

**CPE416: Distributed Systems**

**CSC411: Distributed Computing**

**Lab Report**

Hoang Tri Tam – 33%

Linh Vu – 33%

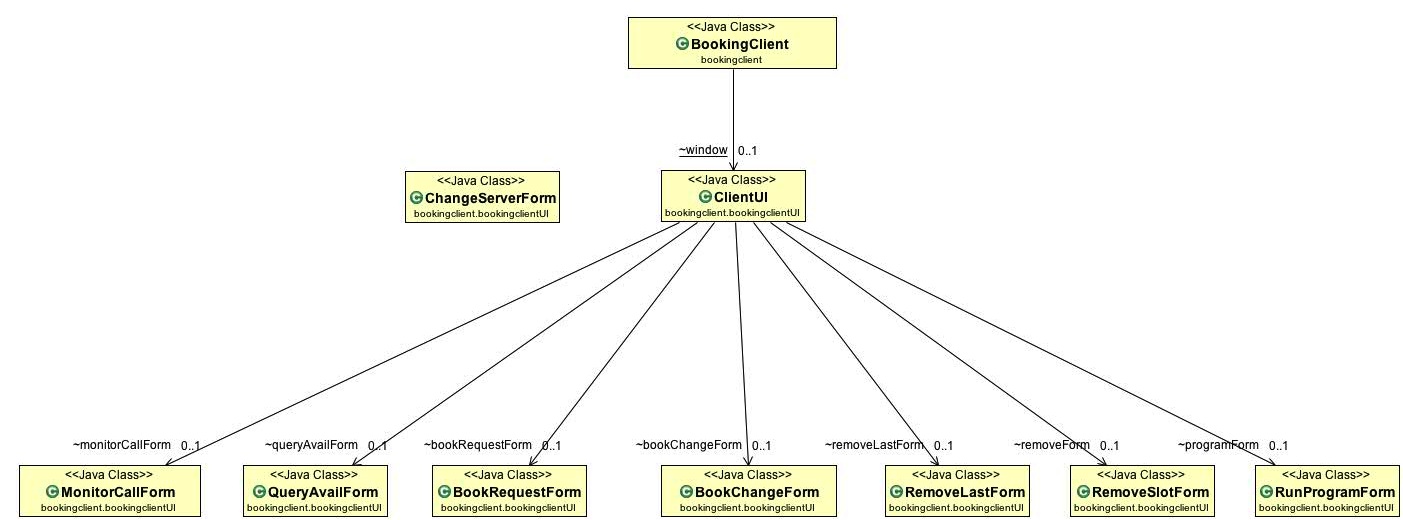
Gaurav Gupta – 33%

# 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.

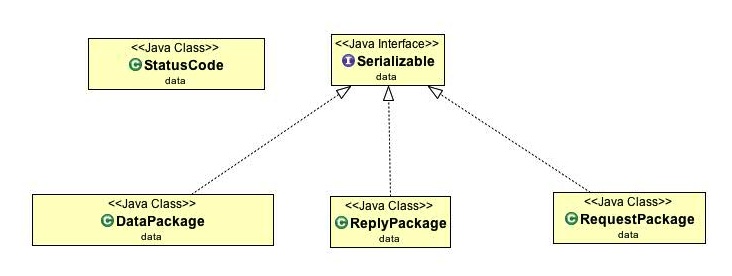
## Client

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.



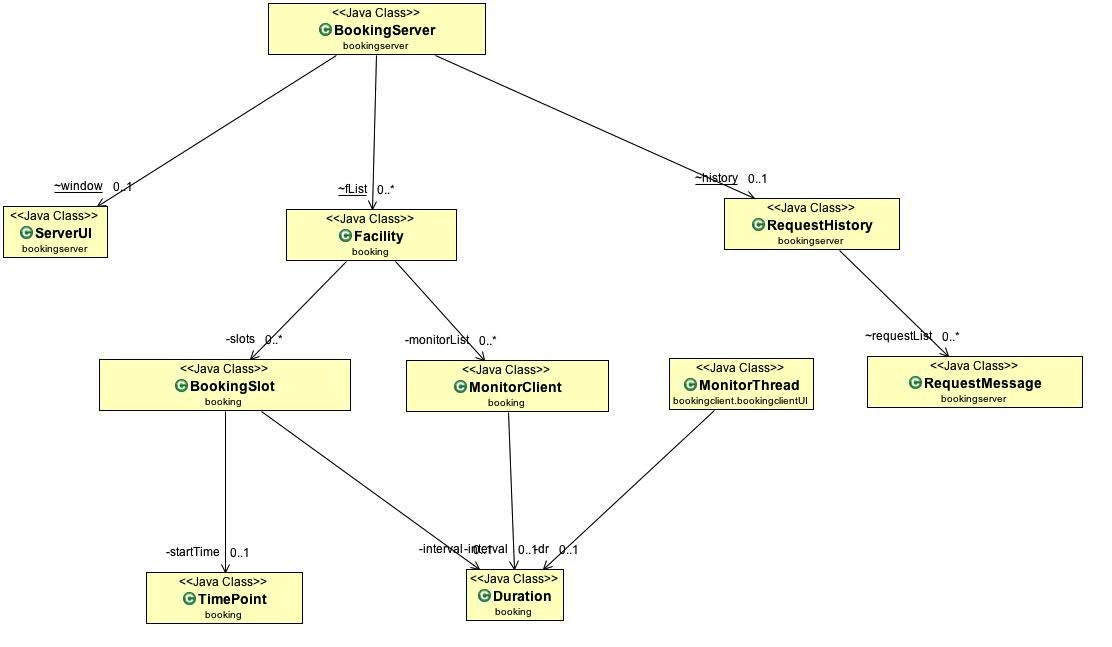
## Interface

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.

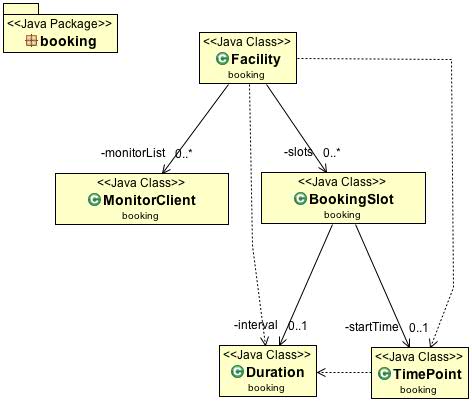


# Implementation Strategy

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.

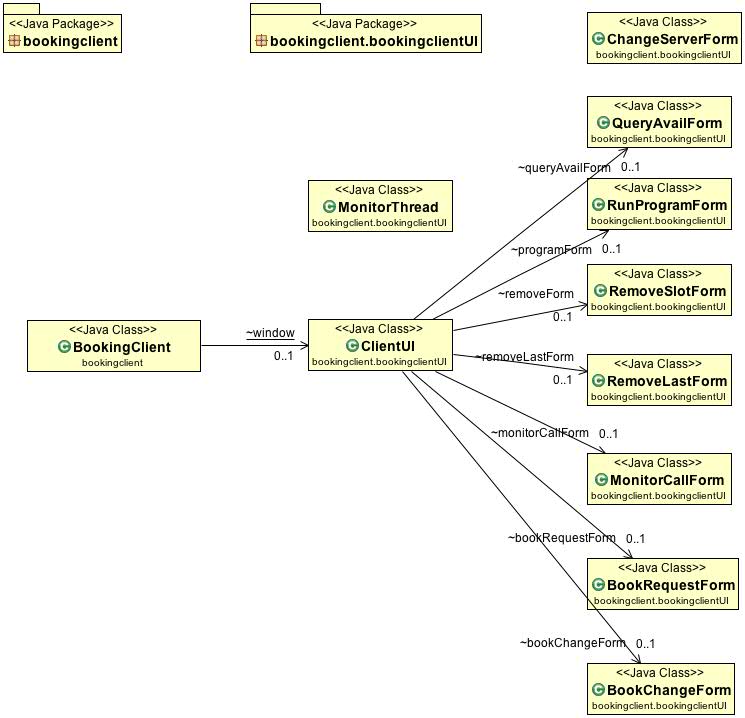


Classes:

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.

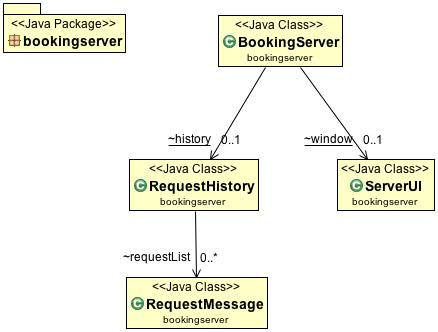


Classes:

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.

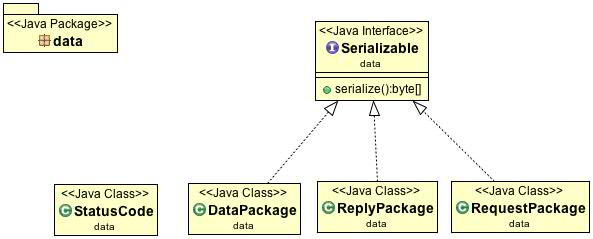


Classes:

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:

1. **StatusCode:** This class contains all the status codes that are used by the client and server to make decisions and reply to the client.
2. **DataPackage:** This class contains implements serializable and contains methods to do the marshaling and un-marshaling of data objects, strings and integers.
3. **ReplyPackage:** This class contains implements serializable and contains the status code and the methods to serialize this status code into byte array.
4. **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.

# Services

# Marshaling and Un-Marshaling

# Loss Simulation and Fault Tolerance