Lab: Remote Method Invocation

Alexander Hoffmann

March 21, 2020

RMI stands for Remote Method Invocation. It is a mechanism that allows an object residing in one system (JVM) to access/invoke an object running on another JVM. RMI is used to build distributed applications; it provides remote communication between Java programs.

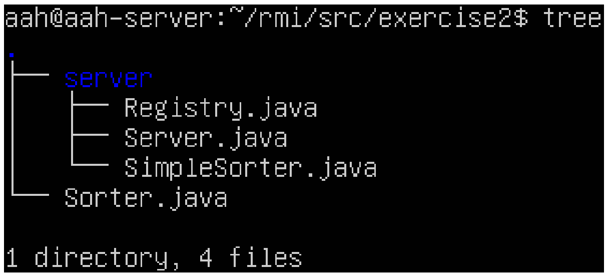
1 Local execution

Q1. The package contains three main methods in `Registry.java`, `Server.java` and `Client.java`.
Q2. The Java processes will be executed until terminated manually. This means, even if the main methods of each and every class is done, the processes will keep on running.
Q3. First, run `Registry.java` as is it referenced in `Server.java`. Next, run `Server.java`. Finally, run `Client.java`.
Q4. If we run `Registry.java` alone, everything works as it should because this is the process to execute first. Now suppose we run `Server.java` alone. We are getting an error. The console states that the hostname is undefined. This is because the host is initialized in the Registry class. Finally, if we try to execute `Client.java` independently, this app crashes as the local server cannot send the `Object` to the client.
Q5. 1. Suppose we launch `Registry.java`, then `Server.java` and stop `Registry.java`. Now if we run `Client.java`, the process crashes because of a `ConnectionRefused` exception.
2. Now `Registry.java` has been restarted. We try to run `Client.java` but it crashes because of an `NotBoundException`. The server has to rebind with the new registry.

3. Restart the server which will now bind. If we run the client now, it works fine.
4. Finally, restart the server and try again to launch the client. It works.
Q6. The client needs the server which needs to bind to the registry. The client is direclty dependent of the server and indirectly dependent of the registry. The server directly depends on the registry. The registry is completely independent.

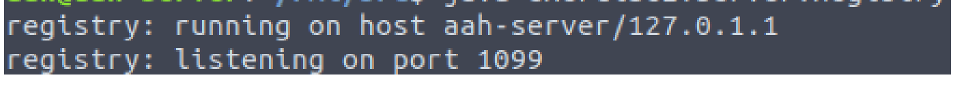
2 Distributed execution

Q1. The server side classes should be deployed on one machine whereas the client side classes should be executed on the other machine.
Q2. `Registry.java`, `Server.java`, `Sorter.java` and `SimpleSorter.java` are stored on the server machine. `Client.java` is executed on the client machine.
Q3. Since we are not running the server locally anymore, we need to change the IP address of the server to be accessible by the other machine on the network.
Q4. We create a virtual machine running an Ubuntu server where we deploy the server side of the application. Here is the file structure on the server.



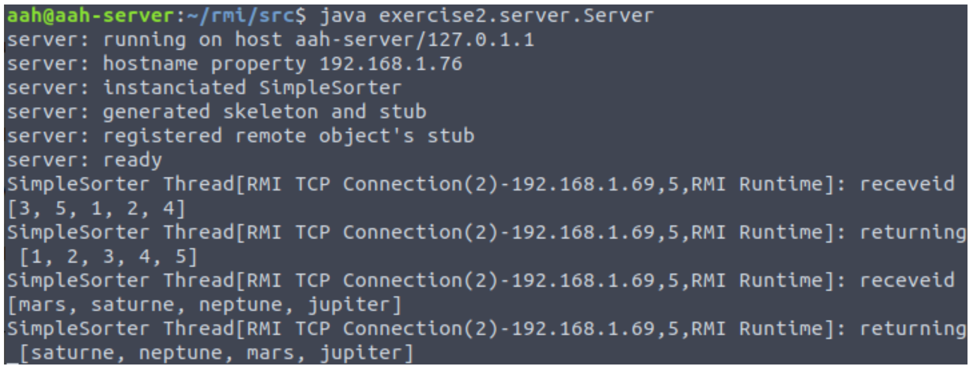
```
aah@aah-server:~/rmi/src/exercise2$ tree
.
├── server
│   ├── Registry.java
│   ├── Server.java
│   └── SimpleSorter.java
└── Sorter.java

1 directory, 4 files
```

On the client, change the IP address in the constant `SERVICE_HOST` to `192.168.1.76` which corresponds to the IP address of the server.
After all is set and done, run `Registry.java` on the server.



```
aah@aah-server:~/rmi/src$ java exercise2.server.Registry
registry: running on host aah-server/127.0.1.1
registry: listening on port 1099
```

And `Server.java`. Then run `Client.java` on the host. Here is the output on the console of the server.



```
aah@aah-server:~/rmi/src$ java exercise2.server.Server
server: running on host aah-server/127.0.1.1
server: hostname property 192.168.1.76
server: instanciated SimpleSorter
server: generated skeleton and stub
server: registered remote object's stub
server: ready
SimpleSorter Thread[RMI TCP Connection(2)-192.168.1.69,5,RMI Runtime]: receveid
[3, 5, 1, 2, 4]
SimpleSorter Thread[RMI TCP Connection(2)-192.168.1.69,5,RMI Runtime]: returning
 [1, 2, 3, 4, 5]
SimpleSorter Thread[RMI TCP Connection(2)-192.168.1.69,5,RMI Runtime]: receveid
[mars, saturne, neptune, jupiter]
SimpleSorter Thread[RMI TCP Connection(2)-192.168.1.69,5,RMI Runtime]: returning
 [saturne, neptune, mars, jupiter]
```

The rest of the lab is described in the source code provided in the compressed folder.