Singletons und Factory-Methoden

Peter Willadt

2022-03-71

Zusammenfassung

Manchmal ist es schlecht, wenn ein Objekt über den Konstruktor einer Klasse erzeugt wird. Für solche Fälle gibt es Singletons und Factory-Methoden.

1 Singleton

Ein Singleton ist eine Klasse, von der es genau *eine* Instanz gibt.

Singletons sind sinnvoll, wenn es globale Objekte (zum Beispiel eine Datenbankverbindung oder Programmeinstellungen) gibt. Auch in der Java-Bibliothek gibt es Singletons. Ein Beispiel ist die Klasse *Calendar*.

In Java gibt es aus gutem Grund keine globalen Variablen. Mit Singletons kann man leider auch dieses Sprachfeature unterlaufen. Man sollte es also nicht übertreiben.

In Java kann man einfach ein Singleton schreiben:

- Die Klasse erhält einen privaten Konstruktor.
- Die Klasse hat eine private statische Variable für ein Objekt dieser Klasse.
- Die Klasse hat eine öffentliche statische Methode getInstance().
- getInstance() macht folgendes:
  - Wenn die statische Variable keinen Inhalt hat, wird der Konstruktor aufgerufen und ihr das neue Objekt zugewiesen.
  - Sie liefert den Wert der statischen Variablen zurück.

1.1 Beispiel: Datenbankverbindung

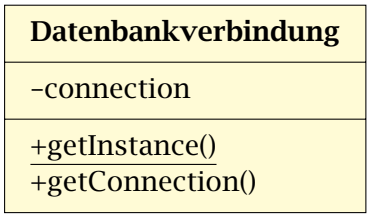
```
1  public class Datenbankverbindung {
2    static private Datenbankverbindung datenbankVerbindung;
3    private Connection connection;
4    private boolean verbinde()  {
5      if(connection != null)
6        return true;
7      try {
8        // kein try-with-resources, die connection soll offen bleiben.
9        connection = DriverManager.getConnection( /* Verbindungsdaten */);
10     } catch (SQLException e) {
11       e.printStackTrace();
12       return false;
13     }
14     // gleich automatische Trennung bei Programmende aktivieren.
15     class Datenbanktrenner extends Thread {
16       private Connection verbindung;
17       public void run() {
18         try {
19           if(verbindung != null)
20             verbindung.close();
21         } catch (SQLException e) {
22           // eh alles egal.
23           e.printStackTrace();
24         }
25       }
26       public void setVerbindung(Connection v) {
27         verbindung = v;
28       }
29     }
30     // und setzen
31     Datenbanktrenner trenner = new Datenbanktrenner();
32     trenner.setVerbindung(connection);
33     Runtime.getRuntime().addShutdownHook(trenner);
34     return true;
35   }
36   private Datenbankverbindung() {
37     verbinde();
38   }
39   public static Datenbankverbindung getInstance() {
40     if(datenbankVerbindung == null) {
41       datenbankVerbindung = new Datenbankverbindung();
42     }
43     return datenbankVerbindung;
44   }
45   public Connection getConnection() {
46     return connection;
47   }
48 }
```

Der Beispielcode ist etwas länger, weil die automatische Trennung von der Datenbank beim Programmende gleich mit eingebunden ist. Das Singleton wird dann wie folgt genutzt:

```
1 Connection connection = Datenbankverbindung.getInstance().getConnection
    ();
2 // oder ausfuehrlicher:
3 Datenbankverbindung verbindung = Datenbankverbindung.getInstance();
4 Connection connection = verbindung.getConnection();
```

Das Singleton hat folgende Vorteile:

- Der Code zur Datenbankanbindung ist in einer separaten Klasse gekapselt.
- Wenn eine Methode eine Datenbankverbindung benötigt, kann sie sie selbst anfordern, sie muss nicht mehr als Parameter übergeben werden.
- Die Verbindung wird bei Bedarf aufgebaut, versehentlich mehrfaches Aufbauen einer Verbindung ist nicht möglich.



Wichtigste Nachteile:

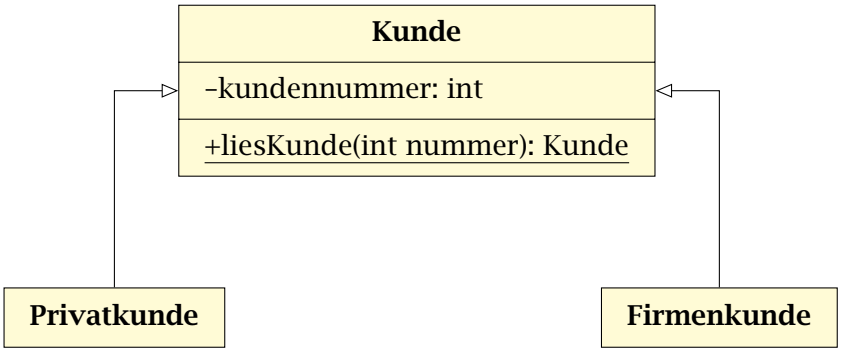
- Bisher konnte man am Kopf einer Methode sehen, ob sie einen Datenbankzugriff braucht. Das geht jetzt nicht mehr.
- Der Zugriff über die Singleton-Klasse ist geringfügig langsamer als ein direkter Zugriff.

2 Factory-Methode

Eine Factory-Methode wird ähnlich wie ein Singleton benutzt, kann aber intern durchaus neue Objekte erstellen. Wann braucht man eine Factory-Methode?

- Bei der Erstellung eines Objektes müssen Dinge ausgeführt werden, die sich nicht einfach aus dem Konstruktor ausführen lassen.
- Es wird eine (besser geeignete) Unterklasse statt der Oberklasse aufgerufen.

2.1 Beispiel: Kunde



Es gibt Privatkunden und Firmenkunden. Beide sind Unterklassen von *Kunde*. Für viele Funktionen der Software ist es egal, ob ein Kunde Privatkunde ist. Der Factory-Code sieht so aus (Pseudo-Code):

```
Kunde liesKunde(int nummer) {
  Kunde kunde = null;
  // Kundenvorabinfo aus der Datenbank holen
  // SELECT Kundenart FROM KUNDE WHERE Kundennummer = ?
  if(rs.getInteger(1) == Kunde.PRIVATKUNDE) {
    kunde = new Privatkunde();
    kunde.liesBasisdaten(nummer);
  }else {
    kunde = new Firmenkunde();
    kunde.liesBasisdaten(nummer);
  }
  return kunde;
}
```

Die Klasse *Kunde* hat eine Methode `liesKunde()`, die einen Kunden aus der Datenbank holt. Je nachdem erhält die aufrufende Methode einen Privatkunden oder einen Firmenkunden zurück. Falls die aufrufende Methode genaueres wissen möchte, kann sie zum Beispiel mit `instanceof` nachforschen. So wird der Factory-Code benutzt:

```
  Kunde kunde = Kunde.liesKunde(nummer);
  // jede Menge Code fuer alle Arten von Kunde
  Zahlungsart zahlungsart = kunde.getZahlungsart();
  // wenn es speziell wird:
  if(kunde instanceof(Privatkunde)) {
    // Extrawurst
  }
```