Observer pattern

Consider the following application:

```
public class Application {

      public static void main(String[] args) {
            StockMarket market = new StockMarket();
            StockBuyer buyer = new StockBuyer();
            StockViewer viewer = new StockViewer();
            market.addStock("ORC", 12.23);
            market.addStock("MSC", 45.78);

            market.update("ORC", 12.34);
            market.update("MSC", 44.68);
      }
}


public abstract class AbstractMarket {
      Map<String,Double> stocklist = new HashMap<String,Double>();

      public Map<String, Double> getStocklist() {
            return stocklist;
      }
}


public class StockMarket extends AbstractMarket{
      private StockBuyer buyer = new StockBuyer();
      private StockViewer viewer = new StockViewer();

      public void addStock(String stockSymbol, Double price){
            stocklist.put(stockSymbol, price);
      }

      public void update(String stockSymbol, Double price){
            stocklist.put(stockSymbol, price);
            buyer.update(stocklist);
            viewer.update(stocklist);
      }
}


public class StockBuyer {
      private AbstractMarket stockMarket;

      public void update(Map<String, Double> stocklist) {
            System.out.println("StockBuyer: stocklist is changed:");
            Iterator iter = stocklist.entrySet().iterator();
            while (iter.hasNext()) {
                  Map.Entry entry = (Map.Entry) iter.next();
                  String key = (String) entry.getKey();
                  Double value = (Double) entry.getValue();
                  System.out.println(key + " - $" + value);
            }
      }
}
```

```
public class StockViewer {

      private AbstractMarket stockMarket;

      public void update(Map<String, Double> stocklist) {
            System.out.println("StockViewer: stocklist is changed:");
            Iterator iter = stocklist.entrySet().iterator();
            while (iter.hasNext()) {
                  Map.Entry entry = (Map.Entry) iter.next();
                  String key = (String) entry.getKey();
                  Double value = (Double) entry.getValue();
                  System.out.println("StockViewer" + key + " - $" + value);
            }
      }
}
```

The problem with this application is that the StockMarket knows all Objects (StockBuyer and StockViewer) that are interested in its state.
Redesign this application with the Observer pattern applied such that the StockMarket is independent of the StockBuyer and StockViewer.

Draw the class diagram with the Observer pattern. Draw a sequence diagram that shows how the Observer pattern works.