The University of Melbourne

Assignment Cover Sheet

Student Number/Name:	Walid Moustafa – 563080
Subject Code/Name:	COMP90015: Distributed Systems
Assignment Title:	Assignment 2 (Distributed Whiteboard)
Tutorial Day/Time:	Wednesday 2:15pm
Tutor/Instructor Name:	Prof. Rajkumar Buyya, Dr. Rodrigo Calheiros
Due Date/Time:	04 Nov 2013
Submitted Date/Time:	04 Nov 2013

PLAGIARISM
Plagiarism is the presentation by a student of an assignment which has in fact been copied in whole or in part from another student's work, or from any other source (e.g. published books, periodicals, or the web) without due acknowledgement in the text.

COLLUSION
Collusion is the presentation by a student of an assignment as his or her own which is in fact the result in whole or in part of unauthorized collaboration with another person or persons.
Before submitting my assignment, I have:

1. Made a copy of the assignment and of any material submitted with the assignment
2. Ensured that my assignment and any material submitted are clearly identified.
3. Retained a copy of the email submission of this assignment (if appropriate).
4. Attached all files and required material to the email (if appropriate).

DECLARATION
I declare that this assignment is my own work and does not involve plagiarism or collusion. I also declare that the material contained in this assignment has not previously been submitted for assessment in any other formal course of study.

Signatures	Date
Walid Moustafa	04/11/2013

Note: For electronic submissions the signature may be typed.

Table of Contents

INTRODUCTION

This assignment is about implementing a distributed Whiteboard application. The whiteboard will be shared by several users connected to a central server using Java RMI (Remote Method Invocation) framework. The implemented architecture is a star schema with a simple messaging server in the centre and several desktop clients exchanging and synchronizing user interface as well as control events via RMI function calls the central server.

The first user connects to the server is granted the Admin role which authorizes her to receive following users requests-to-join and decide which requests to accept and grant access to the shared whiteboard and which requests to simply revoke and bounce out of the board.

Even authorized board users can be bounced out of the whiteboard at any time by the Admin user. Moreover, when the admin user quits the application, all connected board users are sent a notification message and all are bounced out and the board is cleared out.

By the end of a board session (which normally happens when all users quit or the Admin user terminates the running whiteboard session), the board server discards all the session data, but does NOT shutdown. Instead, the board server keeps waiting for the first user to connect to be granted the Admin role and start a brand new whiteboard session.

MAIN FEATURES

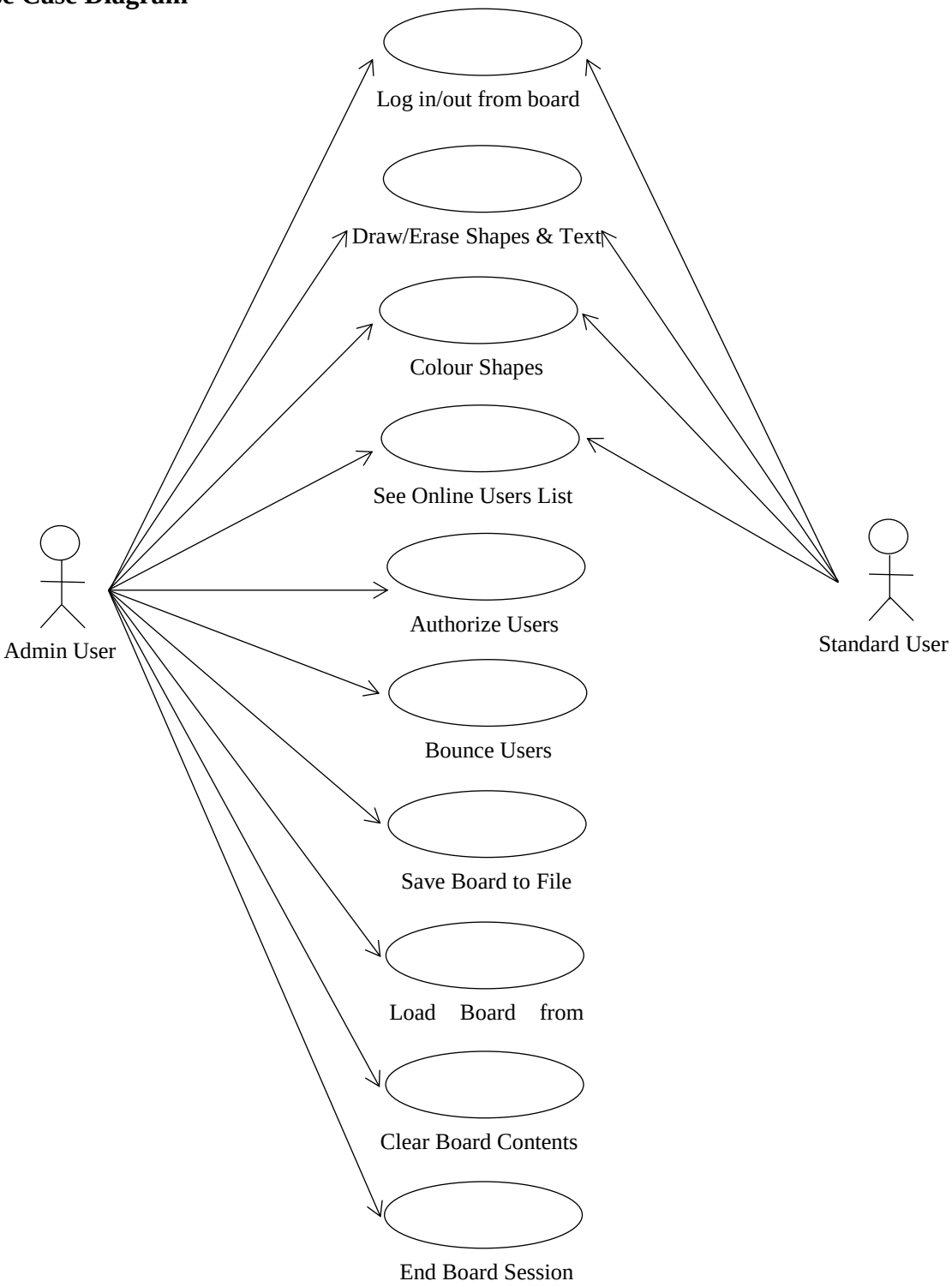
Real Time Whiteboard

The whiteboard application is designed to be “real time”. In other words, the sharing and synchronization of user GUI contributions, happens at the level of mouse and keyboard simple events like mouse clicks and keyboard hits. This level of concurrency enables the users to enjoy the highest level of collaboration and sharing. All users see exactly the same thing and can contribute to the same piece of artefact at the same time as oppose to sharing already completed shapes which erodes the feeling of real-time viewing and collaboration.

Serialization for Mutability

The board admin is able to save the current board to a file on disk and also load a previously saved board apply more collaborative changes to it (editing) with the other board users and re-save the modified version back to disk. To achieve this feature, the board was required to be saved in a mutable format. Apparently, saving the board as an image file would NOT achieve mutability. Java Serialization was leveraged to serialize all board objects (shapes) to a proprietary file so the original objects can be de-serialized back and loaded to memory for further edits and contributions.

Use Case Diagram



ARCHITECTURE

Client-Server Model

The application is designed as a centralized server and several clients representing the board users. Clients communicate and share board events as well as administrative message via the centralized server. In simple terms, the server acts as a message oriented middleware (MOM) accessed by the clients to add events to the centralized message queue when the user commits changes to the board, and poll the message queue for newly added events. Each client keeps record of the last message it has already receive and pools the server for only newer messages to avoid inconsistencies associated with omitting or re-processing events.
Communication via centralized server and avoiding inter-client direct communication makes the board design simpler and avoids concurrency and timing issues if clients were disseminating their events directly to each other. Interestingly enough, when a client commits some drawings or updates to the board, it is not displayed immediately on-screen. Instead, it is sent to the messaging server to take turn among the other users' drawings and comes back like any other global board event as part of server latest updates.

RMI Distribution Framework

Remote Message Invocation (RMI) is the technology used by the clients to communicate to the centralized server. Submitting user events to the centralized message queue and polling the server for new messages take the form of RMI remote method calls.
RMI was picked as the distribution framework due to the following two reasons:

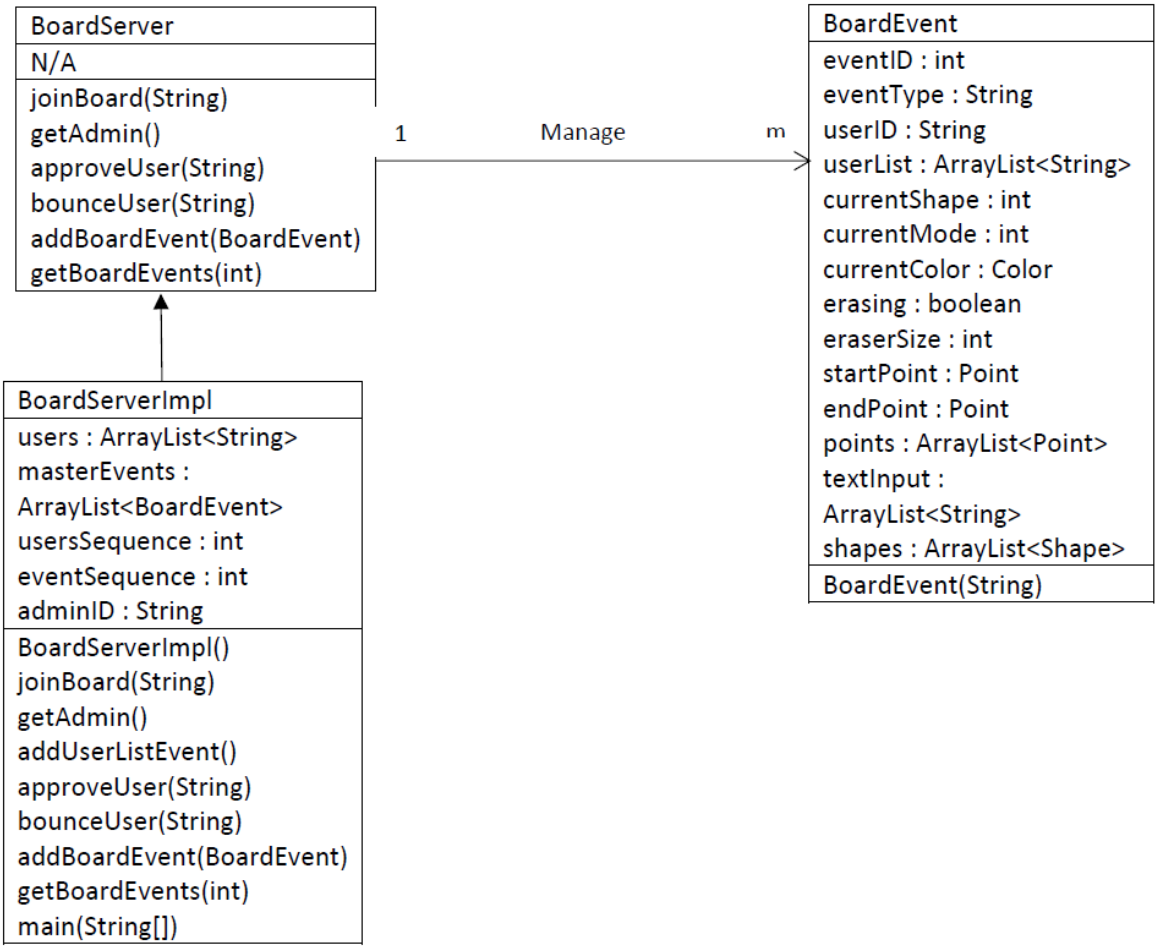
- Take advantage of the simplicity and transparency RMI provides compared to Socket programming. RMI takes care of the messaging and communication details like error handling, concurrency, object marshalling and un-marshalling. This makes the distance communication becomes transparent and remote services are accessed as simple as local method calls whereas it is not.
- RMI is a lightweight framework if compared to Enterprise Java Beans (EJB) or Web Services. It introduces much less overhead which makes it more efficient for interactive application like the distributed board. Also, it is already provided by the Java virtual machine (JSE) and does not require server installation. This important advantage emphasizes the application portability and enables the whiteboard admin to launch the messaging server on any Java virtual machine.

Multithreaded Clients

Each client runs as exactly two threads; one main thread for getting user input and forwarding it to the centralized messaging server via RMI method calls, and another thread (EventDispatcher) that keeps on polling the centralized server for recent events, and updates the Graphical User Interface (GUI) accordingly. This multithreaded client design avoids locking the user experience waiting for remote server call or the other way around.

SERVER DESIGN

Server Class Diagram



Server Classes Described

BoardServer

This is the RMI service interface (extending Remote) that includes the services provided to RMI clients. The lsit of services available are all around submitting events or polling the latest events. For instance, addBoardEvent() and getBoardEvents() are sample server methods. All server methods throw RemoteException.

BoardServerImpl

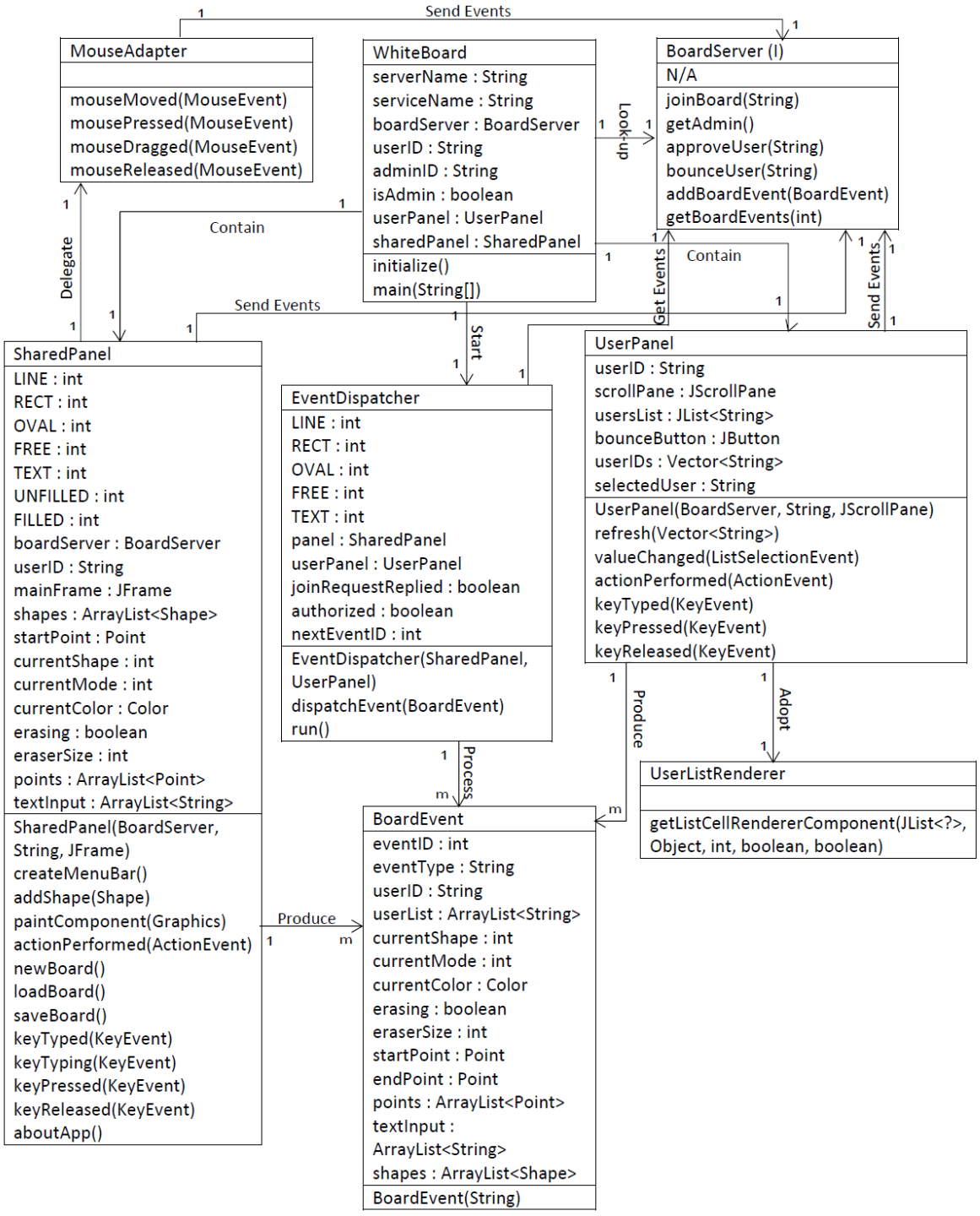
This is the RMI service implementation class (extending UnicastRemoteObject class and implements BoardServer interface). It implements the methods of BoardServer Interface and also includes two important member variables; users ArrayList which maintains a list of the board users and masterEvents ArrayList which maintains a list of the board events (Admin and GUI events).

BoardEvent

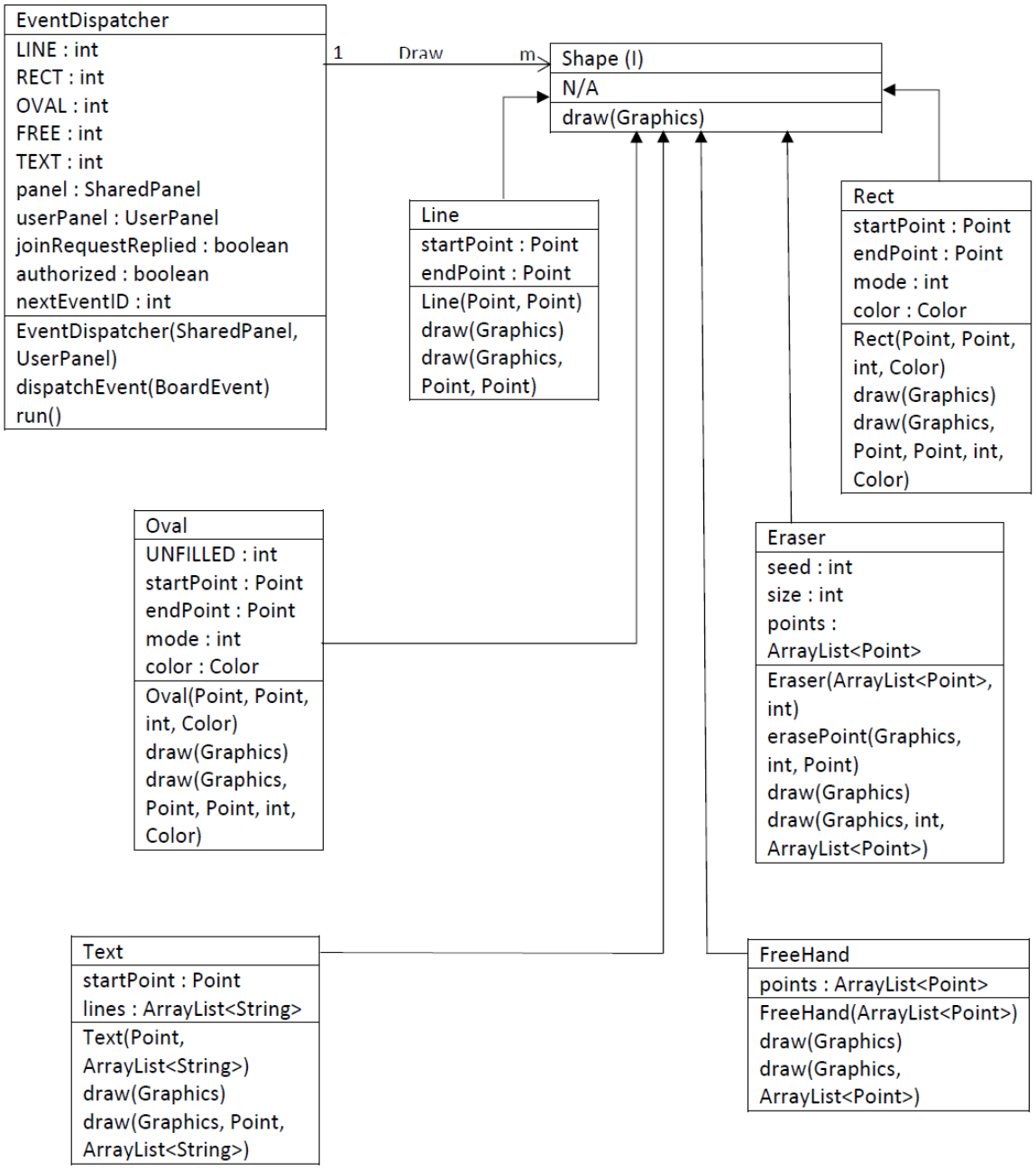
This class represents the board events exchanged among the board server and the clients. It implements Serializable to suit RMI methos calls as exchanged parameter ad also to beable to be serialized to and from disk files. It includes eventID, eventType as well as some GUI related attributes. Event type takes some Admin values like, "joinRequest" and "userList" as well as GUI related values like "keyTyped" and "mouseDragged".

CLIENT DESIGN

Client Class Diagram



Client Class Diagram (Cont.)



Client Classes Described

WhiteBoard

This is the client main class (with the main() method). It looks-up and connects to the RMI server and displays the client application main frame window.

It also spawns another thread for the EventDispatcher class that keeps polling the server for board events and updates the GUI accordingly.

SharedPanel

This class extends JPanel and represents the drawing board. It implements the ActionListener and KeyListener interfaces to capture the menu and kebaord events and includes MouseAdapter inner class that extends the MouseInputAdapter class to capture the mouse events. Captured events are sent to the RMI server via calling the boardServer.addBoardEvent() remote method. It also includes the board serialization methods; saveBoard() which saves the board shapes array-list to a disk file with the extension (.brd) and loadBoard() method which reads back the shapes array-list from a (.brd) file.

UserPanel

This class extends JPanel and displays the board user list pane. It handles the Admin user action on the user panel (picking a user and bouncing him out of the board application). Also includes the refresh() member method, that is frequently called by the EventDispatcher class when receiving events of type "userList" from the server. The refresh() method loads the userList member variable of class (javax.swing.JList) with the most recent board user list.

UserListRenderer

This is a secondary class that extends javax.swing.DefaultListCellRenderer class. Its main purpose it to define the coloring theme of the userList (javax.swing.JList), the member variable of the UserPanel.

EvenDispatcher

This class runs in its own thread and implements Runnable interface. Its run() method keeps calling the server method boardServer.getBoardEvents() and forwards each retrieved event to the member method dispatchEvent() that uses the Sharedpanel.getGraphics() method to obtain a Graphics context and updates the shared panel with the latest drawing events. In case of user list events, it calls userPanel.refresh() method to refresh the displayed panel users list.

Shape

This is the interface that all objects drawn on the board should implement. Indeed, all board drawings are store in SharedPanel.shapes (a member variable of type ArrayList<Shape>). This interface got one method draw() that accepts a graphics context as a parameter.

All the classes implementing the Shape interface has two draw() methods; one member object method that draws the object when called by the EventDispatcher.dispatchEvent() method and the other is a static method (class method) that draws the object before it is finalized and created (draws the intermediate drawing) to give the end user a better drawing experience while he is dragging the mouse to draw a particular object on the board.

Line

This class represents the line shape and implements the Shape and Serializable interfaces. It stores the line object as start and end points and draws it on the board when the EventDispatcher calls its member method Line.draw() (the Static or the object method).

Rect

This class represents the rectangle shape and implements the Shape and Serializable interfaces. It stores the rectangle object as start and end points, and mode (filled/unfilled) and colour. It draws the rectangle on the board when the EventDispatcher calls its member method Rect.draw() (the Static or the object method).

Oval

This class represents the oval shape and implements the Shape and Serializable interfaces. It stores the oval object as start and end points, and mode (filled/unfilled) and colour. It draws the oval on the board when the EventDispatcher calls its member method Oval.draw() (the Static or the object method).

FreeHand

This class represents the freehand shape (drawing) and implements the Shape and Serializable interfaces. It stores the freehand object as a list of consecutive points. It draws the free drawing on the board when the EventDispatcher calls its member method FreeHand.draw() (the Static or the object method).

Text

This class represents the user text typing on the board and also implements the Shape and Serializable interfaces. It stores the text object as a list of strings, a new string is started each time the user hits the Enter key. This enables the user to draw multi-line text on the board. It draws the text on the board when the EventDispatcher calls its member method Text.draw() (the Static or the object method).

Eraser

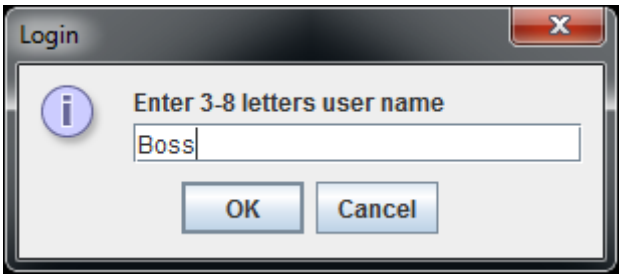
This class represents the eased areas on the board. Actually, erased areas are drawn as a sequence of small rectangles (size of the used rectangle depends on the picked eraser size). The eraser uses the background colour to draw the erasing filled rectangles and hence wipes the other objects below. Erased area is represented as a sequence of points. These points are the centres of the erased rectangles. Eraser got two attributes that determine the size (seed, size).

Snapshots

The following subsections will present some of the application snapshots. The Admin snapshots section will present screens related to user administration tasks related to entering and leaving the board. The Art snapshot will focus on enabling authorized whiteboard users to collaborate and share real-time free-hand drawings, standard shapes and text using different colours.

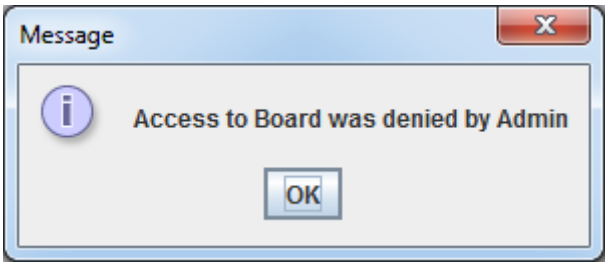
Admin Snapshots

User Login



Join Request to Admin

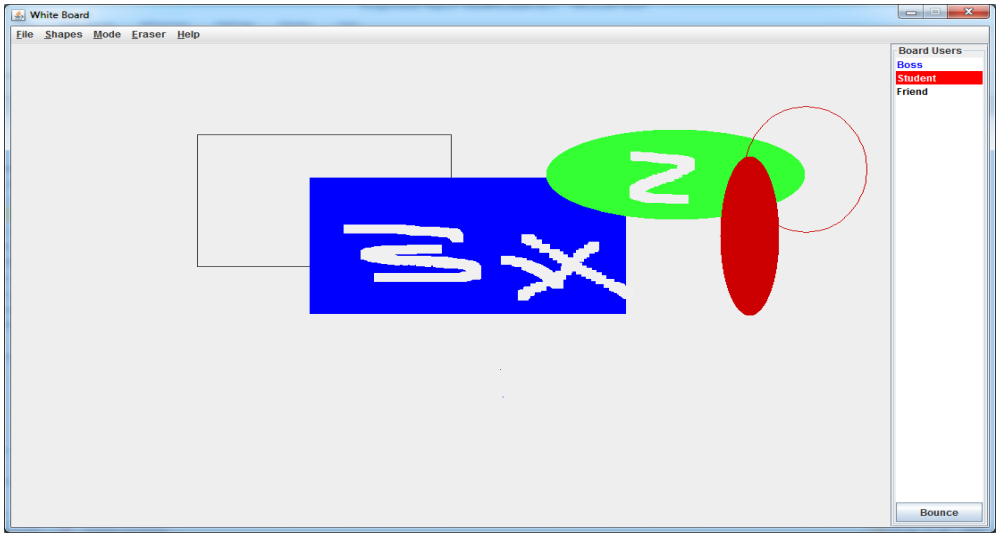
Access Denial by Admin



User List for Admin user

Hints:

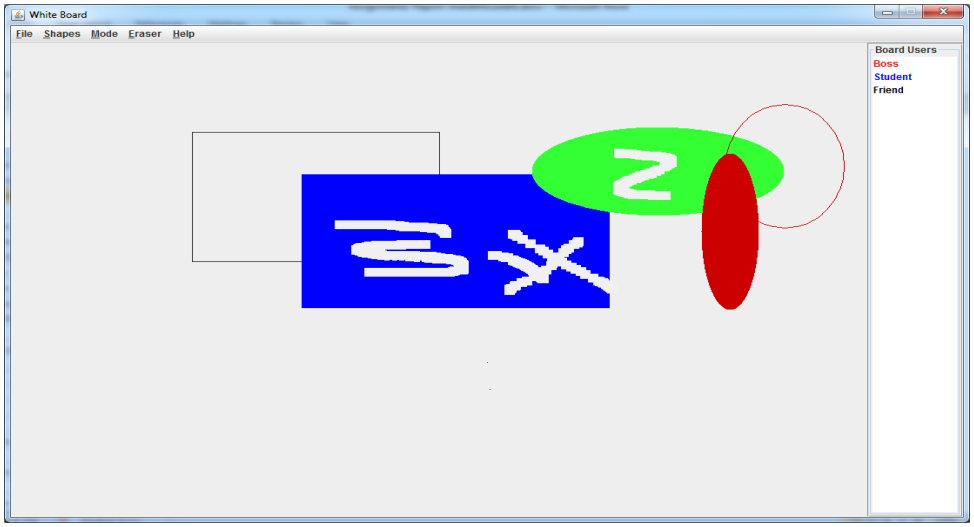
1. The “Bounce” button is only available to the Admin user
2. The user list displays the current user name in blue
3. The user list displays the selected user name in white on red background
4. Clicking the “Bounce button” will kick out the selected user name.



User List for ordinary users

Hints:

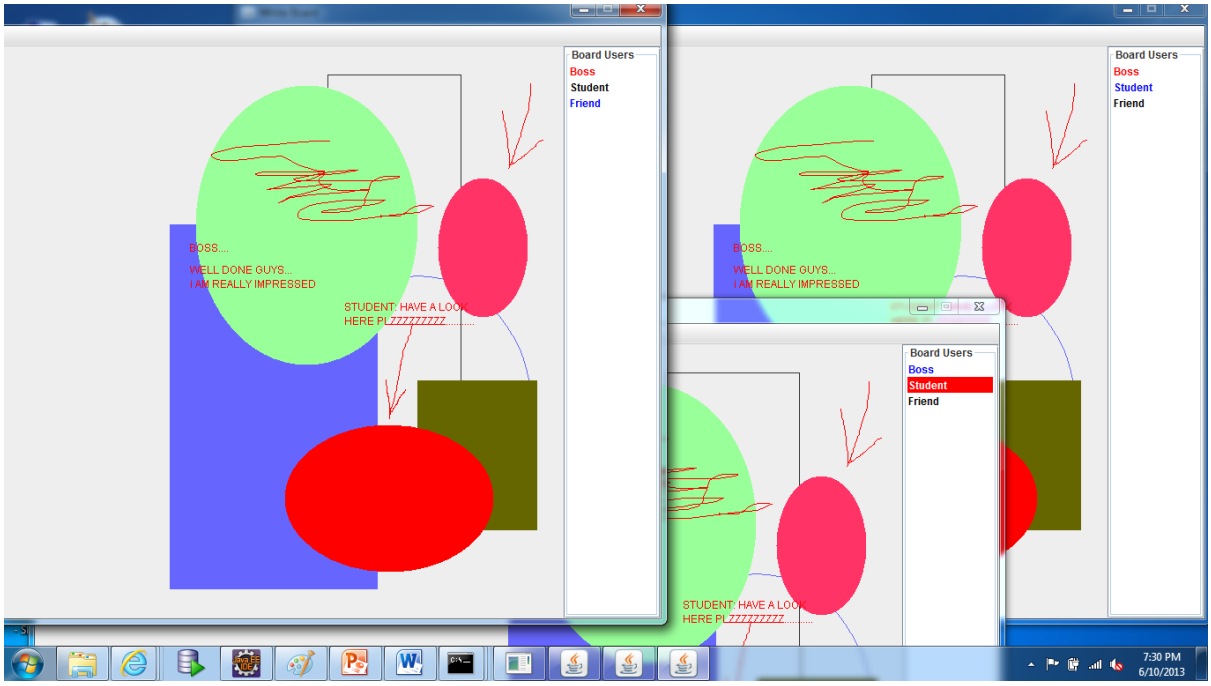
1. The “Bounce” button is NOT available to the ordinary users
2. The user list displays the Admin user name in red
3. The user list displays the current user name in blue



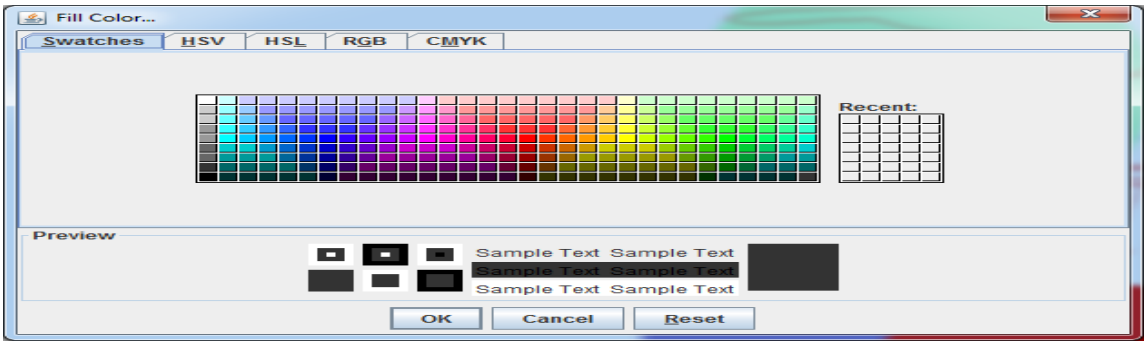
Art Snapshots

Collaboration

The below collaborative art snapshot shows three users collaborating to produce coloured shapes and leveraging pointing arrows and text messages to communicate viewpoints.



Colour Palette



SERVER SOURCE CODE

BoardEvent.java

```
/*
 * Name: Walid Moustafa
 * Student ID: 563080
 * Subject: COMP90015 - Distributed Systems
 * Assignment: Assignment 2 - Distributed Whiteboard
 * Project: BoardServer
 * File: BoardEvent.java
*/

import java.awt.*;
import java.io.Serializable;
import java.util.ArrayList;


public class BoardEvent implements Serializable {

	public int eventID;
	public String eventType;
	public String userID;
	public ArrayList<String> userList;
	public int currentShape;
	public int currentMode;
	public Color currentColor;
	public boolean erasing;
	public int eraserSize;
	public Point startPoint;
	public Point endPoint;
	public ArrayList<Point> points;
	public ArrayList<String> textInput;
	public ArrayList<Shape> shapes;

	public BoardEvent(String eType) {
		eventType = eType;
	}

	private static final long serialVersionUID = 1L;

}
```

BoardServer.java

```
/*
 * Name: Walid Moustafa
 * Student ID: 563080
 * Subject: COMP90015 - Distributed Systems
 * Assignment: Assignment 2 - Distributed Whiteboard
 * Project: BoardServer
 * File: BoardServer.java
*/

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.ArrayList;


public interface BoardServer extends Remote {

	String joinBoard(String candidateID) throws RemoteException;
```

```
	String getAdmin() throws RemoteException;
	void approveUser(String userID) throws RemoteException;
	void bounceUser(String userID) throws RemoteException;
	void addBoardEvent(BoardEvent event) throws RemoteException;
	ArrayList<BoardEvent> getBoardEvents(int startFrom) throws RemoteException;

}
```

BoardServerImpl.java

```
/*
 * Name: Walid Moustafa
 * Student ID: 563080
 * Subject: COMP90015 - Distributed Systems
 * Assignment: Assignment 2 - Distributed Whiteboard
 * Project: BoardServer
 * File: BoardServerImpl.java
*/

import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;


public class BoardServerImpl extends UnicastRemoteObject implements BoardServer {

	protected BoardServerImpl() throws RemoteException {
		super();
	}

	ArrayList<String> users = new ArrayList<String>();
	ArrayList<BoardEvent> masterEvents;
	int usersSequence;
	int eventSequence;
	String adminID;

	@Override
	public String joinBoard(String candidateID) {
		String userID;
		ArrayList<String> list = null;
		synchronized(users) {
			list = new ArrayList<String>(users);
		}

		// check if no current board users exist
		boolean usersExist = false;
		if(list != null) {
			for(String auser : list) {
				if(auser.charAt(0) != '#') {
					usersExist = true;
					break;
				}
			}
		}

		if ((list == null) || (!usersExist)) {
			// start a fresh new board
			users = new ArrayList<String>();
```

```
			masterEvents = new ArrayList<BoardEvent>();
			usersSequence = 0;
			eventSequence = 0;
			adminID = candidateID;
			userID = candidateID;
			approveUser(userID);
		} else {
			for (String auser : list) {
				if(auser.charAt(0) == '#') {
					auser = auser.substring(1);
				}
				if(auser.equals(candidateID)) {
					candidateID = candidateID + usersSequence;
					break;
				}
			}
			userID = candidateID;
			BoardEvent event = new BoardEvent("joinRequest");
			event.userID = userID;
			addBoardEvent(event);
		}
		usersSequence++;
		return userID;
	}

	@Override
	public String getAdmin()  {
		return adminID;
	}

	public void addUserListEvent () {
		BoardEvent event = new BoardEvent("userList");
		synchronized(users) {
			event.userList = new ArrayList<String>(users);
		}
		addBoardEvent(event);
	}


	@Override
	public void approveUser(String userID) {
		synchronized(users) {
			users.add(userID);
		}
		addUserListEvent();
	}

	@Override
	public void bounceUser(String userID) {
		boolean found = false;
		ArrayList<String> list = null;
		synchronized(users) {
			list = new ArrayList<String>(users);
		}
		for (String auser:list) {
			if(auser.equals("#" + userID)) {
				found = true;
				break;
			} else if(auser.equals(userID)) {
```

```
                        found = true;
                        int idx = list.indexOf(auser);
                        synchronized(users) {
                              users.set(idx, "#" + userID);
                        }
                        break;
                  }
            }
            if(! found) {
                  synchronized(users) {
                        users.add("#" + userID);
                  }
            }
            addUserListEvent();
      }

      @Override
      public synchronized void addBoardEvent(BoardEvent event) {
            event.eventID = eventSequence++;
            masterEvents.add(event);
      }

      @Override
      public synchronized ArrayList<BoardEvent> getBoardEvents(int startFrom) {
            return new ArrayList<BoardEvent>(
                        masterEvents.subList(startFrom, masterEvents.size()));
      }

      public static void main(String args[])
    {
        try
        {
            String serverName = "localhost";
            String serviceName = "BoardServer";

            //System.setSecurityManager(new RMISecurityManager());

            BoardServer obj = new BoardServerImpl();
           // Bind this object instance to the name "BoardServer"
           Naming.rebind("rmi://"+serverName+"/"+serviceName, obj);

           System.out.println("Successful rebind service call");

        }
        catch (Exception e)
        {
            System.out.println("BoardServerImpl err: " + e.getMessage());
            e.printStackTrace();
        }
    }

      private static final long serialVersionUID = 1L;
}
```

CLIENT SOURCE CODE

Eraser.java

```
/*
 * Name: Walid Moustafa
 * Student ID: 563080
 * Subject: COMP90015 - Distributed Systems
 * Assignment: Assignment 2 - Distributed Whiteboard
 * Project: WhiteBoard
 * File: Eraser.java
*/

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Point;
import java.io.Serializable;
import java.util.ArrayList;


public class Eraser implements Shape, Serializable
{
	static final int seed = 3;
	private int size = 1;
	private ArrayList<Point> points = new ArrayList<Point>();

    public Eraser(ArrayList<Point> pts, int size)
    {
      this.size = size;
      points = pts;
    }

    public static void erasePoint(Graphics gfx, int size, Point point)
      {
		size = size * seed;
      int x = (point.x - size) > 0 ? (point.x - size) : 0;
		int y = (point.y - size) > 0 ? (point.y - size) : 0;
		int width = size * 2;
		int height = size * 2;

		gfx.fillRect(x, y, width, height);
      }

      @Override
      public void draw(Graphics gfx)
      {
		Color clr = gfx.getColor();
		gfx.setColor(new Color(240, 240, 240));
		for (Point point : points) {
			erasePoint(gfx, size, point);
	  }
		gfx.setColor(clr);
      }

      public static void draw(Graphics gfx, int size, ArrayList<Point> points)
      {
		Color clr = gfx.getColor();
		gfx.setColor(new Color(240, 240, 240));
		for (Point point : points) {
```

```
                    erasePoint(gfx, size, point);
             }
                  gfx.setColor(clr);
            }

            private static final long serialVersionUID = 1L;
}
```

EventDispatcher.java

```
/*
 * Name: Walid Moustafa
 * Student ID: 563080
 * Subject: COMP90015 - Distributed Systems
 * Assignment: Assignment 2 - Distributed Whiteboard
 * Project: WhiteBoard
 * File: EventDispatcher.java
*/

import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.Vector;

import javax.swing.JOptionPane;



public class EventDispatcher implements Runnable {

            final int LINE = 0;
            final int RECT = 1;
            final int OVAL = 2;
            final int FREE = 3;
            final int TEXT = 4;

            SharedPanel panel;
            UserPanel userPanel;
            boolean joinRequestReplied = false;
```

```
	boolean authorized = false;


	int nextEventID = 0;


	public EventDispatcher(SharedPanel mainPanel, UserPanel uPanel) {

		panel = mainPanel;

		userPanel = uPanel;

	}


	public void dispatchEvent(BoardEvent event) {

		if (event.eventType.equals("Terminate")) {

			JOptionPane.showMessageDialog(panel.mainFrame,	"Whiteboard
Session was terminated by Admin",

				"Session				Terminated",
JOptionPane.INFORMATION_MESSAGE);

			try {

				panel.boardServer.bounceUser(WhiteBoard.userID);

			} catch (RemoteException e) {

				e.printStackTrace();

			}

			System.exit(0);

		} else if (event.eventType.equals("joinRequest")) {

			int choice = JOptionPane.showConfirmDialog(panel.mainFrame,

				event.userID + " wants to join the Whiteboard.
Allow user?",

				"Join Request", JOptionPane.YES_NO_OPTION);

			try {

				if(choice == JOptionPane.YES_OPTION) {

					WhiteBoard.boardServer.approveUser(event.userID);

				} else {

					WhiteBoard.boardServer.bounceUser(event.userID);

				}

			} catch (RemoteException e1) {
```

```
                        e1.printStackTrace();
                }
            } else if (event.eventType.equals("userList")) {
                authorized = false;
                for (String auser:event.userList) {
                    if(auser.equals(WhiteBoard.userID)) {
                        joinRequestReplied = true;
                        authorized = true;
                        break;
                    } else if(auser.equals("#" + WhiteBoard.userID)) {
                        joinRequestReplied = true;
                        authorized = false;
                        break;
                    }
                }
                if(joinRequestReplied) {
                    if(authorized) {
                        panel.requestFocusInWindow();
                        panel.mainFrame.setVisible(true);
                        Vector<String> uIDs = new
Vector<String>(event.userList);
                        userPanel.refresh(uIDs);
                    } else {
                        JOptionPane.showMessageDialog(panel.mainFrame,
"Access to Board was denied by Admin",
                                "Access Denied",
JOptionPane.INFORMATION_MESSAGE);
                        System.exit(0);
                    }
                }
            } else if (event.eventType.equals("loadBoard")) {
                panel.shapes = event.shapes;
                panel.mainFrame.repaint();
```

```
            } else if (event.eventType.equals("keyTyped")) {
                  Text.draw(panel.getGraphics(),               event.startPoint,
event.textInput);
            } else if (event.eventType.equals("mouseMoved")) {
                  panel.addShape(new Text(event.startPoint, event.textInput));
            } else if (event.eventType.equals("mouseDragged")) {
                  if(event.erasing) {
                        Eraser.draw(panel.getGraphics(),       event.eraserSize,
event.points);
                  } else {
                        switch (event.currentShape)
                        {
                        case LINE:
                              Line.draw(panel.getGraphics(),   event.startPoint,
event.endPoint);
                              break;
                        case RECT:
                              Rect.draw(panel.getGraphics(),   event.startPoint,
event.endPoint, event.currentMode, event.currentColor);
                              break;
                        case OVAL:
                              Oval.draw(panel.getGraphics(),   event.startPoint,
event.endPoint, event.currentMode, event.currentColor);
                              break;
                        case FREE:
                              FreeHand.draw(panel.getGraphics(), event.points);
                              break;
                        case TEXT:
                              break;
                        default:
                              System.err.println("Unsupported shape type");
                              break;
                        }
                  }
```

```
                    panel.mainFrame.repaint();
             } else if (event.eventType.equals("mouseReleased")) {
                    if(event.erasing) {
                          panel.addShape(new                  Eraser(event.points,
event.eraserSize));
                    } else {
                          switch (event.currentShape) {
                          case LINE:
                                panel.addShape(new         Line(event.startPoint,
event.endPoint));
                              break;
                          case RECT:
                                panel.addShape(new         Rect(event.startPoint,
event.endPoint, event.currentMode, event.currentColor));
                              break;
                          case OVAL:
                                panel.addShape(new         Oval(event.startPoint,
event.endPoint, event.currentMode, event.currentColor));
                              break;
                          case FREE:
                                panel.addShape(new FreeHand(event.points));
                                break;
                          case TEXT:
                                break;
                          default:
                                System.err.println("Unsupported shape type");
                              break;
                          }
                    }
                    panel.mainFrame.repaint();
             }
       }

       @Override
```

```
	public void run() {
		while (true) {
			ArrayList<BoardEvent> boardEvents;
			try {
				boardEvents = panel.boardServer.getBoardEvents(nextEventID);

				for (BoardEvent event : boardEvents) {
					if(event.eventType.equals("joinRequest") && WhiteBoard.isAdmin) {
						dispatchEvent(event);
					}
				}

				BoardEvent latestUserList = null;
				for (BoardEvent event : boardEvents) {
					if(event.eventType.equals("userList")) {
						latestUserList  = event;
					}
				}

				if(latestUserList != null) {
					dispatchEvent(latestUserList);
				}

				if(!joinRequestReplied){
					continue;
				}

				for (BoardEvent event : boardEvents) {
					if(!(event.eventType.equals("joinRequest") || event.eventType.equals("userList"))){
						dispatchEvent(event);
```

```
                                }

                        }


                        if (boardEvents.size() > 0) {

                                nextEventID = boardEvents.get(boardEvents.size()-
1).eventID + 1;

                        }

                } catch (RemoteException e) {

                        e.printStackTrace();

                }

        }

      }


}
```

FreeHand.java

```
/*
 * Name: Walid Moustafa
 * Student ID: 563080
 * Subject: COMP90015 - Distributed Systems
 * Assignment: Assignment 2 - Distributed Whiteboard
 * Project: WhiteBoard
 * File: FreeHand.java
*/

import java.awt.Graphics;
import java.awt.Point;
import java.io.Serializable;
import java.util.ArrayList;


public class FreeHand implements Shape, Serializable
{
    private ArrayList<Point> points = new ArrayList<Point>();

    public FreeHand(ArrayList<Point> pts)
    {
       points = pts;
    }

       @Override
       public void draw(Graphics gfx)
       {
              for (int i = 1; i < points.size(); i++)
        {
            Point p1 = points.get(i - 1);
            Point p2 = points.get(i);
            gfx.drawLine(p1.x, p1.y, p2.x, p2.y);
```

```
        }
    }

    public static void draw(Graphics gfx, ArrayList<Point> points)
    {
        for (int i = 1; i < points.size(); i++)
        {
            Point p1 = points.get(i - 1);
            Point p2 = points.get(i);
            gfx.drawLine(p1.x, p1.y, p2.x, p2.y);
        }
    }

    private static final long serialVersionUID = 1L;
}
```

Line.java

```
/*
 * Name: Walid Moustafa
 * Student ID: 563080
 * Subject: COMP90015 - Distributed Systems
 * Assignment: Assignment 2 - Distributed Whiteboard
 * Project: WhiteBoard
 * File: Line.java
*/

import java.awt.*;

public class Line implements Shape {

    Point startPoint;
    Point endPoint;

    public Line (Point sPoint, Point ePoint) {
        startPoint = sPoint;
        endPoint = ePoint;
    }

    @Override
    public void draw (Graphics gfx) {
        gfx.drawLine(startPoint.x, startPoint.y, endPoint.x, endPoint.y);
    }

    public static void draw (Graphics gfx, Point sPoint, Point ePoint)
    {
        gfx.drawLine(sPoint.x, sPoint.y, ePoint.x, ePoint.y);
    }

}
```

Oval.java

```
/*
 * Name: Walid Moustafa
 * Student ID: 563080
 * Subject: COMP90015 - Distributed Systems
 * Assignment: Assignment 2 - Distributed Whiteboard
 * Project: WhiteBoard
```

```
 * File: Oval.java
*/

import java.awt.*;

public class Oval implements Shape {

	Point startPoint;
	Point endPoint;
	int mode;
	Color color;

	public Oval (Point sPoint, Point ePoint, int currentMode, Color
currentColor) {
		startPoint = sPoint;
		endPoint = ePoint;
		mode = currentMode;
		color = currentColor;
	}

	@Override
	public void draw (Graphics gfx) {
		int x, y, width, height;

		if (startPoint.x < endPoint.x ) {
			x = startPoint.x;
			width = endPoint.x - startPoint.x;
		} else {
			x = endPoint.x;
			width = startPoint.x - endPoint.x;
		}

		if (startPoint.y < endPoint.y ) {
			y = startPoint.y;
			height = endPoint.y - startPoint.y;
		} else {
			y = endPoint.y;
			height = startPoint.y - endPoint.y;
		}

		if(mode == SharedPanel.UNFILLED) {
			gfx.drawOval(x, y, width, height);
		} else {
			gfx.setColor(color);
			gfx.fillOval(x, y, width, height);
		}
	}

	public static void draw (Graphics gfx, Point sPoint, Point ePoint, int
currentMode, Color currentColor) {
		int x, y, width, height;

		if (sPoint.x < ePoint.x ) {
			x = sPoint.x;
			width = ePoint.x - sPoint.x;
		} else {
			x = ePoint.x;
			width = sPoint.x - ePoint.x;
		}
```

```
		if (sPoint.y < ePoint.y ) {
			y = sPoint.y;
			height = ePoint.y - sPoint.y;
		} else {
			y = ePoint.y;
			height = sPoint.y - ePoint.y;
		}

		if(currentMode == SharedPanel.UNFILLED) {
			gfx.drawOval(x, y, width, height);
		} else {
			gfx.setColor(currentColor);
			gfx.fillOval(x, y, width, height);
		}
	}

}
```

Rect.java

```
/*
 * Name: Walid Moustafa
 * Student ID: 563080
 * Subject: COMP90015 - Distributed Systems
 * Assignment: Assignment 2 - Distributed Whiteboard
 * Project: WhiteBoard
 * File: Rect.java
*/

import java.awt.*;

public class Rect implements Shape {

	Point startPoint;
	Point endPoint;
	int mode;
	Color color;

	public Rect (Point sPoint, Point ePoint, int currentMode, Color
currentColor) {
		startPoint = sPoint;
		endPoint = ePoint;
		mode = currentMode;
		color = currentColor;
	}


	@Override
	public void draw (Graphics gfx) {
		int x, y, width, height;

		if (startPoint.x < endPoint.x ) {
			x = startPoint.x;
			width = endPoint.x - startPoint.x;
		} else {
			x = endPoint.x;
			width = startPoint.x - endPoint.x;
		}
```

```
            if (startPoint.y < endPoint.y ) {
                  y = startPoint.y;
                  height = endPoint.y - startPoint.y;
            } else {
                  y = endPoint.y;
                  height = startPoint.y - endPoint.y;
            }

            if(mode == SharedPanel.UNFILLED) {
                  gfx.drawRect(x, y, width, height);
            } else {
                  gfx.setColor(color);
                  gfx.fillRect(x, y, width, height);
            }
      }

      public static void draw (Graphics gfx, Point sPoint, Point ePoint, int
currentMode, Color currentColor) {
            int x, y, width, height;

            if (sPoint.x < ePoint.x ) {
                  x = sPoint.x;
                  width = ePoint.x - sPoint.x;
            } else {
                  x = ePoint.x;
                  width = sPoint.x - ePoint.x;
            }

            if (sPoint.y < ePoint.y ) {
                  y = sPoint.y;
                  height = ePoint.y - sPoint.y;
            } else {
                  y = ePoint.y;
                  height = sPoint.y - ePoint.y;
            }

            if(currentMode == SharedPanel.UNFILLED) {
                  gfx.drawRect(x, y, width, height);
            } else {
                  gfx.setColor(currentColor);
                  gfx.fillRect(x, y, width, height);
            }
      }

}
```

Shape.java

```
/*
 * Name: Walid Moustafa
 * Student ID: 563080
 * Subject: COMP90015 - Distributed Systems
 * Assignment: Assignment 2 - Distributed Whiteboard
 * Project: WhiteBoard
 * File: Shape.java
*/

import java.awt.*;
```

```
public interface Shape
{

	public void draw (Graphics gfx);

}
```

SharedPanel.java

```
/*
 * Name: Walid Moustafa
 * Student ID: 563080
 * Subject: COMP90015 - Distributed Systems
 * Assignment: Assignment 2 - Distributed Whiteboard
 * Project: WhiteBoard
 * File: SharedPanel.java
*/

import java.awt.*;
import java.awt.event.*;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.rmi.RemoteException;
import java.util.ArrayList;
import javax.swing.*;
import javax.swing.event.MouseInputAdapter;
import javax.swing.filechooser.FileFilter;
import javax.swing.filechooser.FileNameExtensionFilter;

public class SharedPanel extends JPanel implements ActionListener, KeyListener {

	final int LINE = 0;
	final int RECT = 1;
	final int OVAL = 2;
	final int FREE = 3;
	final int TEXT = 4;

	public static final int UNFILLED = 0;
	public static final int FILLED = 1;

	BoardServer boardServer;
	String userID;
	JFrame mainFrame;
	ArrayList<Shape> shapes = new ArrayList<Shape>();

	Point startPoint = new Point(5, 10);
	int currentShape = LINE;
	int currentMode = UNFILLED;
	Color currentColor = getForeground();
	boolean erasing = false;
	int eraserSize = 1;
	ArrayList<Point> points;
	ArrayList<String> textInput;
```

```
	public SharedPanel(BoardServer bServer, String uID, JFrame frame)
	{
		boardServer = bServer;
		userID = uID;
		mainFrame = frame;
		MouseAdapter mouseAdapter = new MouseAdapter();
		addMouseListener(mouseAdapter);
		addMouseMotionListener(mouseAdapter);
		addKeyListener(this);
		createMenuBar();
	}

	private void createMenuBar() {

		JMenuBar menuBar = new JMenuBar();


		JMenu menu = new JMenu("File");
		menu.setMnemonic(KeyEvent.VK_F);

		if(WhiteBoard.isAdmin) {
			JMenuItem menuItem = new JMenuItem("New Board");
			menuItem.setActionCommand("NewBoard");
			menuItem.addActionListener(this);
			menuItem.setMnemonic(KeyEvent.VK_N);
			menu.add(menuItem);

			menuItem = new JMenuItem("Open File...");
			menuItem.setActionCommand("OpenFile");
			menuItem.addActionListener(this);
			menuItem.setMnemonic(KeyEvent.VK_O);
			menu.add(menuItem);

			menu.addSeparator();

			menuItem = new JMenuItem("Save As...");
			menuItem.setActionCommand("SaveAs");
			menuItem.addActionListener(this);
			menuItem.setMnemonic(KeyEvent.VK_S);
			menu.add(menuItem);

			menu.addSeparator();
		}

		menu.add(new AbstractAction("Exit") {
			@Override
			public void actionPerformed(ActionEvent e) {
				if (mainFrame.isActive())
				{
					WindowEvent windowClosing = new
WindowEvent(mainFrame,
							WindowEvent.WINDOW_CLOSING);
					mainFrame.dispatchEvent(windowClosing);
				}
			}
			private static final long serialVersionUID = 1L;
		});

		menuBar.add(menu);
```

```
menu = new JMenu("Shapes");
menu.setMnemonic(KeyEvent.VK_S);
JRadioButtonMenuItem rbMenuItem;

ButtonGroup group = new ButtonGroup();

rbMenuItem = new JRadioButtonMenuItem("Line");
rbMenuItem.setActionCommand("Line");
rbMenuItem.addActionListener(this);
rbMenuItem.setSelected(true);
rbMenuItem.setMnemonic(KeyEvent.VK_L);
group.add(rbMenuItem);
menu.add(rbMenuItem);

rbMenuItem = new JRadioButtonMenuItem("Rectangle");
rbMenuItem.setActionCommand("Rectangle");
rbMenuItem.addActionListener(this);
rbMenuItem.setMnemonic(KeyEvent.VK_R);
group.add(rbMenuItem);
menu.add(rbMenuItem);

rbMenuItem = new JRadioButtonMenuItem("Oval");
rbMenuItem.setActionCommand("Oval");
rbMenuItem.addActionListener(this);
rbMenuItem.setMnemonic(KeyEvent.VK_O);
group.add(rbMenuItem);
menu.add(rbMenuItem);

rbMenuItem = new JRadioButtonMenuItem("Free Hand");
rbMenuItem.setActionCommand("Free Hand");
rbMenuItem.addActionListener(this);
rbMenuItem.setMnemonic(KeyEvent.VK_F);
group.add(rbMenuItem);
menu.add(rbMenuItem);

rbMenuItem = new JRadioButtonMenuItem("Text");
rbMenuItem.setActionCommand("Text");
rbMenuItem.addActionListener(this);
rbMenuItem.setMnemonic(KeyEvent.VK_T);
group.add(rbMenuItem);
menu.add(rbMenuItem);

menuBar.add(menu);

JMenu menuMode = new JMenu("Mode");
menuMode.setMnemonic(KeyEvent.VK_M);

group = new ButtonGroup();

rbMenuItem = new JRadioButtonMenuItem("Unfilled");
rbMenuItem.setActionCommand("Unfilled");
rbMenuItem.addActionListener(this);
rbMenuItem.setSelected(true);
rbMenuItem.setMnemonic(KeyEvent.VK_U);
group.add(rbMenuItem);
menuMode.add(rbMenuItem);

rbMenuItem = new JRadioButtonMenuItem("Filled");
```

```
rbMenuItem.setActionCommand("Filled");
rbMenuItem.addActionListener(this);
rbMenuItem.setMnemonic(KeyEvent.VK_F);
group.add(rbMenuItem);
menuMode.add(rbMenuItem);

menuMode.addSeparator();

JMenuItem menuItem = new JMenuItem("Colors");
menuItem.setActionCommand("Colors");
menuItem.addActionListener(this);
menuItem.setMnemonic(KeyEvent.VK_C);
menuMode.add(menuItem);

menuBar.add(menuMode);

JMenu menuEraser = new JMenu("Eraser");
menuEraser.setMnemonic(KeyEvent.VK_E);
JCheckBoxMenuItem cbMenuItem;

cbMenuItem = new JCheckBoxMenuItem("Erase");
cbMenuItem.setActionCommand("ToggleErase");
cbMenuItem.addActionListener(this);
cbMenuItem.setMnemonic(KeyEvent.VK_E);
menuEraser.add(cbMenuItem);

menuEraser.addSeparator();

group = new ButtonGroup();

rbMenuItem = new JRadioButtonMenuItem("Small");
rbMenuItem.setActionCommand("Small");
rbMenuItem.addActionListener(this);
rbMenuItem.setSelected(true);
rbMenuItem.setMnemonic(KeyEvent.VK_S);
group.add(rbMenuItem);
menuEraser.add(rbMenuItem);

rbMenuItem = new JRadioButtonMenuItem("Medium");
rbMenuItem.setActionCommand("Medium");
rbMenuItem.addActionListener(this);
rbMenuItem.setSelected(true);
rbMenuItem.setMnemonic(KeyEvent.VK_M);
group.add(rbMenuItem);
menuEraser.add(rbMenuItem);

rbMenuItem = new JRadioButtonMenuItem("Large");
rbMenuItem.setActionCommand("Large");
rbMenuItem.addActionListener(this);
rbMenuItem.setSelected(true);
rbMenuItem.setMnemonic(KeyEvent.VK_M);
group.add(rbMenuItem);
menuEraser.add(rbMenuItem);

menuBar.add(menuEraser);

menu = new JMenu("Help");
menu.setMnemonic(KeyEvent.VK_H);
```

```
		menuItem = new JMenuItem("About...");
		menuItem.setActionCommand("About");
		menuItem.addActionListener(this);
		menuItem.setMnemonic(KeyEvent.VK_A);
		menu.add(menuItem);

		menuBar.add(menu);

		mainFrame.setJMenuBar(menuBar);
	}

	public synchronized void addShape(Shape shape) {
		shapes.add(shape);
	}

	@Override
	public synchronized void paintComponent(Graphics gfx) {
		for (Shape shape : shapes) {
			shape.draw(gfx);
		}
	}

	@Override
	public void actionPerformed(ActionEvent e)
	{
		String action = e.getActionCommand();
		if (action.equals("NewBoard")) {
			newBoard();
		} else if (action.equals("OpenFile")) {
			loadBoard();
		} else if (action.equals("SaveAs")) {
			saveBoard();
		} else if (action.equals("Line")) {
			currentShape = LINE;
		} else if (action.equals("Rectangle")) {
			currentShape = RECT;
		} else if (action.equals("Oval")) {
			currentShape = OVAL;
		} else if (action.equals("Free Hand")) {
			currentShape = FREE;
		} else if (action.equals("Text")) {
			currentShape = TEXT;
		} else if (action.equals("Colors")) {
			currentColor = JColorChooser.showDialog(this, "Fill Color...",
getForeground());
		} else if (action.equals("Unfilled")) {
			currentMode = UNFILLED;
		} else if (action.equals("Filled")) {
			currentMode = FILLED;
		} else if (action.equals("ToggleErase")) {
			erasing = ! erasing;
		} else if (action.equals("Small")) {
			eraserSize = 1;
		} else if (action.equals("Medium")) {
			eraserSize = 2;
		} else if (action.equals("Large")) {
			eraserSize = 3;
		} else if (action.equals("About")) {
			aboutApp();
```

```
		}
	}

	public void newBoard()
	{
		BoardEvent event = new BoardEvent("loadBoard");
		event.shapes = new ArrayList<Shape>();
		try {
			boardServer.addBoardEvent(event);
		} catch (RemoteException e1) {
			e1.printStackTrace();
		}
	}

	@SuppressWarnings("unchecked")
	public void loadBoard()
	{
		JFileChooser chooser = new JFileChooser();
		FileFilter[] filefilters = chooser.getChoosableFileFilters();
		for(FileFilter filter : filefilters) {
			chooser.removeChoosableFileFilter(filter);
		}
		FileNameExtensionFilter filter = new FileNameExtensionFilter(
			"Board Files (*.brd)", "brd");
		chooser.setFileFilter(filter);

		int returnVal = chooser.showOpenDialog(mainFrame);
		if (returnVal == JFileChooser.APPROVE_OPTION) {
			try {
				ObjectInputStream is = new ObjectInputStream(new
FileInputStream(
						chooser.getSelectedFile()));
				Object board = is.readObject();
				if (board instanceof ArrayList<?>) {
					BoardEvent event = new BoardEvent("loadBoard");
					event.shapes = (ArrayList<Shape>) board;
					try {
						boardServer.addBoardEvent(event);
					} catch (RemoteException e1) {
						e1.printStackTrace();
					}
				} else {
					JOptionPane.showMessageDialog(mainFrame,
"Corrupted file contents",
							"Corrupted File",
JOptionPane.INFORMATION_MESSAGE);
				}
				is.close();
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
	}

	public void saveBoard()
	{
		JFileChooser chooser = new JFileChooser();
		FileFilter[] filefilters = chooser.getChoosableFileFilters();
		for(FileFilter filter : filefilters) {
```

```
                        chooser.removeChoosableFileFilter(filter);
                }
                FileNameExtensionFilter filter = new FileNameExtensionFilter(
                                "Board Files (*.brd)", "brd");
                chooser.setFileFilter(filter);

                int returnVal = chooser.showSaveDialog(mainFrame);
                if (returnVal == JFileChooser.APPROVE_OPTION) {
                        try {
                                ObjectOutputStream os = new ObjectOutputStream(new
FileOutputStream(
                                                chooser.getSelectedFile()));
                                os.writeObject(shapes);
                                os.close();
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                }
        }

        @Override
        public void keyTyped(KeyEvent e)
        {
                keyTyping(e);
        }

        public void keyTyping(KeyEvent e)
        {
                if(currentShape == TEXT) {
                        if(textInput == null) {
                                textInput = new ArrayList<String>();
                                textInput.add(new String());
                        }

                        char c = e.getKeyChar();
                        if(c == '\n') {
                                textInput.add(new String());
                        } else {
                                int lastLine = textInput.size()-1;
                                String lastStr = textInput.get(lastLine);
                                textInput.set(lastLine, lastStr + c);
                        }
                        //send to server
                        BoardEvent event = new BoardEvent("keyTyped");
                        event.startPoint = startPoint;
                        event.textInput = textInput;
                        try {
                                boardServer.addBoardEvent(event);
                        } catch (RemoteException e1) {
                                e1.printStackTrace();
                        }
                }
        }

        @Override
        public void keyPressed(KeyEvent e) {
        }

        @Override
```

```
	public void keyReleased(KeyEvent e) {
	}

	public void aboutApp()
	{
		JOptionPane.showMessageDialog(mainFrame, "Distributed Board\nBy:
Walid Moustafa\nMIT\nMelbourne University",
			"About Shared Board", JOptionPane.INFORMATION_MESSAGE);
	}

	class MouseAdapter extends MouseInputAdapter
	{

		@Override
		public void mouseMoved(MouseEvent e)
		{
			if((textInput != null) && ( ! textInput.isEmpty())) {
				//send to server
				BoardEvent event = new BoardEvent("mouseMoved");
				event.startPoint = startPoint;
				event.textInput = textInput;
				try {
					boardServer.addBoardEvent(event);
				} catch (RemoteException e1) {
					e1.printStackTrace();
				}
				//
				int spacing =
getGraphics().getFontMetrics().getHeight();
				startPoint = new Point(startPoint.x, startPoint.y +
spacing);
				textInput = null;
			}
		}

		@Override
		public void mousePressed(MouseEvent e)
		{
			startPoint = e.getPoint();
			if(erasing || (currentShape == FREE)) {
				points = new ArrayList<Point>();
			}
		}

		@Override
		public void mouseDragged(MouseEvent e)
		{
			Point endPoint = e.getPoint();

			if(erasing || (currentShape == FREE)) {
				points.add(new Point(e.getX(), e.getY()));
			}

			//send to server
			BoardEvent event = new BoardEvent("mouseDragged");
			event.currentShape = currentShape;
			event.currentMode = currentMode;
			event.currentColor = currentColor;
			event.erasing = erasing;
```

```
                        event.eraserSize = eraserSize;
                        event.startPoint = startPoint;
                        event.endPoint = endPoint;
                        event.points = points;
                        try {
                                boardServer.addBoardEvent(event);
                        } catch (RemoteException e1) {
                                e1.printStackTrace();
                        }
                }

                @Override
                public void mouseReleased (MouseEvent e)
                {
                        Point endPoint = e.getPoint();

                        //send to server
                        BoardEvent event = new BoardEvent("mouseReleased");
                        event.currentShape = currentShape;
                        event.currentMode = currentMode;
                        event.currentColor = currentColor;
                        event.erasing = erasing;
                        event.eraserSize = eraserSize;
                        event.startPoint = startPoint;
                        event.endPoint = endPoint;
                        event.points = points;
                        try {
                                boardServer.addBoardEvent(event);
                        } catch (RemoteException e1) {
                                e1.printStackTrace();
                        }
                }
        }

        private static final long serialVersionUID = 1L;
}
```

Text.java

```
/*
 * Name: Walid Moustafa
 * Student ID: 563080
 * Subject: COMP90015 - Distributed Systems
 * Assignment: Assignment 2 - Distributed Whiteboard
 * Project: WhiteBoard
 * File: Text.java
*/

import java.awt.Graphics;
import java.awt.Point;
import java.io.Serializable;
import java.util.ArrayList;

public class Text implements Shape, Serializable
{

        Point startPoint;
        ArrayList<String> lines = new ArrayList<String>();
```

```
	public Text (Point sPoint, ArrayList<String> input)
	{
		startPoint = sPoint;
		lines = input;
	}

	@Override
	public void draw(Graphics gfx)
	{
		int spacing = gfx.getFontMetrics().getHeight();
		int n=0;
		for (String line : lines) {
			gfx.drawString(line, startPoint.x, startPoint.y +
(n*spacing));
			n++;
		}
	}

	public static void draw (Graphics gfx, Point sPoint, ArrayList<String>
lines) {
		int spacing = gfx.getFontMetrics().getHeight();
		int n=0;
		for (String line : lines) {
			gfx.drawString(line, sPoint.x, sPoint.y + (n*spacing));
			n++;
		}
	}

	private static final long serialVersionUID = 1L;
}
```

UserListRenderer.java

```
/*
 * Name: Walid Moustafa
 * Student ID: 563080
 * Subject: COMP90015 - Distributed Systems
 * Assignment: Assignment 2 - Distributed Whiteboard
 * Project: WhiteBoard
 * File: UserListRenderer.java
*/

import java.awt.Color;
import java.awt.Component;

import javax.swing.DefaultListCellRenderer;
import javax.swing.JList;

public class UserListRenderer extends DefaultListCellRenderer {

	@Override
	public Component getListCellRendererComponent(JList<?> list,
			Object value, int index, boolean isSelected, boolean
cellHasFocus) {

		String currentUser = value.toString();
		setText(currentUser);
```

```
        Color background;
        Color foreground;

        if (currentUser.equals(WhiteBoard.userID)) {
             background = Color.WHITE;
            foreground = Color.BLUE;
       } else if (currentUser.equals(WhiteBoard.adminID)) {
             background = Color.WHITE;
            foreground = Color.RED;
       } else {
             if(WhiteBoard.isAdmin) {
             if (isSelected) {
                    background = Color.RED;
                    foreground = Color.WHITE;
                    } else {
                           background = Color.WHITE;
                   foreground = Color.BLACK;
                    }
            } else {
             background = Color.WHITE;
                foreground = Color.BLACK;
            }
       }

        setBackground(background);
        setForeground(foreground);

        return this;

       }

       private static final long serialVersionUID = 1L;
}
```

UserPanel.java

```
/*
 * Name: Walid Moustafa
 * Student ID: 563080
 * Subject: COMP90015 - Distributed Systems
 * Assignment: Assignment 2 - Distributed Whiteboard
 * Project: WhiteBoard
 * File: UserPanel.java
*/

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.rmi.RemoteException;
import java.util.Vector;

import javax.swing.*;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;

public class UserPanel extends JPanel implements ActionListener,
ListSelectionListener, KeyListener {
```

```
BoardServer boardServer;
String userID;
JScrollPane scrollPane;
JList<String> usersList;
JButton bounceButton;
Vector<String>      userIDs = new Vector<String>();
String selectedUser;


public UserPanel(BoardServer bServer, String uID, JScrollPane sPane)
{
       boardServer = bServer;
       userID = uID;
       scrollPane = sPane;
       setBorder(BorderFactory.createTitledBorder("Board Users"));
       setLayout(new BorderLayout());

  if(WhiteBoard.isAdmin) {
       bounceButton = new JButton("Bounce");
      add(bounceButton, BorderLayout.SOUTH);
      bounceButton.addActionListener(this);
  }

  usersList = new JList<String>(userIDs);
  UserListRenderer renderer = new UserListRenderer();
  usersList.setCellRenderer(renderer);
       add(usersList, BorderLayout.CENTER);
       usersList.addListSelectionListener(this);
       addKeyListener(this);
}

public synchronized void refresh(Vector<String> uIDs) {
       userIDs.removeAllElements();
       for(String auser : uIDs) {
              if(auser.charAt(0) != '#') {
                     userIDs.add(auser);
              }
       }
       usersList.setListData(userIDs);
       if(userIDs.size() >=2) {
              usersList.setSelectedIndex(1);
       } else {
              usersList.setSelectedIndex(0);
       }
       scrollPane.revalidate();
       scrollPane.repaint();
}

@Override
public void valueChanged(ListSelectionEvent event) {
       if(event.getSource() == usersList && !event.getValueIsAdjusting()) {
              String stringValue = (String)usersList.getSelectedValue();
              if(stringValue != null) {
                     selectedUser = stringValue;
                     if(bounceButton != null) {

if(selectedUser.equalsIgnoreCase(WhiteBoard.adminID)) {
                                   bounceButton.setEnabled(false);
```

```
                                } else {
                                        bounceButton.setEnabled(true);
                                }
                        }
                }
        }

        @Override
        public void actionPerformed(ActionEvent event) {
                if(event.getSource() == bounceButton) {
                        int selection = usersList.getSelectedIndex();
                        if(selection >= 0) {
                                try {
                                        boardServer.bounceUser(selectedUser);
                                } catch (RemoteException e) {
                                        e.printStackTrace();
                                }
                        }
                }
        }

        @Override
        public void keyTyped(KeyEvent e) {
                WhiteBoard.sharedPanel.keyTyping(e);
        }

        @Override
        public void keyPressed(KeyEvent e) {
        }

        @Override
        public void keyReleased(KeyEvent e) {
        }

        private static final long serialVersionUID = 1L;

}
```

WhiteBoard.java

```
/*

 * Name: Walid Moustafa

 * Student ID: 563080

 * Subject: COMP90015 - Distributed Systems

 * Assignment: Assignment 2 - Distributed Whiteboard

 * Project: WhiteBoard

 * File: WhiteBoard.java

*/


import java.awt.BorderLayout;
```

```
import java.awt.Dimension;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import javax.swing.*;

public class WhiteBoard
{
       static String serverName = "localhost";
       static String serviceName = "BoardServer";
       static BoardServer boardServer;
       static String userID;
       static String adminID;
       static boolean isAdmin;
       static UserPanel userPanel;
       static SharedPanel sharedPanel;

       private static void initialize()
    {

         //Create and set up the window.
         final JFrame frame = new JFrame("White Board");
         frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
         frame.addWindowListener(new WindowAdapter() {
               public void windowClosing(WindowEvent we) {
                     if(!isAdmin) {
                           int choice = JOptionPane.showConfirmDialog(frame,
                                       "Are you sure you want to exit the
application?",
```

```
                                        "Exit                          Confirmation",
JOptionPane.YES_NO_OPTION);
                            if(choice == JOptionPane.YES_OPTION) {
                                  try {
                                        boardServer.bounceUser(WhiteBoard.userID);
                                  } catch (RemoteException e) {
                                        e.printStackTrace();
                                  }
                                  System.exit(0);
                            } else {
                                  frame.setVisible(true);
                            }
                      } else {
                            int choice = JOptionPane.showConfirmDialog(frame,
                                        "Are you sure you want to end the current
Whiteboard Session?",
                                        "Exit                          Confirmation",
JOptionPane.YES_NO_OPTION);
                            if(choice == JOptionPane.YES_OPTION) {
                                  if(boardServer != null) {
                                        BoardEvent        event        =        new
BoardEvent("Terminate");
                                        try {

      boardServer.addBoardEvent(event);
                                              } catch (RemoteException e) {
                                                    e.printStackTrace();
                                              }
                                  }
                                  //System.exit(0);
                            } else {
                                  frame.setVisible(true);
                            }
                      }
```

```
            }
        });

        frame.getContentPane().setLayout(new BorderLayout());

        sharedPanel = new SharedPanel(boardServer, userID, frame);
        frame.getContentPane().add(sharedPanel, BorderLayout.CENTER);

        JScrollPane scrollPane = new JScrollPane();
        userPanel = new UserPanel(boardServer, userID, scrollPane);
        scrollPane.setPreferredSize(new Dimension(110,700));
             scrollPane.getViewport().add(userPanel);
             frame.getContentPane().add(scrollPane, BorderLayout.EAST);

             //frame.pack();
        frame.setSize(1100, 700);
        frame.setResizable(false);
        frame.setLocationRelativeTo(null);
        frame.setVisible(false);

        frame.addWindowFocusListener(new WindowAdapter() {
            public void windowGainedFocus(WindowEvent e) {
                sharedPanel.requestFocusInWindow();
            }
        });

        // start EventDispatcher thread
       new                  Thread(new                  EventDispatcher(sharedPanel,
userPanel),"EventDispatcher").start();

    }

    public static void main(String[] args) {
```

```
        if (args.length != 2) {
              System.err.println("usage: WhiteBoard ServerName ServiceName");
              System.exit(0);
          }

        serverName  = args[0];
        serviceName = args[1];

        try {
               /*if(System.getSecurityManager()==null) {
                            System.setSecurityManager(new RMISecurityManager());
                      }*/
              boardServer                    =                    (BoardServer)
Naming.lookup("rmi://"+serverName+"/"+serviceName);
              String candidateID = null;
              while  ((candidateID  ==  null)  ||  (candidateID.length()  <  3)  ||
(candidateID.length() > 8)) {
                      candidateID  =  JOptionPane.showInputDialog(null,  "Enter  3-8
letters user name","Login",JOptionPane.INFORMATION_MESSAGE);
              }
              userID = boardServer.joinBoard(candidateID);
              adminID = boardServer.getAdmin();
              isAdmin = userID.equalsIgnoreCase(adminID) ? true : false;
              System.out.println("Assigned userID " + userID);
              } catch (MalformedURLException e) {
                      e.printStackTrace();
                      System.exit(-1);
              } catch (RemoteException e) {
                      e.printStackTrace();
                      System.exit(-1);
              } catch (NotBoundException e) {
                      e.printStackTrace();
```

```
                System.exit(-1);
            }

        //Schedule a job for the event-dispatching thread:
          //creating and showing this application's GUI.
          javax.swing.SwingUtilities.invokeLater(new Runnable() {
              public void run()
              {
                  initialize();
              }
          });

    }
}
```