In this project, I implement all required features except “open” button for manager.

1. General Design

I use RMI in JAVA library as my server architecture. The general process for my system is that the server starts to run before all clients. After the server is active, clients can join the server. The first client joining the server is the manager of the server. If the manager leaves, then the whole server will be shut down. Other clients want to join the server after the manager should ask for permission from the manager. A dialog will pop up in the manager window to ask if the new client is allowed to join. If the manager clicks “Yes”, then the new client can join the whiteboard application. A JList is at the right of the frame, which will display all the joining clients and update if there are any new clients joining in. Only the manager can kick out a client. The manager does this simply by selecting the client he/she wants to kick out and then click kick button. The client being kicked out will simply be unable to see the frame and his/her window will be closed. Only the manager can open, save, save as and new a whiteboard. For save, an image of jpanel will be saved with default name. For save as, an image of jpanel will be save with a customised name given by the manager. A dialog will pop up to ask for the name. For new, all components on the jpanel will be removed and all clients will receive a message from command line window. The client names for each client are generated automatically and are always unique.

2. Detailed Design

In details, when a server is being initialised, it is being registered to a registry server. It will also create a ClientManager object which will be used to accept clients, remove clients, and see if a client is still active. I use a class called ClientWatchDog to ping all clients managered by the server’s clientManager. If a client in the clientManager lose connection, then the ClientWatchDog class will remove the client automatically. I set a separate thread for the ClientWatchDog so that it can detect the inactive clients concurrently as the program is running.

A client will initialise only when a server is active which mean that a server is in the correct registry server. Otherwise, clients will not be initialised. Once a client is being able to initialise, it will attach to a server. When the client registers to the server, the server will check if it has any client. If there are no client in the server, the server will automatically set the first client be manager. If there are some clients in the server, the server will ask permission from its manager by calling WhiteboardClient.permit(). A diaglog will pop up in the manager’s window. The result collected by the dialog will be delivered by CommunicationContext to the server and the server will decide whether allowing the user to join based on the decision from manager. If manager does not allow, the client program will be terminated.

Once the client is allowed to join, he/she will see the same panel as all other clients can see. This is because the WhiteboardPanel class belonging to the client will get all the shapes in the panel from server and update to the latest joining client. The server’s WhiteboardManager object will also call resyncClientNamesList function to update the clients’ names list so that all clients can see the same and latest name list.

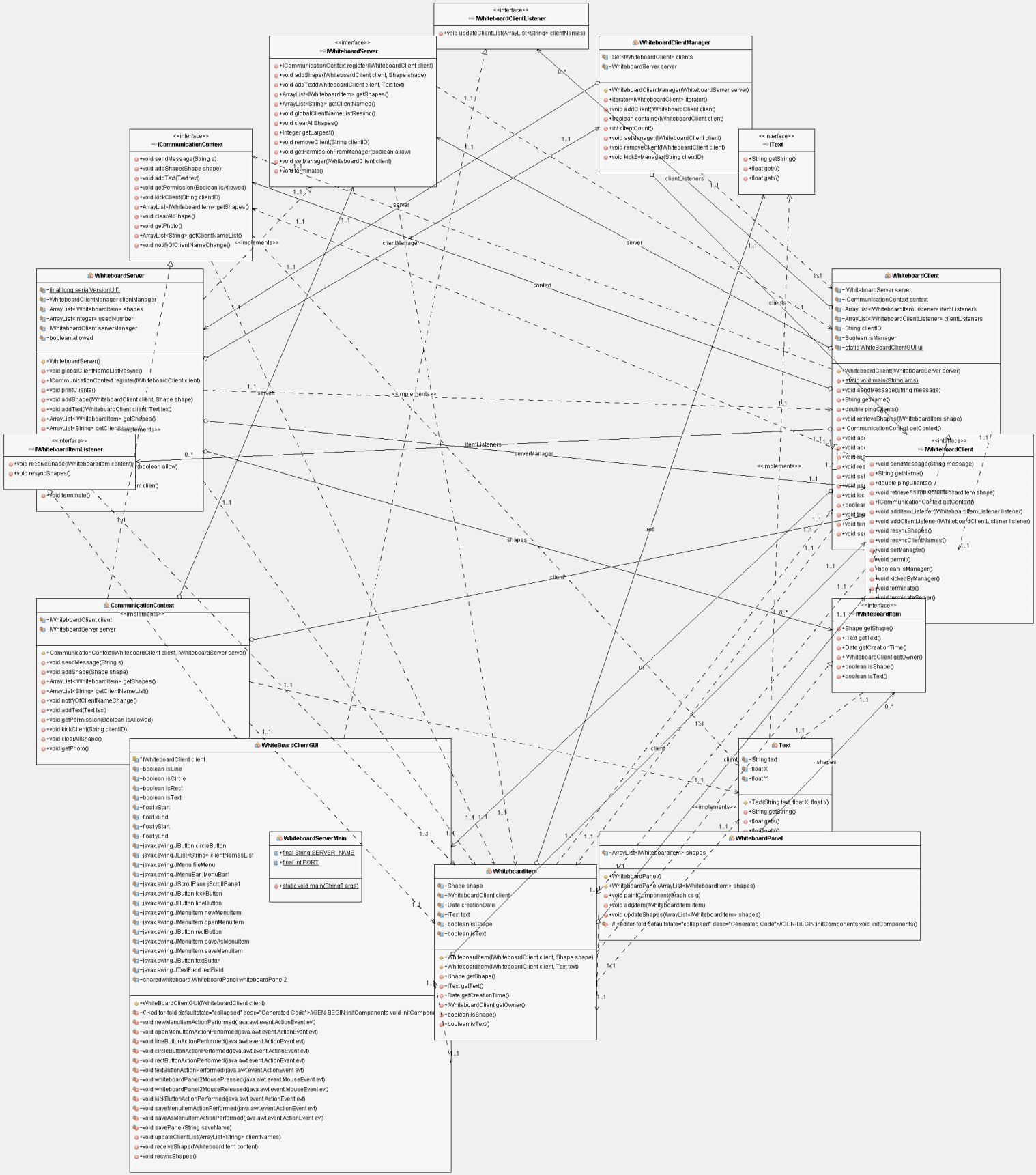
When a client wants to draw something, some key information of his/her drawing will be collected by the CommunicationContext object to the client. After that, the CommunicationContext object will call the corresponding functions from server. If the things drawed by the client is any shapes except text, then the CommunicationContext object will call addShapes while it will call addText if the clients want to draw a text on the panel. Then, the server will keep the things passed from clients and let the WhiteboardClient call retrieveShape. This will let all the clients connecting to the server view the latest change.

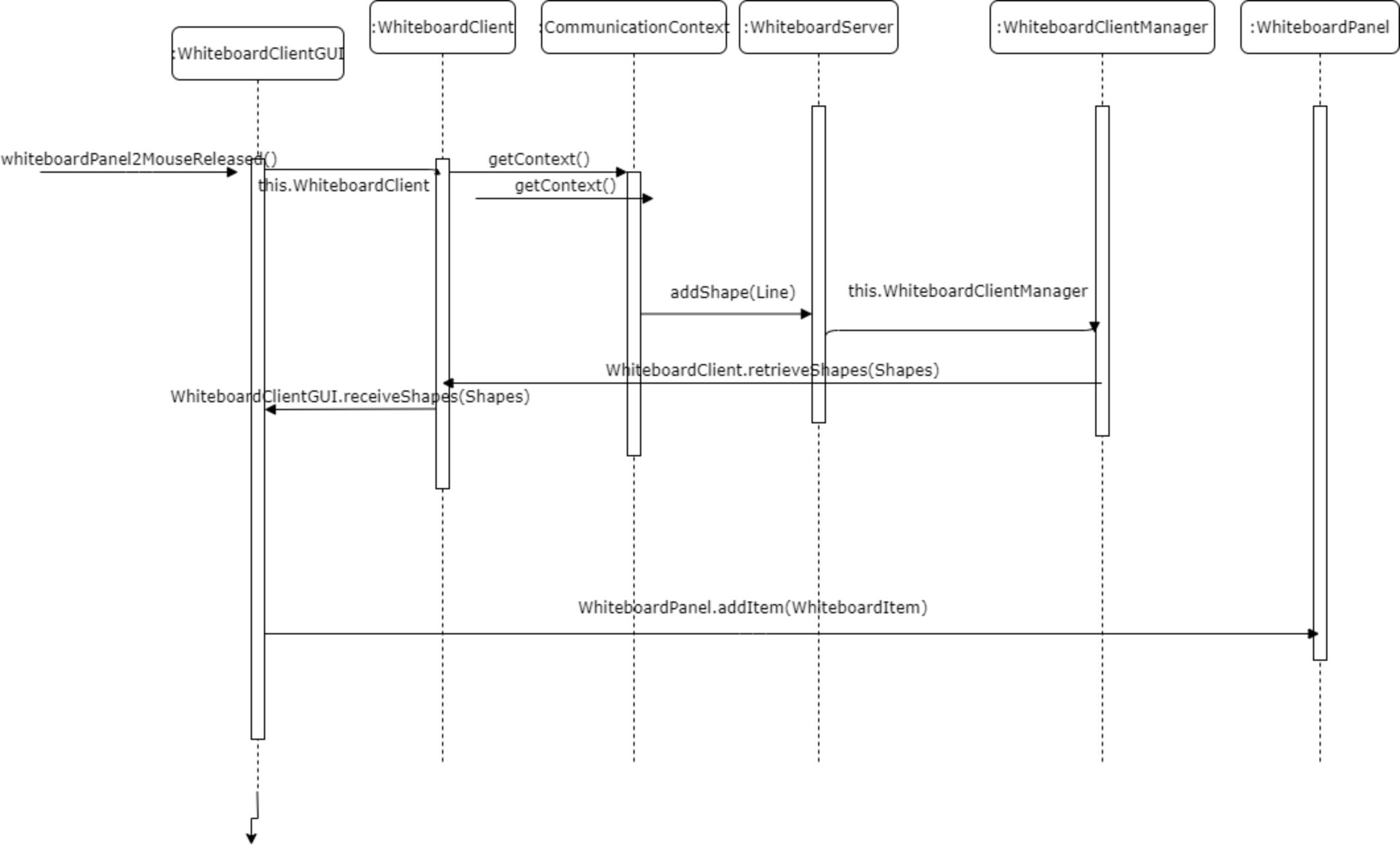
The logic behind the update of JList is similar. At any time, a client registers to a server, the server will automatically assign a unique integer to the client. Then the WhiteboardClientManager class will add the client to itself and call globalClientNameListResync() in servers to resync client names list. Therefore, each client will get the latest name list from the server.

Only the manager of the server is allowed to kick a certain member. When the manager selects the client he/she wants to kick out, the manager will call kickClient() in the CommunicationContext class. The class will call kickClient() at server. The server will call kickByManager() function in WhiteboardClientManager class. The WhiteboardClientManager call will call kickedByManager() method in client, which will force the client to terminate his/her window. If the manager wants to leave, he/she can only leave by close his/her window. When he/she clicks the “x” button, WhiteboardClientGUI class associated with the manager will call the manager’s terminateServer() method. This method will call terminate() function in the server. Terminate() function in server will call serverDown() function in each client connected to it to notify the server will close. A message will be printed in the command line window for each user. The server will also shut down.

Save and Save As functions are similar, they just save the jpanel as image by savePanel() function in WhiteboardGUI except that save as will pop up a dialog for the manager to rename the saved image. For the “New” button, all shapes in the jpanel will be removed. The logic is similar to adding a shape to the panel. Once the new button is clicked, WhiteboardClientGUI function will call the clearAllShape() function in CommunicationContext class. The CommunicationContext class will call clearAllShape() function in server. The server will replace its ArrayList containing all the shapes with an empty ArrayList and resync to all clients.

A sequence diagram and a class diagram are attached at the end. The sequence diagram shows the sequence when a client wants to draw a line. Other functionality should have similar sequence diagram, so I do not include.


<<interface>>
IWhiteboardClientListener
+void updateClientList(ArrayList<String> clientNames)
<<interface>>
IWhiteboardServer
+ICommunicationContext register(IWhiteboardClient client)
+void addShape(IWhiteboardClient client, Shape shape)
+void addText(IWhiteboardClient client, Text text)
+ArrayList<IWhiteboardItem> getShapes()
+ArrayList<String> getClientNames()
+void globalClientNameListResync()
+void clearAllShapes()
+Integer getLargest()
+void removeClient(String clientID)
+void getPermissionFromManager(boolean allow)
+void setManager(IWhiteboardClient client)
+void terminate()
WhiteboardClientManager
-Set<IWhiteboardClient> clients
-IWhiteboardServer server
+WhiteboardClientManager(IWhiteboardServer server)
+Iterator<IWhiteboardClient> iterator()
+void addClient(IWhiteboardClient client)
+boolean contains(IWhiteboardClient client)
+int clientCount()
+void setManager(IWhiteboardClient client)
+void removeClient(IWhiteboardClient client)
+void kickByManager(String clientID)
<<interface>>
ICommunicationContext
+void sendMessage(String s)
+void addShape(Shape shape)
+void addText(Text text)
+void getPermission(Boolean isAllowed)
+void kickClient(String clientID)
+ArrayList<IWhiteboardItem> getShapes()
+void clearAllShape()
+void getPhoto()
+ArrayList<String> getClientNameList()
+void notifyOfClientNameChange()
<<interface>>
IText
+String getString()
+float getX()
+float getY()
WhiteboardServer
-final long serialVersionUID
-WhiteboardClientManager clientManager
-ArrayList<IWhiteboardItem> shapes
-ArrayList<Integer> usedNumber
-IWhiteboardClient serverManager
-boolean allowed
+WhiteboardServer()
+void globalClientNameListResync()
+ICommunicationContext register(IWhiteboardClient client)
+void printClients()
+void addShape(IWhiteboardClient client, Shape shape)
+void addText(IWhiteboardClient client, Text text)
+ArrayList<IWhiteboardItem> getShapes()
+ArrayList<String> getClientNameList()
+void terminate()
WhiteboardClient
-IWhiteboardServer server
-ICommunicationContext context
-ArrayList<IWhiteboardItemListener> itemListeners
-ArrayList<IWhiteboardClientListener> clientListeners
-String clientID
-Boolean isManager
-static WhiteBoardClientGUI ui
+WhiteboardClient(IWhiteboardServer server)
+static void main(String args)
+void sendMessage(String message)
+String getName()
+double pingClients()
+void retrieveShapes(IWhiteboardItem shape)
+ICommunicationContext getContext()
<<interface>>
IWhiteboardItemListener
+void receiveShape(IWhiteboardItem content)
+void resyncShapes()
<<interface>>
IWhiteboardClient
+void sendMessage(String message)
+String getName()
+double pingClients()
+void retrieveShapes(IWhiteboardItem shape)
+ICommunicationContext getContext()
+void addItemListener(IWhiteboardItemListener listener)
+void addClientListener(IWhiteboardClientListener listener)
+void resyncShapes()
+void resyncClientNames()
+void setManager()
+void permit()
+boolean isManager()
+void kickedByManager()
+void terminate()
+void terminateServer()
CommunicationContext
-IWhiteboardClient client
-IWhiteboardServer server
+CommunicationContext(IWhiteboardClient client, IWhiteboardServer server)
+void sendMessage(String s)
+void addShape(Shape shape)
+ArrayList<IWhiteboardItem> getShapes()
+ArrayList<String> getClientNameList()
+void notifyOfClientNameChange()
+void addText(Text text)
+void getPermission(Boolean isAllowed)
+void kickClient(String clientID)
+void clearAllShape()
+void getPhoto()
<<interface>>
IWhiteboardItem
+Shape getShape()
+IText getText()
+Date getCreationTime()
+IWhiteboardClient getOwner()
+boolean isShape()
+boolean isText()
WhiteBoardClientGUI
-IWhiteboardClient client
-boolean isLine
-boolean isCircle
-boolean isRect
-boolean isText
-float xStart
-float xEnd
-float yStart
-float yEnd
-javax.swing.JButton circleButton
-javax.swing.JList<String> clientNamesList
-javax.swing.JMenu fileMenu
-javax.swing.JMenuBar jMenuBar1
-javax.swing.JScrollPane jScrollPane1
-javax.swing.JButton kickButton
-javax.swing.JButton lineButton
-javax.swing.JMenuItem newMenuItem
-javax.swing.JMenuItem openMenuItem
-javax.swing.JButton rectButton
-javax.swing.JMenuItem saveAsMenuItem
-javax.swing.JMenuItem saveMenuItem
-javax.swing.JButton textButton
-javax.swing.JTextField textField
-sharedwhiteboard.WhiteboardPanel whiteboardPanel2
+WhiteBoardClientGUI(IWhiteboardClient client)
-// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents void initComponents()
-void newMenuItemActionPerformed(java.awt.event.ActionEvent evt)
-void openMenuItemActionPerformed(java.awt.event.ActionEvent evt)
-void lineButtonActionPerformed(java.awt.event.ActionEvent evt)
-void circleButtonActionPerformed(java.awt.event.ActionEvent evt)
-void rectButtonActionPerformed(java.awt.event.ActionEvent evt)
-void textButtonActionPerformed(java.awt.event.ActionEvent evt)
-void whiteboardPanel2MousePressed(java.awt.event.MouseEvent evt)
-void whiteboardPanel2MouseReleased(java.awt.event.MouseEvent evt)
-void kickButtonActionPerformed(java.awt.event.ActionEvent evt)
-void saveMenuItemActionPerformed(java.awt.event.ActionEvent evt)
-void saveAsMenuItemActionPerformed(java.awt.event.ActionEvent evt)
-void savePanel(String saveName)
+void updateClientList(ArrayList<String> clientNames)
+void receiveShape(IWhiteboardItem content)
+void resyncShapes()
WhiteboardServerMain
+final String SERVER_NAME
+final int PORT
+static void main(String[] args)
WhiteboardItem
-Shape shape
-IWhiteboardClient client
-Date creationDate
-IText text
-boolean isShape
-boolean isText
+WhiteboardItem(IWhiteboardClient client, Shape shape)
+WhiteboardItem(IWhiteboardClient client, Text text)
+Shape getShape()
+IText getText()
+Date getCreationTime()
+IWhiteboardClient getOwner()
+boolean isShape()
+boolean isText()
Text
-String text
-float X
-float Y
+Text(String text, float X, float Y)
+String getString()
+float getX()
WhiteboardPanel
-ArrayList<IWhiteboardItem> shapes
+WhiteboardPanel()
+WhiteboardPanel(ArrayList<IWhiteboardItem> shapes)
+void paintComponent(Graphics g)
+void addItem(IWhiteboardItem item)
+void updateShapes(ArrayList<IWhiteboardItem> shapes)
-// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents void initComponents()
server
clientManager
server
context
clients
clientListeners
server
<<implements>>
itemListeners
serverManager
<<implements>>
shapes
text
client
ui
client
client
shapes
<<implements>>
0..*
1..1


:WhiteboardClientGUI
:WhiteboardClient
:CommunicationContext
:WhiteboardServer
:WhiteboardClientManager
:WhiteboardPanel
whiteboardPanel2MouseReleased()
this.WhiteboardClient
getContext()
getContext()
addShape(Line)
this.WhiteboardClientManager
WhiteboardClient.retrieveShapes(Shapes)
WhiteboardClientGUI.receiveShapes(Shapes)
WhiteboardPanel.addItem(WhiteboardItem)