Chapter 2
Distributed Information Systems

Presented by: Dr. R. BENTRCIA

Department of Computer Science, M'sila University

Outline

- RMI Application Example:
  - The Remote Interface
  - The Server Application
  - The Client Application
- Running the RMI Application

RMI Application Example

- In the distributed Hello World example, a client makes a remote method call to the server, to retrieve the message "Hello world!".
- We have followed all the 6 steps to create and run this RMI application.
- The application consists of:
  - The remote interface
  - The server application
  - The client application
- Both client and server interact with the remote interface.

The Remote Interface

- The remote interface declares each of the methods that you want to call remotely.
- Remote interface has the following characteristics:
  - It extends the java.rmi.Remote interface.
  - It declares the RemoteException.

The Remote Interface

Hello.java | HelloImpl.java | HelloClient.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;


//The Remote Interface

public interface Hello extends Remote //Extend the Remote interface
{
    String sayHello() throws RemoteException; //Declare the RemoteException
}
```

The Server Application

- The server application provides the implementation of the remote interface.
- A "server" class is the class which has a *main* method that:
  - Creates an instance of the remote object implementation (constructor):
    - The constructor exports the remote object: Once created, the remote object is ready to accept incoming remote method invocations by listening for incoming calls.
  - Binds that instance to a name in the rmiregistry.
    - The RMI registry is a simple server-side name server that allows remote clients to get a reference to a remote object.
- The class must extend the UnicastRemoteObject class:
  - When you extend java.rmi.server.UnicastRemoteObject, your class is automatically exported upon creation.

The Server Application

Hello.java | **HelloImpl.java** | HelloClient.java

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;


//The Server Application

public class HelloImpl extends UnicastRemoteObject implements Hello
{
    public HelloImpl() throws RemoteException {} // Define a constructor that declares RemoteException

    public String sayHello() { return "Hello world!"; } //Implement the remote method sayHello

    public static void main(String args[])
    {
        try
        {
            HelloImpl obj = new HelloImpl(); //Create an instance of the remote object
            Naming.rebind("//localhost/MyServer", obj); // Bind this object instance to the name "MyServer" in the rmiregistry
        }
        catch (Exception e)
        {
            System.out.println("HelloImpl err: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

The Client Application

- To “find” the server, the client uses an Interface object that “looks” for a reference for the remote object associated with the name we pass as parameter.
- It gets a reference to the remote object implementation from the server host's *rmiregistry.*
- It invokes the remote method on the server's remote object.

The Client Application

Hello.java | HelloImpl.java | **HelloClient.java** ×

```
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;


//The Client Application

public class HelloClient
{
    public static void main(String arg[])
    {
        String message = "blank";

      try
        {
           Hello obj = (Hello) Naming.lookup("//localhost/MyServer"); //It returns the reference of the remote object obj in registry
           System.out.println(obj.sayHello()); //Invoking the remote method on this object
        }
        catch (Exception e)
        {
           System.out.println("HelloClient exception: " + e.getMessage());
           e.printStackTrace();
        }
    }
}
```

Running the RMI Application

- **Write** the Java sources.
- **Compile** and deploy class files.
- **Start** the RMI registry, server, and client as shown in the output.

Running the RMI Application