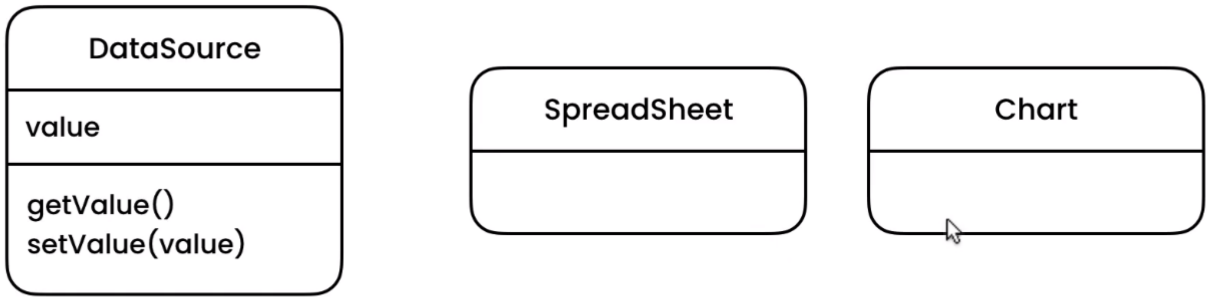
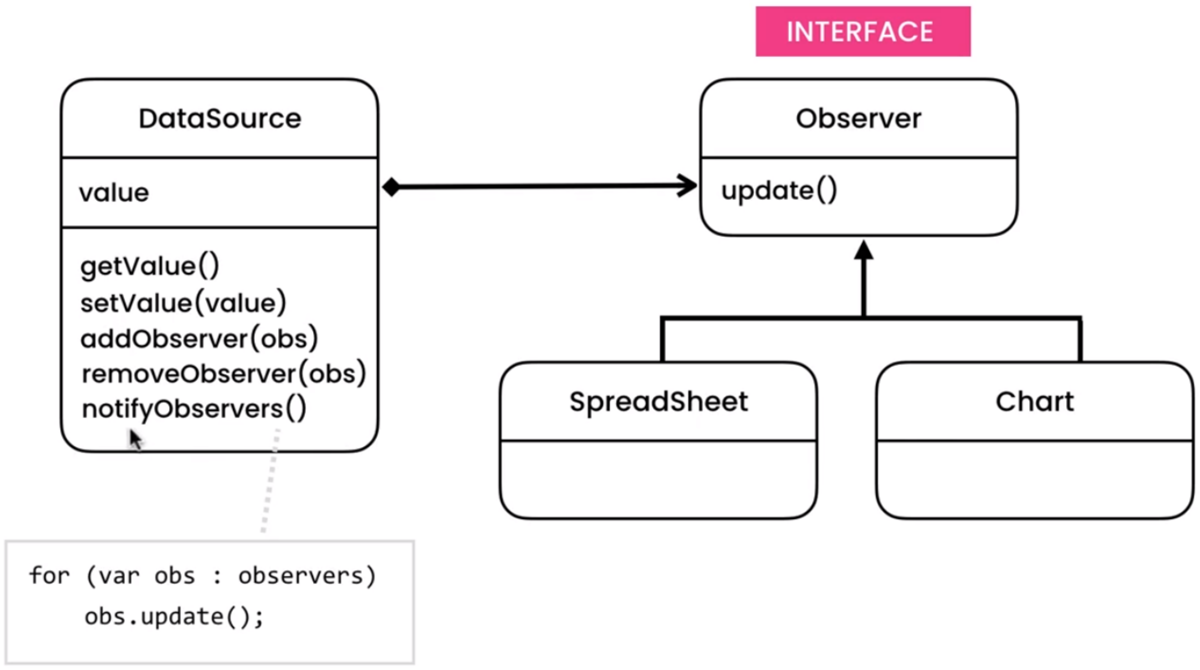
Observer Pattern

Example Problem

You have an object e.g a spreadsheet (datasource) and it's state change i.e one of the row's values changes or a new row is aded the observer pattern then notifies objects which depend on this object of the change e.g. a pie chart may be updated. the focus on the communication between objects and reduces coupling / dependecies betwen classes allowing more observer to be added without affcting the datasource :



Using the Open for Extension Closed for Modification Principle becomes (the set method in the DataSource (which extends the subject class on the observer pattern) class calls the notifyObservers method then each observer has a concrete implementation of the update method to handle the new data) :



```
for (var obs : observers)
    obs.update();
```

Gang Of Four representation of the Observer Pattern - Datasource is our ConcreteSubject and the observers are all Concrete Observers :

`Subject`	DataSource	`Observer`	`SpreadSheet`

```
import java.util.ArrayList;
import java.util.List;

// Observable
public class Subject {
  private List<Observer> observers
= new ArrayList<>();

 public void addObserver(Observer
observer) {
    observers.add(observer);
 }

  public void removeObserver
(Observer observer) {
    observers.remove(observer);
 }

  public void notifyObservers() {
    for (var observer : observers)
      observer.update();
 }
}
```

```
public class DataSource extends Subject {

  private int value;

 public int getValue() {
    return value;
 }

  public void setValue(int valu
e) {
    this.value = value;
 notifyObservers();
 }
}
```

```
public interface
Observer {
  void update();
}
```

```
public class SpreadSheet implements
Observer {
  private DataSource dataSource;

 public SpreadSheet(DataSource
dataSource) {
    this.dataSource = dataSource;
 }

  @Override
  public void update() {
    System.out.println("SpreadSheet
got notified: " + dataSource.
getValue());
 }
}
```

Chart

```
public class Chart implements Observer {
  private DataSource dataSource;

 public Chart(DataSource dataSource) {
    this.dataSource = dataSource;
 }

//below is an example of the pull style of communication in which the

//observer requests data from the datasource (dataSource.getValue()

//- coupling the observer to the datasource / concrete subject - which

//is fine as long as its not the other way around - if the concrete

//subject was couple to the concrete observer then any changes in the

//concrete observer would cascade to all of the observers)

//the push style of communnication where the subject pushes to the observers results in

//the cascading changes mentioned above and does not allow observers to flexibly

//request the data they need and has the follwoing update method signature :

//public void update(int value)


  @Override
  public void update() {
    System.out.println("Chart got updated: " + dataSource.getValue());
 }
}
```

Main

```
public class Main {

public static void main
(String[] arg) {
    var dataSource = new
DataSource();

 var sheet1 = new
SpreadSheet();

 var sheet2 = new
SpreadSheet();

 var chart = new Chart();


datasource.addObserver(sheet1);

datasource.addObserver(sheet2);

datasource.addObserver(chart);


datasource.setValues(1)

//prints

SpreadSheet got notified: 1

SpreadSheet got notified: 1

Chart got updated: 1
```