My Lottery: A prototype of an online lottery application

K. M. Sabidur Rahman@01343723

Md. Mejbah ul Alam@013

Department of Computer Science

University of Texas at San Antonio

[wmr727@my.utsa.edu](mailto:wmr727@my.utsa.edu)

@my.utsa.edu

**Abstract**

**This project is to be submitted for Fall 2012 Operating Systems course instructed by Dr. Dakai Zhu. [1] The target of the project is to learn and practice distributed application and systems and client-server architecture using middlewares, socket programming and multithreaded programming.**

**1. Introduction**

The middleware is a software layer on top of the operating system that uses its facilities, integrates with it, and extends its functionality in order to support the development of effective and reliable distributed systems. Examples of middleware include Remote Procedure Calls, Remote Method Invocations, Distributed File Systems, Distributed Object Systems etc.

Remote objects are the shared state of a distributed program as a collection of objects and can receive remote invocations. In this project we implement the middleware to support remote objects for the Object Request Broker (ORB) architecture.

The objective of this project is following [1]:

* enhance the understanding of distributed applications and systems
* enhance the understanding of client-server architecture of distributed systems
* enhance the knowledge of middleware to support remote objects and RMI
* practice multi-thread programming
* practice network and socket programming
* Learn to write scientific project report

We have completed all the requirement of the project and also performed the extra credit work- implementation of persistent ClientAccount Objects. The special thing about this project was the fact that it was a group project. The workload was devided equally, **K. M. Sabidur Rahman** has worked with the Naming Server and Lottery Servant Server and Lottery Servant Server proxy part on Lottery Portal Server and **Md. Mejbah ul Alam** has worked with the Lottery Portal Server and Client program.

**2. Project Description**

In this project, there are four major entities, namely Naming Server, Lottery Servant Server, Lottery Portal Server and Client program. On the Naming Server, a table of bindings (names vs. RORs) is maintained. Three operations are supported by the server: registration(), delete() and lookup().

Lottery Servant Server is responsible for two remote object: MegaMillion and PowerBall. Both the objects provides following remote methods: sellTicketQuickPick(), sellTicketManual(), getPastjackpotNumbers() and checkWinner().

After selling 500 tickets, each object re-starts a new round and before that, selects a jackpot number among the sold tickets. [7]

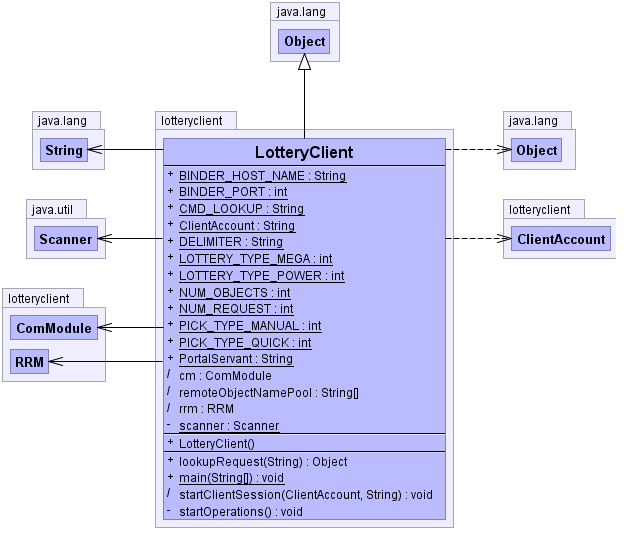
Lottery Portal Server manages PortalServant object which maintains a list of client accounts and handles following functions: openAccount(), login(0, closeAccount(). Client accounts are also remote objects and are provided with following functions: addBalance(), checkBalance(), buyTicket() and getHistoryTickets().

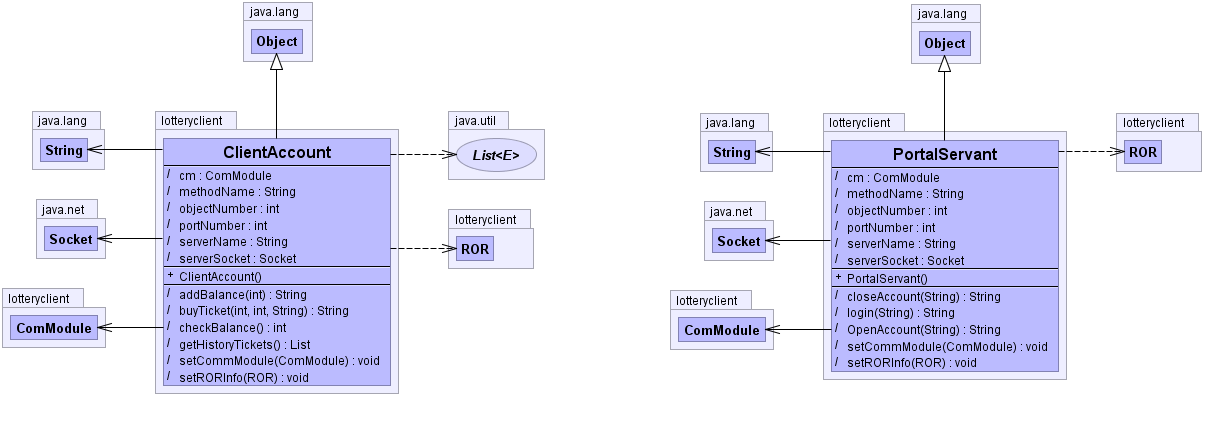
Lastly, Client program obtains the ROR of the PortalServent remote object from Naming Server. It allows the user to manage account, buy tickets and check winning history.[1]

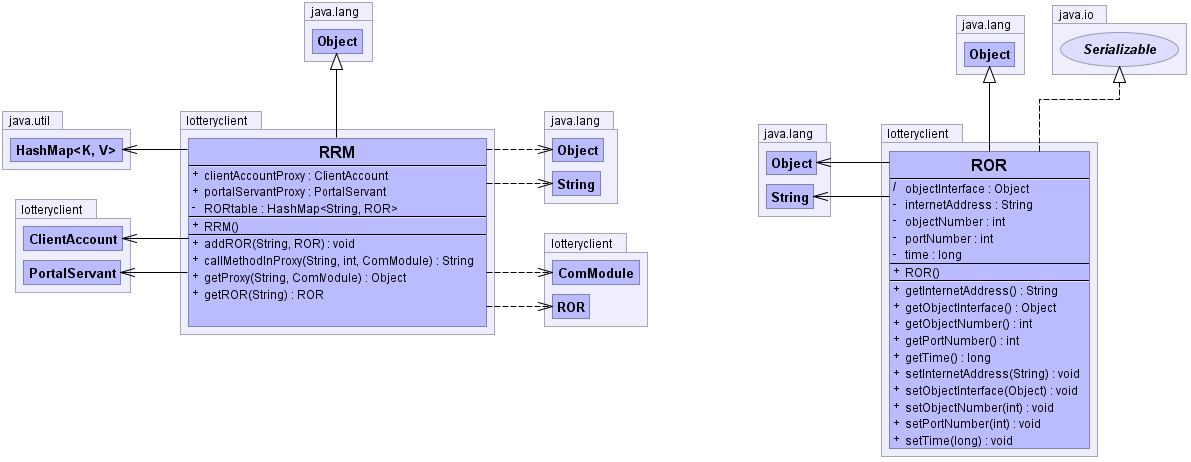
One of the requirements of the project is to handcraft proxy and skeleton of the remote objects.

**3. Design**

3.1 Client Program

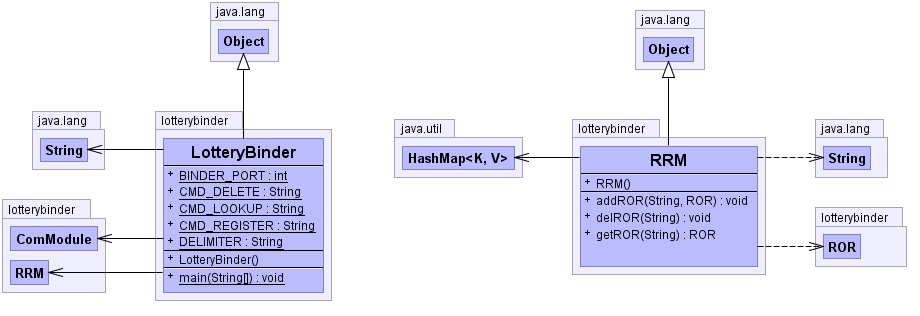






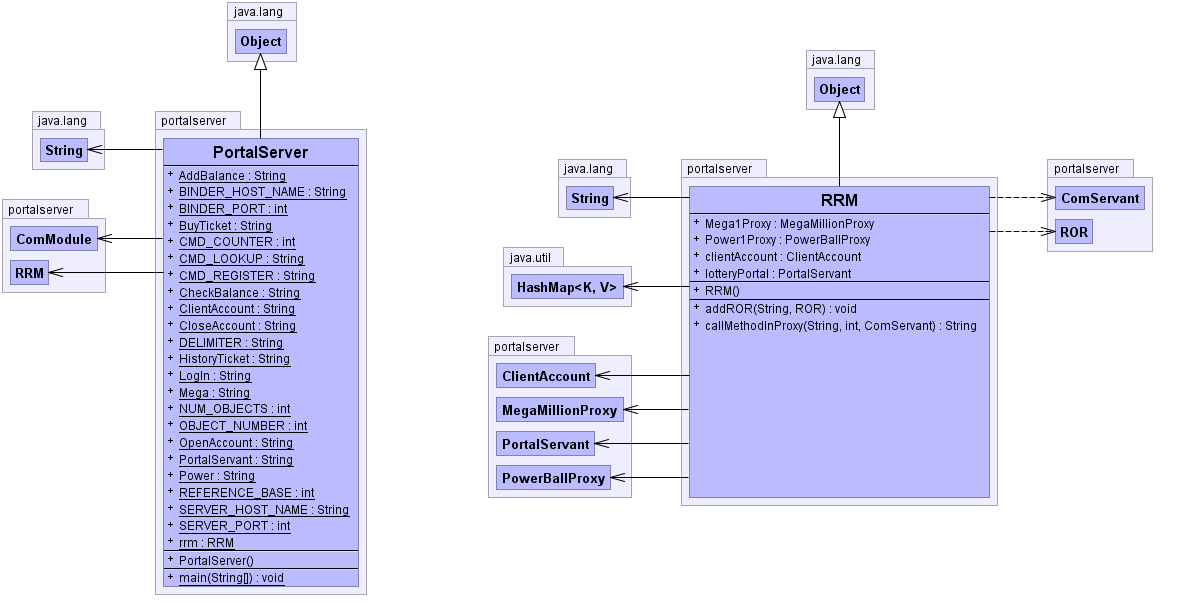
**Figure 1: Client program.**

3.2 Naming Server



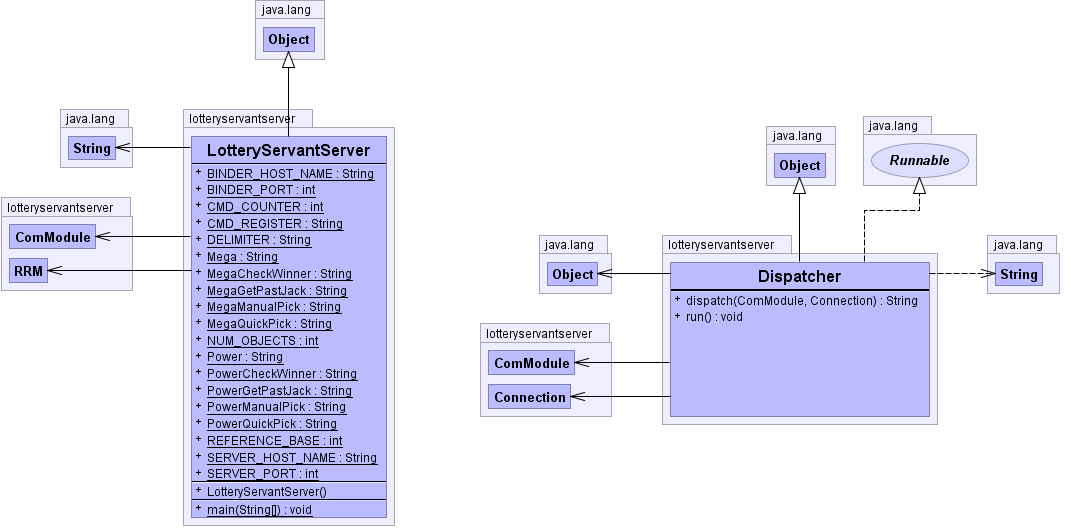
**Figure 2: Namming Server.**

3.3 Lottery Portal Server



**Figure 3: Lottery Portal Server.**

3.4 Lottery Servant Server



**Figure 4: Lottery Servant Server.**

**4. Technical** **Content**

4.1 Sockets

A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program. The java.net package provides two classes a) Socket and b) ServerSocket that implement the client side of the connection and the server side of the connection, respectively. [2]

In server side, after opening the socket server listens to the socket using accept() function and it creates an object of java.io.ObjectInputStream and another object of java.io.ObjectOutputStream to perform read-write to the client.

In JAVA, Client and server both entity use readObject() and writeObject() method of these streams to send requests/replies to each other.

4.2 Synchronization

Synchronized statements are useful for improving concurrency with fine-grained synchronization. A way to create synchronized code in Java is with synchronized statements. Synchronized statements must specify the object that provides the intrinsic lock:

double myFunc() {

synchronized (this) {

counter++;

}

return counter;

}

In this example, the myFunc method synchronizes changes to counter. In this project, different threads accessing the same remote object is synchronized by their methods having synchronized statements.

4.3 Threads

Threads share the process's resources, including memory and open files. Multithreaded execution is an essential feature of the Java platform.

An application that creates an instance of Thread must provide the code that will run in that thread. There are two ways to do this. Both the ways are used in this project. For the main project, threads are implemented by “extends Thread”.

Public class Xclass extends Thread

{

public void run()

{// do the task

}

}

Another way of implementing thread which is used for implementing the Thread Pool is to make a class of Runnable type.

Public class YClass implements Runnable

{

public void run()

{// do the task

}

}

4.4 Java Serialization

Serialization is the process of converting a data structure or object state into a format that can be stored (for example, in a file or memory buffer, or transmitted across a network connection link) and "resurrected" later in the same or another computer environment. [8] In this project ROR classes implements java.io.Serializable and thus we can read-write the objects of these classes directly to the input and output stream.

4.5 Java RMI

RMI applications often comprise two separate programs, a server and a client. A typical server program creates some remote objects, makes references to these objects accessible, and waits for clients to invoke methods on these objects. A typical client program obtains a remote reference to one or more remote objects on a server and then invokes methods on them. RMI provides the mechanism by which the server and the client communicate and pass information back and forth. Such an application is sometimes referred to as a distributed object application. [9]

4.6 Remote Interfaces, Objects, and Methods

Like any other Java application, a distributed application built by using Java RMI is made up of interfaces and classes. The interfaces declare methods. The classes implement the methods declared in the interfaces and, perhaps, declare additional methods as well. In a distributed application, some implementations might reside in some Java virtual machines but not others. Objects with methods that can be invoked across Java virtual machines are called remote objects.

An object becomes remote by implementing a remote interface, which has the following characteristics:

A remote interface extends the interface java.rmi.Remote.

Each method of the interface declares java.rmi.RemoteException in its throws clause, in addition to any application-specific exceptions.

RMI treats a remote object differently from a non-remote object when the object is passed from one Java virtual machine to another Java virtual machine. Rather than making a copy of the implementation object in the receiving Java virtual machine, RMI passes a remote stub for a remote object. The stub acts as the local representative, or proxy, for the remote object and basically is, to the client, the remote reference. The client invokes a method on the local stub, which is responsible for carrying out the method invocation on the remote object.

A stub for a remote object implements the same set of remote interfaces that the remote object implements. This property enables a stub to be cast to any of the interfaces that the remote object implements. However, only those methods defined in a remote interface are available to be called from the receiving Java virtual machine.

**5. Methodology**

5.1 System Configuration

The following system is used for the performance measurement of processes and threads.

* Operating System

Windows 7 (64 bit)

* Hardware

Processor: Intel(R) Core(TM) i3-2330 CPU @ 2.20GHz

Installed Memory: 4.00GB

5.2 Workloads

The implementation is tested with concurrent 3 clients invoking the remote method in both the Lottery Portal server and Lottery Servant Server. The implementation can handle as many as clients needed.

6. **Extra Features**

This brings us to the extra features. Client Accounts objects are made persistent, i.e. the client information can survive server or client restart. And more over all parts of the project are implemented with a perfect team work.

**7.** **Results**

The client can use remote objects successfully to manage accounts and to buy/check tickets. The whole design of the project follows the RMI steps, also the proxy and skeletons are handcrafted as required.

**8. Discussion and Problems faced**

While creating the ticket we need to synchronize the whole process, to ensure the uniqueness of tickets. Also to ensure the number of tickets in a round, we need to synchronize the access to ticket table.

In our opinion, the hardest part of the project is to design it following proper architecture. Once the architecture – class/methods are finalized, implementation becomes fairly straight forward. So, we spent more time in design than in implementation.

One of the problems we faced during the implementation is “ClassNotFound” error due to java hierarchy. [10] We solved the issue by avoiding the use of java packaging.

The project is a necessary one for the hands on of basic ideas of distributed system and RMI and also multi threaded client-server architecture. The handcrafting of RMI modules specially the binder, skeletons and proxy has been helpful to understand the architecture. But, it would be better if the requirements were little bit more illustrated. Over all, authors think that it was a very learning and engaging project.

**9. Conclusion**

This project has been a convenient away of learning implementation of distributed application and system, client-server architecture and incorporating threading and synchronization with them. The author would like to thank instructor Dr. Dakai Zhu for assigning such an effective project.

**10. References**

[1] http://www.cs.utsa.edu/~dzhu/

[2]docs.oracle.com/javase/tutorial/networking/sockets/index.html

[3] <http://java.dzone.com/news/java-concurrency-thread-pools>

[4]<http://javarevisited.blogspot.com/2011/11/static-keyword-method-variable-java.html>

[5]<http://www.java-tips.org/java-se-tips/java.util.concurrent/pooling-threads-to-execute-short.html>

[6]<http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ExecutorService.html>

[7] [www.txlottery.org](http://www.txlottery.org)

[8] http://www.parashift.com/c++-faq-lite/ serialization.html

[9] <http://docs.oracle.com/javase/tutorial/rmi/index.html>

[10] http://javaeesupportpatterns.blogspot.com/2012/06/javalangnoclassdeffounderror-how-to.html