TP Report

Bendris Samar and Benramoul Asma

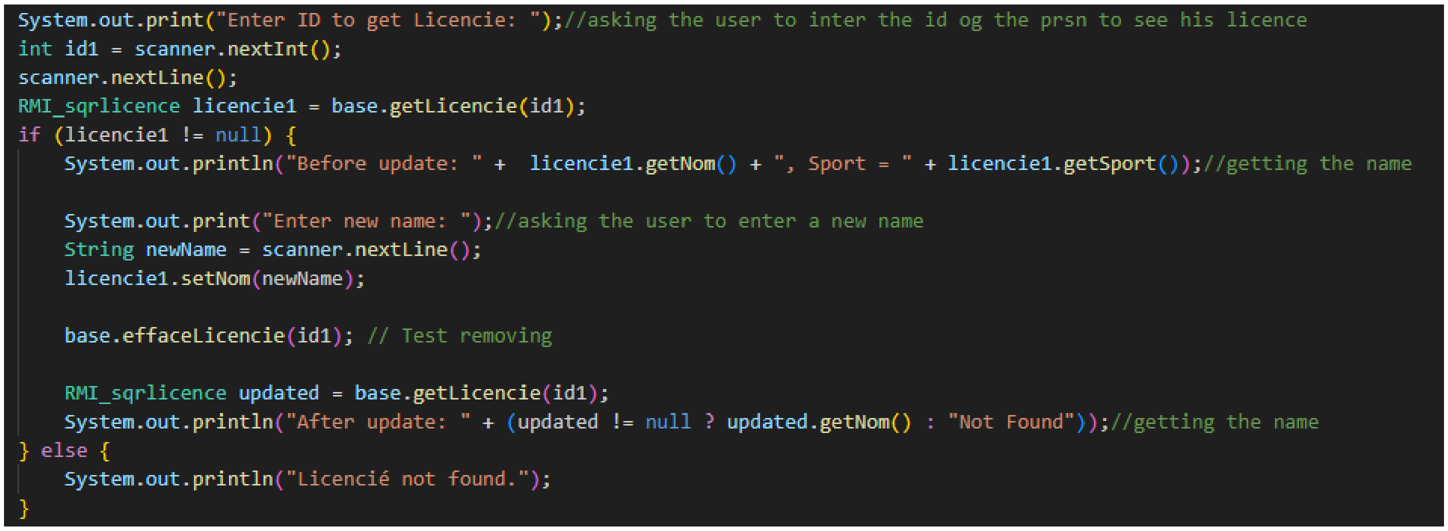
TP 4:

This tp about java RMI **Remote Methode Invocation** system that performs the following:

Client operation:

Allows users to interact with a remote server to manage **licencies** (members/players).

The user can add members with a name and sport, retrieve their details by ID, update their names, or delete them.



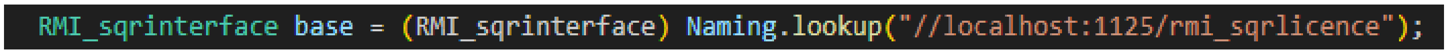
```java
System.out.print("Enter ID to get Licencie: ");//asking the user to inter the id og the prsn to see his licence
int id1 = scanner.nextInt();
scanner.nextLine();
RMI_sqrlicence licencie1 = base.getLicencie(id1);
if (licencie1 != null) {
    System.out.println("Before update: " +  licencie1.getNom() + ", Sport = " + licencie1.getSport());//getting the name

    System.out.print("Enter new name: ");//asking the user to enter a new name
    String newName = scanner.nextLine();
    licencie1.setNom(newName);

    base.effaceLicencie(id1); // Test removing

    RMI_sqrlicence updated = base.getLicencie(id1);
    System.out.println("After update: " + (updated != null ? updated.getNom() : "Not Found"));//getting the name
} else {
    System.out.println("Licencié not found.");
}
```

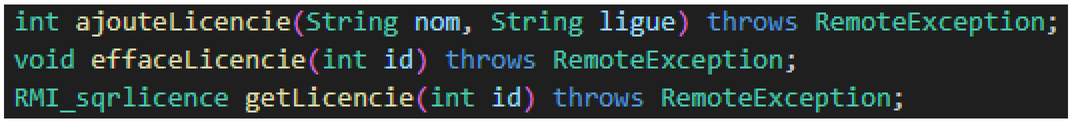
Interacts with the remote interface **RMI_sqrinterface** through a network connection.



```java
RMI_sqrinterface base = (RMI_sqrinterface) Naming.lookup("//localhost:1125/rmi_sqrlicence");
```

Interface:

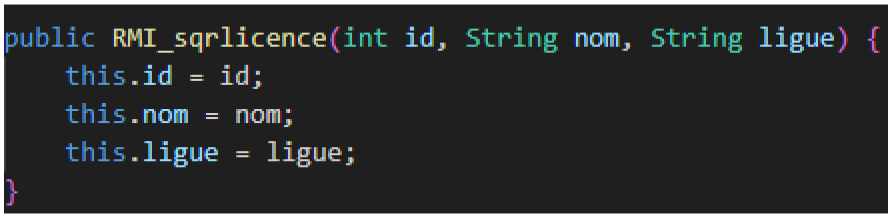
Defines methods for adding, retrieving and removing members. These methods throw **RemoteException** to handle communication errors.



```
int ajouteLicencie(String nom, String ligue) throws RemoteException;
void effaceLicencie(int id) throws RemoteException;
RMI_sqrlicence getLicencie(int id) throws RemoteException;
```

Licencie Class:

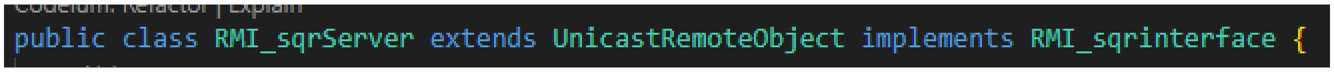
A Serializable class representing a member with attributes: ID, name, and sport/league.



```
public RMI_sqrlicence(int id, String nom, String ligue) {
    this.id = id;
    this.nom = nom;
    this.ligue = ligue;
}
```

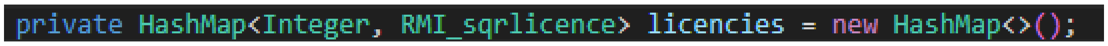
Server Implementation:

Provides concrete implementations of the methods declared in the **interface**.



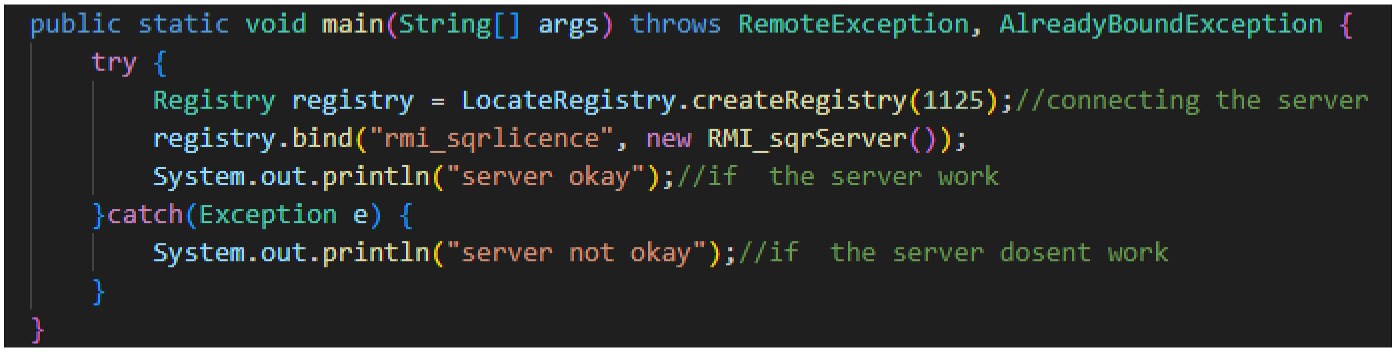
```
Codeium: Refactor | Explain
public class RMI_sqrServer extends UnicastRemoteObject implements RMI_sqrinterface {
```

Maintains a **HashMap** to store members with their IDs.



```
private HashMap<Integer, RMI_sqrlicence> licencies = new HashMap<>();
```

Binds the server to a specific port **1125** for client connections.



```
public static void main(String[] args) throws RemoteException, AlreadyBoundException {
    try {
        Registry registry = LocateRegistry.createRegistry(1125);//connecting the server
        registry.bind("rmi_sqrlicence", new RMI_sqrServer());
        System.out.println("server okay");//if  the server work
    }catch(Exception e) {
        System.out.println("server not okay");//if  the server dosent work
    }
}
```

Key Features:

A functioning RMI server-client setup that demonstrates the principles of remote method invocation in Java.

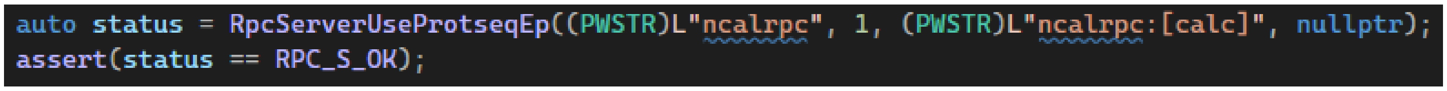
Demonstrates object serialization for data transfer and exception handling for robust network communication.

TP 8:

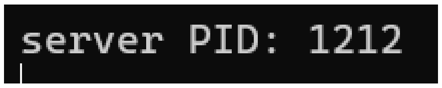
This tp about RPC **Remote Procedure Call** system that performs the following:

Server Operation:

The server uses the **ncalrpc = Named Local RPC** protocol with the endpoint **calc** for local communication.

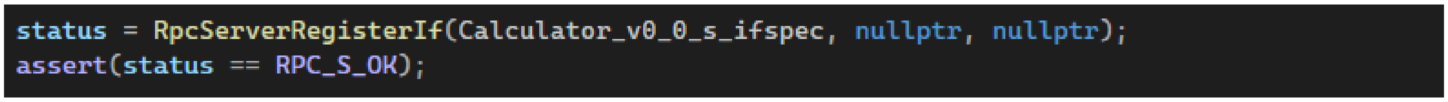


```
auto status = RpcServerUseProtseqEp((PWSTR)L"ncalrpc", 1, (PWSTR)L"ncalrpc:[calc]", nullptr);
assert(status == RPC_S_OK);
```

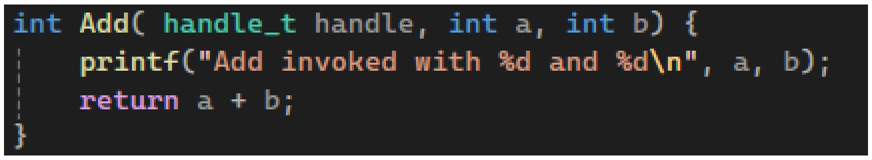


```
server PID: 1212
```

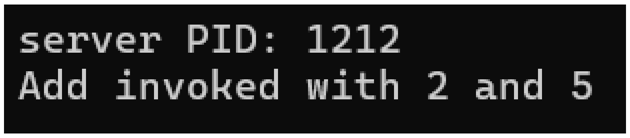
Registers the Calculator interface, which defines the Add method, with the RPC runtime.



```
status = RpcServerRegisterIf(Calculator_v0_0_s_ifspec, nullptr, nullptr);
assert(status == RPC_S_OK);
```

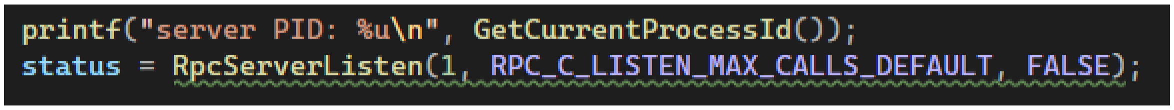


```
int Add( handle_t handle, int a, int b) {
    printf("Add invoked with %d and %d\n", a, b);
    return a + b;
}
```



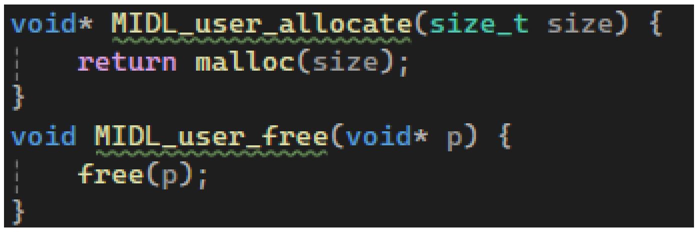
```
server PID: 1212
Add invoked with 2 and 5
```

Server Listening uses **RpcServerListen** to handle client requests with default maximum calls.



```
printf("server PID: %u\n", GetCurrentProcessId());
status = RpcServerListen(1, RPC_C_LISTEN_MAX_CALLS_DEFAULT, FALSE);
```

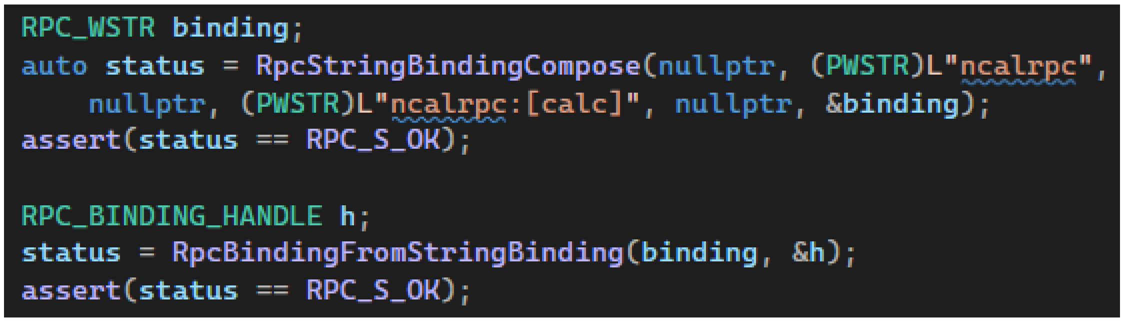
Implements **MIDL_user_allocate** and **MIDL_user_free** for RPC memory management.



```
void* MIDL_user_allocate(size_t size) {
    return malloc(size);
}
void MIDL_user_free(void* p) {
    free(p);
}
```

Client Operation:

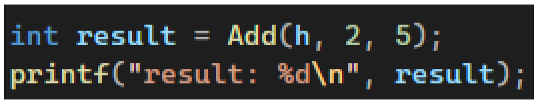
Establishes a connection to the server using the same **ncalrpc** endpoint.



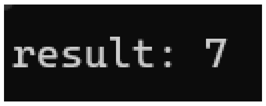
```
RPC_WSTR binding;
auto status = RpcStringBindingCompose(nullptr, (PWSTR)L"ncalrpc",
    nullptr, (PWSTR)L"ncalrpc:[calc]", nullptr, &binding);
assert(status == RPC_S_OK);

RPC_BINDING_HANDLE h;
status = RpcBindingFromStringBinding(binding, &h);
assert(status == RPC_S_OK);
```

Calls the **Add** method with two integers and retrieves the result.



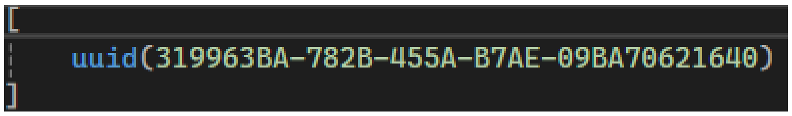
```
int result = Add(h, 2, 5);
printf("result: %d\n", result);
```



```
result: 7
```

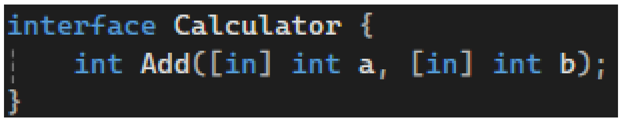
IDL File:

describes the Calculator interface, which includes the methods that the client can call on the server.



```
[
    uuid(319963BA-782B-455A-B7AE-09BA70621640)
]
```

uniquely identifies the Calculator interface. It ensures no conflicts occur when multiple interfaces are registered in the same RPC runtime.



```
interface Calculator {
    int Add([in] int a, [in] int b);
}
```

The work is in github