CALLING SERVER SIDE C PROGRAM FROM JAVA THROUGH JNI AND RMI

REPORT OF PROJECT SUBMITTED FOR PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF

BACHELOR OF TECHNOLOGY
In
COMPUETR SCIENCE ENGINEERING
By

Rituparna Bhowmick
REGISTRATION NO.-091170110037
UNIVERSITY ROLL NO.-09117001028

Sagnik Mitra
REGISTRATION NO.-091170110039
UNIVERSITY ROLL NO.-09117001032

Anirban Jash
REGISTRATION NO – 091170110005
UNIVERSITY ROLL NO -09117001040

UNDER THE SUPERVISION OF
Ms. Sukla Banerjee
Assistant professor in
COMPUTER SCIENCE ENGINEERING DEPT.

AT
RCC INSTITUTE OF INFORMATION TECHNOLOGY
[Affiliated to West Bengal University of Technology]
CANAL SOUTH ROAD, BELIAGHATA, KOLKATA – 700015

RCC INSTITUTE OF INFORMATION TECHNOLOGY

KOLKATA – 7OOO15, INDIA

CERTIFICATE

The report of the Project titled CALLING SERVER SIDE C PROGRAM FROM JAVA THROUGH JNI AND RMI
submitted by **Rituparna Bhowmick** (Roll No. :CSE/2009/007) of B-Tech (CSE), **Anirban Jash** (Roll No.: CSE/2009/049) of B. Tech. (CSE) ,**Sagnik Mitra** (Roll No. : CSE/2009/024) of B.tech(CSE) 8thSemester of 2013 has been prepared under our supervision for the partial fulfilment of the requirements for B Tech CSE degree in West Bengal. University of Technology
The report is hereby forwarded.

[Mrs. Sukla Banerjee]
Dept. of COMPUTER SCIENCE
RCCIIT,Kolkata
(Internal Supervisor)

Countersigned by

………………………………..
[Mr. Harinandan Tunga]
HOD Department CSE
RCC Institute of Information Technology, Kolkata – 700 015, India

ACKNOWLEDGEMENT

We express our sincere gratitude to **Ms. Sukla Banerjee** of Department of CSE, RCCIIT and for extending her valuable times for us to take up this problem as a Project. She was a source of constant inspiration and help throughout the project work.

We are also indebted to **Mr. Harinandan Tunga** , H.O.D, Department of Computer Science , RCCIIT for his unconditional help and inspiration.

Date: /05/ 2013

Rituparna Bhowmick
Reg. No.: 091170110037
Roll No. :09117001028

Sagnik Mitra
Reg. No.: 091170110039
Roll No. :09117001032
&
Anirban Jash
Reg. No: 091170110005
Roll No.:09117001040

RCC INSTITUTE OF INFORMATION TECHNOLOGY

KOLKATA – 7OOO15, INDIA

<u>CERTIFICATE of ACCEPTANCE</u>

The report of the Project titled CALLING SERVER SIDE C PROGRAM FROM JAVA THROUGH JNI AND RMI

submitted by

Rituparna Bhowmick
REGISTRATION NO.-091170110037
UNIVERSITY ROLL NO.-09117001028

Sagnik Mitra
REGISTRATION NO.-091170110039
UNIVERSITY ROLL NO.-09117001032

Anirban Jash
REGISTRATION NO – 091170110005
UNIVERSITY ROLL NO -09117001040

of B. Tech. (CSE) 8th
Semester of
2012) is hereby recommended to be accepted for the partial fulfillment of the requirements for B Tech(CSE) degree in West Bengal. University of Technology

Name of the Examiner	**Signature with Date**
1. ………………………	……………………..
2. ………………………	……………………..
3. ………………………	…………………….
4. ………………………	…………………….

ABSTRACT

The JNI is a native programming interface. It allows Java code that runs inside a Java Virtual Machine (JVM) to interoperate with applications and libraries written in other programming languages, such as C, C++, and assembly. JNI enables one to write native methods to handle situations when an application cannot be written entirely in the Java programming language.
The JNI framework lets a native method use Java objects in the same way that Java code uses these objects. The advantage of JNI are that using JNI, we can access c and c++ code which adds performance boost to JAVA.JNI allows JAVA to access some hardware features using other languages like c and c++.

RMI (Remote Method Invocation) is a way that a programr, using the Java programming language and development environment, can write object-oriented programming in which objects on different computers can interact in a distributed network.

Table of Contents

1. Introduction

This project CALLING SERVER SIDE C PROGRAM FROM JAVA THROUGH JNI AND RMI is about making a request from the client side that is java program and getting the final results from the server side that is the C program via JNI.

Through this project we can make an effort to let a native method use Java objects. Through this project we can access c and c++ code which adds performance boost to JAVA. This project allows JAVA to access some hardware features using other languages like c and c++.

2. Problem Analysis

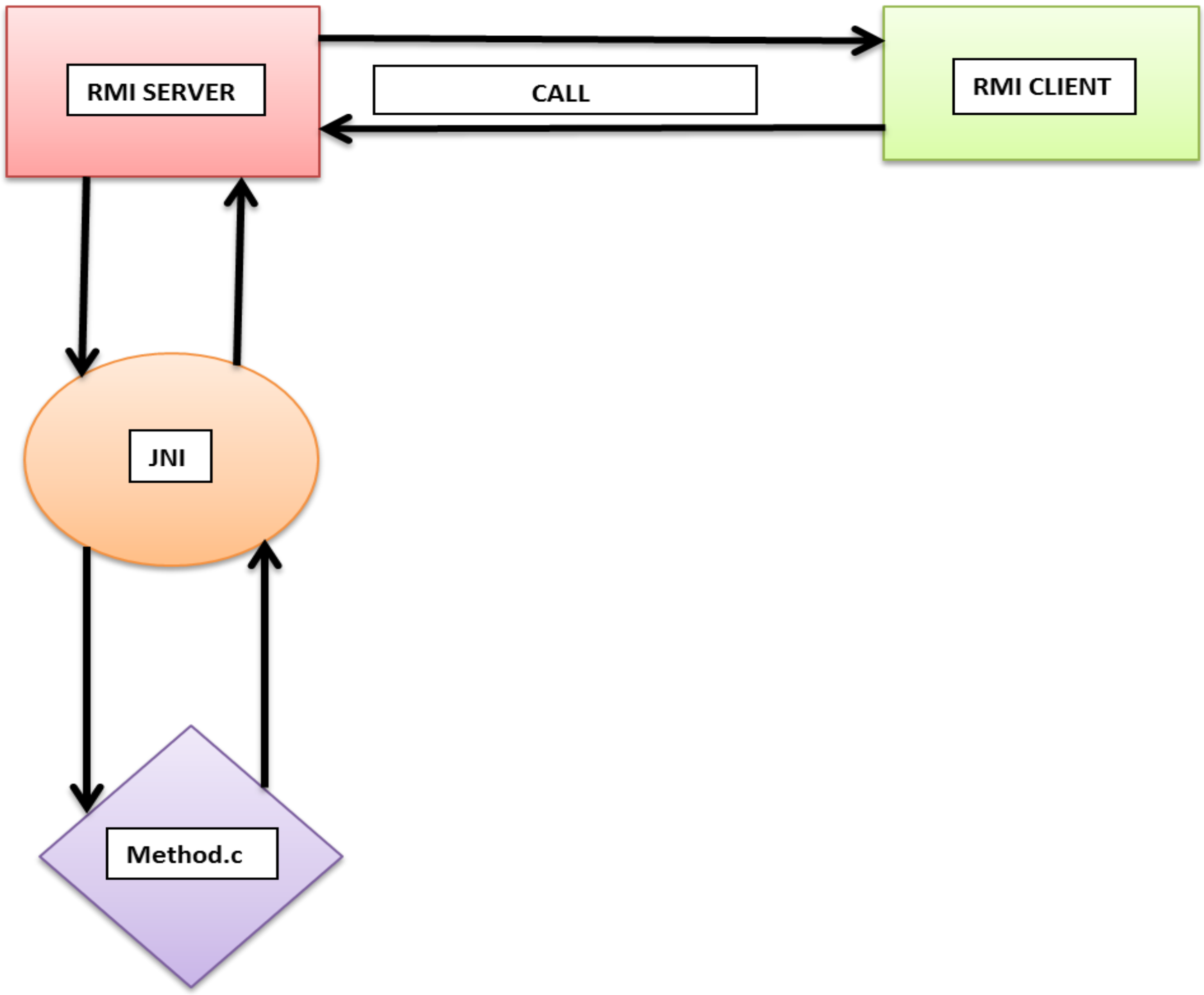
When an application cannot be written entirely in the Java programming language, we use JNI that enables one to write native methods to handle such situations, e.g. when the standard Java class library does not support the platform-specific features or program library. It is also used to modify an existing application—written in another programming language—to be accessible to Java applications.

We are using RMI because RMI (Remote Method Invocation) is a way that a programr, using the Java programming language and development environment, can write object-oriented programming in which objects on different computers can interact in a distributed network. RMI is the Java version of what is generally known as a remote procedure call (RPC), but with the ability to pass one or more objects along with the request. The object can include information that will change the service that is performed in the remote computer. For example, when a user at a remote computer fills out an expense account, the Java program interacting with the user could communicate, using RMI, with a Java program in another computer that always had the latest policy about expense reporting. In reply, that program would send back an object and associated method information that would enable the remote computer program to screen the user's expense account data in a way that was consistent with the latest policy. The user and the company both would save time by catching mistakes early. Whenever the company policy changed, it would require a change to a program in only one computer. An RMI request is a request to invoke the method of a remote object. The request has the same syntax as a request to invoke an object method in the same (local) computer. In general, RMI is designed to preserve the object model and its advantages across a network.

So , here in our project RMI client (i.e., java client) makes a request to the RMI server (c server) and through JNI.

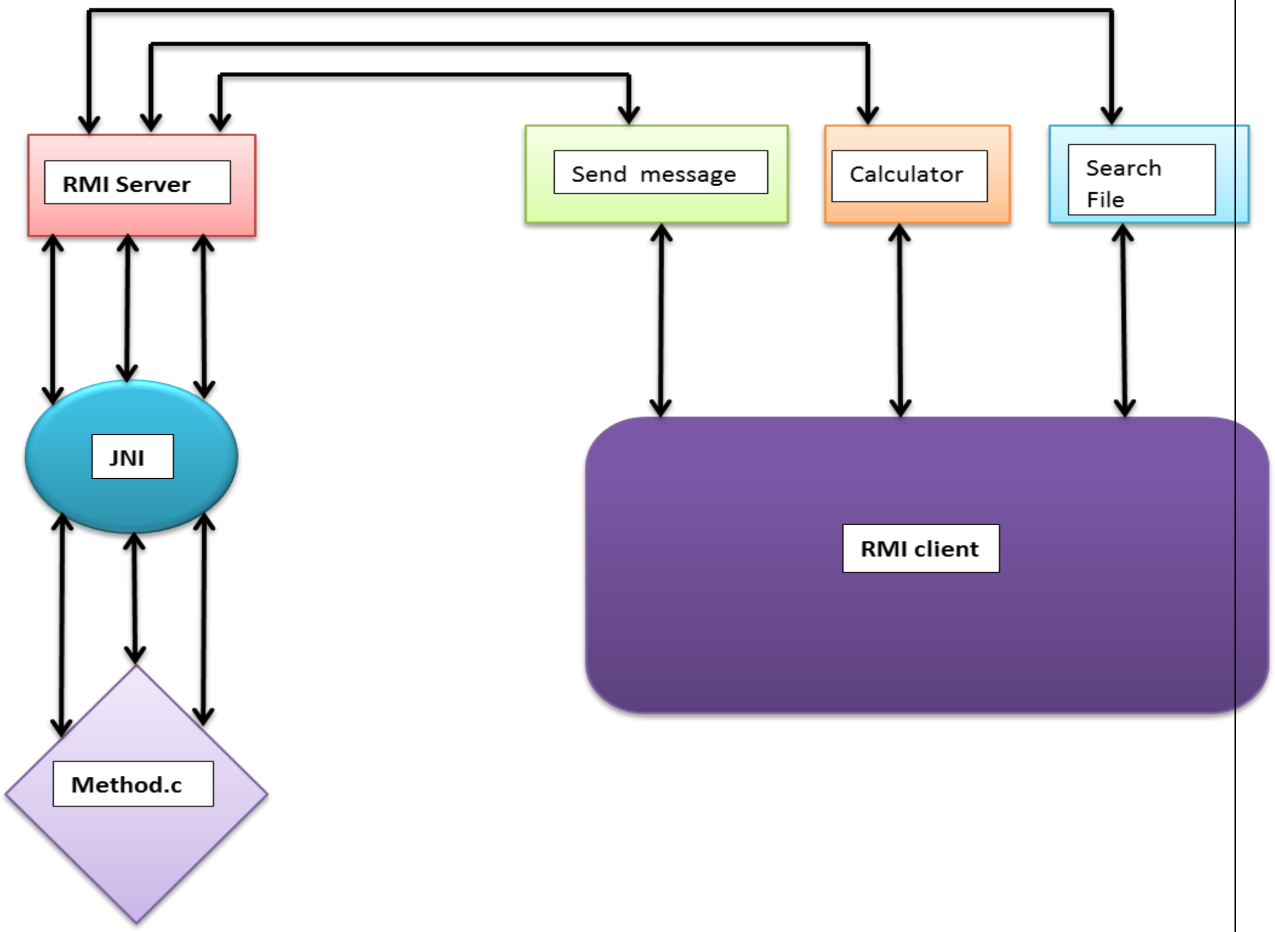
3. PROBLEM DISCUSSION

Our aim is to send a request from a RMI client (java client) to the RMI server (c server) through JNI. And getting the results back though the server via JNI.



Here the RMI client makes a request to the RMI server and RMI server through JNI

calls the methods.c program. methods.c returns the result through JNI and RMI server to the RMI client.

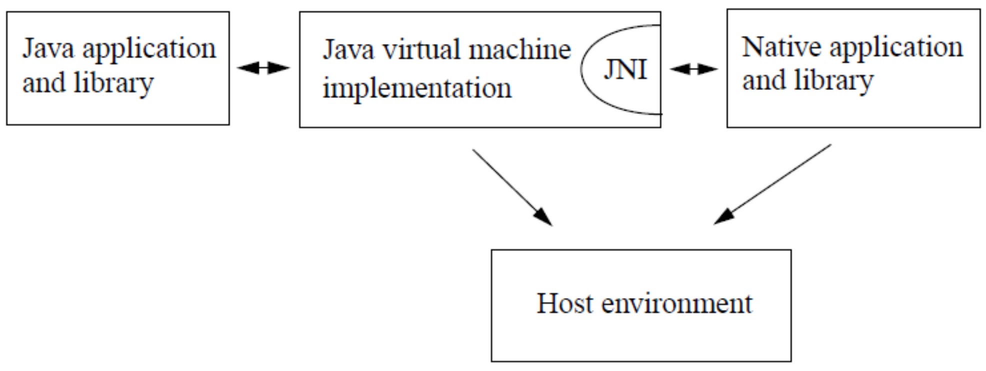


When we are sending a request for example sending a message from RMI client to the RMI server,that time we have a program nativethisorthatserverimpl.java and the client is calling that program in the RMI server.Now the nativethisorthatserverimpl.java calls the native method that is method.c program.The method.c program processes the result and sends it back through the RMI server to the RMI client.

4. JNI

➢Role of JNI :

When the Java platform is deployed on top of host environments, it may become
desirable or necessary to allow Java applications to work closely with native code
written in other languages. Programrs have begun to adopt the Java platform to
build applications that were traditionally written in C and C++. Because of the existing investment in legacy code, however, Java applications will coexist with C
and C++ code for many years to come.
The JNI is a powerful feature that allows us to take advantage of the Java
platform, but still utilize code written in other languages. As a part of the Java virtual
machine implementation, the JNI is a *two-way* interface that allows Java
applications to invoke native code and vice versa.


Java application and library
Java virtual machine implementation
JNI
Native application and library
Host environment

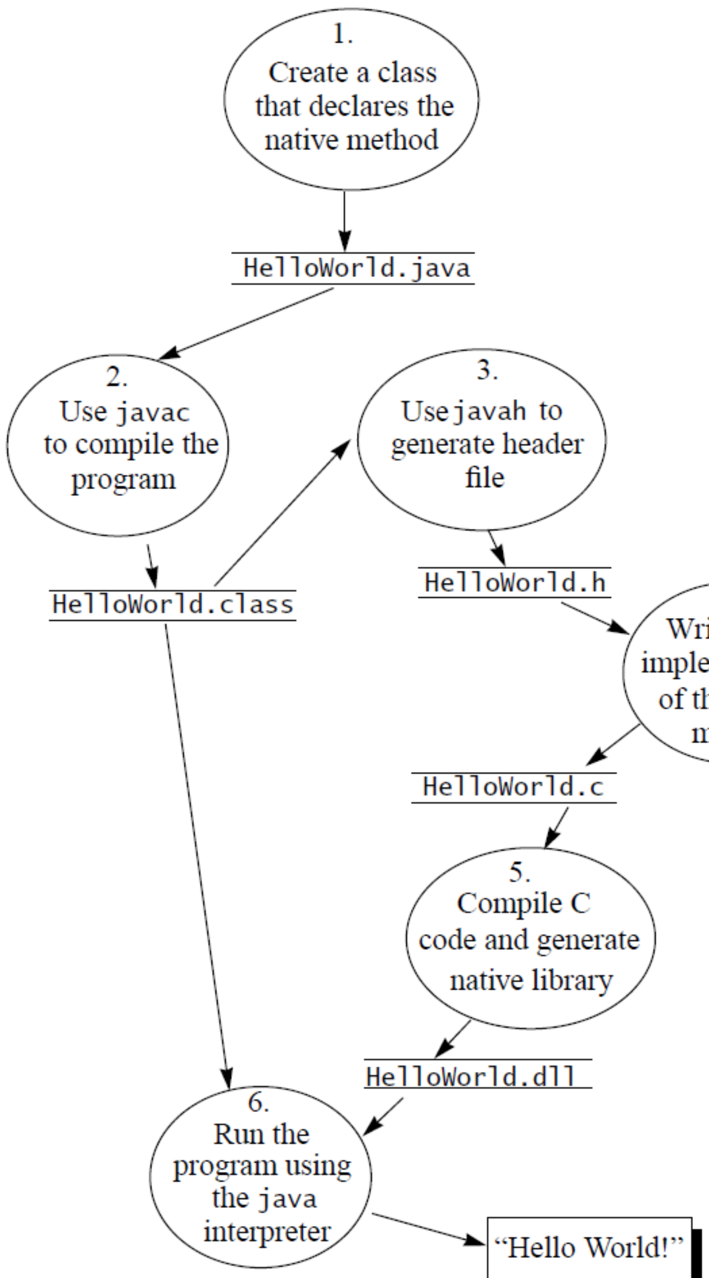
➢Using the JNI:

1. Create a class (HelloWorld.java) that declares the native method.

2. Use javac to compile the HelloWorld source file, resulting in the class file
HelloWorld.class. The javac compiler is supplied with JDK or Java 2 SDK
releases.

3. Use javah -jni to generate a C header file (HelloWorld.h) containing the
function prototype for the native method implementation. The javah tool is
provided with JDK or Java 2 SDK releases.

4. Write the C implementation (HelloWorld.c) of the native method.
5. Compile the C implementation into a native library, creating HelloWorld.dll
or libHelloWorld.so. Use the C compiler and linker available on the host
environment.
6. Run the HelloWorld program using the java runtime interpreter. Both the
class file (HelloWorld.class) and the native library (HelloWorld.dll or
libHelloWorld.so) are loaded at runtime.



1. Create a class that declares the native method
HelloWorld.java
2. Use javac to compile the program
3. Use javah to generate header file
HelloWorld.class
HelloWorld.h
4. Write the C implementation of the native method
HelloWorld.c
5. Compile C code and generate native library
HelloWorld.dll
6. Run the program using the java interpreter
"Hello World!"

When we are embedding C in Java we have the following steps:
Declare the method using the keyword native, provide no implementation.
Make sure the Java loads the needed library
Run the javah utility to generate names/headers
Implement the method in C
Compile as a shared library

We are illustrating this with an example below:

```
class HelloWorld
{
   public          native          void
displayHelloWorld();
   static
     {
         System.loadLibrary("hello");
   }
   public  static  void  main(String[]
args)
   {
         new
HelloWorld().displayHelloWorld();
   }
}
```

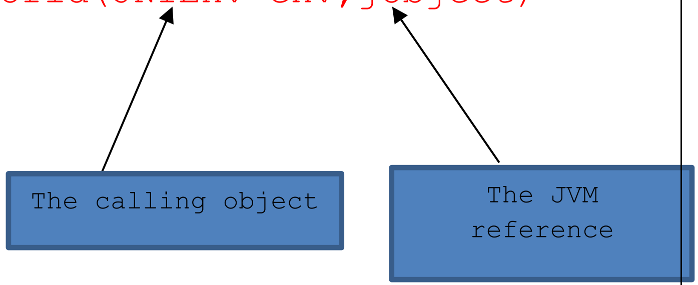
When generating JNI Header we have the following steps:

Compile HelloWorld.java
-javac HelloWorld.java

Generate HelloWorld.h

```
-javah HelloWorld
HelloWorld.h
#include "jni.h"
/* Header for class HelloWorld */
#ifndef _Included_HelloWorld
#define _Included_HelloWorld
```

```
#ifdef __cplusplus
      extern “C” {
#endif

/*
 * Class: HelloWorld
 * Method: displayHelloWorld
 * Signature: ()V
 */
JNIEXPORT void JNICALL
Java_HelloWorld_displayHelloWorld(JNIEnv*env,jobject)
;

#ifdef __cplusplus
  }
#endif
#endif
HelloWorldImp.c
#include <jni.h>
#include "HelloWorld.h"
#include <stdio.h>
JNIEXPORT void JNICALL
Java_HelloWorld_displayHelloWorld(JNIEnv*env,jobject
obj)
{
    printf("Hello world!\n");
    return;
}
```



5. RMI

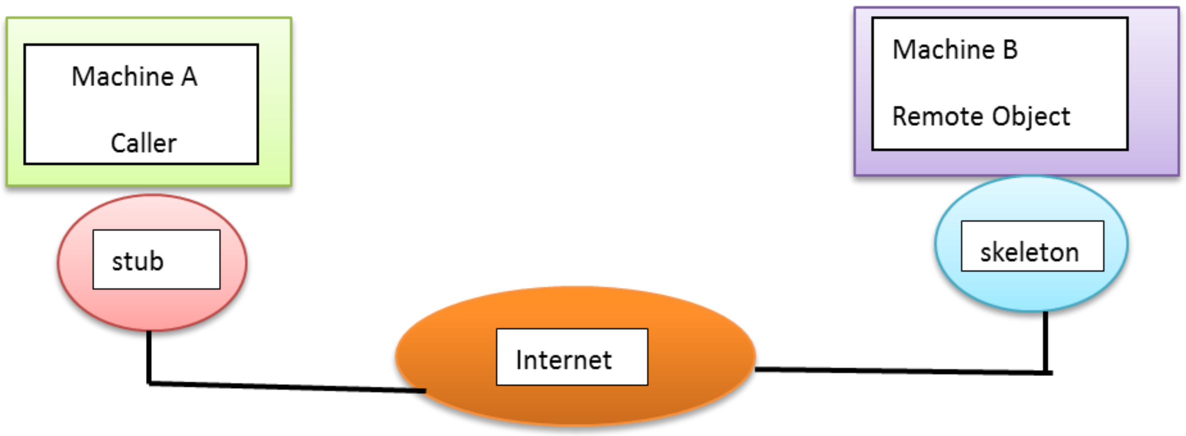
The remote method invocation is an API that provides a mechanism to create distributed application IN JAVA.The rmi allows an object to invoke methods on an object running in another JVM.The RMI provides remote communication between the application using two objects stub and skeleton.
RMI uses stub and skeleton object for communication with the remote object.A remote object is an object whose method can be invoked from another JVM.

➢STUB:

The stub is an object which acts as a gateway for the client side.All the outgoing request are routed through it.It resides at the client side and represents the remote object.When the caller invokes method the stub object,it does the following tasks:
1.it initiates a connection with remote virtual machine(JVM)
2.It writes and transmits (marshals)the parameters to the remote virtual machine(JVM).
3.It waits for the result.
4.It reads (unmarshals)the return value or exception.
5.It finally,returns the value to the caller.

➢SKELETON:

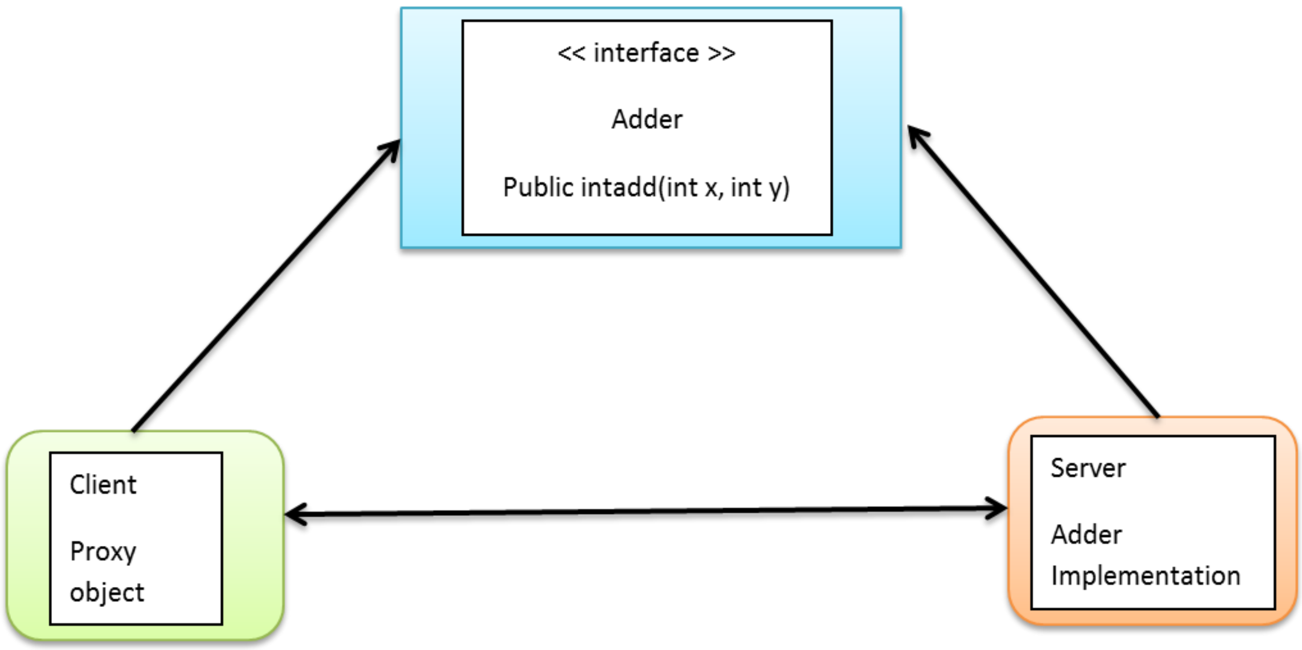
The skleton is an object which acts as a gateway for the server side object.All the incoming request are routed through it when the skeleton receives the incoming request it does the following tasks:
1.it reads the parameter for the remothe method.
2. it invokes the method on the actual remote object.
3it writes and transmits (marshals) the result to the caller.



Steps to write RMI program
1.Create the remote interface
2.provide the implementation of remote interface.
3. compile the implementation class and create the stub and skeleton object using rmic tool.
4.start the registry service by rmiregistry tool.
5. create and start remote application
6. create and start client application.

Example of creating simple RMI application:

In this example we have followed all the six steps to create and run the rmi application. The client application need only two files remote interface and client application. In the rmi application both client and server interacts with the remote interface the client application invokes methods on the proxy object , rmi sends the request to the remote jvm. The return value is send back to the proxy object and then to the client application.



6. IMPLEMENTATION DETAILS

In the RMI Server we have the following :

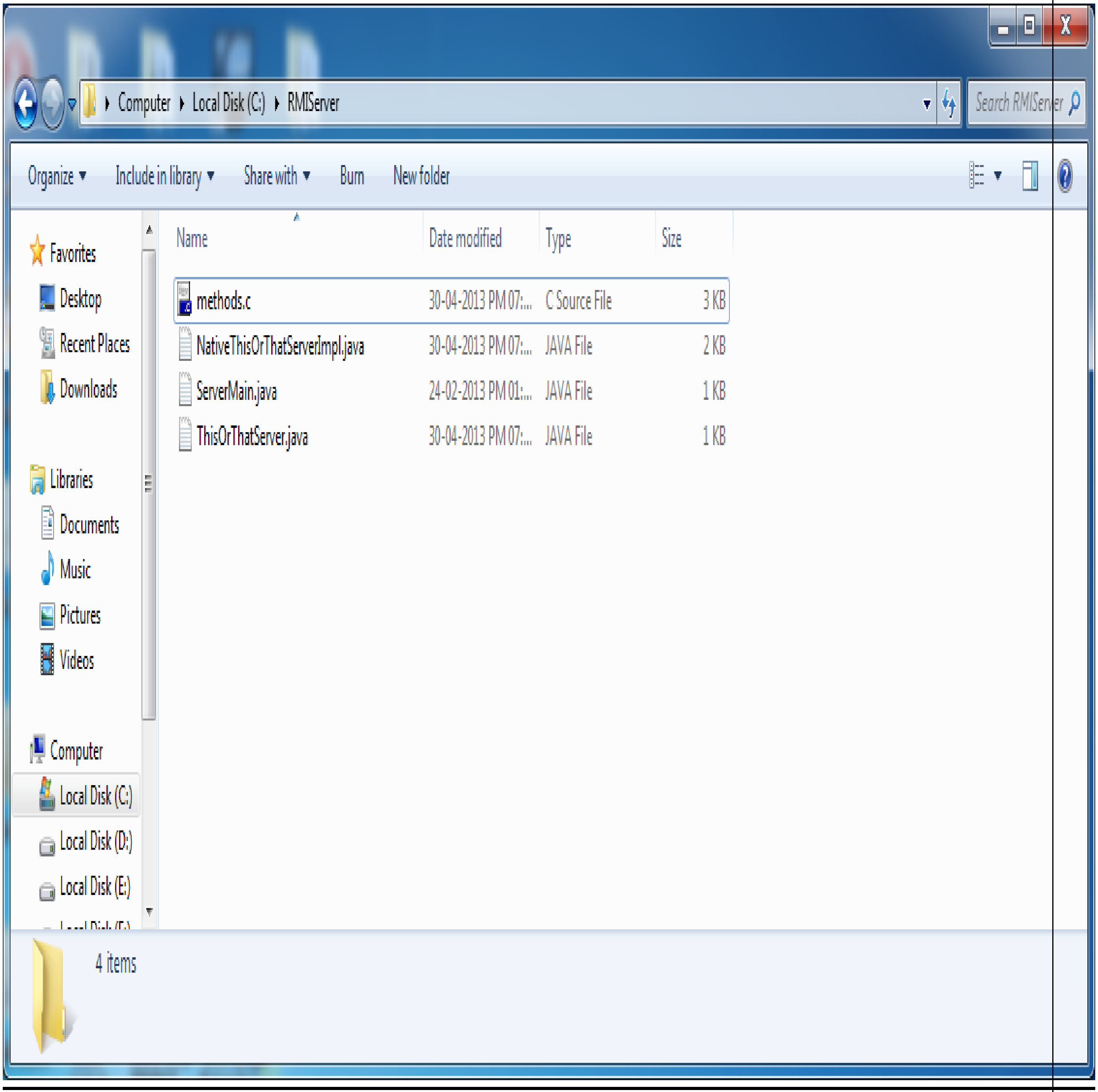
1. ThisorThatServer.java : Here we just declare dothis, dothat , add, substract , multiply , division and search file method.
2. NativeThisorThatServer.java: Here basically we implement dothis, dothat , add, substract , multiply , division and search file method.
3. NativeThisorThatServerimpl.h : This is the JNI header file.
4. method.c : here dothis, dothat , add, substract , multiply , division and search file operations are performed.
5. ServerMain.java : while running this the server gets ready.

In the RMIClientGUI we have the following :

1. Client.java : here the operations which have to be performed are written. These are main , average , calculator and search file.
2. Calculator.java : in this prograg the calculation operations that had tobe performed are written. They are mainly addition substraction , multiplication and division.
3. Searchfile.java : this program search a particular file in server.

7.Screen shots :

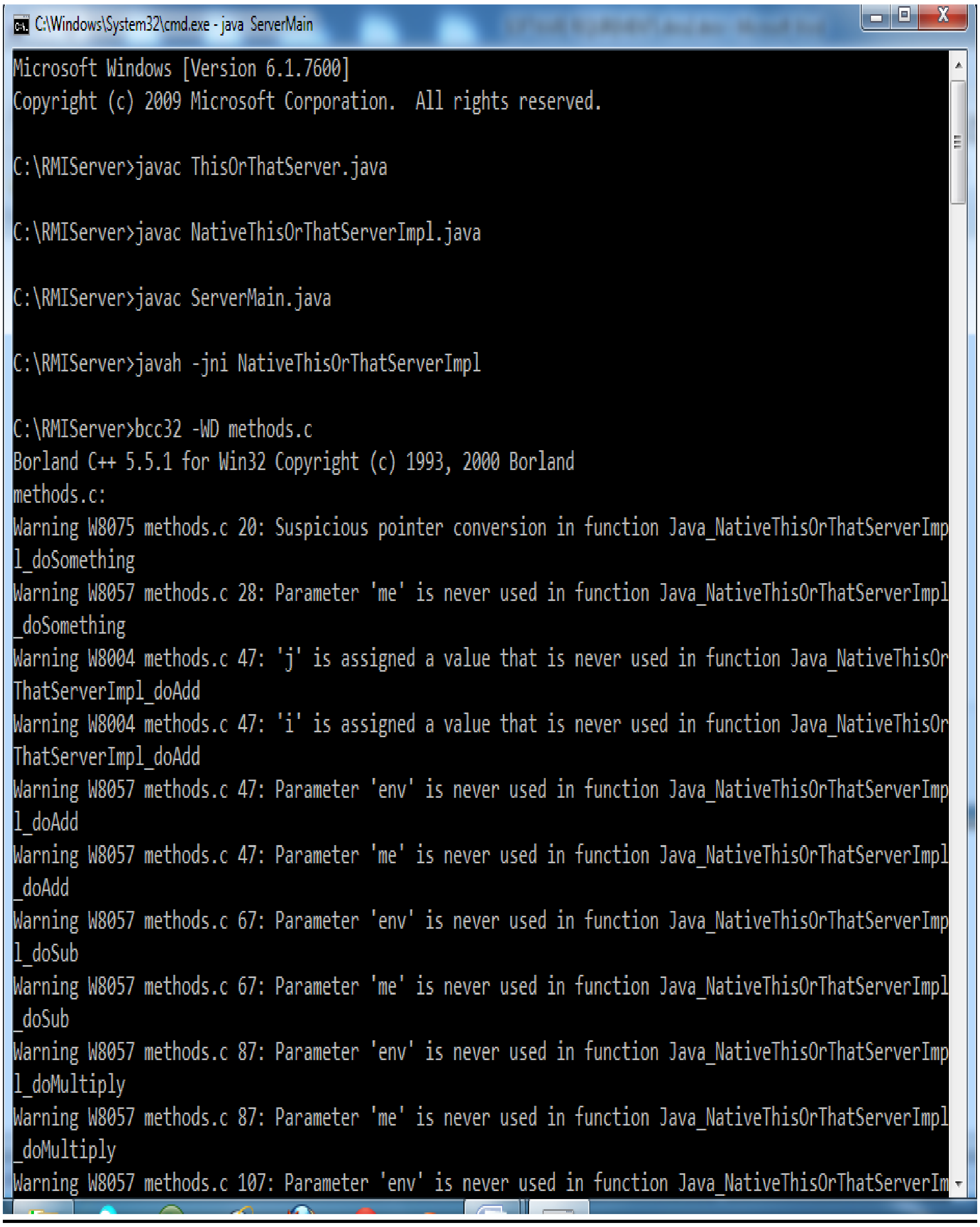
➢Creation of RMI server



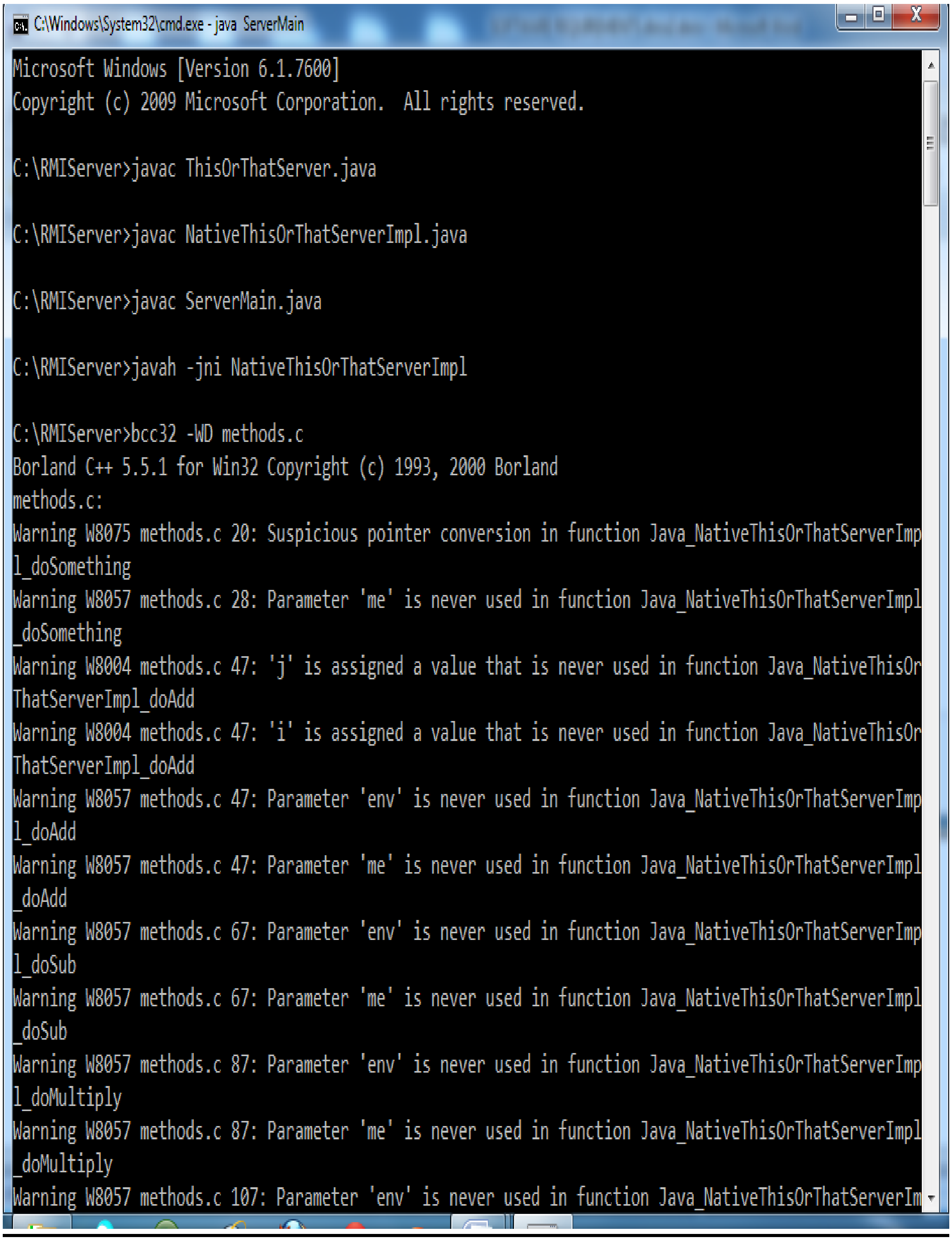
Computer ▸ Local Disk (C:) ▸ RMIServer

Name	Date modified	Type	Size
methods.c	30-04-2013 PM 07:...	C Source File	3 KB
NativeThisOrThatServerImpl.java	30-04-2013 PM 07:...	JAVA File	2 KB
ServerMain.java	24-02-2013 PM 01:...	JAVA File	1 KB
ThisOrThatServer.java	30-04-2013 PM 07:...	JAVA File	1 KB

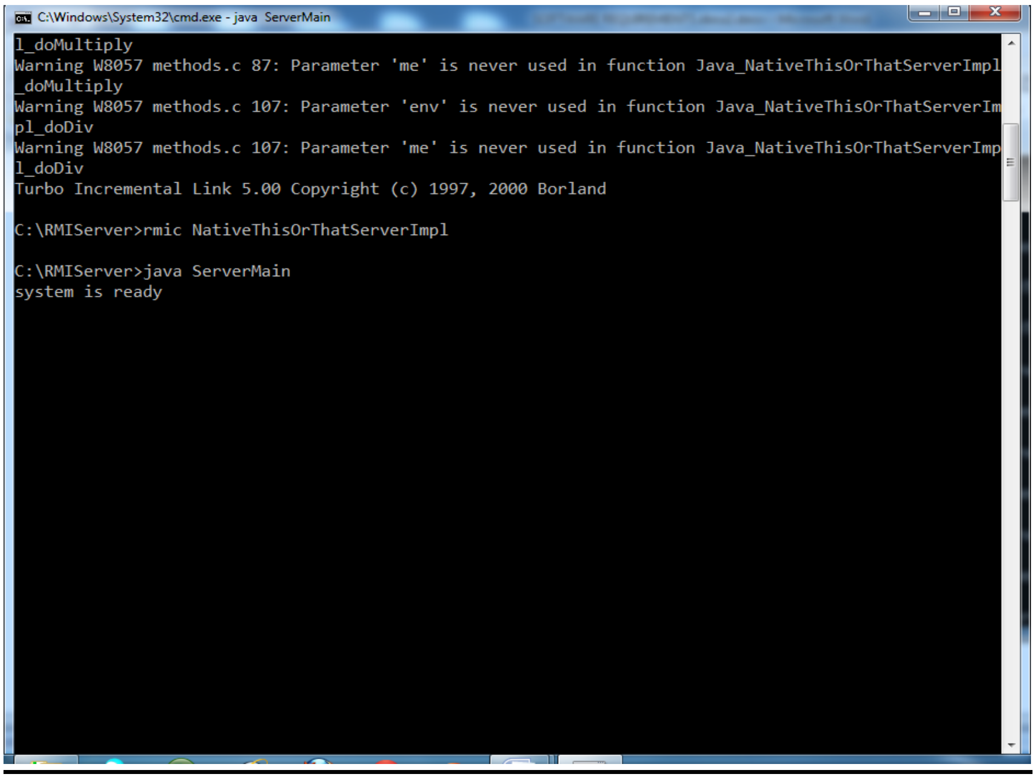
4 items


C:\Windows\System32\cmd.exe - java ServerMain

```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\RMIServer>javac ThisOrThatServer.java

C:\RMIServer>javac NativeThisOrThatServerImpl.java

C:\RMIServer>javac ServerMain.java

C:\RMIServer>javah -jni NativeThisOrThatServerImpl

C:\RMIServer>bcc32 -WD methods.c
Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland
methods.c:
Warning W8075 methods.c 20: Suspicious pointer conversion in function Java_NativeThisOrThatServerImp
l_doSomething
Warning W8057 methods.c 28: Parameter 'me' is never used in function Java_NativeThisOrThatServerImpl
_doSomething
Warning W8004 methods.c 47: 'j' is assigned a value that is never used in function Java_NativeThisOr
ThatServerImpl_doAdd
Warning W8004 methods.c 47: 'i' is assigned a value that is never used in function Java_NativeThisOr
ThatServerImpl_doAdd
Warning W8057 methods.c 47: Parameter 'env' is never used in function Java_NativeThisOrThatServerImp
l_doAdd
Warning W8057 methods.c 47: Parameter 'me' is never used in function Java_NativeThisOrThatServerImpl
_doAdd
Warning W8057 methods.c 67: Parameter 'env' is never used in function Java_NativeThisOrThatServerImp
l_doSub
Warning W8057 methods.c 67: Parameter 'me' is never used in function Java_NativeThisOrThatServerImpl
_doSub
Warning W8057 methods.c 87: Parameter 'env' is never used in function Java_NativeThisOrThatServerImp
l_doMultiply
Warning W8057 methods.c 87: Parameter 'me' is never used in function Java_NativeThisOrThatServerImpl
_doMultiply
Warning W8057 methods.c 107: Parameter 'env' is never used in function Java_NativeThisOrThatServerIm
```

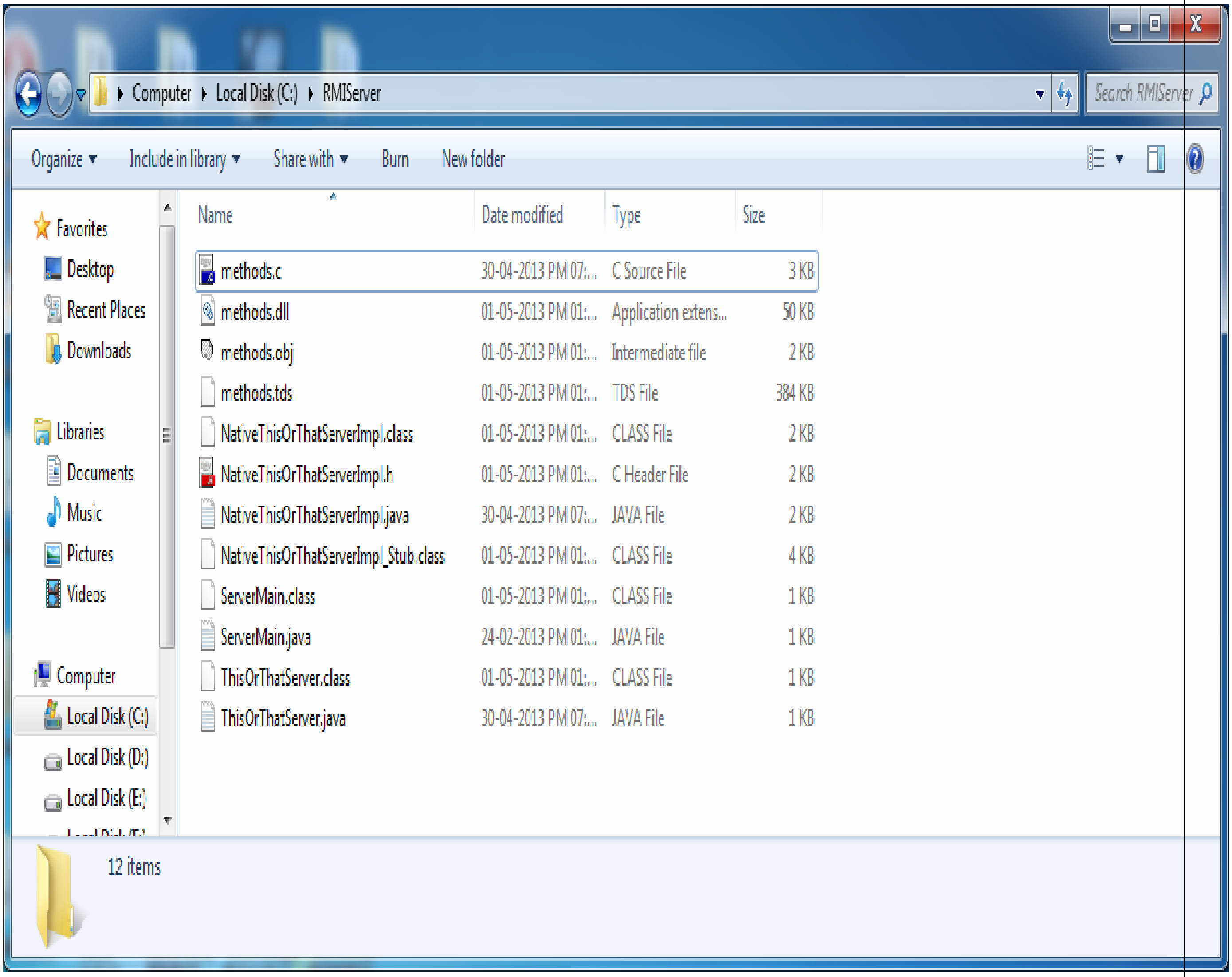

C:\Windows\System32\cmd.exe - java ServerMain

```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\RMIServer>javac ThisOrThatServer.java

C:\RMIServer>javac NativeThisOrThatServerImpl.java

C:\RMIServer>javac ServerMain.java

C:\RMIServer>javah -jni NativeThisOrThatServerImpl

C:\RMIServer>bcc32 -WD methods.c
Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland
methods.c:
Warning W8075 methods.c 20: Suspicious pointer conversion in function Java_NativeThisOrThatServerImp
l_doSomething
Warning W8057 methods.c 28: Parameter 'me' is never used in function Java_NativeThisOrThatServerImpl
_doSomething
Warning W8004 methods.c 47: 'j' is assigned a value that is never used in function Java_NativeThisOr
ThatServerImpl_doAdd
Warning W8004 methods.c 47: 'i' is assigned a value that is never used in function Java_NativeThisOr
ThatServerImpl_doAdd
Warning W8057 methods.c 47: Parameter 'env' is never used in function Java_NativeThisOrThatServerImp
l_doAdd
Warning W8057 methods.c 47: Parameter 'me' is never used in function Java_NativeThisOrThatServerImpl
_doAdd
Warning W8057 methods.c 67: Parameter 'env' is never used in function Java_NativeThisOrThatServerImp
l_doSub
Warning W8057 methods.c 67: Parameter 'me' is never used in function Java_NativeThisOrThatServerImpl
_doSub
Warning W8057 methods.c 87: Parameter 'env' is never used in function Java_NativeThisOrThatServerImp
l_doMultiply
Warning W8057 methods.c 87: Parameter 'me' is never used in function Java_NativeThisOrThatServerImpl
_doMultiply
Warning W8057 methods.c 107: Parameter 'env' is never used in function Java_NativeThisOrThatServerIm
```



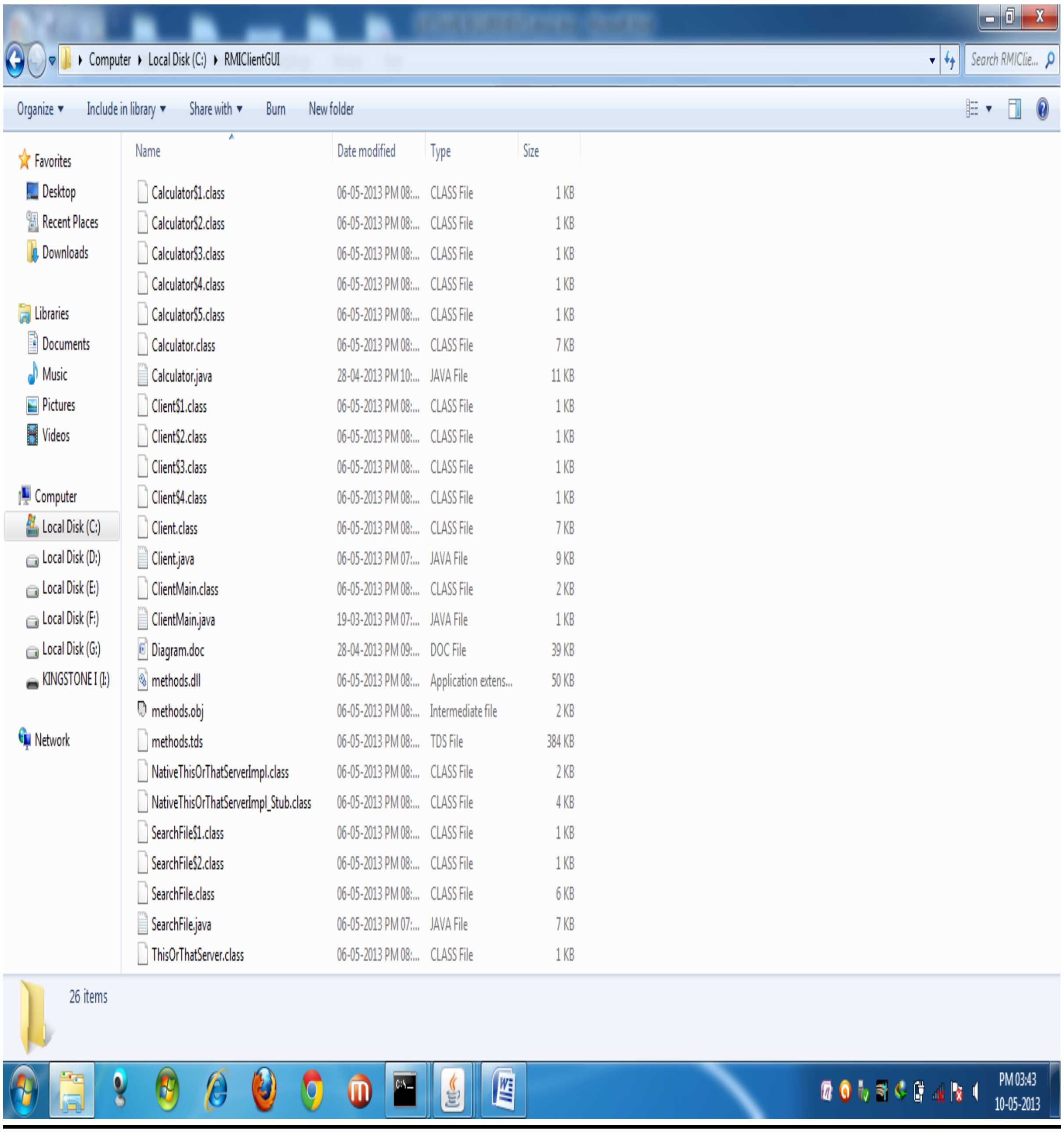
```
C:\Windows\System32\cmd.exe - java  ServerMain
l_doMultiply
Warning W8057 methods.c 87: Parameter 'me' is never used in function Java_NativeThisOrThatServerImpl
_doMultiply
Warning W8057 methods.c 107: Parameter 'env' is never used in function Java_NativeThisOrThatServerIm
pl_doDiv
Warning W8057 methods.c 107: Parameter 'me' is never used in function Java_NativeThisOrThatServerImp
l_doDiv
Turbo Incremental Link 5.00 Copyright (c) 1997, 2000 Borland

C:\RMIServer>rmic NativeThisOrThatServerImpl

C:\RMIServer>java ServerMain
system is ready
```

- Here we are Compiling ThisOrThatServer.java file which creates ThisOrThatServer.class file.
- Compiling NativeThisOrThatServerImpl.java file creates NativeThisOrThatServerImpl.class file.
- Compiling ServerMain.java file creates ServerMain.class file.
- Javah –jni NativeThisOrThatServerImpl creates the jni header file.
- bcc32 –WD methods.c means bcc32 is compiling methods.c program and it is generating the windows dll file.
- rmic NativeThisOrThatServerImpl creates NativeThisOrThatServerImpl_stub.class file.
- Java ServerMain means we are executing the ServerMain program.
- System is ready means server is ready.



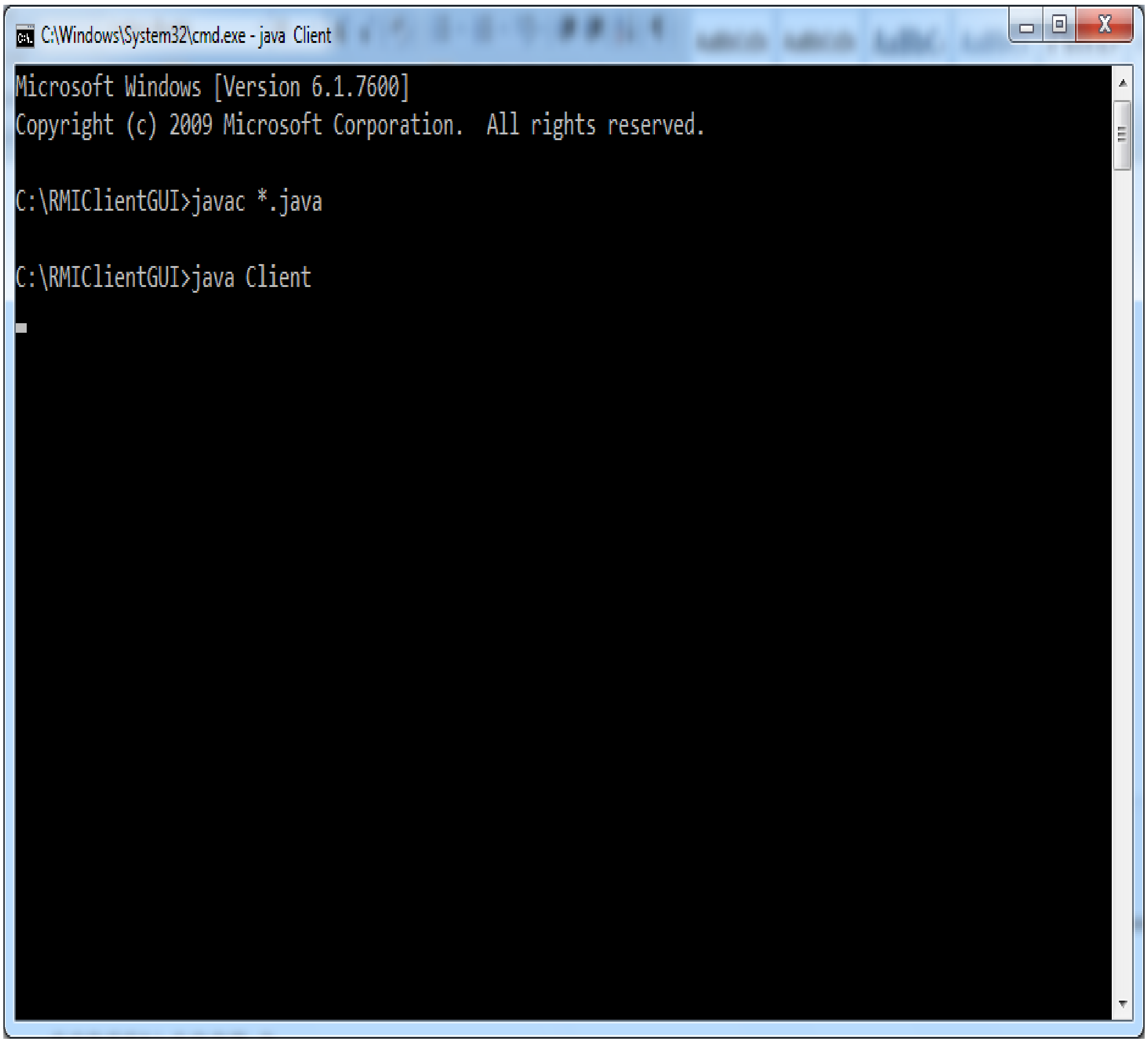
Name	Date modified	Type	Size
methods.c	30-04-2013 PM 07:...	C Source File	3 KB
methods.dll	01-05-2013 PM 01:...	Application extens...	50 KB
methods.obj	01-05-2013 PM 01:...	Intermediate file	2 KB
methods.tds	01-05-2013 PM 01:...	TDS File	384 KB
NativeThisOrThatServerImpl.class	01-05-2013 PM 01:...	CLASS File	2 KB
NativeThisOrThatServerImpl.h	01-05-2013 PM 01:...	C Header File	2 KB
NativeThisOrThatServerImpl.java	30-04-2013 PM 07:...	JAVA File	2 KB
NativeThisOrThatServerImpl_Stub.class	01-05-2013 PM 01:...	CLASS File	4 KB
ServerMain.class	01-05-2013 PM 01:...	CLASS File	1 KB
ServerMain.java	24-02-2013 PM 01:...	JAVA File	1 KB
ThisOrThatServer.class	01-05-2013 PM 01:...	CLASS File	1 KB
ThisOrThatServer.java	30-04-2013 PM 07:...	JAVA File	1 KB

➢Creation of RMI ClientGUI



Name	Date modified	Type	Size
Calculator$1.class	06-05-2013 PM 08:...	CLASS File	1 KB
Calculator$2.class	06-05-2013 PM 08:...	CLASS File	1 KB
Calculator$3.class	06-05-2013 PM 08:...	CLASS File	1 KB
Calculator$4.class	06-05-2013 PM 08:...	CLASS File	1 KB
Calculator$5.class	06-05-2013 PM 08:...	CLASS File	1 KB
Calculator.class	06-05-2013 PM 08:...	CLASS File	7 KB
Calculator.java	28-04-2013 PM 10:...	JAVA File	11 KB
Client$1.class	06-05-2013 PM 08:...	CLASS File	1 KB
Client$2.class	06-05-2013 PM 08:...	CLASS File	1 KB
Client$3.class	06-05-2013 PM 08:...	CLASS File	1 KB
Client$4.class	06-05-2013 PM 08:...	CLASS File	1 KB
Client.class	06-05-2013 PM 08:...	CLASS File	7 KB
Client.java	06-05-2013 PM 07:...	JAVA File	9 KB
ClientMain.class	06-05-2013 PM 08:...	CLASS File	2 KB
ClientMain.java	19-03-2013 PM 07:...	JAVA File	1 KB
Diagram.doc	28-04-2013 PM 09:...	DOC File	39 KB
methods.dll	06-05-2013 PM 08:...	Application extens...	50 KB
methods.obj	06-05-2013 PM 08:...	Intermediate file	2 KB
methods.tds	06-05-2013 PM 08:...	TDS File	384 KB
NativeThisOrThatServerImpl.class	06-05-2013 PM 08:...	CLASS File	2 KB
NativeThisOrThatServerImpl_Stub.class	06-05-2013 PM 08:...	CLASS File	4 KB
SearchFile$1.class	06-05-2013 PM 08:...	CLASS File	1 KB
SearchFile$2.class	06-05-2013 PM 08:...	CLASS File	1 KB
SearchFile.class	06-05-2013 PM 08:...	CLASS File	6 KB
SearchFile.java	06-05-2013 PM 07:...	JAVA File	7 KB
ThisOrThatServer.class	06-05-2013 PM 08:...	CLASS File	1 KB

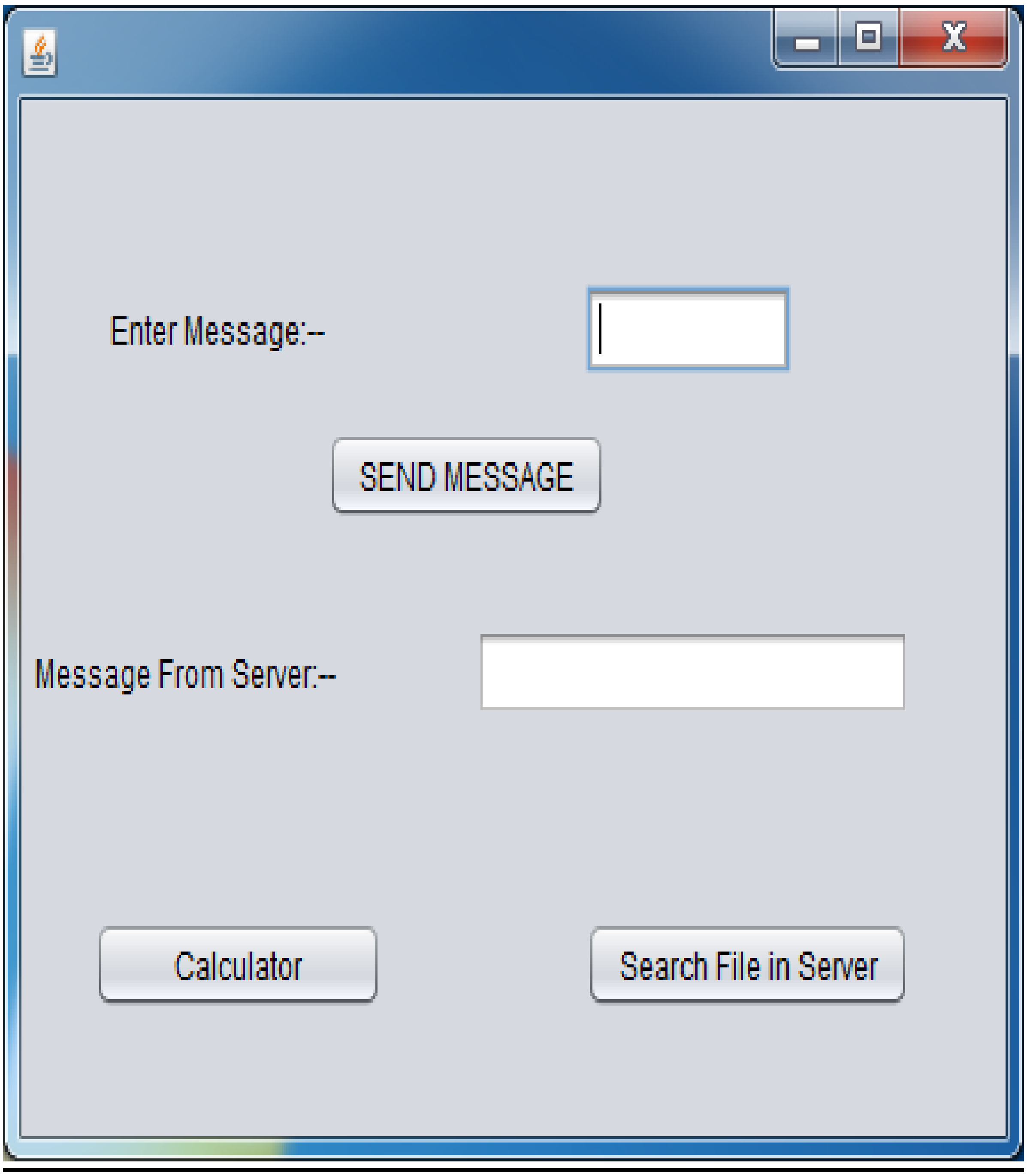
26 items



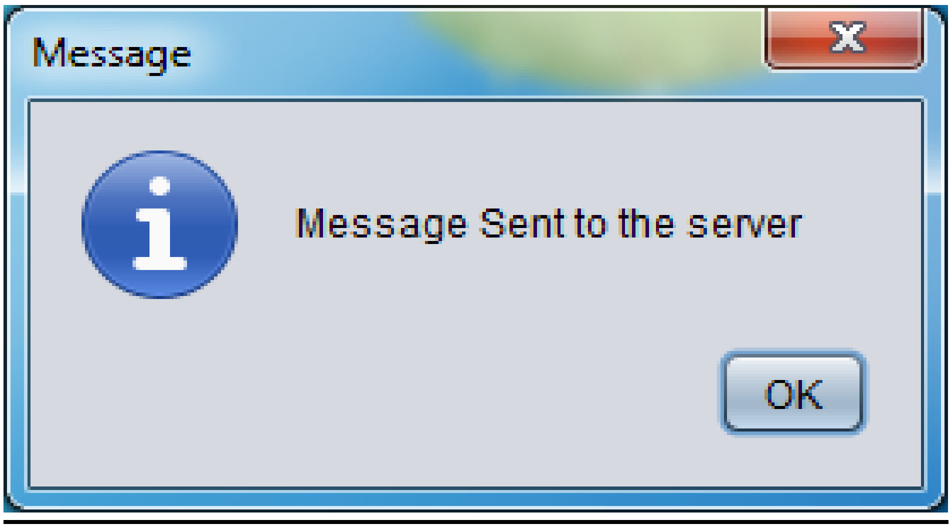
```
C:\Windows\System32\cmd.exe - java  Client
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\RMIClientGUI>javac *.java

C:\RMIClientGUI>java Client
```

- Executing the client.java program we can perform three operations-send message, calculator, and search file in server.

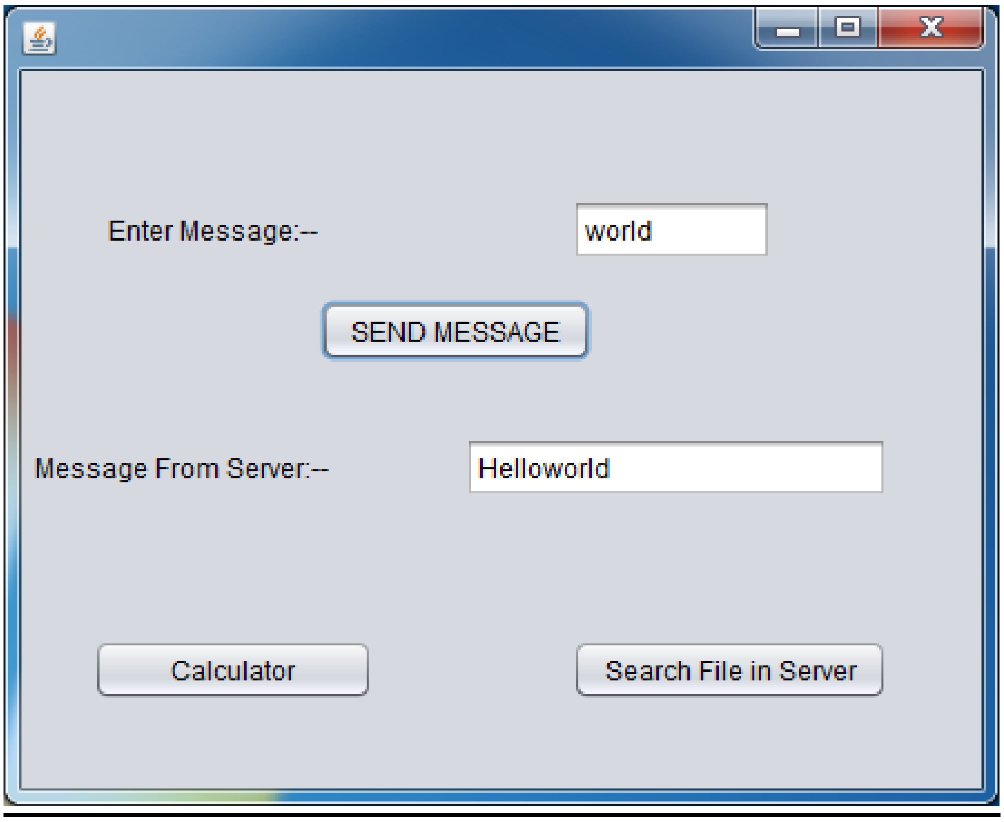
➢Outputs


Enter Message:--
SEND MESSAGE
Message From Server:--
Calculator
Search File in Server

- ## SEND MESSAGE


Message

Message Sent to the server

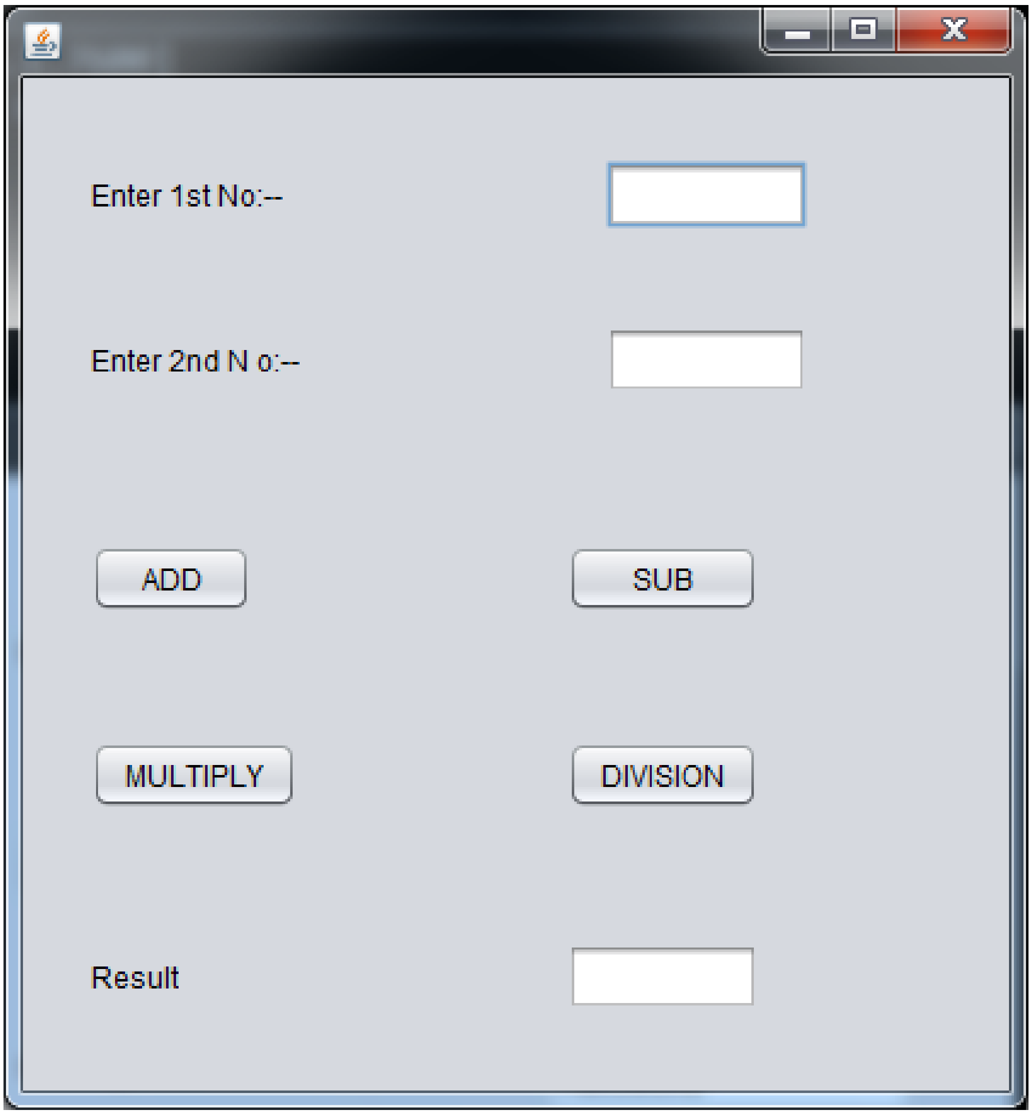

Enter Message:--

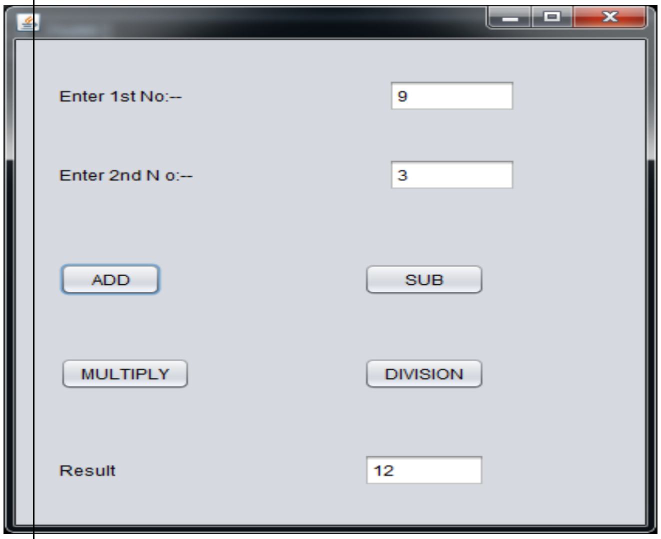
world

Message From Server:--

Helloworld

- ## CALCULATOR

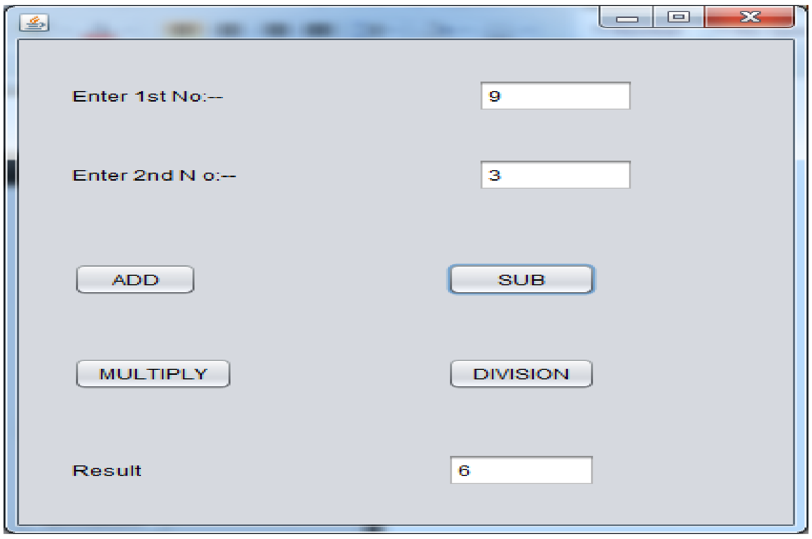




Enter 1st No:-- 9

Enter 2nd N o:-- 3

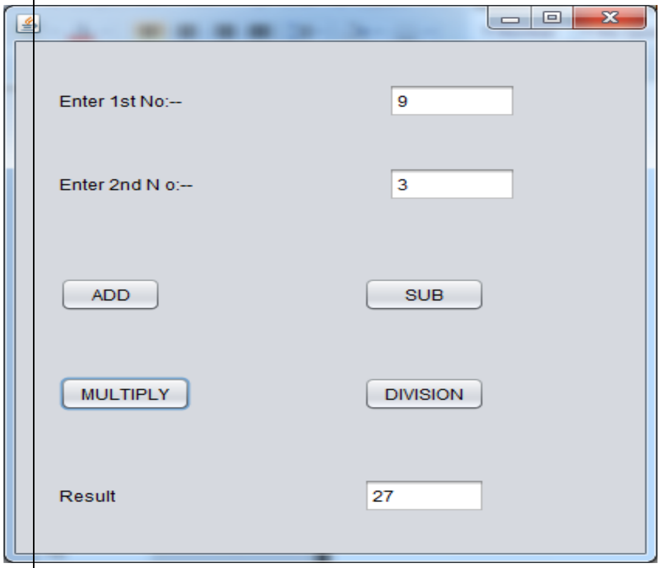
Result 12



Enter 1st No:-- 9

Enter 2nd N o:-- 3

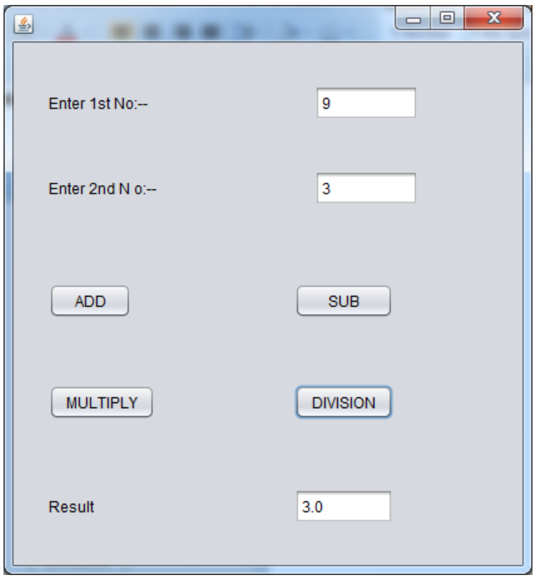
Result 6



Enter 1st No:-- 9

Enter 2nd N o:-- 3

Result 27



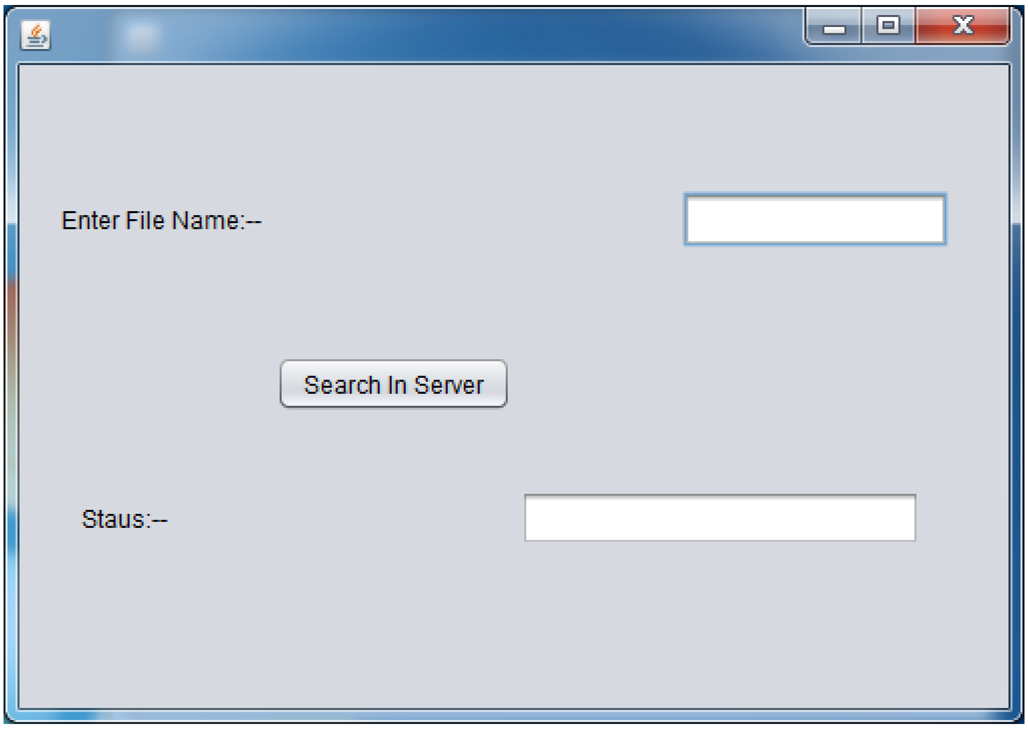
Enter 1st No:-- 9

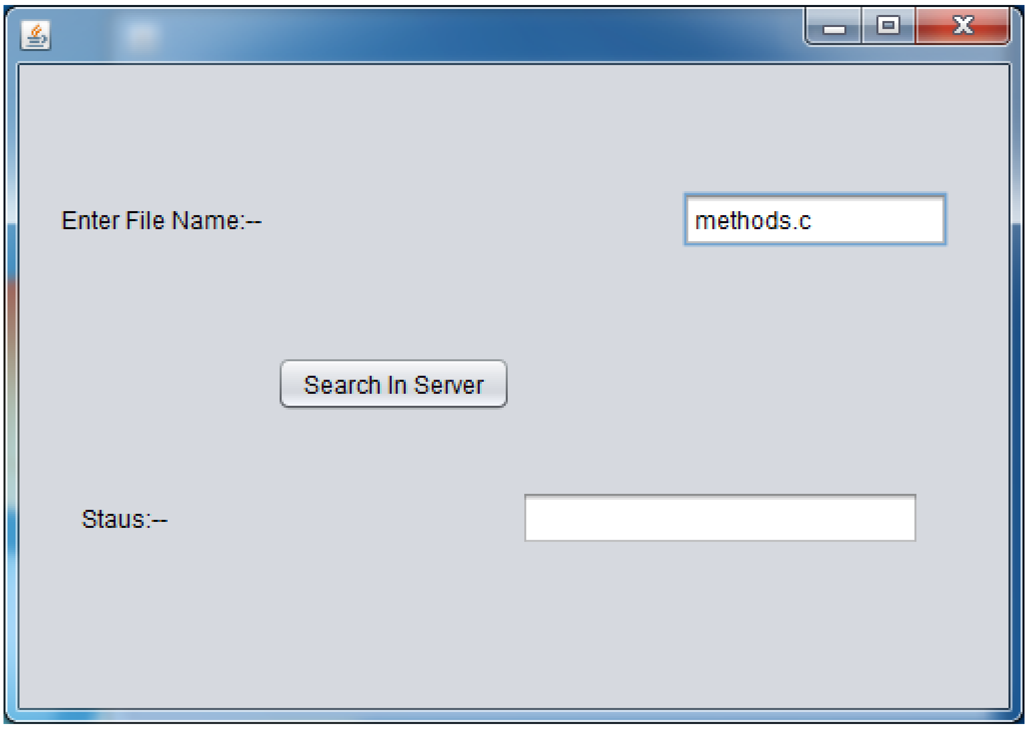
Enter 2nd N o:-- 3

Result 3.0

Taking two number 9 and 3, here we perform Addition, Subtruction, Multiplication and division operations.

• SEARCH FILE IN SERVER

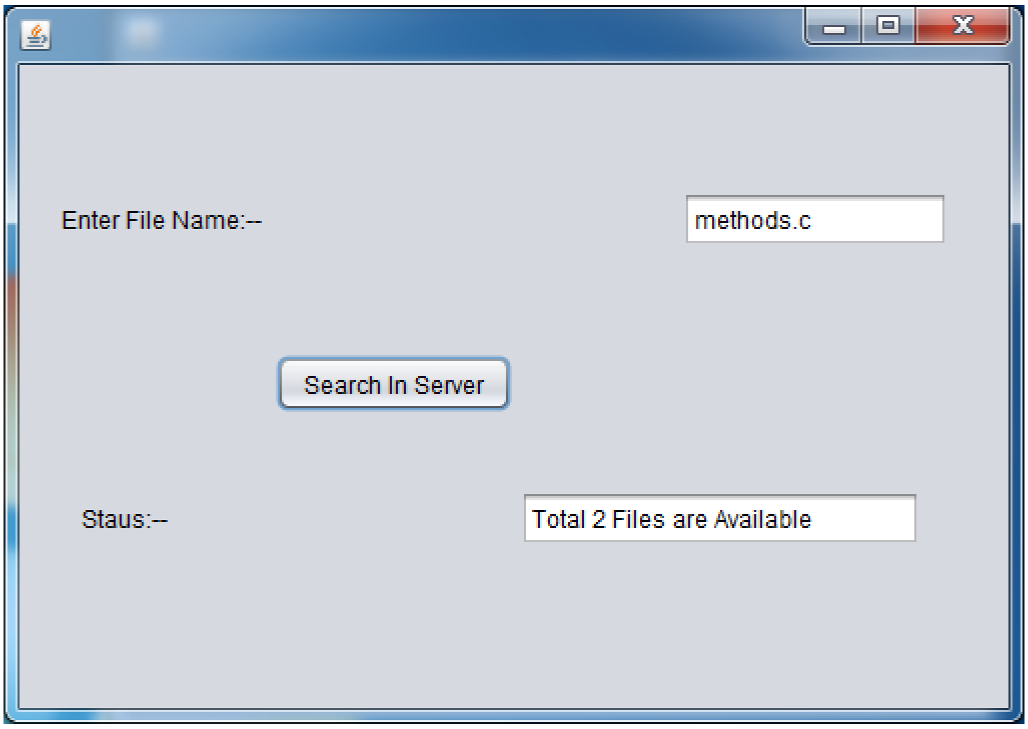




Enter File Name:-- methods.c

Staus:--

Here we search methods.c file in the server


Enter File Name:--

methods.c

Staus:--

Total 2 Files are Available

Return status from the server.

8. FUTURE WORK

9. CODES