Java Remote Method Invocation Java RMI

61FIT3NPR -Network Programming

Faculty of Information Technology
Hanoi University
Fall 2020

Tutorial Contents

- Java Remote Method Invocation
- Historical Background
- Related Terminologies
- RMI System Architecture
  - Layered Structures
  - Working Principles
- A Simple RMI Application
  - Server, Client, Interface, Stubs
  - Security, Deployment, Invocation
- Strength & Weakness of RMI
- RMI vs. CORBA Case Study

Java Remote Method Invocation

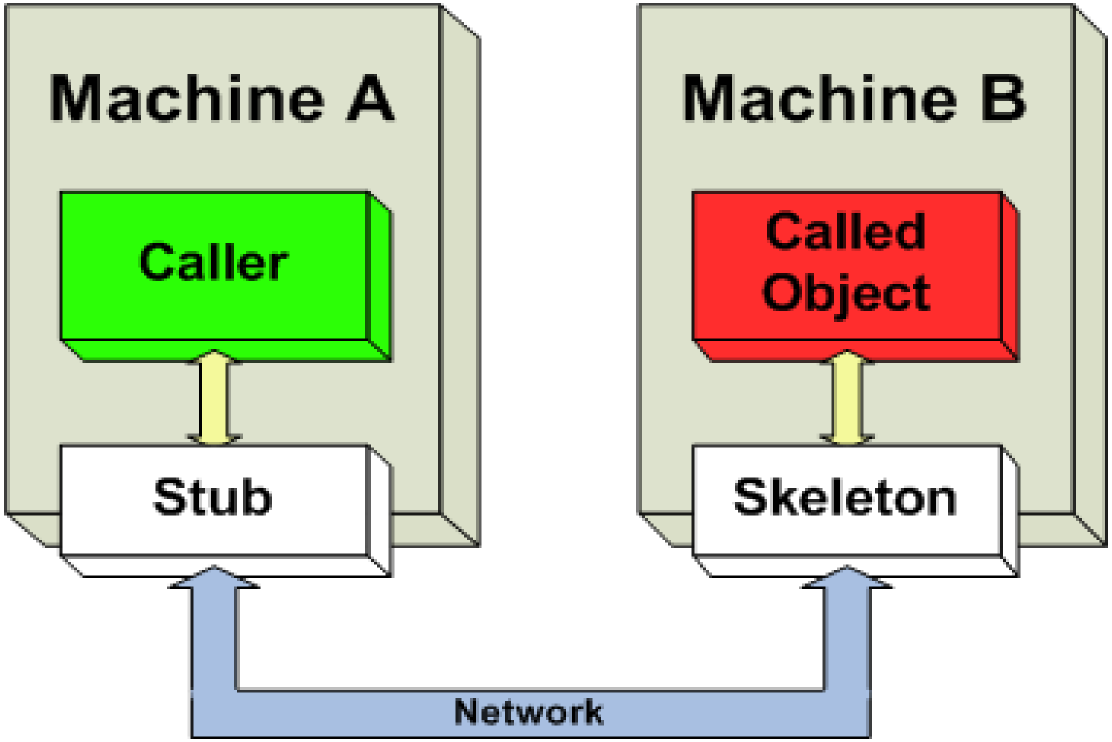
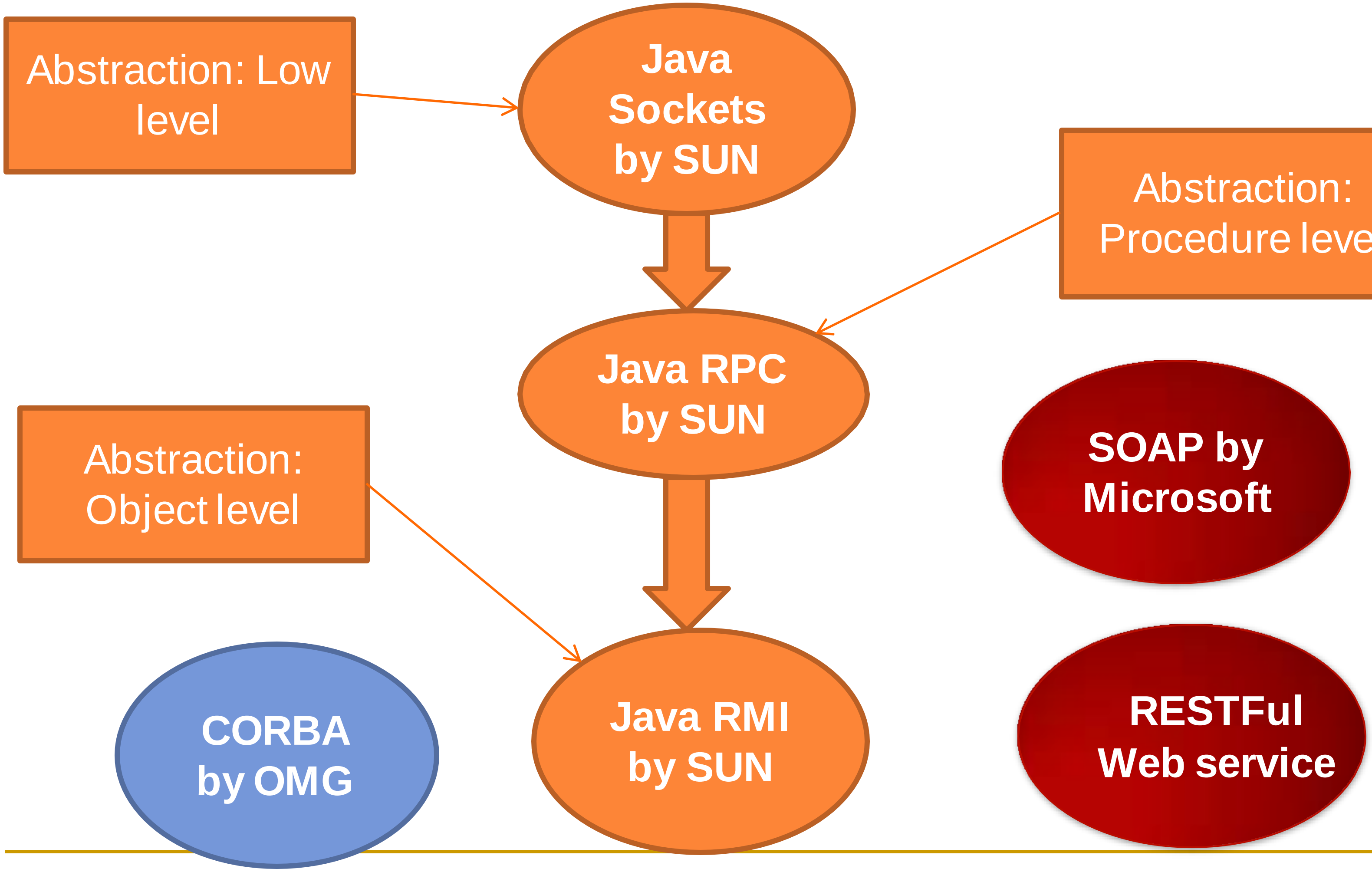


Fig: Distributed Object Technology

Java Remote Method Invocation

- RMI Server, client, interface, stubs, skeletons
- RMI Registry
- Object version of RPC
- Method Invocation between JVMs
- Java RMI API
- JRMP (Java Remote Method Protocol)
- Java object serialization
- Parameter Marshalling

Historical Background



Abstraction: Low level

Java Sockets by SUN

Abstraction: Procedure level

Java RPC by SUN

SOAP by Microsoft

Abstraction: Object level

CORBA by OMG

Java RMI by SUN

RESTFul Web service

Related Terminologies

- RPC (Remote Procedure Call)
- XDR (External Data Representation)
- C**ORB**A (Common **Object Request Broker** Architecture)
- IIOP (Internet Inter-ORB Protocol)
- Java IDL (Interface Definition Language)
- RMI-IIOP
- SOAP (Simple Object Access Protocol)

RMI System Architecture

Lets divide into two perspectives:

- Layered Structure
- Working Principles

RMI Layered Structure

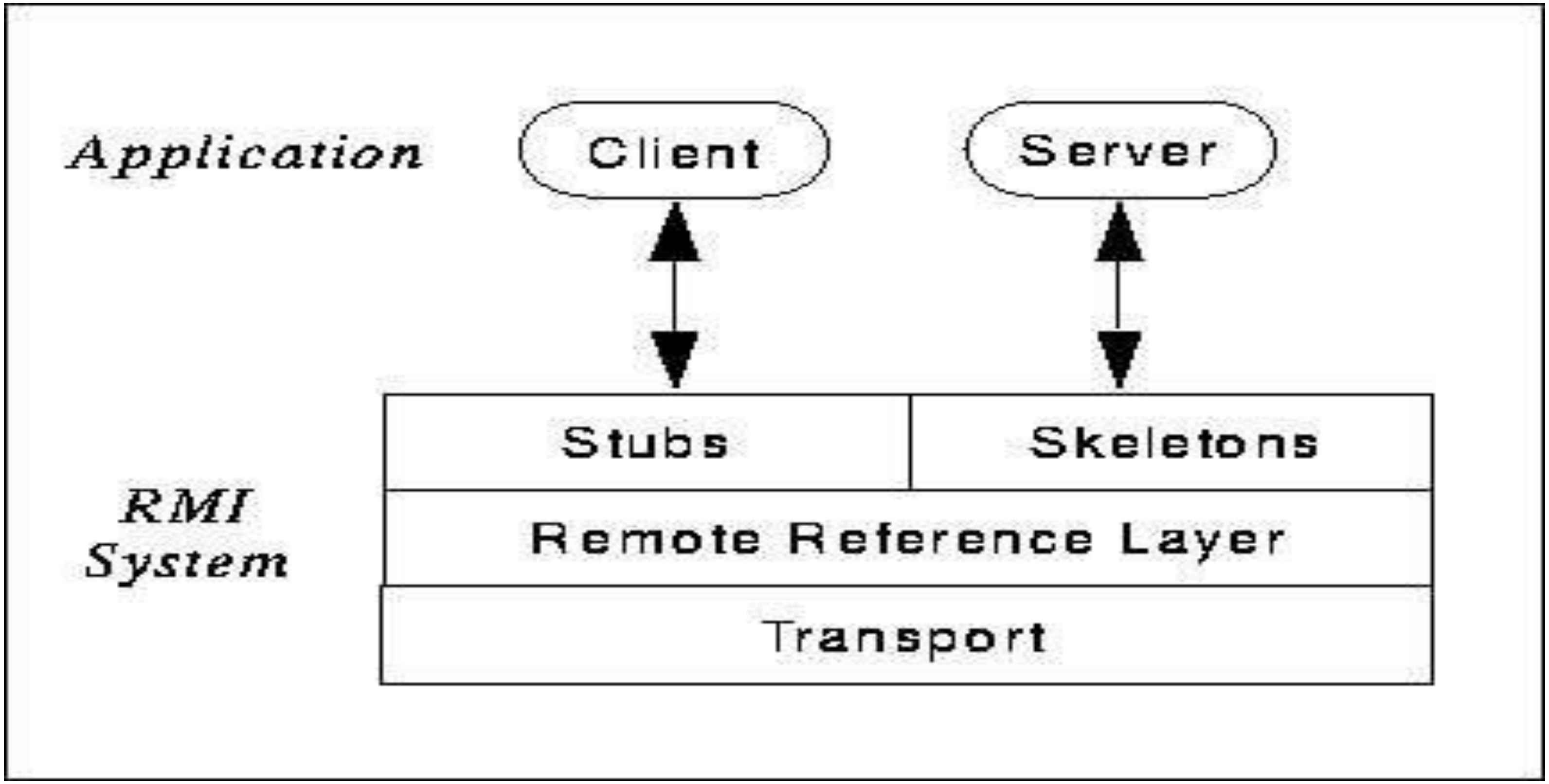


Fig: RMI Layered Structure

RMI Layered Structure

- Application layer: Server, Client
- Interface: Client stub, Server skeleton
- Remote Reference layer: RMI registry
- Transport layer: TCP

RMI Working Principles

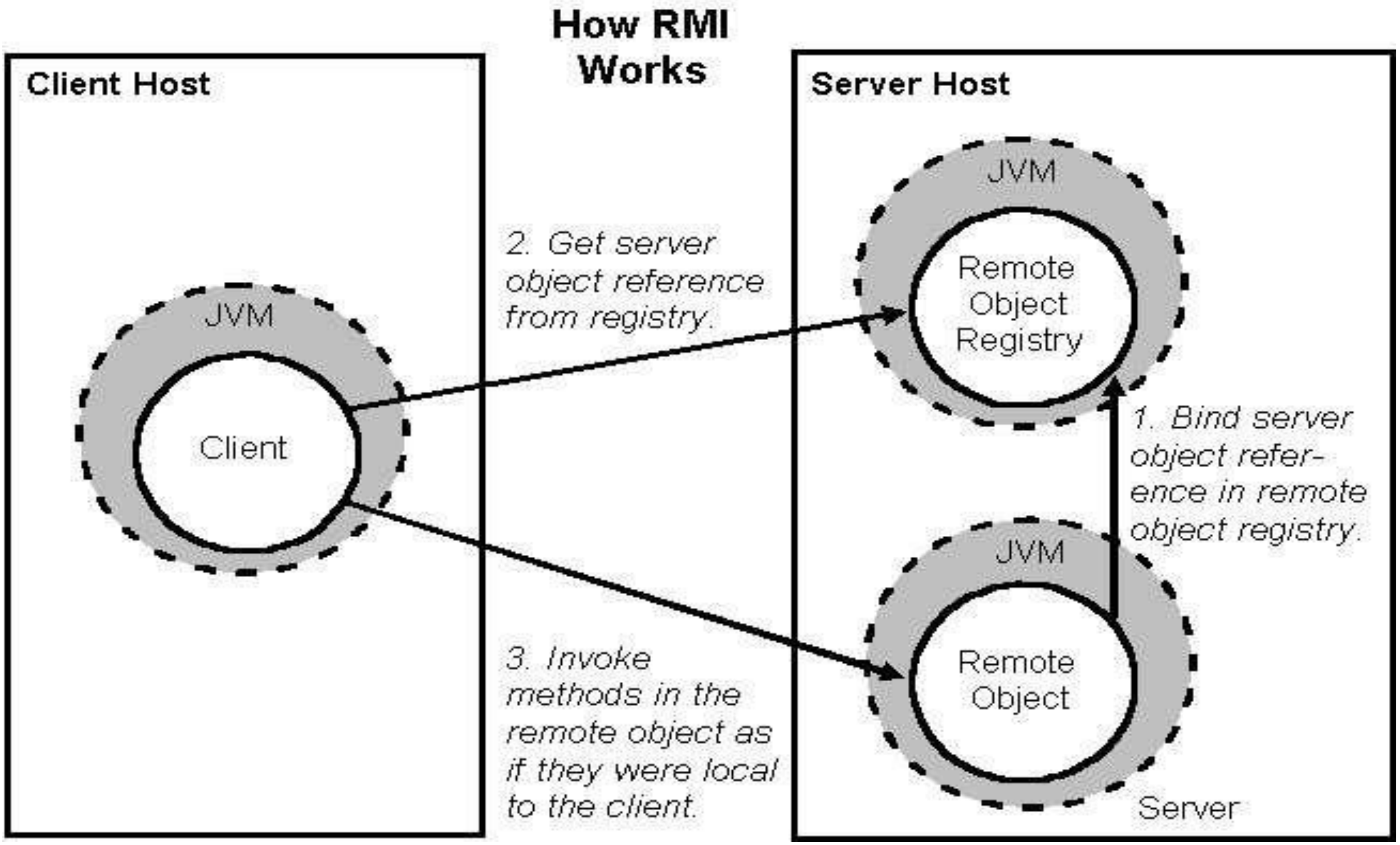
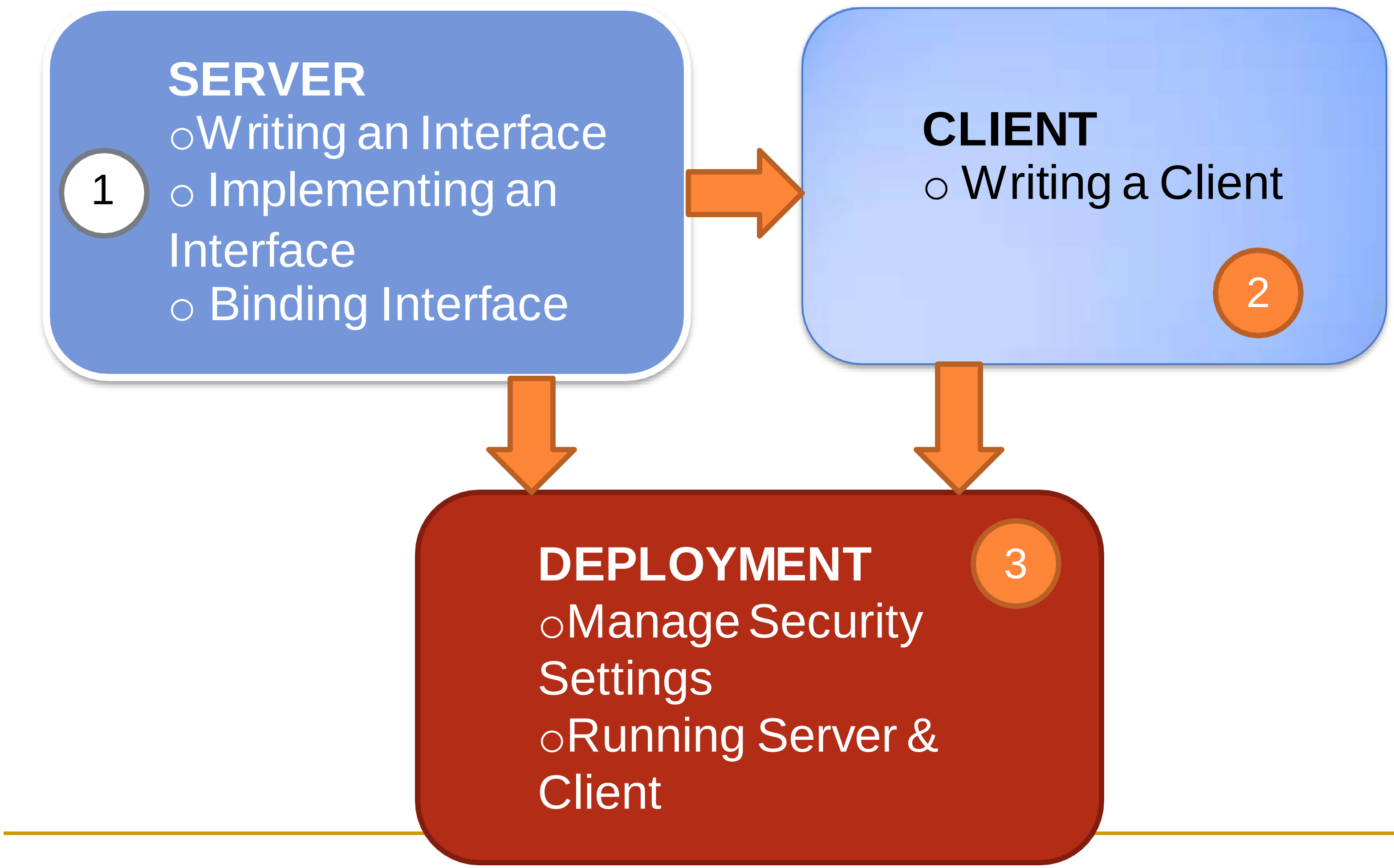

How RMI Works
Client Host
JVM
Client
2. Get server object reference from registry.
3. Invoke methods in the remote object as if they were local to the client.
Server Host
JVM
Remote Object Registry
1. Bind server object refer- ence in remote object registry.
JVM
Remote Object
Server

Fig: RMI Working principles

Ready to Develop One?

A SIMPLE RMI APPLICATION



SERVER

- ○Writing an Interface
- ○ Implementing an Interface
- ○ Binding Interface

CLIENT

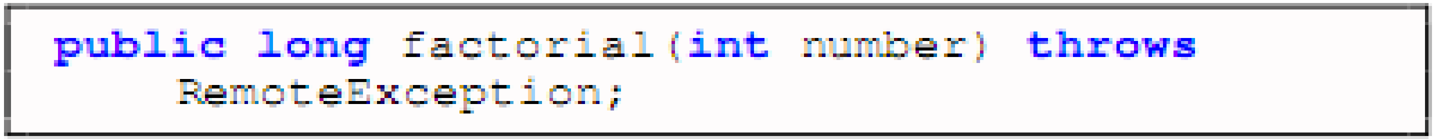
- ○ Writing a Client

DEPLOYMENT

- ○Manage Security Settings
- ○Running Server & Client

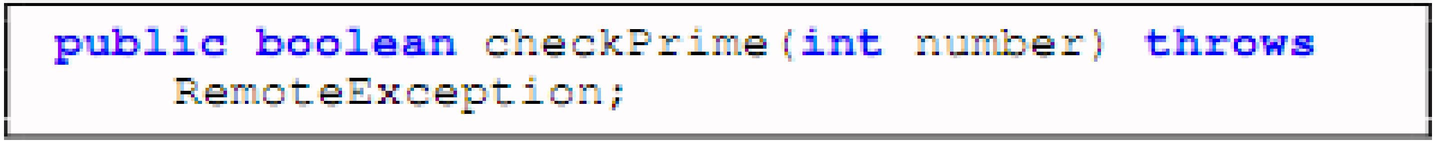
Service Interface: An Agreement Between Server & Client

- Factorial Operation



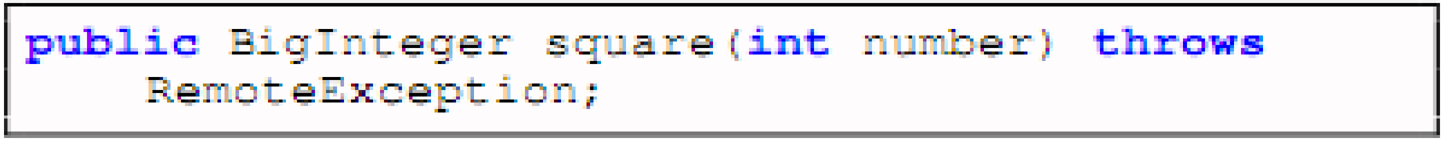
```
public long factorial(int number) throws
        RemoteException;
```

- Check Prime Operation



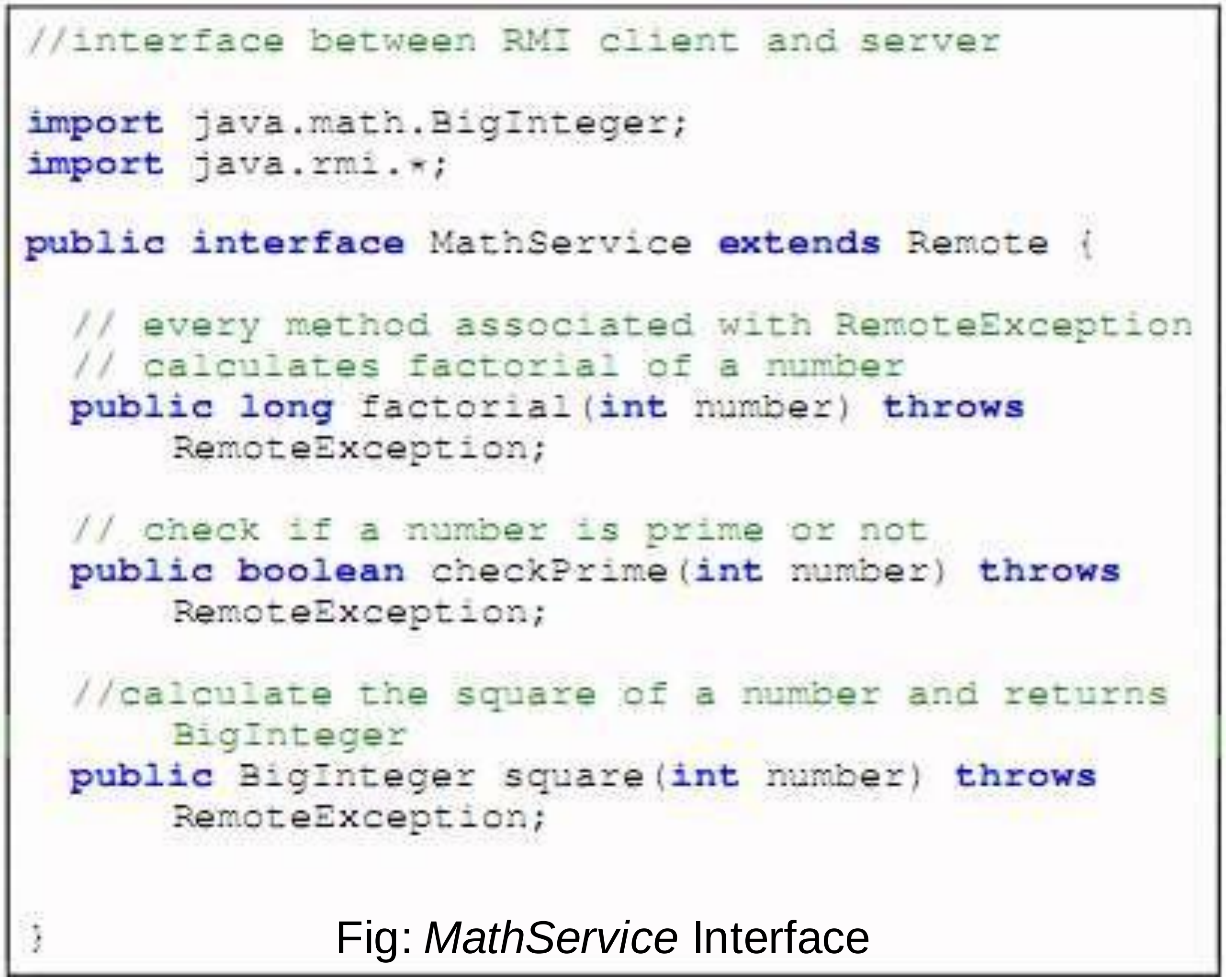
```
public boolean checkPrime(int number) throws
        RemoteException;
```

- Square Operation



```
public BigInteger square(int number) throws
        RemoteException;
```

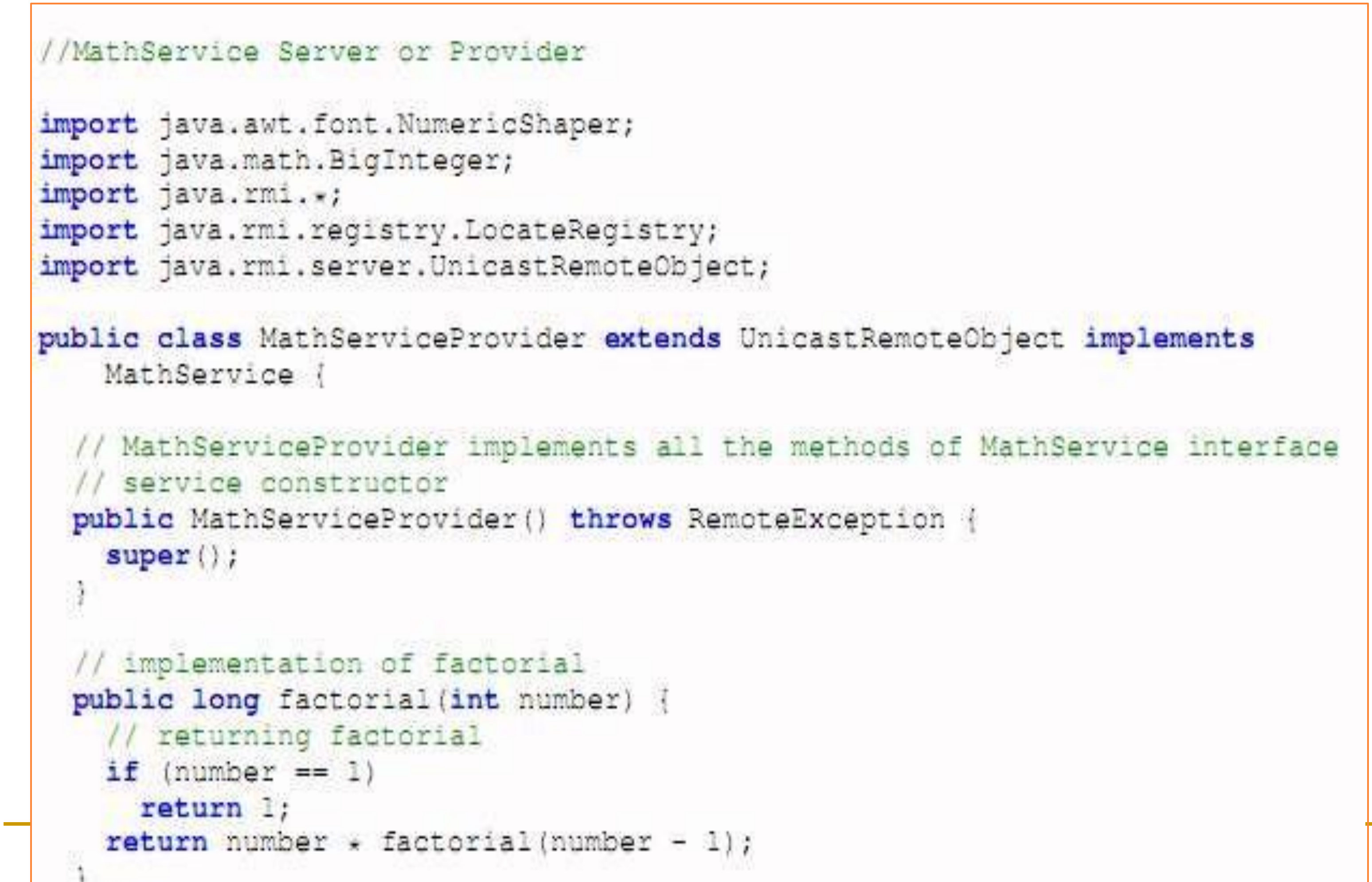
Server Application: Writing a Service Interface



```
//interface between RMI client and server

import java.math.BigInteger;
import java.rmi.*;

public interface MathService extends Remote {

   // every method associated with RemoteException
   // calculates factorial of a number
   public long factorial(int number) throws
        RemoteException;

   // check if a number is prime or not
   public boolean checkPrime(int number) throws
        RemoteException;

   //calculate the square of a number and returns
        BigInteger
   public BigInteger square(int number) throws
        RemoteException;


}
```

Fig: *MathService* Interface

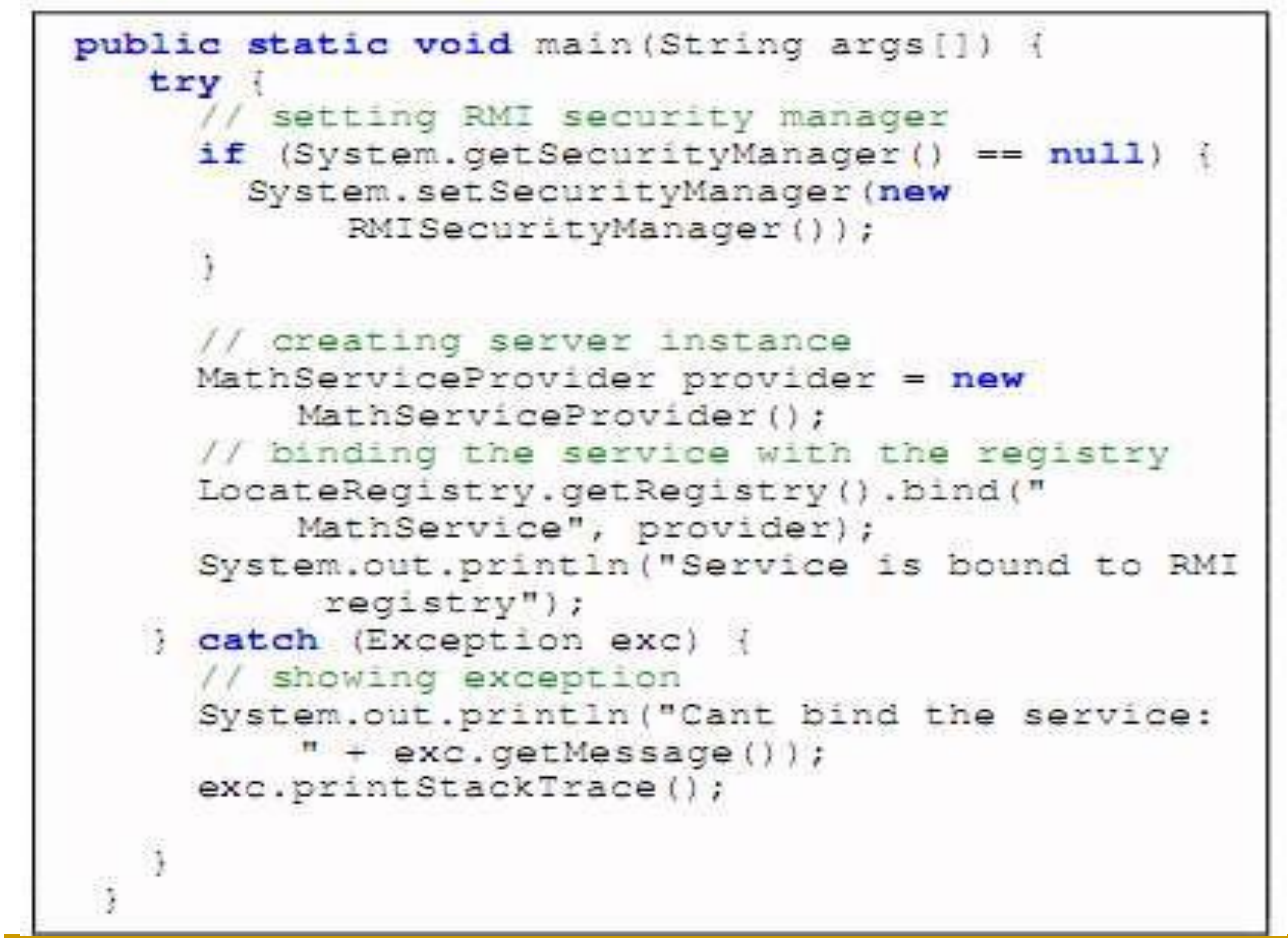
Server Application: Implementing the Service Interface



```
//MathService Server or Provider

import java.awt.font.NumericShaper;
import java.math.BigInteger;
import java.rmi.*;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;

public class MathServiceProvider extends UnicastRemoteObject implements
    MathService {

  // MathServiceProvider implements all the methods of MathService interface
  // service constructor
  public MathServiceProvider() throws RemoteException {
    super();
  }

  // implementation of factorial
  public long factorial(int number) {
    // returning factorial
    if (number == 1)
      return 1;
    return number * factorial(number - 1);
  }
```

Fig: *MathServiceProvider* implements *MathService* Interface

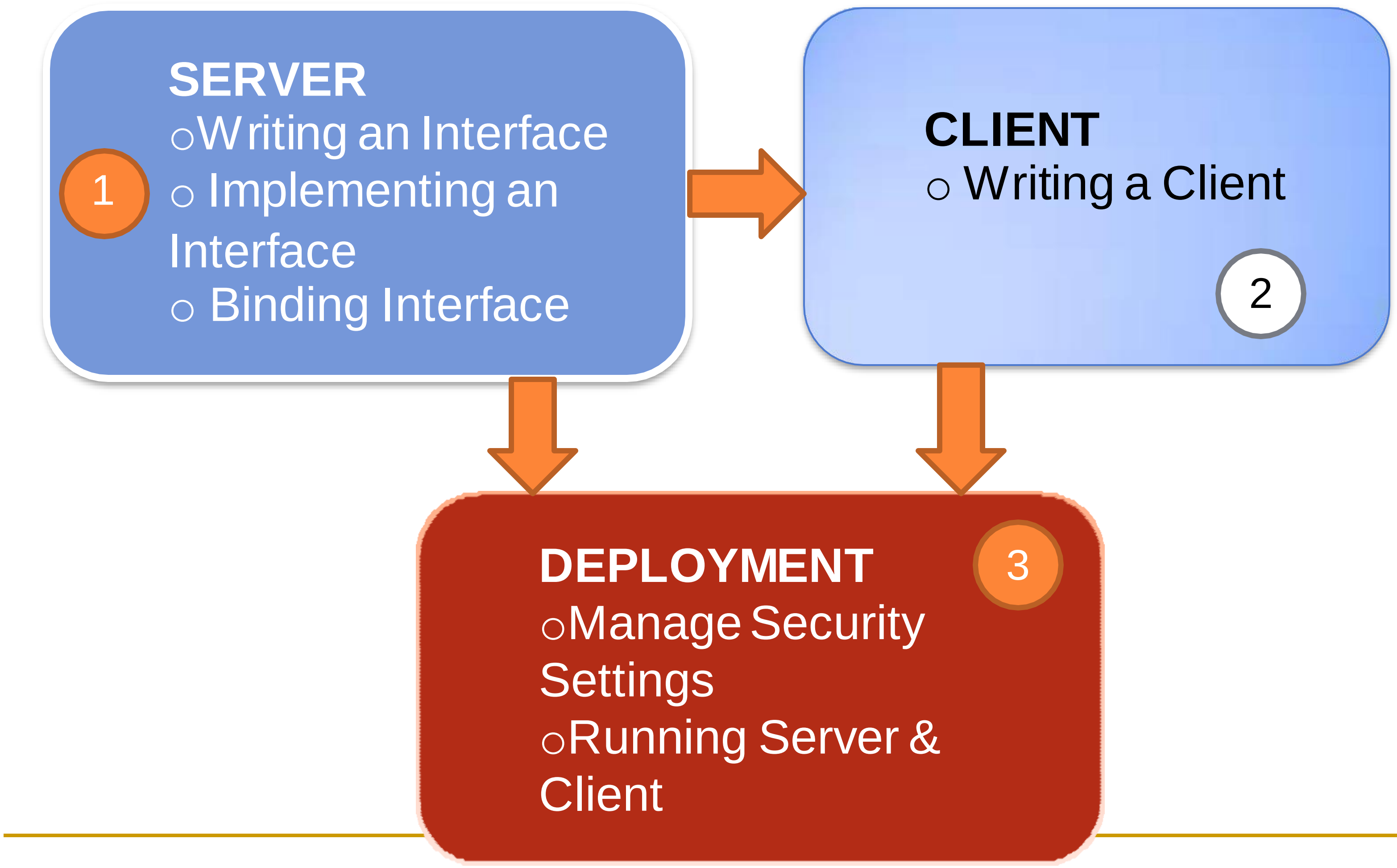
Server Application: Instantiating & Binding the Service



```
public static void main(String args[]) {
   try {
      // setting RMI security manager
      if (System.getSecurityManager() == null) {
         System.setSecurityManager(new
             RMISecurityManager());
      }

      // creating server instance
      MathServiceProvider provider = new
          MathServiceProvider();
      // binding the service with the registry
      LocateRegistry.getRegistry().bind("
          MathService", provider);
      System.out.println("Service is bound to RMI
           registry");
   } catch (Exception exc) {
      // showing exception
      System.out.println("Cant bind the service:
          " + exc.getMessage());
      exc.printStackTrace();

   }
 }
```

Fig: Instantiating and Binding *MathService* Interface

A Simple RMI Application



1

SERVER

- ○Writing an Interface
- ○ Implementing an Interface
- ○ Binding Interface

CLIENT

- ○ Writing a Client

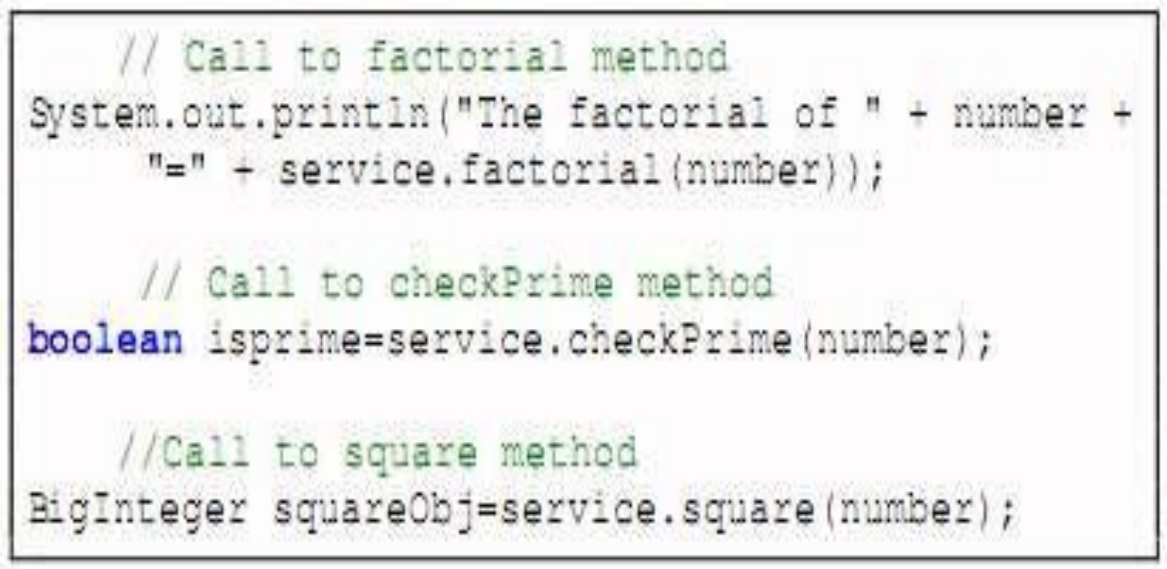
2

DEPLOYMENT

3

- ○Manage Security Settings
- ○Running Server & Client

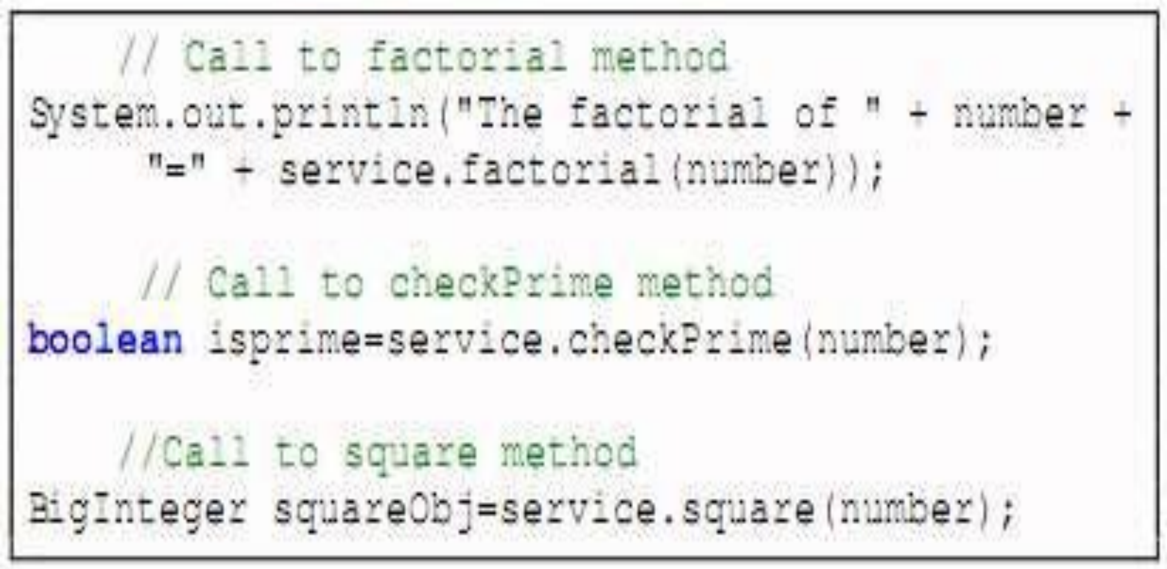
Client Application: Service Lookup



```
    // Call to factorial method
System.out.println("The factorial of " + number +
     "=" + service.factorial(number));

     // Call to checkPrime method
boolean isprime=service.checkPrime(number);

    //Call to square method
BigInteger squareObj=service.square(number);
```

Fig: Client locating *MathService* service

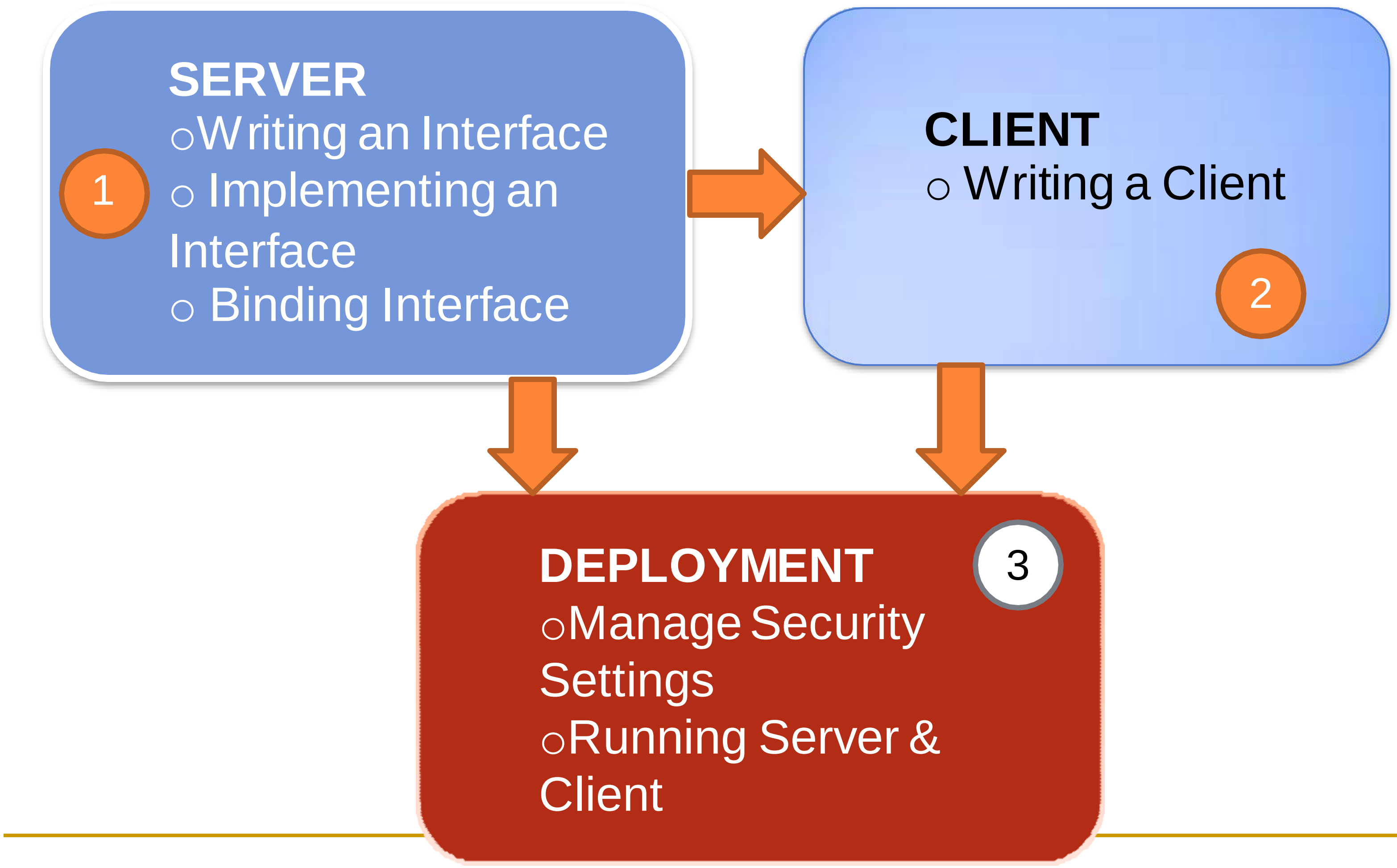
Client Application: Accessing Service



```
    // Call to factorial method
System.out.println("The factorial of " + number +
     "=" + service.factorial(number));

     // Call to checkPrime method
boolean isprime=service.checkPrime(number);

    //Call to square method
BigInteger squareObj=service.square(number);
```

Fig: Client accessing *MathService* service

A Simple RMI Application



SERVER

- Writing an Interface
- Implementing an Interface
- Binding Interface

CLIENT

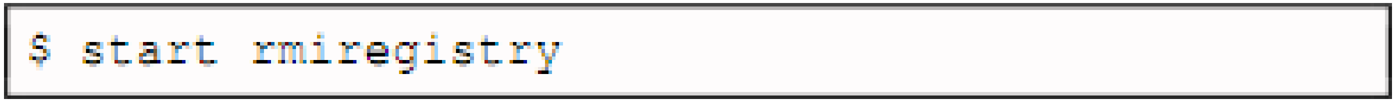
- Writing a Client

DEPLOYMENT

- Manage Security Settings
- Running Server & Client

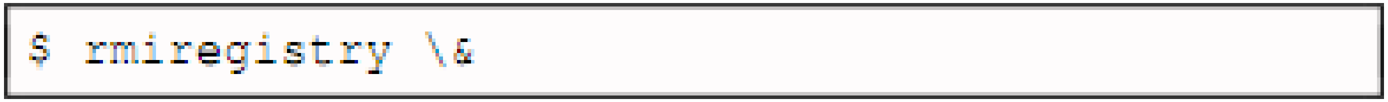
Server Deployment: Start RMI Registry

- To start RMI registry on windows



```
$ start rmiregistry
```

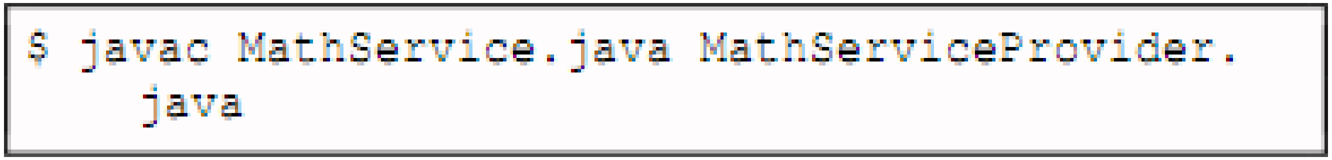
- To start RMI registry on Unix



```
$ rmiregistry \&
```

Server Deployment: Compile the Server

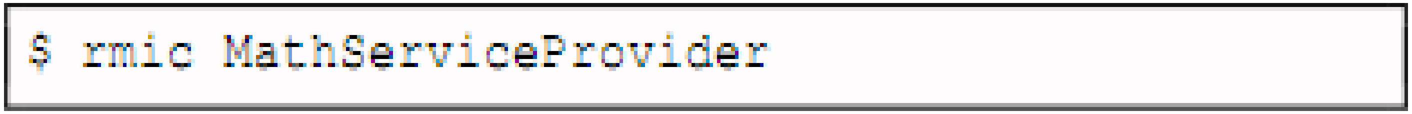
- Compile both MathService interface and MathServiceProvider class



```
$ javac MathService.java MathServiceProvider.
    java
```

Server Deployment: Create Server Stub

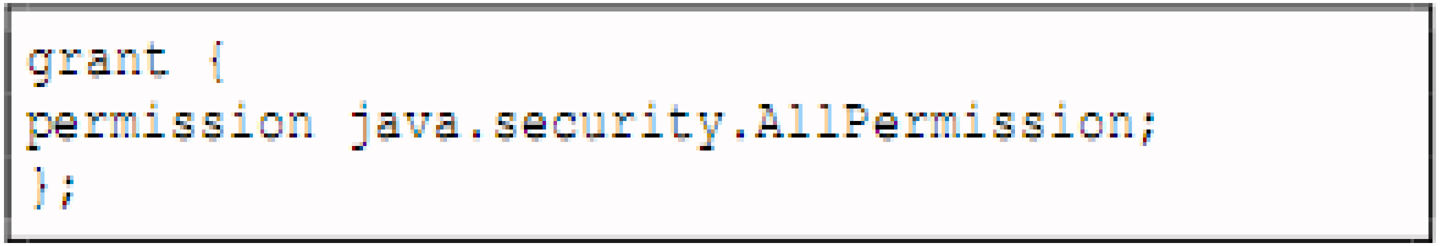
- Create the server stub that will handle client call



```
$ rmic MathServiceProvider
```

Security Deployment: Create Security Policy file (Both Client & Server)

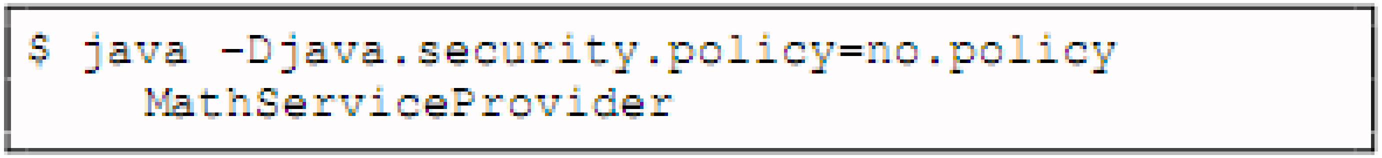
- Create a security policy file called *no.policy* with the following content and add it to CLASSPATH
- This step implies for both server and client



```
grant {
permission java.security.AllPermission;
};
```

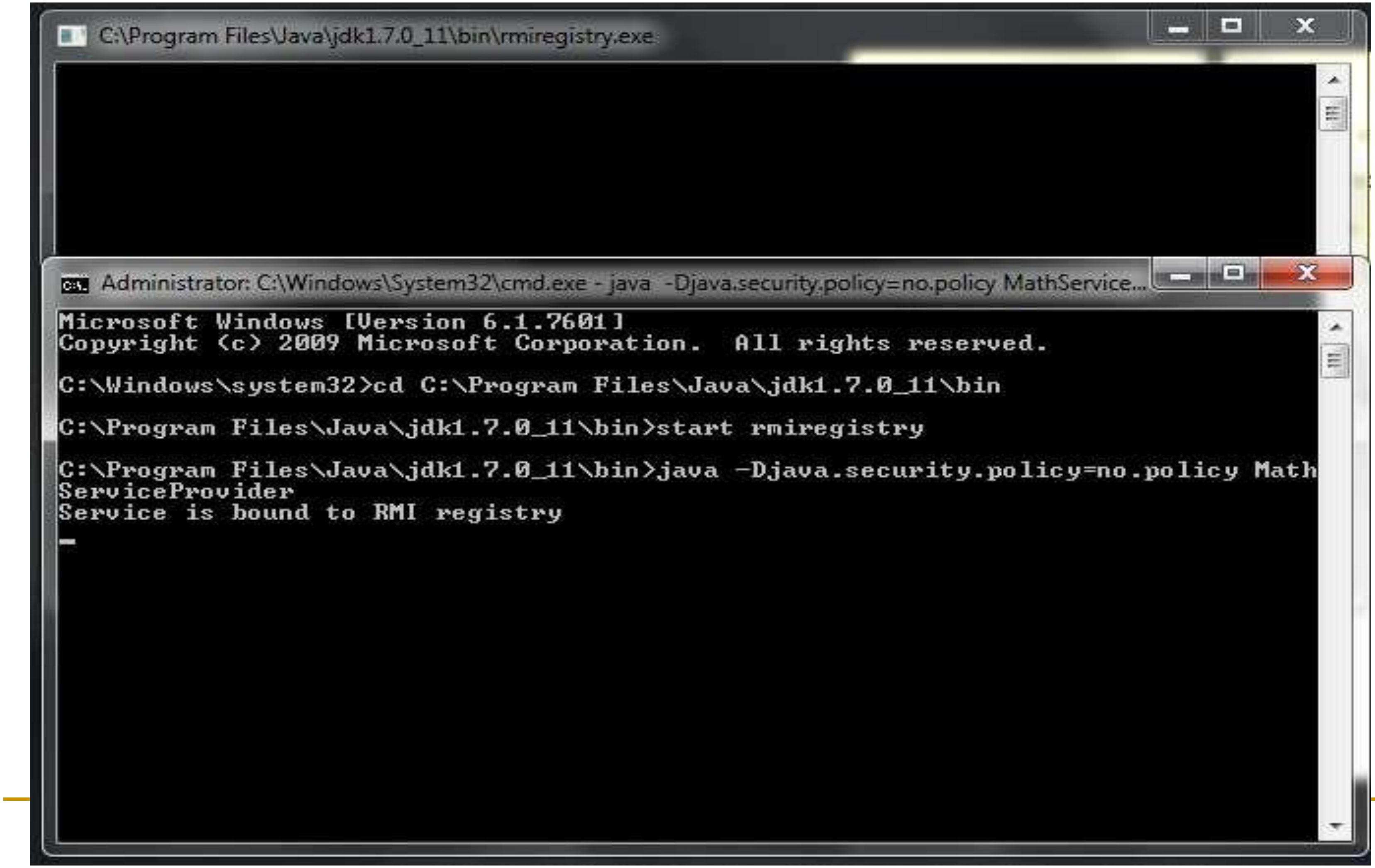
START THE SERVER

- Execute the command to run server



```
$ java -Djava.security.policy=no.policy
    MathServiceProvider
```

Server Running



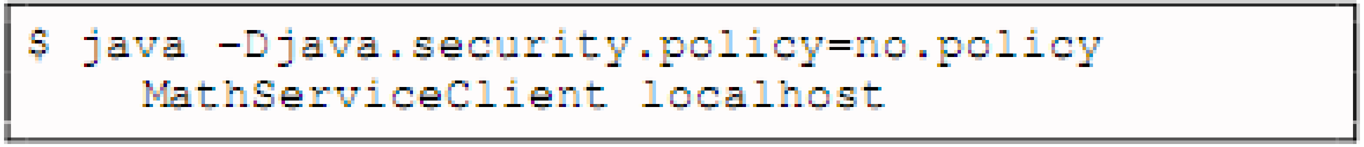
C:\Program Files\Java\jdk1.7.0_11\bin\rmiregistry.exe

Administrator: C:\Windows\System32\cmd.exe - java -Djava.security.policy=no.policy MathService...

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>cd C:\Program Files\Java\jdk1.7.0_11\bin

C:\Program Files\Java\jdk1.7.0_11\bin>start rmiregistry

C:\Program Files\Java\jdk1.7.0_11\bin>java -Djava.security.policy=no.policy Math
ServiceProvider
Service is bound to RMI registry
_
```

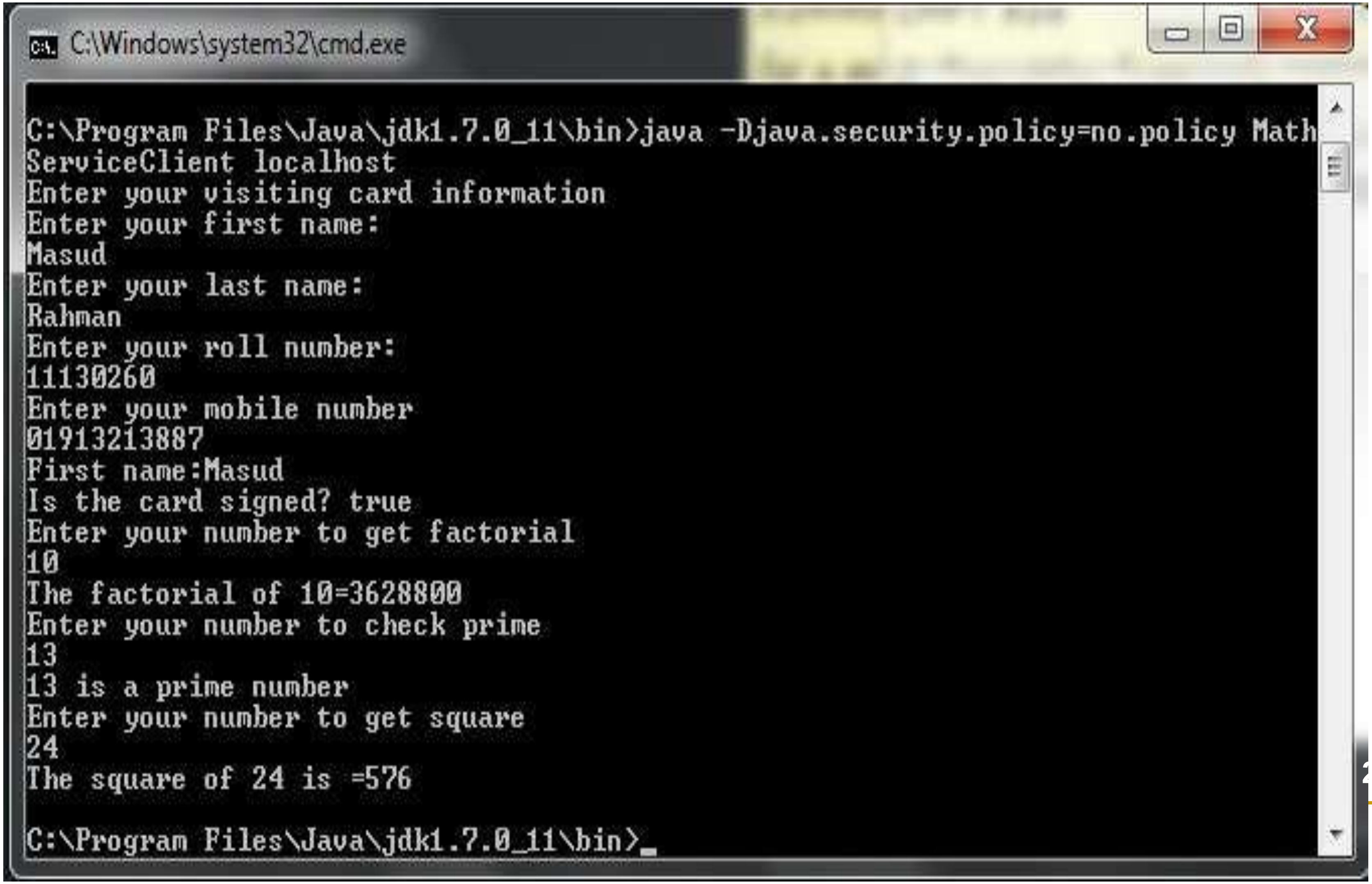
START THE CLIENT

- Execute the command to run client



```
$ java -Djava.security.policy=no.policy
    MathServiceClient localhost
```

Client Interface



```
C:\Program Files\Java\jdk1.7.0_11\bin>java -Djava.security.policy=no.policy Math
ServiceClient localhost
Enter your visiting card information
Enter your first name:
Masud
Enter your last name:
Rahman
Enter your roll number:
11130260
Enter your mobile number
01913213887
First name:Masud
Is the card signed? true
Enter your number to get factorial
10
The factorial of 10=3628800
Enter your number to check prime
13
13 is a prime number
Enter your number to get square
24
The square of 24 is =576

C:\Program Files\Java\jdk1.7.0_11\bin>_
```

Advanced Concepts

- Java Object Serialization
- Parameter Marshalling & Demarshalling
- Object Activation

Strength Of Java RMI

- *Object Oriented*: Can pass complex object rather than only primitive types
- *Mobile Behavior*: Change of roles between client and server easily
- *Design Patterns*: Encourages OO design patterns as objects are transferred
- *Safe & Secure*: The security settings of Java framework used
- *Easy to Write /Easy to Use*: Requires very little coding to access service

Strength Of Java RMI

- *Connects to Legacy Systems*: JNI & JDBC facilitate access.
- *Write Once, Run Anywhere*: 100% portable, run on any machine having JVM
- *Distributed Garbage Collection*: Same principle like memory garbage collection
- *Parallel Computing*: Through multi-threading RMI server can serve numerous clients
- *Distributed Computing Solutions*: Available from JDK 1.1, can communicate between all versions of JDKs

Weakness of Java RMI

- *Tied to Java System*: Purely Java-centric technology, does not have good support for legacy system written in C, C++, Ada etc.
- *Performance Issue* : Only good for large-grain computation
- *Security Restrictions & Complexities*: Threats during downloading objects from server, malicious client request, added security complexity in policy file.
- *Overhead*: Extra usage of *rmic* tool.

Case Study: Java RMI vs. Web service (SOAP/RESTFul)

Case Study: Java RMI vs. WS

Language Dependence:

✓ RMI service interface is in Java

✓WS service interface is platform independent

Mobile Behavior

✓ Server and client can change roles in RMI

✓ Not feasible in WS

Performance Issue:

✓RMI needs extra overhead for conversion from byte code to machine code

✓WS performs better for massive computation like fluid mechanics

CASE STUDY: JAVA RMI VS. WS

Ease of Use:
✓ RMI is easy to master for experienced programmers.
✓WS is a rich, extensive family of standards, hard to master

Maturity of Technology
✓ RMI is less matured
✓WS is more matured and already has many implementations running

CASE STUDY: JAVA RMI VS. WS

Results of case study:

- No one is better than other necessarily
- Applicability of one on another depends on
  - Purpose of the application
  - Experience of the designer and developer
  - Necessity of interoperability with non-java systems.

Conclusion

- Distributed Object Technology
- Object level abstraction
- Object version of Java RPC
- Java centric Technology
- Comparable to CORBA/SOAP/RESTFul
- Provides non-java support with the help IDL, IIOP and CORBA/SOAP/RESTFul
- Lightweight and Easy to use
- Object serialization
- Concurrent support for clients

Thank You !!! Questions Please?

References

[1].Advantages of java RMI http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138781.html

2 Java RMI architecture. http://www.cs.mun.ca/michael/java/jdk1.1-beta2-docs/guide/rmi/rmi-arch.doc.html

3 Introduction to java RMI http://www.javacoffeebreak.com/articles/javarmi/javarmi.html.

4 Java RMI: Remote method invocation. http://www1.cs.columbia.edu/dcc/nestor/presentations/java-rmi/java-rmi-handouts.pdf

5 Disadvantages of RMI. http://www.coderanch.com/t/180297/javadeveloper-SCJD/certification/RMI-Advantages-Disadvantages.

6 Background of java rmi. http://docs.oracle.com/javase/1.5.0/docs/guide/rmi/spec/rmi-intro2.html.

7 How RMI works. http://www.sce.carleton.ca/netmanage/simulator/rmi/RMIExplanation.htm