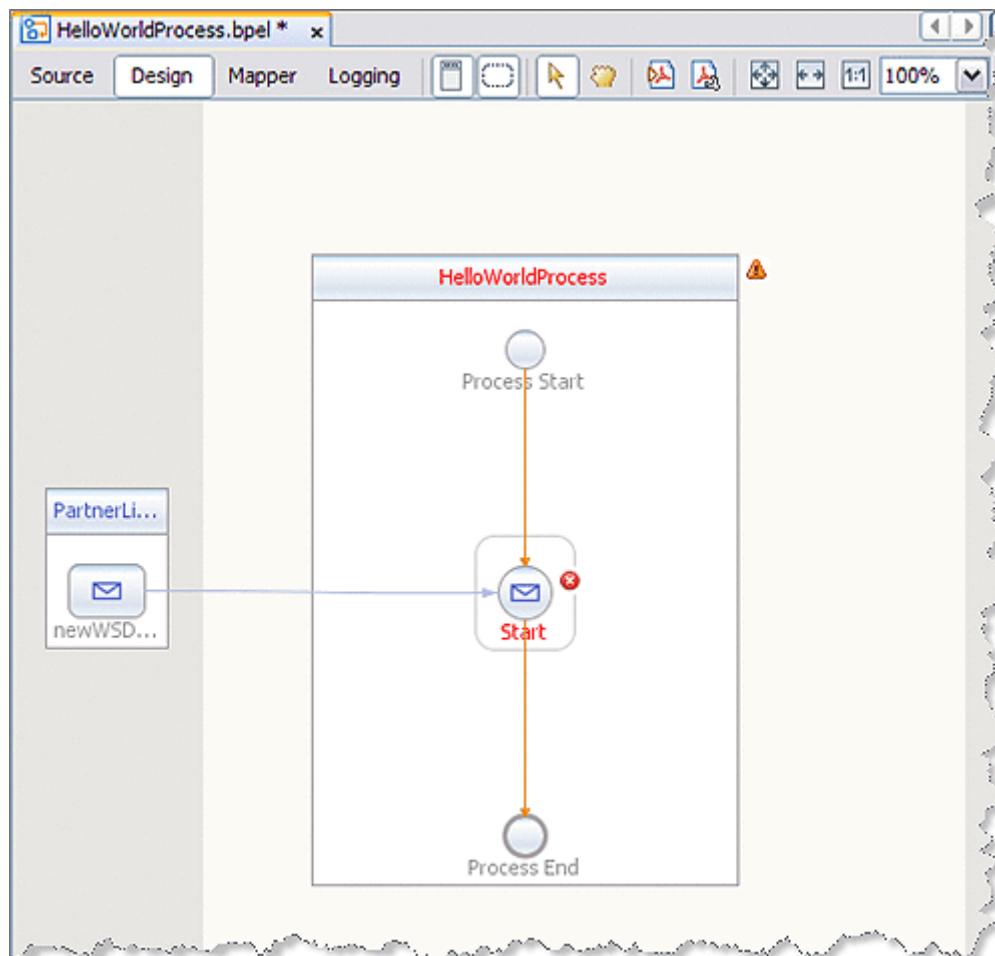
f. Click the Create button next to the Input Variable Field.

The New Input Variable dialog box appears.

g. Click OK to accept the default values.

h. Click OK to close the Receive1 Property Editor.

The Design view displays the new connection between PartnerLink1 and the Start activity in the process box.



5. Add a Reply activity to the BPEL Process.
 - a. Select the Reply activity in the Web Service section of the Palette. Drag and drop the Reply to the prompt between the Start activity and the Process End activity in the process box on the design view canvas.

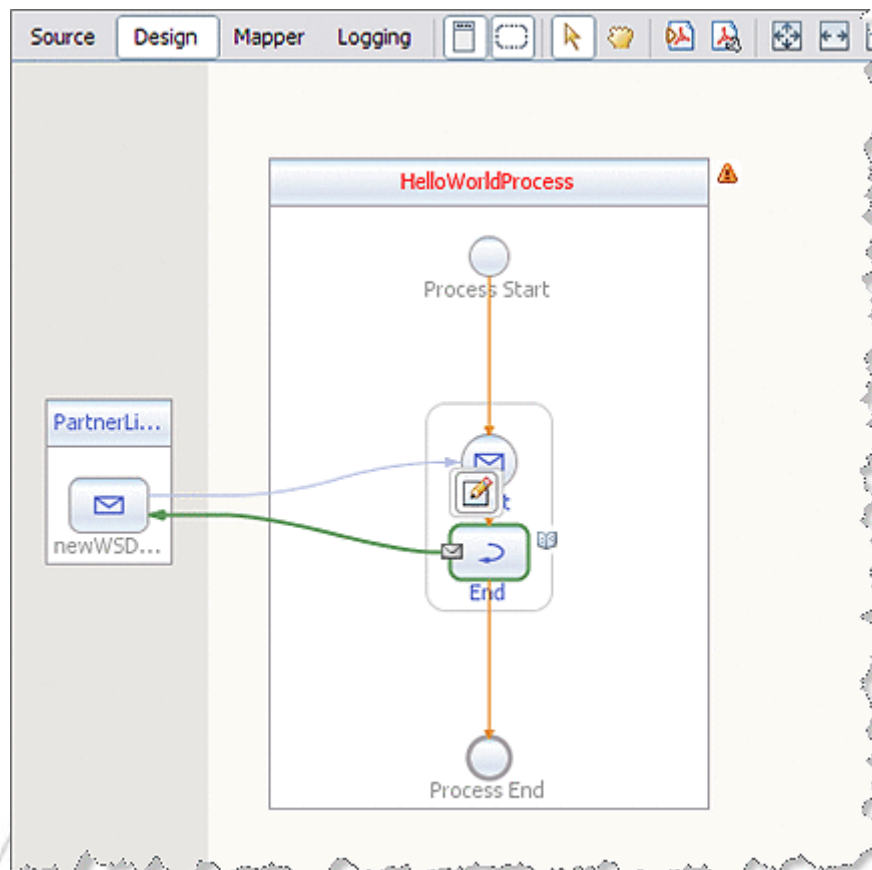
A Reply1 activity is added to the design view canvas.

- b. Click the Reply1 activity's Edit icon.
The Reply1 Property Editor appears.
 - c. On the Main tab, change the value in the Name field to End.
 - d. From the Partner Link drop-down list, choose PartnerLink1.

The IDE populates the Operation field with NewWSDLOperation.

- e. To create a new output variable, make sure that Normal Response is selected, and click the Create button next to the Input Variable Field.
The New Input Variable dialog box appears. Click OK to accept the default values.

- f. Click OK to close the Reply1 Property Editor.
The Design view displays the new connection between the End activity in the process box and PartnerLink1.



6. Add a Assign activity to the BPEL Process.

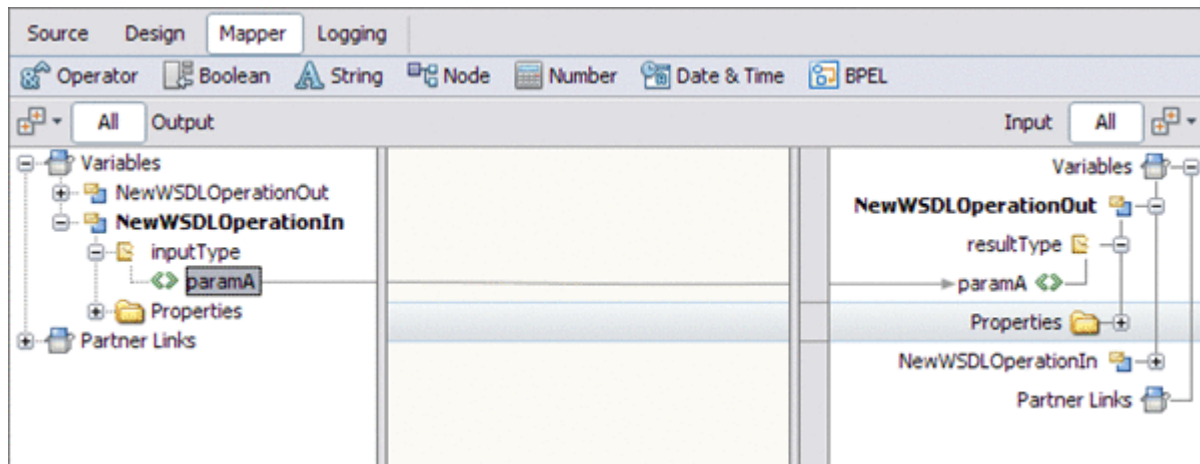
- a. Select the Assign activity in the Basic Activities section of the Palette. Drag and drop the Assign to the prompt between the Start activity and the End activity in the process box on the design view canvas.

The Assign1 activity is added to the design view canvas.

- b. Select the Assign1 activity and click the Mapper button on the editors toolbar. The BPEL Mapper appears.

- c. Map the paramA node under Variables -> NewWSDLOperationIn -> inputType in the Output pane of the BPEL Mapper, to the paramA node under Variables -> NewWSDLOperationOut -> resultType in the Input pane of the Mapper. To do this, select the paramA node under Variables -> NewWSDLOperationIn -> inputType in the Output pane, and drag your cursor to the paramA node under Variables -> NewWSDLOperationOut -> resultType in the Input pane.

This assignment copies the input statement into the output.



7. To save your changes click the Save All icon in the IDE menu bar.

Creating a Composite Application Project

A BPEL Module project is not directly deployable. You must first add a BPEL Module project, as a JBI module, to a Composite Application project. You can then deploy the Composite Application project. Deploying the project makes the service assembly available to the application server and enables its service units to run.

To Create a New Composite Application Project

1. Choose File > New Project (Ctrl-Shift-N).
2. In the Categories list choose Service Oriented Architecture, in the Projects list choose Composite Application, and click Next.
3. In the Name and Location page, change the project name to HelloWorldApplication, and specify the location of project files.
4. To set the new Composite Application the main project as main, leave the Set as Main Project checkbox selected, and click Finish.
5. To add the BPEL Module as a JBI module to the Composite Application project, right-click the new Composite Application and choose Add JBI Module.

The Select Project dialog box opens.

6. Select the HelloWorld project you created earlier and click Add Project JAR Files. The Select Project dialog box closes and the HelloWorld.jar file is added to the JBI Modules node of the HelloWorldApplication Composite Application

Building and Deploying the Composite Application Project

Building a project compiles the BPEL source file and packages the BPEL file and web service artifacts, including WSDL and XSD files, into a JAR archive. Deploying the project compiles the files in the Composite Application project, packages the compiled BPEL and related web service artifacts (including WSDL and XSD files) into an archive, and deploys them to the Application Server.

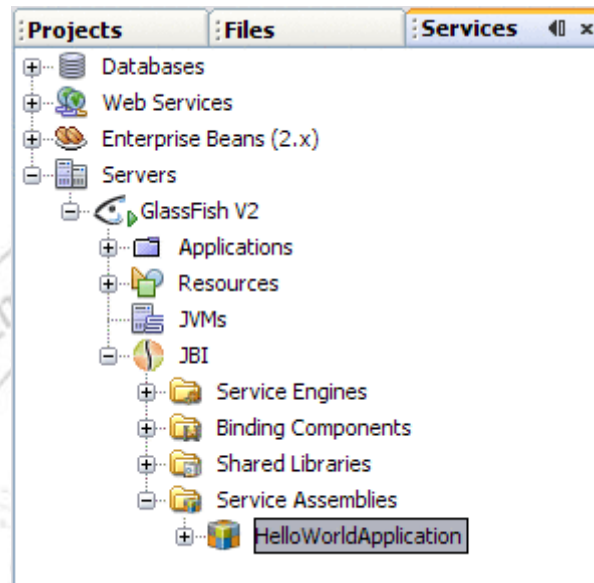
To Build and Deploy the Composite Application Project

1. Right-click the Composite Application project's node, and choose Build.

When the build is complete the Output window reports Build Successful. If the Output window is not visible, choose Window -> Output -> Output.

2. Right-click the Composite Application project's node, and choose Deploy.
3. Deployment has succeeded when you see a Build successful message in the GlassFish tab of the Output window.
4. Open the Services window and expand Servers -> GlassFish V2 -> JBI -> Service Assemblies to see your new deployed Service Assembly.

If you do not see the deployed project, right-click the Service Assemblies node and choose Refresh.



Testing the Composite Application

You can test your Composite Application project by adding test cases, binding to the operation, supplying input, and then using the tester.

Test the HelloWorldApplication Composite Application Project

1. In the Projects window, expand the HelloWorldApplication project node, right-click the Test node, and choose New Test Case.

The New Test Case wizard opens.

2. Accept the default test case name, TestCase1, and click Next.
3. From the Select the WSDL Document page, expand the HelloWorld - Process Files node, select Synchronous.wsdl, and click Next.
4. From the Select the Operation to Test page, select the Operation1 and click Finish.

A new TestCase1 node is added under the project's Test node in the Projects window, containing two subnodes, Input and Output.

The Source Editor appears containing the Input file, Input.xml

Note –

If the Source Editor does not contain a tab for Input.xml, double-click the Input node in the Projects window to open the file.

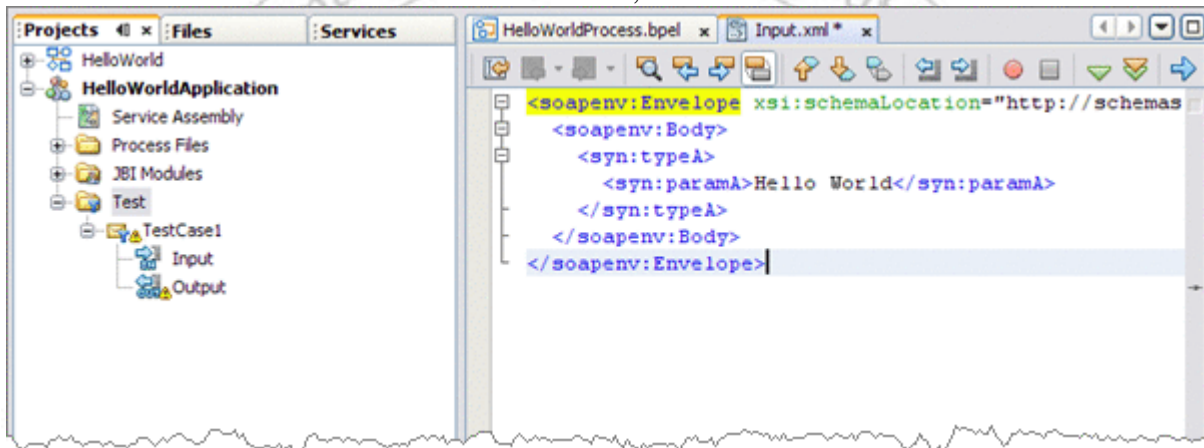
5. From the Input.xml tab of the Source Editor, do the following:
 - a. Locate the line:

```
<syn:paramA>?string?</syn:paramA>
```

- b. Replace ?string? with Hello World, so that the line appears as follows:

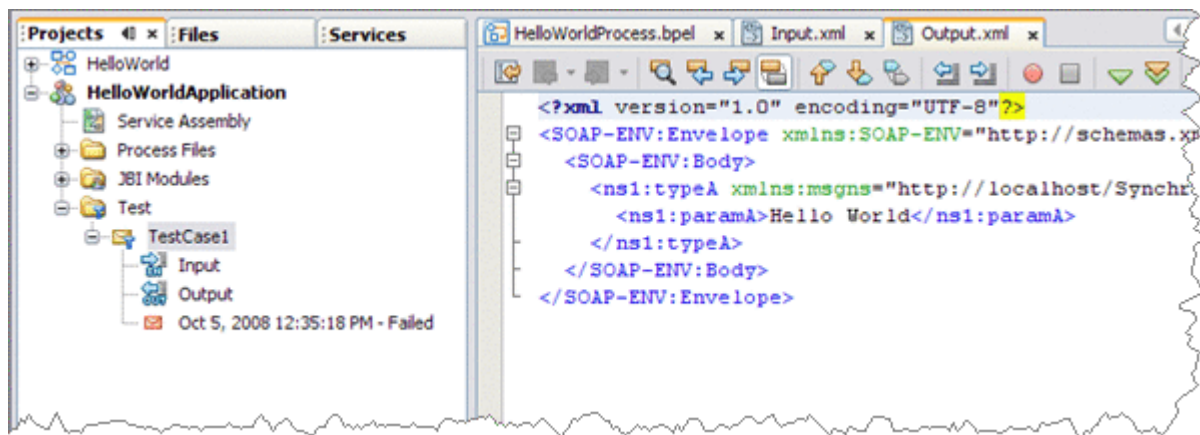
```
<syn:paramA>Hello World</syn:paramA>
```

- c. From the NetBeans IDE menu bar, click Save All.



6. In the Projects window, double-click the Output node under Test -> TestCase1. Output.xml is opened in the Source Editor. Initially, Output.xml is empty until the first test run populates the file.

7. In the Projects window, right-click the TestCase1 node and choose Run from the pop-up menu.
When the Overwrite Empty Output dialog box appears, click Yes to accept new output. The first test run populates the Output.xml file displayed in the Source Editor.



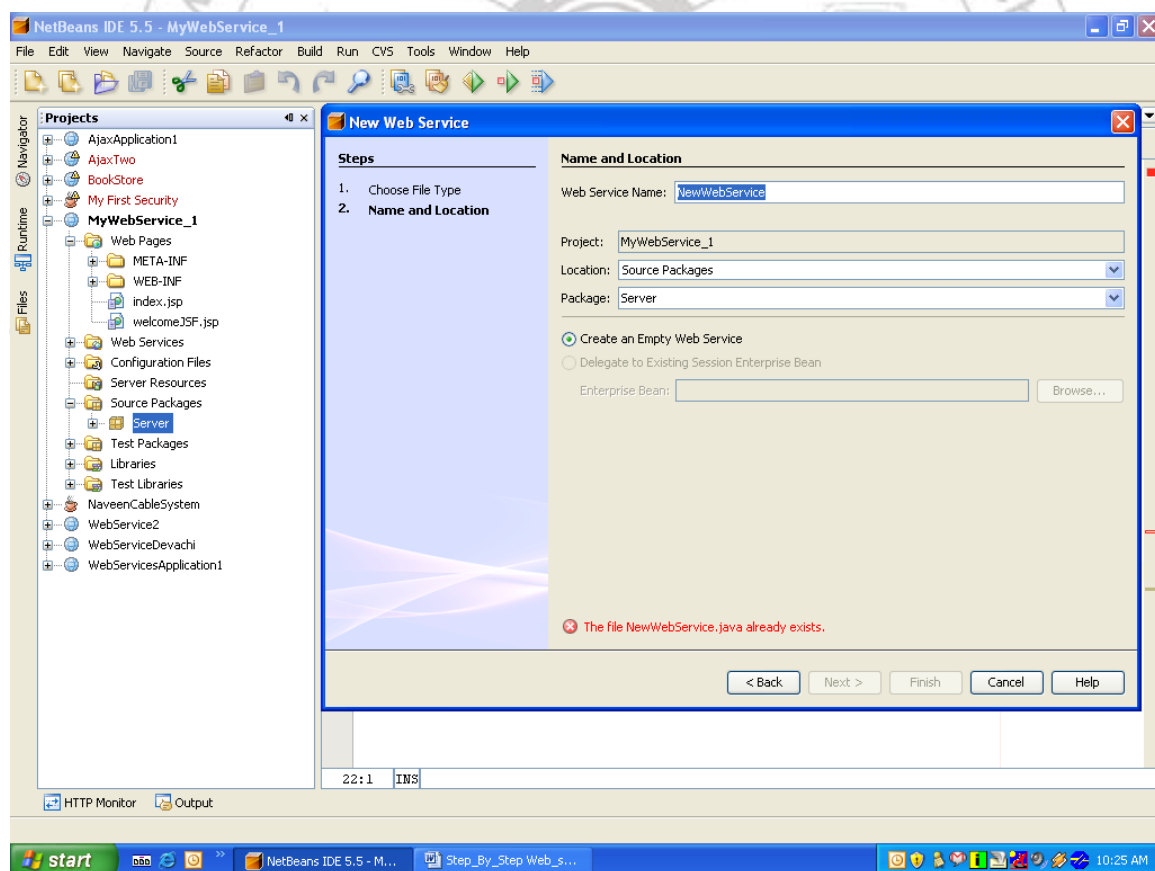
The test compares the output to the contents of the output file. Because the first run of the test has nothing to compare itself to, the first test fails. Subsequent test runs will compare their output with the contents of Output.xml and should succeed.

8. Run the test again.

The test case is compared to the current output file and succeeds.

Problem Statement:



EXPERIMENT-9**AIM: Web Service Programming****LOGIC:****STEP-1)****Create WebApplication|→MyWebService****SourcePackage |→ Create Web Service(Righth Click).****|→ Create Package Server (Righth Click).****|→First_Web_Service(Name Of the File).****Check It → Select Location .****→ Package.**

```
/*  
  
 * First_Web_Service.java *  
  
 * Created on July 2, 2007, 10:24 AM  
  
 *  
  
 * To change this template, choose Tools | Template Manager  
  
 * and open the template in the editor.  
  
*/
```

```
package Server;  
  
import javax.jws.WebService;  
  
@WebService()  
public class First_Web_Service {  
  
}
```

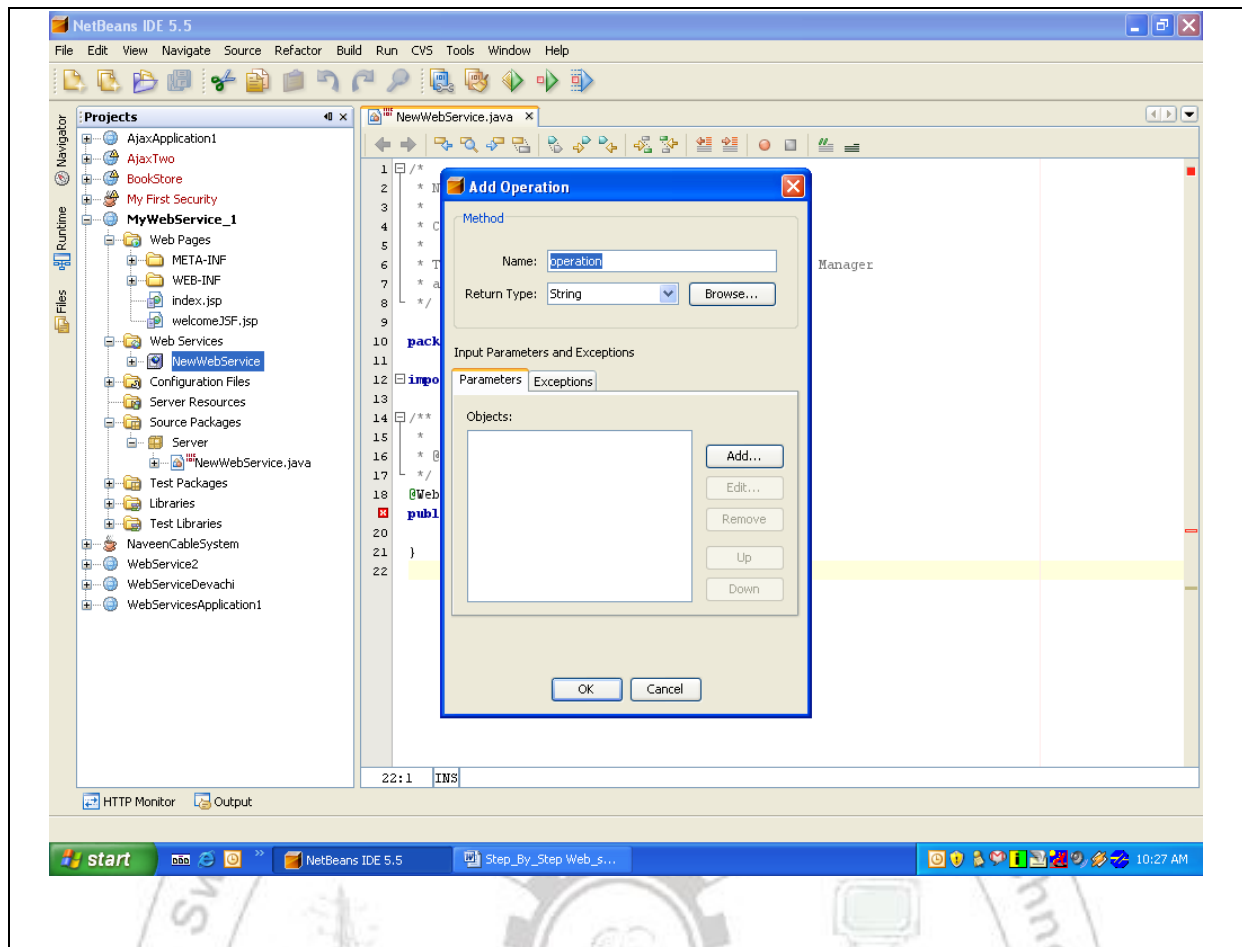
STEP-2)

First_Web_Service → Right Click.

|--→ go to WEB-Service Option.

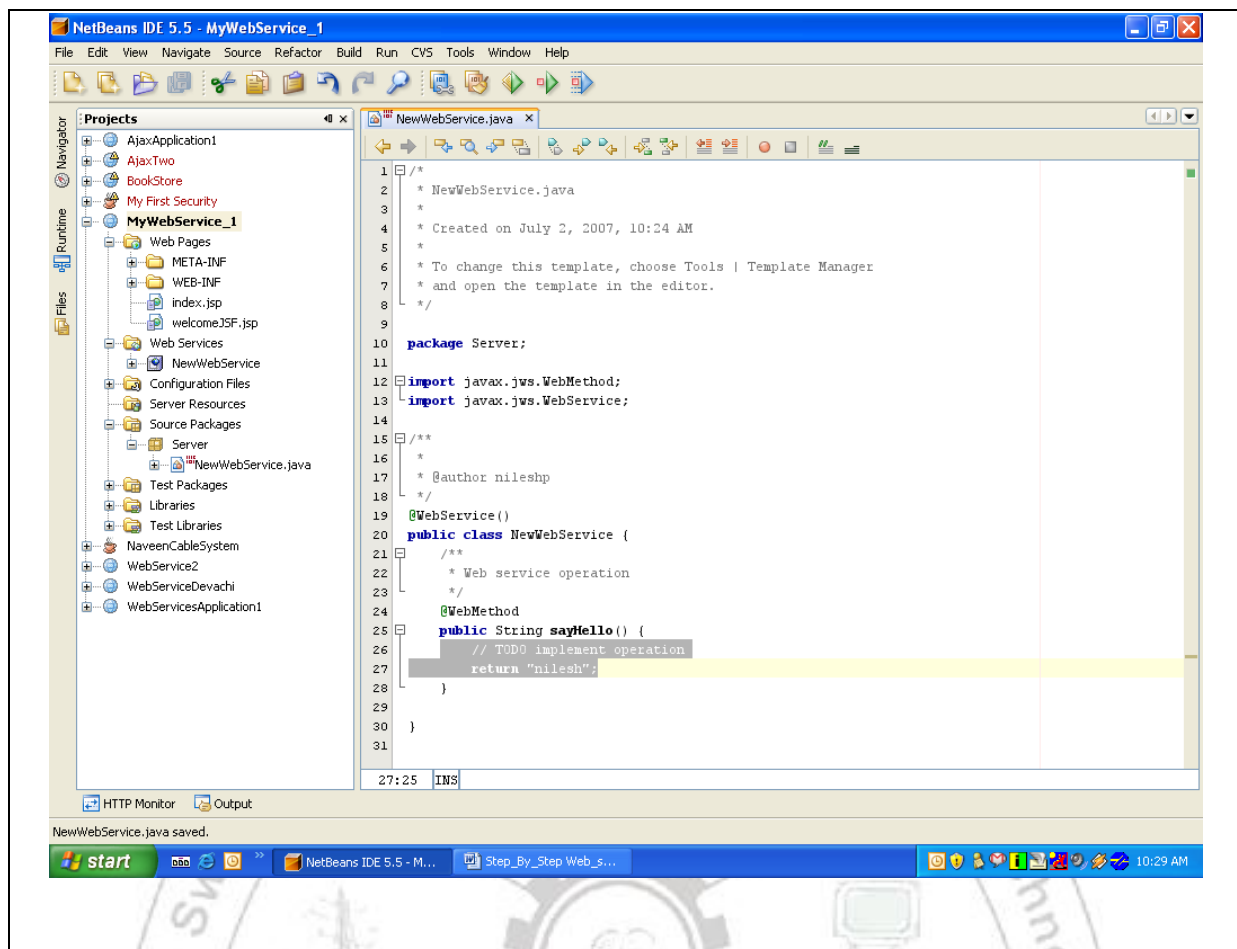
|--→ Add Operation.



**STEP-4)**

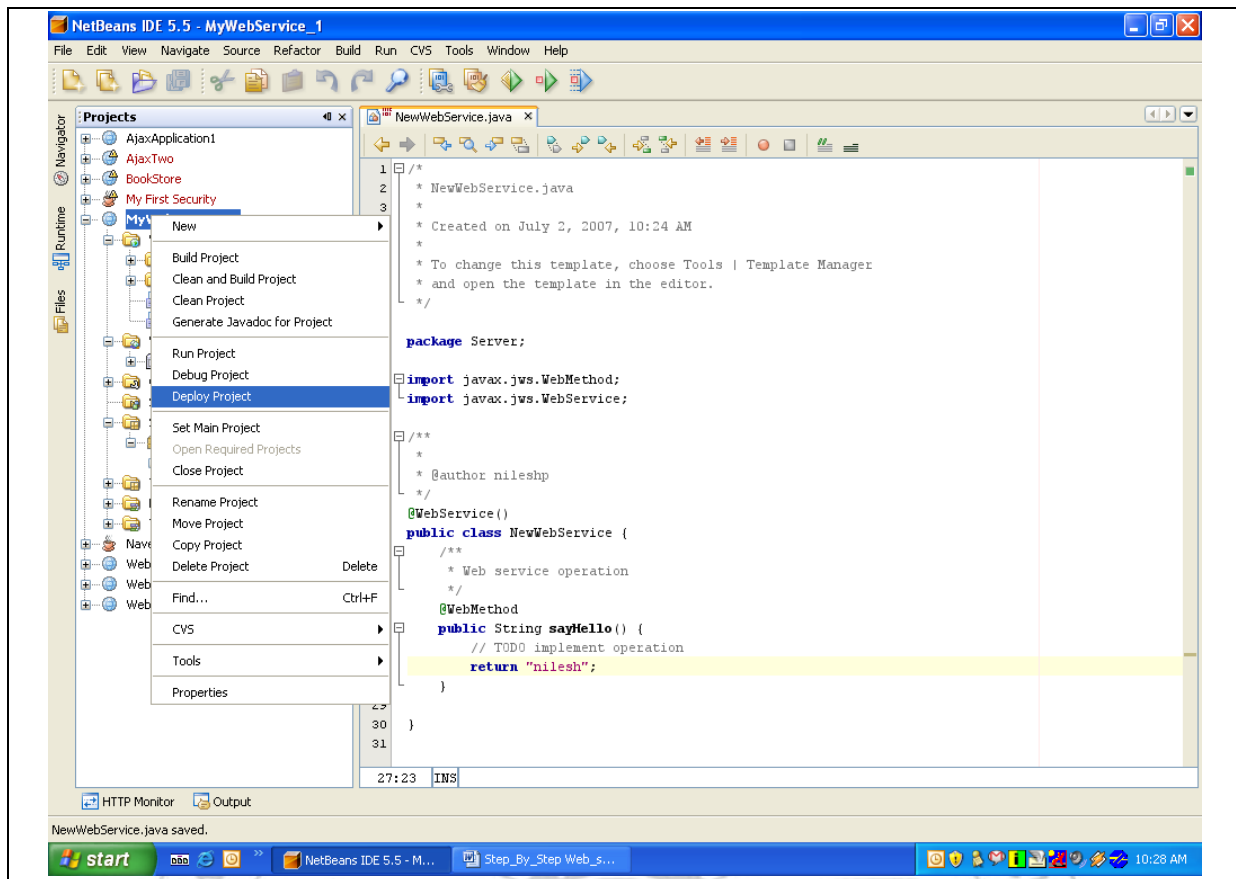
In First_Web_Service Class.

Return null; change it to “SSASIT”; whatever which returns string.



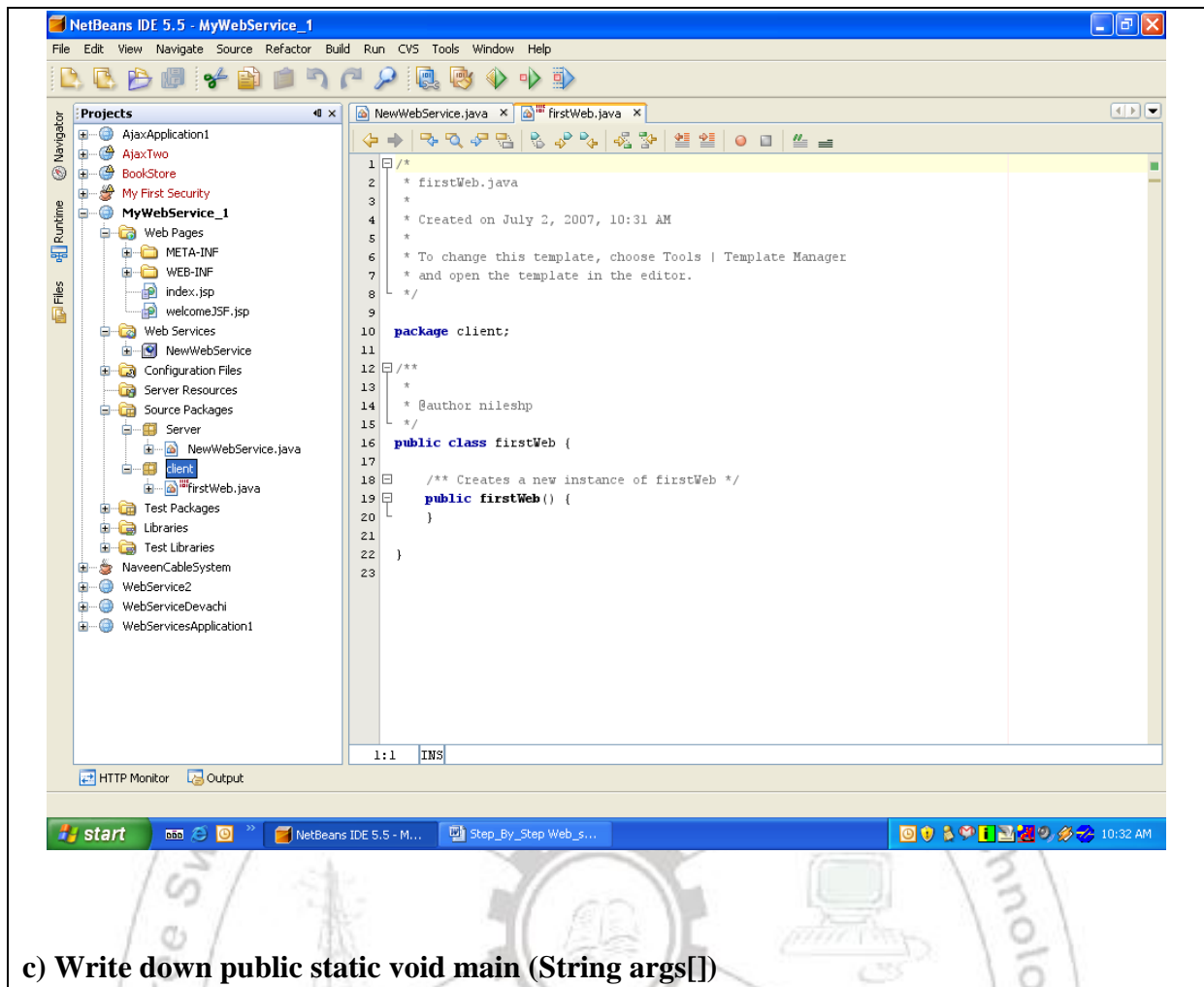
STEP-5)

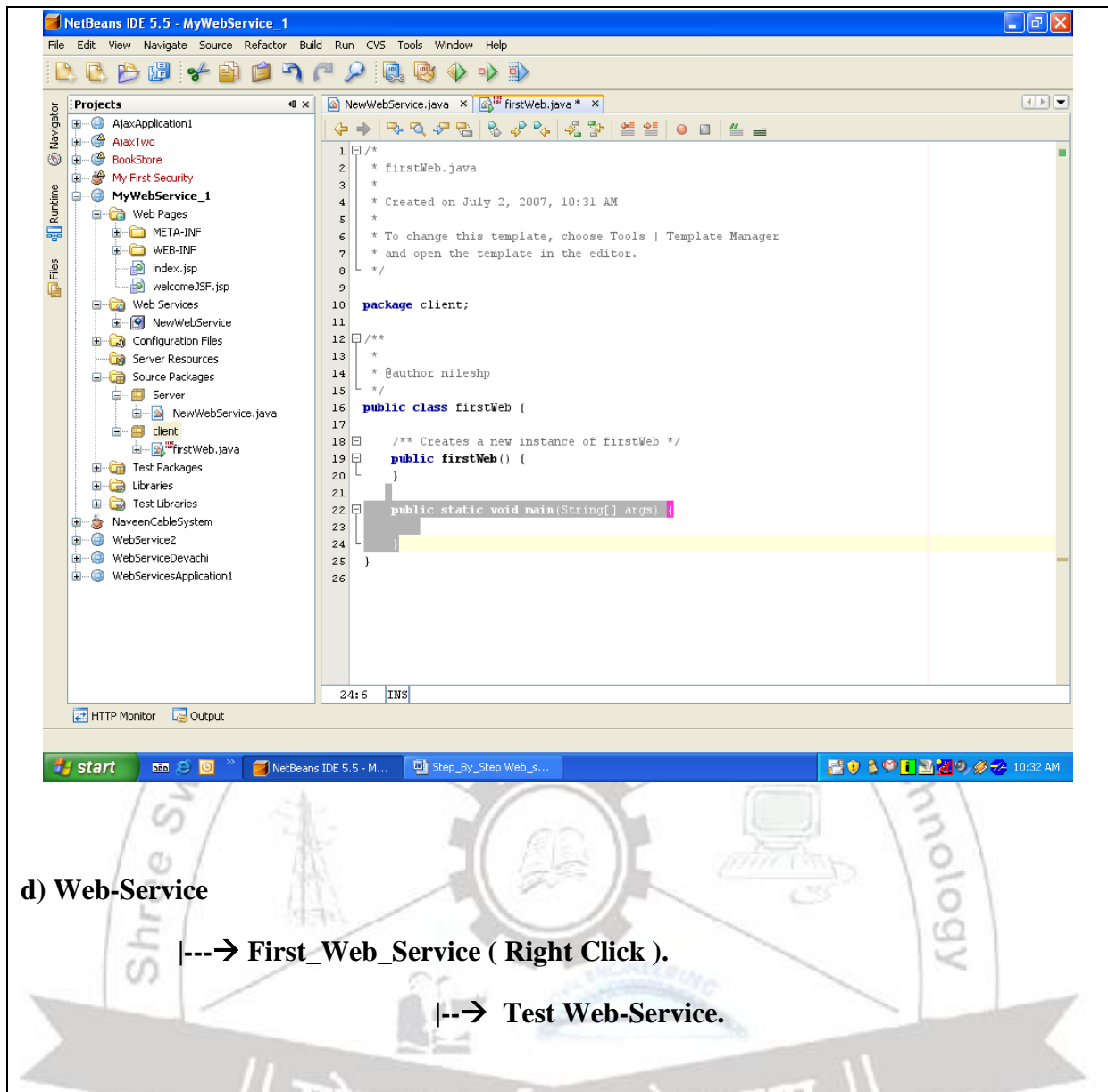
“Deploy Project.”

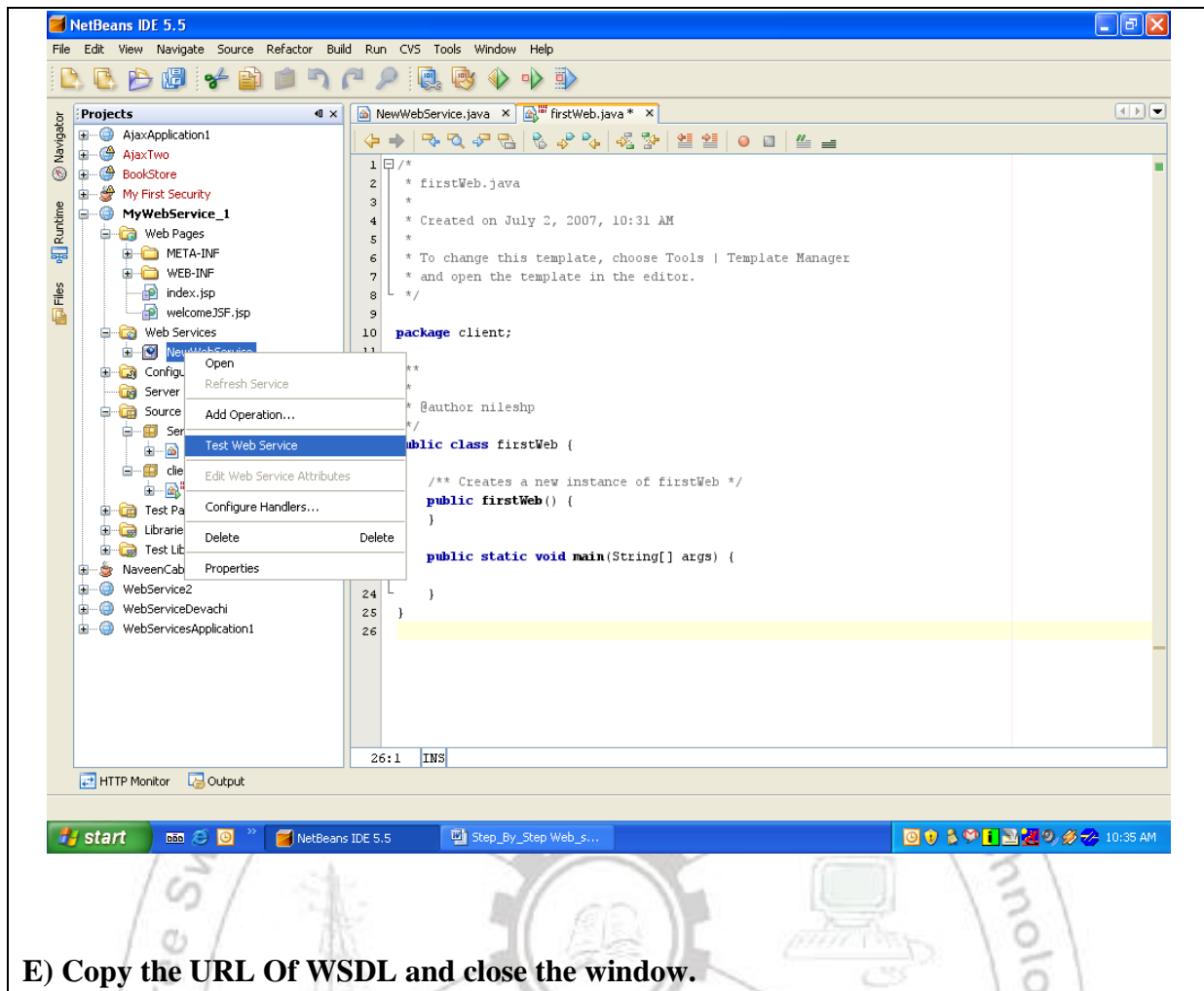



STEP-5: After Deploying Project.

- Create Client (Package Name) as u like.
- Create Java Class in Client Package. (FirstWeb → Class name .)







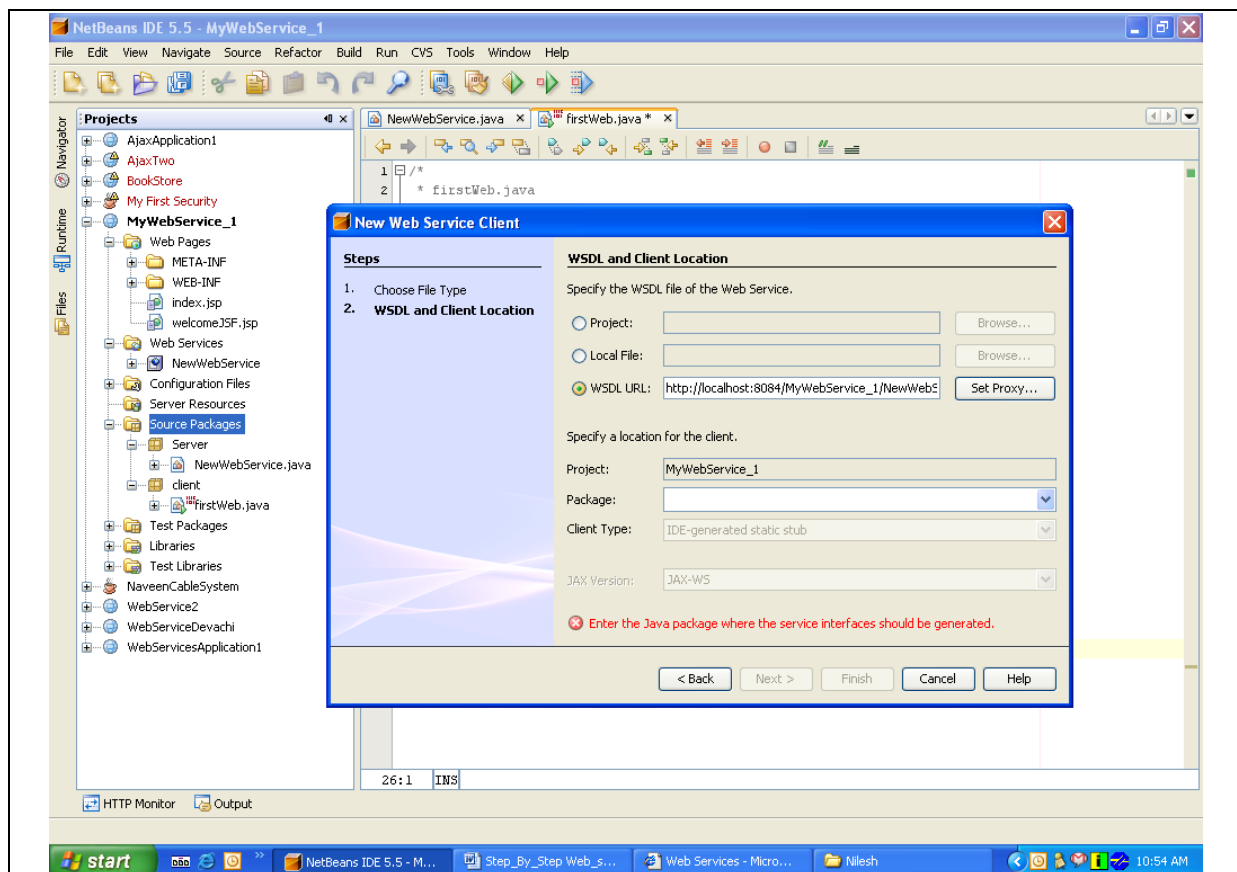


The screenshot shows a Microsoft Internet Explorer window titled "Web Services - Microsoft Internet Explorer". The address bar displays `http://localhost:8084/MyWebService_1/NewWebService`. The main content area is titled "Web Services" and contains a table with the following data:

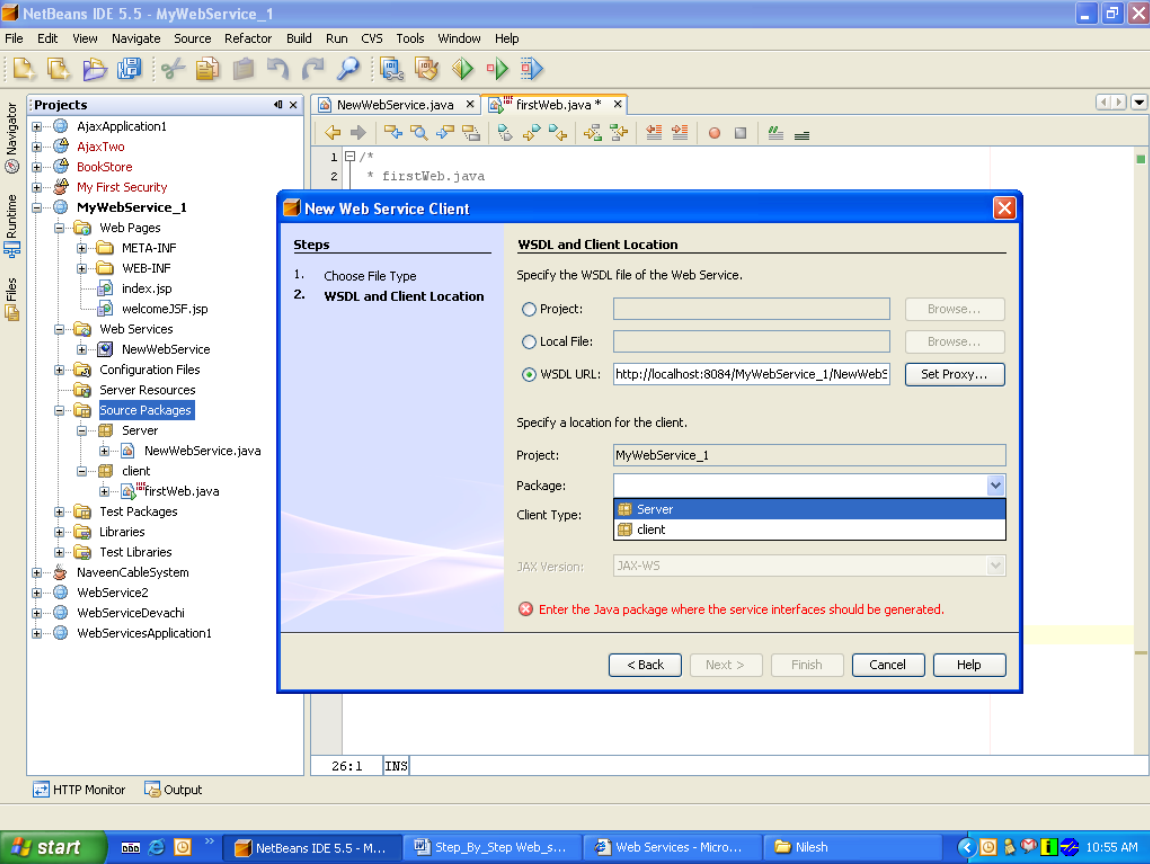
Port Name	Status	Information
NewWebService	ACTIVE	<p>Address: <code>http://localhost:8084/MyWebService_1/NewWebService</code></p> <p>WSDL: <code>http://localhost:8084/MyWebService_1/NewWebService?wsdl</code></p> <p>Port QName: <code>{http://Server/} NewWebServicePort</code></p> <p>Implementation class: <code>Server.NewWebService</code></p>

Below the table, the text "STEP-5)" is visible. Below that, the text "a) Source Package ... Right Click" is visible. Below that, the text "new -> Web-Service Client ." is visible.



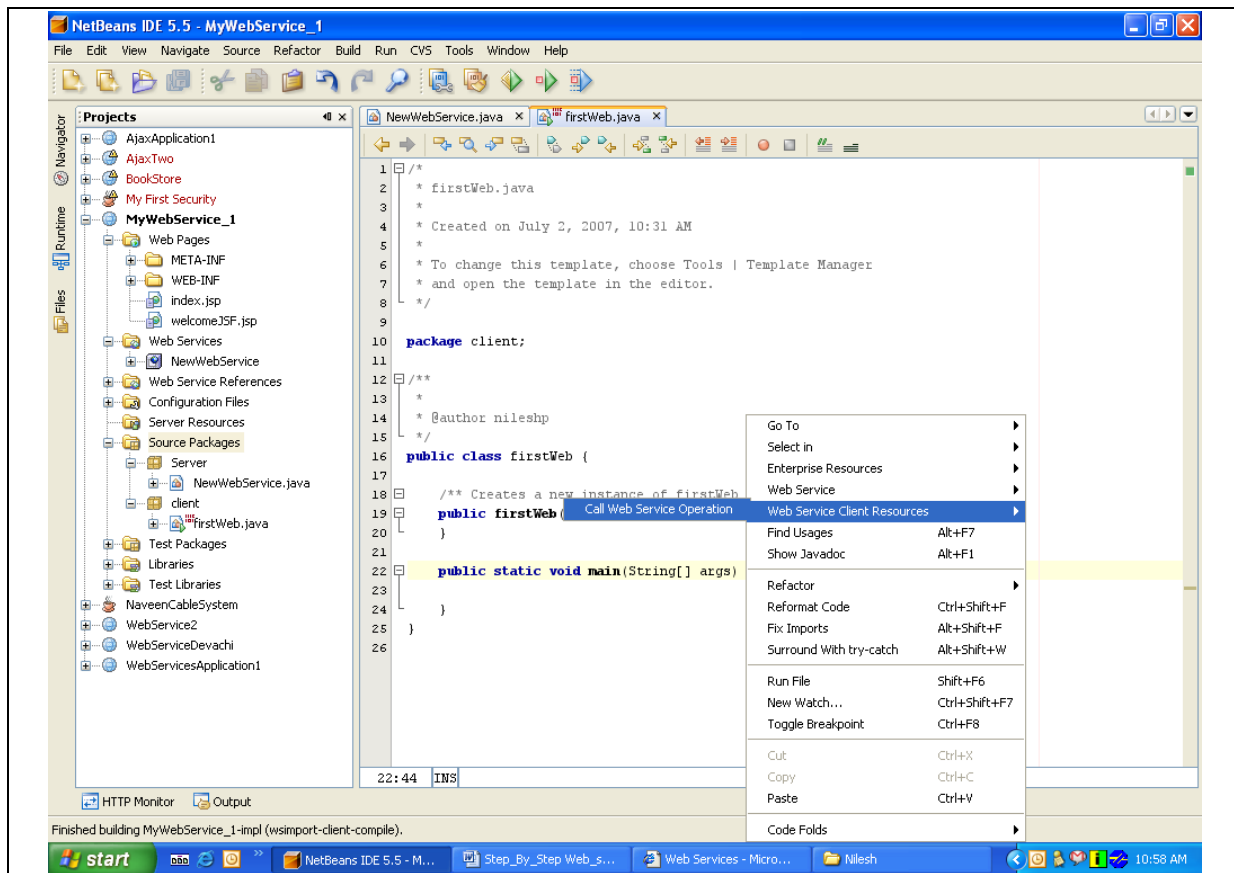


c) Select Server Package. And click on Finish



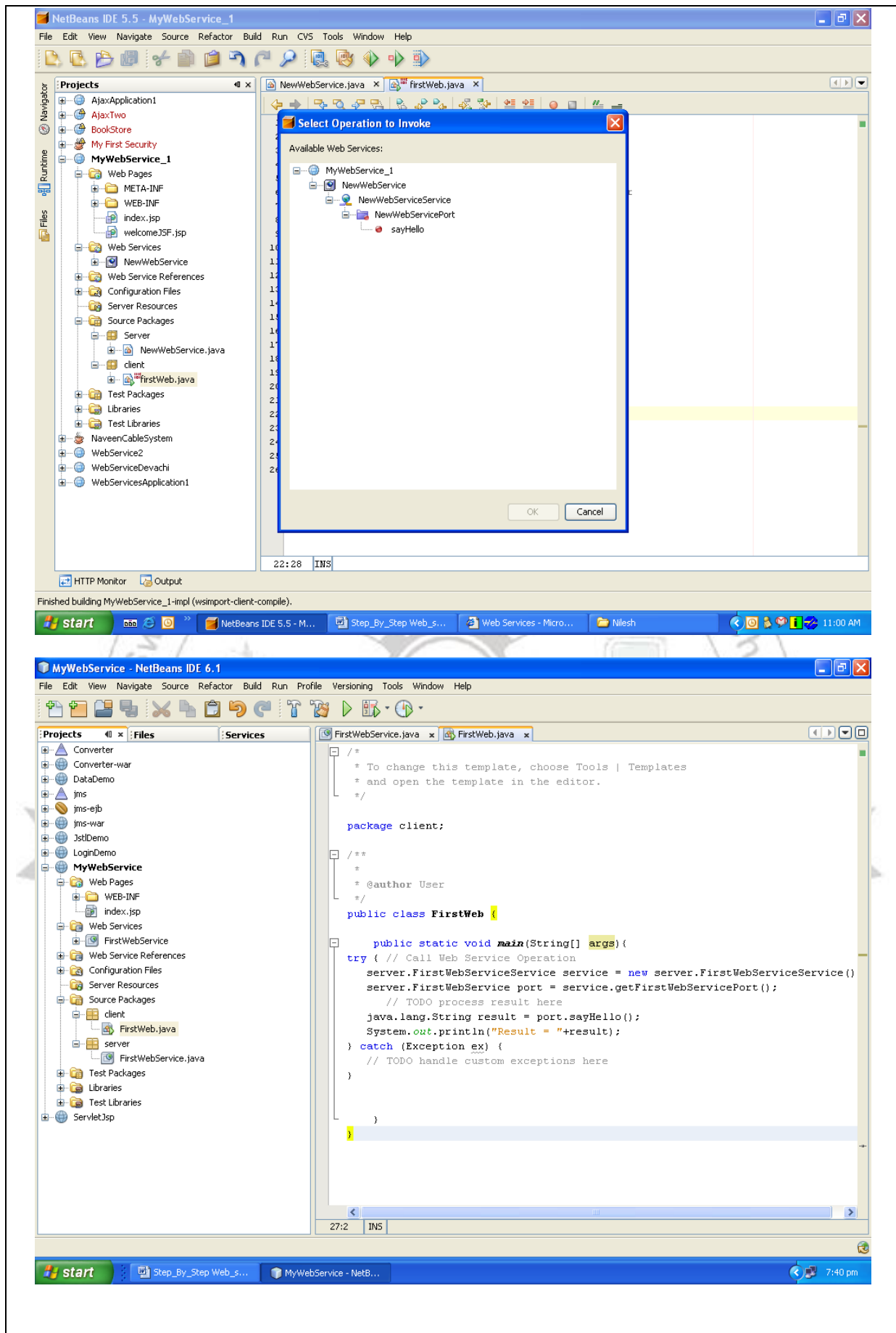
Step – 6)

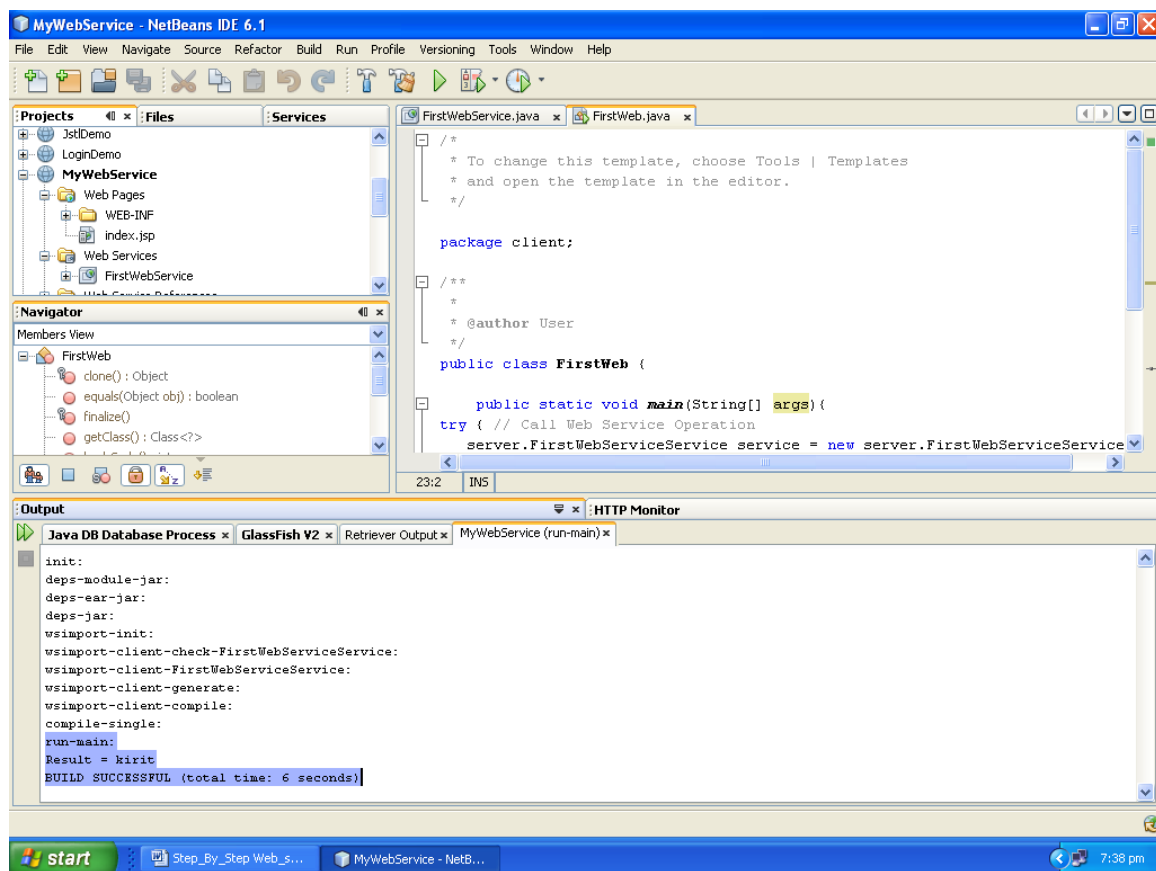
- Open the Java File in client Package. In that where main () is written, Right click on main.**
- Select Web-Service client Resources**
- Call Web-Service operation.**



D) select “sayHello” ok.

Compile the file and Run Client FirstWeb.



OUTPUT.**Problem Statement:**