Softwaretechnik

Hausaufgabenblatt 9

Patrick Gustav Blaneck

Letzte Änderung: 17. Januar 2022

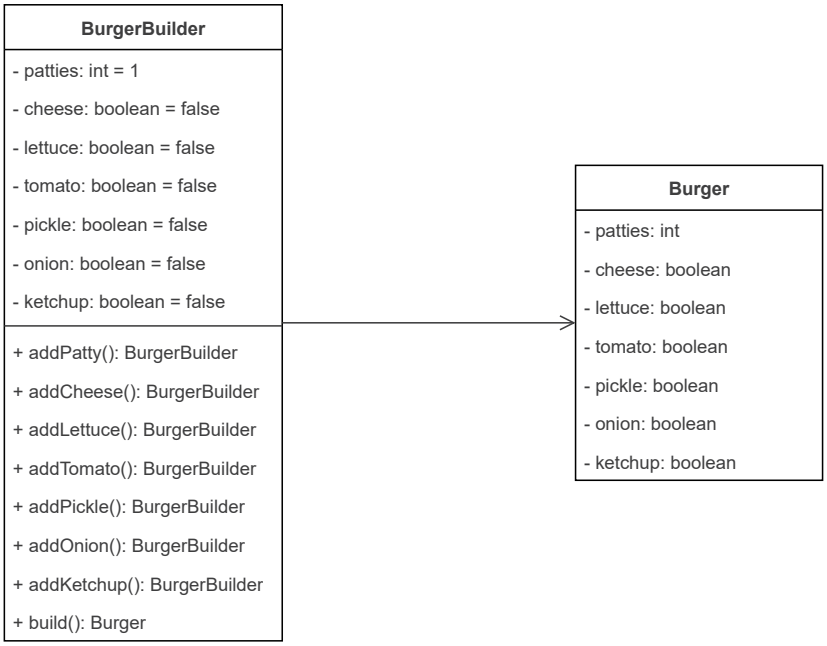
1. **Builder Pattern**
   In der Vorlesung wurde das Builder Pattern *nicht* vorgestellt.

   (a) Erklären Sie mit eigenen Worten, welchen Nutzen das Builder Pattern bringt und wo die typischen Einsatzgebiete liegen.

> **Lösung:**
>
> StackOverflow-Nutzer Hoopjie fasst es sehr gut zusammen:
>
> > Using a builder pattern has a few advantages:
> >
> > 1. Unlike with setters (which make your class mutable), a builder can be used to contruct immutable objects. In many cases immutable objects are preferred over mutable objects, because they are easier to understand and maintain, and because they avoid the need for locking in multithreaded environments.
> > 2. A builder can make sure that the object satisfies some invariants even directly after construction. For example, if your class has a `name` field which must never be null, the builder can check this condition and fail to construct the object when not satisfied.
> >
> > Both things you can also accomplish by using a constructor which takes all the class contents as parameters, but that will be quite unreadable when your class has more than a few fields to initialize.

(b) Finden Sie ein geeignetes Beispiel und erklären Sie dieses. Modellieren (Klassendiagramm) und *implementieren* Sie für diesen Anwendungsfall das Builder Pattern.

Lösung:



Der dazugehörige Code befindet sich im Anhang.

2. **Sequenzdiagramm: Wetterstation**

In der Vorlesung haben Sie das Observer Pattern (zu deutsch Beobachter Entwurfsmuster) am Beispiel einer Wetterstation kennengelernt. Als Beispiel für konkrete Implementierungen der Interfaces Subject und Observer werden die Klassen `WeatherData` bzw. `CurrentConditionsDisplay` aus dem Foliensatz verwendet.

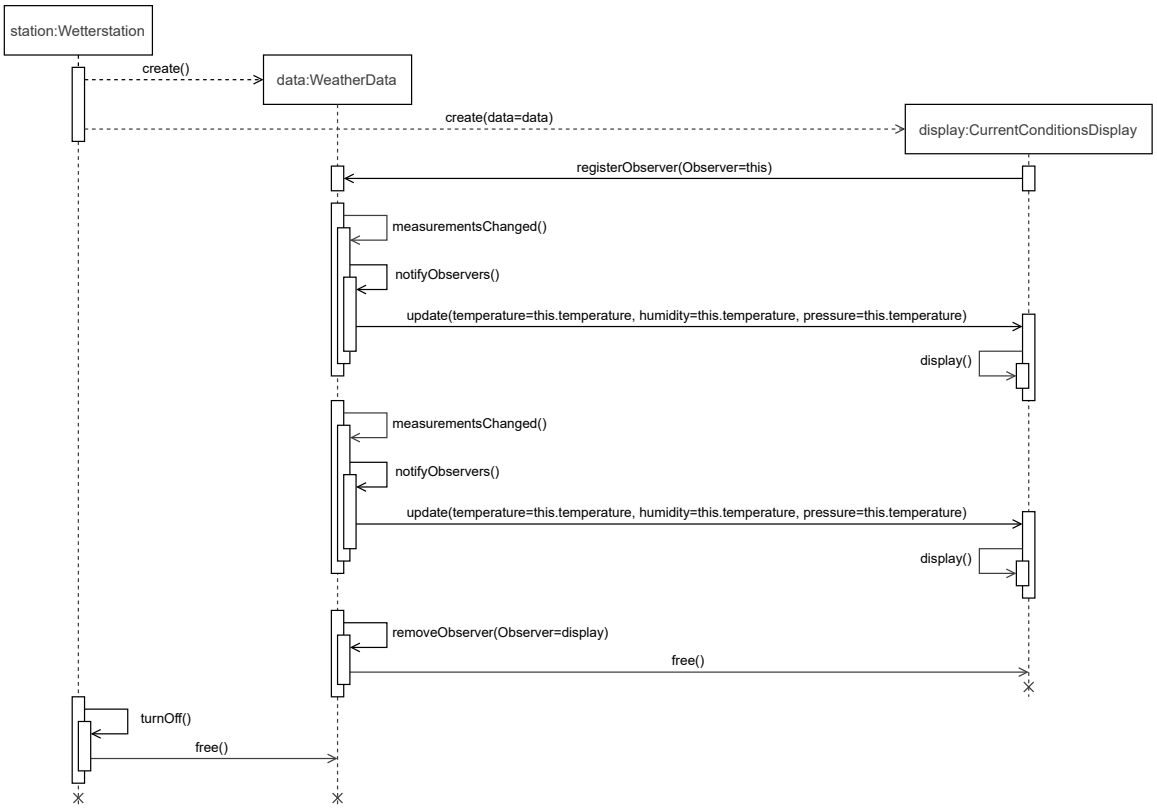
Modellieren Sie in Visual Paradigm ein Sequenzdiagramm, das diesen Sachverhalt darstellt. Das Diagramm soll die Interaktion der Klassen für den folgenden Programmablauf modellieren:

1. Eine Wetterstation startet und erstellt jeweils eine neue Instanz der Klassen `WeatherData` und `CurrentConditionsDisplay`. (Das Display bekommt die Wetterdaten im Konstruktor übergeben und registriert sich anschließend als Observer, siehe Folien)
2. Es treten zwei Veränderungen des Wetters auf. Modellieren Sie einen Pull oder einen Push-Observer (Orientieren Sie sich an den Folien für einen Push-Observer)
3. Der Observer wird vom Subject entfernt.
4. Die Wetterstation wird abgeschaltet und der Speicher wird freigegeben.

Hinweise:

- Achten Sie zudem darauf, dass Methodenaufrufe innerhalb der selben Klasse richtig dargestellt werden.
- Benutzen Sie explizite Namen mit Typ für die Benennung der Teilnehmer.
- Benutzen Sie für Nachrichten die Form
  `<Nachrichtenname> (<Parametername>=<Argumentwert>,...)`

Lösung:



```java
class BurgerBuilder {
    int patties = 1;
    boolean cheese = false;
    boolean lettuce = false;
    boolean tomato = false;
    boolean pickle = false;
    boolean onion = false;
    boolean ketchup = false;

    public BurgerBuilder addPatty() {
        patties++;
        return this;
    }

    public BurgerBuilder addCheese() {
        cheese = true;
        return this;
    }

    public BurgerBuilder addLettuce() {
        lettuce = true;
        return this;
    }

    public BurgerBuilder addTomato() {
        tomato = true;
        return this;
    }

    public BurgerBuilder addPickle() {
        pickle = true;
        return this;
    }

    public BurgerBuilder addOnion() {
        onion = true;
        return this;
    }

    public BurgerBuilder addKetchup() {
        ketchup = true;
        return this;
    }

    public Burger build() {
        return new Burger(this);
    }
}
```

Listing 1: `BurgerBuilder.java`

```java
class Burger {
    private int patties;
    private boolean cheese;
    private boolean lettuce;
    private boolean tomato;
    private boolean pickle;
    private boolean onion;
    private boolean ketchup;

    public Burger(BurgerBuilder builder) {
        this.patties = builder.patties;
        this.cheese = builder.cheese;
        this.lettuce = builder.lettuce;
        this.tomato = builder.tomato;
        this.pickle = builder.pickle;
        this.onion = builder.onion;
        this.ketchup = builder.ketchup;
    }
}
```

Listing 2: `Burger.java`