Java.rmi.Naming:

The Naming class provides methods for storing and obtaining references to remote objects in a remote object registry. Each method of the Naming class takes as one of its arguments a name that is a java.lang.String in URL format (without the scheme component) of the form: //host:port/name

Naming.lookup():

```
public static Remote lookup(String name)
    throws NotBoundException,
       java.net.MalformedURLException,
       RemoteException
{
    ParsedNamingURL parsed = parseURL(name);
    Registry registry = getRegistry(parsed);

    if (parsed.name == null)
       return registry;
    return registry.lookup(parsed.name);
}
```

Naming.bind():

```
public static void bind(String name, Remote obj)
    throws AlreadyBoundException,
       java.net.MalformedURLException,
       RemoteException
{
    ParsedNamingURL parsed = parseURL(name);
    Registry registry = getRegistry(parsed);

    if (obj == null)
       throw new NullPointerException("cannot bind to null");

    registry.bind(parsed.name, obj);
}
```

Naming.rebind():

```
public static void rebind(String name, Remote obj)
    throws RemoteException, java.net.MalformedURLException
{
    ParsedNamingURL parsed = parseURL(name);
    Registry registry = getRegistry(parsed);

    if (obj == null)
       throw new NullPointerException("cannot bind to null");

    registry.rebind(parsed.name, obj);
}
```

Naming.unbind():

```
public static void unbind(String name)
    throws RemoteException,
       NotBoundException,
       java.net.MalformedURLException
{
    ParsedNamingURL parsed = parseURL(name);
    Registry registry = getRegistry(parsed);

    registry.unbind(parsed.name);
}
```

RMI 实例：
Registry:

```
public class ServerSimple {
    public static void main(String[] args) {
        try {
            Service service1 = new ServiceImpl("service1");
            Service service2 = new ServiceImpl("service2");
            // 创建并启动注册器 Registry
            Registry registry = LocateRegistry.createRegistry(9999);
            System.out.println("\n 服务器已启动...\n");

            registry.rebind("RemoteService1",service1); // service1 是一个 远程对象，名字是
RemoteService1
            registry.rebind("RemoteService2",service2);

            System.out.println("创建了两个远程对象: ");
            System.out.println(service1.getClass().getName());
            System.out.println(service2.getClass().getName());
            // 该程序不会停止，因为 RMI 注册器会一直监听 9999 端口，监听客户端有没有查找远程对象
        } catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

Service:

```
public class ServiceImpl extends UnicastRemoteObject implements Service {

    private String name;
    protected ServiceImpl(String name) throws RemoteException {
        this.name = name;
    }
    // client -远程调用-> server.service
    // 在 server 执行完毕后，将结果返回 client
    @Override
    public String echo(String msg) throws RemoteException{
        System.out.println(name + ": 调用 echo()方法。");
        return "echo: " + msg + " from " + name;
    }
}
```

Client:

```
public class ClientSimple {
    public static void main(String[] args) {
        try{
            Registry registry = LocateRegistry.getRegistry(9999); // 得到端口号为 9999 的
RMI 注册器
            Service service1 = (Service) registry.lookup("RemoteService1");
            Service service2 = (Service) registry.lookup("RemoteService2");

            Class stubClass = service1.getClass();
            System.out.println("service1 是"+ stubClass.getName() +"的实例");
            Class[] stubInterface = stubClass.getInterfaces();
            for (int i = 0; i < stubInterface.length; i++){
                System.out.println(stubInterface[i]);
            }
            System.out.println(service1.echo("Hello!"));
            System.out.println(service2.echo("Hi!"));
        } catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

Socket 多线程实例：

Server:

```
public class MyServer {
    public static void main(String[] args) throws IOException {
        // 默认本机 IP
        // 暴露一个服务，本机 IP：9999
        ServerSocket server = new ServerSocket(9999);
        System.out.println("启动 Server，本机 IP：端口" + server.getInetAddress() + ":" +
server.getLocalPort());
        // 阻塞在此，等待客户端的连接，才能继续执行
        while (true){
            Socket serverSocket = server.accept(); // 用于监听是否有客户端访问，返回一个
Socket 对象
            System.out.println("与客户端连接成功！");
            MultiProc multiProc = new MultiProc(serverSocket);
            // Runnable -> Thread
            new Thread(multiProc).start(); // 启动线程
        }
    }
}
```

Multi-Threads:

```
public  class MultiProc implements Runnable {
    // 构造方法传参
    public MultiProc(Socket socket) {
        this.socket = socket;
    }

    private Socket socket;

    @Override
    public  void run() {
        try{
            // Server 向 Client 发送消息
            OutputStream out = socket.getOutputStream();
            //byte[] buffer = new byte[1024];
            // String -> byte : String.getBytes();
            byte[] write Buffer  = "Hello Client! I am Server".getBytes
(StandardCharsets.UTF_8);
            out.write(write Buffer);

            // 接收 Client 发送的消息
            InputStream in = socket.getInputStream();
            byte[] in Buffer = new byte[100];
            in.read(in Buffer);
            System.out.println("Client -> Server 接收到的消息是："+new String(in Buffer));

            in.close();
            out.close();
            socket.close();
        } catch (Exception e){
            e.printStackTrace();
        }

    }
}
```

Client:

```
public class MyClient {
    public static void main(String[] args) throws IOException {
        // 客户端 连接 服务端 发布的服务
        Socket client = new Socket("127.0.0.1", 9999);

        // 接收 Server 发送的消息 InputStream
        InputStream in = client.getInputStream();
        byte[] in Buffer = new byte[100];
        in.read(in Buffer); // 把 in 的数据 读取到 buffer 里
        // byte[] -> String
        System.out.println("Server -> Client 接收到的消息是："+new String(in Buffer));

        // Client 向 Server 发送消息
        OutputStream out = client.getOutputStream();
        out.write("你好 Server，我是 Client".getBytes(StandardCharsets.UTF_8));

        in.close();
        out.close();
        client.close();

    }
}
```