# 15640 Lab2 Remote Method Invocation

Name: Yue Zhuo Andrew ID: yuezhuo Name: Yusi Zhang AndrewID: yusiz

## Project Design

This project implements a Remote Method Invocation (RMI) facility for Java. It provides a mechanism by which objects within one Java Virtual Machine (JVM) can invoke methods on objects within another JVM.

There is one RMI registry server, some remote servers and some clients. The registry server is responsible for the servers to register its services, and the handle requests such as locate and lookup from the client side.

The client side, registry server and the server side communicates with each other through socket, according to the IP address and port number of every node. Our system runs locally on one machine, but we assign every node a specific IP address and port number, so as to simulate the environment of the distributed system. The message delivered across them is wrapped as RMIMessage. There are mainly three types of the RMIMessage, method invocation, the return value of a method, and the exception information. The RMIMessage carries a list of information, including the type, the method name, the return value, the arguments, the exception, and the remote object reference.

The project is consisted of the following modules:

1. Registry
   1) Registry_Server
      The Registry_Server contains a hashtable of <service name, remote object reference>.
      It contains a main() function that invokes the rebind() method that binds the new server instance and the RemoteObjectRef instance, and then invokes the lookup()method.
      The lookup() method first connects the client via socket, and listens to the request sent from the client.
      If the request sent from the client is "locate", the server will pass the result to LocateSimpleRegistry on the client side via stream.
      If the request sent from the client is "lookup", the server will find the remote object reference based on the service name.
   2) RemoteObjectRef
      This class represents the remote object reference, which contains the basic information of a remote object reference, including the IP address, the port number, the object key, and the remote interface name.

The localise() method creates a new stub object.

3) SimpleRegistry
4) LocateSimpleRegistry

   This class is responsible for locating the RMI registry.

   It listens to the Registry server via socket and get the result of whether locating the registry successfully or not through the stream.

2. Communication
   1) MyRemote

      This interface extends the Serializable interface.
   2) MyStub

      The stub for a remote object acts as a client's local representative or proxy for the remote object. A stub for a remote object implements the same set of remote interfaces that a remote object implements. Since every stub corresponds to a remote object reference, we decide to let every stub to contain the RemoteObjectRef instance. Thus, when the stub transmits messages to remote servers, the server can easily know which remote object reference the stub is representing.
   3) RMIMessage

      There are three types of communication message in the system, invoking a method, the return value, and the exception information. The RMIMessage class represents the serializable RMI message. It contains the RMI message's type(invoke, return, or exception), method name, return value, parameters, exception, and the remote object reference.
   4) yourRMI

      This class represents the communication module for the RMI.

3. Test
   1) ZipCodeClient

      This class represents the client side of the project, and contains a main() method that locates a registry, lookup the service name, initializes the stub, and use the stub to invokes the find(), findAll() methods remotely on the remote server.
   2) ZipCodeList

      This class represents the data structure of the list of cities and their zipcodes.
   3) ZipCodeServer_Stub

      Every remote object reference has a stub. As soon as the client looks up a service name and gets a remote object reference, a stub is generated to represent the remote object reference.

This class implements ZipCodeServer interface, and extends MyStub class.

4) ZipCodeServer

   This interface extends the MyRemote interface, and has the following functions:

   Initialize, find, findAll, and printAll.

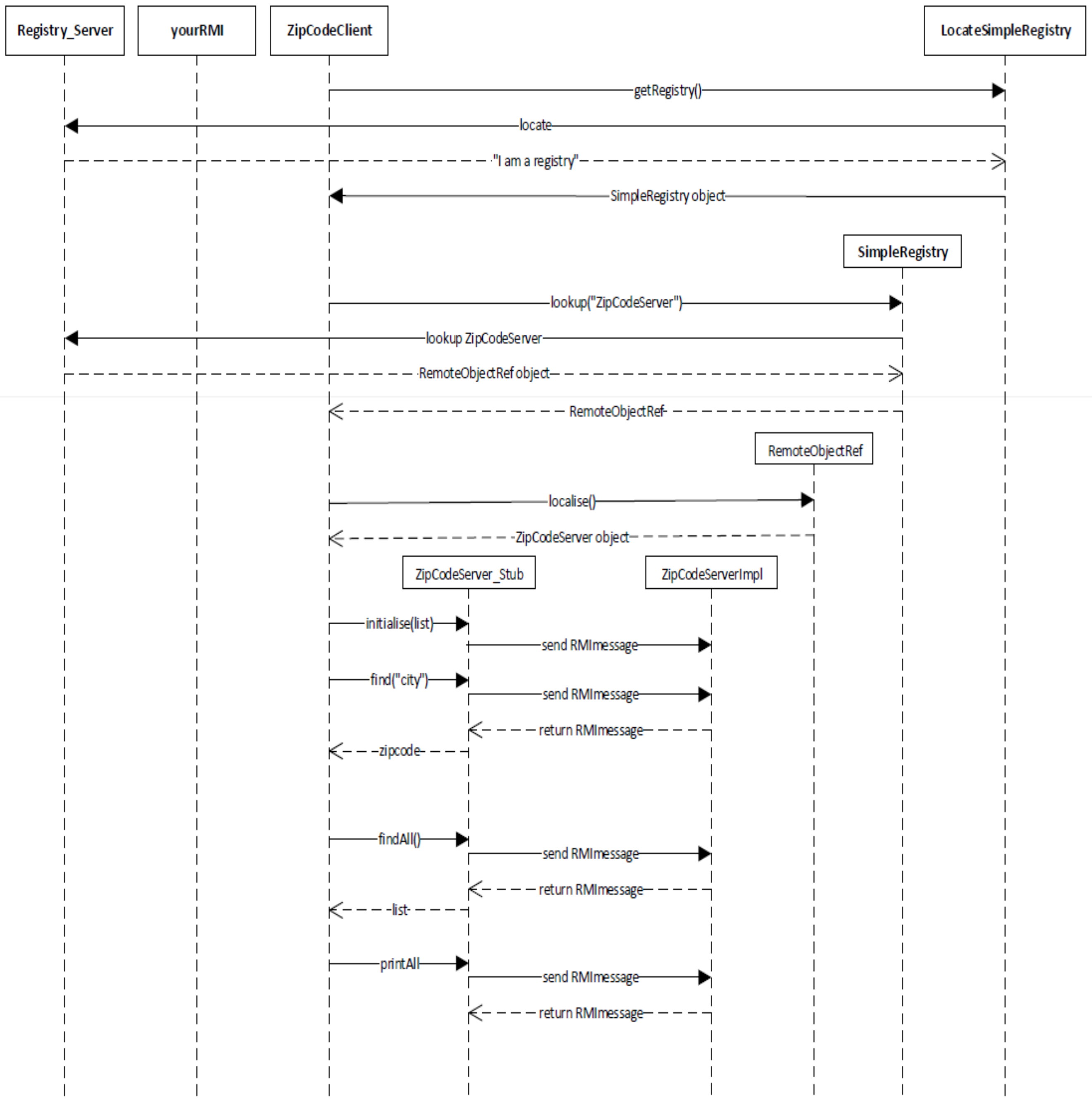
5) ZipCodeServerImpl

   This class implements the ZipCodeServer interface, and represents the server side of the project.

The project works mainly with the following steps:

1. The Registry server starts
2. The client locates the registry service
3. The client lookup the service name ("ZipCodeServer")on the server, and the server returns the remote object reference of the service to the client
4. The client generates a stub for the remote object reference
5. The client reads the city and zipcode list from a local file and initialize the list
6. The client invokes the find() method to find the zipcode of a certain city
   1) The find() method on the stub is called
   2) The stub marshals (writes and transmits) an RMIMessage to invoke the find() method on the remote server
   3) The stub unmarshals (reads) the RMIMessage returned from the remote server as an result
   4) The stub return the value to the client
7. The client invokes the findAll() method (to test exception message, we intentionally send wrong method name)
8. The client invokes the printAll() method

Here is the sequence diagram of the project:



## Implemented and unimplemented portions

We implemented the following funcitons:

1. The function of locating the remote objects, such as a registry service.
2. The function of invoking methods on remote objects, including passing the arguments and transmitting the return values back to the local objects.

However, there are some functions to be implemented:

1. The function of automatic stub compiler.
2. The function of retrieving the stub .class file via URL

## Build, deploy, and run the project

1. Copy the project to your computer.
2. Run the Registry_Server.java
3. Run the yourRMI.java
4. Run the ZipCodeClient.java

## Dependencies and software or system requirements

OS: Mac / windows

JRE: Java 1.7.0

## Test the project

To simplify the testing process, we decide to hardcode localhost as "127.0.0.1" for both service side and client side. But using different port to simulate "distributed system" environment.  yourRMI class(port 15641) and registry_server(port 15640) class runs on service client. Zipcodeclient class(dynamic port) runs on client side. Based on that, the following is testing process. All java filed have been compiled in bin folder.

1. Run the Registry_Server.java(registry package)
2. Run the yourRMI.java(communication package)
3. Run the ZipCodeClient.java (test package)

The console information of ZipcodeClient:

```
sr found //locate registry
Socket[addr=/127.0.0.1,port=15640,localport=49231]
socket made.
stream made.
```

```
Send interface name ZipCodeServer
command and service name sent.
found ZipCodeServer //lookup interface in service
Received ror 127.0.0.1115641ZipCodeServer
ror.localise 15641 //init stub for client
This is the original list.
city: Brick Lane, code: E2
city: Barrier Point, code: E16
city: Clindale, code: NW9
city: Brondesbury, code: NW6
city: Baker Street, code: W1
Stub init is called //call init method
Stub : init target port is15641
Stub init send message invoke class ZipCodeServer methodName: initialise/targs:
test.ZipCodeList@6808680
Get return from server null //no result from server

Server initalised.

This is the remote list given by find.
Stub find is called //find method called
Stub : target port is15641
Stub send message invoke class ZipCodeServer methodName: find/targs: Brick Lane
Get return from server E2 //followings are results from server
city: Brick Lane, code: E2
Stub find is called
Stub : target port is15641
Stub send message invoke class ZipCodeServer methodName: find/targs: Barrier Point
Get return from server E16
city: Barrier Point, code: E16
Stub find is called
Stub : target port is15641
Stub send message invoke class ZipCodeServer methodName: find/targs: Clindale
Get return from server NW9
city: Clindale, code: NW9
Stub find is called
Stub : target port is15641
Stub send message invoke class ZipCodeServer methodName: find/targs: Brondesbury
Get return from server NW6
city: Brondesbury, code: NW6
Stub find is called
Stub : target port is15641
Stub send message invoke class ZipCodeServer methodName: find/targs: Baker Street
Get return from server W1
city: Baker Street, code: W1

This is the remote list given by findall. //I intentionally wrongly call findall
metod to test exception message.
Stub findAll is called
Stub : target port is15641
Stub send message invoke class ZipCodeServer methodName: findAll/targs: testArg
exception catched in stub!!
java.lang.NoSuchMethodException: test.ZipCodeServerImpl.findAll(java.lang.String)
//because I send a wrong method name
 We test the remote site printing.
city: Brick Lane, code: E2
```

```
city: Barrier Point, code: E16

city: Clindale, code: NW9

city: Brondesbury, code: NW6

city: Baker Street, code: W1
```

The console information of yourRMI:

```
socket from Socket[addr=/127.0.0.1,port=49232,localport=15641]is received
Object is read invoke class ZipCodeServer methodName: initialise/targs:
test.ZipCodeList@5c5fba1c
args aretest.ZipCodeList@5c5fba1c
method name is initialise
ZipCodeServerImpl
socket from Socket[addr=/127.0.0.1,port=49233,localport=15641]is received
Object is read invoke class ZipCodeServer methodName: find/targs: Brick Lane
args areBrick Lane
method name is find
ZipCodeServerImpl
socket from Socket[addr=/127.0.0.1,port=49234,localport=15641]is received
Object is read invoke class ZipCodeServer methodName: find/targs: Barrier Point
args areBarrier Point
method name is find
ZipCodeServerImpl
socket from Socket[addr=/127.0.0.1,port=49235,localport=15641]is received
Object is read invoke class ZipCodeServer methodName: find/targs: Clindale
args areClindale
method name is find
ZipCodeServerImpl
socket from Socket[addr=/127.0.0.1,port=49236,localport=15641]is received
Object is read invoke class ZipCodeServer methodName: find/targs: Brondesbury
args areBrondesbury
method name is find
ZipCodeServerImpl
socket from Socket[addr=/127.0.0.1,port=49237,localport=15641]is received
Object is read invoke class ZipCodeServer methodName: find/targs: Baker Street
args areBaker Street
method name is find
ZipCodeServerImpl
socket from Socket[addr=/127.0.0.1,port=49238,localport=15641]is received
Object is read invoke class ZipCodeServer methodName: findAll/targs: testArg
args aretestArg
method name is findAll
ZipCodeServerImpl
exception catched in yourRMI
```

The console information of Registry_Server:

```
ServiceName is locate
ServiceName is lookup
interfacename ZipCodeServer
```