- Client-Server communication
    - Transport protocol - UPD
        - Socket programming
    - Server
        - Stores information about facilities (Meeting rooms, lecture theaters)
            - name
            - availability
                - Date format
                    - day/hour/minute
                    - Day - enum : Monday - Sunday
                    - Hours - Integer
                    - Minute - Integer
                - Requires exclusive use
        - Server program implements a set of services on facilities
    - Client
        - Provides an interface for users to access and invoke services on facilities provided by server
        - Sends requests to server
        - Represents the result of request in the console
    - Services provided by the server:
        - Query availability
            - Input: Facility
            - Input: Days (could be multiple)
            - Return: Error if there is no such facility
            - Return: Availability during for specified facility and time
        - Book facility
            - Input: Facility
            - Input: Time
                - Start
                - End
            - Return: Confirmation ID if successful
            - Return: Error if there is no such facility
            - Return: Error if the time slot is already taken
            - Server: Change state on server (db)
        - Change booking
            - Input: Confirmation ID
            - Input: time offset
                - Cannot change length of booking
            - Return: Acknowledgement if successful
            - Return: Error if the change is not possible due to availability
            - Return: Error if confirmation ID is incorrect
            - Return: Error if the time offset is not allowed
                - Specify what the overlap is
            - Server: Changes state on server (db)
        - Monitor facility

        - Input: Facility name
        - Input: Length of interval to monitor
        - Return: Updated availability on change via callback
        - Return: Confirmation of being added as observer of the facility
        - Server: Records the IP and port of the requester
        - Server: Activates callback when updates or bookings are made to facility
        - Client: Waits for the server to return something during interval
            - This may cause some issues
        - Server: Removes the registered observer after expiration of interval
    - Idempotent operation/service on availability of facilities
    - Non-idempotent operation/service on availability of facilities
- Work needed to be done
    - Design request and reply message format
    - Design marshalling and unmarshalling
    - Use byte array to store marshalled data
    - Server records IP and port of client upon request received
    - Client interface keeps asking the user to input commands
    - Print messages returned by server on screen (text based UI)
    - Client is given option to terminate the program
    - Server prints requests and returned results
    - Two different invocation semantics: at-least-once and at-most-once
        - Will include techniques such as:
            - Time-outs (client side)
            - Filter duplicate requests (server side)
            - Maintain request history (server side)
        - Specify semantics as an argument when starting the server/client
    - Design experiment to test the different invocation methods
        - Simulate loss of messages
        - Show:
            - At-least-once method will cause problem with idempotent operations
            - At-most-once method will be fine with both idempotent and non-idempotent operations
        -
- Restrictions
    - No ready made JAVA RMI
    - No ready made JAVA RPC
    - No ready made JAVA Object Serialization
    - No ready made CORBA
    - No ready made JAVA Input streams
    - No ready made JAVA Output streams
    - Server and client uses UDP calls in socket programming ONLY
- Hints
    - Server IP and port are known to client (can be hardcoded or specified as

input argument on program startup)
        - No threads needed to handle concurrent calls from different clients (Clients calls are separated in time)
        - No GUI needed - text based is enough
        - No persistent storage needed, saving state in working memory is enough
- Language : JAVA
- Deliverables
    - Well commented code
    - Working and functional program at presentation
    - Lab report of maximum 12 pages
- Deadline: April 1, 2013