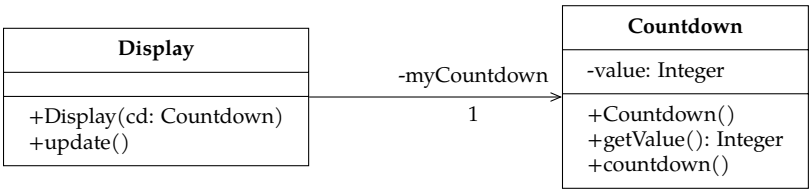
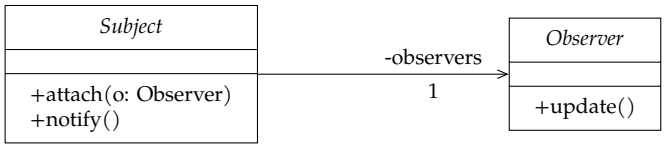
Aufgabe 2: (Anwendung des Observer-Patterns)

Es soll eine (kleine) Anwendung entwickelt werden, in der ein Zähler in 1-er Schritten von 5000 bis 0 herunterzählt. Der Zähler soll als Objekt der Klasse `Countdown` realisiert werden, die in UML-Notation dargestellt ist. Das Attribut value soll den aktuellen Zählerstand speichern, der mit dem Konstruktor zu initialisieren ist. Die Methode `getValue` soll den aktuellen Zählerstand liefern und die Methode `countdown` soll den Zähler von 5000 bis 0 herunterzählen.

Der jeweilige Zählerstand soll von einem Objekt der in untenstehender Abbildung angegebenen Klasse `Display` am Bildschirm ausgegeben werden. Bei der Konstruktion eines `Display`-Objekts soll es mit einem `Countdown`-Objekt verbunden werden, indem dessen Referenz unter `myCountdown` abgespeichert wird. Die Methode `update` soll den aktuellen Zählerstand vom `Countdown`-Objekt holen und mit `System.out.println` am Bildschirm ausgeben. Dies soll zu Beginn des Zählprozesses und nach jeder Änderung des Zählerstands erfolgen.



Damit das `Display`-Objekt über Zählerstände des `Countdown`-Objekts informiert wird, soll das Observer-Pattern angewendet werden. Untenstehende Abbildung zeigt die im Observer-Pattern vorkommenden abstrakten Klassen. (Kursivschreibweise bedeutet abstrakte Klasse bzw. abstrakte Methode.)



(a) Welche Wirkung haben die Methoden `attach` und `notify` gemäß der Idee des Observer-Patterns?

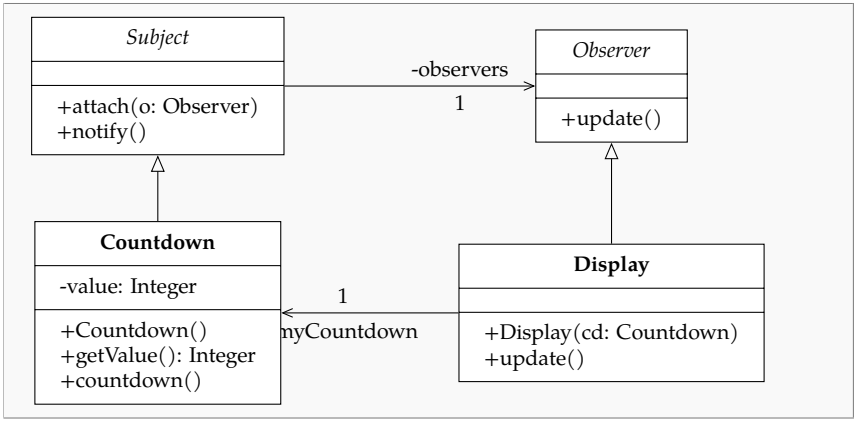
> Das beobachtete Objekt bietet mit der Methode `attach` einen Mechanismus, um Beobachter anzumelden und diese über Änderungen zu informieren.
>
> Mit der Methode `notify` werden alle Beobachter benachrichtigt, wenn sich das beobachtete Objekt ändert.

(b) Welche der beiden Klassen `Display` und `Countdown` aus obenstehender Abbildung spielt die Rolle eines `Subject` und welche die Rolle eines `Observer`?

> Die Klasse `Countdown` spielt die Rolle des `SubjectS`, also des Gegenstands, der beobachtet wird.
>
> Die Klasse `Display` spielt die Rolle eines `Observer`, also die Rolle eines

Beobachters.

(c) Erstellen Sie ein Klassendiagramm, das die beiden obenstehenden gegebenen Diagramme in geeigneter Weise, d. h. entsprechend der Idee des Observer-Patterns, zusammenfügt. Es reicht die Klassen und deren Beziehungen anzugeben. Eine nochmalige Nennung der Attribute und Methoden ist nicht notwendig.



(d) Unsere Anwendung soll nun in einer objektorientierten Programmiersprache Ihrer Wahl (z. B. Java oder C++) implementiert werden. Dabei soll von folgenden Annahmen ausgegangen werden:

- Das Programm wird mit einer main-Methode gestartet, die folgenden Rumpf hat:

```
1  public static void main(String[] args){
2    Countdown cd = new Countdown();
3    new Display(cd);
4    cd.countdown();
5  }
```

- Die beiden Klassen Subject und Observer sind bereits gemäß der Idee des Observer-Patterns implementiert.

Geben Sie auf dieser Grundlage eine Implementierung der beiden Klassen `Display` und `Countdown` an, so dass das gewünschte Verhalten, d. h. Anzeige der Zählerstände und Herunterzählen des Zählers, realisiert wird. Die Methoden der Klassen `Subject` und `Observer` sind dabei auf geeignete Weise zu verwenden bzw. zu implementieren. Geben Sie die verwendete Programmiersprache an.

```
3  public class Client {
4    public static void main(String[] args){
5      Countdown cd = new Countdown();
6      new Display(cd);
7      cd.countdown();
8      cd.countdown();
9      cd.countdown();
10   }
11 }
```

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

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

  public void attach(Observer o) {
    observers.add(o);
  }

  public void notifyObservers() {
    for (Observer o : observers) {
      o.update();
    }
  }
}
```

```java
public abstract class Observer {
  public abstract void update();
}
```

```java
public class Countdown extends Subject {

  private int value;

  public Countdown() {
    value = 5000;
  }

  public int getValue() {
    return value;
  }

  public void countdown() {
    if (value > 0) {
      notifyObservers();
      value--;
    }
  }
}
```

```java
public class Display extends Observer {
  Countdown myCountdown;
  public Display(Countdown cd) {
    myCountdown = cd;
    myCountdown.attach(this);
  }

  public void update() {
    System.out.println(myCountdown.getValue());
  }
}
```