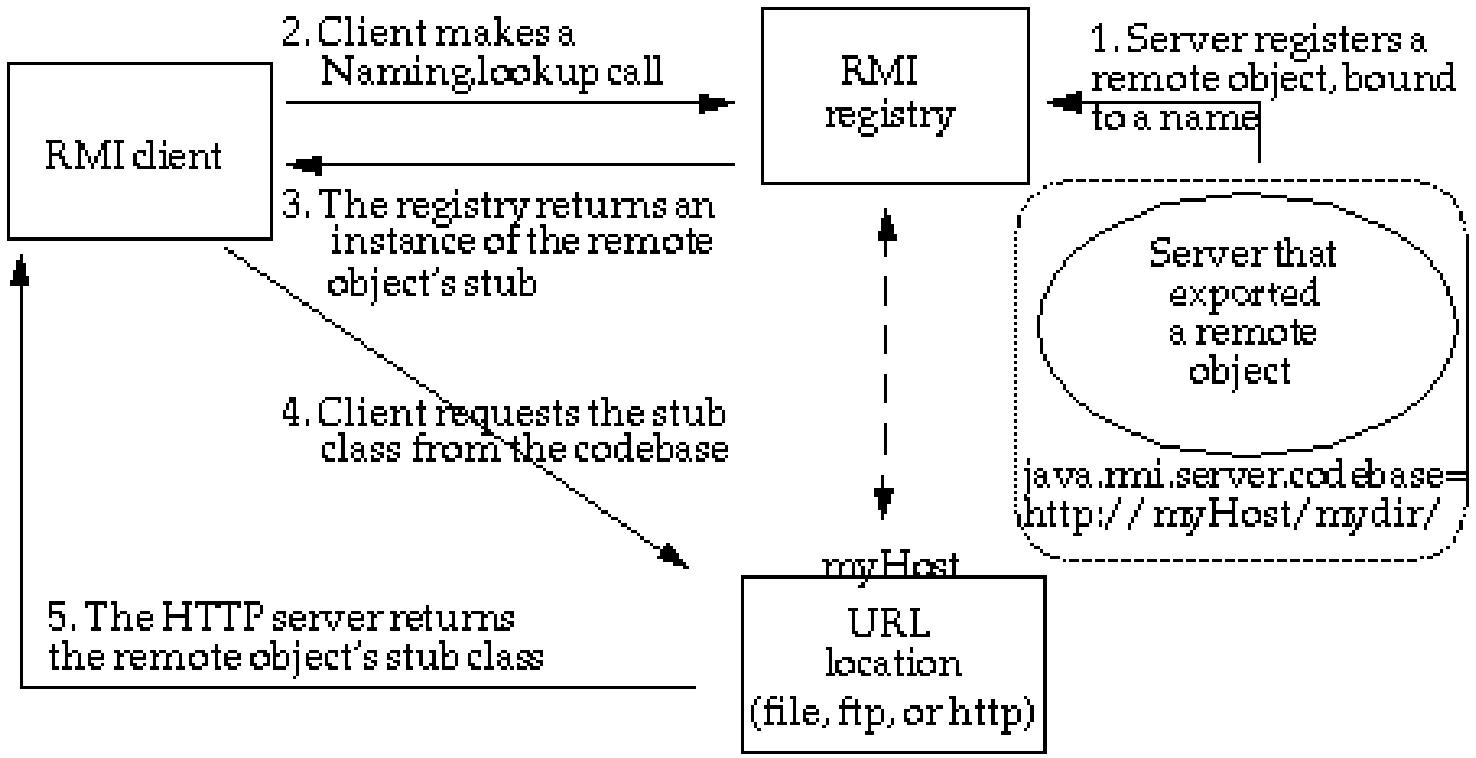
Java RMI
(Remote Method Invocation)

Dr. Abdul Haseeb Malik

Introduction

- RMI is a Java-implementation of RPC (Remote Procedure Call)
- RMI server
  - creates remote objects
  - makes references to those objects accessible by publishing them using an RMI registry.
  - waits for clients to invoke methods on those objects
- RMI client
  - can locate remote objects from RMI registry and obtain the remote reference to remote objects on a server
  - invokes methods
- Remote communication looks like regular Java method invocations to the programmer and details of remote communication between server and client are handled by RMI


2. Client makes a Naming.lookup call
RMI registry
1. Server registers a remote object, bound to a name
RMI client
3. The registry returns an instance of the remote object's stub
Server that exported a remote object
java.rmi.server.codebase= http://myHost/mydir/
4. Client requests the stub class from the codebase
myHost
5. The HTTP server returns the remote object's stub class
URL location (file, ftp, or http)

Simple Example

- In the following slides a simple example of Java RMI is presented in which:
  - Only a single add method will be provided on the Server
  - The client will locate the object remotely and call this method to add two numbers.

Simple Example: Remote Interface

```
public interface RemMethodInt extends java.rmi.Remote {
        public int add(int a, int b) throws RemoteException;
}
```

- This interface needs to be present at both sides (server and client).
- It tells both sides which methods can be remotely accessible. Here it is only one method which is named *add.*
- The java.rmi.Remote interface is a marker interface which is empty and is only used to let the JVM know that these methods will be used for remote communication.

Simple Example: RMI Server

```
public class RemImpl extends UnicastRemoteObject implements RemMethodInt{
//Constructor should be explicitly called.
//It creates and exports a remote object.
        protected RemImpl() throws RemoteException {super();}
// Implementation of the add method defined in the interface.
        public int add(int a, int b) throws RemoteException {return((a+b));          }

        public static void main(String[] args) {
try {   RemImpl obj = new RemImpl();
        Registry reg = LocateRegistry.createRegistry(5556);
        reg.bind("adder", obj);}
         catch (RemoteException e) {e.printStackTrace();}
         catch (AlreadyBoundException e) {        e.printStackTrace();}
        }
}
```

Simple Example: RMI Client

```
public class clientImp {
	public static void main(String[] args) {
try {
	Registry reg = LocateRegistry.getRegistry("192.168.0.1",5556);
	RemMethodInt obj = (RemMethodInt)reg.lookup("adder");
	System.out.println(obj.add(7, 5));
}
catch (RemoteException | NotBoundException e) {e.printStackTrace();}
	}

}
```

Important Notes for RMI Server in Example

- A class “RemImpl” extends the java.rmi.server.UnicastRemoteObject to provide support for creating and exporting remote objects.
- It also implements the “RemMethodInt” interface and implements the methods defined in the interface.
- At the server side an object of “RemImpl” is created.
- A Registry is created using

```
Registry reg = LocateRegistry.createRegistry(5556);
```

Note: In this case no need to run the registry explicitly

- The object reference will be made available at the following URL address “adder”

```
reg.bind("adder", obj);
```

Important Notes for RMI Client in Example

- We locate the registry which contains references of remote objects on the server system.

Registry reg = LocateRegistry.getRegistry("192.168.0.1",5556);

- Within the registry the client can lookup for the adder object and return the reference of the object.

RemMethodInt obj = (RemMethodInt)reg.lookup("adder");

- Once the client has reference of remote object, it can call methods on it as if it was a local object.

System.out.println(obj.add(7, 5));