Java Programming
Unit 10

Stock Price Quotes with URL, Sockets, and RMI

(c) Yakov Fain 2014

Getting Stock Quotes From Yahoo!

1. Visit http://finance.yahoo.com, enter AAPL - the symbol of the Apple's stock, and press the button Get Quotes .
   Note the URL. Change it to http://finance.yahoo.com/q?s=AAPL

2. To get the price quote **programmatically** create an instance of the URL object and open the stream:

```
url   = new URL("http://finance.yahoo.com/q?s=AAPL");
   InputStream in = url.getInputStream();
   BufferedReader buff= new BufferedReader(new InputStreamReader(in));
```

One way of getting the stock info from Yahoo! is to read the entire Web page:

```
String theWholePage;
  Srting txt;
  while (txt =buff.readLine() != null ){
       theWholePage=theWholePage + txt;
   }
```

…and then parse it trying to find the fragment with the AAPL price.

An alternative URL for stock quotes

The previous solution produces lots of text to filter out. Here you can get the quotes for AAPL or any other stock symbol in the CSV format:

http://quote.yahoo.com/d/quotes.csv?s=**AAPL**&f=sl1d1t1c1ohgv&e=.csv

The classes `StringTokenizer` or `Scanner` can help you in parsing the received `String` with comma-separated values.

The textbook has a sample code (pg. 191 – 193) that uses the stock symbol **MOT**, which used to represent Motorola. Since the MOT stock symbol has been recently modified, use **MSI** for Motorola or any other valid stock symbol, e.g. **AAPL** for Apple.

Walkthrough 1

- Review with the instructor the code of the program StockQuote.java from Eclipse project Lesson18
- Go to the Eclipse menu Run Configurations and enter **AAPL** in the field Program Arguments
- Run the program and observe the current price of the stock
- Change the stock symbol to be **IBM** and re-run the program to see the latest price of IBM stock
- Run the same program through a debugger
  placing a breakpoint at line 26.
  Debug the program and watch the work
  of the class `StringTokenizer`

Programming Sockets

What's Socket

The package `java.net` includes classes `Socket` and `ServerSocket`.

A socket is a *connection end-point* in IP networking.

The *TCP/IP protocol* maintains a socket connection for the whole period of communication, while *UDP* is a connectionless protocol, which sends data in small chunks called *datagrams.*

The socket address is a pairof IP address and port.

An instance of the `ServerSocket` class becomes
 a server that listens to the specified port
 for requests.

Sockets: the Client and the Server

The following two lines **create a server** that is listening to port 3000:

```
ServerSocket serverSocket = new ServerSocket(3000);
Socket client = serverSocket.accept();
```

The client program creates an instance of the class Socket pointing at the computer/port on which the ServerSocket is running:

```
Socket clientSocket = new Socket("124.67.98.101", 3000);
```

WebSocket is an new HTML5 standard of communication over the Internet. More details here:
http://enterprisewebbook.com/ch8_websockets.html

Getting Stock Quotes with Sockets

JVM 1 (the server):

1. Start the ServerSocket on some port
```
new ServerSocket(3000);
```
2. Put it in a listening mode with accept().
3. Process the request and return the result to the client via the stream.

JVM 2 (the client):

1. Connect to the server socket by instantiating the class Socket
```
clientSocket = new Socket("124.67.98.101", 3000);
```
2. Get the reference to the server's stream
```
outbound = clientSocket.getOutputStream();
```
3. Send your requests to this stream.
4. Process server's responses

Walkthrough 2

- Review with instructor the code of the program StockQuoteServer and Client from Eclipse project Lesson18

- Follow the instructions from the TRY IT section to Lesson 18 to test the client-server communications using sockets.

Non-Blocking Sockets

If our stock server needs to process multiple requests, the StockQuoteServer would need to create a new thread for each request. Each thread introduces an overhead limiting the number requests that can be processed concurrently.

The package java.nio includes a number of classes that support non-blocking i/o in general and non-blocking sockets in particular. which allows i/o communications on the socket channel without blocking the processes using it.

You can read a short tutorial on non-blocking I/O by Jacob Jenkov: http://tutorials.jenkov.com/java-nio/index.html

Remote Method Invocation (RMI)

RMI allows JVMs communicate with each other.

With sockets, the Java client was directly connecting to Java server running on a different JVM.

With RMI, Java client will make a method call *that looks as if this method is running in the same JVM*, but it's not. Only a *proxy* (*a stub*) of the remote method exists in the client's JVM.

Finding Remote Objects

RMI clients find remote services by using a naming service, which must run on a known host and port number.

The RMI server can start its own *registry* that offers naming services for RMI clients. The behavior of the registry is defined by the interface `java.rmi.registry.Registry`

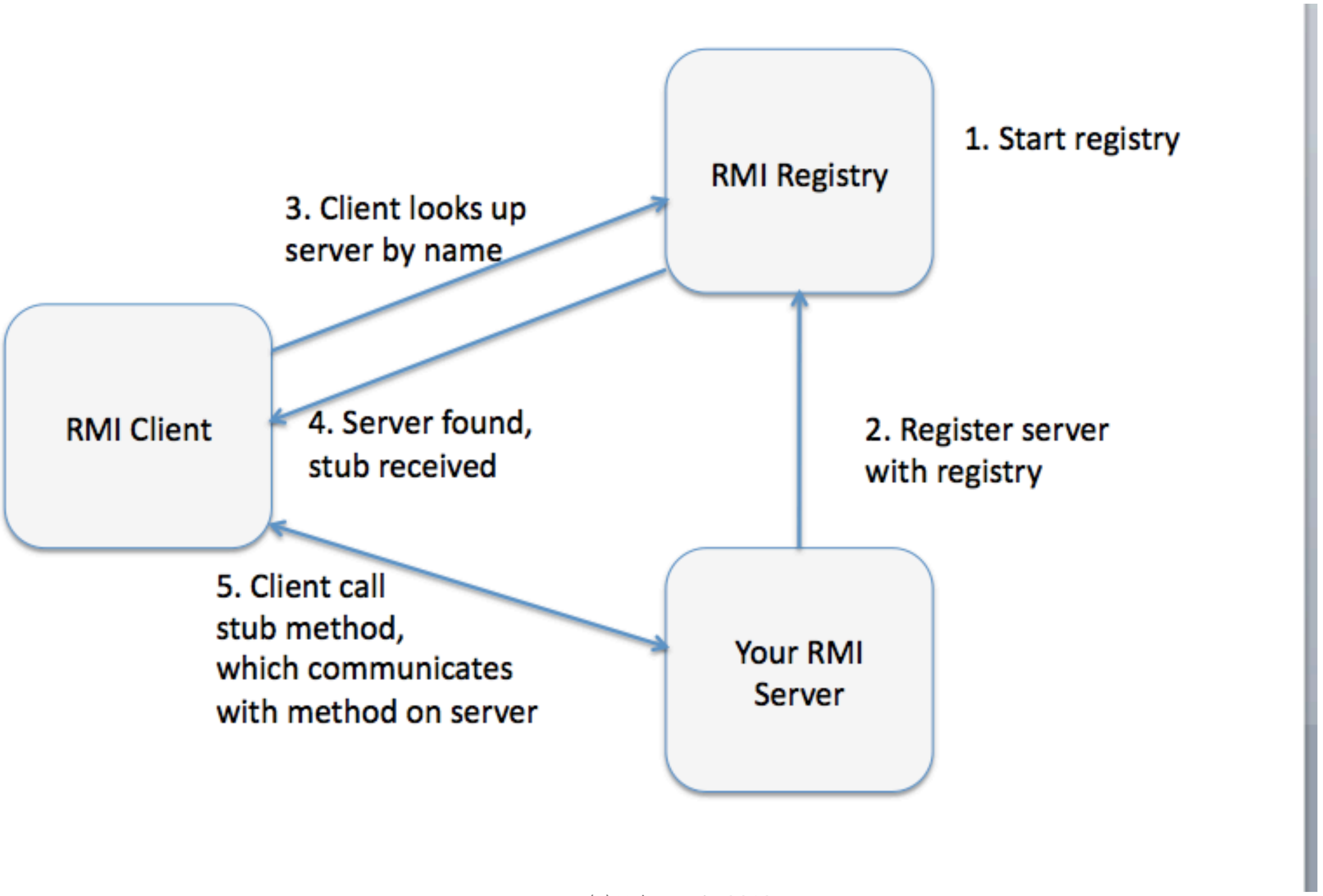
By default, the RMI registry runs on port 1099

The client obtains a reference to a remote object by looking up its name in the registry. This lookup returns to the client a remote reference a.k.a. **stub**.

The method `lookup()` takes the service name URL as an argument in the following format:

rmi://<host_name>[:<name_service_port>]/<service_name>

RMI Players


RMI Registry
1. Start registry
3. Client looks up server by name
RMI Client
4. Server found, stub received
2. Register server with registry
5. Client call stub method, which communicates with method on server
Your RMI Server

Developing and Running an app using RMI

- Declare a remote Java interface

- Implement the remote interface in a Java class

- **Computer A:** Start the registry and register the RMI server with it

- **Computer B:** Start the server

- **Computer C:** Start a Java client that will look up the service in the registry A to get stubs from the server B and calls remote methods.

(c) Yakov Fain 2013

Walkthrough 3

1. Download and import the project Lesson25 and review the code with the instructor

2. Add the following statement at the line 12 of StartServer.java:
`LocateRegistry.createRegistry(1099);`

3. Add the import statement for `LocateRegistry`

4. Run StartServer and it should give a prompt
<QuoteService> server is ready

5.Run configuration for Client.java to specify one program argument: AAPL

6. Run the Client and you should get the random price quote like
The price of AAPL is: $1.3335365174267477

Homework

1. Study the materials from Lesson 18 and lesson 25 from the textbook.

Additional Reading/Watching

- Read about working with Yahoo stock quote data feed http://www.gummy-stuff.org/Yahoo-data.htm .

- RMI: http://docs.oracle.com/javase/tutorial/rmi/

- Watch the video on code refactoring and testing: http://www.java-tv.com/2013/10/23/testing-and-refactoring-legacy-code-2/

- Watch the video about automated testing: http://www.infoq.com/presentations/Testing-Java.
  Create a project that uses unit tests with JUnit.