RMI (Remote Method Invocation)

1. [Remote Method Invocation (RMI)](http://www.javatpoint.com/RMI)
2. [Understanding stub and skeleton](http://www.javatpoint.com/RMI#rmistubandskeleton)
   1. [stub](http://www.javatpoint.com/RMI#stub)
   2. [skeleton](http://www.javatpoint.com/RMI#skeleton)
3. [Requirements for the distributed applications](http://www.javatpoint.com/RMI#reqdistributed)
4. [Steps to write the RMI program](http://www.javatpoint.com/RMI#rmisteps)
5. [RMI Example](http://www.javatpoint.com/RMI#rmiex)

The **RMI** (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 applications using two objects *stub* and *skeleton*.

Understanding 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. Let's understand the stub and skeleton objects:

stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on 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, and
5. It finally, returns the value to the caller.

skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, an stub protocol was introduced that eliminates the need for skeletons.

Understanding requirements for the distributed applications

If any application performs these tasks, it can be distributed application.

.

1. The application need to locate the remote method
2. It need to provide the communication with the remote objects, and
3. The application need to load the class definitions for the objects.

The RMI application have all these features, so it is called the distributed application.

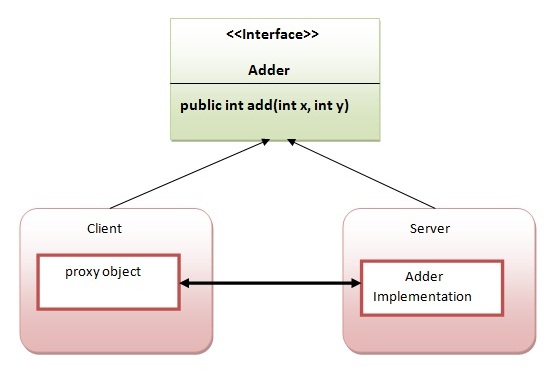
Steps to write the RMI program

The is given the 6 steps to write the RMI program.

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Create and start the remote application
6. Create and start the client application

RMI Example

In this example, we have followed all the 6 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 sent back to the proxy object and then to the client application.



1) create the remote interface

For creating the remote interface, extend the Remote interface and declare the RemoteException with all the methods of the remote interface. Here, we are creating a remote interface that extends the Remote interface. There is only one method named add() and it declares RemoteException.

1. **import** java.rmi.\*;
2. **public** **interface** Adder **extends** Remote{
3. **public** **int** add(**int** x,**int** y)**throws** RemoteException;
4. }

2) Provide the implementation of the remote interface

Now provide the implementation of the remote interface. For providing the implementation of the Remote interface, we need to

* Either extend the UnicastRemoteObject class,
* or use the exportObject() method of the UnicastRemoteObject class

In case, you extend the UnicastRemoteObject class, you must define a constructor that declares RemoteException.

1. **import** java.rmi.\*;
2. **import** java.rmi.server.\*;
3. **public** **class** AdderRemote **extends** UnicastRemoteObject **implements** Adder{
4. AdderRemote()**throws** RemoteException{
5. **super**();
6. }
7. **public** **int** add(**int** x,**int** y){**return** x+y;}
8. }

**3) create the stub and skeleton objects using the rmic tool.**

Next step is to create stub and skeleton objects using the rmi compiler. The rmic tool invokes the RMI compiler and creates stub and skeleton objects.

1. rmic AdderRemote

4) Start the registry service by the rmiregistry tool

Now start the registry service by using the rmiregistry tool. If you don't specify the port number, it uses a default port number. In this example, we are using the port number 5000.

1. rmiregistry 5000

5) Create and run the server application

Now rmi services need to be hosted in a server process. The Naming class provides methods to get and store the remote object. The Naming class provides 5 methods.

1. **public static java.rmi.Remote lookup(java.lang.String) throws java.rmi.NotBoundException, java.net.MalformedURLException, java.rmi.RemoteException;** it returns the reference of the remote object.
2. **public static void bind(java.lang.String, java.rmi.Remote) throws java.rmi.AlreadyBoundException, java.net.MalformedURLException, java.rmi.RemoteException;** it binds the remote object with the given name.
3. **public static void unbind(java.lang.String) throws java.rmi.RemoteException, java.rmi.NotBoundException, java.net.MalformedURLException;** it destroys the remote object which is bound with the given name.
4. **public static void rebind(java.lang.String, java.rmi.Remote) throws java.rmi.RemoteException, java.net.MalformedURLException;** it binds the remote object to the new name.
5. **public static java.lang.String[] list(java.lang.String) throws java.rmi.RemoteException, java.net.MalformedURLException;** it returns an array of the names of the remote objects bound in the registry.

In this example, we are binding the remote object by the name sonoo.

1. **import** java.rmi.\*;
2. **import** java.rmi.registry.\*;
3. **public** **class** MyServer{
4. **public** **static** **void** main(String args[]){
5. **try**{
6. Adder stub=**new** AdderRemote();
7. Naming.rebind("rmi://localhost:5000/sonoo",stub);
8. }**catch**(Exception e){System.out.println(e);}
9. }
10. }

6) Create and run the client application

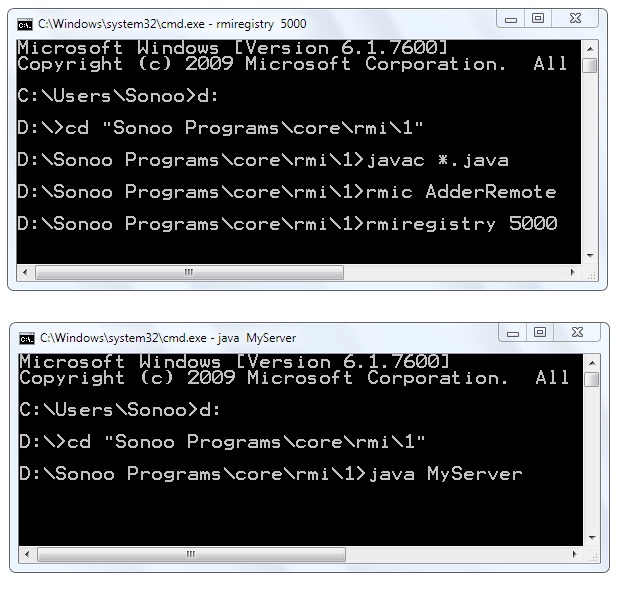
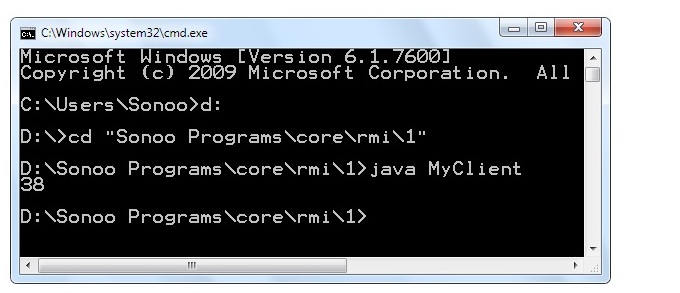
At the client we are getting the stub object by the lookup() method of the Naming class and invoking the method on this object. In this example, we are running the server and client applications, in the same machine so we are using localhost. If you want to access the remote object from another machine, change the localhost to the host name (or IP address) where the remote object is located.

1. **import** java.rmi.\*;
2. **public** **class** MyClient{
3. **public** **static** **void** main(String args[]){
4. **try**{
5. Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");
6. System.out.println(stub.add(34,4));
7. }**catch**(Exception e){}
8. }
9. }

[download this example of rmi](http://www.javatpoint.com/src/rmi/rmi1.zip)

1. For running **this** rmi example,
3. 1) compile all the java files
5. javac \*.java
7. 2)create stub and skeleton object by rmic tool
9. rmic AdderRemote
11. 3)start rmi registry in one command prompt
13. rmiregistry 5000
15. 4)start the server in another command prompt
17. java MyServer
19. 5)start the client application in another command prompt
21. java MyClient

Output of this RMI example

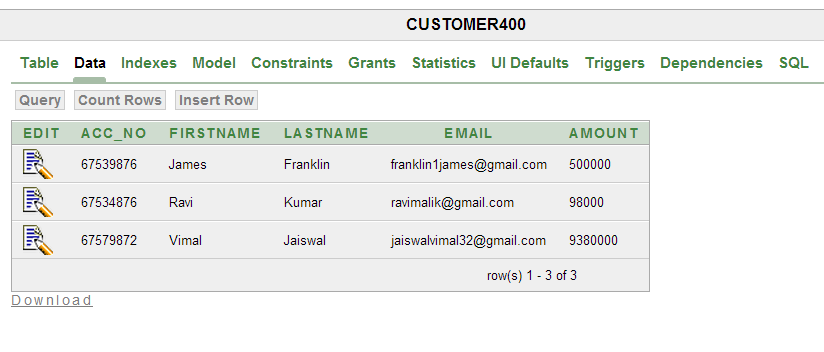
Meaningful example of RMI application with database

Consider a scenario, there are two applications running in different machines. Let's say MachineA and MachineB, machineA is located in United States and MachineB in India. MachineB want to get list of all the customers of MachineA application.

Let's develop the RMI application by following the steps.

**1) Create the table**

First of all, we need to create the table in the database. Here, we are using Oracle10 database.



**2) Create Customer class and Remote interface**

*File: Customer.java*

1. **package** com.javatpoint;
2. **public** **class** Customer **implements** java.io.Serializable{
3. **private** **int** acc\_no;
4. **private** String firstname,lastname,email;
5. **private** **float** amount;
6. //getters and setters
7. }

***Note: Customer class must be Serializable.***

*File: Bank.java*

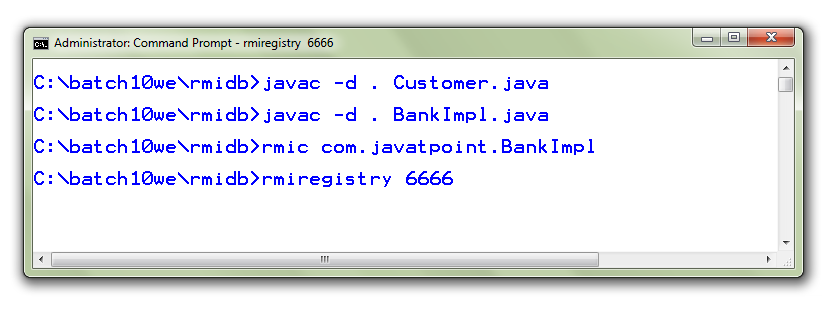
1. **package** com.javatpoint;
2. **import** java.rmi.\*;
3. **import** java.util.\*;
4. **interface** Bank **extends** Remote{
5. **public** List<Customer> getCustomers()**throws** RemoteException;
6. }

**3) Create the class that provides the implementation of Remote interface**

*File: BankImpl.java*

1. **package** com.javatpoint;
2. **import** java.rmi.\*;
3. **import** java.rmi.server.\*;
4. **import** java.sql.\*;
5. **import** java.util.\*;
6. **class** BankImpl **extends** UnicastRemoteObject **implements** Bank{
7. BankImpl()**throws** RemoteException{}
9. **public** List<Customer> getCustomers(){
10. List<Customer> list=**new** ArrayList<Customer>();
11. **try**{
12. Class.forName("oracle.jdbc.driver.OracleDriver");
13. Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
14. PreparedStatement ps=con.prepareStatement("select \* from customer400");
15. ResultSet rs=ps.executeQuery();
17. **while**(rs.next()){
18. Customer c=**new** Customer();
19. c.setAcc\_no(rs.getInt(1));
20. c.setFirstname(rs.getString(2));
21. c.setLastname(rs.getString(3));
22. c.setEmail(rs.getString(4));
23. c.setAmount(rs.getFloat(5));
24. list.add(c);
25. }
27. con.close();
28. }**catch**(Exception e){System.out.println(e);}
29. **return** list;
30. }//end of getCustomers()
31. }

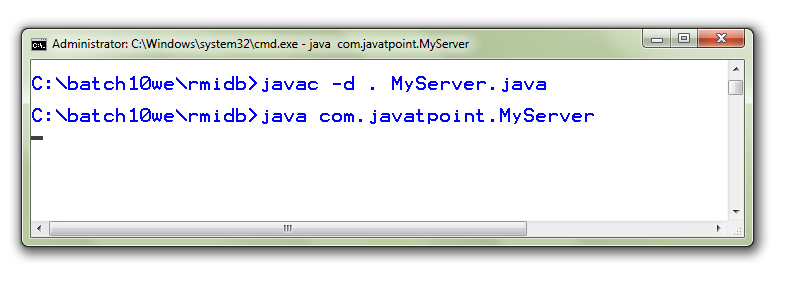
**4) Compile the class rmic tool and start the registry service by rmiregistry tool**



**5) Create and run the Server**

*File: MyServer.java*

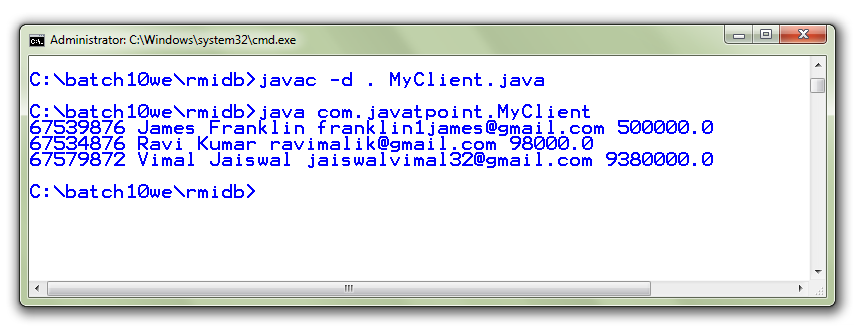
1. **package** com.javatpoint;
2. **import** java.rmi.\*;
3. **public** **class** MyServer{
4. **public** **static** **void** main(String args[])**throws** Exception{
5. Remote r=**new** BankImpl();
6. Naming.rebind("rmi://localhost:6666/javatpoint",r);
7. }}



**6) Create and run the Client**

*File: MyClient.java*

1. **package** com.javatpoint;
2. **import** java.util.\*;
3. **import** java.rmi.\*;
4. **public** **class** MyClient{
5. **public** **static** **void** main(String args[])**throws** Exception{
6. Bank b=(Bank)Naming.lookup("rmi://localhost:6666/javatpoint");
8. List<Customer> list=b.getCustomers();
9. **for**(Customer c:list){
10. System.out.println(c.getAcc\_no()+" "+c.getFirstname()+" "+c.getLastname()
11. +" "+c.getEmail()+" "+c.getAmount());
12. }
14. }}



[download this example of rmi with database](http://www.javatpoint.com/src/rmi/rmidb.zip)

[← prev](http://www.javatpoint.com/generics-in-java)[next →](http://www.javatpoint.com/internationalization)

package dustin.examples.rmi;

import java.rmi.ConnectException;

import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

/\*\*

\* Display names bound to RMI registry on provided host and port.

\*/

public class RmiPortNamesDisplay

{

private final static String NEW\_LINE = System.getProperty("line.separator");

/\*\*

\* Main executable function for printing out RMI registry names on provided

\* host and port.

\*

\* @param arguments Command-line arguments; Two expected: first is a String

\* representing a host name ('localhost' works) and the second is an

\* integer representing the port.

\*/

public static void main(final String[] arguments)

{

if (arguments.length < 2)

{

System.err.println(

"A host name (String) and a port (Integer) must be provided.");

System.err.println(

"\tExample: java dustin.examples.rmi.RmiPortNamesDisplay localhost 1099");

System.exit(-2);

}

final String host = arguments[0];

int port = 1099;

try

{

port = Integer.valueOf(arguments[1]);

}

catch (NumberFormatException numericFormatEx)

{

System.err.println(

"The provided port value [" + arguments[1] + "] is not an integer."

+ NEW\_LINE + numericFormatEx.toString());

}

try

{

final Registry registry = LocateRegistry.getRegistry(host, port);

final String[] boundNames = registry.list();

System.out.println(

"Names bound to RMI registry at host " + host + " and port " + port + ":");

for (final String name : boundNames)

{

System.out.println("\t" + name);

}

}

catch (ConnectException connectEx)

{

System.err.println(

"ConnectionException - Are you certain an RMI registry is available at port "

+ port + "?" + NEW\_LINE + connectEx.toString());

}

catch (RemoteException remoteEx)

{

System.err.println("RemoteException encountered: " + remoteEx.toString());

}

}

}