REPORT ON COMPUTATION SERVER

Student’s Name

Institution’s name

Course

Date

**Introduction**

The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM. In this report, we will provide a detailed overview of the implementation of a remote method invocation (RMI) Computation Service. The Computation Service allows clients to perform addition and sorting operations on integers. The implementation comprises a client program, a server program, and the necessary interfaces and callbacks to facilitate remote communication.

**Implementation Details:**

**ComputationService Interface**

The ComputationService interface serves as a contract between the client and the server, defining the remote methods that can be invoked. It uses the following techniques:

add (int i, int j) synchronously adds two numbers and outputs the outcome.

Adds two numbers asynchronously and calls the callback with the result by using addAsync(int i, int j, CallbackInteger> callback). Sorts an integer array synchronously and returns the sorted array with the call sort(int[] array).sortAsync(int[] array,

Callback<int[]> callback): Performs asynchronous sorting of an integer array and invokes the callback with the sorted array.

**Callback Interface**

The Callback interface is a generic interface that allows the server to notify the client asynchronously. It declares two methods:

onResult(T result): Invoked by the server to deliver the asynchronous result to the client.

onAck(): Invoked by the server to acknowledge the reception of the asynchronous request.

**Client Implementation**

The ComputationClient class represents the client program. It establishes a connection with the server using the LocateRegistry.getRegistry() method and retrieves the remote ComputationService object from the registry. The client then utilizes both synchronous and asynchronous methods provided by the service. Synchronous operations, such as addition and sorting, return the result directly, while asynchronous operations utilize the Callback interface to handle the results asynchronously.

**Server Implementation:**

The ComputationServer class represents the server program. It implements the ComputationService interface and provides the actual implementation for the remote methods. Synchronous methods, such as addition and sorting, execute the operations and return the results directly to the client. Asynchronous methods, on the other hand, execute the operations in separate threads to avoid blocking the main execution flow. Upon completion, they invoke the provided callback object with the result.

**Callback Implementations:**

To handle callbacks effectively, two callback implementations were provided: AddCallback and SortCallback. These classes extend the UnicastRemoteObject class and implement the Callback interface. When the onResult() method is invoked, they get the callback function as an argument and invoke it. These implementations enable asynchronous results delivery from the server to the client.

**Lessons Learned**

**RMI (Remote Method Invocation):** The implementation of RMI allows distributed systems to invoke methods on remote objects, enabling remote communication and interaction.

**RMI Registry:** The RMI registry acts as a centralized lookup service, allowing clients to locate remote objects by name.

**Synchronous vs. Asynchronous Operations**: Synchronous operations block until they receive a response, while asynchronous operations continue execution and provide a callback mechanism for receiving results at a later time.

**Thread Usage:** Asynchronous operations are performed in separate threads to prevent blocking the main execution flow, ensuring a responsive and efficient system.

**Interface Definitions**: The use of well-defined interfaces facilitates loose coupling between the client and server components, allowing for flexibility and easier maintenance.

**Challenges Faced**

During the implementation of the RMI Computation Service, several challenges were encountered:

1. Remote Exception Handling: Remote Exceptions occurred during RMI communication, such as network failures and server unavailability. Proper exception handling was implemented to gracefully handle such situations and provide appropriate error messages.
2. Callback Serialization: Passing callback objects between the client and server required serialization. Serializable implementations of the callbacks were created to facilitate this process, ensuring proper communication between the components.

**Conclusion**

The implementation of the RMI Computation Service has provided valuable insights into developing distributed systems with remote method invocations. The client-server architecture, coupled with synchronous and asynchronous interactions, allows for flexible and efficient communication between components. By overcoming challenges related to exception handling and callback serialization, we have gained practical experience in implementing robust and responsive distributed systems. The use of well-defined interfaces and thread management has further enhanced the reliability and performance of the system. Overall, this implementation has deepened our understanding of RMI and its applications in distributed computing.