Observer Pattern

מתי אנחנו משתמשים ב Observer?

כשאנחנו צריכים שאובייקטים רבים יקבלו עדכון כאשר מידע משתנה.

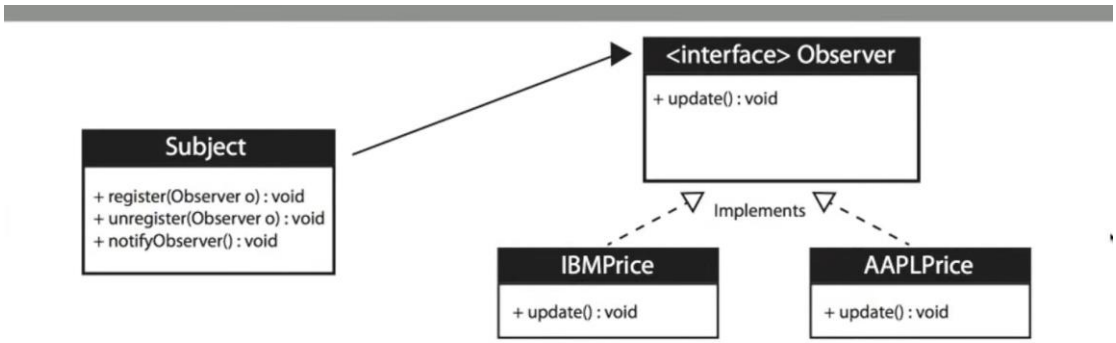
לדוגמא יהיו לנו כמה משקיעים שירצו לקבל מידע על מניות שונות נרצה שבכל פעם שמניה תעודכן כל האובייקטים יעודכנו.

אז מה היתרון של דפוס הצופה -> האזור שיאחסן את המידע(subject) לא צריך לדעת כלום על הצופים

החיסרון של observer -> הsubject עשוי לשלוח עדכונים שלא חשובים לצופים

הגדרה נפוצה לobserver היא – דפוס עיצוב תוכנה שבו אובייקט שנקרא הנושא(subject) שומר רשימה של התלויים שלו הנקראים צופים(Observer), ומודיע להם אוטומטית על כל שינוי מצב שמתרחש בנושא(subject)

דיאגרמת UML



The **Observer pattern** is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods.

Subject – יש לו 3 מתודות:

- register(Observer o) – היכולת לרשום צופים חדשים – אנחנו נשמור את רשימת הצופים במערך
- unRegister(Observer o) – הצופים לא יקבל יותר עדכונים מהנושא
- notifyObserver – היכולת להודיע לצופה כשהתבצעו שינויים

Observer – ממשק לצופה יהיה בו את המתודה update שיעדכן את המידע שבתוך המחלקות שיממשו אותו

ניצור שתי צופים שיממשו את הממשק

Subject.java

ממשק שיטפל בהוספת מחיקת ועדכון של כל הצופים

```
// This interface handles adding, deleting and updating
// all observers

public interface Subject {

    public void register(Observer o);
    public void unregister(Observer o);
    public void notifyObserver();

}
```

Observer.java

ממשק שיטפל בעדכונים של הצופים

```
// The Observers update method is called when the Subject changes

public interface Observer {

    public void update(double ibmPrice, double aaplPrice, double googPrice);

}
```

StockGrabber.java

מחלקה שתירש מsubject

כל הצופים של יהיו ברשימה

ויהיו לי את כל השדות של מחירי המניות שעליהן אני ארצה לקבל עדכונים

בבנאי נאתחל את הרשימה

```
// Uses the Subject interface to update all Observers

public class StockGrabber implements Subject{

    private ArrayList<Observer> observers;
    private double ibmPrice;
    private double aaplPrice;
    private double googPrice;

    public StockGrabber(){

        // Creates an ArrayList to hold all observers

        observers = new ArrayList<Observer>();
    }

    public void register(Observer newObserver) {

        // Adds a new observer to the ArrayList

        observers.add(newObserver);

    }
```

```java
    public void unregister(Observer deleteObserver) {

        // Get the index of the observer to delete

        int observerIndex = observers.indexOf(deleteObserver);

        // Print out message (Have to increment index to match)

        System.out.println("Observer " + (observerIndex+1) + " deleted");

        // Removes observer from the ArrayList

        observers.remove(observerIndex);

    }

    public void notifyObserver() {

        // Cycle through all observers and notifies them of
        // price changes

        for(Observer observer : observers){

            observer.update(ibmPrice, aaplPrice, googPrice);

        }
    }

    // Change prices for all stocks and notifies observers of changes

    public void setIBMPrice(double newIBMPrice){

        this.ibmPrice = newIBMPrice;

        notifyObserver();

    }

    public void setAAPLPrice(double newAAPLPrice){

        this.aaplPrice = newAAPLPrice;

        notifyObserver();

    }

    public void setGOOGPrice(double newGOOGPrice){

        this.googPrice = newGOOGPrice;

        notifyObserver();

    }

}
```

StockObserver.java

מחלקה שתירש מהממשק observer

יהיה לנו את המחירים של המניות

יהיה לנו משתנה סטטי שיעזור לנו למספר את הצופים שלנו

כדי שנוכל בStockGrabber ניצור משתנה של הממשק subject

בבנאי נקבל את הsubject שלנו וניתן לovserver שלנו id

ואז נפעיל באובייקט subject שלנו את התמתודה register ונרשום את הobserver שיצרנו

```
// Represents each Observer that is monitoring changes in the subject

public class StockObserver implements Observer {

    // Static used as a counter
    private static int observerIDTracker = 0;

    // Used to track the observers
    private final int observerID;

    private double ibmPrice;
    private double aaplPrice;
    private double googPrice;


    // Will hold reference to the StockGrabber object

    private Subject stockGrabber;

    public StockObserver(Subject stockGrabber) {

        // Store the reference to the stockGrabber object so
        // I can make calls to its methods

        this.stockGrabber = stockGrabber;

        // Assign an observer ID and increment the static counter

        this.observerID = ++observerIDTracker;

        // Message notifies user of new observer

        System.out.println("New Observer " + this.observerID);

        // Add the observer to the Subjects ArrayList

        stockGrabber.register(this);

    }

    // Called to update all observers

    public void update(double ibmPrice, double aaplPrice, double googPrice) {

        this.ibmPrice = ibmPrice;
        this.aaplPrice = aaplPrice;
        this.googPrice = googPrice;

        printThePrices();

    }

    public void printThePrices() {
```

```
        System.out.println(observerID + "\nIBM: " + ibmPrice + "\nAAPL: " +
                aaplPrice + "\nGOOG: " + googPrice + "\n");

    }

}
```

GrabStocks.java

איפה המain

ניצור אובייקט subject שבו יקרו כל העדכונים ומשם ישאבו את המידע

ואז ניצור את הצופים שלנו שהם יקבלו לאן הם צריכים להסתכל

ואז בsubject שלנט נעדכן את הנתונים

```
public class GrabStocks{

    public static void main(String[] args){

        // Create the Subject object
        // It will handle updating all observers
        // as well as deleting and adding them

        StockGrabber stockGrabber = new StockGrabber();

        // Create an Observer that will be sent updates from Subject

        StockObserver observer1 = new StockObserver(stockGrabber);

        stockGrabber.setIBMPrice(197.00);
        stockGrabber.setAAPLPrice(677.60);
        stockGrabber.setGOOGPrice(676.40);

        StockObserver observer2 = new StockObserver(stockGrabber);

        stockGrabber.setIBMPrice(197.00);
        stockGrabber.setAAPLPrice(677.60);
        stockGrabber.setGOOGPrice(676.40);

        // Delete one of the observers

        // stockGrabber.unregister(observer2);

    }

}
```