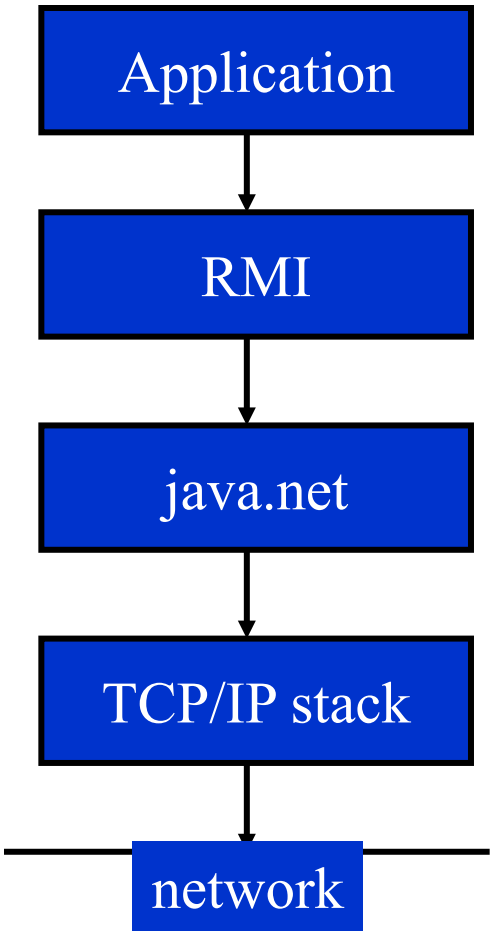
Lab 10
Remote Method Invocation

Java RMI Example

Java RMI



- Java RMI is a projection of RPC onto Java Object Model;
- Support seamless remote invocation on objects in different virtual machines;
- Support callbacks from server to applets
- Preserve the safety provided by the Java runtime environment;
- Less rich object services infrastructure.

Java RMI - Example

- The files needed for creating a Java RMI application are:
  - A **remote interface** defines the remote interface provided by the service. Usually, it is a single line statement specifies the service function (**HelloInterface.java)**. (An interface is the skeleton for a public class.)
  - A **remote object** implements the remote service. It contains a constructor and required functions. (**Hello.java)**
  - A **client** that invokes the remote method. (**HelloClient.java)**
  - The **server** offers the remote service, installs a security manager and contacts rmiregistry with an instance of the service under the name of the remote object. (**HelloServer.java**)

HelloInterface.java

- **import** java.rmi.*;

  public interface HelloInterface **extends** Remote {
  public String say(String msg) **throws** RemoteException;
  }

Hello.java

```
import java.rmi.*;
import java.rmi.server.*;

public class Hello extends UnicastRemoteObject implements
HelloInterface {
   private String message;
   public Hello(String msg) throws RemoteException {
      message = msg;
   }
   public String say(String m) throws RemoteException {
      return new StringBuffer(m).reverse().toString() + "\n" +
message;
   }
}
```

HelloClient.java

```
import java.rmi.*;

public class HelloClient {
  public static void main(String args[]) {
    String path = "//localhost/Hello";
    try {
      if(args.length < 1)
        System.out.println("usage: java HelloClient <host:port> <string>
\n");
      else
        path = "//" + args[0] + "/Hello";
      HelloInterface hello = (HelloInterface)Naming.lookup(path);
      for(int i = 1; i < args.length; ++i)
        System.out.println(hello.say(args[i]));
    } catch(Exception e) {
      System.out.println("Hello Client exeption: " + e);
    }
  }
}
```

HelloServer.java

```
import java.rmi.*;
import java.rmi.server.*;

public class HelloServer {
   public static void main(String args[]) {
       if(System.getSecurityManager()==null) {
          System.setSecurityManager(new SecurityManager());
       }
      try {
         Naming.rebind("Hello", new Hello("Hello, world!"));
         System.out.println("server is running...");
      } catch(Exception e) {
         System.out.println("Hello server failed: " + e.getMessage());
      }
   }
}
```

rmi.policy

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

Steps to Build a RMI application

- **Write the following Java code:**
  - **Interface -** HelloInterface.java
  - **Implementation class –** Hello.java
  - **Client** – HelloClient.java
  - **Server** – HelloServer.java

- **Compile the code**

  javac Hello.java HelloClient.java HelloInterface.java HelloServer.java

- **Generate Stub and Skeleton class files**

  rmic Hello

Steps to Build a RMI application

- **Make security policy**

  rmi.policy

- **Start the RMI registry** (in a separate window or in the background)

  rmiregistry &

- **Start the server** in one window or in the background with the security policy

  java -Djava.security.policy=rmi.policy HelloServer

- **Run the client** in another window

  java -Djava.security.policy=rmi.policy HelloClient 127.0.0.1:1099 testing

Test Hello Example

- **Write the following Java code:**

```
SeongHwanui-MacBook-Pro:Hello jihwankim$ ls
Hello.java              HelloInterface.java     rmi.policy
HelloClient.java        HelloServer.java
```

- **Compile the code**

```
SeongHwanui-MacBook-Pro:Hello jihwankim$ javac Hello.java  HelloClient.java
 HelloInterface.java  HelloServer.java
SeongHwanui-MacBook-Pro:Hello jihwankim$ ls
Hello.class             HelloClient.java        HelloServer.class
Hello.java              HelloInterface.class    HelloServer.java
HelloClient.class       HelloInterface.java     rmi.policy
```

- **Generate Stub and Skeleton class files**

```
SeongHwanui-MacBook-Pro:Hello jihwankim$ rmic Hello
SeongHwanui-MacBook-Pro:Hello jihwankim$ ls
Hello.class             HelloInterface.class    Hello_Stub.class
Hello.java              HelloInterface.java     rmi.policy
HelloClient.class       HelloServer.class
HelloClient.java        HelloServer.java
```

Test Hello Example

- **Start the RMI registry** (in a separate window or in the background)

```
SeongHwanui-MacBook-Pro:Hello jihwankim$ rmiregistry &
```

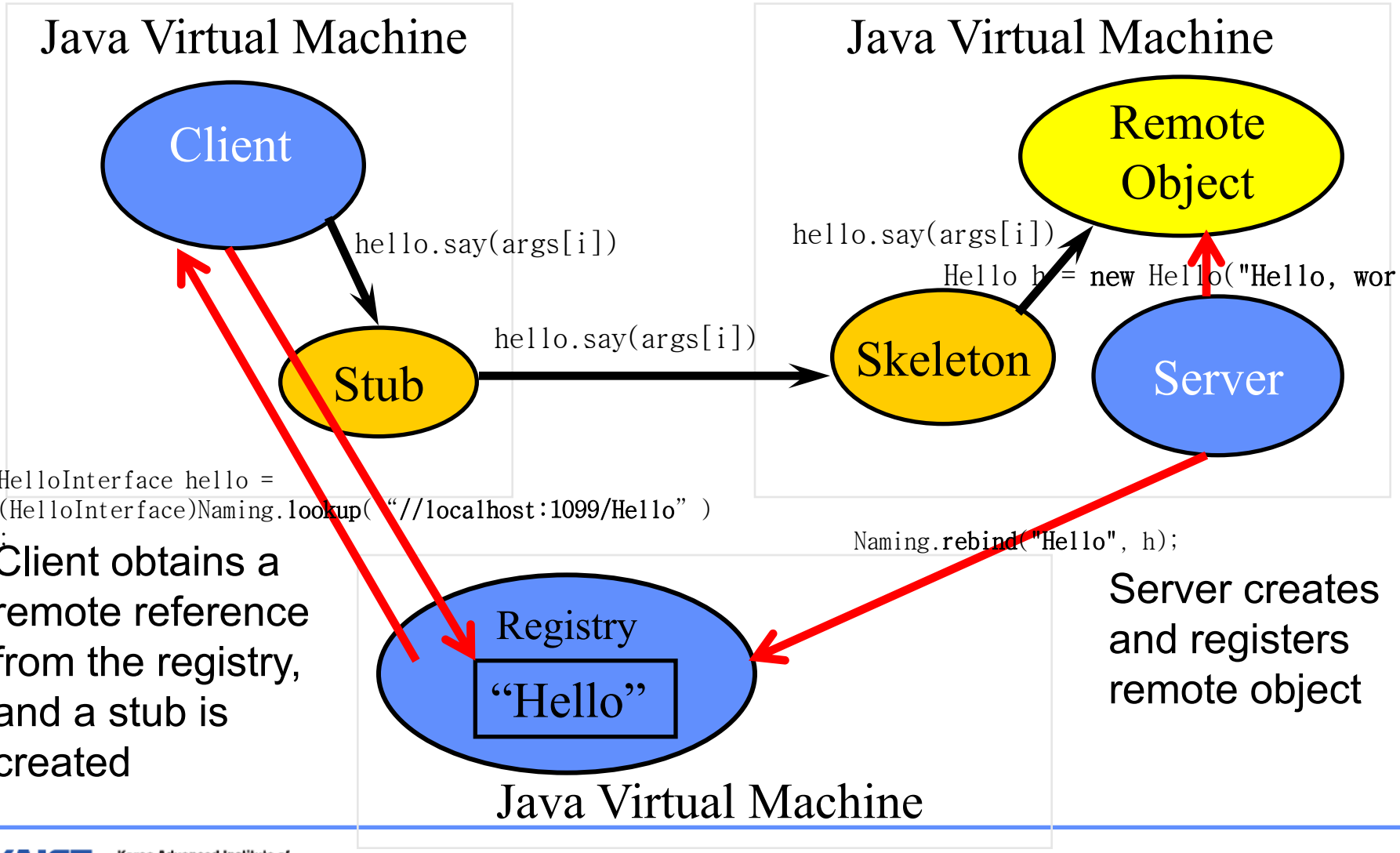
- **Start the server** in one window or in the background with the security policy

```
SeongHwanui-MacBook-Pro:Hello jihwankim$ java -Djava.security.policy=rmi.policy HelloServer
server is running...
```

- **Run the client** in another window

```
SeongHwanui-MacBook-Pro:Hello jihwankim$ java -Djava.security.policy=rmi.policy H
elloClient 127.0.0.1:1099 testing
gnitset
Hello, world!
```

Hello Example



Client obtains a remote reference from the registry, and a stub is created

Server creates and registers remote object

Procedure of Implementation

- Step 1: Make Interface of Remote Object for service
- Step 2: Make both Remote Object and Server program
- Step 3: Generate Stub/Skeleton
- Step 4: Make a Client program using Remote Object
- Step 5: Execute RMI Registry
- Step 6: Execute both Server program and Client program

Step 1: Make Interface of Remote Object

- Interface of Bank
  - Extends Remote
- Three abstract method
  - getBalance()
  - Deposit(int amount)
  - Withbraw(int amount)

Bank.java

- **import** java.rmi.*;

  public interface Bank **extends** Remote {
  public int getBalance() **throws** RemoteException;
  public int deposit(**int** amount) **throws** RemoteException;
  public int withdraw(**int** amount) **throws** RemoteException;
  }

Step 2: Make Remote Object and Server program

- Implementation of Bank Interface
  - All method of remote interface should be implemented
  - Use java.rmi.server.UnicastRemoteObject

BankImpl.java

```
import java.rmi.*;
import java.rmi.server.*;

public class BankImpl extends UnicastRemoteObject implements Bank {
  private int total;
  public BankImpl(int total) throws RemoteException {
    this.total = total;
  }

  public int getBalance() throws RemoteException {
    return total;
  }

  public int deposit(int amount) throws RemoteException {
    total += amount;
    return getBalance();
  }

  public int withdraw(int amount) throws RemoteException {
    total -= amount;
    return getBalance();
  }

  public static void main(String[] args) throws Exception {
    if (System.getSecurityManager() == null) {
      System.setSecurityManager(new SecurityManager());
    }
    BankImpl bankip = new BankImpl(10000);
    Naming.rebind("//localhost/BankIp", bankip);
    System.out.println("bank was rebinded with name BankIp");
  }
}
```

Method of Naming Class

- Public static void bind(String name, Remote obj);
  - Bind given name with object
- Public static String[] list (String name)
  - Return a name list from RMI Registry
- Public static Remote lookup(String name)
  - Return the reference of remote object which is bound to name
- Public void rebind(String name, Remote obj)
  - Ignore previous binding. Bind given name with object
- Public static void unbind(String name);
  - Release current binding

Compile java code

- **Write the following Java code:**

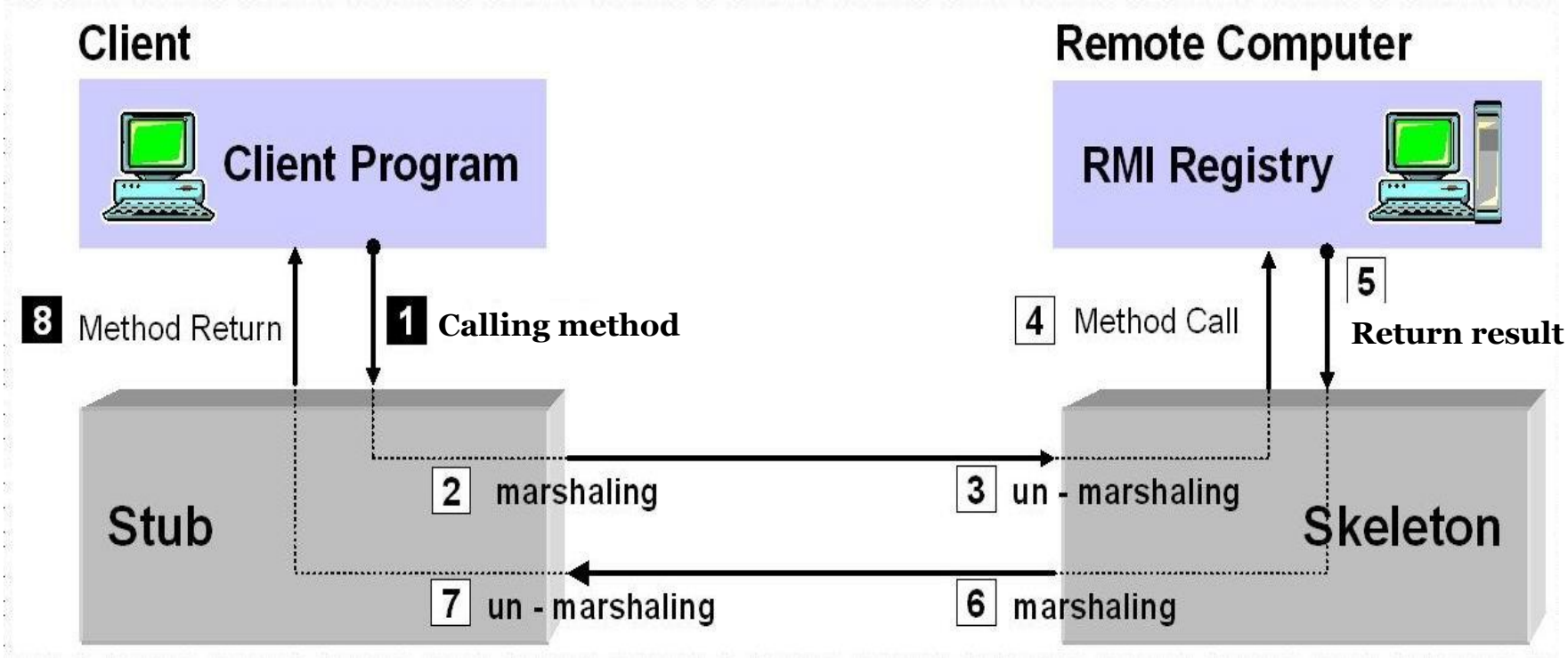
```
SeongHwanui-MacBook-Pro:Bank jihwankim$ ls
Bank.java       BankClient.java BankImpl.java   rmi.policy
```

- **Compile the code**

```
SeongHwanui-MacBook-Pro:Bank jihwankim$ javac Bank.java BankClient.java BankImpl.java
```

Step 3: Generation of Stub/Skeleton

- Stub and Skeleton



Step 3: Generation of Stub/Skeleton

- **Generate Stub and Skeleton class files**

```
SeongHwanui-MacBook-Pro:Bank jihwankim$ rmic BankImpl
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
```

```
[SeongHwanui-MacBook-Pro:Bank jihwankim$ ls
Bank.class              BankClient.java         BankImpl_Stub.class
Bank.java               BankImpl.class          rmi.policy
BankClient.class        BankImpl.java
```

Step 4: Client program

- BankClient
  - Get remote reference from RMI Registry in server
  - Calling methods by using remote reference

BankClient.java

- **import** java.rmi.*;

```
public class BankClient {
  public static void main(String[] args) throws Exception {
    int balance = 0;
    Bank bank = (Bank)Naming.lookup("//localhost/BankIp");
    System.out.println("Bank was given from server");
    balance = bank.getBalance();
    System.out.println("current balance: " + balance);
    balance = bank.deposit(1000);
    System.out.println("deposit 1000\ncurrent balance: " + balance);
    balance = bank.withdraw(5000);
    System.out.println("withdraw 5000\ncurrent balance: " +
balance);
  }
}
```

Step 5: Execute RMI Registry

- Two way of execution
  - Execution with default: rmiregistry
  - Execution with specific port number: rmiregistry port#

- **Start the RMI registry** (in a separate window or in the background)

```
SeongHwanui-MacBook-Pro:Bank jihwankim$ rmiregistry
```

Step 6: Execute Server and Client appl.

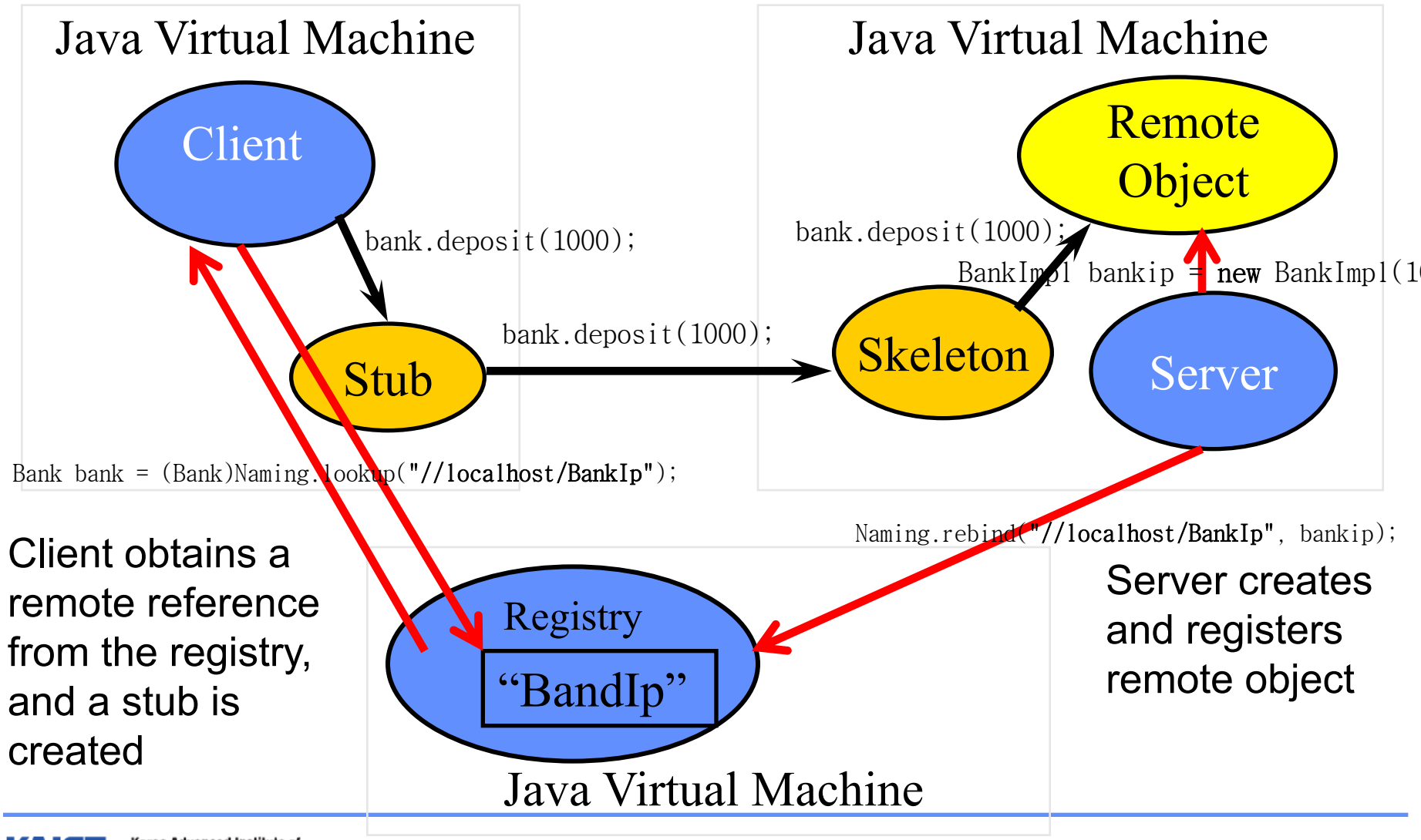
- **Start the server** in one window or in the background with the security policy

```
SeongHwanui-MacBook-Pro:Bank jihwankim$ java -Djava.security.policy=rmi.policy BankImpl
bank was rebinded with name BankIp
```

- **Run the client** in another window

```
SeongHwanui-MacBook-Pro:Bank jihwankim$ java -Djava.security.policy=rmi.policy
 BankClient
Bank was given from server
current balance: 10000
deposit 1000
current balance: 11000
withdraw 5000
current balance: 6000
```

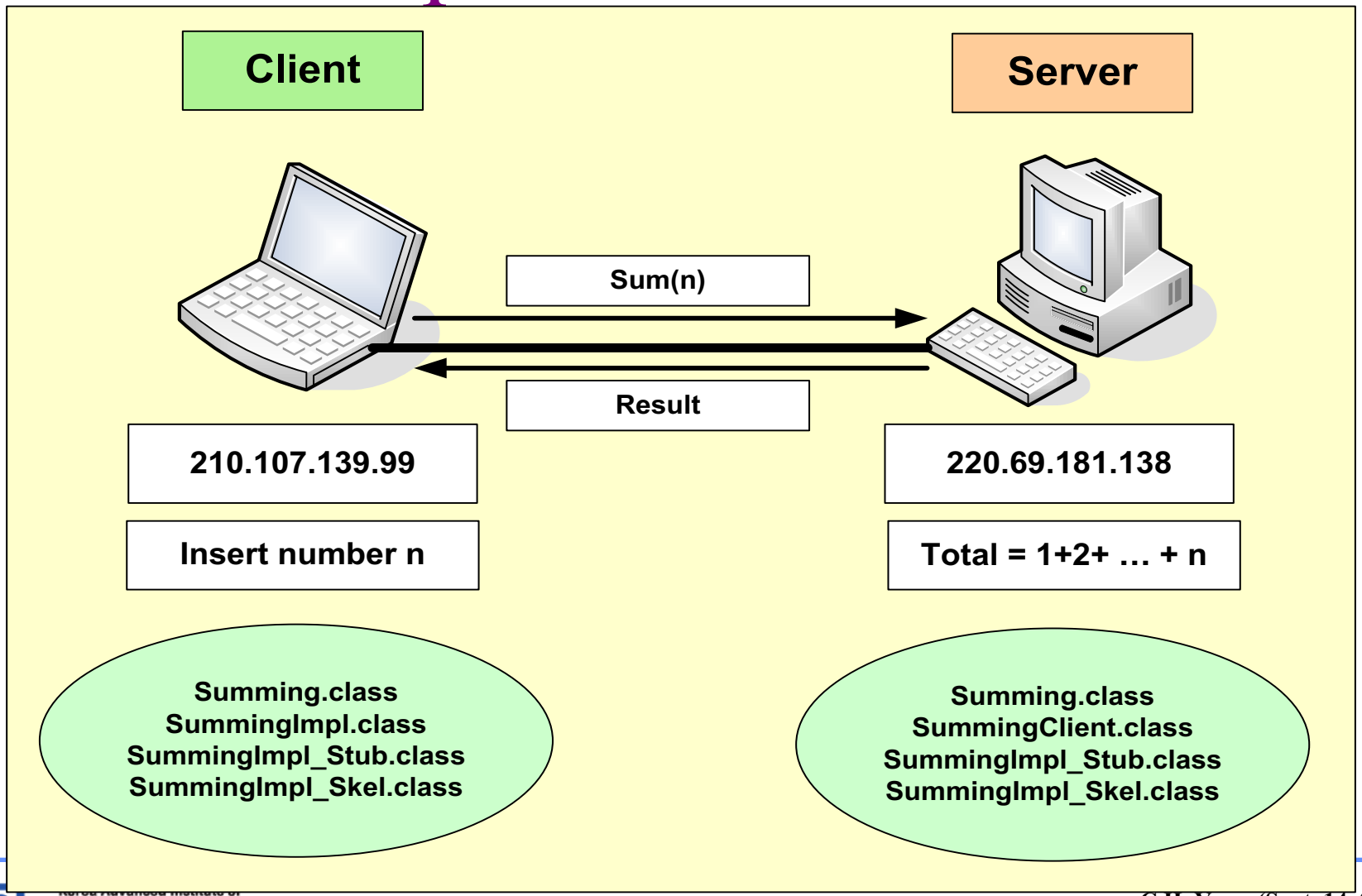
Bank Example



Client obtains a remote reference from the registry, and a stub is created

Server creates and registers remote object

Example 2: Sum method



Interface Implementation

```
import java.rmi.*;
public interface Summing extends Remote {
   int sum(int max) throws RemoteException;
}
```

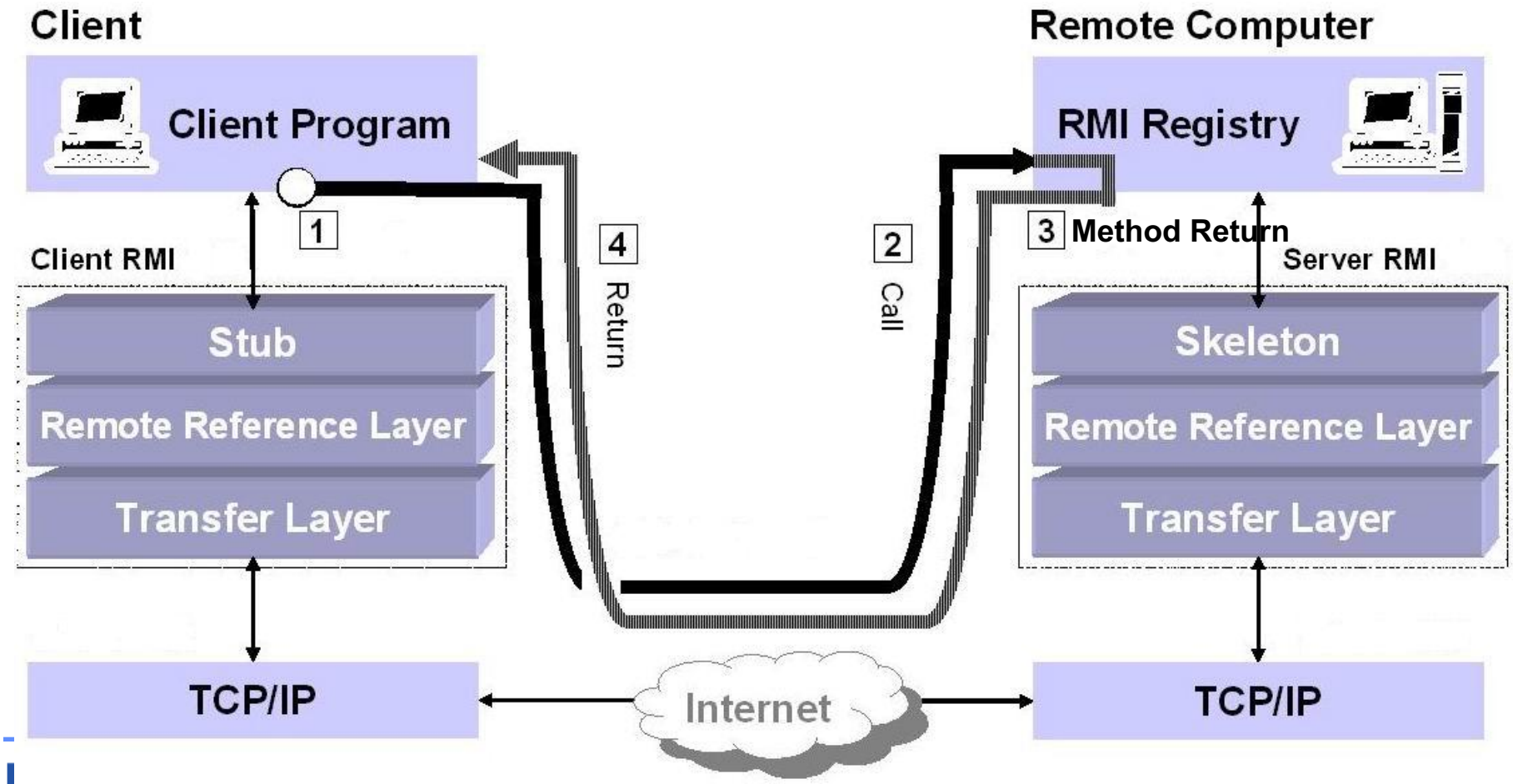
SummingImpl.java

```
import java.rmi.*;
import java.rmi.server.*;

public class SummingImpl extends UnicastRemoteObject implements Summing {
   public SummingImpl() throws RemoteException {
   }

   public int sum(int max) throws RemoteException {
      if(max <= 0) return 0;
      else return (max + sum(max - 1));
   }

   public static void main(String[] args) throws Exception {
      if (System.getSecurityManager() == null) {
         System.setSecurityManager(new SecurityManager());
      }

      SummingImpl s = new SummingImpl();
      Naming.rebind("SumServer", s);
      System.out.println("Summing was rebinded with name SumServer");
   }
}
```

SummingClient.java

```
import java.rmi.*;

public class SummingClient {
   public static void main(String args[]) {
      if(args.length < 2) {
         System.out.println("usage: java HelloClient <host:port> <string> \n");
         System.exit(1);
      }

      try {
         String serverURL = "rmi://" + args[0] + "/SumServer";
         Summing s = (Summing) Naming.lookup(serverURL);
         System.out.println("Your Input = " + args[1]);
         int max = Integer.parseInt(args[1]);
         System.out.println("The sum 1 to " + max + " is " + s.sum(max));
      } catch(Exception e) {
         System.out.println("Sum Client exeption: " + e);
      }
   }
}
```

Procedure of execution



Server Side

```
[SeongHwanui-MacBook-Pro:Summing jihwankim$ ls
Summing.class           SummingClient.java      SummingImpl_Stub.class
Summing.java            SummingImpl.class
SummingClient.class     SummingImpl.java
```

```
[SeongHwanui-MacBook-Pro:Summing jihwankim$ rmiregistry
```

```
SeongHwanui-MacBook-Pro:Summing jihwankim$ java -Djava.security.policy=rmi.polic
y SummingImpl
Summing was rebinded with name SumServer
```

Result

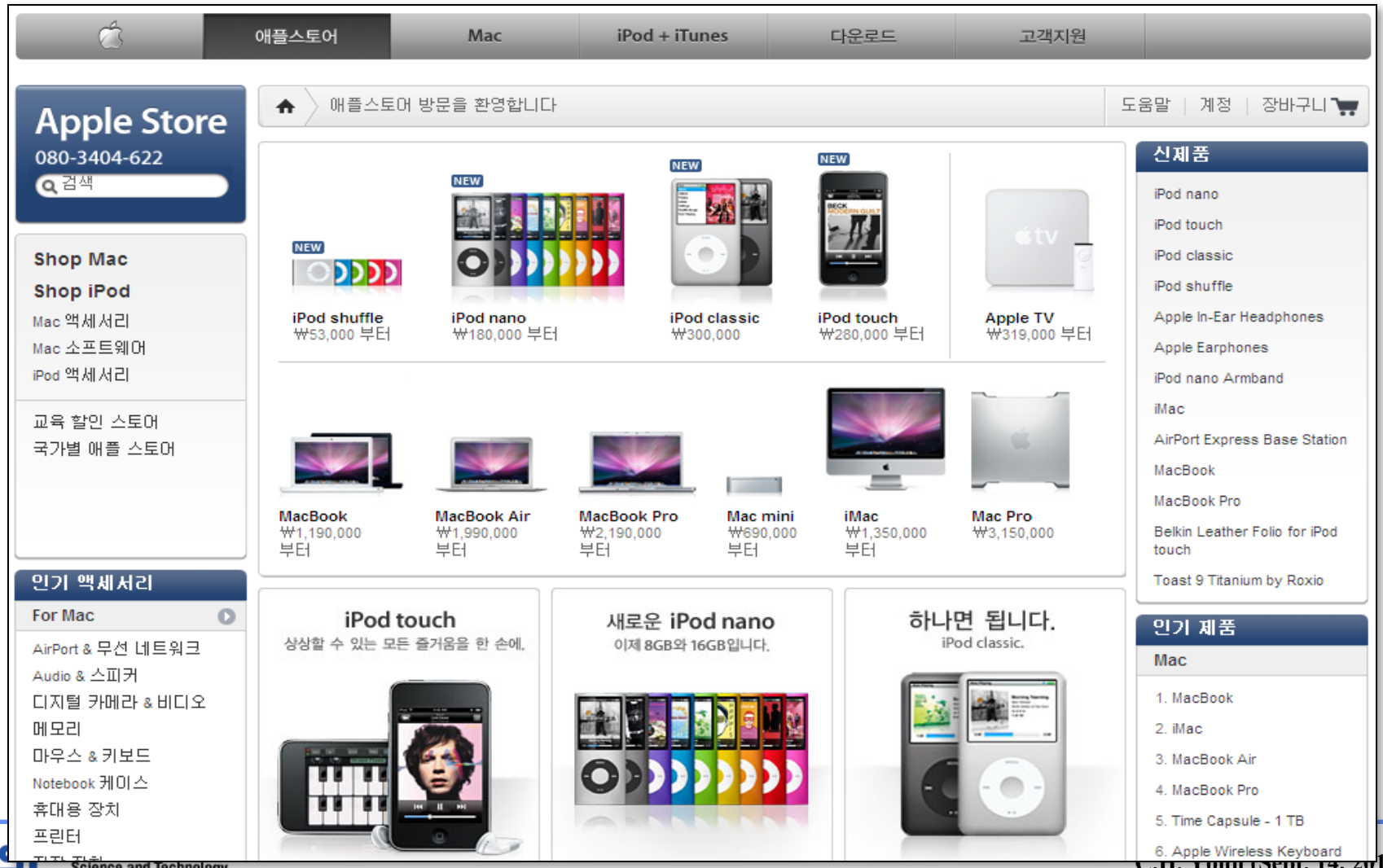
```
SeongHwanui-MacBook-Pro:Summing jihwankim$ java -Djava.security.policy=rmi.policy
 SummingClient 127.0.0.1:1099 100
Your Input = 100
The sum 1 to 100 is 5050
SeongHwanui-MacBook-Pro:Summing jihwankim$ java -Djava.security.policy=rmi.policy
 SummingClient 127.0.0.1:1099 1000
Your Input = 1000
The sum 1 to 1000 is 500500
```

Transaction Processing using Java RMI

(On-Line Transaction Processing System)

Real-life example

< apple online shopping mall for electronic device >



implementation environment

- Integrated Development Environment
  - IntelliJ
- Language
  - JAVA 1.8
- Communication method
  - Remote Method Invocation (RMI)
- Library
  - Swing : Seller and Buyer interface

Mini on-line Shopping Mall

- In online shopping mall practices, Transaction middleware is used for elevating efficiency of whole process , such as buy and sell, accounting, inventory and etc.

- **System components**
  - ❖ **Customer** :

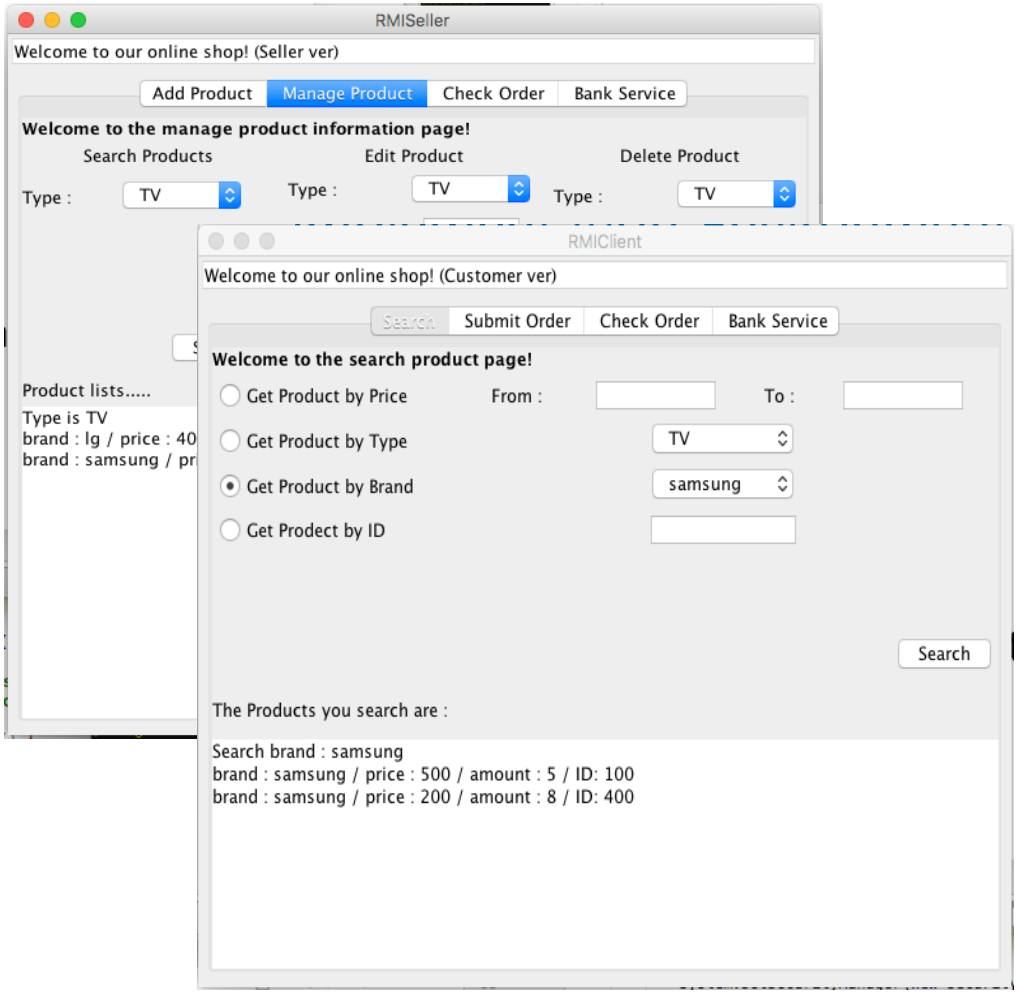
    GUI for customers to complete the whole transaction process.
  - ❖ **Seller**:

    GUI for sellers to complete the whole transaction process and management information of products.
  - ❖ **Shop server**

    information of products and customer information
  - ❖ **Bank server**

    Store user account and guarantee safe transaction

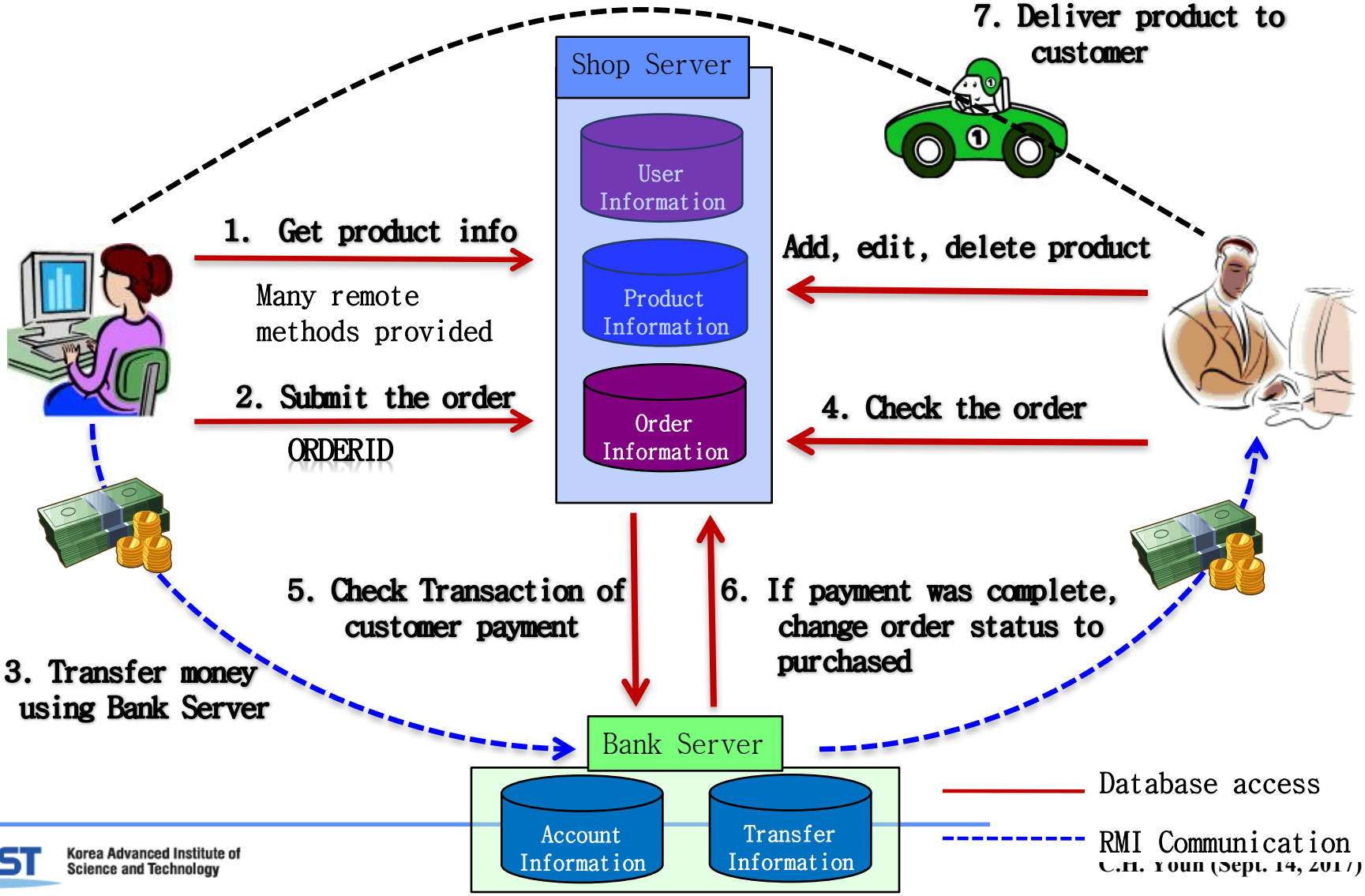
Transaction middleware using RMI



- User friendly GUI
  - Using JAVA Swing Library
  - Using IntelliJ IDE
- JAVA RMI for method invocation
  - Bank server
  - Shop server

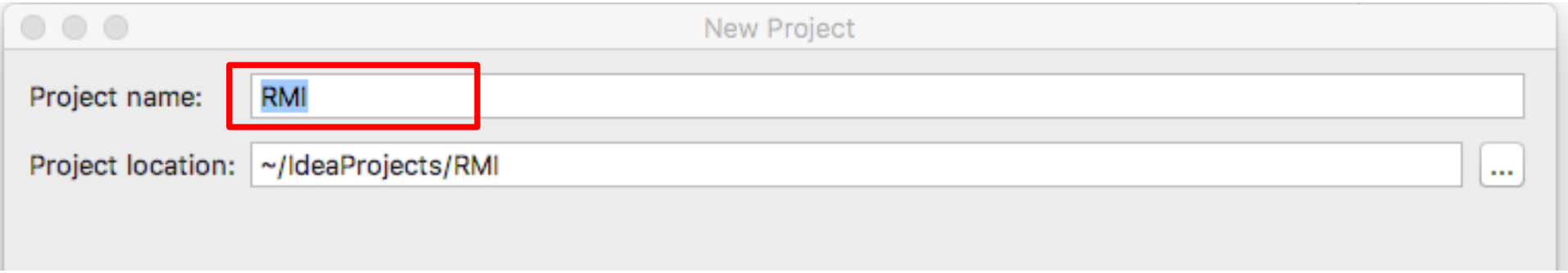
< Simple transaction middleware GUI>

Implement scenario

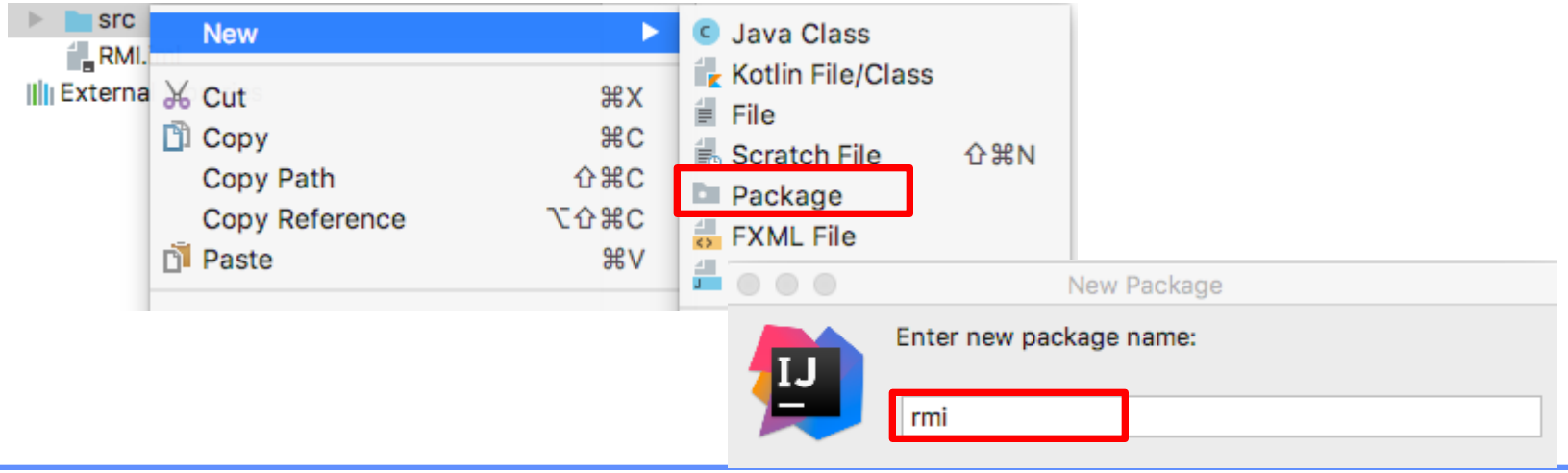


Make IntelliJ Project

1. Project name as RMI



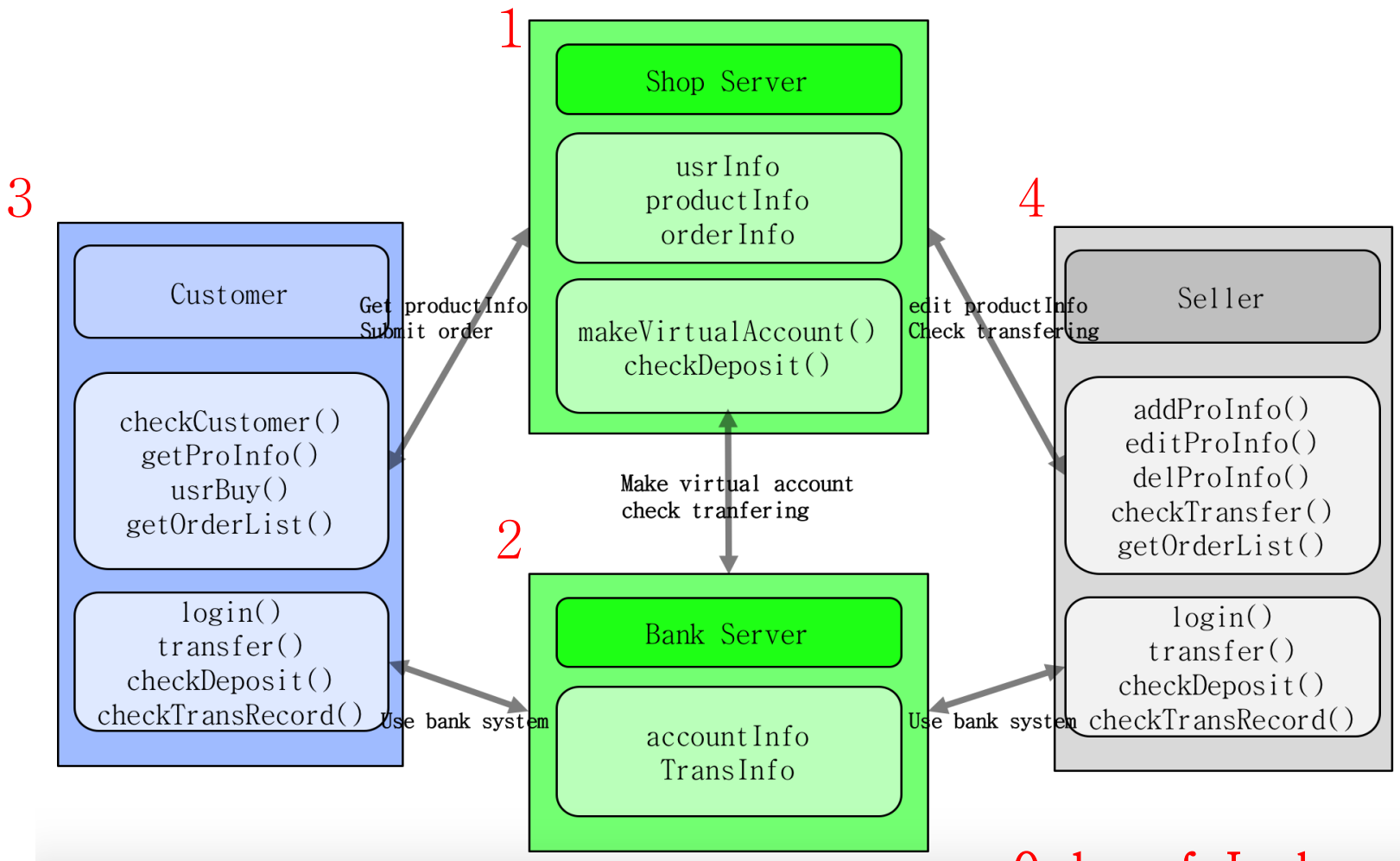
2. Make package name as rmi



Procedure of implement a RMI program

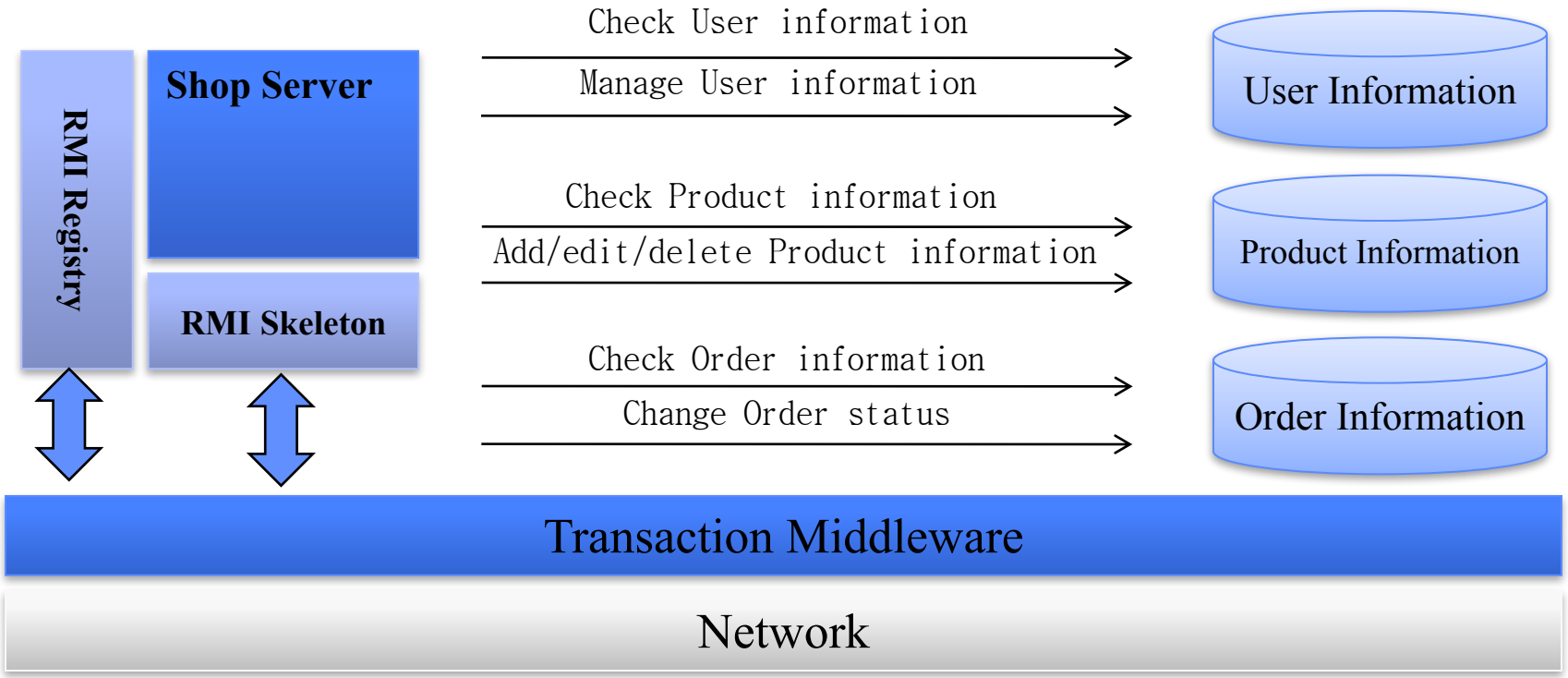
1. Make interface of remote object for service
2. Make both remote object and server program
3. Generate stub/skeleton
4. Make a client program using remote object
5. Execute RMI registry
6. Execute both server and client program

System architecture



Order of Implementation

Shop Server


RMI Registry
Shop Server
RMI Skeleton
Check User information
Manage User information
Check Product information
Add/edit/delete Product information
Check Order information
Change Order status
User Information
Product Information
Order Information
Transaction Middleware
Network

Shop Server

- To implement Shop Server, you need to create and define the information objects that will be stored on the Shop server and managed.
  - User Information
  - Product Information
  - Order Information

Shop Server(1) – User Information UsrInfo.java

```
package rmi;

    import java.util.ArrayList;

    class UsrInfo implements java.io.Serializable{

        String usrName;
        int usrPwd;
        ArrayList<OrderInfo> usrOrder;

        UsrInfo() { }

        UsrInfo(String usrName, int usrPwd) {
            this.usrName = usrName;
            this.usrPwd = usrPwd;
            this.usrOrder = new ArrayList<OrderInfo>();
        }
    }
```

Shop Server(2) – Product Information ProdcutInfo.java

```
package rmi;

public class ProductInfo implements java.io.Serializable {

    public String brand;
    public int price;
    public int amount;
    public String ID;

    ProductInfo() { }

    ProductInfo(String brand, int price, int amount, String ID) {
        this.brand = brand;
        this.price = price;
        this.amount = amount;
        this.ID = ID;
    }

    @Override
    public String toString() {
        return String.format("brand : %s / price : %d / amount : %d / ID: %s", brand,
price, amount, ID);
    }
}
```

Shop Server(3) – Order Information OrderInfo.java

```
package rmi;
        class OrderInfo implements java.io.Serializable{
            private static int orderCount = 1;
            int orderID;
            int virtualAccount;
            String usrAdr;
            String proID;
            int amount;
            int totalPrice;
            String purchased;

            OrderInfo() { }

            OrderInfo(int virtualAccount, String usrAdr, String proID, int amount, int totalPrice, String purchased) {
                this.orderID = orderCount++;
                this.virtualAccount = virtualAccount;
                this.usrAdr = usrAdr;
                this.proID = proID;
                this.amount = amount;
                this.totalPrice = totalPrice;
                this.purchased = purchased;
            }

            boolean notPurchasedOrder () {
                return this.purchased.equals("not purchased");
            } // check purchased or not to know what order was not purchased.

            @Override
            public String toString() {
                return String.format("OrderID : %d / VirtualAccount: %d / Address : %s / Product ID : %s / amount : %d /
        price : %d / purchased : %s", orderID, virtualAccount, usrAdr, proID, amount, totalPrice, purchased);
            }
        }
```

1. Make interface of remote object for service Shop.java

```
package rmi;

    import java.rmi.Remote;
    import java.rmi.RemoteException;
    import java.util.ArrayList;
    import java.util.HashMap;

    public interface Shop extends Remote {
       Boolean checkCustomer(String usrName, int usrPwd) throws RemoteException;
       ArrayList<ProductInfo> getProInfoByPrice(int from, int to) throws RemoteException;
       HashMap<String, ProductInfo> getProInfoByType(String type) throws RemoteException;
       ArrayList<ProductInfo> getProInfoByBrand(String brand) throws RemoteException;
       ProductInfo getProInfoByID(String ID) throws RemoteException;
       OrderInfo usrBuy(String usrName, String usrAdr, String type, String proID, int amount)
    throws RemoteException;
       UsrInfo getOrderList(String usrName) throws RemoteException;
       void addProInfo(String type, String brand, int price, int amount, String ID) throws
    RemoteException;
       void editProInfo(String type, String ID, int price, int amount) throws RemoteException;
       void delProInfo(String type, String ID) throws RemoteException;
       HashMap<String, UsrInfo> checkTransfer() throws RemoteException;
    }
```

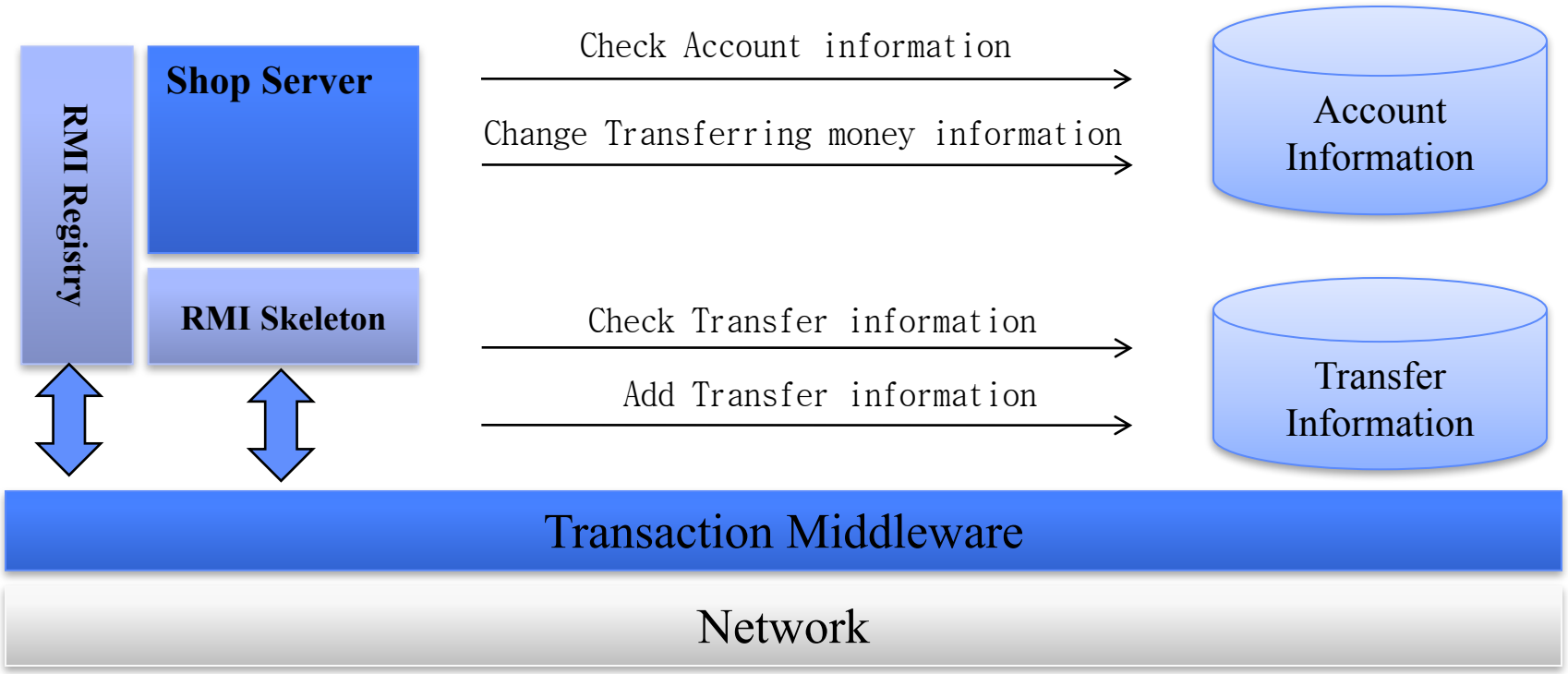
2. Make both remote object and server program ShopImpl.java

```
public class ShopImpl extends UnicastRemoteObject implements Shop {
        Bank b; // create Back object
        HashMap<String, HashMap<String, ProductInfo>> proList = new HashMap<>();
        HashMap<String, UsrInfo> usrList = new HashMap<>();
        HashMap<Integer, Account> orderAccount = new HashMap<>();

        public ShopImpl() throws RemoteException, NotBoundException, MalformedURLException {
            super();
            this.b = (Bank) Naming.lookup("rmi://localhost:1099/TestBank"); // ShopServer try rmi connection with Bank Object.

            proList.put("TV", new HashMap<>());
            proList.put("MP3", new HashMap<>());
            proList.put("COM", new HashMap<>());
            proList.put("PHONE", new HashMap<>());
            proList.get("TV").put("100", new ProductInfo("samsung", 500, 5, "100"));
            proList.get("TV").put("110", new ProductInfo("lg", 400, 3, "110"));
            usrList.put("123", new UsrInfo("123", 123));
            usrList.put("456", new UsrInfo("456", 456));
        }
                                                                    ……................

        @Override
        public HashMap<String, UsrInfo> checkTransfer() throws RemoteException {
            for (String key : usrList.keySet()) {
                for (int i = 0; i < usrList.get(key).usrOrder.size(); i++) {
                    if (usrList.get(key).usrOrder.get(i).notPurchasedOrder()) {
                        if (b.checkDeposit(usrList.get(key).usrOrder.get(i).virtualAccount) == usrList.get(key).usrOrder.get(i).totalPrice) { // call object BankImpl's
    checkDeposit method
                            b.transfer(orderAccount.get(usrList.get(key).usrOrder.get(i).orderID).accName,
    orderAccount.get(usrList.get(key).usrOrder.get(i).orderID).accPwd, 456, b.checkDeposit(usrList.get(key).usrOrder.get(i).virtualAccount));
                            // call object BankImpl's transfer method
                            usrList.get(key).usrOrder.get(i).purchased = "purchased";
                        }
                    }
                }
            }
            return usrList;
        }
}
```

2. Make both remote object and server program ShopServer.java

```
package rmi;

    import java.rmi.Naming;
    import java.lang.SecurityManager;

    public class ShopServer {
       public static void main(String[] args) throws Exception {
          if(System.getSecurityManager()==null) {
             System.setSecurityManager(new SecurityManager());
          }

          ShopImpl s = new ShopImpl();
          Naming.rebind("rmi://localhost:1099/TestShop", s);
          System.out.println("Binding ShopImpl object s name as TestShop");
       }
    }
```

Bank Server


RMI Registry
Shop Server
RMI Skeleton
Check Account information
Change Transferring money information
Account Information
Check Transfer information
Add Transfer information
Transfer Information
Transaction Middleware
Network

Bank Server

- To implement Bank Server, you need to create and define the information objects that will be stored on the Bank server and managed.
  - Account Information
  - Transfer Information

Bank Server(1) – Account Information AccountInfo.java

```
package rmi;

    import java.util.ArrayList;

    class Account implements java.io.Serializable{
       int accName;
       int accPwd;
       int deposit;
       ArrayList<TransInfo> transRecord;

       Account() { }

       Account(int accName, int accPwd, int deposit) {
          this.accName = accName;
          this.accPwd = accPwd;
          this.deposit = deposit;
          this.transRecord = new ArrayList<TransInfo>();
       }
    }
```

Bank Server(2) – Transfer Information

TransInfo.java

```
package rmi;

import java.sql.Timestamp;
import java.text.SimpleDateFormat;

class TransInfo implements java.io.Serializable{
   int sendAccName;
   int receiveAccName;
   int transAmount;
   Timestamp timestamp;


   TransInfo() { }

   TransInfo(int sendAccName, int receiveAccName, int transAmount) {
      this.sendAccName = sendAccName;
      this.receiveAccName = receiveAccName;
      this.transAmount = transAmount;
      this.timestamp = new Timestamp(System.currentTimeMillis());
   }

   @Override
   public String toString() {
      String time = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss").format(this.timestamp);
      return String.format("Send Account : %d / Receive Account: %d / Amount : %d / Time : %s",
sendAccName, receiveAccName, transAmount, time);
   }
}
```

1. Make interface of remote object for service Bank.java

```
package rmi;

    import java.rmi.Remote;
    import java.rmi.RemoteException;
    import java.util.ArrayList;

    public interface Bank extends Remote{
       public boolean login(int name, int pwd) throws RemoteException;
       public Account makeVirtualAccount(int accPwd) throws RemoteException;
       public boolean[] transfer(int name, int pwd, int obj, int num) throws
    RemoteException;
       public int checkDeposit(int name) throws RemoteException;
       public ArrayList<TransInfo> checkTransRecord(int name) throws
    RemoteException;
    }
```

2. Make both remote object and server program BankImpl.java

```
public class BankImpl extends UnicastRemoteObject implements Bank{
        HashMap<Integer, Account> accountList = new HashMap<>(); // hashmap accountList consist of <accName, Account> to find Account using
    accName.

        public BankImpl() throws RemoteException {
            super();
            accountList.put(123, new Account(123, 123, 5000));
            accountList.put(456, new Account(456, 456, 5000));
        }

        @Override
        public boolean login(int name, int pwd) throws RemoteException {
            /*
            proving login user is bank's customer using account ID(name), password(pwd).
             */
            boolean check = true;

            if (accountList.get(name) != null) {
                if (accountList.get(name).accPwd == pwd)
                    check = false;
            }

            return check;
        }

                                                                                    ……..
        @Override
        public ArrayList<TransInfo> checkTransRecord(int name) throws RemoteException {
            /*
            Check transfer records.
             */
            return accountList.get(name).transRecord;
        }
    }
```

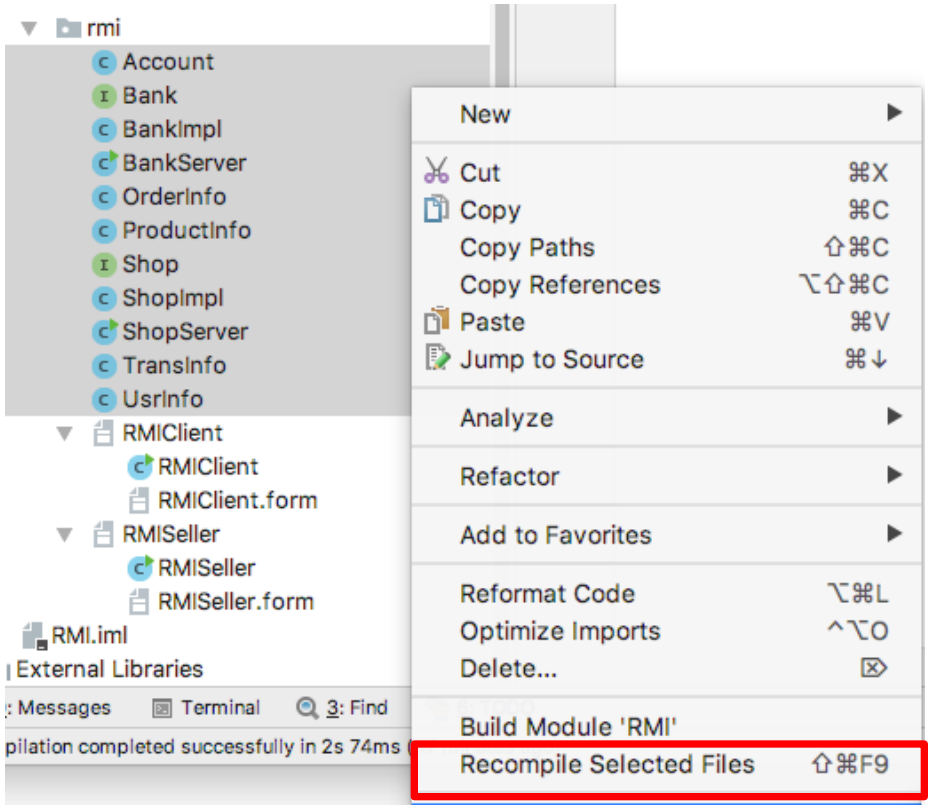
2. Make both remote object and server program BankServer.java

```
package rmi;

    import java.rmi.Naming;
    import java.lang.SecurityManager;

    public class BankServer {
       public static void main(String[] args) throws Exception {
          if(System.getSecurityManager()==null) {
             System.setSecurityManager(new SecurityManager());
          }

          BankImpl b = new BankImpl();
          Naming.rebind("rmi://localhost:1099/TestBank", b);
          System.out.println("Binding BankImpl object b name as TestBank");
       }
    }
```

3. Generate stub/skeleton

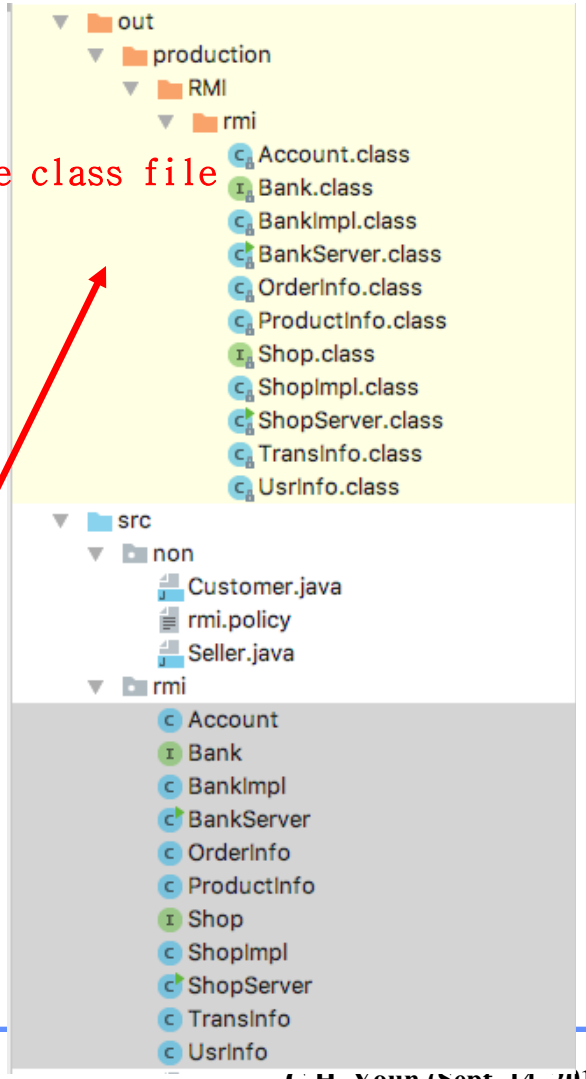
- **Compile the code**

1. Select all java file



2. Recompile Selected Files

3. Create class file



3. Generate stub/skeleton

- **Open terminal, and move to directory including "rmi" package**

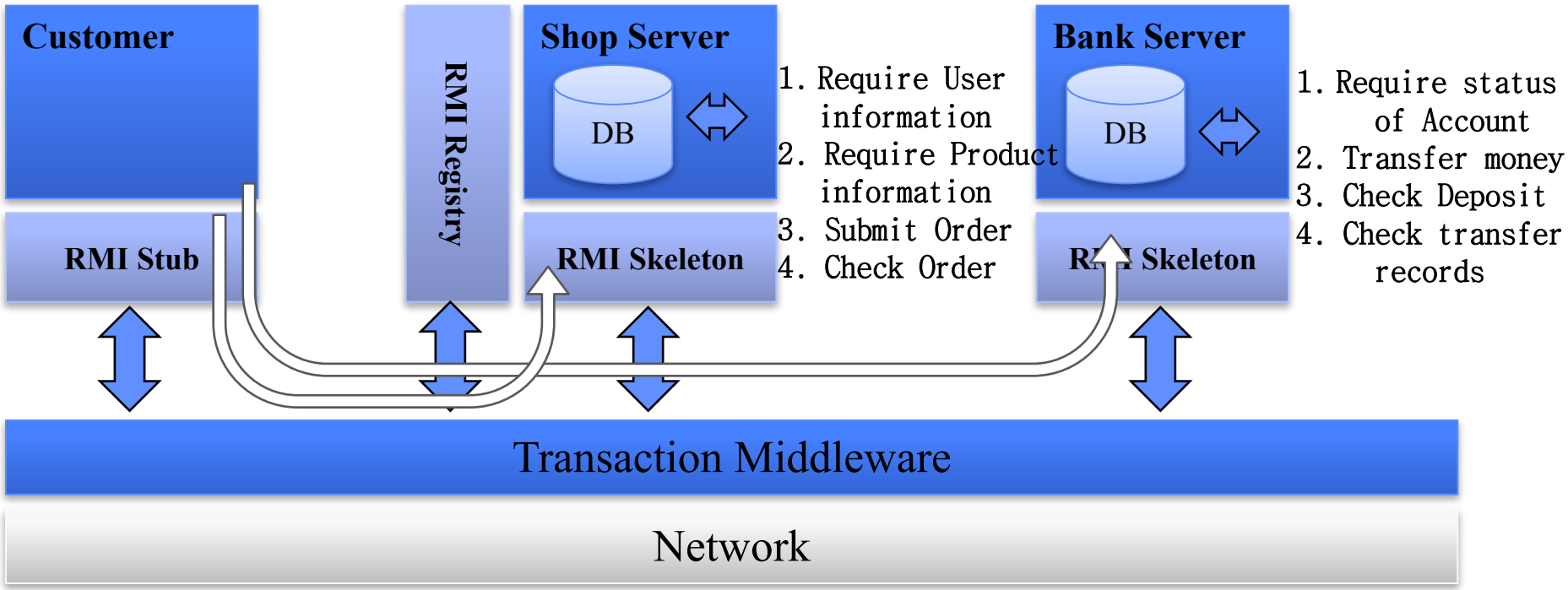
```
[SeongHwanui-MacBook-Pro:~ jihwankim$ cd IdeaProjects/RMI/out/production/RMI/
[SeongHwanui-MacBook-Pro:RMI jihwankim$ ls
rmi
```

- **Generate stub/skeleton**

```
[SeongHwanui-MacBook-Pro:RMI jihwankim$ rmic rmi.ShopImpl rmi.BankImpl
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
```

```
[SeongHwanui-MacBook-Pro:RMI jihwankim$ cd rmi/
[SeongHwanui-MacBook-Pro:rmi jihwankim$ ls
Account.class           OrderInfo.class         ShopServer.class
Bank.class              ProductInfo.class       TransInfo.class
BankImpl.class          Shop.class              UsrInfo.class
BankImpl_Stub.class     ShopImpl.class
BankServer.class        ShopImpl_Stub.class
```

Customer Client



4. Make a client program using remote object RMIClient.java

- **public** RMIClient() **throws** Exception {

```
/*
RMIClient's init method
 */
super("Online Shop (Only Customer)");
if(System.getSecurityManager()==null) {
    System.setSecurityManager(new SecurityManager());
}
Shop i = (Shop)Naming.lookup("rmi://localhost:1099/TestShop");
Bank b = (Bank) Naming.lookup("rmi://localhost:1099/TestBank");
```

4. Make a client program using remote object RMIClient.java

- **submitOrderButton**.addActionListener(**new** ActionListener() {
  @Override
  public void actionPerformed(ActionEvent e) {
  OrderInfo orderInfo = **new** OrderInfo();
  String usrName = **submitNameText**.getText();
  String type = **submitTypeBox**.getSelectedItem().toString();
  String ID = **submitIDText**.getText();
  String usrAdd = **submitAddText**.getText();
  boolean loginCheck = **true**;

  if (checkEmpty(usrName) && checkEmpty(**submitPwdText**.getText()) && checkEmpty(ID) && checkEmpty(usrAdd) && checkEmpty(**submitAmountText**.getText())) {
  int usrPwd = Integer.*parseInt*(**submitPwdText**.getText());
  int amount = Integer.*parseInt*(**submitAmountText**.getText());

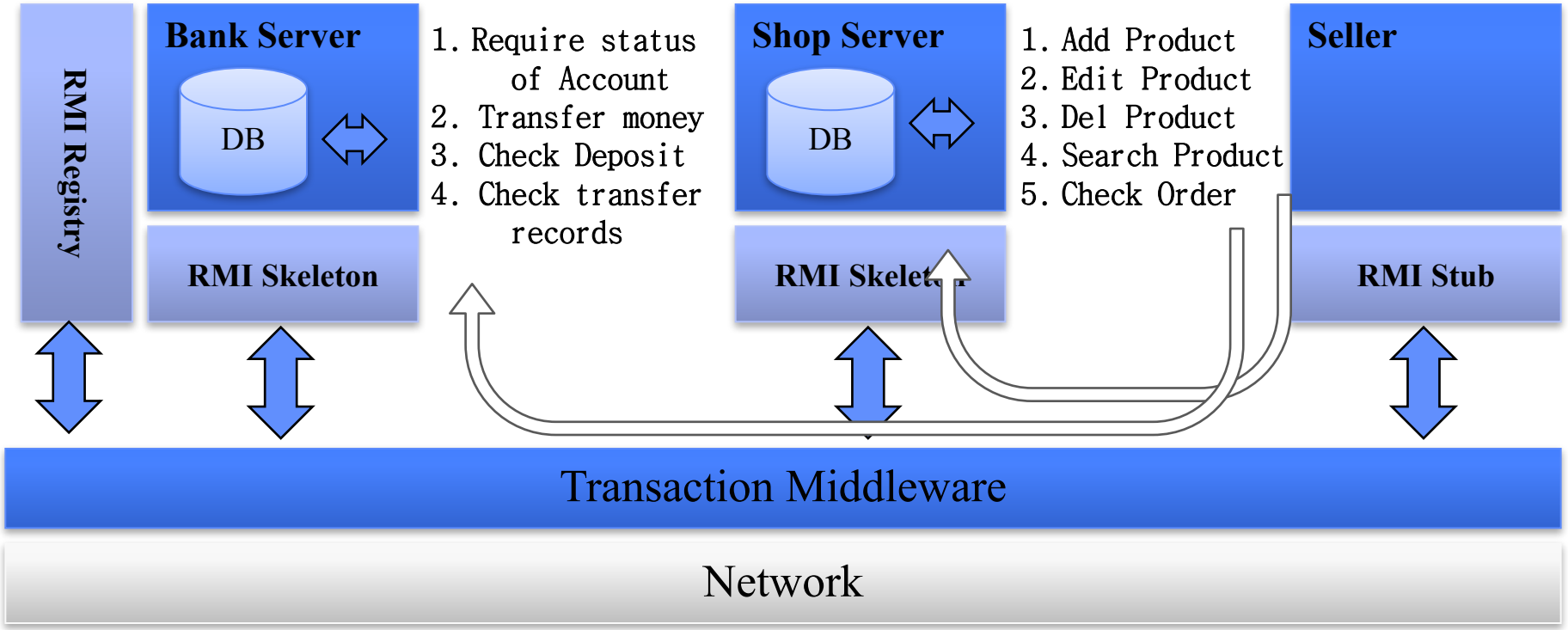
  try {
  loginCheck = i.checkCustomer(usrName, usrPwd); *//call object InfoImpl's checkCustomer method*
  } **catch** (RemoteException e1) {
  e1.printStackTrace();
  }

  Login to Shop Server

  if (!loginCheck) {
  try {
  orderInfo = i.usrBuy(usrName, usrAdd, type, ID, amount); *//call object InfoImpl's usrBuy method*
  } **catch** (RemoteException e1) {
  e1.printStackTrace();
  }

  Submit order to Shop Server

Seller Client



4. Make a client program using remote object RMISeller.java

- **public** RMISeller() **throws** Exception {

```
/*
RMISeller's init method
 */
super("Online Shop (Seller ver)");
if(System.getSecurityManager()==null) {
    System.setSecurityManager(new SecurityManager());
}

Shop i = (Shop)Naming.lookup("rmi://localhost:1099/TestShop");
Bank b = (Bank) Naming.lookup("rmi://localhost:1099/TestBank");
```

4. Make a client program using remote object RMISeller.java

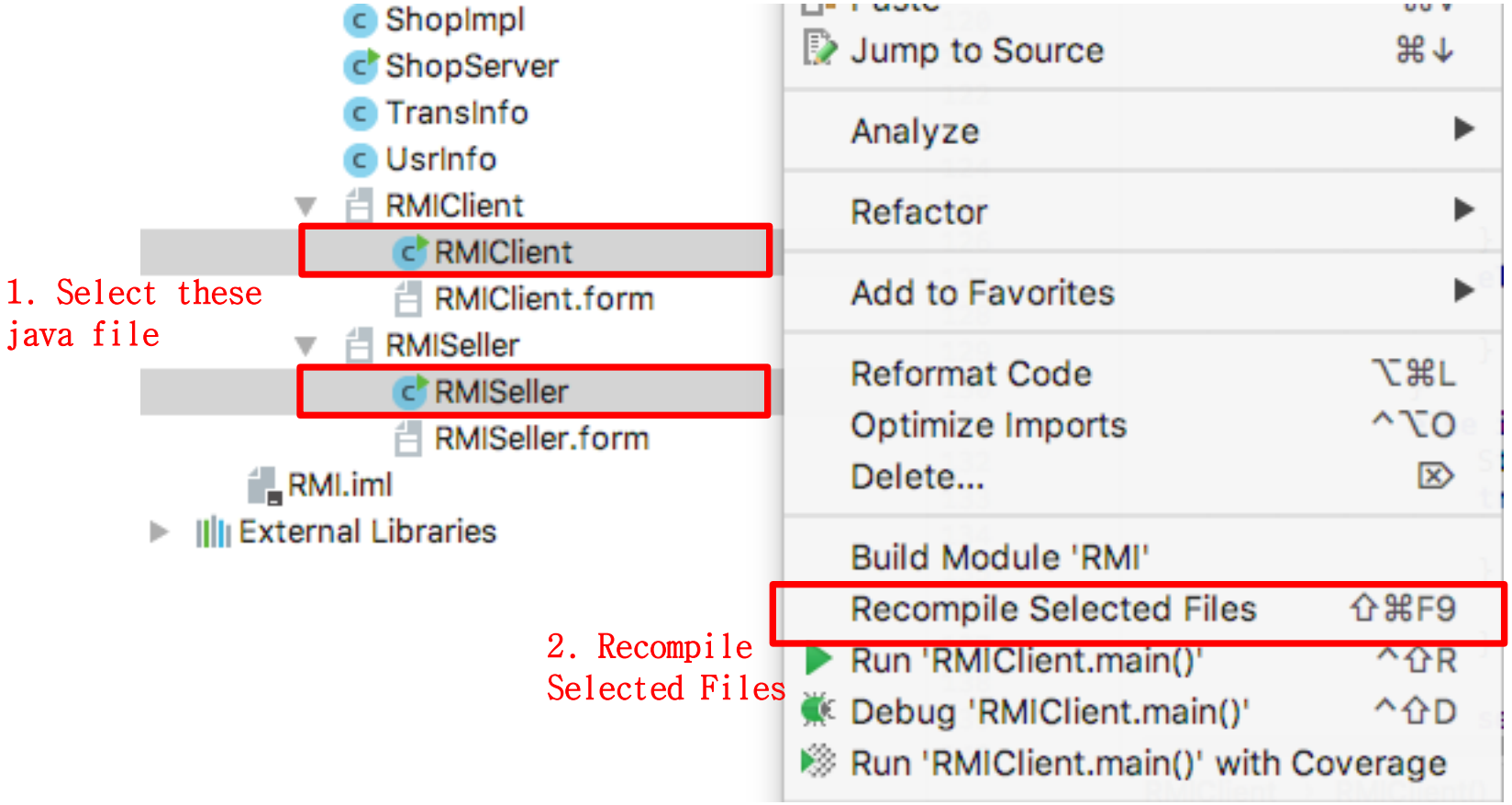
- **bankSendButton**.addActionListener(**new** ActionListener() {

```
    @Override
    public void actionPerformed(ActionEvent e) {
        boolean loginCheck = true;
        boolean transferCheck = true;
        boolean objAccCheck = false;
        boolean[] checks = new boolean[2];

        if (checkEmpty(bankAccNameText.getText()) && checkEmpty(bankAccPwdText.getText()) && checkEmpty(bankTransObjText.getText())
&& checkEmpty(bankTransAmountText.getText())) {
            int accName = Integer.parseInt(bankAccNameText.getText());
            int accPwd = Integer.parseInt(bankAccPwdText.getText());
            int objAcc = Integer.parseInt(bankTransObjText.getText());
            int amount = Integer.parseInt(bankTransAmountText.getText());

            try {
                loginCheck = b.login(accName, accPwd); //call object BankImpl's login method
            } catch (RemoteException e1) {
                e1.printStackTrace();
            }

            if (!loginCheck) {
                try {
                    checks = b.transfer(accName, accPwd, objAcc, amount); //call object BankImpl's transfer method
                } catch (RemoteException e1) {
                    e1.printStackTrace();
                }
                transferCheck = checks[0];
                objAccCheck = checks[1];
```

Login to Bank Server

Transfer money using bank service

4. Compile both Customer and Seller Client



5. Execute RMI registry

- **Make RMI security policy file**

```
SeongHwanui-MacBook-Pro:RMI jihwankim$ ls
com                 rmi                 rmi.policy
```

- **Execute RMI registry**

```
SeongHwanui-MacBook-Pro:RMI jihwankim$ rmiregistry 1099
```

6. Execute both server and client program

- **Run BankServer**

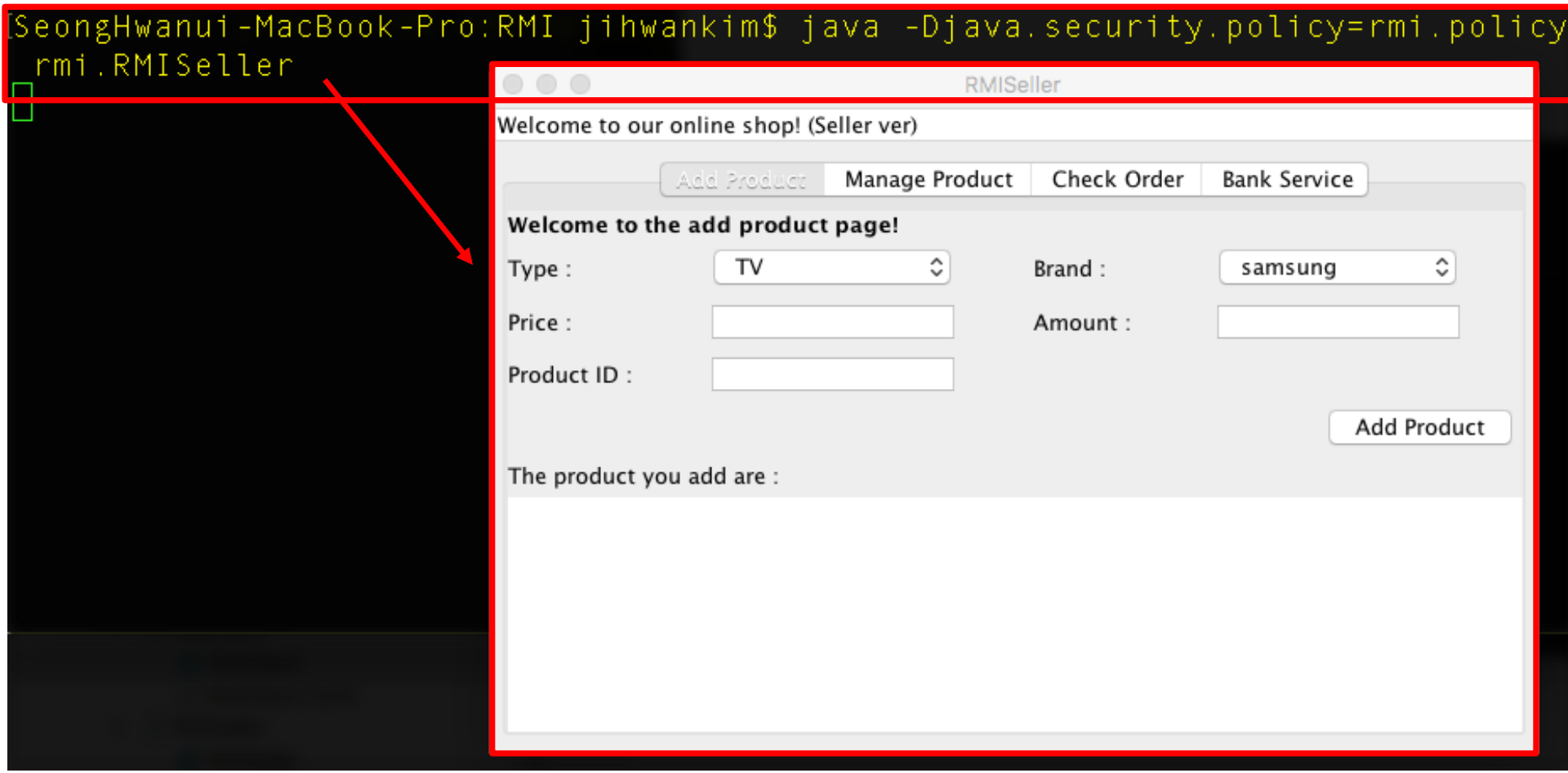
```
SeongHwanui-MacBook-Pro:RMI jihwankim$ java -Djava.security.policy=rmi.policy
 rmi.BankServer
Binding BankImpl object b name as TestBank
```

- **Run ShopServer**

```
SeongHwanui-MacBook-Pro:RMI jihwankim$ java -Djava.security.policy=rmi.policy
 rmi.ShopServer
Binding ShopImpl object s name as Test Shop
```

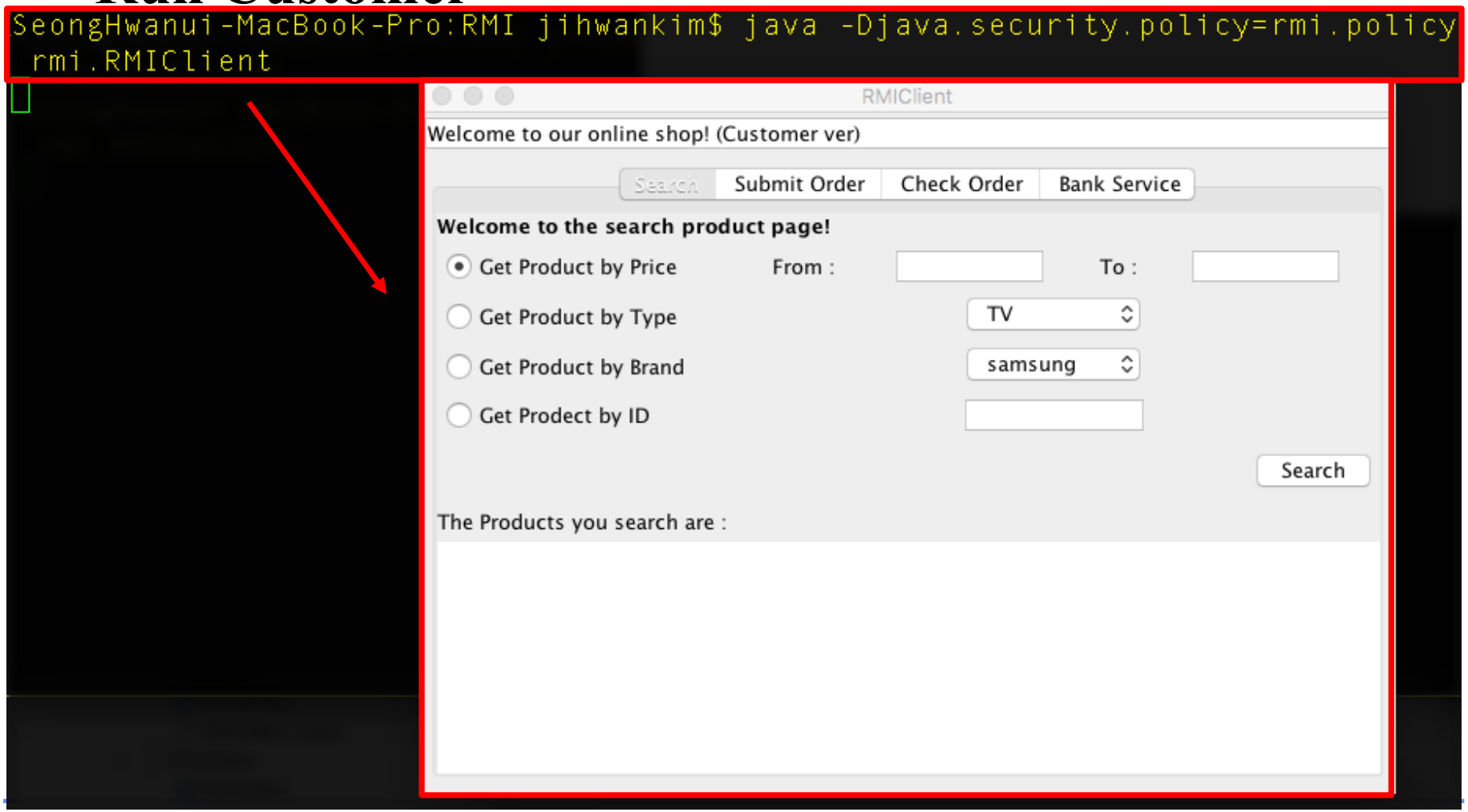
6. Execute both server and client program

- **Run Seller**



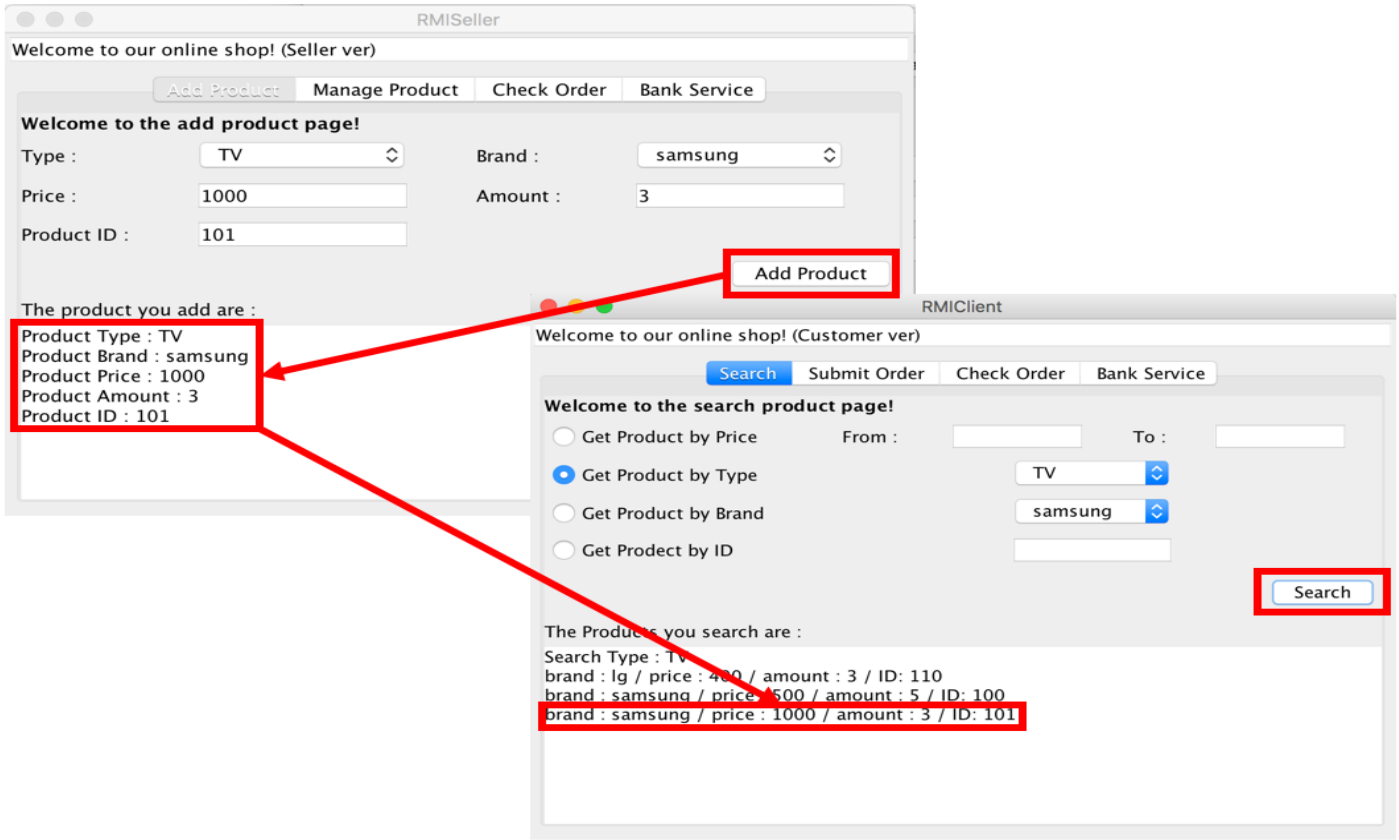
6. Execute both server and client program

- **Run Customer**



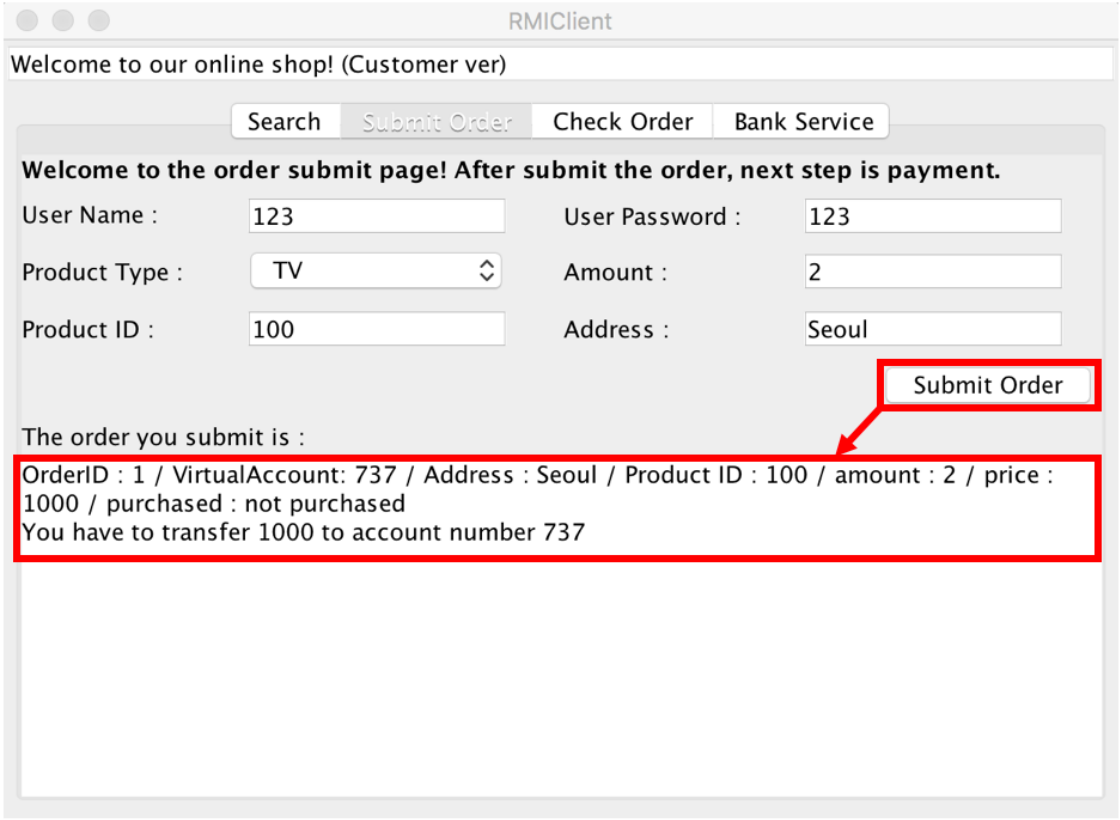
Demo on-line shopping mall

- Seller → method addProInfo(InfoImpl) / Customer → method getProInfoByType(InfoImpl)



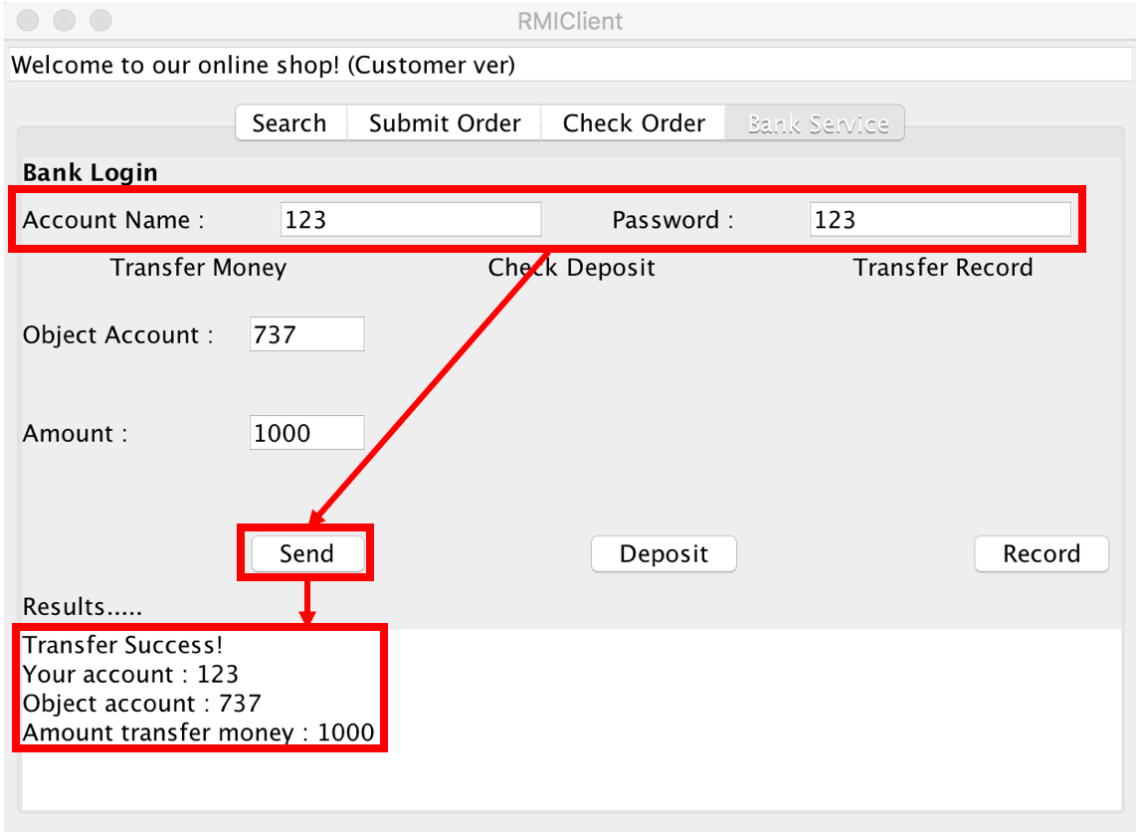
Demo on-line shopping mall

- Customer → method usrBuy(InfoImpl) getOrderList(InfoImpl) makeVirtualAccount(BankImpl)



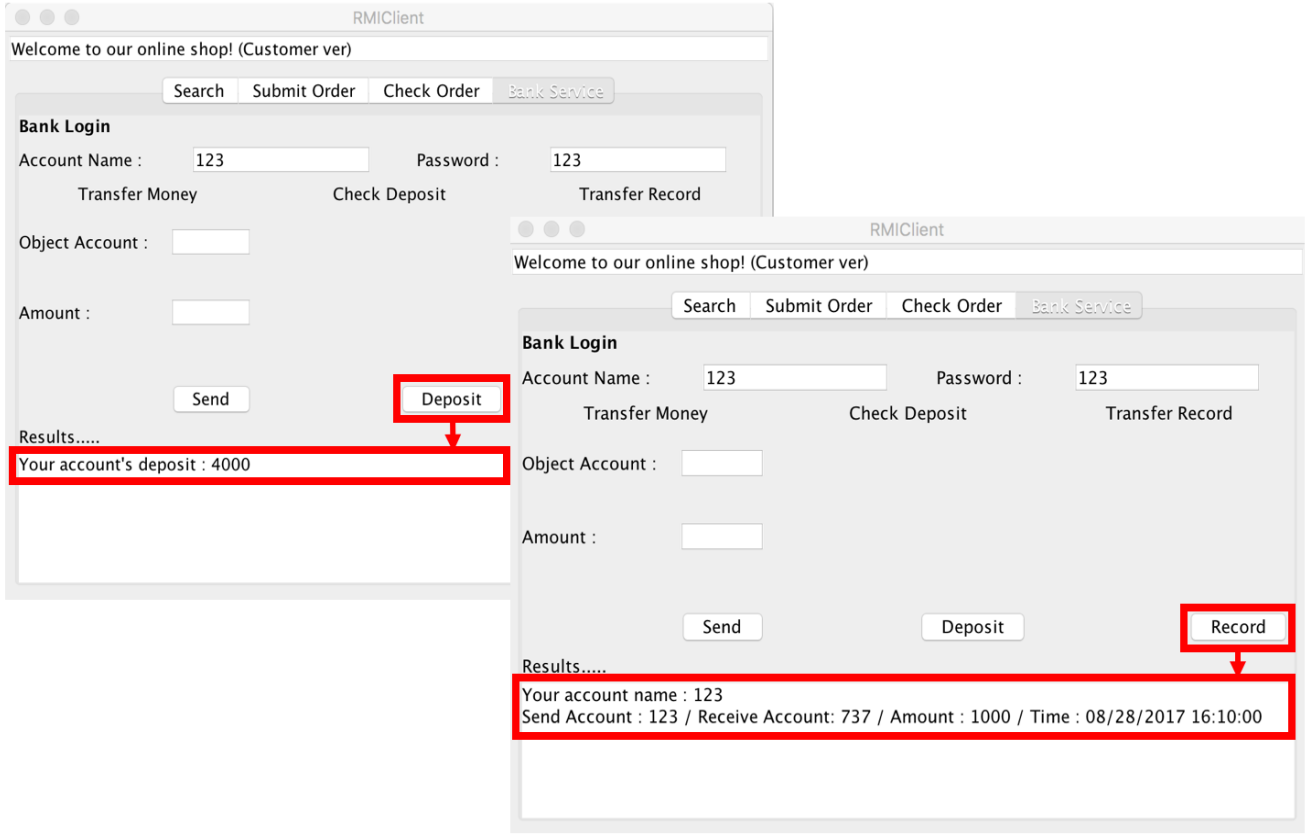
Demo on-line shopping mall

- Customer → method login(BankImpl) transfer(BankImpl)



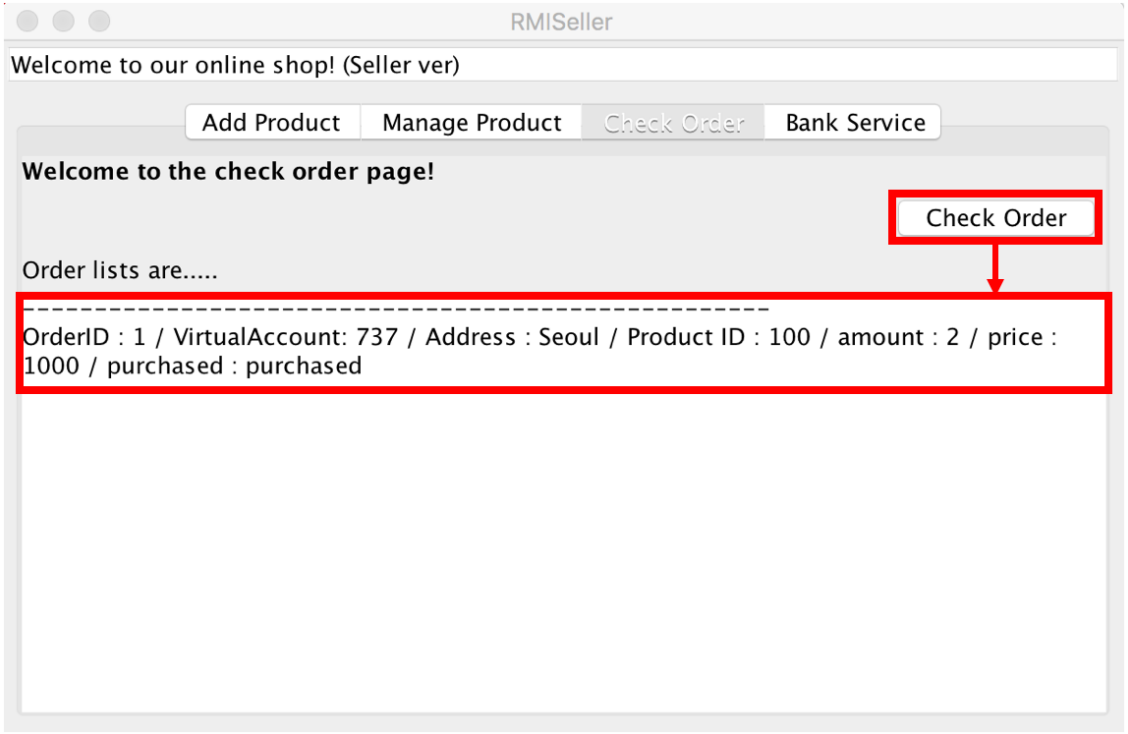
Demo on-line shopping mall

- Customer → method checkDeposit(BankImpl) checkTransRecord(BankImpl)



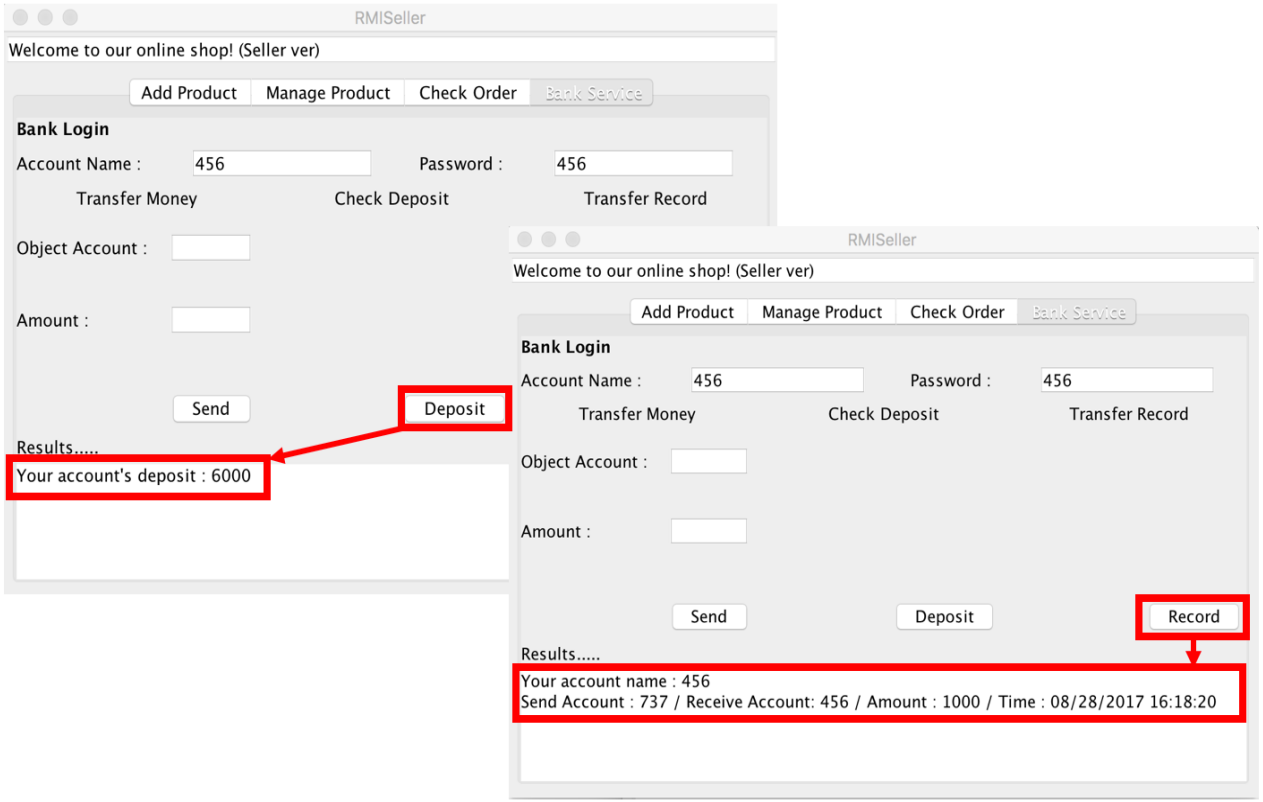
Demo on-line shopping mall

- Seller → method checkOrder(InfoImpl)



Demo on-line shopping mall

- Seller → method login(BankImpl) checkDeposit(BankImpl) checkTransRecord(BankImpl)



Lab 10. Building OOM

- Comparison with Message-oriented Middleware
  - Explain the structural differences between OOM and MOM
  - Compare the two middleware in terms of performance, scalability, ease of development, complexity and etc.
- Build a Mini on-line Shopping Mall program on your PC
  - Illustrate how client, shop, bank and seller works, step by step.
  - Describe the operating procedures and object lifecycle of each entity from the perspective of server, client, and rmiregistry.