Remote Method Invocation (RMI)

- RMI involves the use of remote code to perform functionality.
- It is accomplished by defining a remote interface that is implemented on a server.
- A module of code implementing the remote interface on the client side is created and is called a stub.
- The client calls the function in the stub, which in turn communicates with the server, which then calls the function on the server and returns the result.

Steps to Creating an RMI system

1. Create a remote interface by extending the Remote interface. All remote methods are capable of throwing a RemoteException, so it must be included in the prototype of the method. Example:

```
public interface RemoteAdd extends Remote {
   int add( int number1, int number2 ) throws
 RemoteException;
}
```

2. Implement the remote methods for a server. The remote interface can be implemented in a class and then a static method of the UnicastRemoteObject class used to create a remote object stub, or the UnicastRemoteObject can simply be extended and the interface implemented.

3. Deploy the remote implementation:
   a) In the server, create a SecurityManager; RMI will not run unless a SecurityManager is running.
   b) Then, obtain a reference to the RMI Registry.
   c) Finally, bind the object containing the remote methods to a designated name in the RMI Registry.

4. Use the remote object in a client:
    a) In order to create a client, start by creating a SecurityManager.
    b) Next, obtain a reference to the RMI Registry that is running on the server's host.
    c) Using the Registry object, obtain a reference to the remote object.
    d) Finally, call the remote methods of the remote object.
5. Create a permissions file for the server and one for the client. Place each permissions file in the same directory as the executable Java bytecode.
6. Start the RMI registry on the server's host. Either start it in the directory that contains the remote classes or add the classpath as a codebase argument to the command to start the RMI registry.
7. Start the server, defining the codebase where the remote interface is located and the name of the policy file for the SecurityManager to use.
8. Start the client, defining the name of the policy file for the SecurityManager to use.

- Custom sockets can be created using a socket factory
- Code examples are available on D2L

Links

RMI home page: contains links to articles of various levels of depth on the topic.

Java RMI tutorial

The codebase property-- see diagrams

Change in codebase property-- Java 7