SOEN 423

Assignment 1

Design Specification

Prepared By:

Jesse Tsang (ID: 25692636)

Contents

Summary

This assignment utilises Java's Remote Method Invocation (RMI) API to implement a simple distributed banking system. The actors in this program consist of bank customers and bank managers. Bank managers can create, modify customer accounts, as well as all the customer's operations. Bank customers can deposit, withdraw, and check their balance. There will be four branches available; British Columbia (BC), Manitoba (MB), New Brunswick (NB), Quebec (QC).

Customer records will be stored in serval lists in a hash map according to the first letter of their last name.

Program Architecture

The basic idea is that the client and the server will both be Java Virtual Machine (JVM). They can communicate to each other via remote methods using two intermediary called stub and skeleton.

This program uses the unicast object. In unicast, the caller send the data stream to a single receiver at a time, so it will be a one-to-one communication.

Stub and skeleton are essentially the same thing; stub is client-side and skeleton is server-side. Stub/skeleton is a mechanism to communicate with remote objects. The user invokes a method on their local stub, then the stub will carry out the actual method call on the remote object.

When a stub's method is invoked, the following process will happen:

1. Initiates a connection (with a remote object identifier and a port number) with a remote JVM containing the remote object.
2. Marshals (writes and transmits) the parameters to the remote JVM.
3. Waits for the result of the method invocation from the remote JVM.
4. Unmarshals (reads) the return value or exception.
5. Pass the return value to the caller.

When a skeleton receives an incoming method invocation, the following process will happen:

1. Unmarshals (read) the parameters for the remote method.
2. Invokes the method on the actual remote object implementation
3. Marshals (writes and transmits) the return value or exception to the caller.

RMI Architecture

The RMI architecture consists of three layers; stub/skeleton layer, remote reference layer, and the transport layer. These layers are directly below the application layer.

Stub/Skeleton Layer

The stub/skeleton layer is acting as an interface for the application layer and the rest of the RMI layers. This layer pass the information to the remote reference layer through marshalling of streams. These streams will also perform object serialization.

Remote Reference Layer

The remote reference layer perform remote reference protocol which is independent of stubs and skeletons. The main purpose of this layer is to perform the semantics of the invocation. For example, determining if the server is a single object implementation or a replicated object that required communication with multiple locations. Depends on the remote object implementation, this layer will choose its own remote reference semantic.

Transport Layer

The transport layer performs the operations with the remote address, including initializes, manages and monitors the connection, and listens for any incoming calls. This layer also maintains a remote object table to keep track of remote objects within the transport layer's address space. Furthermore, this layer (on the server side) will also locate the dispatcher for the target of the remote call, then pass along the connection to this dispatcher.

RMI Advantages and Disadvantages

Advantages

- Abstractions: Invocation a remote method is just like invoking a method from a local machine and the stub will hide the serialization of parameters and the network-level details from the caller.

- Distributed Systems:
    - RMI handles threads.
    - RMI decouples client and server objects.

- Object-orientated: Objects can be passed as return value or as parameter, therefore beside primitive values, complex data structures can also be passed, such as hash table or array list.

- Ease of Maintenance: Server-side implementation can be changed without changing the client-side implementation.

Disadvantages

- Overheads:
    - The stub/skeleton layer performs marshaling and unmarshaling will cause overhead for the program performance.
    - Serialization is another source of overhead. Java will need to convert the data to byte stream and revert back to an object every time an object is send or store over the network.

- Performance: In this assignment, connection is done via sockets, however, many real-life scenarios will prevent sockets to be used due to firewalls. In that case, RMI has to be used with HTTP protocol, and that will be magnitude slower than connection through direct sockets.

- Unclear Object Type: Local and remote objects will be mixed together and it will be hard to differentiate which is local and which is remote.

Program Structure

This program consists of four main packages: common, domain, server and client.

- Common Package:
  - This package consists of the remote objects for the RMI.
  - Remote Interface: BankServerInterface.java; this interface consists of all the required operation detailed by the assignment.
  - Interface Implementation: BankServerImpl.java; this class is the implementation of the above mentioned interface. The server object will be an instance of this class.

- Domain Package:
  - This package consists of the core data object for the program.
  - Client.java; this class is a representation of a client. It holds the personal information about the client including first name, last name, address, and balance etc.
  - Enum Classes:
    - BranchID.java; this enum class provides and limits all possible options as the bank's branch ID.
    - EditRecordField.java; this enum class provides and limits all possible options as the possible record modify option, which is used to modify client's information.

- Servers Package:
  - This package consists of the drivers for the server instances. Each bank should have its own instance.
  - ServerDriver.java; a simple driver class to create an instance of BankServerImpl.java.

- Client Package:
  - This package consists of the client class and its driver.
  - AdminClient.java; this class consists of the operations perform by the managers of the bank.
  - CustomerClient.java; this class consists of the operations perform by the clients of the bank.
  - AdminDriver.java; a simple driver class to create an instance of the AdminClient.java.
  - CustomerDriver.java; a simple driver class to create an instance of the CustomerClient.java.

Data Structures

There are two main data structures used in this assignment; client class and the Java's HashMap data structure.

Client Class

This data structure represents the user account of the bank. It consists the following variables and operations.

- Main Variables:
  - String firstName
  - String lastName
  - String address
  - String phone
  - String customerID
  - BranchID branchID
  - double balance

- Main Operations:
  - public void deposit(double amount)
  - public void withdraw(double amount)

Java HashMap

This data structure is used to store client objects in the server. The way the hash key is chosen is by customer's first letter of their last name, which is part of the customer's customer ID. The value of the key-value pair would be the client data structure.

This customer ID is deviated from the assignment specification because the original structure of the customer ID was [branchID][Customer_Type][3 digits]. However, this way when performing the client verification operation such as checking if the client already exists will not use the HashMap's strength because we cannot know the customer's first letter of their last name by their customer ID.

Program Flow

1. The server host will start the ServerDriver.java and an instance of the BankServerImlp.java will be created.
2. The server object will then bind itself to a remote object registry using its branchID and a port number.
3. The client host (be it admin or customer client) will start their respective driver class.
4. The client host will get the server object reference from the registry.
5. The client host will invoke methods in the remote object as if they were local methods.

References:

1. How Java RMI Works (in 5 Pages or Less), http://www.sce.carleton.ca/netmanage/simulator/rmi/RMIExplanation.htm
2. RMI System Architecture, https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/guide/rmi/spec/rmi-arch.doc.html
3. Oracle – 3.1 Stubs and Skeletons, https://docs.oracle.com/javase/7/docs/platform/rmi/spec/rmi-arch2.html