

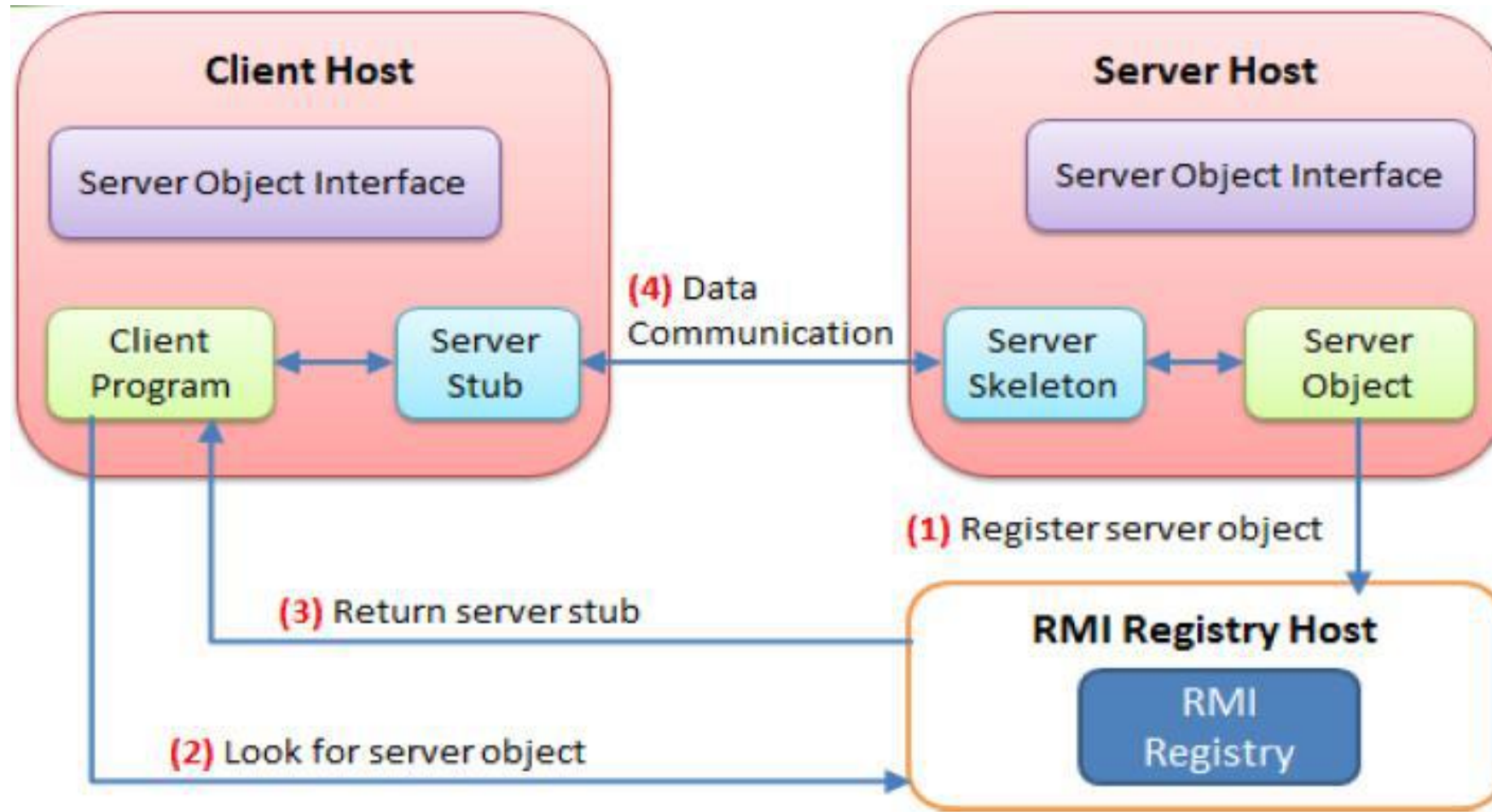
**JAVA RMI**

**TUTORIAL 2**

# Overview

- Basic Concepts
- Installation Process
- Hands on experience (Coding 😊)

# Basic Concepts



# Basic Concepts

## JAVA RMI:

RMI allows a Java object that executes on one machine to invoke a method of a Java object that executes on another machine.

## CLIENT:

CLIENT INVOKES THE METHOD ON REMOTE OBJECT

## SERVER:

IT OWNS THE REMOTE OBJECT

## REGISTRY:

NAMING SERVER THAT RELATES OBJECTS WITH UNIQUE NAMES

# Concepts

## Server Object Interface:

An interface defines the method for the server object

## Server Object:

An instance of the server object interface

## Server Stub:

An object resides on the client host and serves as a representative of the remote server object

# Concepts

## Server Skeleton:

An object that resides on the server host. It communicates with the stub and the actual server object

## RMI Registry:

It is service to register remote objects and to provide naming services for locating objects

## Client Program:

A program that invokes the methods in the remote server object

# Concepts

## Implementation steps:

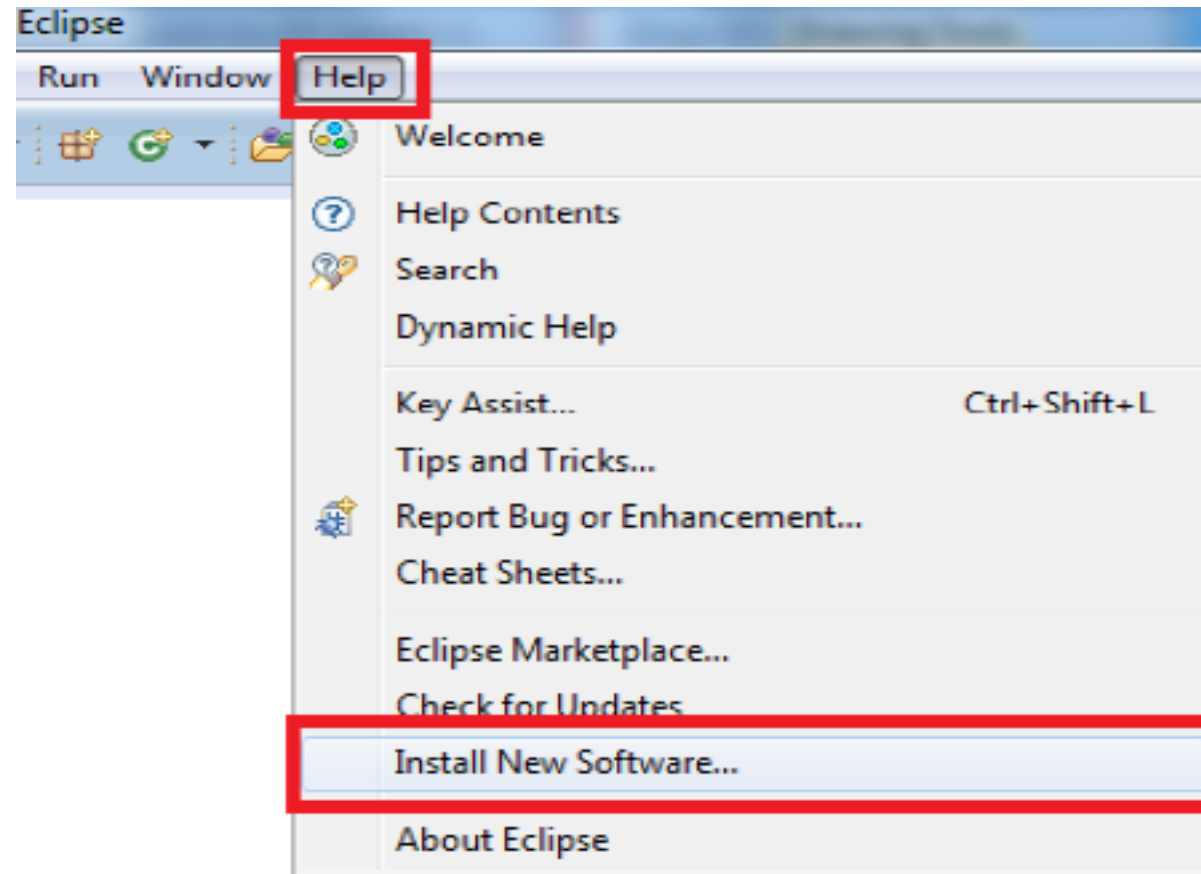
1. Define the interface
2. Implement Server
3. Implement Client

## Execution Steps:

1. Run RMI Registry
2. Run Server
3. Run Client

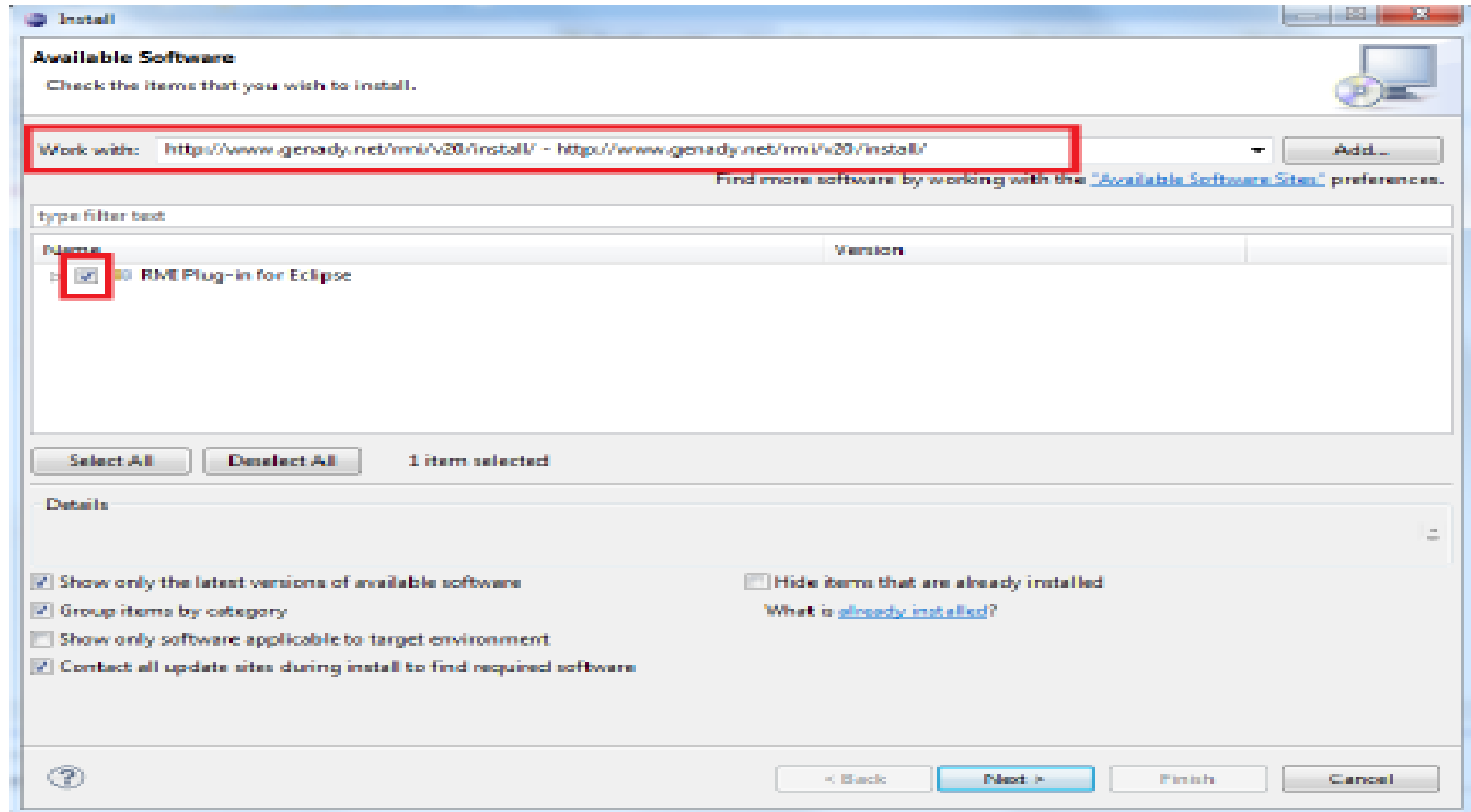
# INSTALLING RMI PLUGIN

1. Open Eclipse
2. Select Help -> Install New Software



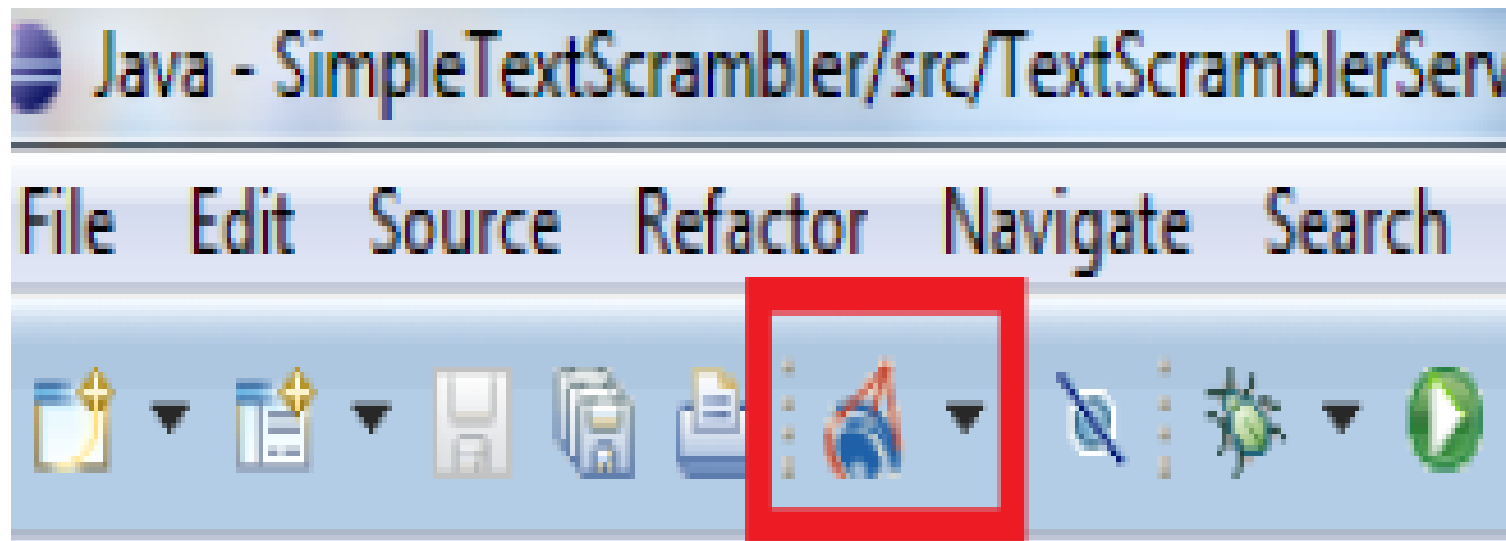


- Enter: <http://www.genady.net/rmi/v20/install/>
- Check RMI plugin for Eclipse



1. Accept the Terms of Services and continue.
2. You may have to restart Eclipse.
3. Troubleshooting:
4. You need Java SDK (1.6+) installed. The JRE does not provide Tools.jar which is essential to Java RMI
5. Your CLASSPATH variable must remain undefined.

1. If successful, you will see the following when you re-open Eclipse:



2. This will enable you to start a local registry manually.

# RMI EXERCISE

Now, let's start coding


# Exercise : Addition RMI

- Objective :
  - The objective of today's tutorial is to successfully run RMI distributed system which gives addition of two numbers to the client.
- Note that this tutorial is designed in a very simple fashion. For your assignment, you are expected to structure your code (put files in relevant packages) and modify it as per your need.

# Steps :

1. Create the remote interface
2. Implementation of Remote Interface
3. Create server
4. Create client
5. Run RMI plugin
6. Run server and then client

# 1. Create the Remote Interface

 AddInterface.java ✕

```
1
2 import java.rmi.*;
3
4 public interface AddInterface extends Remote{
5
6     public int add(int x, int y) throws RemoteException;
7
8 }
```

- Extend Remote Class belongs to (java.RMI) package in your Interface
- Implement the Interface in the class which you want with remote functionality
- The Remote interface serves to identify interfaces whose methods may be invoked from a non-local virtual machine. Any object that is a remote object must directly or indirectly implement this interface. Only those methods specified in a “remote interface”, an interface that extends java.rmi.Remote are available remotely.
- Your methods now potential throw RemoteException. You must declared them explicitly.

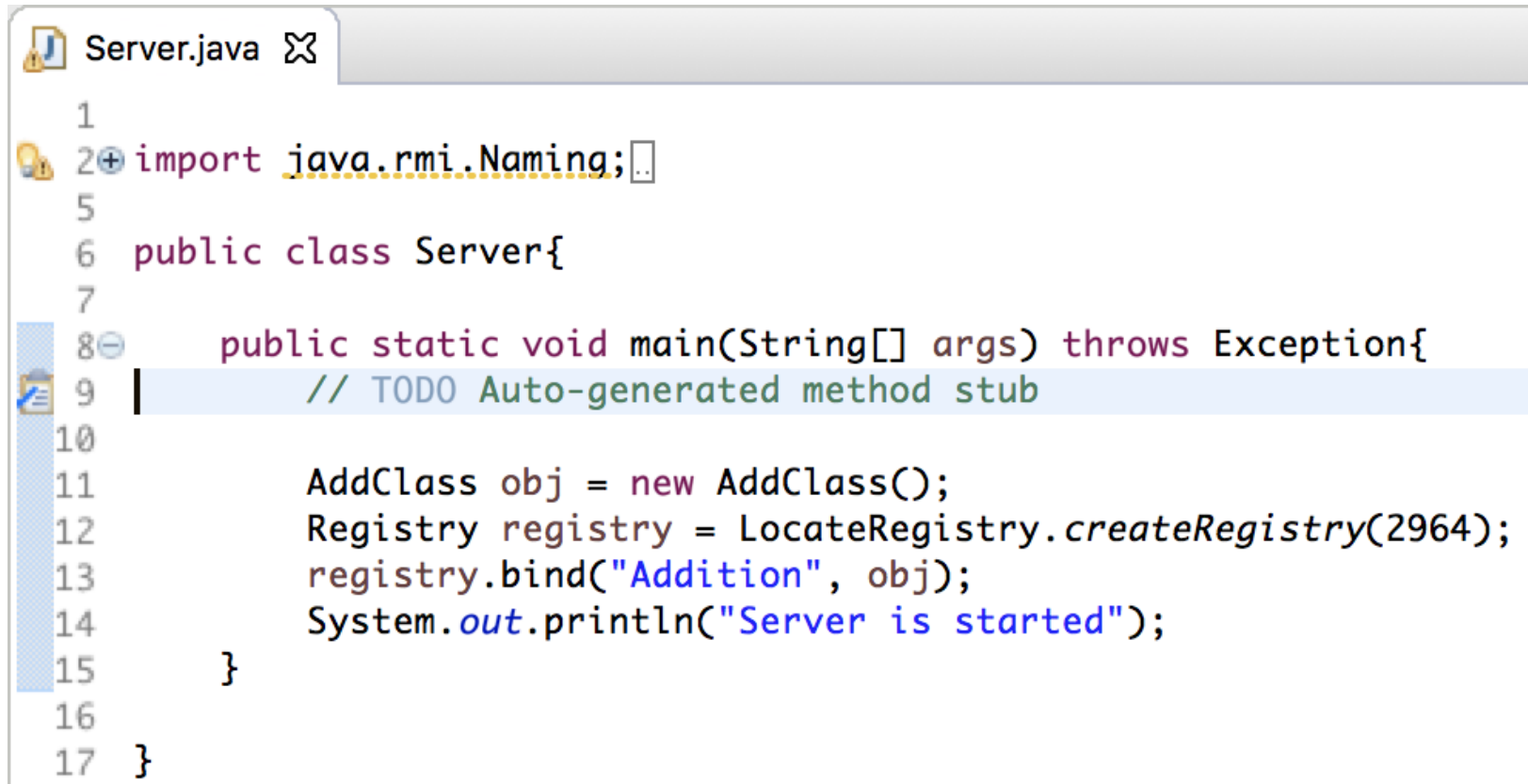


## 2. Implementation of Remote Interface

```
AddClass.java ✕  
1  
2+ import java.rmi.RemoteException;..  
4  
5 public class AddClass extends UnicastRemoteObject implements AddInterface{  
6  
7- public AddClass() throws Exception{  
8     super();  
9 }  
10  
11- public int add(int x, int y){  
12     return x + y;  
13 }  
14 }
```

- UnicastRemote Object:
- Used for exporting a remote object with JRMP(Java Remote Method Protocol) and obtaining a stub that communicates to the remote object.

### 3. Create Server



The screenshot shows a code editor window titled "Server.java". The code is as follows:

```
1
2 import java.rmi.Naming;
3
4
5
6 public class Server{
7
8     public static void main(String[] args) throws Exception{
9         // TODO Auto-generated method stub
10
11         AddClass obj = new AddClass();
12         Registry registry = LocateRegistry.createRegistry(2964);
13         registry.bind("Addition", obj);
14         System.out.println("Server is started");
15     }
16
17 }
```

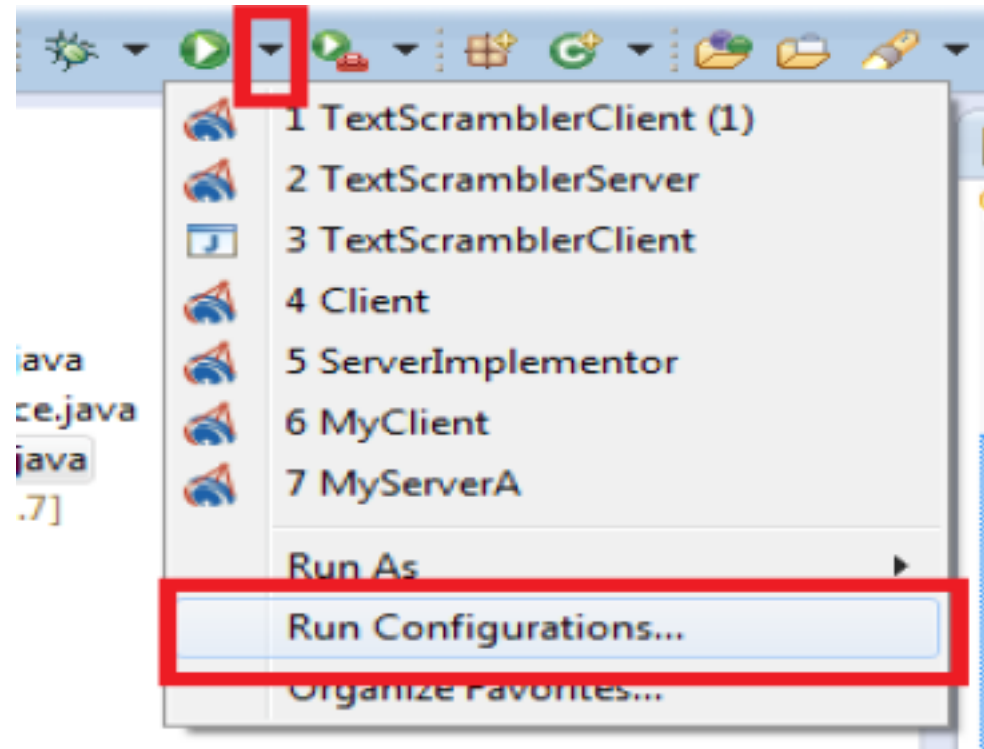
## 4. Create Client

```
Client.java ✕
1
2+ import java.rmi.Naming;
5
6 public class Client {
7
8-     public static void main(String[] args) throws Exception{
9         // TODO Auto-generated method stub
10
11         Registry registry = LocateRegistry.getRegistry(2964);
12
13         AddInterface obj = (AddInterface) registry.lookup("Addition");
14
15         int n = obj.add(5, 4);
16         System.out.println("Addition is : " + n);
17     }
18 }
```

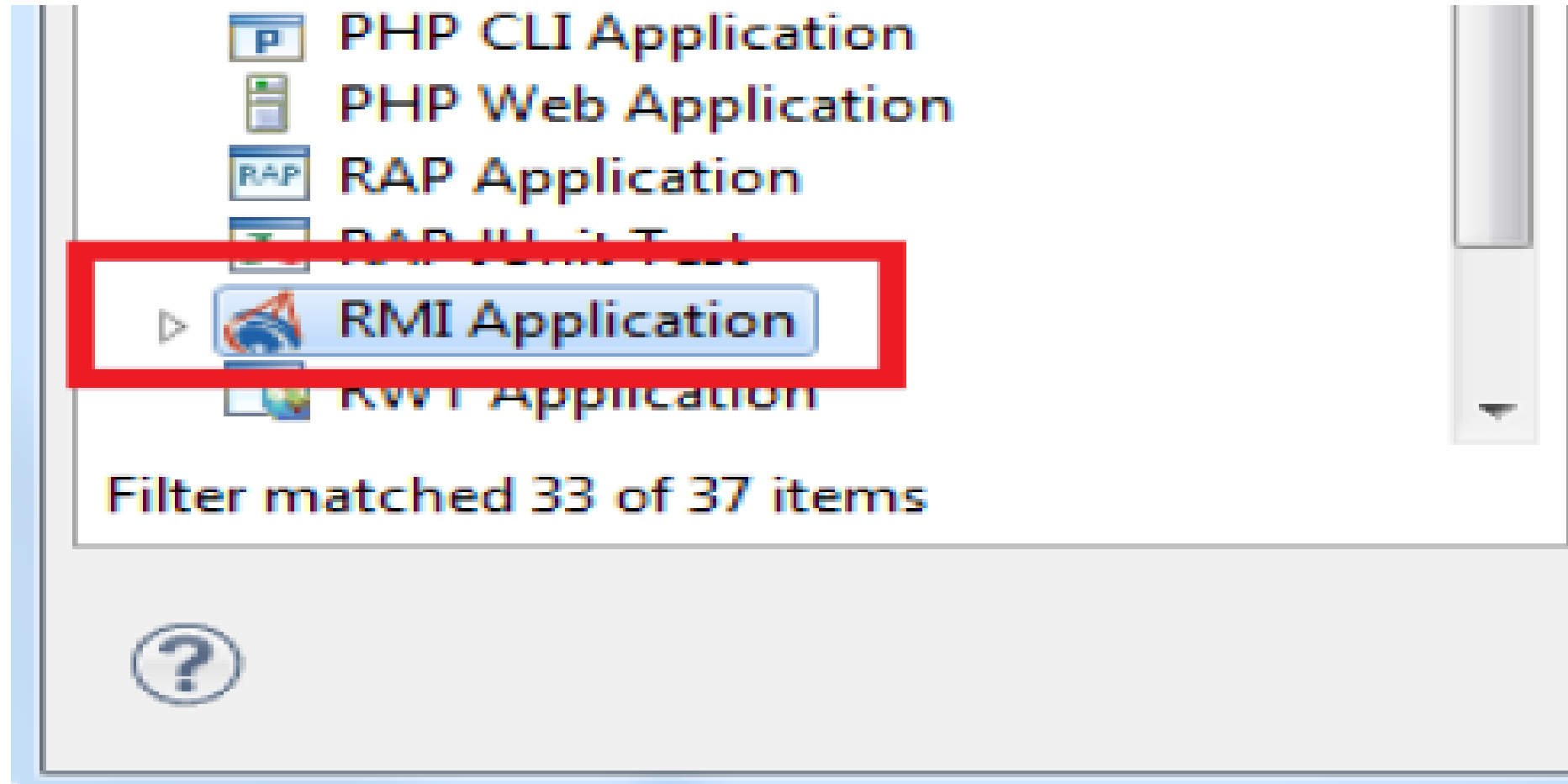
- On the Client!
- The Client must fetch the server from the registry
- If successful, the client will be able to use the methods residing on the server

# 5: Configure RMI Plugin

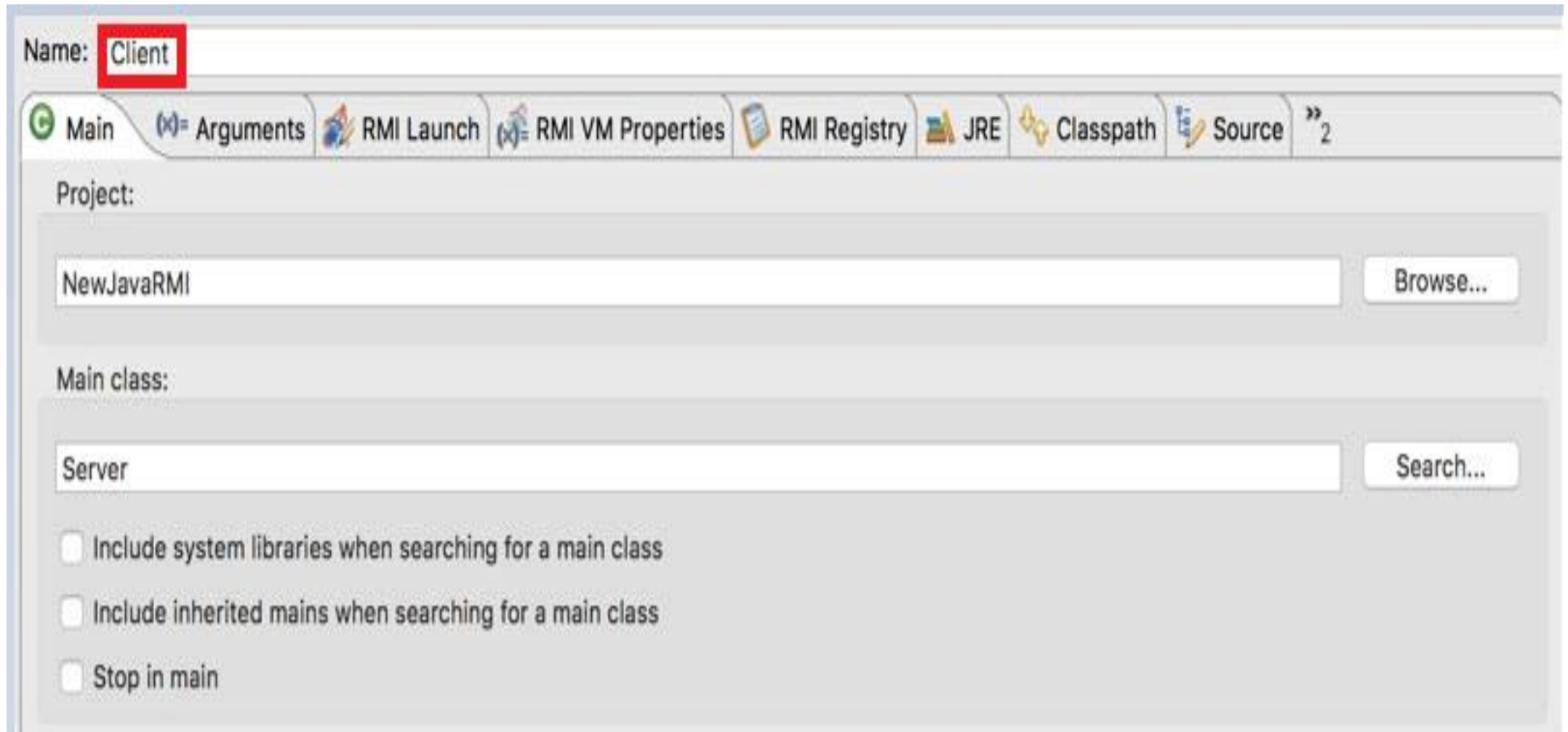
- Now to start our server we have to create a special RMI configuration



- Double Click on RMI Application



- Name your configuration – Client



The screenshot shows the Eclipse IDE configuration window for a Java application. The 'Name' field is set to 'Client' and is highlighted with a red box. The 'Project' field is set to 'NewJavaRMI' with a 'Browse...' button. The 'Main class' field is set to 'Server' with a 'Search...' button. There are three checkboxes at the bottom: 'Include system libraries when searching for a main class', 'Include inherited mains when searching for a main class', and 'Stop in main', all of which are currently unchecked.

Name: **Client**

Project: NewJavaRMI [Browse...](#)

Main class: Server [Search...](#)

☐ Include system libraries when searching for a main class

☐ Include inherited mains when searching for a main class

☐ Stop in main

- Select the appropriate main class by clicking on search



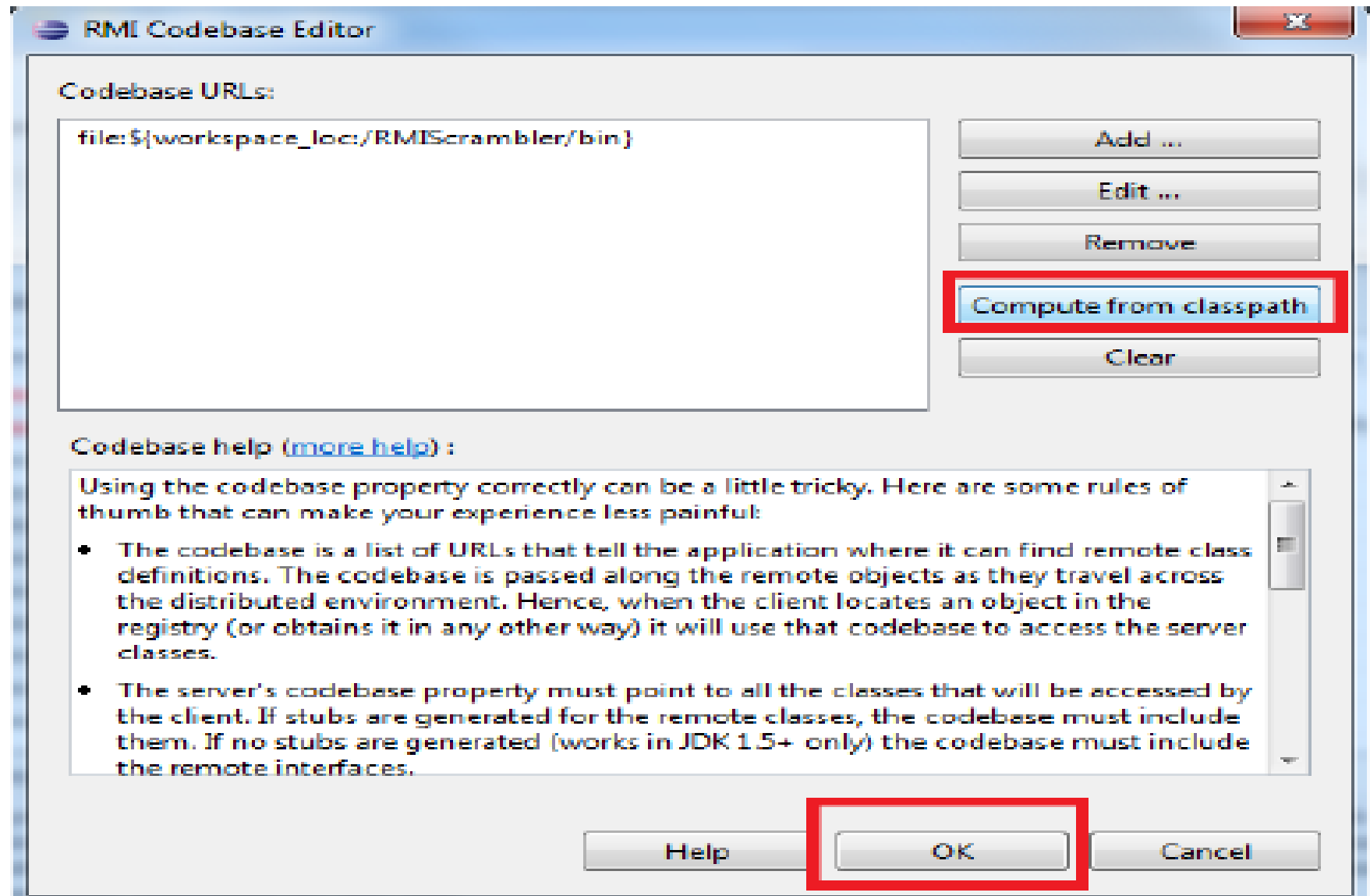
- Next you have to select RMI VM properties and click on the <Empty> to set a codebase.

Edit RMI properties (Click or press F2 to edit):

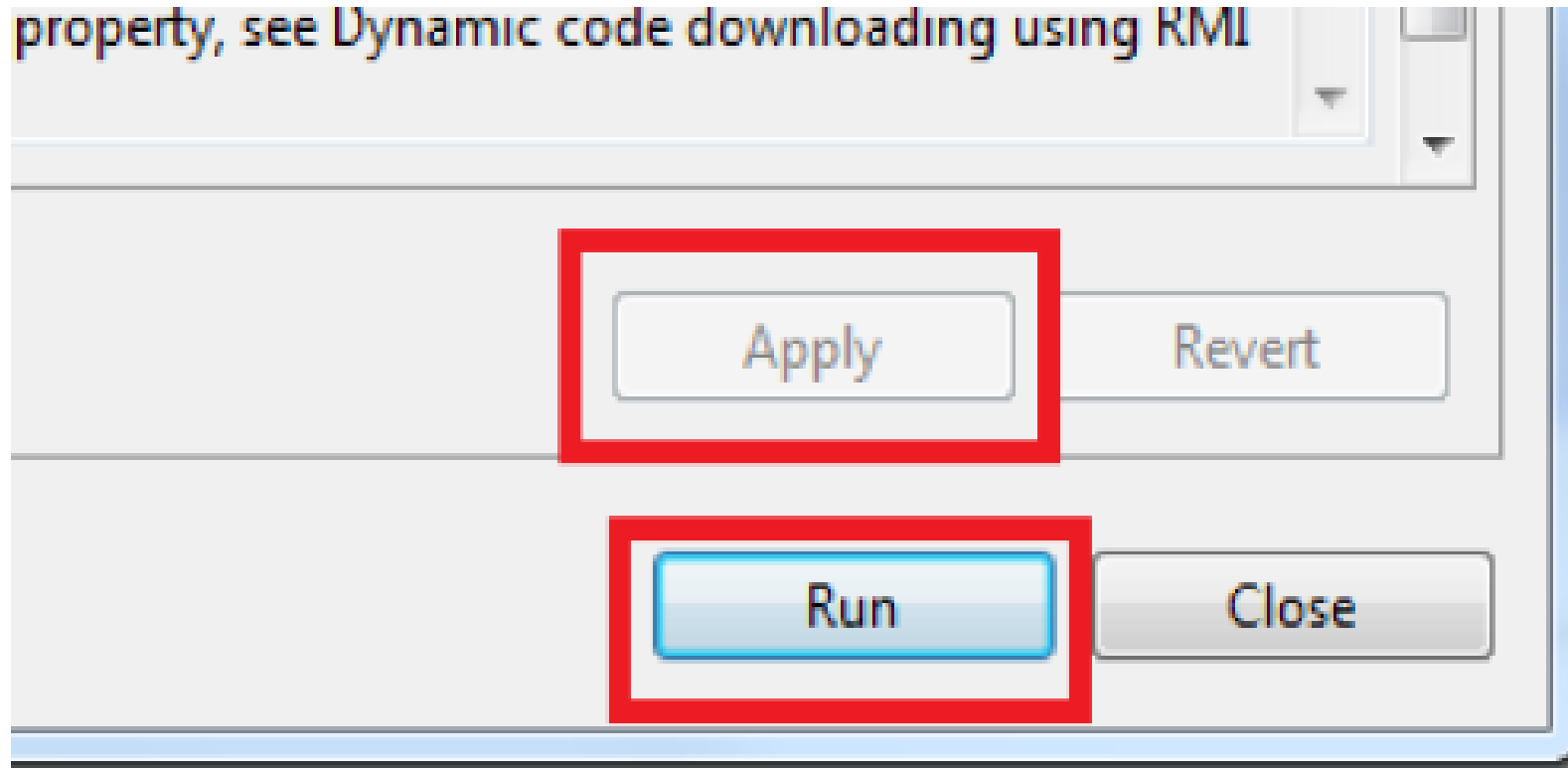
Clear Copy to Clipboard

Name	Version	Value
+ java.security.policy	1.1	
+ java.rmi.server.codebase	1.1	<Empty>
+ java.rmi.activation.port	12	
+ java.rmi.dgc.leaseValue	11	

- Click on compute from ClassPath and then ok.

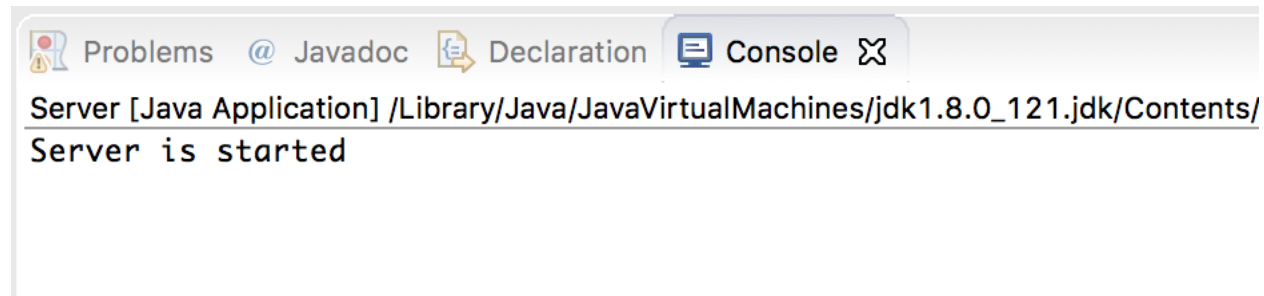


- Then click on Apply and Run. The server is finally starting!



## 6. Run server and then client

- Successful setup of Server
- Output



The screenshot shows the Eclipse IDE's Console window. The title bar includes tabs for 'Problems', 'Javadoc', 'Declaration', and 'Console'. The 'Console' tab is active, displaying the output of a Java application. The output text is: 'Server [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_121.jdk/Contents/Server is started'.

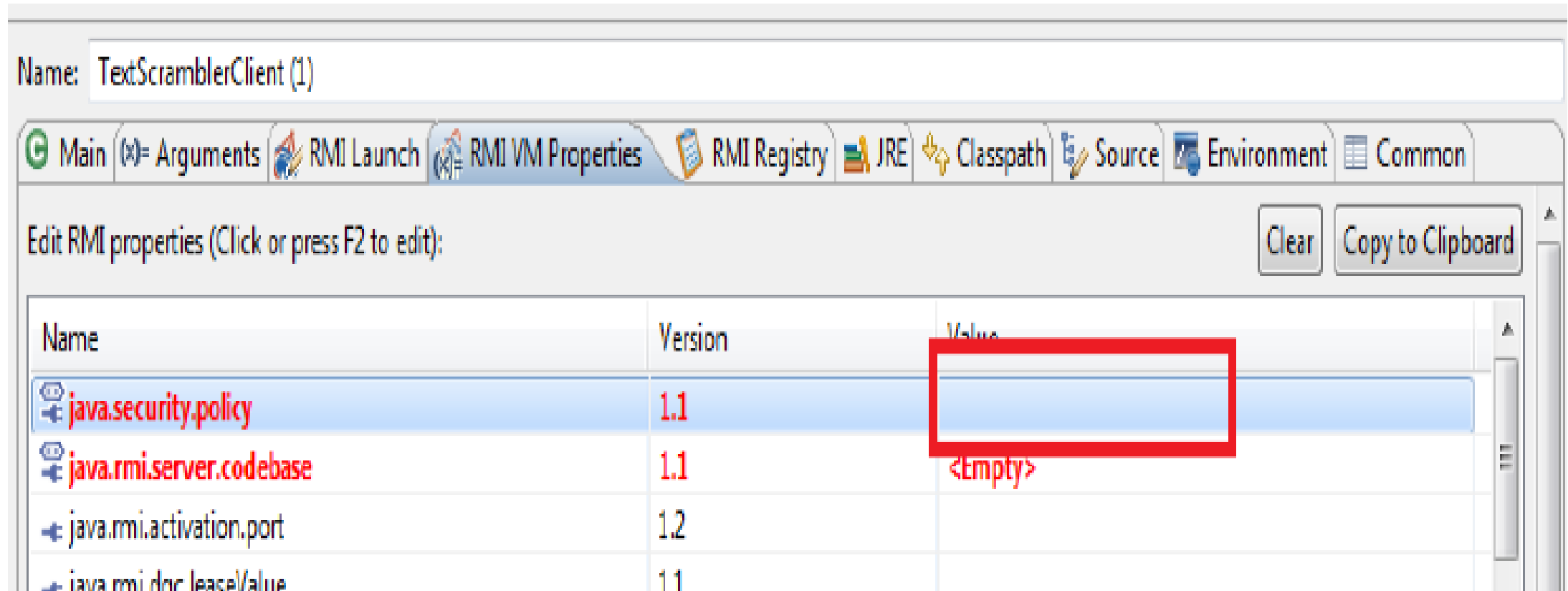
```
Server [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_121.jdk/Contents/  
Server is started
```

# Client :

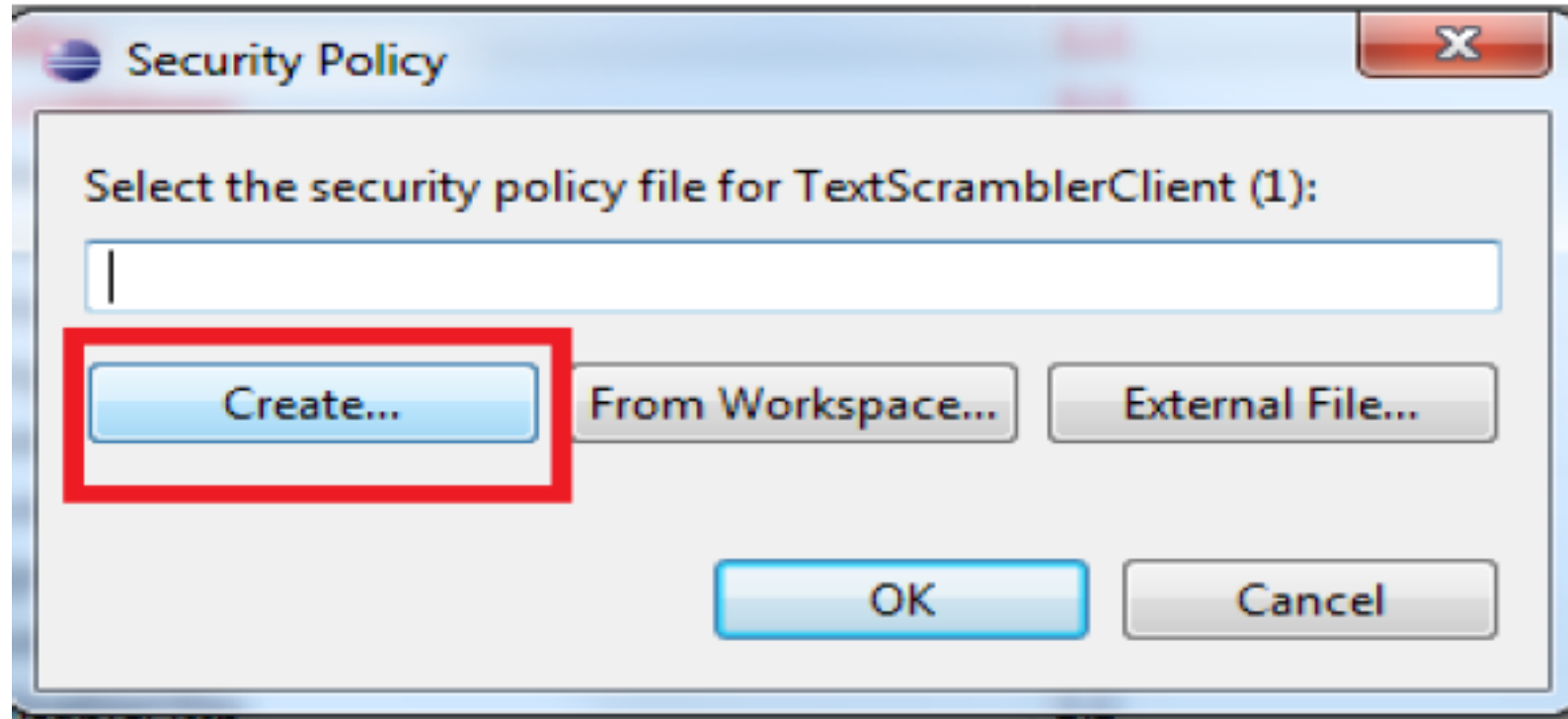
- For the client, you do not have to set the classpath. However, you do have to provide a security policy.
- To do so, go back to run configuration, double click on RMI Application and then RMI VM Properties.

# Security Policy

- Click right above the <Empty>

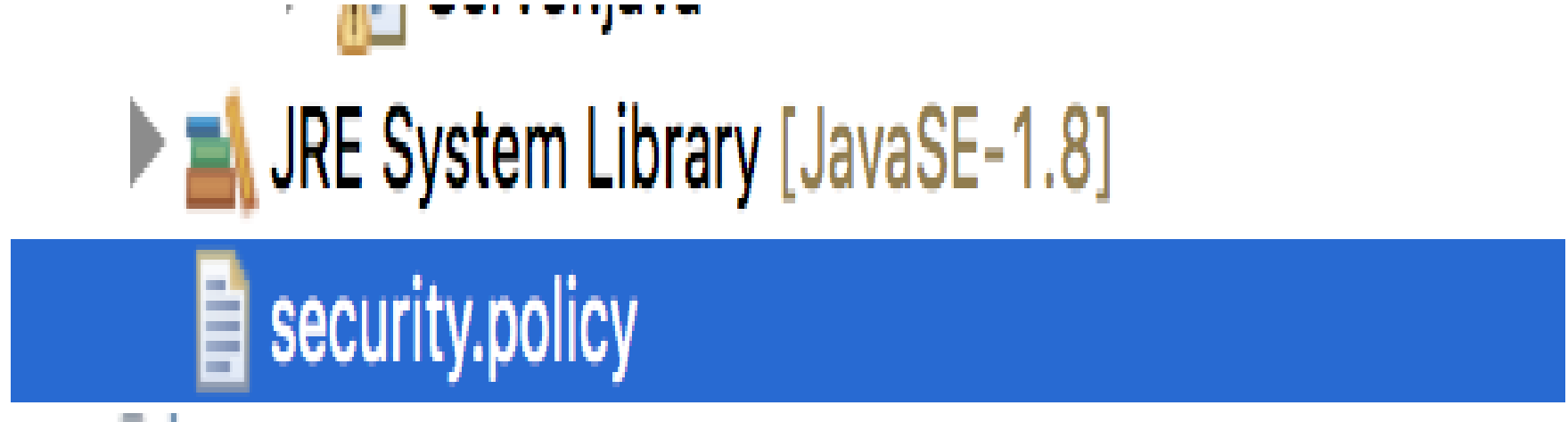


# Create a new Security Policy



# Successful setup of Security Policy

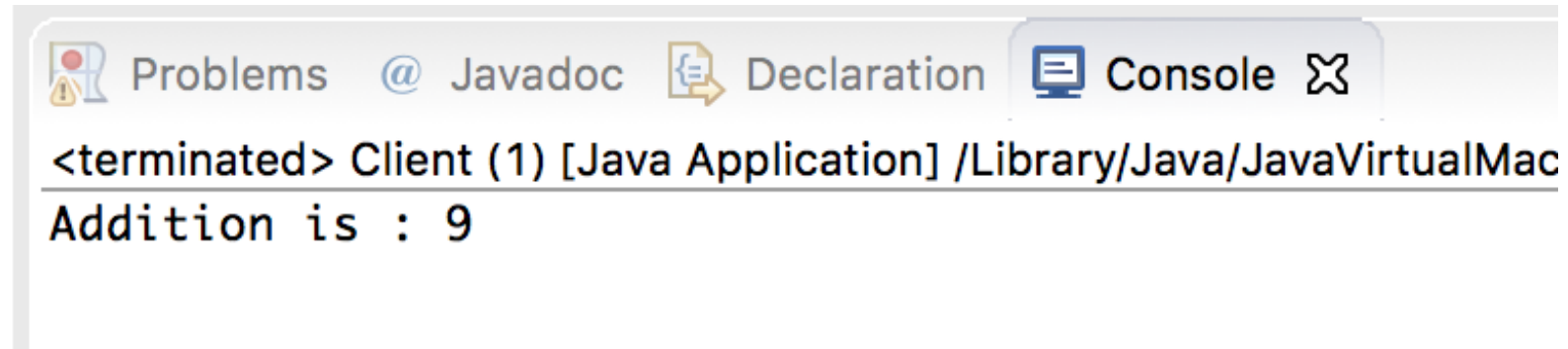
- If successful you will see the following:





# Starting New Client

- You may now start the client.
- Output:



The screenshot shows the Eclipse IDE's Console window. The title bar includes tabs for 'Problems', 'Javadoc', 'Declaration', and 'Console'. The 'Console' tab is active, displaying the output of a Java application. The output text is: '<terminated> Client (1) [Java Application] /Library/Java/JavaVirtualMac' followed by a horizontal line and 'Addition is : 9'.

```
<terminated> Client (1) [Java Application] /Library/Java/JavaVirtualMac  
Addition is : 9
```

# Reference Link

1. <https://docs.oracle.com/javase/tutorial/rmi/>
2. <http://java2all.com/technology/rmi/rmi-program/rmi-example>
3. <https://www.javatpoint.com/RMI>