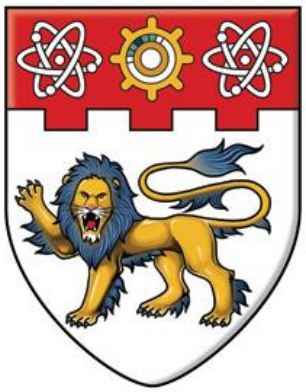


NANYANG TECHNOLOGICAL UNIVERSITY

CPE416: Distributed Systems

CSC411: Distributed Computing

Lab Report

Hoang Tri Tam	U0920150B	33%
Vu Ngoc Linh	U0920229E	33%
Gaurav Gupta	U1020607K	33%

Declaration of contribution

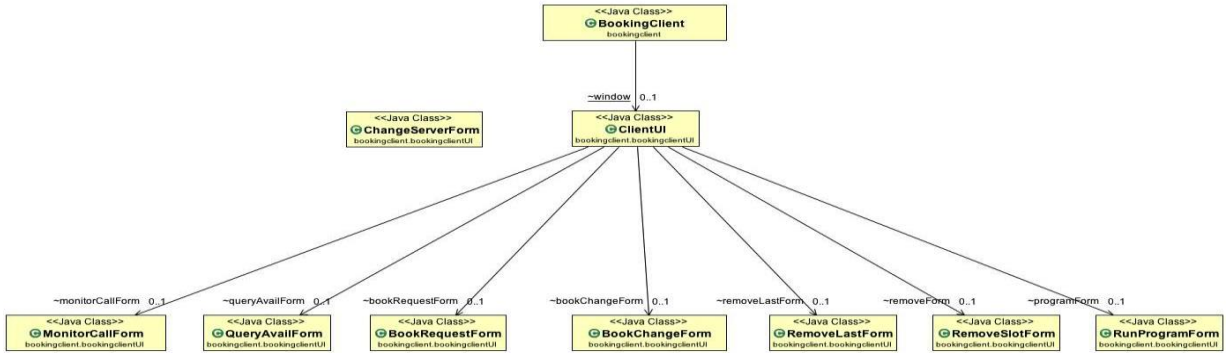
All members equally contributed to this assignment.

1. Software Design

We have chosen a simple software design to implement this system. The system is divided into a client, a server and the interface between the client and the server.

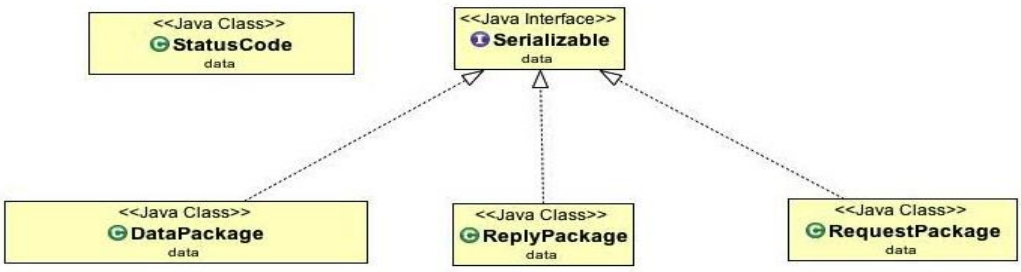
Client

The client provides a Graphical User Interface for the user to select a service and input the required parameters. It then marshals the user's request into a byte array using the interface and sends the request to the server over UDP. It then waits for the acknowledgement from the server and upon receiving the acknowledgement sends the data to the server. After this it awaits the reply from the server and on receiving the reply it un-marshals the reply and display the result to user on the GUI.



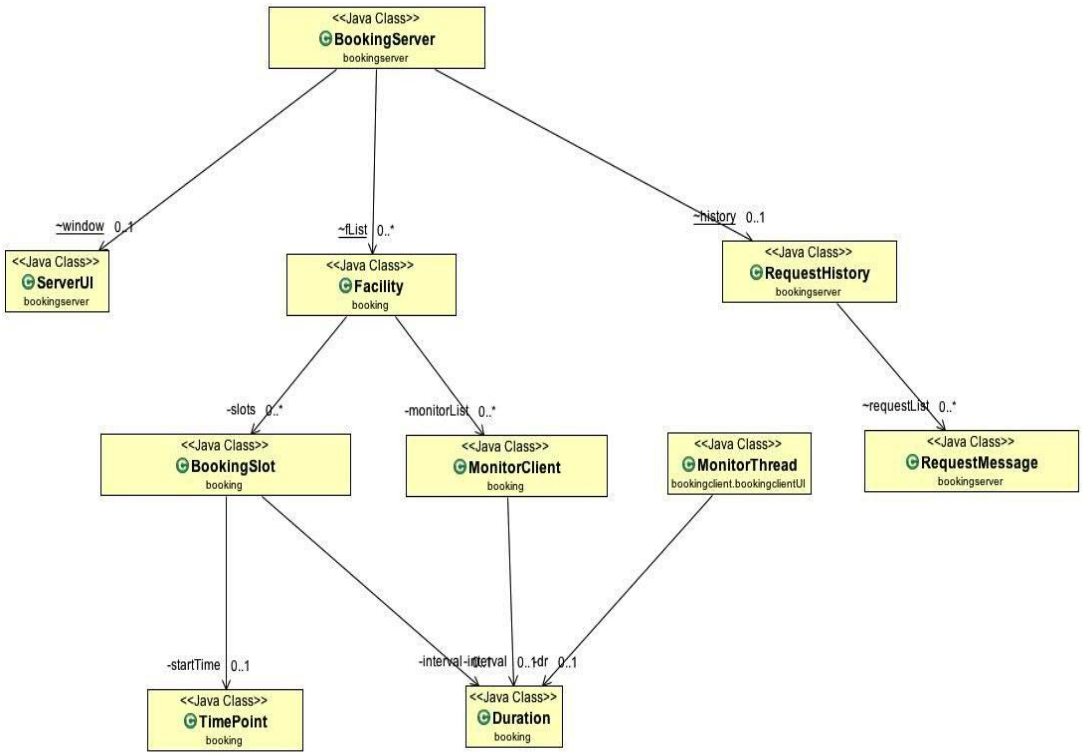
Interface

The interface provides the mechanism for marshaling and un-marshaling of data. For marshaling it converts the data into a byte array to be sent over UDP and for un-marshaling it extracts the data from the byte buffer into objects. It also provides the codes and indicators for signaling and requesting.



Server

The server keeps the record of all the facilities and the user request history. It waits for the request by the client, upon receiving the request is sends it checks for the duplicate and sends the acknowledgement back to the client. After sending the acknowledgement it waits for the data object which it un-marshals using interface and performs the required operation. It then replies the result back to the client and waits for the next request.

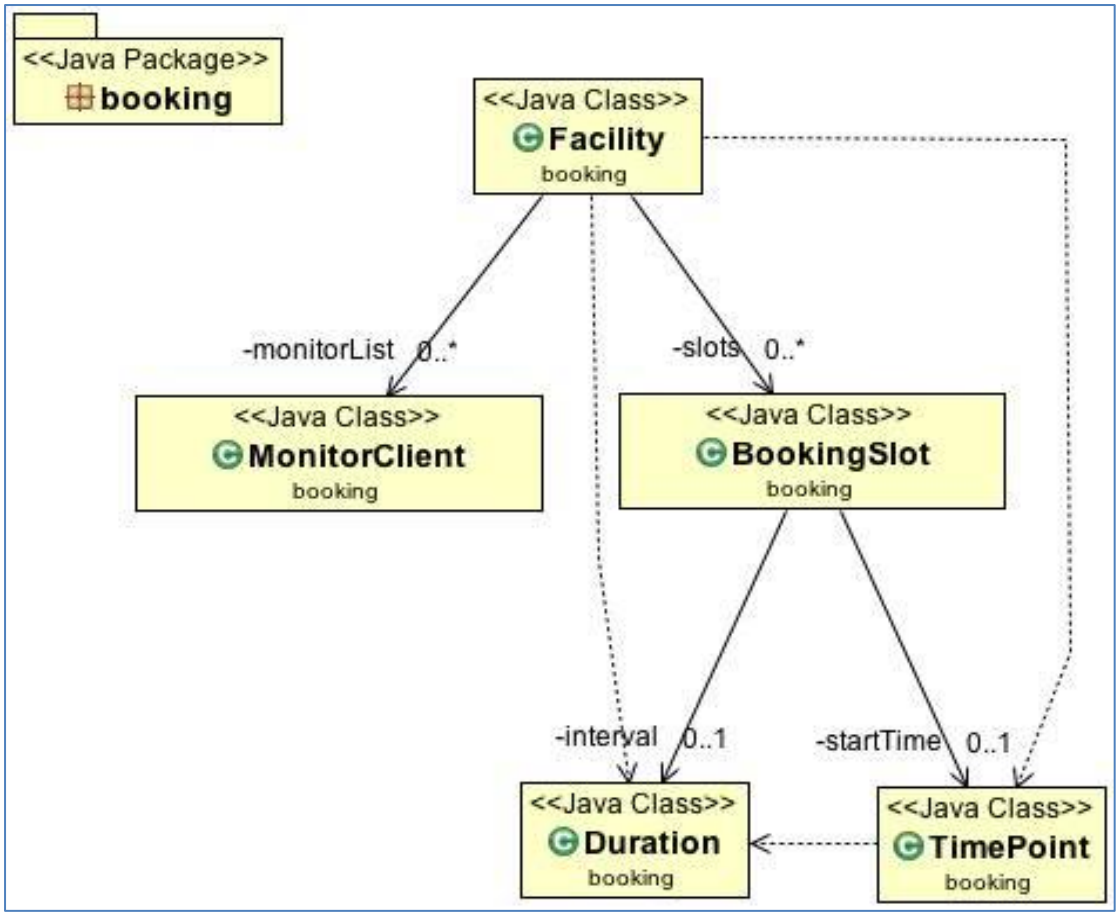


2. Implementation Strategy

The implementation strategy is pretty straightforward. We have four packages to implement the data structure, client, server and the interface.

Package: Booking

This package consists of the classes that form the data structures for the storage and computation of data.



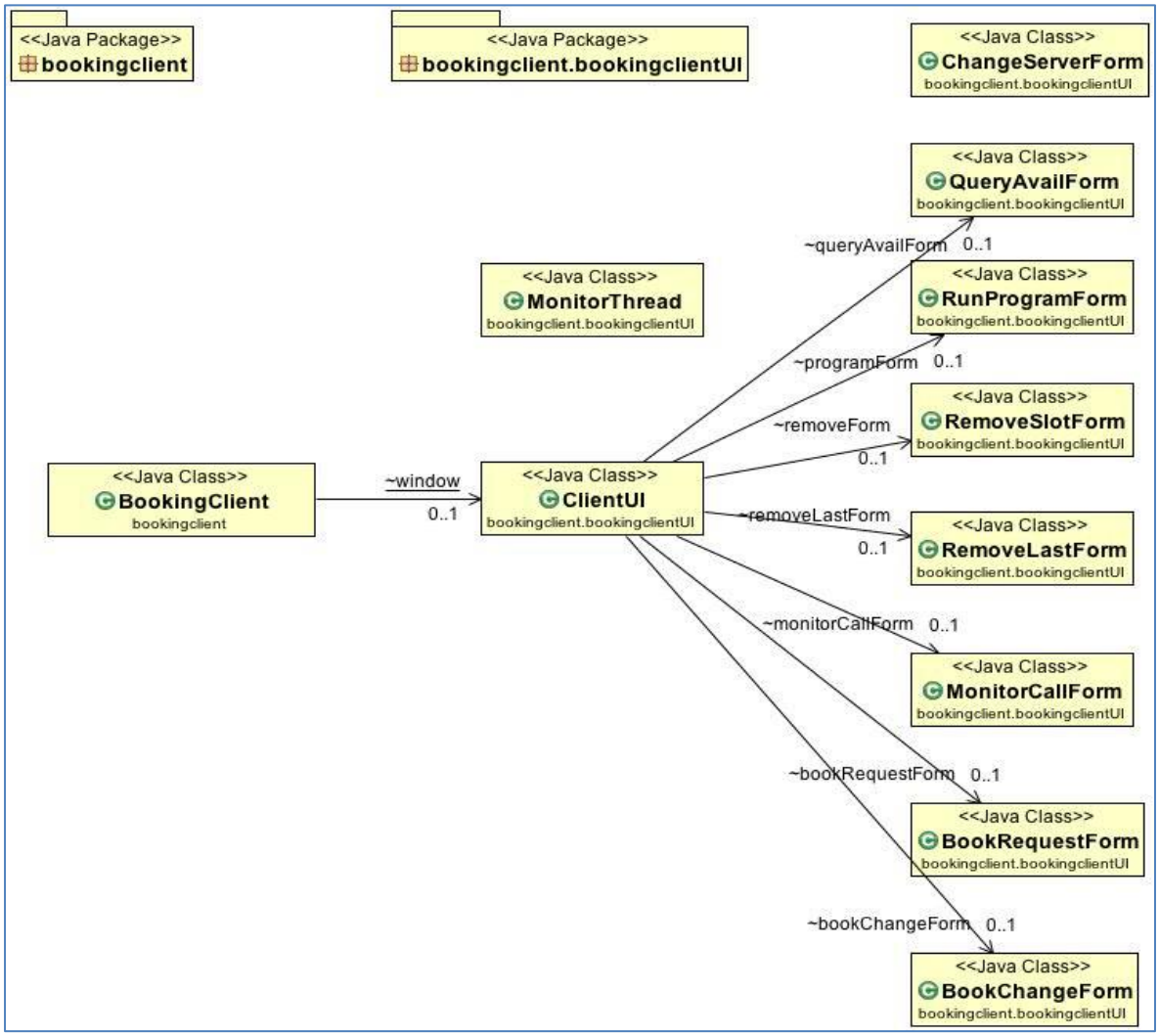
CLASSES:

1. **Duration:** This class' objects represent the basic unit of time duration in number of days, hours and minutes.
2. **TimePoint:** This class' objects represent the basic unit of time point in number of days, hours and minutes. This class provides the functions to manipulate and compare time points.

3. **BookingSlot:** This class' objects represent a booking slot in a facility. It contains the starting time (TimePoint), the interval (Duration), a confirmation ID and the client address of the client that booked this slot.
4. **MonitorClient:** The object of this class holds the information of a monitoring client namely client address and the end time of the monitor.
5. **Facility:** The object of this class represents a facility type. Its attributes include id and description of the facility, list of booking slots and list of clients who monitor this facility.

Package: Bookingclient

This package consists of the classes that make up the client side of the program. It consists of a client and the graphical user interface for the user to interact with the client.

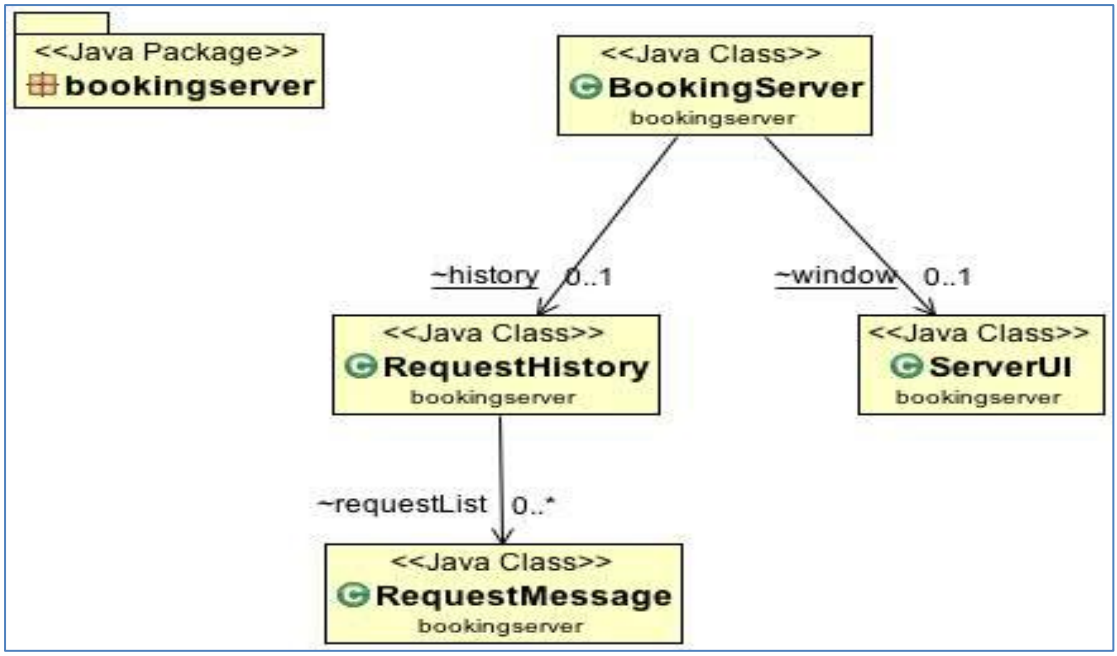


CLASSES:

1. **BookingClient:** This is the main class of the client side. This class sets up the UDP connection with the server along with loss simulation and provides the methods and GUI to the user to request services from the server and displays the result.
2. **MonitorThread:** This class extends java.lang.Thread and creates a thread to monitor the availability of the facility. This thread allows the client to perform other actions while monitoring.
3. **ClientUI:** This class extends javax.swing.JFrame and provides the main GUI window for the client containing buttons and a text area.
4. **BookChangeForm:** This class extends javax.swing.JFrame and provides the input form for "Change Bookslot" service.
5. **BookRequestForm:** This class extends javax.swing.JFrame and provides the input form for "Book Facility" service.
6. **MonitorCallForm:** This class extends javax.swing.JFrame and provides the input form for "Monitor Facility" service. It creates a new MonitorThread to do the monitoring job until duration timeout.
7. **RemoveLastForm:** This class extends javax.swing.JFrame and provides the input form for "Remove The Last Slot" of a facility.
8. **RemoveSlotForm:** This class extends javax.swing.JFrame and provides the input form for "Remove All Slot" of a facility.
9. **RunProgramForm:** This class extends javax.swing.JFrame and provides the input form for "Get Quotes of The Day" service.
10. **QueryAvailForm:** This class extends javax.swing.JFrame and provides the input form for "Query Availability" of a facility.
11. **ChangeServerForm:** This class extends javax.swing.JFrame and provides the input form for "Change Server" service. It sets a new server IP address and port number in the client.

Package: Bookingserver

This package consists of the classes that make up the server side of the program. It consists of a server, a history log and a graphical user interface to perform actions on server.

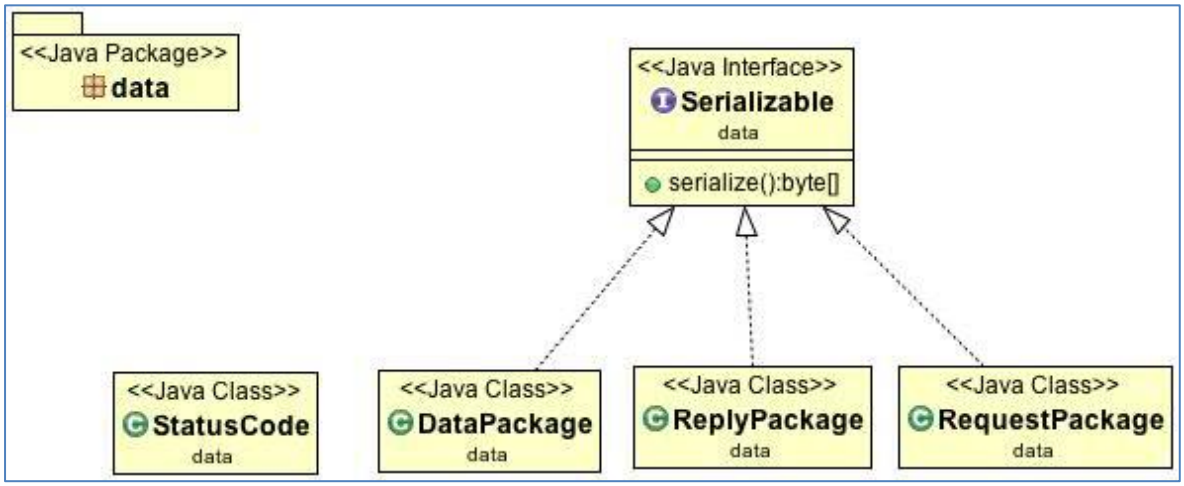


CLASSES:

1. **BookingServer:** This is the main class of the server side. This class sets up the UDP connection along with loss simulation and waits for a client request. It maintains the list of facilities and the history of client requests. It also implements the invocation semantic (At-least-once and At-most-once) and filters duplicate requests.
2. **RequestHistory:** This class' object stores a list of successful request by the client. Each element of the list is an object of RequestMessage Type. This class also provides the methods to add new requests and search the list for a matching attribute RequestMessage to identify a duplicate request.
3. **ServerUI:** This class extends javax.swing.JFrame and provides the main GUI window for the server containing buttons and a text area
4. **RequestMessage:** This class' object stores the information of a client request including the RequestPackage, client address, client port and data buffer of reply message.

Package: Data

This package consists of the classes that make up the interface for communication between client and server. These classes implement the marshaling and un-marshaling of request, reply and data formats.



INTERFACE:

1. **Serializable:** This is an interface that dictates all the classes that implement it to provide a serialize() function for the marshaling of data.

CLASSES:

2. **StatusCode:** This class contains all the status codes that are used by the client and server to make decisions and reply to the client.
3. **DataPackage:** This class contains implements serializable and contains methods to do the marshaling and un-marshaling of data objects, strings and integers.
4. **ReplyPackage:** This class contains implements serializable and contains the status code and the methods to serialize this status code into byte array.
5. **RequestPackage:** This class contains implements serializable and contains the list of all the services. Its objects contain the requestId, serviceId, facilityId and optionalId which can be marshaled into a byte stream to send the request to the server from a client.

3. Services

The following services are provided by the system:

1. **Connet to Server:** This service establishes a connection between client and server and fetches the list of facilities from the server. It is an **idempotent** operation service.
2. **Query Availability:** This service allows a user to query the availability of a facility on a selection of one or multiple days by specifying the facility name and the number of days and time. It is an **idempotent** operation service.
3. **Book Facility:** This service allows a user to book a facility for a period of time by specifying the facility name and the start time and the duration of the booking. It is an **idempotent** operation service.
4. **Change Bookslot:** This service allows a user to change its booking by specifying the facility name, confirmationID and an offset for changing. The change does not modify the length of the time period booked. It is an **idempotent** operation service.
5. **Monitor Facility:** This service allows a user to monitor the availability of a facility over the week through callback from the server for a designated time period (monitor interval). To register, the client provides the facility name and the length of monitor interval to the server.
6. **Get Quotes of the Day:** This service fetches a fresh quote from the server based on the lucky number input by the user. It is an **idempotent** operation service.
7. **Remove All Slot:** This service removes all the bookings of the selected facility. It is an **idempotent** operation service.
8. **Remove the Last Slot:** This service removes the last booked slot of the selected facility. It is a **non-idempotent** operation service.
9. **Change Server:** This service changes the IP address and port number of the server.

4. Marshaling and Un-Marshaling

For the Marshaling and Un-Marshaling of data **java.nio** package has been used to help convert the objects in the byte array and reconstruct them back to objects.

Marshaling

1. **Request-** The request message has been marshaled in a 16 byte array in the format (requestId x 4, serviceId x 4, facilityId x 4, optionalId x4)
2. **Reply-** The reply message has been marshaled in a 4 byte array in the format (statusCode x 4)
3. **Object TimePoint -** The object TimePoint has been marshaled in a 12 byte array in the format (date x 4, hour x 4, min x 4)
4. **Object Duration -** The object Duration has been marshaled in a 12 byte array in the format (day x 4, hour x 4, min x 4)
5. **Object ArrayList<BookingSlot> -** The ArrayList has been marshaled in a 4+ size x 4 x 6 bytes array in the format (size x 4, startDate x 4, startHour x 4, startMin x 4, intervalDay x 4, intervalHour x 4, intervalMin x 4,.....)
6. **String[] -** A string array has been marshaled into a array of bytes with each byte representing a character in US_ASCII format. The format is (String[0] x length, '\n', String[1] x length, '\n', ..., String[n] x length, '!', '!' , '!')

Un-Marshaling

1. **Request-** The request message is reconstructed by extracting the (requestId x 4, serviceId x 4, facilityId x 4, optionalId x4) and passing them as parameters to the constructor of RequestMessage class to create a new object.
2. **Reply-** The reply message is a 4 byte integer value (statusCode x 4) and is extracted from the byte array to reconstruct the integer value.
3. **Object TimePoint -** The object TimePoint is reconstructed from the byte array by extracting the values in the format (date x 4, hour x 4, min x 4) and passing them as parameters to the constructor of TimePoint class to create a new object.

4. **Object Duration -** The object Duration is reconstructed from the byte array by extracting the values in the format (day x 4, hour x 4, min x 4) and passing them as parameters to the constructor of Duration class to create a new object.
5. **Object ArrayList<BookingSlot> -** The ArrayList of BookingSlot is reconstructed by first extracting the size integer from the first 4 bytes of the byte array and then creating another ArrayList of BookingSlot whose elements are restored by extracting the values in the format (startDate x 4, startHour x 4, startMin x 4, intervalDay x 4, intervalHour x 4, intervalMin x 4) and passing them as parameters to create TimePoint, Duration and BookingSlot object.
6. **String[] -** The string array is reconstructed by first extracting the complete concatenated string ending with "!!!" and then extracting and restoring each string array element delimited by '\n' character.

5. Loss Simulation and Fault Tolerance

Invocation Semantics

The invocation semantics can be changed on the server by using the server GUI.

AT-LEAST-ONCE: If At-Least-Once semantic is selected then the server *does not* check for the duplicate requests, it sends the acknowledgement to the client and proceeds on the normal routine to serve the request.

AT-MOST-ONCE: If At-Most-Once semantic is selected then the server checks for the duplicate requests, if duplicate request it detected it informs the client of the duplicate request, displays appropriate message and waits for a new request from the client. If the duplicate request is not detected, it sends the acknowledgement to the client and proceeds on the normal routine to serves the request.

Simulation and Handling of Loss

TIMEOUTS:

1. Client Timeout: A DatagramSocket timeout of 500 millisecond is implemented in the client. A call to receive() for this DatagramSocket will block for 500 milliseconds. If the timeout expires, a java.net.SocketTimeoutException is raised. In the exception handler the client displays an appropriate message and tries to send the request again. It repeats this process again and again for 8 times and then terminates the request and displays that the server is not available.
2. Server Timeout: A DatagramSocket timeout of 750 millisecond is implemented in the server after the server sends the acknowledgement and waits for the data packet. A call to receive() for this DatagramSocket will block for 750 milliseconds. If the timeout expires, a java.net.SocketTimeoutException is raised. In the exception handler the server displays an appropriate message and terminates the current request. It then again waits for a new request from the client.

HISTORY MAINTENANCE: The server maintains a history of client's identity and the requests in the object of RequestHistory class. The RequestHistory class provides the list of RequestMessages and functions to search request based on client's identity and

requestId. This requestId is unique for given clients request in one session and hence re-request in the event of timeout does not change this requestId.

DUPLICATE FILTERING: If the selected semantic is At_Most_Once server searches for the duplicate request in the history and if the requestId for a given client matches the current requestId it means that the resending of same request has occurred from the client side and hence this duplicate request must be filtered out. In the event of duplicate request a message is displayed on the server side and a duplicate indication is sent to the client in the acknowledgement packet. After which the server terminates the current session and waits for a new request.

LOSS SIMULATION: A pseudo random generator function provided by java.lang.Math.random() is employed to do the loss simulation. If the random number is greater than the loss probability then packet is transmitted otherwise the packet is dropped and packet loss message is displayed. Same logic is used on both client and server to simulate the loss of packet.

LOSS HANDLING: In the event of packet loss the waiting side will eventually timeout and perform the handling as stated in the Timeout section above.

If the packet is lost from client side while sending the request, the server will not receive the request and hence will not send acknowledgement. The client waiting for the acknowledgement will eventually timeout and resend the request in second try. This process will continue for 8 times until client finally fails to send the request.

If the packet is lost from the server side while sending acknowledgement then the client will keep waiting for the acknowledgement until it times out and because server timeout is less than client timeout server will timeout first and wait for a new request. The client waiting for the acknowledgement will eventually timeout and resend the request. The server waiting for the request will receive this as a new request and the whole process will begin from the start.

If the packet is lost from the server side while sending data package then the client waiting for the data package will timeout and because client will timeout it will resend the request to the server. The server waiting for a new request will accept this request and the whole process will begin from the start.

Comparison of Invocation Semantics

During the loss simulation and testing it was found that the two invocation semantics worked as expected.

When **At_Least_Once** semantic was selected it had following effect:

- On **Non-Idempotent** operation like **Remove the Last Slot** more than one slots were removed if the duplicate request was executed again. This resulted in erroneous result than desired.
- On **Idempotent** operation like **Remove All Slot** the execution of duplicate request did not had any effect as the bookings were removed whether the command executed once or more than once.

When **At_Most_Once** semantic was selected it had following effect:

- On **Non-Idempotent** operation like **Remove the Last Slot** only the last slot was removed, the duplicate requests were filtered out and appropriate message indicating a duplicate request were displayed. This was in consistence with the desired operation.
- On **Idempotent** operation like **Remove All Slot** the execution of duplicate request not because the operation was idempotent but because the duplicate requests were filtered out.