Distributed String Array with Concurrency Control Using Java RMI

The objective of this project will be to provide hands-on experience working with Java RMI and concurrency control in distributed systems.

In this project, you will implement an array of strings as a remote object. In other words, the object array will be encapsulated into an object which will be available on a RMI server. Let us call this class as "RemoteStringArray". Clients can read and write into this array of strings through appropriate methods (read further for details). In addition, this object enforces concurrency control by using "read" and "write" locks.

The client on the other hand interacts with user and executes various operations on individual strings of the array.

RemoteStringArray Class

At its core, the RemoteStringArray contains an array of strings (size of array specified through constructor). You may also choose to include other structures in the object to incorporate concurrency control functionalities. The RemoteStringArray must include the following methods. Again, you can choose to include additional methods if you find it necessary.

`void RemoteStringArray (int n)` – This constructor creates an object with an string array of capacity "n".
`void insertArrayElement (int l, String str)` – inserts str as the l^{th} element of the string array. You can assume that l < capacity of the String array.
`boolean requestReadLock (int l, int client_id)` – Request read lock on l^{th} element of the array. client_id indicates the identifier of the client requesting the lock. Return "true" if lock is granted and "false" otherwise.
`boolean requestWriteLock (int l, int client_id)` – Request write lock on l^{th} element of the array. client_id indicates the identifier of the client requesting the lock. Return "true" if lock is granted and "false" otherwise.
`void releaseLock(int l, int client_id)` – Release the read/write lock on lth element. client_id indicates the identifier of the client requesting the lock. You can assume that only clients holding a lock will seek to release it.
`String fetchElementRead(int l, int client_id)` – Returns the String at the lth location in the read-only mode. Depending on your design, you can issue a read lock to the client (if possible) as a part of this method. Alternately, you can implement a separate method for obtaining the read lock which has to be successfully executed by the client for this method to succeed. Failure can be indicated by raising an exception or returning a "null" object (design decision left to you).
`String fetchElementRead(int l, int client_id)` – Returns the String at the lth location in the read-only mode. Depending on your design, you can issue a read lock to the client (if possible) as a part of this method. Alternately, you can implement a separate method for obtaining the read lock which has to be successfully executed by the client for this method to succeed. Failure can be indicated by raising an exception or returning a "null" object (design decision left to you).
`String fetchElementWrite(int l, int client_id)` – Returns the String at the lth location in the read/write mode. Analogous to the **`fetchElementRead`** method outlined above.

`boolean WriteBackElement (String str, int l, int client_id)` – copies str into the lth position only if client (indicated by client_id) has a write lock. Returns "true" if successful and "false" if not successful (e.g., client does not have the write lock).

I have intentionally omitted concurrency-related methods because designing appropriate methods for the same are part of the project.

The server program takes a single command line parameter – the name of the configuration file. The configuration file has the following format.

```
Bind name (the name associated with the remote object)
Capacity of the array
List of strings needed to initialize the array
Any other configuration parameters your implementation needs such as
the port number of the registry (if not using the standard registry
port)
```

String Array Client

The client program is rather simple. It can be implemented in the main method or in a separate methods. It interacts with the user, takes commands from the user and displays the results. These commands will include:

1. **`Get_Array_Capacity:`** Displays the capacity (max number of elements) of the string array.
2. **`Fetch_Element_Read <i>`**: Fetch the i^{th} element from the server in "read only" mode. You should make sure that the client has requested and obtained a "read lock" (either explicitly with a separate method call or implicitly within the same method call). Display "Success" or "Failure" based on the response from the server.
3. **`Fetch_Element_Write <i>`**: Fetch the i^{th} element from the server in "read/write" mode. You should make sure that the client has requested and obtained a "write lock" (either explicitly with a separate method call or implicitly within the same method call). Display "Success" or "Failure" based on the response from the server.
4. **`Print_Element <i>`**: Print the string associated with the ith element. You can assume that the element has been previously fetched from the server.
5. **`Concatenate <i> Str`**: Concatenate Str to the contents of the ith element. You can assume that the element has been previously fetched from the server.
6. **`Writeback <i>:`** Attempt to copy the element back to the server. Display "Success" if the element was successfully written back to the server and "Failure" if the writeback did not succeed (for instance the client didn't have the write lock on the element). Note that writeback does not automatically relinquish any locks on the object.
7. **`Release_Lock <i>:`** Release any locks on element i held by this client.

The client program takes a single command line parameter which is the name of the configuration file. The format of the configuration file will be as below:

```
Bind name of the remote object
Host name of the server
Any other configuration parameters your implementation need
```

Points to note:

1. Make sure you handle the concurrency conditions correctly.

2. The remote procedure should be solely responsible for all concurrency handling. For example, if a user calls writeback on an element without write lock, the client should attempt the writeback but should be rejected by the server.
3. Your code implementation will be tested by executing multiple clients simultaneously.
4. You will submit the server and the client code via elc. You will also demo the implementation.