CAMPUS 02 GRAZ
FACHHOCHSCHULE DER WIRTSCHAFT

SOFTWARE DESIGN FOLIENSATZ 3

Observer Pattern

Bernhard Fuchs
ZAM - WS 2025/26

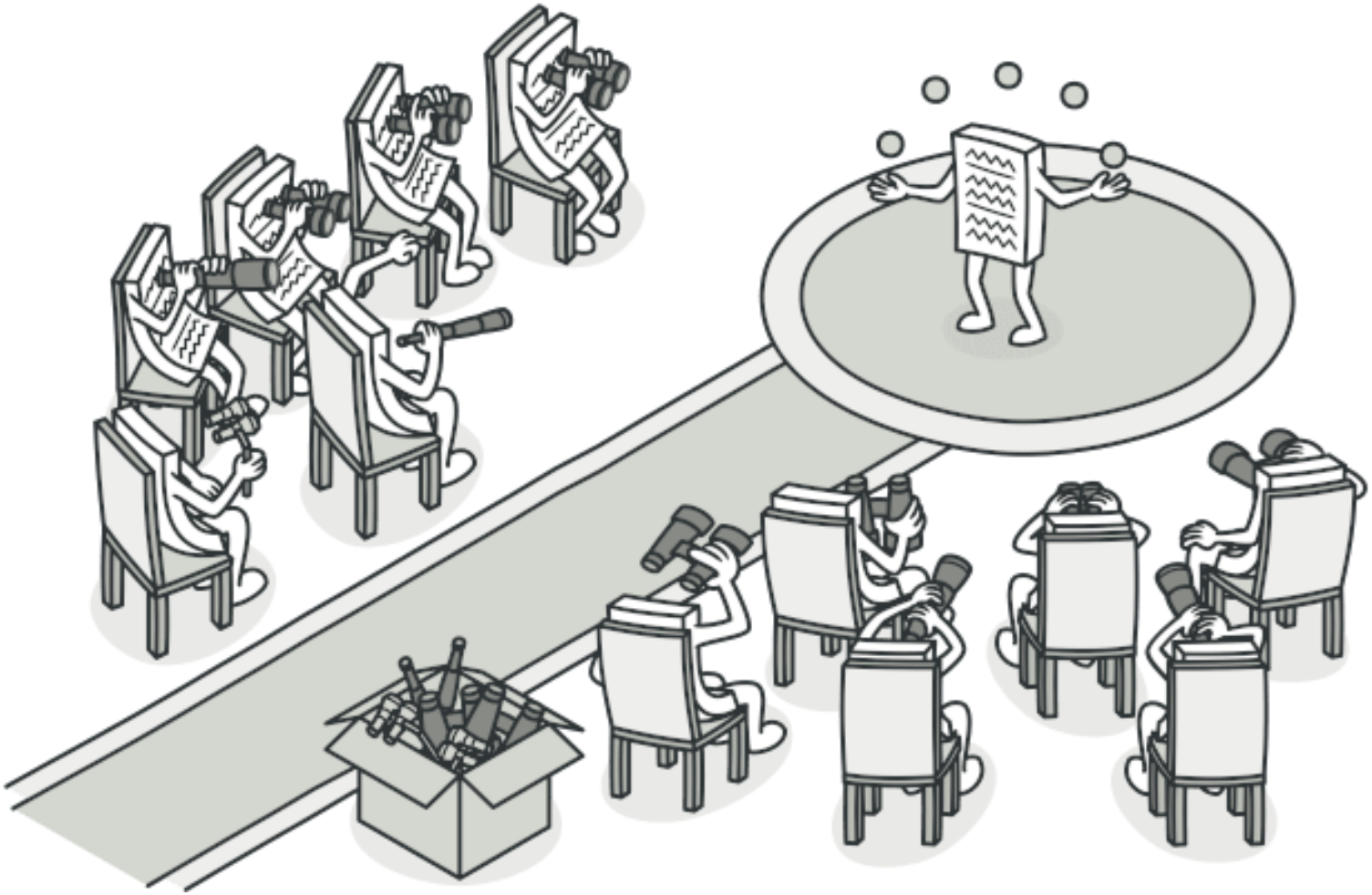
TERMINE

0004VD2005 SOFTWARE DESIGN (29,75UE IL, WS 2025/26)

Gruppe

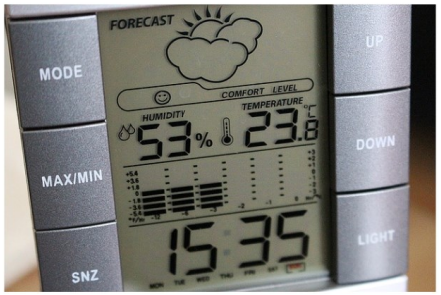
	Tag	Datum	von	bis	Ort	Ereignis	Termintyp	Lerneinheit	Vortragende*r	Anmerkung
	Standardgruppe									
	Do	22.01.2026	08:15	11:45	CZ106	Abhaltung	fix	**1 Organisation + Strategy Pattern**		
	Do	29.01.2026	08:15	11:45	CZ106	Abhaltung	fix	**2 Decorator Pattern + Singleton**		
Heute	Do	05.02.2026	12:30	16:00	CZ106	Abhaltung	fix	**3 Observer**		
	Fr	06.02.2026	08:15	12:15	CZ106	Abhaltung	fix	**4 Factory**		
	Do	12.02.2026	08:15	13:00	CZ106	Abhaltung	fix	**5 Command + Adapter**		
	Fr	13.02.2026	08:15	12:15	CZ106	Abhaltung	fix	**6 Facade, Template, Iterator** mit Christian Hofer!		
	Do	19.02.2026	08:15	11:45	CZ106	Abhaltung	fix	**Wiederholung**		
	Fr	20.02.2026	08:15	12:15	CZ106	Prüfungstermin	fix	**Prüfung**		

OBSERVER PATTERN



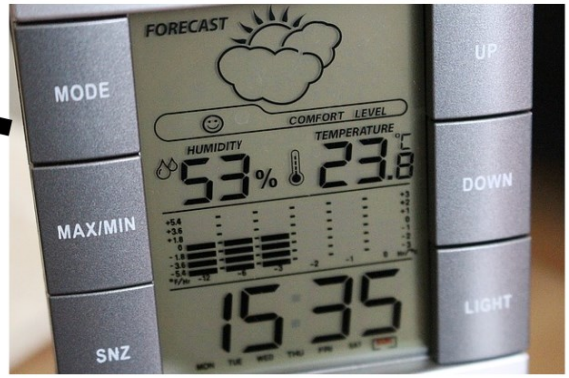
SMARTE WETTERSTATION

Wetterstation soll unterschiedliche Displays unterstützen



MÖGLICHE LÖSUNG 1

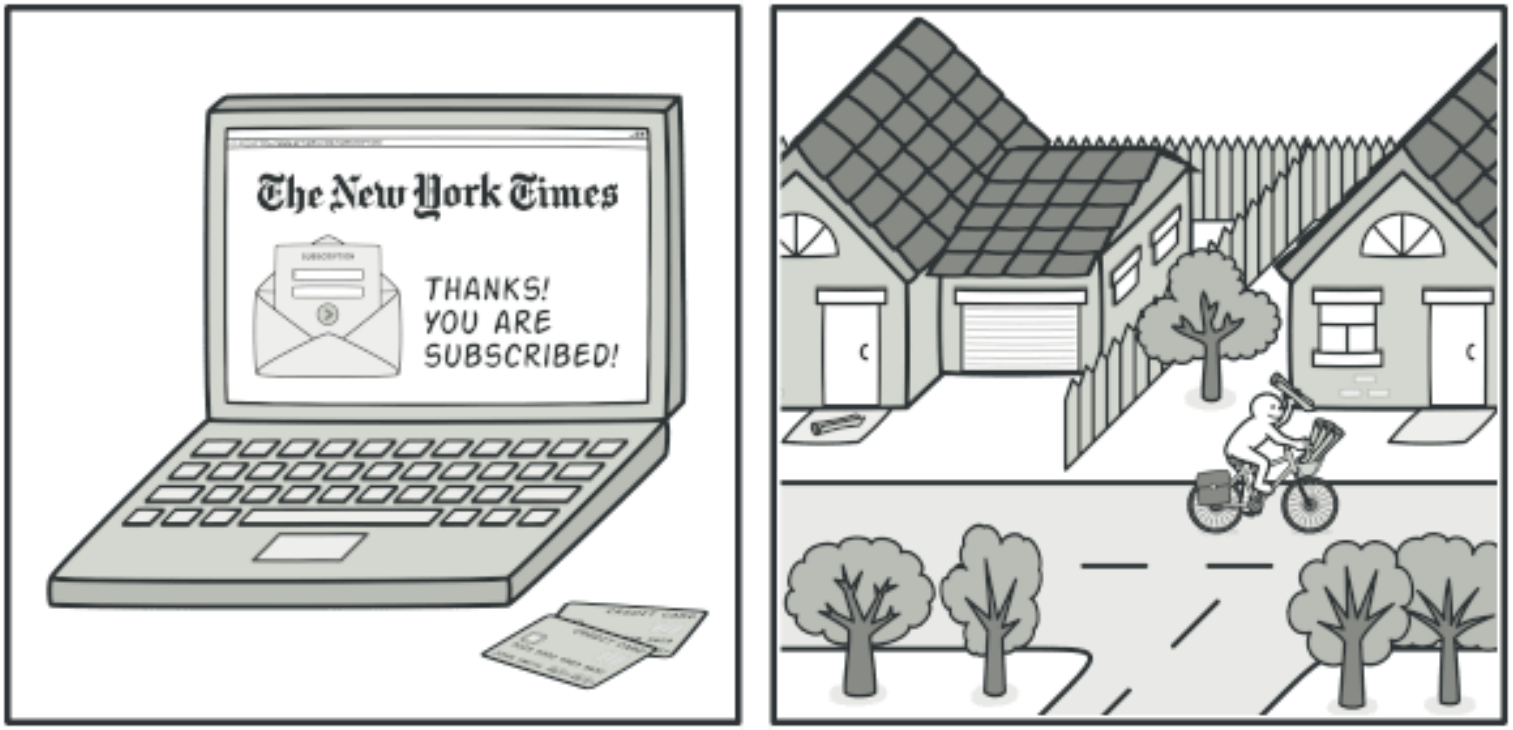
Alle 10 Minuten
getData()



MÖGLICHE LÖSUNG 2

```
public void measurementChanged() {
 double temp = getTemperature();
 double pressure = getPressure();

 digitalDisplay.update(temp, pressure);
 phoneDisplay.update(temp, pressure);
}
```

OBSERVER PATTERN



OBSERVER PATTERN

- Funktioniert wie ein Abo für Zeitung/Magazin

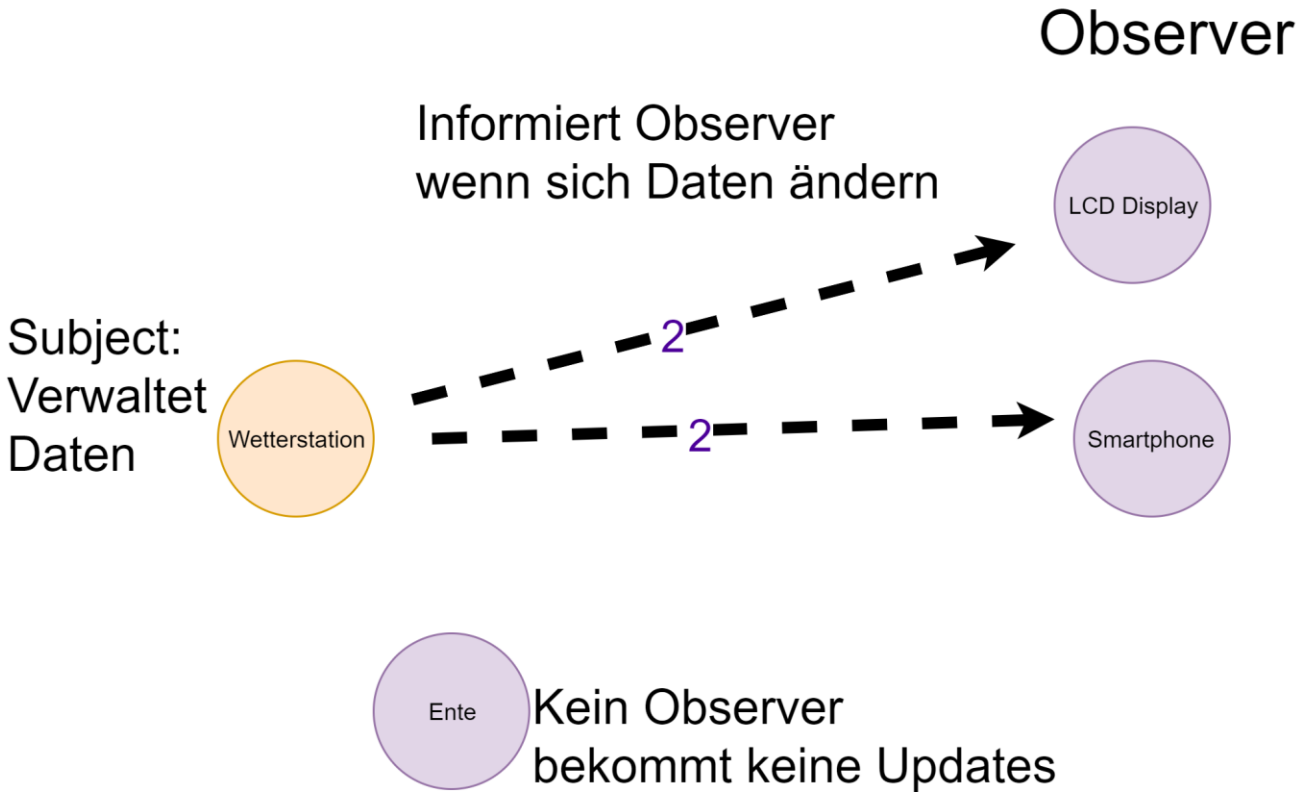
1. Verlag beginnt eine Zeitung zu veröffentlichen

2. Abonnent*innen registrieren sich bei Verlag und erhalten neue Ausgaben

3. Bestellen Abo ab wenn nicht mehr interessiert

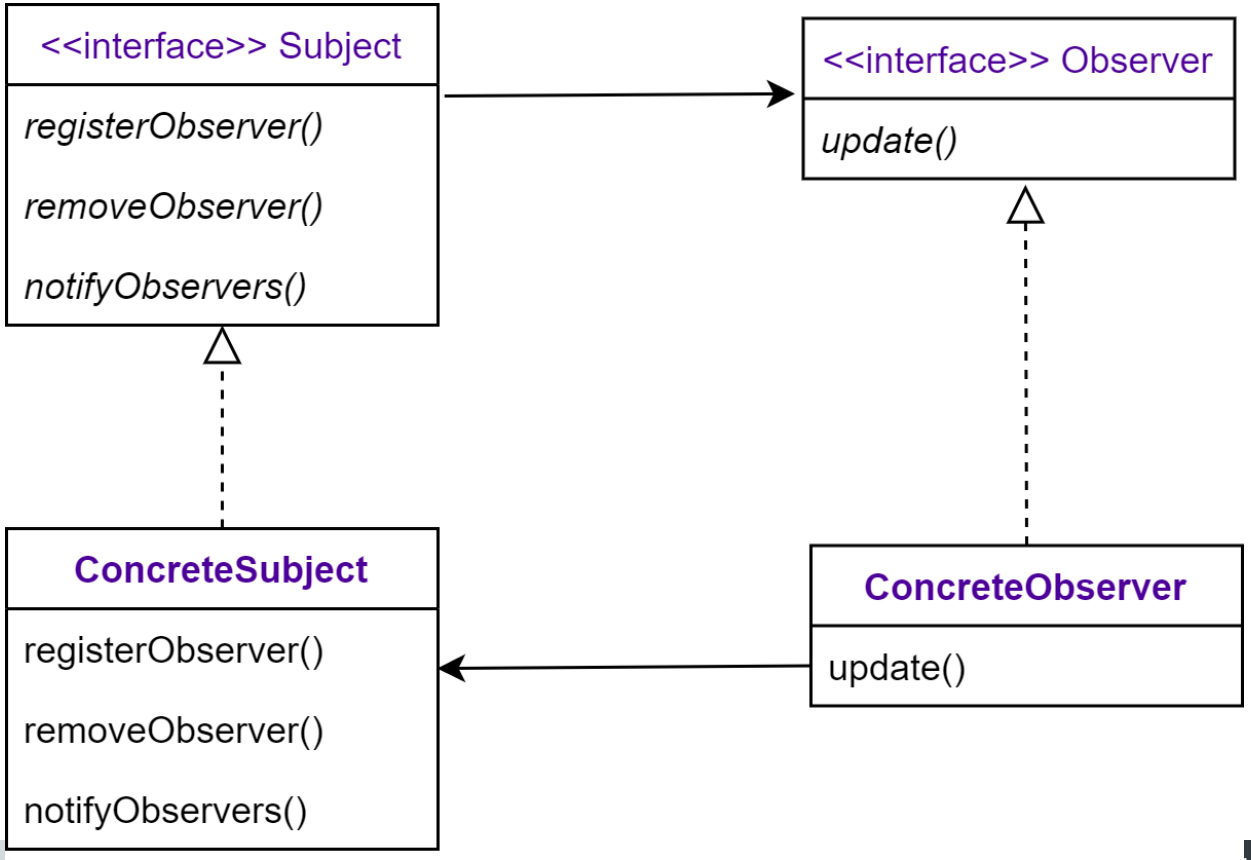
OBSERVER PATTERN

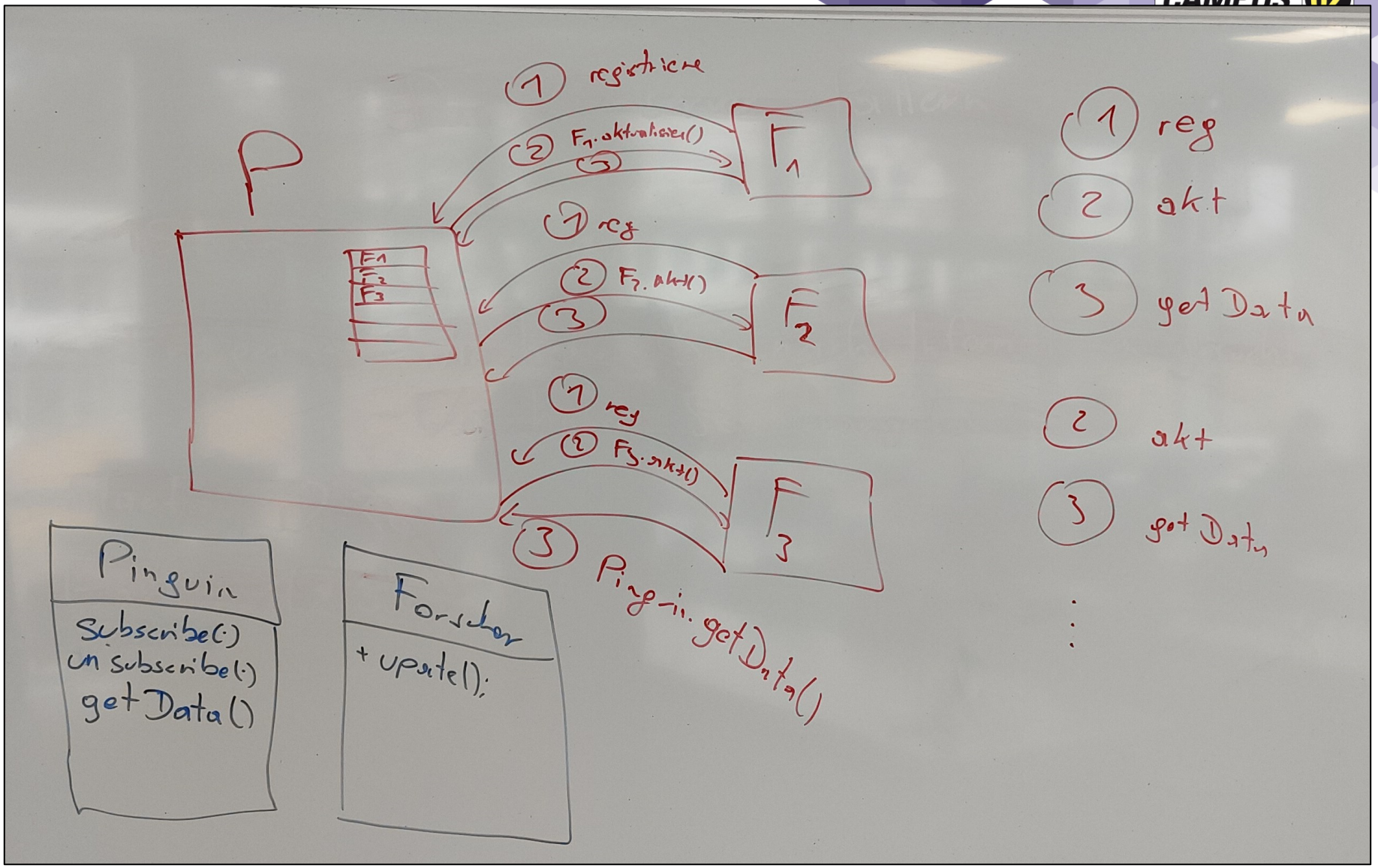
- Subject(Publisher) + Observer (Subscriber)

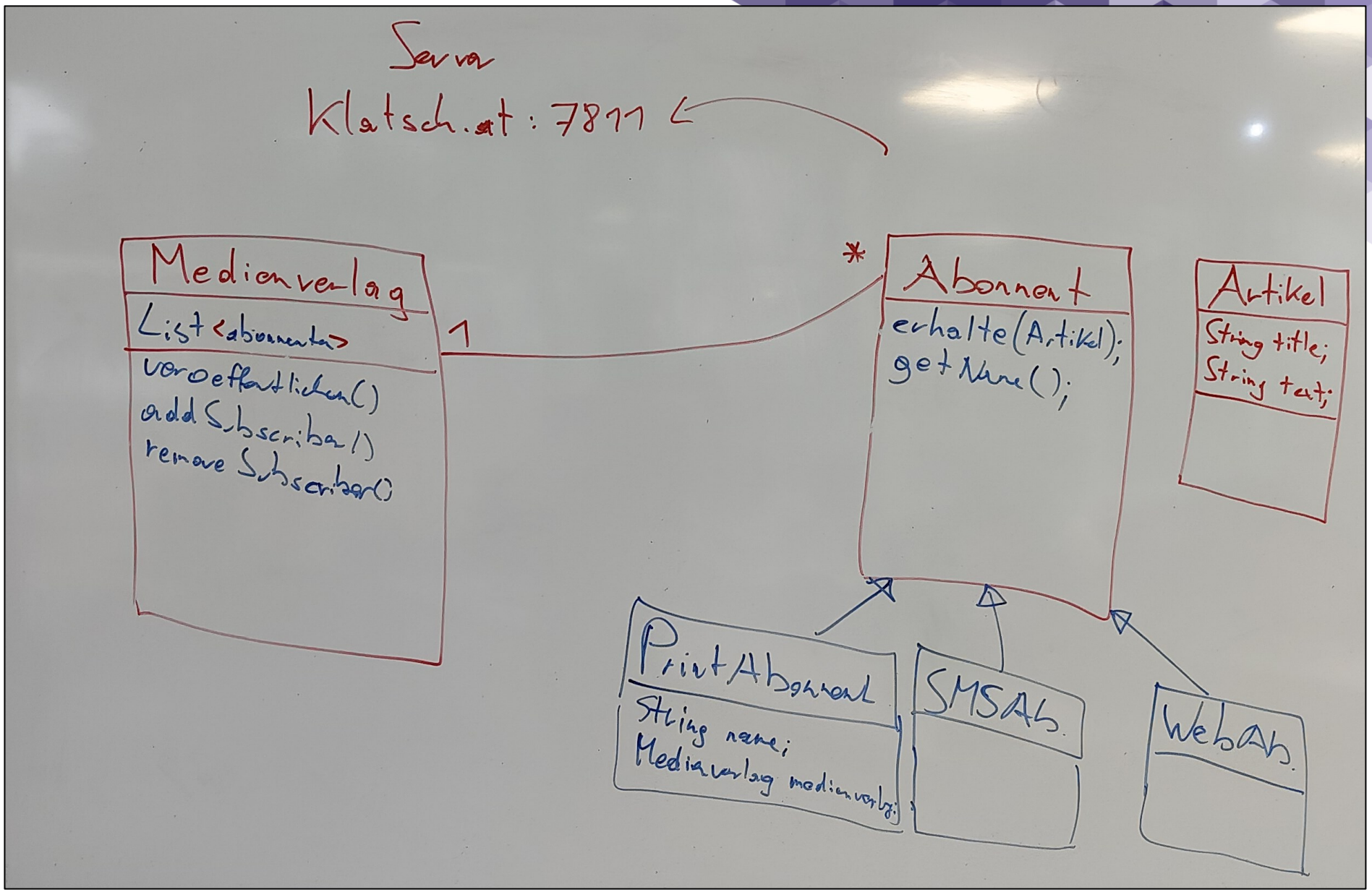


OBSERVER PATTERN

- Subject(Publisher) + Observer (Subscriber)

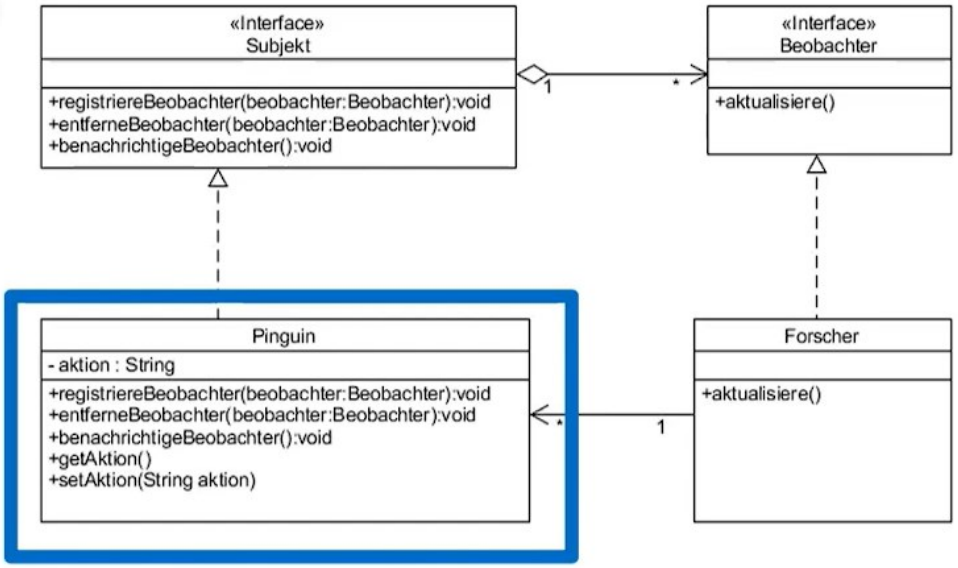



P
F1
F2
F3
1 registrieren
2 F1.aktualisieren()
3
1 reg
2 F2.akt()
3
1 reg
2 F3.akt()
3 Pinguin.getData()
Pinguin
subscribe(·)
unsubscribe(·)
getData()
Forscher
+ update();
1 reg
2 akt
3 getData
2 akt
3 getData


Server
Klatsch.at:7811
Medienverlag
List<Abonnenten>
veroeffentlichen()
addSubscriber()
removeSubscriber()
1
*
Abonnent
erhalte(Artikel);
getName();
Artikel
String title;
String text;
PrintAbonnent
String name;
Medienverlag medienverlag;
SMSAb.
WebAb.

YOUTUBE BEISPIEL 1/2

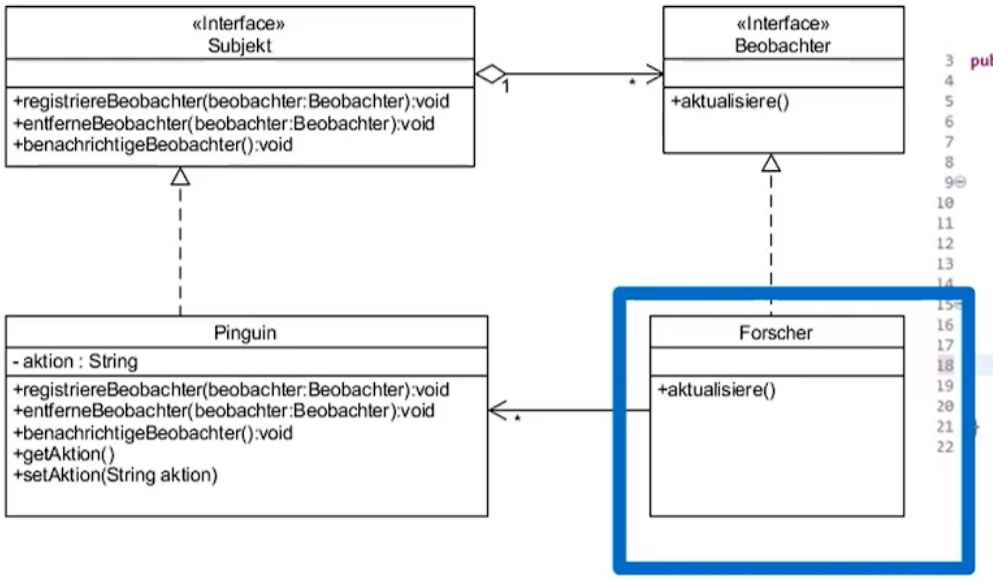
Implementierung




```
public class Pinguin implements Subjekt{

    private List<Beobachter> beobachterListe = new ArrayList<Beob

    private String aktion;

    @Override
    public void registriereBeobachter(Beobachter beobachter) {
        this.beobachterListe.add(beobachter);
    }

    @Override
    public void entferneBeobachter(Beobachter beobachter) {
        this.beobachterListe.remove(beobachter);
    }

    @Override
    public void benachrichtigeBeobachter() {
        for (Beobachter beobachter : beobachterListe) {
            beobachter.aktualsiere();
        }
    }

    public String getAktion() {
        return aktion;
    }

    public void setAktion(String aktion) {
        this.aktion = aktion;
        this.benachrichtigeBeobachter();
    }
}
```

YOUTUBE BEISPIEL 2/2

Implementierung




```
3  public class Forscher implements Beobachter {
4
5      private String name;
6      private Pinguin pinguin;
7
8
9      public Forscher(Pinguin pinguin, String name) {
10         this.pinguin = pinguin;
11         this.name = name;
12         pinguin.registriereBeobachter(this);
13     }
14
15     @Override
16     public void aktualsiere() {
17         System.out.println("Der Forscher " + name + " sieht, "
18             + "dass der Pinguin " + name + " gerade " + pinguin.getAktion());
19     }
20
21 }
22
```

OBSERVER PATTERN

- Let‘s code :)

OBSERVER PATTERN

Definiert eine eins-zu-viele Abhängigkeit zwischen Objekten, sodass bei einer Änderung eines Objekts alle abhängigen Objekte informiert und aktualisiert werden.

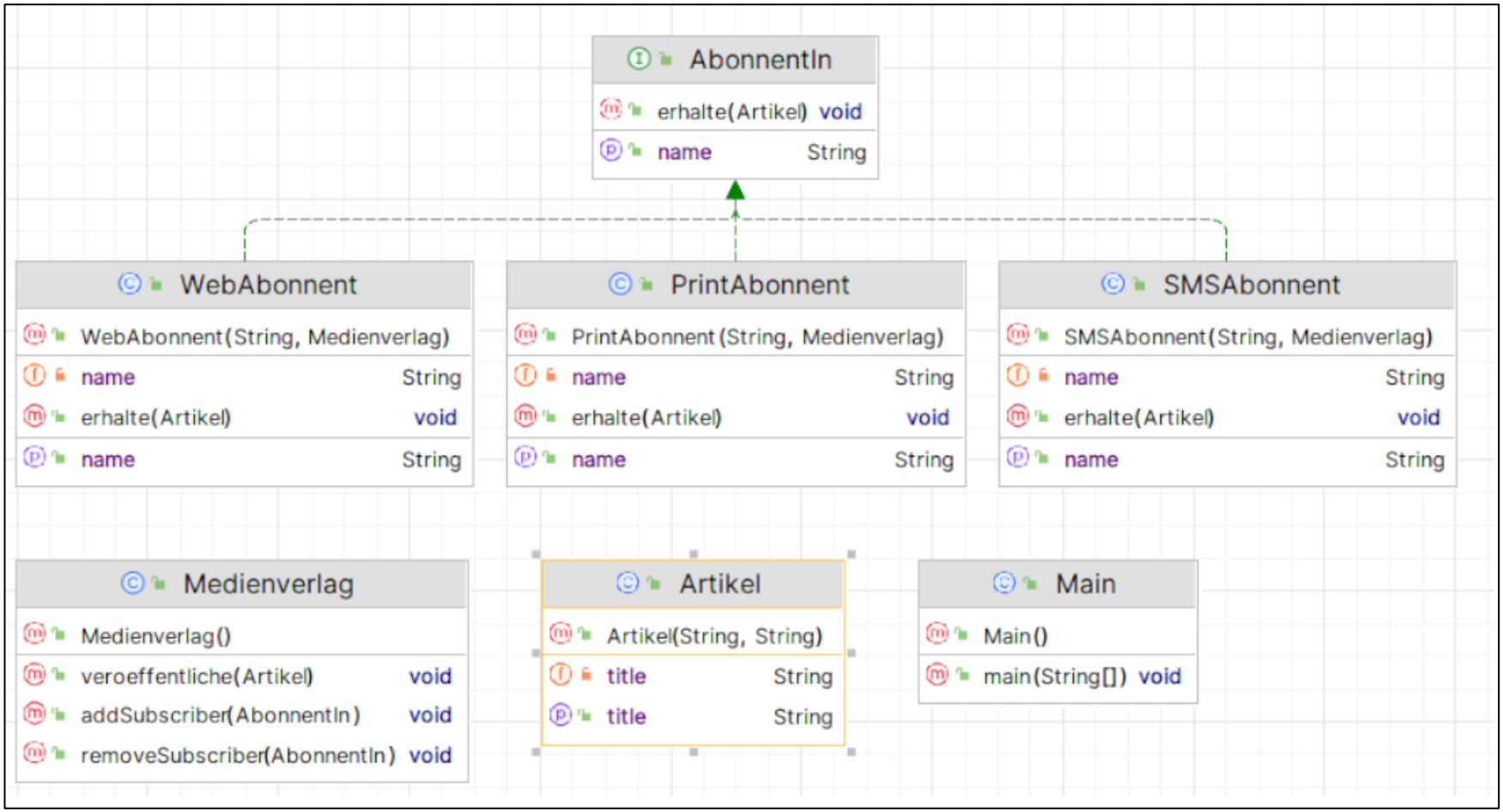
LOSE KOPPLUNG DURCH OBSERVER PATTERN

- Subject weiß über Observer nur, dass Interface implementiert wird
- Neue Observer können während Laufzeit hinzugefügt werden
- Subject muss nicht angefasst werden um Observer hinzuzufügen
- Subjects und Observers können unabhängig weiterverwendet werden
- Änderung von Subject/Observer hat keinen Einfluss auf jeweils anderen

OBSERVER ÜBUNGSBEISPIEL MEDIENVERLAG

Ein Medienverlag verfasst Nachrichtenartikel (bestehen aus Titel und Inhalt) und möchte diese an Print- und Email-Abonnenten*Abonentinnen ausliefern. Setzen Sie das Beispiel mit dem Observer Pattern um.

KLASSENDIAGRAMM



OBSERVER ÜBUNG: PODCAST

- Freie Aufgabenstellung: Beispielangabe lässt einige eigene Annahmen zu!
- Erstellen Sie eine Klasse PodcastServer die einzelne Podcasts verwalten kann. Ein Podcast (eigene Klasse) besteht aus einem Namen (String), einem URL (String) und einer Länge in Minuten (double). Erstellen Sie dafür Konstruktor, getter und setter.
- Der PodcastServer soll alle bisherigen Podcasts speichern und eine Methode bieten um einen neuen Podcast hinzuzufügen. Wenn ein neuer Podcast hinzugefügt wird sollen alle PodcastabonnentInnen über den neuen Podcast informiert werden.
- Die Geräte Notebook, Smartphone und Smartwatch (jeweils Klasse erstellen) haben Podcast Player integriert und sollen die Funktionalität erhalten sich bei Interesse am Podcast Server zu registrieren bzw. informiert zu werden wenn es neue Podcasts gibt.

PODCAST: KLASSENDIAGRAM