**CL-IX**

Assignment 1 (B)

**Roll No.:** 43141

**Name:** Sahil Naphade

**Batch:** P-9

**Problem Statement:** To develop distributed application (Hashing server) through implementing client-server communication programs based on **Java RMI**.

**Objectives:** By the end of this assignment, the student will be able to implement any distributed multi-threaded client-server programs using Java RMI.

**Tools:** Java Programming Environment, JDK 1.8+, rmiregistry, Eclipse.

**Theory:**

* Remote Method Invocation (RMI) is an API which allows an object to invoke a method of an object that exists in another address space, which could be on the same machine or on a remote machine.
* Through RMI, object running in a JVM present on a computer (Client side) can invoke methods on an object present in another JVM (Server side).
* RMI creates a public remote server object that enables client and server-side communications through simple method calls on the server object.

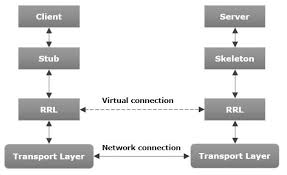


Figure 1. Introduction to Java RMI

* The communication between client and server is handled by two intermediate objects
  + Stub object (on client side)
  + Skeleton object (on server side)

**Stub object:**

The stub object on the client machine builds an information block and sends this information to the server.

The block consists of:

1. An identifier of the remote object to be used
2. Method name which is to be invoked
3. Parameters to the remote JVM.

**Skeleton Object:**

The skeleton object passes the request from the stub object to the remote object. It performs following tasks:

1. It calls the desired method on the real object present on the server.
2. It forwards the parameters received from the stub object to the method.

**Working of RMI:**

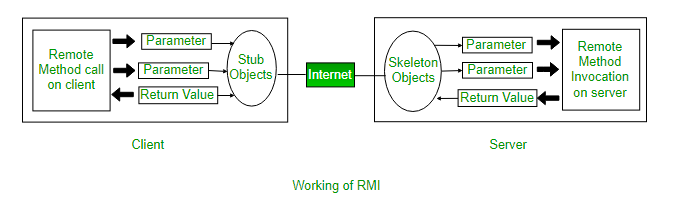


Figure 2: Working of RMI

1. We run the server code on server machine and the client on the client machine.
2. Using RMI Compiler, Skeleton object is created of the server and stubs of the client are created.
3. Whenever the client calls a method in server class (Remote method Invocation), the client sends the stub object over the internet to the skeleton of the server, which in turn executes the request.

**Method of building an RMI application:**

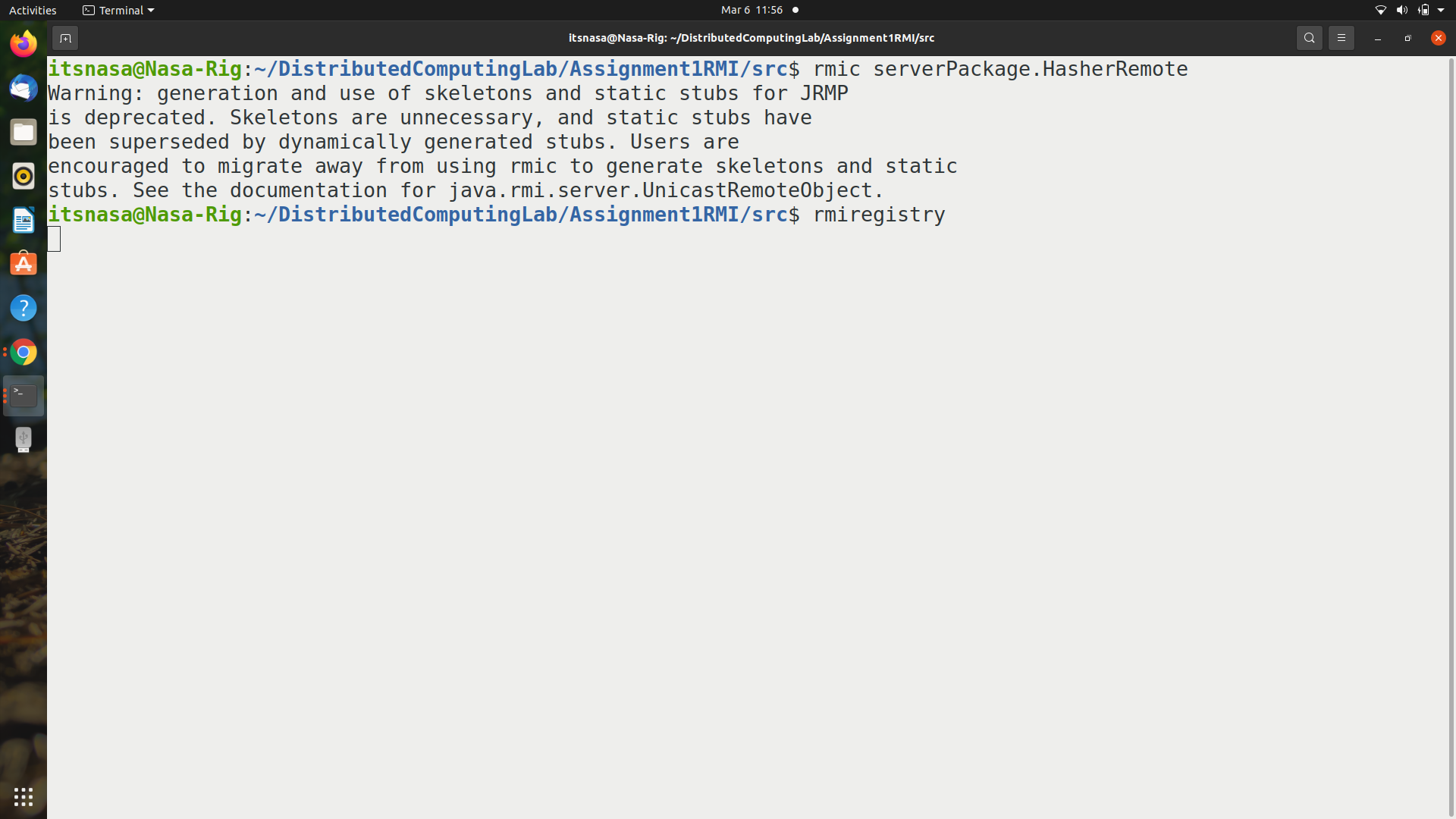
1. Java.rmi.Remote interface:
   1. In RMI, a remote interface is an interface that declares a set of methods that may be invoked from a remote Java virtual machine.
   2. I have declared the interface with a hashing function, which accepts the string input and returns the hash of the string as per the specified algorithm.
2. The RemoteObject class and its subclasses:
   1. RMI server functions are provided by java.rmi.server.RemoteObject and its subclasses, java.rmi.server.RemoteServer and java.rmi.server.UnicastRemoteObject.
   2. The class java.rmi.server.RemoteObject provides implementations for the java.lang.Object methods that are sensible for remote objects.
   3. The methods needed to create remote objects and make them available to remote clients are provided by the class UnicastRemoteObject.
   4. The java.rmi.server.UnicastRemoteObject class defines a singleton (unicast) remote object whose references are valid only while the server process is alive.
   5. I have implemented the hash function in this class.
3. Locating the remote objects:
   1. A simple name server is provided for storing the named references to the remote objects.
   2. A remote object reference can be stored using the URL-based methods of the class java.rmi.Naming.
   3. For a client to invoke a method on a remote object, that client must first obtain a reference to the object.
   4. The java.rmi.Naming class provides Uniform Resource Locator (URL) based methods to look up, bind, rebind, unbind, and list the name-object pairings maintained on a particular host and port.
4. Additional theory:
   1. Stub hides the serialization of parameters and the network-level communication in order to present a simple invocation mechanism to the caller. In the remote JVM, each remote object may have a corresponding skeleton.
   2. The skeleton is responsible for dispatching the call to the actual remote object implementation.

**Steps to implement RMI application:**

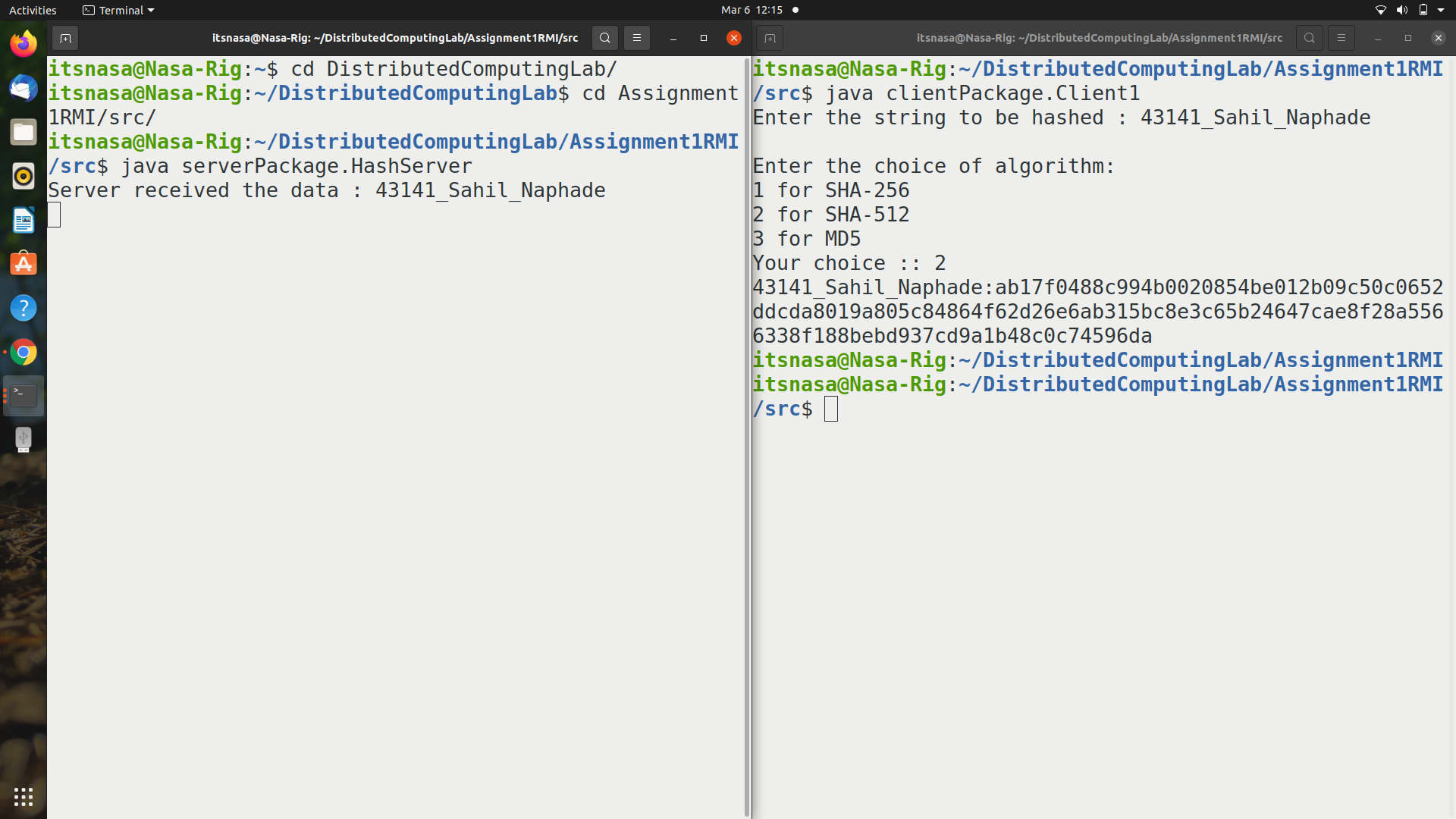
1. Define a remote interface (which extends java.rmi.Remote)
2. Implement the defined remote interface
3. Create stub and skeleton objects from the implementation class using rmic (rmi compiler)
4. Start the rmiregistry.
5. Create and execute the server program
6. Create and execute client application program

**Output:**

1. Calling RMIC to compile and create the stub
2. Start the RMIregistry



1. Run the server class
2. Run the client class and pass the string



**Conclusion:**

Thus, I learnt how to develop a client/server distributed application using JAVA RMI and created a hashing server using the same.