Request for Comment

Date: 01/20/2018
Author: Gareth Sharpe
ID: 090361370
Contact: shar1370 at mylaurier dot ca
Last Edited: 01/20/2018
Status: Proposed

1. Overview

This document outlines the development of a client server application using the Java stream socket API. This multi-threaded application supports the work of a collection of clients on a bibliography data structure containing client-defined books. This application follows HTTP Methods for CRUD API Services as a subset of RESTful API services.

2. Definitions

Client: A client is a piece of computer hardware or software that accesses a service made available by a server. The server is often (but not always) on another computer system, in which case the client accesses the service by way of a network. The term applies to the role that programs or devices play in the client–server model.

Server: A server is a piece of computer hardware or software that provides functionality for other programs or devices, called "clients". This architecture is called the client–server model, and a single overall computation is distributed across multiple processes or devices. Servers can provide various functionalities, often called "services", such as sharing data or resources among multiple clients, or performing computation for a client. A single server can serve multiple clients, and a single client can use multiple servers.

Service: A service is an abstraction of computer resources. The client of a service does not have to be concerned with how the server performs while fulfilling the request and delivering the response. The client only has to understand the response based on the well-known application protocol, i.e. the content and the formatting of the data for the requested service.

Socket: A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.

API: An application programming interface (API) is a set of subroutine definitions, protocols, and tools for building application software. In general terms, it is a set of clearly defined methods of communication between various software components

CRUD: In this application, create, read, update, and delete (as an acronym CRUD) are the four basic functions of persistent storage that will be used.

ISBN: The International Standard Book Number (ISBN) is a unique numeric commercial book identifier. The ISBN is 13 digits long if assigned on or

after 1 January 2007. The ISBN-13 check digit, which is the last digit of the ISBN, must range from 0 to 9 and must be such that the sum of all the thirteen digits, each multiplied by its (integer) weight, alternating between 1 and 3, is a multiple of 10. Full method is described here: https://en.wikipedia.org/wiki/International_Standard_Book_Number#ISBN-13_check_digit_calculation.

3. Implementation Details

3.1 Structure of Data

The overall data structure to be used for this application will be a ConcurrentHashMap.

```
java.util.concurrent
Class ConcurrentHashMap<K,V>

java.lang.Object
    java.util.AbstractMap<K,V>
        java.util.concurrent.ConcurrentHashMap<K,V>

Type Parameters:
    K - the type of keys maintained by this map
    V - the type of mapped values

public class ConcurrentHashMap<K,V>
extends AbstractMap<K,V>
implements ConcurrentMap<K,V>, Serializable
```

This class is a hash table supporting full concurrency of retrievals and high expected concurrency for updates. This class obeys the same functional specification as Hashtable, and includes versions of methods corresponding to each method of Hashtable. Retrievals reflect the results of the most recently completed update operations holding upon their onset. The table is dynamically expanded when there are too many collisions. This class does not allow null to be used as a key or value.

The objects to be used inside the ConcurrentHashMap will be Book objects. A Book object will contain an ISBN (primary key), title, author, and publisher. A Book object will not have to be thread safe since the data structure storing a Book object will itself be thread safe.

```
public class Book {

    private int isbn;                // primary key
    private String title;
    private String author;
    private String publisher;

    public Book(int isbn, String title, String author, String
    publisher) {
        this.isbn = isbn;
        this.title = title;
        this.author = author;
```

```
            this.publisher=publisher;
        }

        ...
    }
```

3.2 Structure of Client(s)

The client will be responsible for creating a stream socket and connecting it to the specified port number at the specified IP address. The client will be responsible for catching and handling any errors associated with the creation of a socket. Once a connection is established, the client will also be responsible for validating all data prior to sending that data to the server (including ISBN according to method described in section 2).

The client process may send four types of messages through established connection:

1. SUBMIT: messages containing Book attributes.
2. GET: messages containing request for a particular Book(s).
3. UPDATE: messages containing an update to a particular Book (excluding ISBN).
4. DELETE: messages containing request to remove a Book.

Once the client has sent a request, the client will be responsible for either displaying the data that was request, confirming an operation has taken place, and/or displaying an appropriate error message.

To aid the user in utilizing the server, the client will have a graphical user interface (GUI). Minimal GUI requirements are:

1. Text field to provide IP address of the server
2. Text field to provide port number
3. Connect/Disconnect button
4. Text area to type in text to be sent to server and
5. “Send” button
6. Text area to display result of the request

3.3 Structure of Server

3.3.1 Server Operations

The server will create a stream socket and continuously listen for and accept new connections. The server will be a multi-threaded application capable of supporting multiple, simutanious client connections while maintaining transaction atomicity. The data stored on the server is a “bibliography list” which is empty at startup and is not persistent. The data structure storing this data is outlined in section 3.1.

The server may process the following messages received from client:

1. SUBMIT: create a new Book and add it to the bibliograph list (if the Book is not a duplicate).

2. GET: send to client a list of entries matching particular request.
3. UPDATE: update a relevant entry in the bibliography list (excluding ISBN).
4. DELETE: remove the relevant entry from the bibliography list.

Once an operation has taken place, the server will sent either the data requested or a confirmation message to the client. If the operation could not take place, the server will send an appropriate error message to the client.

3.3.2 Server Error Codes

The server may return a number of codes depending upon client request success/falure. The codes are as follows:

```
SUCCESS:    client request has completed without error.
DUPLICATE:  client SUBMIT request is a duplicate request.
RESTRICTED: client UPDATE request attempting to update ISBN.
NOTFOUND:   client GET/DELETE reqeust cannot be found.
INVALID:    client attemped to send an undefined message.
```

3.3.3 Server Thread Implementation

To ensure the server can serve multiple, simutanious client connections, the structure of the server will be as follows:

```
while (true) {
    // accept a connection;
    // create a thread to deal with the client;
}
```

In more detail, the server implementation will look similar to the following:

```
... {

    ...

    int portNumber = Integer.parseInt(args[0]);
    boolean listening = true;

    try (ServerSocket serverSocket = new
    ServerSocket(portNumber)) {
        while (listening) {
            Thread serverThread = new
            ServerThread(serverSocket.accept());
            serverThread.start();
        }
    } catch (IOException e) {
        // handle error
    }
}
```

The thread reads from and writes to the client connection as

necessary.

4. Usage Examples

[client request]
SUBMIT
ISBN 9783161484100
TITLE Modular Algorithms in Symbolic Summation and Symbolic Integration
AUTHOR Gerhard
PUBLISHER Mir
[server response on success]
"SUCCESS"
[server response on failure]
"DUPLICATE"

[client request]
UPDATE
ISBN 9783161484100
PUBLISHER Springer
[server response]
"INVALID"

[client requet]
GET
TITLE Modular Algorithms in Symbolic Summation and Symbolic Integration
* Client expects to receive records of all books in bibliography with this title
[server response on success]
ISBN: 9783161484100
Title: Modular Algorithms in Symbolic Summation and Symbolic Integration
Author: Gerhard
Publisher: Mir
[server response on failure]
"NOTFOUND"

[client request]
GET
AUTHOR Gerhard
* Client expects to receive records of all books in the bibliography database with this author
[server response on success]
ISBN: 9783161484100
Title: Modular Algorithms in Symbolic Summation and Symbolic Integration
Author: Gerhard
Publisher: Mir
[server response on failure]
"NOTFOUND"

[client request]
GET
TITLE Modular Algorithms in Symbolic Summation and Symbolic Integration
AUTHOR Gerhard
* Client expects to receive records of all books in the bibliography database with this title and this author
[server response on success]

```
    ISBN: 9783161484100
    Title: Modular Algorithms in Symbolic Summation and Symbolic Integration
    Author: Gerhard
    Publisher: Mir
[server response on failure]
    "NOTFOUND"

[client request]
    GET
    ALL
    * Client expects to receive all entries of all books in the
    bibliography database
[server response]
    * dependent on state of bibliography data structure

[client request]
    DELETE
    AUTHOR Gerhard
    * Client requests removal of all books in the bibliography database
    with this author
[server response on success]
    "SUCCESS"
[server resonse on failure]
    "NOTFOUND"
```