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 ~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 {
(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 {
    public UnixServer( ) throws RemoteException {
    }
    public static void main( String args[] ) {
        if (args.length != 1) {
            System.err.println("usage: java UnixServer port#");
            System.exit( -1 );
        }
        try {
            UnixServer unixserver = new UnixServer( );
            Naming.rebind("rmi://localhost:" + args[0] + "/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 ) {
                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 uw1-320-20 on port 12345,
the client program must look for it through:
ServerInterface server = (ServerInterface)
    Naming.lookup("rmi://uw1-320-20:12345/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 Execution
First, compile your server program with javac and thereafter with rmic. Then, compile
your client program.

    javac UnixServer.java
    rmic UnixServer
    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
Copy UWAgent.jar, UWAgentUserManual.pdf, and other sample programs from the
~css434/hw3/uwagent/ directory to your working directory:

    cp ~css434/hw3/uwagent/* .

(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( "uw1-320-20", "func", null );
With this statement, the calling agent migrates to uw1-320-20 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( “uw1-320-20”, “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

(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 uw1-320-20 and uw1-320-21:

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

Finally, you can submit your agent (from uw1-320-20 in the above example):

    java -cp UWAgent.jar:. UWAgent.UWInject -p 12345 localhost AnAgent uw1-320-21

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 1: Implementation of UnixClient.java
Given 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
java UnixClient 12345 2 uw1-320-11 uw1-320-12 4 who ls ps df
```

it should call UnixServer's exec( ) method at uw1-320-11 four times, each executing who, ls, ps, and df respectively, and thereafter call exec( ) at uw1-320-12 four times to execute these four commands. Your UnixClient.java program receives a Vector object from each exec( ) call as the outputs of the corresponding Unix command. After invoking all these commands at all the servers, your client program prints out these outputs to System.out.

**Server execution:**

```bash
css434@uw1-320-11:~/hw3/rmi$ rmiregistry 12345&
[1] 22613
css434@uw1-320-11:~/hw3/rmi$ java UnixServer 12345

css434@uw1-320-12:~/hw3/rmi$ rmiregistry 12345&
[1] 3254
css434@uw1-320-12:~/hw3/rmi$ java UnixServer 12345
```

**Client execution:**

```bash
css434@uw1-320-10:~/hw3/rmi$ java UnixClient 12345 2 uw1-320-11 uw1-320-12 4 who ls ps df
```

```
port = 12345, nServers = 2, server1 = uw1-320-11, command1 = who
```

```
============================================================
uw1-320-11 command(who):..............................
css434   pts/12       2016-03-22 15:57 (50.46.157.107)
uw1-320-11 command(ls):..............................
ServerInterface.class
ServerInterface.java
UnixClient.class
UnixClient.java
UnixServer.class
UnixServer.java
UnixServer_Stub.class
uw1-320-11 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
uw1-320-11 command(df):..............................
    Filesystem           1K-blocks Used Available Use% Mounted on
    udev              8178948     4  8178944   1% /dev
    tmpfs            16387441412  1637332   1% /run
    /dev/sda1        464121624 232617124 4172608088 6% /
    none               4      0      4  0% /sys/fs/cgroup
    none            5120      0    5120  0% /run/lock
    none           8193716152  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
```

```
============================================================
uw1-320-12 command(who):..............................
css434   pts/12       2016-03-22 15:57 (50.46.157.107)
uw1-320-12 command(ls):..............................
```
ServerInterface.class
ServerInterface.java
UnixClient.class
UnixClient.java
UnixServer.class
UnixServer.java
UnixServer_Stub.class

uw1-320-12 command(ps):

<table>
<thead>
<tr>
<th>PID</th>
<th>TTY</th>
<th>TIME</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>2961</td>
<td>pts/12</td>
<td>00:00:00</td>
<td>bash</td>
</tr>
<tr>
<td>3254</td>
<td>pts/12</td>
<td>00:00:00</td>
<td>rmiregistry</td>
</tr>
<tr>
<td>3273</td>
<td>pts/12</td>
<td>00:00:00</td>
<td>java</td>
</tr>
<tr>
<td>3305</td>
<td>pts/12</td>
<td>00:00:00</td>
<td>ps</td>
</tr>
</tbody>
</table>

uw1-320-12 command(df):

<table>
<thead>
<tr>
<th>Filesystem</th>
<th>1K-blocks</th>
<th>Used</th>
<th>Available</th>
<th>Use%</th>
<th>Mounted on</th>
</tr>
</thead>
<tbody>
<tr>
<td>udev</td>
<td>8178956</td>
<td>4</td>
<td>8178952</td>
<td>1%</td>
<td>/dev</td>
</tr>
<tr>
<td>tmpfs</td>
<td>1638748</td>
<td>1388</td>
<td>1637360</td>
<td>1%</td>
<td>/run</td>
</tr>
<tr>
<td>/dev/sda1</td>
<td>464121624</td>
<td>23258392</td>
<td>417264128</td>
<td>6%</td>
<td>/</td>
</tr>
<tr>
<td>none</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>0%</td>
<td>/sys/fs/cgroup</td>
</tr>
<tr>
<td>none</td>
<td>5120</td>
<td>0</td>
<td>5120</td>
<td>0%</td>
<td>/run/lock</td>
</tr>
<tr>
<td>none</td>
<td>8193728</td>
<td>148</td>
<td>8193580</td>
<td>1%</td>
<td>/run/shm</td>
</tr>
<tr>
<td>none</td>
<td>102400</td>
<td>40</td>
<td>102360</td>
<td>1%</td>
<td>/run/user</td>
</tr>
<tr>
<td>metis.uwb.edu:/usr/apps</td>
<td>5812624384</td>
<td>1897334784</td>
<td>3622327296</td>
<td>35%</td>
<td>/usr/apps</td>
</tr>
<tr>
<td>metis:/home</td>
<td>5812624384</td>
<td>1897334784</td>
<td>3622327296</td>
<td>35%</td>
<td>/net/metis/home</td>
</tr>
</tbody>
</table>

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. For performance evaluation, use three different servers and execute 4, 8, 12, and 16 commands at each server:

```java
java UnixClient 12345 3 uw1-320-11 uw1-320-12 uw1-320-13 4 who ls ps df
java UnixClient 12345 3 uw1-320-11 uw1-320-12 uw1-320-13 8 who ls ps df who ls ps df
java UnixClient 12345 3 uw1-320-11 uw1-320-12 uw1-320-13 12 who ls ps df who ls ps df who ls ps df
java UnixClient 12345 3 uw1-320-11 uw1-320-12 uw1-320-13 16 who ls ps df who ls ps df who ls ps df who ls ps df
```

Part 2: Implementation of WhoAgent.java

Given 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. The output format should be the same as part1. Similar to UnixClient.java, include time measuring code that measures time elapsed for UnixAgent.java to execute its entire sequence of command executions, and conduct the same performance evaluation, using three different servers and executing 4, 8, 12, and 16 commands at each server.
5. 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 soft copy to “CollectIt for Homework”. Your soft copy should include:
   (1) Your report in PDF or MS Word
   (2) Source code (either within your report or separate .java files)
   (3) Execution outputs (either within your report or separate .jpg/.pdf/.tif/.txt files)
The grader’s preference is all in one report.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Documentation</strong> of your algorithm including explanations and illustrations in one or two pages.</td>
<td>5pts</td>
</tr>
<tr>
<td>(1) UnixClient.java: 2.5pts</td>
<td></td>
</tr>
<tr>
<td>(2) UnixAgent.java: 2.5pts</td>
<td></td>
</tr>
<tr>
<td><strong>Source code</strong> that adheres good modularization, coding style, and an appropriate amount of comments.</td>
<td>5pts</td>
</tr>
<tr>
<td>(1) UnixClient.java: 1.5pts</td>
<td></td>
</tr>
<tr>
<td>a. A use of Naming.lookup and RMI call (0.5pts)</td>
<td></td>
</tr>
<tr>
<td>b. A use of the Date class (0.5pts)</td>
<td></td>
</tr>
<tr>
<td>c. Printing all command outputs at the end (0.5pts)</td>
<td></td>
</tr>
<tr>
<td>(2) UnixAgent.java: 2.5pts</td>
<td></td>
</tr>
<tr>
<td>a. hop( ) called at the end of a function but not inside any for/while loop. (0.5pts)</td>
<td></td>
</tr>
<tr>
<td>b. An agent visiting one to another server rather than going back and forth between the local and each remote host (0.5pts)</td>
<td></td>
</tr>
<tr>
<td>c. An agent returning back to where it was inject (0.5pts)</td>
<td></td>
</tr>
<tr>
<td>d. A use of the Date class (0.5pts)</td>
<td></td>
</tr>
<tr>
<td>e. Printing all command outputs at the end (0.5pts)</td>
<td></td>
</tr>
<tr>
<td>(3) Coding: 1pt</td>
<td></td>
</tr>
<tr>
<td>a. Code completeness (0.5pts)</td>
<td></td>
</tr>
<tr>
<td>b. Coding style and readability (0.5pts)</td>
<td></td>
</tr>
<tr>
<td><strong>Execution output</strong> such as a snapshot of your display/windows or contents of standard output redirected to a file.</td>
<td>5pts</td>
</tr>
<tr>
<td>(1) UnixClient.java: 2.5pts</td>
<td></td>
</tr>
<tr>
<td>a. A correct result when using two servers, each executing four commands. This receives 2.5pts.</td>
<td></td>
</tr>
<tr>
<td>b. A result didn’t use two servers or execute four commands at each server. Or the result included minor errors. This case receives 2pts.</td>
<td></td>
</tr>
<tr>
<td>c. Incomplete results receive 1.5pts</td>
<td></td>
</tr>
<tr>
<td>(2) UnixAgent.java: 2.5pts</td>
<td></td>
</tr>
<tr>
<td>a. A correct result when using two servers, each executing four commands. This receives 2.5pts.</td>
<td></td>
</tr>
<tr>
<td>b. A result didn’t use two servers or execute four commands at each server. Or the result included minor errors. This case receives 2pts.</td>
<td></td>
</tr>
</tbody>
</table>
c. Incomplete results receive 1.5pts

<table>
<thead>
<tr>
<th>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).</th>
</tr>
</thead>
<tbody>
<tr>
<td>(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)</td>
</tr>
<tr>
<td>(2) Performance comparison: summarize your performance results in a table or a graph and discuss under what conditions UnixAgent.java performs better than UnixClient.java. You may tell that UnixAgent.java does not perform well at all, based on your experiment. (2.5pts)</td>
</tr>
</tbody>
</table>

Total 20pts

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

<table>
<thead>
<tr>
<th>Source code</th>
<th>0.5pts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outputs</td>
<td>0.5pts</td>
</tr>
</tbody>
</table>