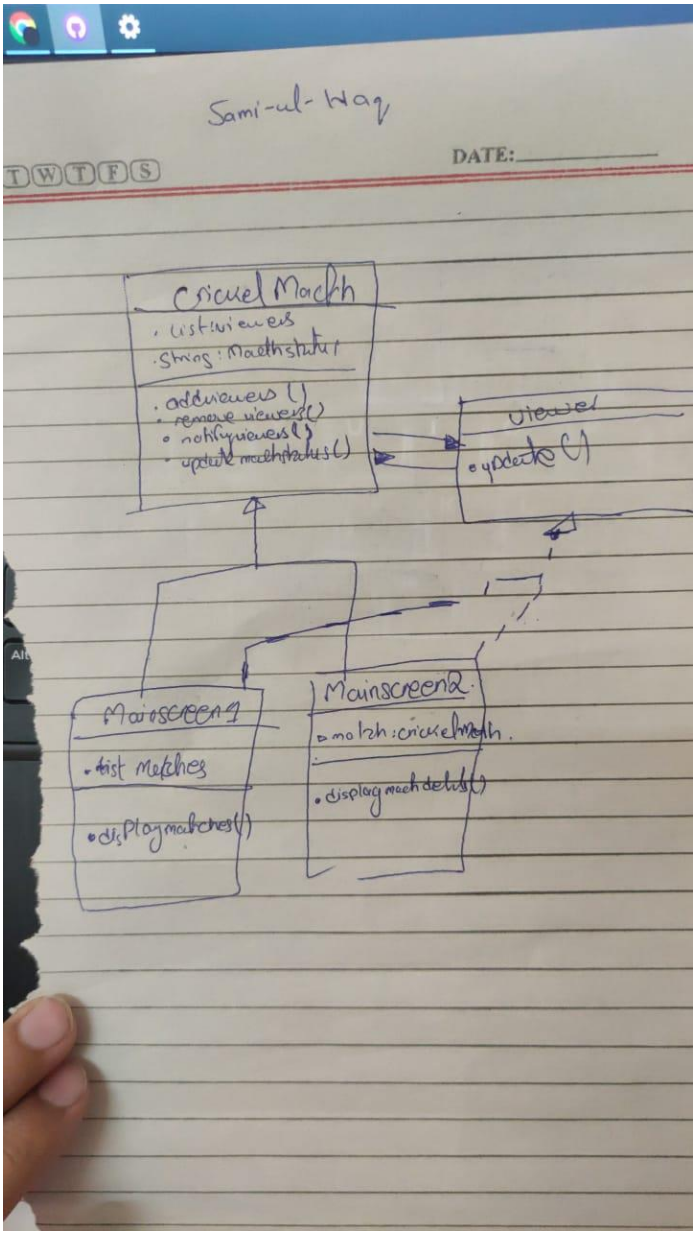
Design pattern lab term

Class diagram



Code

```
import java.util.ArrayList;

import java.util.List;


// Subject interface
```

```
abstract class CricketMatch {
    public abstract void addViewer(MatchViewer viewer);
    public abstract void removeViewer(MatchViewer viewer);
    public abstract void notifyViewers();
    public abstract void updateMatchStatus(String status);
}

// Concrete subject
class LiveCricketMatch extends CricketMatch {
    private List<MatchViewer> viewers;
    private String matchStatus;

    public LiveCricketMatch() {
        viewers = new ArrayList<>();
    }

    @Override
    public void addViewer(MatchViewer viewer) {
        viewers.add(viewer);
    }

    @Override
    public void removeViewer(MatchViewer viewer) {
        viewers.remove(viewer);
    }

    @Override
    public void notifyViewers() {
        for (MatchViewer viewer : viewers) {
```

```
            viewer.update(matchStatus);
        }
    }

    @Override
    public void updateMatchStatus(String status) {
        this.matchStatus = status;
        notifyViewers();
    }

    // Simulating live match updates
    public void simulateMatch() {
        // Simulated ball-by-ball status
        String[] statuses = {"Score: 10/0", "Score: 20/0", "Score: 35/1", "Score: 50/1", "Score: 65/2"};

        for (String status : statuses) {
            updateMatchStatus(status);
            try {
                Thread.sleep(2000); // Simulate delay
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

// Observer interface
abstract class MatchViewer {
    public abstract void update(String matchStatus);
```

```
}

// Concrete observers
class MainScreen extends MatchViewer {
    @Override
    public void update(String matchStatus) {
        System.out.println("Main Screen: Updated match status - " + matchStatus);
    }
}

class MatchScreen extends MatchViewer {
    @Override
    public void update(String matchStatus) {
        System.out.println("Match Screen: Updated match status - " + matchStatus);
    }
}

public class ObserverPatternExample {
    public static void main(String[] args) {
        LiveCricketMatch liveMatch = new LiveCricketMatch();

        MainScreen mainScreen = new MainScreen();
        MatchScreen matchScreen = new MatchScreen();

        liveMatch.addViewer(mainScreen);
        liveMatch.addViewer(matchScreen);
        // Starting match simulation after registering observers
        liveMatch.simulateMatch();
    }
```

}