

Advanced Programming

Week 10

Distributed Programming

- RMI
- RMI architecture
- Serialization
- Steps to develop RMI application



Tilahun Melak(PhD)

October, 2025

Objectives

At the end of this lecture, students will be able to:

- Explain RMI (Remote Method Invocation)
- Describe the RMI architecture
- Explain serialization
- Identify and apply the steps to develop an RMI application

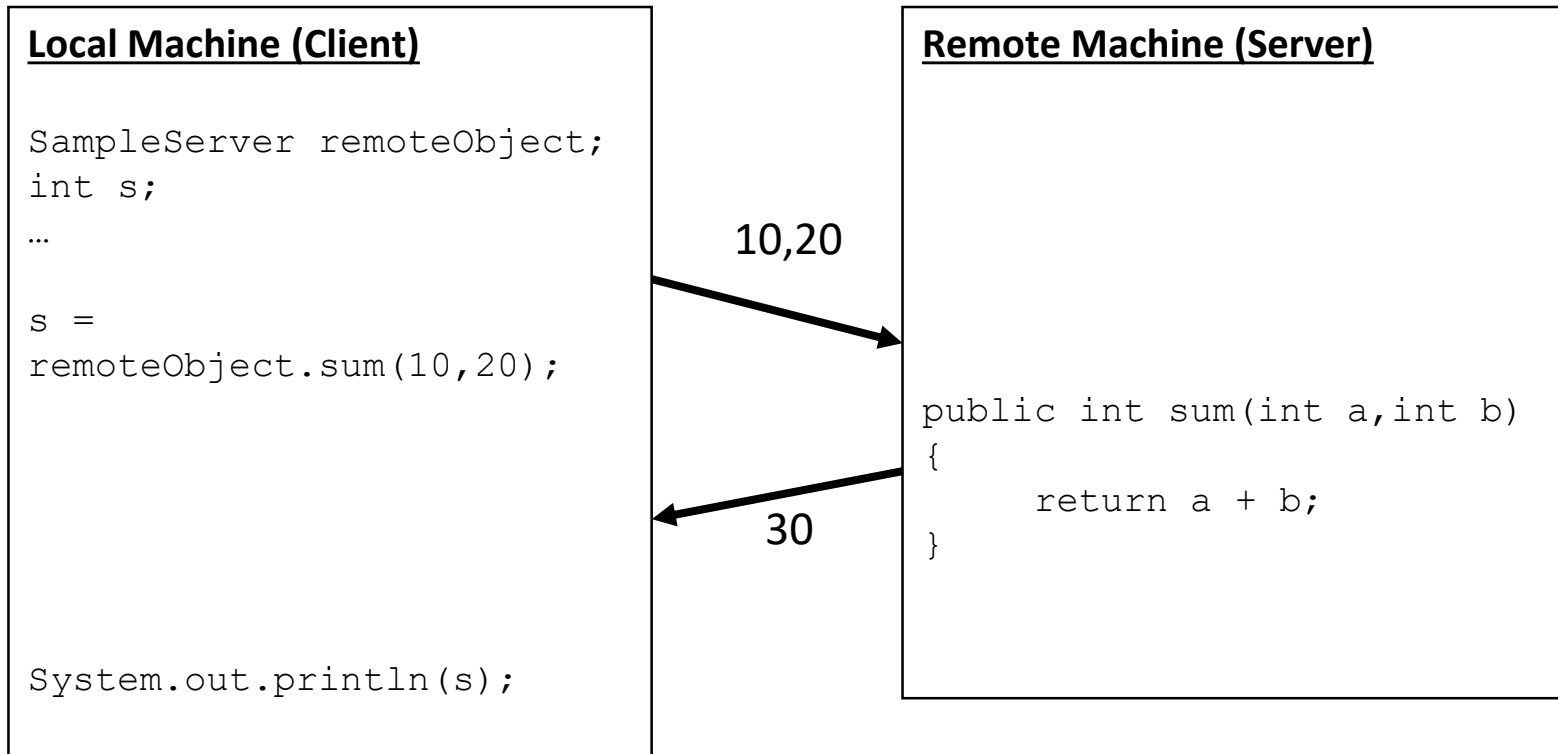
RMI - Remote Method Invocation

- Java RMI is a mechanism to allow the invocation of methods that reside on different Java Virtual Machines (JVMs).
- The JVMs may be on different machines or they could be on the same machine.
 - In either case, the method runs in a different address space than the calling process.
- Java RMI is an object-oriented remote procedure call mechanism.

RMI - Remote Method Invocation...

- RMI is a distributed object system that enables you to easily develop distributed Java applications.
- Developing distributed applications in RMI is simpler than developing with sockets since there is no need to design a protocol, which is an error-prone task
- Java RMI allowed programmer to execute remote function class using the same semantics as local function calls.

RMI - Remote Method Invocation...



Entities involved in RMI Application

1. A **remote object** is an object on another computer
2. The **server object** is the object receiving the request
3. **Client:** program written to access remote methods
4. **Server:** program written to implement the remote methods
 - Clients connect to the server and request that a method be executed.

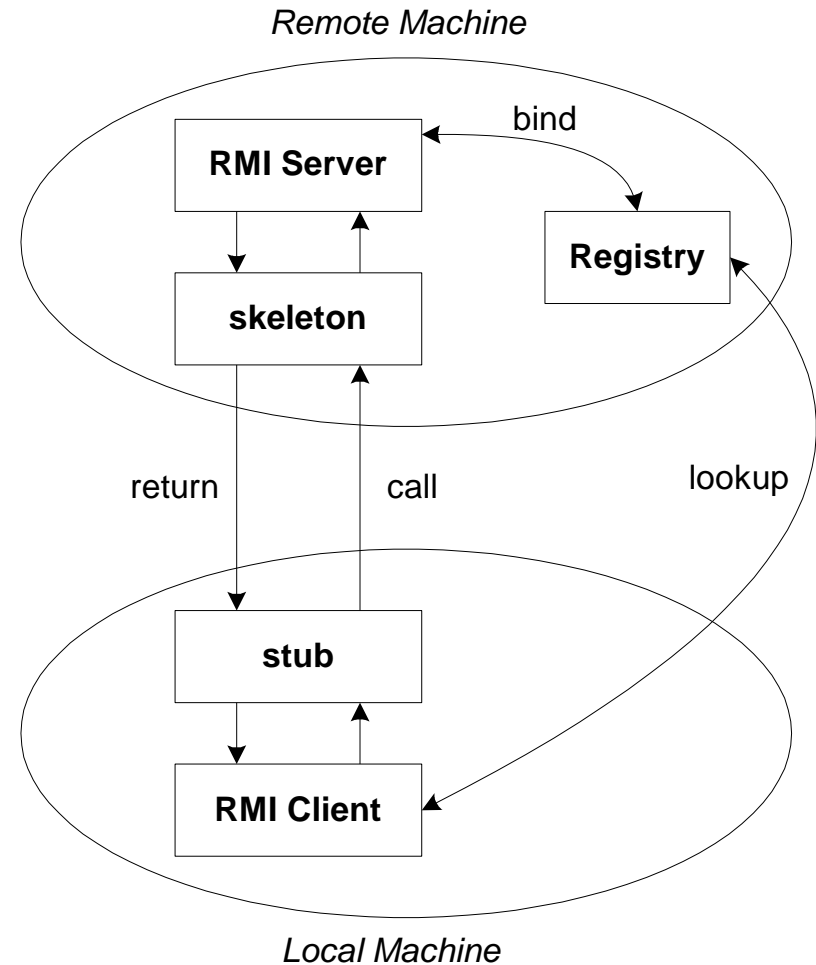
Entities involved in RMI Application...

5. Object registry: is a program

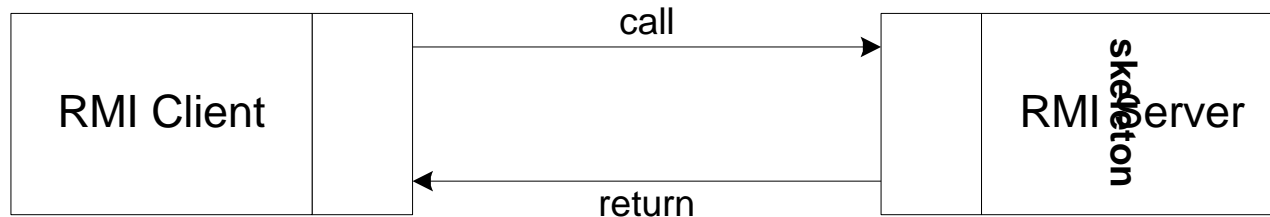
- runs on a known port (1099 by default)
- A server, upon starting, registers its objects with a textual name in the object registry.
- A client, before invoking a remote method, must first contact the object registry to obtain access to the remote object.

General RMI Architecture

- The server must first bind its name to the registry
- The client lookup the server name in the registry to establish remote references.
- The Stub serializing the parameters to skeleton, the skeleton invoking the remote method and serializing the result back to the stub.

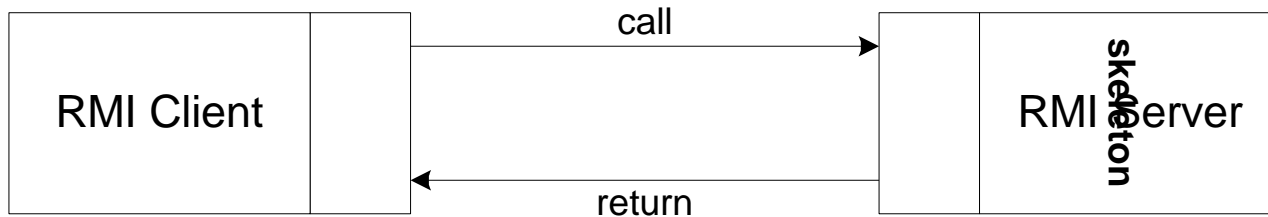


The Stub and Skeleton



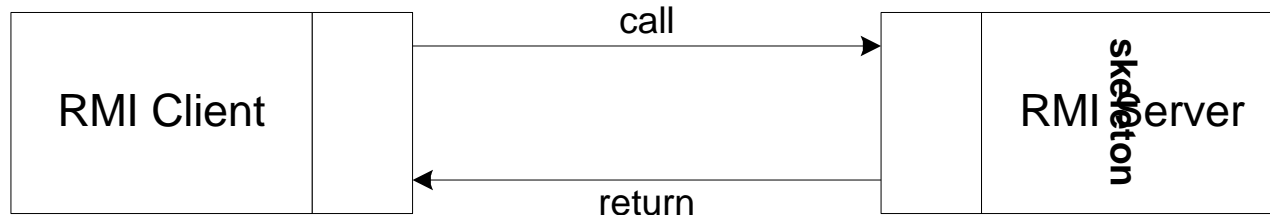
- A client invokes a remote method, the call is first forwarded to stub.
- The stub is responsible for sending the remote call over to the server-side skeleton
- The stub opening a socket to the remote server, marshaling the object parameters and forwarding the data stream to the skeleton.

The Stub and Skeleton...



- A skeleton contains a method that receives the remote calls, unmarshals the parameters, and invokes the actual remote object implementation.

The Stub and Skeleton...



- RMI uses only TCP, UDP is not supported by RMI.
- **Marshalling and UnMarshalling:** refers to the process of converting the data or object being transferred into a byte stream and unmarshalling is the reverse- converting the stream into an object or data; conversion is achieved via object serialization.

Serialization

- Serialization is the process of writing the state of an object to a byte stream.
- This is useful when we want to save the state of our object to a persistent storage such as a file.
- At a later time, we may restore these objects by using the process of de-serialization.

Serialization...

- The class whose object we want to write to a file, must implement the `Serializable` interface.
- `Serializable` interface is a marker interface means it does not have any method, it only informs the tools that are involved in serialization that the class is ready for serialization.
- For eg.

```
class Test implements Serializable{  
  
}
```

Oracle. *Java Object Serialization*. Java SE Documentation.

RMI – Merits and Demerits

- Merits
 - Ease of programming
 - No complicated Interface Definition Language (IDL) to learn
- Demerits
 - Uses Proprietary Protocol
 - Java Remote Method Protocol (JRMP)
 - No cross-language support

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.
6. Start the RMI registry.
7. Start the remote server objects.
8. Run the client

Step 1: Defining the Remote Interface

- To create an RMI application, the first step is the defining of a remote interface between the client and server objects.
- The interface definition:
 - must be public
 - must extend the interface *java.rmi.Remote*
 - every method in the interface must declare that it throws *java.rmi.RemoteException*
 - but other exceptions may be thrown as well

Step 1: Defining the Remote Interface...

```
/* SampleServer.java */  
import java.rmi.*;  
  
public interface SampleServer extends Remote  
{  
    public int sum(int a,int b) throws RemoteException;  
}
```

Develop the remote object and its interface

- The server must implement the Remote interface
- It should extend **java.rmi.server.UnicastRemoteObject** class
 - This will allow the methods to be invoked remotely
- Each method must throw **RemoteException**
 - Because remote procedure calls might fail
- The server uses the **RMISecurityManager** to protect its resources while engaging in remote communication.

Develop the remote object and its interface...

```
/* SampleServerImpl.java */
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;

public class SampleServerImpl extends
    UnicastRemoteObject
                                implements SampleServer
{
    SampleServerImpl() throws RemoteException
    {
        super();
    }
    // Implement the remote methods
    public int sum(int a,int b) throws RemoteException
    {
        return a + b;
    }
}
```

Oracle. (n.d.). *Java Remote Method Invocation (RMI)*

Develop the remote object and its interface...

- The server must **bind** its name to the registry, the client will **look up** the server name.
- Use **java.rmi.Naming** class to bind the server name to registry.

In this example, lets use the name “SAMPLE-SERVER”.

- In the main method of your server object, the RMI security manager is created and installed.

Develop the remote object and its interface...

```
/* SampleServerImpl.java */
public static void main(String args[])
{ try
  {
    //set the security manager
    System.setSecurityManager(new RMISecurityManager());
    //create a local instance of the object
    SampleServerImpl Server = new SampleServerImpl();
    //put the local instance in the registry
    Naming.rebind("SAMPLE-SERVER" , Server);
    System.out.println("Server waiting.....");
  }
  catch (RemoteException re)
  { System.out.println("Remote exception: " + re.toString()); }
}
```

Oracle. (n.d.). *Java Remote Method Invocation (RMI)*

Step 3: Develop the client program...

- In order for the client object to invoke methods on the server, it must first look up the name of server in the registry. Use the **java.rmi.Naming** class to lookup the server name.
- The server name is specified as URL in the form **(rmi://host:port/name)**
 - Default RMI port is 1099.

Step 3: Develop the client program...

- The name specified in the URL must exactly match the name that the server has bound to the registry.
 - In this example, the name is “SAMPLE-SERVER”
- The remote method invocation is programmed using the remote interface name (`remoteObject`) as prefix and the remote method name (`sum`) as suffix.

Step 3: Develop the client program...

```
import java.rmi.*;
import java.rmi.server.*;
public class SampleClient
{ public static void main(String[] args)
  { // set the security manager for the client
    System.setSecurityManager(new RMISecurityManager());
    try
    { //get the remote object from the registry
      String url = "rmi://localhost/SAMPLE-SERVER";
      SampleServer remoteObject = (SampleServer)Naming.lookup(url);
      System.out.println("Got remote object");
      System.out.println(" 1 + 2 = " + remoteObject.sum(10,20) );
    }
    catch (RemoteException exc) {
      System.out.println("Error in lookup: " + exc.toString());
    }
    catch (java.net.MalformedURLException exc) {
      System.out.println("Malformed URL: " + exc.toString());
    }
    catch (java.rmi.NotBoundException exc) {
      System.out.println("NotBound: " + exc.toString());
    }
  }
}
```

Oracle. (n.d.). *Java Remote Method Invocation (RMI)*

Step 4 & 5: Compile the Java source files & Generate the client stubs and server skeletons

- Compile the java source files in command prompt

```
> javac SampleServer.java  
> javac SampleServerImpl.java  
> javac SampleClient.java
```

- Generate stubs and skeleton code. The RMI system provides an RMI compiler (`rmic`) that takes the implemented interface class and produces stub code on itself.

```
> rmic SampleServerImpl
```

Step 6: Start the RMI registry

- The `rmiregistry` uses port 1099 by default. You can also bind `rmiregistry` to a different port by indicating the new port number as `:rmiregistry <new port>`
- *Type the following in the command line:*

```
> start rmiregistry
```

Steps 7 & 8: Start remote server objects & Run client

- Once the Registry is started, the server can be started and will be able to store itself in the Registry.
- In the command line, to start the remote server:

```
> java SampleServerImpl
```

- NOTE: The registry and server processes are running in the background. Be sure to kill the registry and server processes when you're done.

Steps 7 & 8: Start remote server objects & Run client...

- Finally, to run the client:

```
> java SampleClient 10 20
```

- The Final Output becomes:

```
Got remote object  
10 + 20 = 30
```

Java Policy File

- Because of the grained security model in Java 2.0, you must setup a security policy for RMI by updating the `java.policy` file.
- In Java 2, the java application must first obtain information regarding its privileges.
- It can obtain the security policy through a policy file.
- In above example, we allow Java code to have all permissions, the contents of the `java.policy` file are:

```
grant {  
    permission java.security.AllPermission;  
};
```

Distributing the Application

- The easiest way to try a distributed architecture is *to copy all required files manually* to correct machines/directories:
- Copy **SampleClient.class** (client), **SampleServerImpl_Stub.class** (stub), **SampleServer.class** (interface) to a directory on the **client** machine .
- Copy **SampleServer.class** (interface), **SampleServerImpl.class** (its implementation), and **SampleServerImpl_Stub.class** (stub) to a directory on the **server** machine.
- *** If your implementation has a database connection, you can consider putting the database in a third machine=> 3 Tier

Summary

- In today's lecture we have discussed about;
 - Introduces Remote Method Invocation (RMI) in Java
 - Explaining its architecture and the concept of serialization.
 - outlines the steps involved in developing an RMI application.
 - Examples.

References

- Oracle. (n.d.). Getting started with RMI. Oracle Java Documentation.
- Java SE Documentation. *Java Platform, Standard Edition: Remote Method Invocation (RMI) Overview*. Oracle.
- Oracle. *Java Object Serialization*. Java SE Documentation.
- Sharma, R. *Java Remote Method Invocation (RMI) – Advantages and Limitations*
- Oracle. (n.d.). *Java Remote Method Invocation (RMI)*
- Oracle. (n.d.). *Java Platform, Standard Edition Security Architecture*. Java SE Documentation.
- Oracle. (n.d.). *Java Remote Method Invocation (RMI) – Getting Started*. Java SE Documentation.