CSS 533
Program 3: Mobile-Agent Execution Platform

Professor: Munehiro Fukuda

Due date: see the syllabus

1. Purpose

This assignment implements a mobile-agent execution platform that is in general facilitated with three distributed-computing technologies: RPC, dynamic linking, and object serialization/deserialization. We exercise how to use these technologies in Java, which correspond to RMI, class loader and reflection, and Java object input/output streams.

2. Components of Our Mobile-Agent System

The /home/mfukuda/css533/programming/prog3/ directory has Mobile.jar that implemented our mobile-agent systems. Its actual source code and class files are located under the Mobile/ subdirectory whose contents are:

Java programs	Descriptions
AgentLoader.java	Defines the class of an incoming agent and registers it into its local class hash. (It has been already implemented.)
AgentInputStream.java	Reads a byte array from ObjectInputStream and deserializes it into an agent object, using the AgentLoader class. (It has been already implemented.)
Agent.java	Is the base class of all user-define mobile agents. It carries an agent identifier, the next host IP and port, the name of the function to invoke at the next host, arguments passed to this function, its class name, and its byte code. It runs as an independent thread that invokes a given function upon migrating to the next host. (Partially implemented. You must complete the implementation.)
Inject.java	Reads a given agent class from local disk, instantiates a new object from it, and transfers this agent to a given destination IP where the agent starts with the init( ) function. (It has been already implemented.)
PlaceInterface.java	Defines Place's RMI method that will be called from an Mobile.Agent.hop( ) to transfer an agent. (It has been already implemented.)
Place.java	Is the our mobile-agent execution platform that accepts an agent transferred by Mobile.Agent.hop( ), deserializes it, and resumes it as an independent thread. (Partially implemented. You must complete the implementation.)

The goal of HW3 is to complete the implementation of Agent.java and Place.java.

3. Agent Specification

The programming framework of our mobile agents is shown below:

```
1.  import Mobile.*;
2.
3.  public class MyAgent extends Agent {
4.    public String destination = null;
5.
6.    public MyAgent( String[] args ) { // instantiated an ordinary object in local
7.      destination = args[0];
8.    }
9.
10.   public void init( ) {     // dispatched to a node specified with Mobile.Inject
11.     System.out.println( "agent( " + getId( ) + ") invoked init: " );
12.     String[] args = new String[1];
13.     args[0] = "Hello!";
14.     hop( destination[0], "step", args );
```

```
15.   }
16.   public void step( String[] args ) { // dispatched to a user-specified node
17.     System.out.println( "agent( " + getId( ) + ") invoked step: " );
18.   }
19. }
```

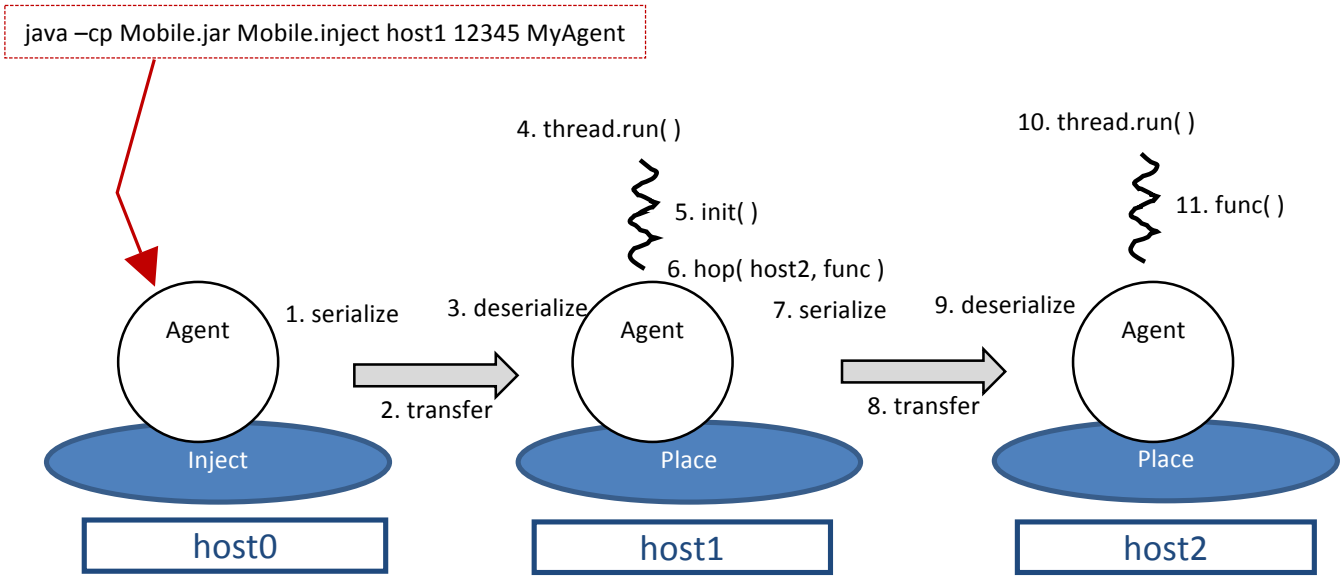
A user must derive an application-specific mobile agent from the Mobile.Agent base class (line 3) that is instantiated as an independent thread at run time. Her/his mobile agent must implement at least two methods: *i) the constructor* receiving a String array (line 6) and ii) *the init( )* method that is invoked right after the constructor execution (line 10). The mobile agent is executed in the following four steps. Its execution is also illustrated below:

Step 1: Injection
The agent is instantiated where a user injects it through the Mobile.Inject program, (i.e. the computing node local to the user), and receives a String array as the constructor argument. At this time, an agent is still an ordinary (passive) Java object.

Step 2: System-initiated migration
Upon an instantiation, the agent is dispatched to the computing node that has been specified with the Mobile.Inject program. Dispatched there, the agent starts to run as an independent thread and automatically invokes its init( ) method. If init( ) has no hop( ) method call, a return from init( ) means the termination of this agent.

Step 3: User-initiated migration
If the agent invokes the *hop(destination, function, arguments)* function within init( ), it will migrate to the next computing-node, (i.e., *destination*) specified in hop( ). Upon each user-initiated migration, the agent will resume its execution as an independent thread and invoke the *function* specified in hop( ).

Step 4: Termination
If the agent returns from the function that was invoked upon a migration (including init( )), the thread to run this agent is stopped and the object is garbage-collected by the system.



Note that our mobile agents are based on so-called weak migration where hop( ) will never return back to the calling function. Although HW3's main focus is hop( ), the Mobile.Agent base class should have the following methods:

Methods			Descriptions
public	**void**	**hop(String host, String function)**	Transfers this agent to a given *host*, and invokes a given *function* of this agent.
public	**void**	**hop(String host, String function, String[] arguments)**	Transfers this agent to a given *host*, and invokes a given *function* of this agent as passing given *arguments* to it.
public	**void**	**run( )**	Is the body of Mobile.Agent that is executed upon an injection or a migration as an independent thread. The run( ) method identifies the *function* and *arguments* given in hop( ), and invokes it. The invoked function may include hop( ) to further transfer the calling agent to a remote host or simply return back to run( ) that terminates the agent.
public	**void**	**setPort(int port)**	Sets a *port* that is used to contact a remote RMI server when migrating there.
public	**void**	**setId(int id)**	Sets this agent identifier, (i.e., *id*).
public	**int**	**getId( )**	Returns this agent identifier, (i.e., *id*).
public static	**byte[]**	**getByteCode(String className)**	Reads a byte code from the file whose name is *className* + ".class".
public	**byte[]**	**getByteCode( )**	Reads this agent's byte code from the corresponding file.
private	**byte[]**	**serialize( )**	Serializes this agent into a byte array.

Agent.run( ) must perform the following tasks:

(1) Find the method to invoke, through this.getClass( ).getMethod( ).
(2) Invoke this method through Method.invoke( ).

Agent.hop(String hostname, String function, String[] args) must perform the following tasks:

(1) Load this agent's byte code into the memory.
(2) Serialize this agent into a byte array.
(3) Find a remote place through Naming.lookup( ).
(4) Invoke an RMI call.
(5) Kill this agent with Thread.currentThread( ).stop( ), which is deprecated but do so anyway.

4. Place Specification

Place behaves as an RMI server to receive incoming agents and to execute them as independent threads. It should have the following methods:

Methods			Descriptions
public static	**void**	**main(String[] args)**	Starts an RMI registry in local, instantiates a Mobile.Place object, (i.e., an agent execution platform), and registers into the registry. The main( ) should receive the port #, (i.e., 5001-65535) to launch its local rmiresitry.
private static	**void**	**startRegistry(int port) throws RemoteException**	Is called from main( ) and starts an RMI registry in local to this Place.
public		**Place( ) throws RemoteException**	Instantiates an AgentLoader object that should be passed to AgentInputStream to deserialize an incoming agent.
public	**boolean**	**transfer(String classname, byte[] bytecode, byte[] entity) throws RemoteException**	Is called from Agent.hop( ) remotely. The transfer( ) method receives this calling agent, deserializes it, sets this agent's identifier if it has not yet been set, instantiates a Thread object as passing this agent to its constructor, (in other words, the agent is an Runnable interface), and invokes this thread's start( ) method. If everything goes well, transfer( ) should return true, otherwise false.
private	**Agent**	**deserialize(byte[] buf)**	Receives a byte array of an agent, and deserializes it from the array.

Place.main(String args[]) must perform the following tasks:

(1) Read args[0] as the port number and checks its validity.
(2) Invoke startRegistry(int port).
(3) Instantiate a Place object.
(4) Register it into rmiregistry through Naming,rebind( ).

Place.transfer(String classname, byte[] bytecode, byte[] entity) must perform the following tasks:

(1) Register this calling agent's classname and bytecode into AgentLoader.
(2) Deserialize this agent's entity through deserialize(entity).
(3) Set this agent's identifier if it has not yet been set. How to give a new agent id is up to your implementation. An example is to have each Place maintain a sequencer, to generate a unique agent id with a combination of the Place IP address and this sequence number, and increment the sequencer.
(4) Instantiate a Thread object as passing the deserialized agent to the constructor.
(5) Invoke this thread's start( ) method.
(6) Return true if everything is done in success, otherwise false.

5. Compilation

Let us assume that you have created a new working directory prog3/.

You need to make sure:

(1) You must also create the Mobile/ directory under prog3/.
(2) AgentInputStream.java, AgentLoader.java, Agent.java, PlaceInterface.java, Place.java, and Inject.java should be stored under prog3/Mobile/.
(3) Your Agent.java and Place.java must include "package Mobile;" at the top of their file.
(4) MyAgent.java, (a simple test program) must be stored under prog3/.
(5) /home/mfukuda/css533/prog3/compile.sh, runPlace.sh, and runAgent.sh should be copied under prog3/

Thereafter, to compile them all, run compile.sh at prog3/

```
#!/bin/sh
javac Mobile/*.java
rmic Mobile.Place
jar cvf Mobile.jar Mobile/*.class
javac -cp Mobile.jar:. *.java
```

You will find Mobile.jar and MyAgent.class, both under prog3/.

6. Execution

(1) Open four terminal windows, two of which are running on the same machine and the other two on a different machine respectively. For example, let us assume that two terminals are running on cssmpi1 and named cssmpi1A and cssmpi1B respectively, whereas the 3rd and 4th terminals are on cssmpi2 and cssmpi3.

(2) Run Mobile.Place on cssmpi1B, cssmpi2, and cssmpi3.

```
java -cp Mobile.jar Mobile.Place 50763
```

You may use runPlace.sh. However, you must change the port number into your own.

```
#!/bin/sh
java -cp Mobile.jar Mobile.Place 50763
```

(3) Inject MyAgent.class from cssmpi1A

```
java -cp Mobile.jar Mobile.Inject localhost 50763 MyAgent uw1-320-11 uw1-320-12
```

You may use runAgent.sh that accepts an IP address to dispatch a given agent from the local host. However, you must change the port number into your own.

```
#!/bin/sh
java -cp Mobile.jar Mobile.Inject localhost 50763 MyAgent $1 $2
```

cssmpi1A

cssmpi1B **cssmpi2** **cssmpi3**

7. Statement of Work

Requirement 1: Implement the following two programs:

(a) Agent.java: our mobile-agent base class
(b) Place.java: our mobile-agent execution platform

Requirement 2: Add a new feature to this mobile-agent system. Example features are:

(a) Direct inter-agent communication (quite difficult, though)
(b) Indirect inter-agent communication via Place (fairly easy)
(c) Child agent creation (moderate)

Requirement 3: Verify the correctness of your Mobile.Agent/Place implementation as well as your own additional feature. For this verification, you must:

(a) Run MyAgent.java over three difference Places as shown in "Section 6. Execution" above.
(b) Write your own TestAgent.java and run it over four different Places for the purpose of verifying the correctness of your additional feature.

8. What to Turn in

This programming assignment is due at the beginning of class on the due date. Please turn in a PDF-formatted report through CollectIt, which includes the following items:

Criteria	**Grade**
Documentation of your implementation of Agent.java and Place.java in one page. Insufficient or too much (i.e., less than 1 page or more than 2 pages) documentation receives 2pts.	3pts
Source code that adheres good modularization, coding style, and an appropriate amount of comments. *Agent.java* 1pts: Agent.run( ) invokes a given method without any arguments. 1pts: Agent.run( ) also invokes a given method with some arguments. 1pts: Agent.hop( ) invoke the destination node's transfer( ) function. *Place.java* 1pt: Place.main( ) instantiates and register a Place object into the local RMI registry. 1pt: Place.transfer( ) deserializes and executes an incoming agent. 1pt: Place.transfer( ) gives a unique identifier to the incoming agent. *TestAgent.java* 1pt: TestAgent.java verifies your additional feature Additional feature(s) 1pt: Implemented in Agent.java and/or Place.java. 2pts: Good modularization, coding, style, and an appropriate amount of comments throughout your code (1pt: fair, 0pts: terrible)	10pts
Execution output that verifies the correctness of all your implementation. 2pts: A snapshot of MyAgent.class execution with three different Places 2pts: A snapshot of TestAgent.class execution with four different Places	6pts

2pts: A snapshot of TestAgent.class to demonstrate your own additional agent/place feature	
Discussions about your own additional agent/place feature, limitations, and possible improvements of your program in one or two pages (firm). 4pts: Good discussions (including limitations and possible improvements) with clearly mentioning the new feature(s) you added to your mobile-agent execution platform 3pts: Good discussions (including limitations and possible improvements) but no new feature added to the original specification. 2pts: Unremarkable discussions without any limitations/possible improvements nor any new feature added to the original specification.	4pts
Lab Sessions 6 and 7 counts 1pt for each. Please include these lab exercises in your program 3's report	2pts
Total Note that program 2 (including Lab 3) takes 14% of your final grade.	25pts