Remote Method Invocation (RMI)

Instructor

Mehrnaz Zhian

mehrnaz.zhian@senecacollege.ca

Introduction

- Remote Method Invocation (RMI) provides a framework for building distributed Java systems.
- Using RMI, a Java object on one system can invoke a method in an object on another system on the network.
- A **distributed Java system** can be defined as a collection of cooperative distributed objects on the network.
- Today, you will learn how to use RMI to create useful distributed applications.

RMI – Basics

- RMI is the **Java Distributed Object Model** for facilitating communications among distributed objects.
- RMI is a higher-level API built on top of sockets.
- Socket-level programming allows you to pass data through sockets among computers.
- RMI enables you not only to pass data among objects on different systems, but also to invoke methods in a remote object.

The Differences between RMI and Traditional Client/Server Approach

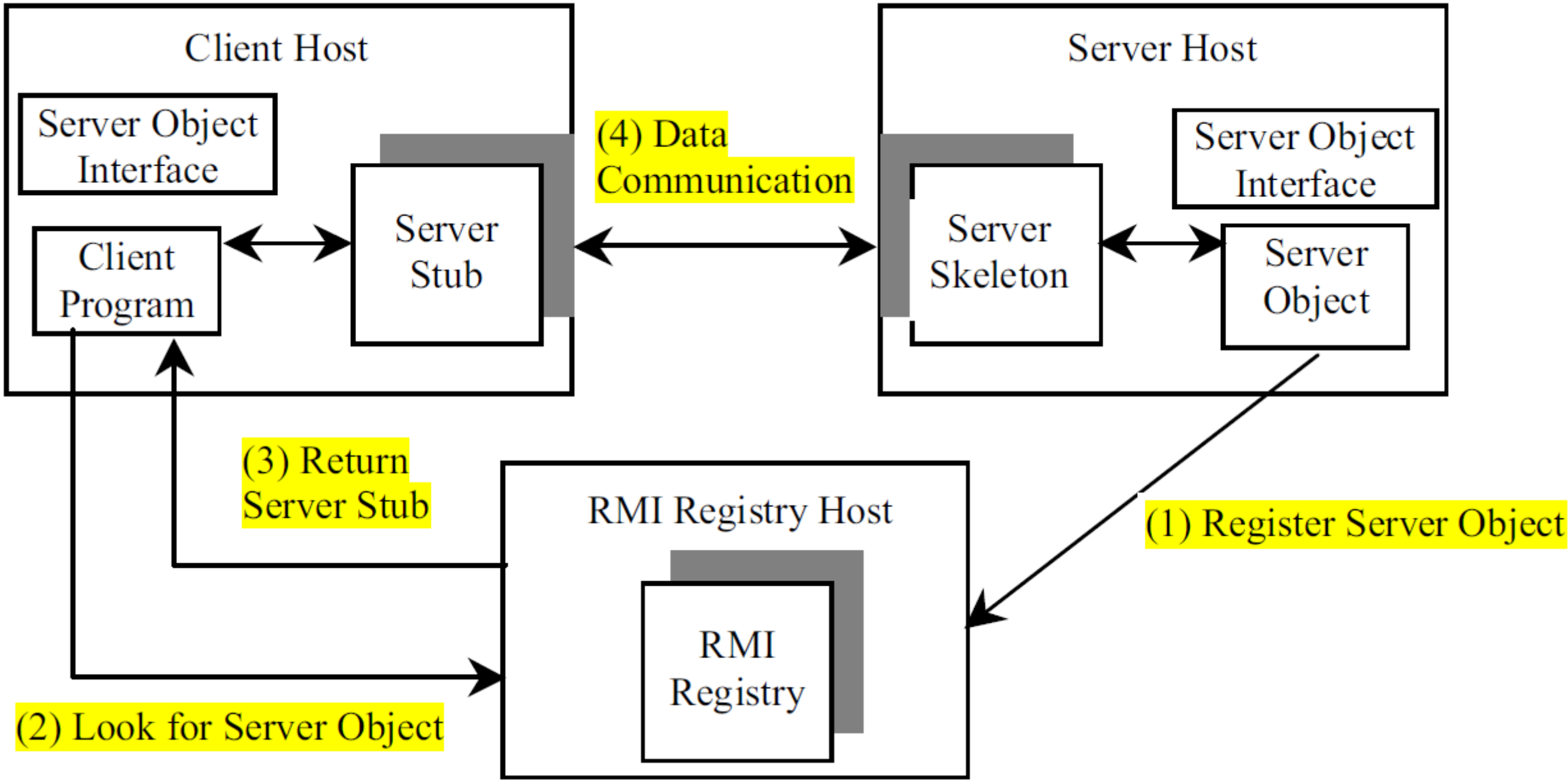
- RMI is an evolution of the client/server architecture.
- A **client** is a component that issues requests for services, and a **server** is a component that delivers the requested services.
- An RMI component can act as both a client and a server, depending on the scenario in question.
- An RMI system can pass functionality from a server to a client and vice versa.
- A client/server system typically only passes data back and forth between server and client.

How Does RMI Work?

- All the objects we have used so far are called **local objects**.
- Local objects are accessible only within the local host.
- Objects that are accessible from a remote host are called **remote objects**.
- For an object to be invoked remotely, it must be defined in a Java interface accessible to both the server and the client.
- Furthermore, the interface must extend the java.rmi.Remote interface.
- java.rmi.Remote is an interface that contains no constants or methods and it is used only to identify remote objects.

Key Components of RMI Architecture

- **Server object interface**: A sub-interface of java.rmi.Remote that defines the methods for the server object.
- **Server class**: A class that implements the remote object interface.
- **Server object**: An instance of the server class.
- **RMI registry**: A utility that registers remote objects and provides naming services for locating objects.
- **Client program**: A program that invokes the methods in the remote server object.
- **Server stub**: An object that resides on the client host and serves as a surrogate for the remote server object.
- **Server skeleton**: An object that resides on the server host and communicates with the stub and the actual server object.


Client Host
Server Object Interface
Client Program
Server Stub
(4) Data Communication
Server Host
Server Object Interface
Server Skeleton
Server Object
(3) Return Server Stub
RMI Registry Host
RMI Registry
(1) Register Server Object
(2) Look for Server Object

How Does RMI Work? (Cont.)

- Java RMI uses a registry to provide naming services for remote objects, and uses the stub and the skeleton to facilitate communications between client and server.
- RMI works as follows:
  1) A server object is registered with the RMI registry.
  2) A client looks through the RMI registry for the remote object.
  3) Once the remote object is located, its stub is returned in the client.
  4) The remote object can be used in the same way as a local object. Communication between the client and the server is handled through the stub and the skeleton.

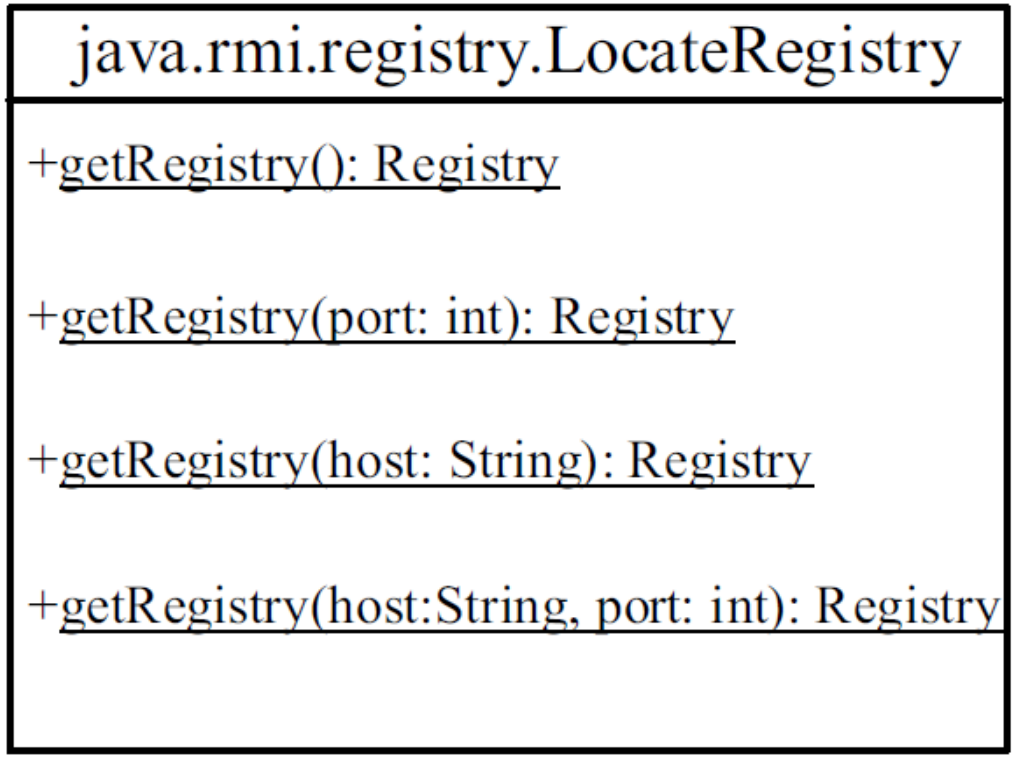
Stub and Skeleton

- The implementation of the RMI architecture is complex, but the good news is that RMI provides a mechanism that liberates you from writing the tedious code for handling parameter passing and invoking remote methods.
- The basic idea is to use two helper classes known as the **stub** and the **skeleton** for handling communications between client and server.
- The stub and the skeleton are automatically generated.

RMI Registry

- How does a client locate the remote object?
- The RMI registry provides the registry services for the server to register the object and for the client to locate the object.
- You can use several overloaded static getRegistry() methods in the LocateRegistry class to return a reference to a Registry.
- Once a Registry is obtained, you can bind an object with a unique name in the registry using the bind or rebind method or locate an object using the lookup method.

The LocateRegistry class provides the methods for obtaining a registry on a host.



Returns a reference to the remote object Registry for the local host on the default registry port of 1099.

Returns a reference to the remote object Registry for the local host on the specified port.

Returns a reference to the remote object Registry on the specified host on the default registry port of 1099.

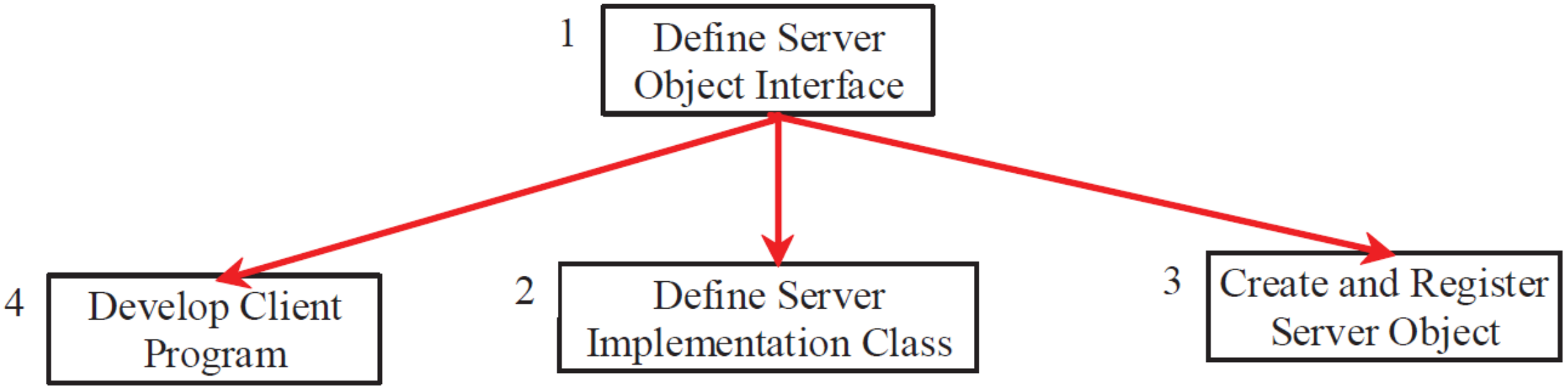
Returns a reference to the remote object Registry on the specified host and port.

The Registry class provides the methods for binding and obtaining references to remote objects in a remote object registry

java.rmi.registry.Registry	
+bind(name: String, obj: Remote): void	Binds the specified name with the remote object.
+rebind(name: String, obj: Remote): void	Binds the specified name with the remote object. Any existing binding for the name is replaced.
+unbind(name: String): void	Destroys the binding for the specified name that is associated with a remote object.
+list(name: String): String[]	Returns an array of the names bound in the registry.
+lookup(name: String): Remote	Returns a reference, a stub, for the remote object associated with the specified name.

Developing RMI Applications

- Now that we have a basic understanding of RMI, we are ready to write simple RMI applications.
- Steps in developing an RMI application:



Step 1: Define Server Object Interface

- Define a server object interface that serves as the contract between the server and its clients, as shown in the following outline:

```
public interface ServerInterface extends Remote {
        public void service1(...) throws RemoteException;
        // Other methods
}
```

- A server object interface must extend the java.rmi.Remote interface.

Step 2: Define Server Implementation Object

- Define a class that implements the server object interface, as shown in the following outline:

```
public class ServerInterfaceImpl extends UnicastRemoteObject
                implements ServerInterface {
        public void service1(...) throws RemoteException {
                // Implement it
        }
        // Implement other methods
}
```

- The server implementation class must extend the **java.rmi.server.UnicastRemoteObject** class.
- The UnicastRemoteObject class provides support for point-to-point active object references using TCP streams.

Step 3: Create and Register Server Object

- Create a server object from the server implementation class and register it with an RMI registry:

```
ServerInterface server = new ServerInterfaceImpl(...);
Registry registry = LocateRegistry.getRegistry();
registry.rebind("RemoteObjectName", server);
```

Step 4: Develop Client Program

- Develop a client that locates a remote object and invokes its methods, as shown in the following outline:

```
Registry registry = LocateRegistry.getRegistry(host);
ServerInterface server = (ServerInterface)
        registry.lookup("RemoteObjectName");
server.service1(...);
```

Example: Retrieving Student Scores from an RMI Server

- This example creates a client that retrieves student scores from an RMI server.
- The client displays the score for the specified name.
- You can get the score by entering a student name and clicking the Get Score button.

Example Steps

1) Create a server interface named **StudentServerInterface**. The interface tells the client how to invoke the server's **findScore** method to retrieve a student score.
2) Create a server implementation named **StudentServerInterfaceImpl** that implements **StudentServerInterface**. The **findScore** method returns the score for a specified student. It returns -1 if the score is not found.

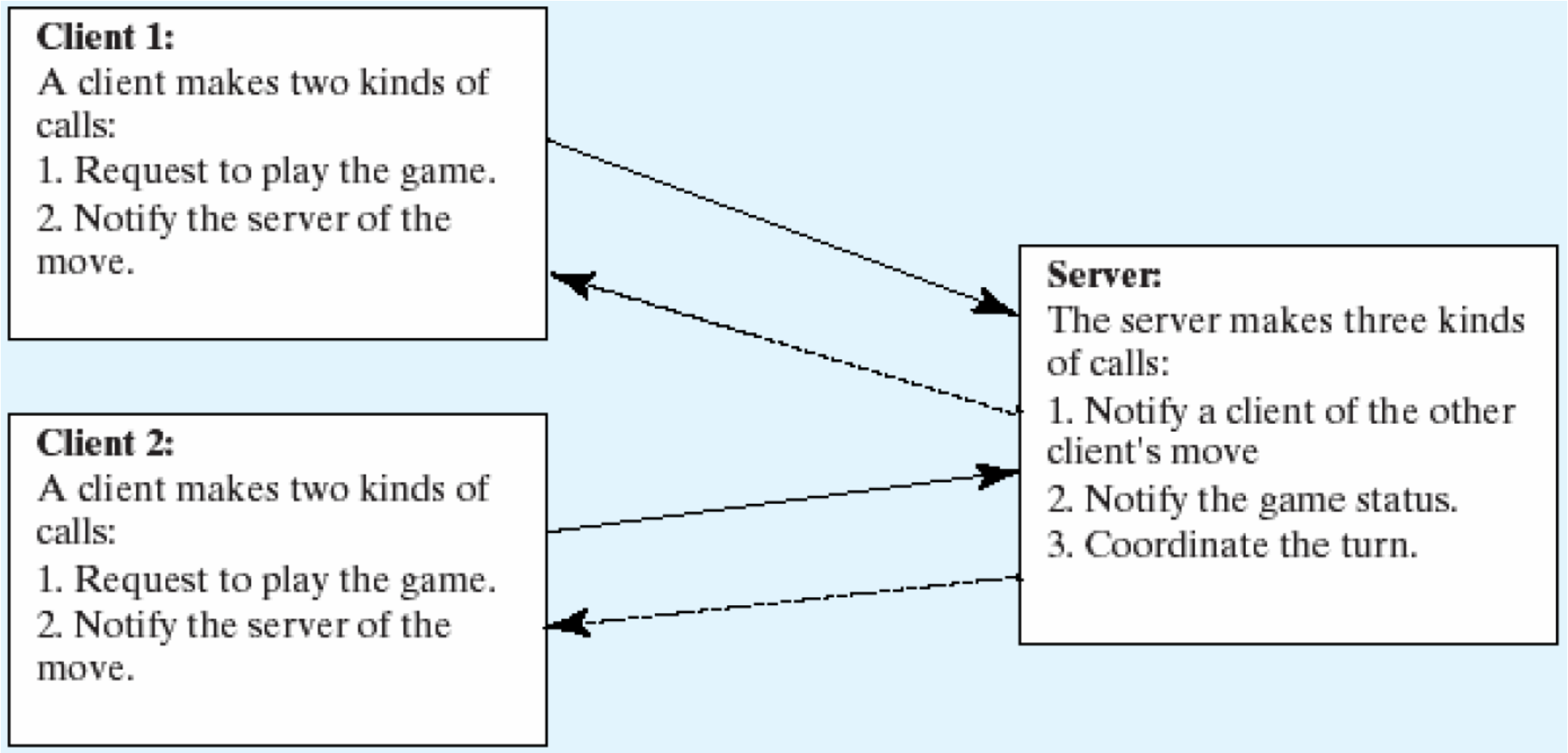
Example Steps (Cont.)

3) Create a server object from the server implementation and register it with the RMI server.
4) Create a client as an application named StudentServerInterfaceClient. The client locates the server object from the RMI registry and uses it to find the scores.

How to Run

- Start the RMI Registry by typing "**start rmiregistry**" at a DOS prompt from the bin directory. By default, the port number 1099 is used by rmiregistry. To use a different port number, simply type the command "**start rmiregistry *portnumber***" at a DOS prompt.
- Compile **All Classes**
- Run the server **RegisterWithRMIServer**
- Run the client **StudentServerInterfaceClient**

Optional Project



Optional Project - Cont'd

- All the calls a client makes can be encapsulated in one remote interface named **TicTacToe.java**
- All the calls the server invokes can be defined in another interface named **CallBack.java**
- Create **TicTacToeImpl.java** to implement TicTacToeInterface. Add a main method in the program to register the server with the RMI.
- Create **CallBackImpl.java** to implement the CallBack Interface.