Hochschule
Kaiserslautern
University of
Applied Sciences

Andreas Eitel
University of Applied Sciences
Kaiserslautern
andreas.eitel@hs-kl.de

SOFTWARE-ARCHITEKTUR:

AUFGABENPAKET 2

13. Februar 2022

Vorbemerkung

Die Hauptthemen dieses Arbeitspakets sind nochmal „Verteilte Systeme“ und die SOLID-Prinzipien. Lesen Sie vor Bearbeitung der Übung die fehlenden Abschnitte 6.3 und 6.4 des Kapitels „Verteilte Systeme mit RMI und JMS“. Bei den verteilten Systemen steht diesmal der Einsatz von JMS und REST als Integrationstechnologien im Vordergrund. Für die Implementierung von JMS können Sie wie in der Vorlesung gezeigt (Folien) vorgehen. Außerdem behandelt dieses Aufgabenpaket die Theorie zu den SOLID-Prinzipien. Lesen Sie zur Vorbereitung der SOLID-Prinzipien Kapitel 3 im Skript. Praktische Übungen zu den SOLID-Prinzipien folgen in der nächsten Übung.

Weitere vorbereitende Literatur

Die SOLID-Prinzipien können auch nochmal auf Wikipedia und in dem Artikel *Design Principles and Design Patterns* [Mar00] nachgelesen werden. Neben den SOLID-Eigenschaften wird in dem Artikel auch auf die Kopplung und Kohäsion eingegangen und verschiedene Maßzahlen dafür definiert. Der Artikel [Mar00] wurde später von Robert Martin überarbeitet und in sein Buch [Mar03] integriert. Den Artikel [Mar00] und ein weiterer aus der Zeitschrift Dr. Dobb's finden Sie im Übungspaket.

Theoretischer Teil

1. Beantworten Sie schriftlich:
   (a) (2 Punkte) Nennen Sie mindestens zwei Gründe für den Einsatz eines Messaging-Systems.
   (b) (2 Punkte) Erklären Sie kurz und mit eigenen Worten den Unterschied zwischen den Messaging-Modellen „Publisher-Subscriber“ und „Sender-Receiver“.
   (c) (2 Punkte) Beschreiben Sie die beiden Begriffe „(enge und lose) Kopplung“ und „Kohäsion“ mit eigenen Worten.

(d) (2½ Punkte) Beschreiben Sie die fünf SOLID-Prinzipien anhand von selbstgewählten Beispielen. Fertigen Sie hierzu ein kleines Dokument an (max. 1 Seite), das Sie über die Plattform einreichen.

Praktischer Teil

Nutzen Sie eclipse oder IntelliJ zum Bearbeiten der nachfolgenden Aufgaben. Laden Sie Ihre Lösung als ZIP-Datei hoch. Nutzen Sie Maven, um eine (bzw. mehrere) startfähige *.jar-Dateien zu bauen und laden Sie diese ebenfalls hoch. Build-Artefakte (*.class-Dateien) sollten nicht hochgeladen werden.

2. **Buchlager: Anbindung eines Lieferanten (ActiveMQ) und der Hauptzentrale (REST)**
Diese Aufgabe erweitert das Buchlager von Aufgabenblatt 1, Aufgabe 4. Die Anwendung verwaltet intern den Lagerbestand der einzelnen Bücher. Die `BuchlagerFacade` stellt hierzu entsprechende Methoden bereit. Wird bei der Warenkorbanzeige (Klasse `JPanelWarenkorb`) der Kaufen-Button betätigt, wird in dem zugehörigen Listener (siehe Listing 1, ❶) der Buchbestand verringert. Im Beispiel sieht man den ursprünglichen Quellcode im Client. Nachdem die Schichten in Übung 1 getrennt wurden, ist die Implementierung im Server durchzuführen.

```
1  this.kaufenButton.addActionListener(new ActionListener() {
2    @Override
3    public void actionPerformed(ActionEvent e)
4    {
5      Enumeration<Buch> en = model.elements();
6      while( en.hasMoreElements() )
7      {
8          Buch buch = en.nextElement();
9          try
10         {
11           BuchlagerFacade.getInstance()
12                          .bestandAusbuchen(buch.getId(), 1);  ❶
13           System.out.println("Das␣Buch␣" + buch.getTitel()
14                                          + "␣wird␣versendet");
15         }
16         catch (BuchlagerFacadeException e1)
17         {
18           System.out.println("Das␣Buch␣" + buch.getTitel()
19                                 + "␣zur␣Zeit␣nicht␣lieferbar");
20         }
21       }
22       model.removeAllElements();
23     }
24   });
```

Listing 1: Beim „Kaufen“ des Warenkorbs werden die Lagerbestände aktualisiert

Ziel dieser Aufgabe ist es nun, das Buchlager mit einem Lieferanten und der Hauptzentrale zu verknüpfen. Wenn ein Buchbestand auf Null gesunken ist, sollen drei Exemplare des Buchs automatisch beim Lieferanten nachbestellt werden. Dem Lieferanten wird hierzu über ein Messaging-System (ActiveMQ) eine entsprechende Nachricht gesendet. Der Lieferant gibt nach Erhalt der Nachricht einfach eine Log-Ausgabe auf der Konsole aus.

Der Buchladen wird nach einer „Bearbeitungszeit“ von 5 Sekunden den entsprechenden Buchbestand automatisch aktualisieren[1].

Die Hauptzentrale erhält eine Nachricht über die Kundenbestellung über eine REST-Schnittstelle. In der Zentrale werden die eingegangenen Bestellungen ebenfalls einfach auf der „Konsole“ ausgegeben.

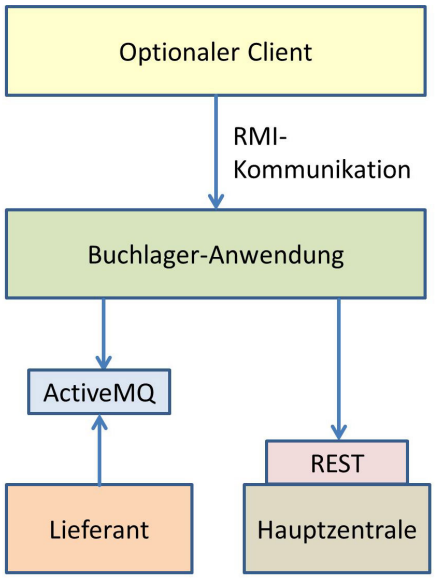

Abbildung 1: Integration von Buchlager- und Lieferantenanwendung.

Bei der Aufgabe sollten Sie an der ursprünglichen Client/Server-Variante (Aufgabenblatt 1, Aufgabe 4) anknüpfen (Abbildung 1 stellt den Aufbau nochmal grafisch dar).

Zu implementieren sind eine einfache Lieferanten- und Hauptzentrale-Anwendung. Eine Bestellung besteht aus der Buch-Id und der Bestellanzahl. Sind mehrere Bücher zu bestellen, wird pro Buch eine Nachricht versendet. Die Bestellungen sollen über eine Queue an den Lieferanten und über eine REST-Schnittstelle an die Hauptzentrale gegeben werden. Dabei sollen die Nachrichten nicht als `String` übermittelt werden, sondern als „Objekte“.

(a) (2 Punkte) Die Lieferantenanwendung (JMS-Anwendung) ist korrekt implementiert. Sie gibt nachbestellte Bücher auf der Konsole aus. Bestellungen werden als „Objekt“ übermittelt.

(b) (2 Punkte) Die Hauptzentrale-Anwendung (REST-Server) ist korrekt implementiert. Sie gibt bestellte Bücher auf der Konsole aus. Bestellungen werden als „Objekt“ übermittelt.

(c) (3 Punkte) In die Buchlager-Anwendung ist ein Bestell-Client (JMS-Client) und eine Report-Funktion (REST-Client) integriert. Das Gesamtsystem funktioniert.

[1] Nach dem Absenden der Bestellung in die Queue kann ein Thread gestartet werden, der nach max. 5 Sekunden den Bestand aktualisiert.

Tipps zur Implementierung

Da sich die Menge der Abhängigkeiten bzw. Bibliotheken erhöht, empfiehlt es sich mit „Maven"-Projekten zu arbeiten. Sie können die Vorlagen (Projekte) aus der Präsenz einfach entsprechend kopieren und dann modifizieren. Damit in den Projektgerüsten, die in der Präsenz benutzt wurden, ActiveMQ benutzt werden kann, muss in der `pom.xml`-Datei noch zusätzlich folgende Dependency angegeben werden:

```
<dependency>
  <groupId>org.apache.activemq</groupId>
      <artifactId>activemq-client</artifactId>
      <version>5.16.3</version>
</dependency>
```

Das folgende Listing zeigt eine Möglichkeit, wie der Bestand eines Buches nach einer Bestellung um 5 Sekunden verzögert aktualisiert werden kann. Man beachte, dass in dem Fall die Methode `bestandEinbuchen` *thread safe* sein muss. Wie bereits oben beschrieben: Beachten Sie, dass das Listing den ursprünglichen Quellcode zeigt und nicht Ihre umgebaute Variante. Ihre Implementierung sollte daher an der entsprechenden Stelle im Server erfolgen.

```
this.kaufenButton.addActionListener(new ActionListener() {
  @Override
  public void actionPerformed(ActionEvent e)
  {
    Enumeration<Buch> en = model.elements();
    while( en.hasMoreElements() )
    {
      Buch buch = en.nextElement();
      try
      {
        BuchlagerFacade.getInstance()
                       .bestandAusbuchen(buch.getId(), 1);
        System.out.println("Das Buch " + buch.getTitel()
                           + " wird versendet");

        if( BuchlagerFacade.getInstance()
                           .getBestand(buch.getId()) < 1 )
        {
          // Sende Nachricht an Lieferent
          // per JMS und bestelle 5 Exemplare
          // ....
          // --> Hier steht der JMS-Code

          // Einfache Realisierung einer verzoegerten
          // Bestandsaktualisierung
          // Nach 5 Sekunden werden 5 Buchexemplare eingebucht
          new Thread( new Runnable()
          {
              @Override
              public void run()
              {
                try
                {
                  TimeUnit.SECONDS.sleep(5);
                  BuchlagerFacade.getInstance()
```

```
36                                      .bestandEinbuchen(buch.getId(), 5);
37                  }
38                  catch (Exception e)
39                  {
40                    e.printStackTrace();
41                  }
42                }
43              } ).start();
44            }
45
46          }
47          catch (BuchlagerFacadeException e1)
48          {
49            System.out.println("Das Buch " + buch.getTitel()
50                                + " zur Zeit nicht lieferbar");
51          }
52        }
53        model.removeAllElements();
54      }
55    });
```

Literatur

[Mar00] Martin, Robert: *Design Principles and Design Patterns*. http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf. Version: 2000

[Mar03] Martin, Robert C.: *Agil Software Development. Principles, Patterns, and Practices*. Prentice-Hall, 2003