Distributed Objects

Request-Response

- A message exchange pattern
  - Knock-Knock Protocol.
  - High-Low Guess (HW #1)
    - HTTP
- Tightly coupled software components
- Synchronous communication
- Advantage: Simple. Easy to detect problems.
- Disadvantage: Less throughput because client waits

Remote Procedure Call

- Causing a subroutine to execute in another process (maybe on another computer)

- Implemented as a request-response protocol

- To the programmer an RPC looks like a local procedure call.

  - There is some additional overhead

    - Server has to publish the remote object

    - Client has to obtain a reference to the remote object

Java RMI

- Source: https://docs.oracle.com/javase/tutorial/rmi/

RMI Overview

- Client-server model using distributed objects
- Server
  - Creates remote objects
  - Makes references to remote objects available
- Client
  - Gets references to remote objects
  - Calls methods on remote objects

RMI Registry

- A name server for remote Java objects
- Can be embedded in server or run as a separate process
- Methods
  - void bind(String name, Remote obj)
  - String[] list()
  - Remote lookup(String name)
  - void rebind(String name, Remote obj)
  - void unbind(String name)

RMI Example

Hello Interface

```
package myrmi;

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

public interface Hello extends Remote {
    String sayHello(String name) throws RemoteException;
}
```

Server 1

```
package myrmi;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

public class Server implements Hello {

    public Server() {}

    @Override
    public String sayHello(String name) throws RemoteException {
        StringBuilder sb = new StringBuilder("Hello, ");
        if (name == null) {
            sb.append("world");
        } else {
            sb.append(name);
        }

        return sb.toString();
    }
```

Server 2

```
public static void main(String[] args) {
    try {
        Server obj = new Server();
        Hello stub = (Hello)UnicastRemoteObject.exportObject(obj, 0);

        Registry registry = LocateRegistry.getRegistry();
        registry.rebind("Hello", stub);

        System.out.println("Server ready");
    } catch (Exception re) {
        re.printStackTrace();
    }
  }
}
```

Client

```
package myrmi;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {
    public static void main(String[] args) {
        String host = (args.length < 1 ? null : args[0]);
        try {
            Registry registry = LocateRegistry.getRegistry(host);
            Hello stub = (Hello) registry.lookup(“Hello");

            String response = stub.sayHello(null);
            System.out.println("response: " + response);
            response = stub.sayHello("David");
            System.out.println("response: " + response);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```