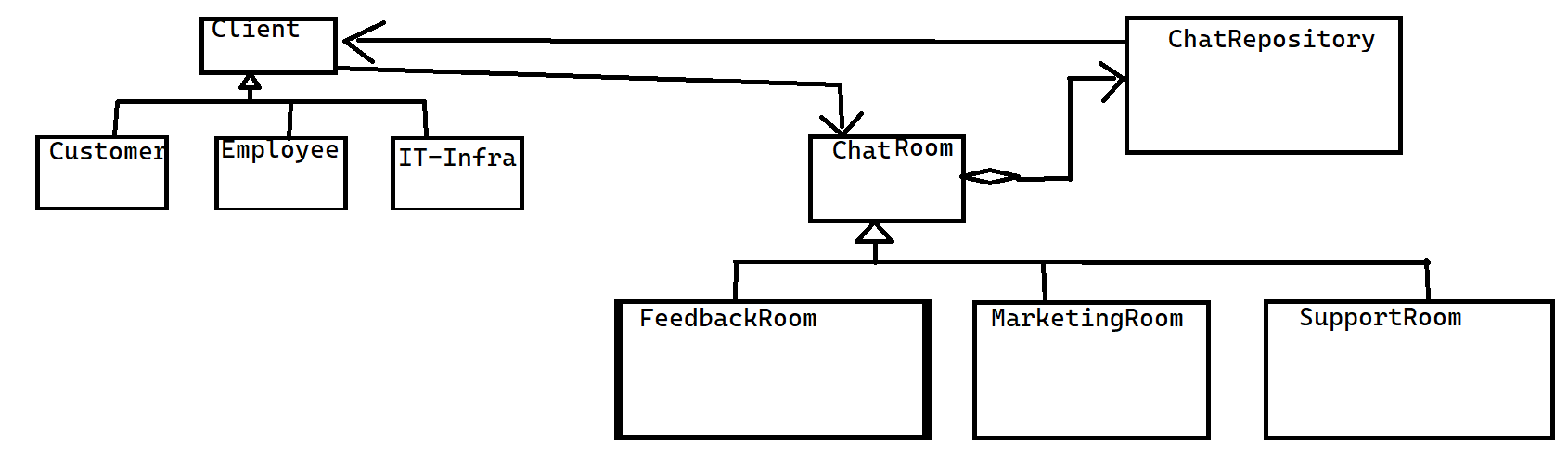
# ChatObject

* Create the class Hierarchy prescribed in the UML and detect the state and behavior for each class and interfaces.
* Identify Additional Classes to be injected.
* Ensure you follow SOLID principles.
* The Chat user should be in a position to switch the chat room.
* The Chat User should be able to send message to another particular chat user.
* The chat user should broadcast the message in the current chat room.
* The chat user should broadcast message to all users in all chatrooms.



* Note the below items to be considered and correctly used in your code
  + Use separate header and cpp files for each class. Define only function signature in .h files
  + Include all header files properly in your cpp files
  + Correctly use public, private or protected member variables and functions
  + Const correctness
  + Note the Return type of functions and keywords (e.g. virtual)
  + Unit Testing needs to be done for each function (test a couple of known inputs with known outputs and display “PASS” if function works correctly)
* Project specific requirements

|  |  |  |
| --- | --- | --- |
|  | Task | Output in console window |
| 1 | Connect | Joined chat room \*\*\* |
| 2 | Disconnect | Left chat room \*\*\* |
| 3 | Change Room | Left chat room \*\*\*, Joined \*\*\* |
| 4 | Send Message to particular User | Message received from \*\*\* |
| 5 | Broadcast Message within chatroom | Message from \*\*\* to all users |
| 6 | Broadcast Message to all users in All Chatroom | Message[chatroom] to all users |
| 7 | Shutdown chat | All Users should be notified |

# Message Interceptor

* Create the classes to Intercept messages being sent.
* The user can configure the interceptors and the code should respond.
* The code should provide provision to chain interceptor.
* Interceptors are like Encoder (binary, text), Encryptor (Symmetric, Asymmetric)   
  Logger (Console Log, File Log, etc)
* Use Reflection to load all interceptors from config file.
* Sketch a UML Diagram
* Detail the task list.
* Should be Norms Complaint

Norms are.

* + Use separate header and cpp files for each class. Define only function signature in .h files
  + Include all header files properly in your cpp files
  + Correctly use public, private or protected member variables and functions
  + Const correctness
  + Note the Return type of functions and keywords (e.g. virtual)
  + Unit Testing needs to be done for each function (test a couple of known inputs with known outputs and display “PASS” if function works correctly)