Remote Method Invocation (RMI): An API that allows an object to invoke a method on an object in another address space, which can be on the same machine or remote.
- https://www.geeksforgeeks.org/remote-method-invocation-in-java/

CSS 434
Program 3: RMI versus Mobile Agents

Instructor: Munehiro Fukuda
Due date: see the syllabus

1. Purpose

This assignment is intended to compare RMI and mobile agents in terms of programmability and performance. You will convert a pair of RMI client and server programs into the corresponding mobile agent, execute them, and evaluate their programmability and performance.

2. Remote Method Invocation

RMI (Remote Method Invocation) allows a client program to call a server function as passing predefined objects to it and to receive a return object. In the following example, we will see a design of simple RMI client, UnixClient.java that requests UnixServer.java to execute a given Unix command remotely and receives its output from the server. The corresponding server program and its bytecode, (i.e., UnixServer.java and UnixServer.class) are available from the /home/NETID/css434/hw3/rmi/ directory.

(1) Definition of objects passed from and returned to the client.

You may need to define such objects passed from and returned to the client. They must implement the Serializable interface that allows them to be automatically packed in and extracted from a byte-presented stream when transferred.

ReturnObj.java

```
import java.io.*;
public class ReturnObj implements Serializable {
       public ReturnObj( … ) {
       }
       // other data/method members.
}
```

Needless to say, if arguments and a return value are a predefined Java class, there are no needs to define new serializable classes. Indeed, UnixServer.java receives a String and returns a Vector of Strings as its argument to and return value from the execute( ) method. So, in program 3, we will skip this definition.

(2) Definition of server interface

Our next step is to define a server interface that inherits the Remote class. This interface simply defines a prototype of all RMI functions that will be made available at your server. In UnixServer.java, we will define only one RMI function, execute( ) that returns a given Unix command's execution output in a Vector object.

ServerInterface.java

```
import java.rmi.*;  // needed to extend the Remote class
import java.util.*;  // needed to use the Vector class
public interface ServerInterface extends Remote {
```

```
   public Vector execute( String command ) throws RemoteException;
   // exception needed for error detection
}
```

(3) Implementation of server program

An RMI server program must satisfy the following four requirements:

(a) Extend UnicastRemoteObject and implements ServerInterface

(b) Define a constructor that throws RemoteException

(c) Include the main( ) function that instantiates the server itself and binds this instance to “rmi://localhost:port/sympoblic_name”.

(d) Implements all RMI methods defined in the server interface.

The following shows the UnixServer.java code:

UnixServer.java

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

public class UnixServer extends UnicastRemoteObject
    implements ServerInterface {
    private boolean print = false;

    public UnixServer( String print ) throws RemoteException {
        this.print = print.startsWith( "P" );
    }

    public static void main( String args[] ) {
        if ( args.length != 2 ) {
            System.err.println( "usage: java UnixServer P/S port#" ); // print or silence
            System.exit( -1 );
        }
        try {
            UnixServer unixserver = new UnixServer( args[0] );
            Naming.rebind( "rmi://localhost:" + args[1] + "/unixserver", unixserver );
        } catch ( Exception e ) {
            e.printStackTrace( );
            System.exit( 1 );
        }
    }

    public Vector execute( String command ) {
        Vector<String> output = new Vector<String>( );
        String line;
        try {
            Runtime runtime = Runtime.getRuntime( );
            Process process = runtime.exec( command );
            InputStream input = process.getInputStream();
            BufferedReader bufferedInput
                = new BufferedReader( new InputStreamReader( input ) );
            while ( ( line = bufferedInput.readLine( ) ) != null ) {
                if ( print )
                    System.out.println( line );
                output.addElement( line );
            }
        } catch ( IOException e ) {
            e.printStackTrace( );
            return output;
        }
        return output;
    }
}
```

(4) Implementation of client program

An RMI client program must follow the following two instructions:

(a) Import java.rmi.*.

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

(b) Look for the server instance it wants to access.

```
ServerInterface server = ( ServerInterface )
   Naming.lookup( "rmi://serverIp:serverPort/symbolic_name" );
```

For instance, if you have started "UnixServer" at cssmpi2 on port 12345, the client program must look for it through:

```
ServerInterface server = ( ServerInterface )
   Naming.lookup( "rmi://cssmpi2:12345/unixserver" );
```

Make sure to catch Exception when calling Naming.lookup( ).

(c) Catch Exception when calling an RMI function.

```
Vector returnValue;
try {
   returnValue = server.execute( command );
} catch ( Exception e ) { }
```

A complete code of UnixClient.java will not be shown here, since this is a part of your programming assignment.

(5) Compilation and Exectuion

First, compile your server program with javac. Then, compile your client program.

```
javac UnixServer.java
javac UnixClient.java
```

To run the server program, type as follows:

```
rmiregistry port&
java UnixServer port
```

where port is the last five digit of your student id.

Make sure to kill rmiregistry after you terminate your server program.

3. UWAgent Mobile Agent Execution Platform

UWAgent is a Java-based mobile agent execution platform developed in the CSS/UWB Distributed Computing Laboratory. It is used for implementing a grid-computing middleware. To use UWAgent, follow the instructions below:

(1) Download the system

```
> ssh mary2003@cssmpi1h.uwb.edu
$ cd css434/programs/prog3
```

Copy UWAgent.jar, UWAgentUserManual.pdf, and other sample programs from the /home/NETID/css434/hw3/uwagent/ directory to your working directory:

```
cp /home/NETID/css434/hw3/uwagent/* .
```

```
> scp mary2003@cssmpi1h.uwb.edu:/home/NETID/mary2003/css434/programs/prog3/* Downloads/
```

(2) Code a mobile agent

There are three requirements you have to follow:

(a) A mobile agent in UWAgent must extend the UWAgent class and implement the Serializable interface

(b) The constructor should receive only String[] or no arguments.

(c) The agent starts init( ) right after its constructor call. The init( ) function receives no arguments and returns void.

(d) The agent migrates to another site with hop( ). The hop( ) function receives three arguments: the ip name of the next site, the function name to call there, and String[]. The third argument may be null. For instance,

```
hop( "cssmpi2", "func", null );
```

With this statement, the calling agent migrates to cssmpi2 and calls func( ) without any arguments.

(e) The agent must define functions called upon a migration. Such functions may receive String[] or nothing and must return void. For instance,

```
public void func( ) {
}
```

will be called upon a migration when the agent executes hop("cssmpi2", "func", null);

If there are no more hops, the agent will be terminated upon the return of the function invoked by hop. The following shows a simple agent code:

```
import java.io.*;
import UWAgent.*;

public class AnAgent extends UWAgent implements Serializable {
    private String destination = null;
    public AnAgent( String[] args ) {
        System.out.println( "Injected" );
        destination = args[0];
    }
    public AnAgent( ) {
        System.out.println( "Injected" );
        destination = "localhost";
    }

    public void init( ) {
        System.out.println( "I'll hop to " + destination );
        String[] args = new String[1];
        args[0] = "hello";
        hop( destination, "func", args );
    }

    public void func( String[] args ) {
        System.out.println( args[0] );
    }
}
```

(3) Compilation

```
javac -classpath UWAgent.jar:. AnAgent
```

Ignore a deprecated API warning message that is originated from UWAgent.jar but not your own agent program.

(4) Execute the platform and start a mobile agent

Start UWPlace at each site you would like to dispatch your mobile agent. For instance, in the above example, you can run UWPlace at cssmpi1h and cssmpi2h:

```
java -cp UWAgent.jar:. UWAgent.UWPlace -p 12345
```

Finally, you can submit your agent (from cssmpi1 in the above example):

```
java -cp UWAgent.jar:. UWAgent.UWInject -p 12345 localhost AnAgent cssmpi2h
```

To shut down the UWPlace, simply kill it at each site. (Type control + c. If you run UWPlace in background, type fg. Then type control + c.)

4. Statement of Work

Part 1a: Implementation of UnixClient.java

Given the P or C option, a server port number, the number of servers, a list of server IP names, the number of Unix commands, and a list of Unix commands, UnixClient.java calls UnixServer's execute( ) function at each of these servers as many times as the number of the commands to execute each of these commands remotely. For instance, if UnixClient.java is invoked with the following parameters:

java UnixClient P/C 12345 2 cssmpi2h cssmpi3h 4 who ls ps df

it should call UnixServer's execute( ) method at cssmpi2h four times, each executing who, ls, ps, and df respectively, and thereafter call execute( ) at cssmpi3h four times to execute these four commands. Your UnixClient.java program receives a Vector object from each execute( ) call as the outputs of the corresponding Unix command. After invoking all these commands at all the servers, if your client program received "P" as its first argument, it prints out these outputs to System.out. If your client program received non "P" string, (e.g., "C") as its first argument, the client program should print out only the number of lines in the messages it received from servers, (i.e., the size of Vector object). In the execution example below, messages in red are from servers, in which case UnixClient.java should print 46.

Server execution:

```
[css434@cssmpi2h rmi]$ rmiregistry 12345&
[1] 22613
[css434@cssmpi2h rmi]$ java UnixServer P 12345

[css434@cssmpi3h rmi]$ rmiregistry 12345&
[1] 3254
[css434@cssmpi3h rmi]$ java UnixServer P 12345
```

Client execution:

```
[css434@cssmpi1h rmi]$ java UnixClient P 12345 2 cssmpi2h cssmpi3h 4 who ls ps df
port = 12345, nServers = 2, server1 = cssmpi2h, command1 = who
============================================================
cssmpi2h command(who):.............................
css434   pts/12       2016-03-22 15:57 (50.46.157.107)
cssmpi2h command(ls):.............................
ServerInterface.class
ServerInterface.java
UnixClient.class
UnixClient.java
UnixServer.class
UnixServer.java
UnixServer_Stub.class
cssmpi2h command(ps):.............................
  PID TTY          TIME CMD
22502 pts/12   00:00:00 bash
22613 pts/12   00:00:00 rmiregistry
22808 pts/12   00:00:00 java
22839 pts/12   00:00:00 ps
cssmpi2h command(df):.............................
Filesystem              1K-blocks       Used  Available Use% Mounted on
```

```
udev                         8178948          4    8178944   1% /dev
tmpfs                        1638744       1412    1637332   1% /run
/dev/sda1                  464121624   23261712  417260808   6% /
none                             4          0          4   0% /sys/fs/cgroup
none                          5120          0       5120   0% /run/lock
none                       8193716        152    8193564   1% /run/shm
none                        102400         40     102360   1% /run/user
metis.uwb.edu:/usr/apps 5812624384 1897334784 3622327296  35% /usr/apps
metis:/home             5812624384 1897334784 3622327296  35% /net/metis/home

============================================================
cssmpi3h command(who):.............................
css434   pts/12       2016-03-22 15:57 (50.46.157.107)
cssmpi3h command(ls):.............................
ServerInterface.class
ServerInterface.java
UnixClient.class
UnixClient.java
UnixServer.class
UnixServer.java
UnixServer_Stub.class
cssmpi3h command(ps):.............................
  PID TTY          TIME CMD
 2961 pts/12   00:00:00 bash
 3254 pts/12   00:00:00 rmiregistry
 3273 pts/12   00:00:00 java
 3305 pts/12   00:00:00 ps
cssmpi3h command(df):.............................
Filesystem               1K-blocks       Used  Available Use% Mounted on
udev                       8178956          4    8178952   1% /dev
tmpfs                      1638748       1388    1637360   1% /run
/dev/sda1                464121624   23258392  417264128   6% /
none                             4          0          4   0% /sys/fs/cgroup
none                          5120          0       5120   0% /run/lock
none                       8193728        148    8193580   1% /run/shm
none                        102400         40     102360   1% /run/user
metis.uwb.edu:/usr/apps 5812624384 1897334784 3622327296  35% /usr/apps
metis:/home             5812624384 1897334784 3622327296  35% /net/metis/home

Execution Time = 253
```

Include time measuring code that measures time elapsed for UnixClient.java to execute its entire sequence of operations. For this measurement, use the Date class.

Part 1b: Performance evaluation of UnixClient.java:
Conduct the following three tests as increasing the number of remote servers from 1 to 3, (e.g., cssmpi1h, cssmpi2h, and cssmpi3h).

Test 1: Execute multiple commands (4 or 12 commands) at remote servers. This test intends to see how RMI client/servers would run longer with more commands.

Test 2: Execute a grep at remote servers. This test finds all occurrences of a given string, (i.e., “123”) in text1.txt (with 10GB) at each remote server and prints out only the total number of occurrences at the client side.

Test 3: Download text1.txt (with 10GB) from each remote server, locally finds all occurrences of a given string, (i.e., “123”), and prints out the total number of occurrences at the client side.

Naturally, test 3 has a big data-downloading overhead. For your convenience, the following lists all executions you have to conduct. Note that the executions below were done with old cssmpi1-cssmpi4 in year 2019 but not with new cssmpi1h-cssmpi4h, and thus execution time may be slightly different with the new machines.

```
RMI
Test 1: Executing multiple commands at remote servers
[16:12:21] mfukuda@cssmpi1: ~/css434/hw3/rmi $ java UnixClient P 50763 1 cssmpi2 4 who ls
ps df
print/count = print port = 50763, nServers = 1, server1 = cssmpi2, command1 = who
…
Exectuion Time = 248

[16:15:13] mfukuda@cssmpi1: ~/css434/hw3/rmi $ java UnixClient P 50763 2 cssmpi2 cssmpi3
4 who ls ps df
print/count = print port = 50763, nServers = 2, server1 = cssmpi2, command1 = who
…
Exectuion Time = 279

[16:16:04] mfukuda@cssmpi1: ~/css434/hw3/rmi $ java UnixClient P 50763 3 cssmpi2 cssmpi3
cssmpi4 4 who ls ps df
print/count = print port = 50763, nServers = 3, server1 = cssmpi2, command1 = who
Exectuion Time = 306

[16:18:10] mfukuda@cssmpi1: ~/css434/hw3/rmi $ java UnixClient P 50763 1 cssmpi2 12 who
ls ps df who ls ps df who ls ps df
print/count = print port = 50763, nServers = 1, server1 = cssmpi2, command1 = who
…
Exectuion Time = 285

[16:20:04] mfukuda@cssmpi1: ~/css434/hw3/rmi $ java UnixClient P 50763 2 cssmpi2 cssmpi3
12 who ls ps df who ls ps df who ls ps df
print/count = print port = 50763, nServers = 2, server1 = cssmpi2, command1 = who
…
Exectuion Time = 357

[16:21:14] mfukuda@cssmpi1: ~/css434/hw3/rmi $ java UnixClient P 50763 3 cssmpi2 cssmpi3
cssmpi4 12 who ls ps df who ls ps df who ls ps df
…
Exectuion Time = 421

Test 2: Executing a grep at remote servers

[15:59:25] mfukuda@cssmpi1: ~/css434/hw3/rmi $ java UnixClient C 50763 1 cssmpi2 1 grep\
-o\ 123\ ../files/text1.txt
print/count = count port = 50763, nServers = 1, server1 = cssmpi2, command1 = grep -o
123 ../files/text1.txt
count = 359
Exectuion Time = 330

[16:00:11] mfukuda@cssmpi1: ~/css434/hw3/rmi $ java UnixClient C 50763 2 cssmpi2 cssmpi3
1 grep\ -o\ 123\ ../files/text1.txt
print/count = count port = 50763, nServers = 2, server1 = cssmpi2, command1 = grep -o
123 ../files/text1.txt
count = 718
Exectuion Time = 424

[16:02:54] mfukuda@cssmpi1: ~/css434/hw3/rmi $ java UnixClient C 50763 3 cssmpi2 cssmpi3
cssmpi4 1 grep\ -o\ 123\ ../files/text1.txt
print/count = count port = 50763, nServers = 3, server1 = cssmpi2, command1 = grep -o
123 ../files/text1.txt
```

```
count = 1077
Exectuion Time = 547
```

Test 3: Downloading a file from remote servers and thereafter executing a grep locally

```
[16:05:45] mfukuda@cssmpi1: ~/css434/hw3/rmi $ java UnixClient P 50763 1 cssmpi2 1
cat\ ../files/text1.txt | grep -o 123 | wc -l
Exectuion Time = 7863
359

[16:08:15] mfukuda@cssmpi1: ~/css434/hw3/rmi $ java UnixClient P 50763 2 cssmpi2 cssmpi3
1 cat\ ../files/text1.txt | grep -o 123 | wc -l
Exectuion Time = 14651
718

[16:11:35] mfukuda@cssmpi1: ~/css434/hw3/rmi $ java UnixClient P 50763 3 cssmpi2 cssmpi3
cssmpi4 1 cat\ ../files/text1.txt | grep -o 123 | wc -l
Exectuion Time = 23762
```

After your performance evaluation, summarize your results in a table, as shown below:

Note: Choose the fastest result among 5 executions

RMI

Test 1 java UnixClient P 50763 #nodes cssmpi2-4 #commands who ls ps df

# nodes	4 commands	12 commands
1		
2		
3		

Test 2 java UnixClient C 50763 #nodes cssmpi2-4 1 grep\ -o\ 123\ ../files/text1.txt

# nodes	
1	
2	
3	

Test 3 java UnixClient P 50763 #nodes cssmpi2-4 1 cat\ ../files/text1.txt | grep -o 123 | wc -l

# nodes	
1	
2	
3	

Part 2a: Implementation of WhoAgent.java

Given the P or C option, the number of servers, a list of server IP names, the number of Unix commands, and a list of Unix commands, UnixAgent.java visits each of these servers, executes each command through the Java Runtime class, and finally comes back to where it was injected to print out all the command outputs if its first argument was "P". The output format should be the same as part1a. If the first argument was non "P" string, (e.g., "C"), WhoAgent.java should print out only the number of lines it received from the servers. Similar to UnixClient.java, include time measuring code that measures time elapsed for UnixAgent.java to execute its entire sequence of command executions.

Part 2b: Performance evaluation of WhoAgent.java
Conduct the following two tests as increasing the number of remote servers from 1 to 3, (e.g., cssmpi1, cssmpi2, and cssmpi3).

Test 1: Execute multiple commands (4 or 12 commands) at remote servers. This test intends to see how UWAgent would run longer with more commands.

Test 2: Execute a grep at remote servers. This test finds all occurrences of a given string, (i.e., "123") in text1.txt (with 10GB) at each remote server and prints out only the total number of occurrences at the client side.

For your convenience, the following lists all executions you have to conduct. Note that the executions below were done with old cssmpi1-cssmpi4 in year 2019 but not with new cssmpi1h-cssmpi4h, and thus execution time may be slightly different with the new machines.

```
UWAgents
Test 1: Executing multiple commands at remote servers.

[16:42:10] mfukuda@cssmpi1: ~/css434/hw3/uwagent $ java -cp UWAgent.jar:.
UWAgent.UWInject -p 50763 localhost UnixAgent P 1 cssmpi2 4 who ls ps df
Execution Time = 53

[16:48:48] mfukuda@cssmpi1: ~/css434/hw3/uwagent $ java -cp UWAgent.jar:.
UWAgent.UWInject -p 50763 localhost UnixAgent P 2 cssmpi2 cssmpi3 4 who ls ps df
Execution Time = 87

[16:49:04] mfukuda@cssmpi1: ~/css434/hw3/uwagent $ java -cp UWAgent.jar:.
UWAgent.UWInject -p 50763 localhost UnixAgent P 3 cssmpi2 cssmpi3 cssmpi4 4 who ls ps df
Execution Time = 147

[16:52:40] mfukuda@cssmpi1: ~/css434/hw3/uwagent $ java -cp UWAgent.jar:.
UWAgent.UWInject -p 50763 localhost UnixAgent P 1 cssmpi2 12 who ls ps df who ls ps df
who ls ps df
Execution Time = 83

[16:52:31] mfukuda@cssmpi1: ~/css434/hw3/uwagent $ java -cp UWAgent.jar:.
UWAgent.UWInject -p 50763 localhost UnixAgent P 2 cssmpi2 cssmpi3 12 who ls ps df who ls
ps df who ls ps df
Execution Time = 135

[16:51:47] mfukuda@cssmpi1: ~/css434/hw3/uwagent $ java -cp UWAgent.jar:.
UWAgent.UWInject -p 50763 localhost UnixAgent P 3 cssmpi2 cssmpi3 cssmpi4 12 who ls ps df
who ls ps df who ls ps df
Execution Time = 218

Test 2: Executing a grep at remote servers

[21:28:02] mfukuda@cssmpi1: ~/css434/hw3/uwagent $ java -cp UWAgent.jar:.
UWAgent.UWInject -p 50763 localhost UnixAgent C 1 cssmpi2 1 grep\ -o\
123\ ../files/text1.txt
Execution Time = 139

[21:29:24] mfukuda@cssmpi1: ~/css434/hw3/uwagent $ java -cp UWAgent.jar:.
UWAgent.UWInject -p 50763 localhost UnixAgent C 2 cssmpi2 cssmpi3 1 grep\ -o\
123\ ../files/text1.txt
Execution Time = 240
```

```
[21:30:21] mfukuda@cssmpi1: ~/css434/hw3/uwagent $ java -cp UWAgent.jar:.
UWAgent.UWInject -p 50763 localhost UnixAgent C 3 cssmpi2 cssmpi3 cssmpi4 1 grep\ -o\
123\ ../files/text1.txt
Execution Time = 363
```

After your performance evaluation, summarize your results in a table, as shown below:

Note: Choose the fastest result among 5 executions

UWAgents

Test 1

java -cp UWAgent.jar:. UWAgent.UWInject -p 50763 localhost UnixAgent P #nodes cssmpi2-4 4 #commands who ls ps df

# nodes	4 commands	12 commands
1		
2		
3		

Test 2

java -cp UWAgent.jar:. UWAgent.UWInject -p 50763 localhost UnixAgent C #nodes cssmpi2-4 grep\ -o\ 123\ ../files/text1.txt

# nodes	
1	
2	
3	

What to Turn in

The homework is due at the beginning of class on the due date. You have to submit the following materials in a PDF file to "Canvas". Your PDF file should include:

(1) Your report
(2) Source code pasted in your PDF report
(3) Execution outputs pasted in your PDF report

Criteria	**Grade**
Documentation of your algorithm including explanations and illustrations in one or two pages. (1) UnixClient.java: 2.5pts (2) UnixAgent.java: 2.5pts	5pts
Source code that adheres good modularization, coding style, and an appropriate amount of comments. (1) UnixClient.java: 1.5pts a. A use of Naming.lookup and RMI call (0.5pts) b. A use of the Date class (0.5pts) c. Printing all command outputs at the end if the P option is given, otherwise #lines of outputs with the S option. (0.5pts) (2) UnixAgent.java: 2.5pts a. hop( ) called at the end of a function but not inside any for/while loop. (0.5pts) b. An agent visiting one to another server rather than going back and forth between the local and each remote host (0.5pts)	5pts

c. An agent returning back to where it was inject (0.5pts) d. A use of the Date class (0.5pts) e. Printing all command outputs at the end if the P option is given, otherwise #lines of outputs with the S option (0.5pts) (3) Coding: 1pt a. Code completeness (0.5pts) b. Coding style and readability (0.5pts)	
Execution output such as a snapshot of your display/windows or contents of standard output redirected to a file. (1) UnixClient.java: 2.5pts a. A correct result when using two (or more) servers, each executing four commands. This receives 2.5pts. b. A result didn't use two servers (or more) or execute four commands at each server. Or the result included minor errors. This case receives 2pts. c. Incomplete results receive 1.5pts (2) UnixAgent.java: 2.5pts a. A correct result when using two (or more) servers, each executing four commands. This receives 2.5pts. b. A result didn't use two servers (or more) or execute four commands at each server. Or the result included minor errors. This case receives 2pts. c. Incomplete results receive 1.5pts	5pts
Discussions about the programmability and performance comparison between the RMI version (i.e., UnixClient.java and UnixServer.java) and the UWAgent version (i.e., UnixAgent.java). (1) Programmability: discuss about the total # LOC of UnixClient.java/UnixServer.java versus UnixAgent.java, difficulty in your paradigm shift to agent programming, etc. (2.5pts) (2) Performance comparison: summarize your performance results in the tables that are specified in the "statement of work" part 1b and part 2b. Discuss under what conditions UnixAgent.java outperforms UnixClient.java. You may tell that UnixAgent.java does not perform well at all, based on your experiment. (2.5pts)	5pts
Total	20pts

Your lab3a and lab3b will be graded together with program 3. For each lab:

Source code	0.5pts
Outputs	0.5pts