

# MidEng 7.2 Warehouse Message Oriented Middleware

Kevin Bauer 4CHIT - 21.11.2023

## CODE + Erklärung:

### Sender:

```
package tradearea.warehouse;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.activemq.ActiveMQConnection;
import org.apache.activemq.ActiveMQConnectionFactory;
import tradearea.model.Product;
import tradearea.model.WarehouseData;

import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.DeliveryMode;
import javax.jms.Destination;
import javax.jms.MessageProducer;
import javax.jms.Session;
import javax.jms.TextMessage;
import java.util.ArrayList;
import java.util.List;

public class MOMSender {

    private static String user = "admin";
    private static String password = "admin";
    private static String url = ActiveMQConnection.DEFAULT_BROKER_URL;
    private static String subject = "warehouse-001";

    public MOMSender() throws JsonProcessingException {

        WarehouseSimulation warehouseSimulation = new WarehouseSimulation();
        ObjectMapper mapper = new ObjectMapper();
        String jsonString =
mapper.writeValueAsString(warehouseSimulation.getData("1"));

        System.out.println( "Sender started." );

        // Create the connection.
        Session session = null;
        Connection connection = null;
        MessageProducer producer = null;
        Destination destination = null;

        try {
            System.out.println(user);
```

```

        System.out.println(password);
        System.out.println(url);
        ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(
user, password, url );
        connection = connectionFactory.createConnection();
        connection.start();

        // Create the session
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        destination = session.createTopic( subject );

        // Create the producer.
        producer = session.createProducer(destination);
        producer.setDeliveryMode( DeliveryMode.NON_PERSISTENT );

        // Create the message
        TextMessage message = session.createTextMessage(jsonString);
        producer.send(message);
        System.out.println( message.getText() );

        connection.stop();

    } catch (Exception e) {

        System.out.println("[MessageProducer] Caught: " + e);
        e.printStackTrace();

    } finally {

        try { producer.close(); } catch ( Exception e ) {}
        try { session.close(); } catch ( Exception e ) {}
        try { connection.close(); } catch ( Exception e ) {}

    }
    System.out.println( "Sender finished." );

} // end main
}

```

Hier sieht man den Code des Senders.

Zuerst wird die Connection hergestellt. Dann werden die Daten in einen String umgewandelt und gesendet.

## Receiver:

```

package tradearea.warehouse;

import org.apache.activemq.ActiveMQConnection;
import org.apache.activemq.ActiveMQConnectionFactory;

import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.DeliveryMode;

```

```

import javax.jms.Destination;
import javax.jms.MessageConsumer;
import javax.jms.Session;
import javax.jms.TextMessage;

public class MOMReceiver {

    private static String user = "admin";
    private static String password = "admin";
    private static String url = ActiveMQConnection.DEFAULT_BROKER_URL;
    private static String subject = "warehouse-001";

    public MOMReceiver() {

        System.out.println( "Receiver started." );

        // Create the connection.
        Session session = null;
        Connection connection = null;
        MessageConsumer consumer = null;
        Destination destination = null;

        try {

            ConnectionFactory connectionFactory = new
ActiveMQConnectionFactory(user, password, url);
            connection = connectionFactory.createConnection();
            connection.start();

            // Create the session
            session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
            destination = session.createTopic( subject );

            // Create the consumer
            consumer = session.createConsumer( destination );

            // Start receiving
            TextMessage message = (TextMessage) consumer.receive();
            while ( message != null ) {
                System.out.println("Message received: " + message.getText() );
                message.acknowledge();
                message = (TextMessage) consumer.receive();
            }
            connection.stop();

        } catch (Exception e) {

            System.out.println("[MessageConsumer] Caught: " + e);
            e.printStackTrace();

        } finally {

            try { consumer.close(); } catch ( Exception e ) {}
            try { session.close(); } catch ( Exception e ) {}
            try { connection.close(); } catch ( Exception e ) {}

        }

    }

}

```

```
}  
    System.out.println( "Receiver finished." );  
  
} // end main  
  
}
```

Hier sieht man den Code des Senders.

Die Daten werden in der while Schleife konstant empfangen.

## Fragestellung für Protokoll

### 1. Nennen Sie mindestens 4 Eigenschaften der Message Oriented Middleware?

- **Asynchrone Kommunikation:** Erlaubt das Senden und Empfangen von Nachrichten ohne dass Sender und Empfänger gleichzeitig aktiv sein müssen.
- **Entkopplung:** Sender und Empfänger sind voneinander unabhängig, was die Flexibilität und Skalierbarkeit verbessert.
- **Zuverlässigkeit:** Sicherstellung, dass Nachrichten auch bei Systemausfällen nicht verloren gehen.
- **Skalierbarkeit:** Fähigkeit, mit der Last zu wachsen, indem weitere Instanzen hinzugefügt werden.

### 2. Was versteht man unter einer transienten und synchronen Kommunikation?

- **Transiente Kommunikation:** Nachrichten werden nur übertragen, wenn Sender und Empfänger gleichzeitig verfügbar sind. Die Nachrichten werden nicht gespeichert.
- **Synchrone Kommunikation:** Der Sender wartet auf eine unmittelbare Antwort vom Empfänger, bevor er mit der Verarbeitung fortfährt.

### 3. Beschreiben Sie die Funktionsweise einer JMS Queue?

- Eine JMS Queue ist ein Punkt-zu-Punkt-Kommunikationsmodell. Eine Nachricht, die an eine Queue gesendet wird, wird von genau einem Empfänger konsumiert.
- Der Sender sendet Nachrichten an die Queue, ohne zu wissen, wer sie empfangen wird.
- Der Empfänger holt sich die Nachricht aus der Queue, wenn er bereit ist. Dies gewährleistet eine entkoppelte und asynchrone Kommunikation.

### 4. JMS Overview - Beschreiben Sie die wichtigsten JMS Klassen und deren Zusammenhang?

- **ConnectionFactory:** Erstellt Verbindungen zu einem JMS-Provider.
- **Connection:** Repräsentiert eine Verbindung zum JMS-Provider.
- **Session:** Ermöglicht das Erstellen von Nachrichten, Publishern und Subscribiers.
- **MessageProducer:** Sendet Nachrichten an eine Destination (Queue oder Topic).
- **MessageConsumer:** Empfängt Nachrichten von einer Destination.
- **Destination:** Ein Ort (Queue oder Topic), an den Nachrichten gesendet werden.

### 5. Beschreiben Sie die Funktionsweise eines JMS Topic?

- Ein JMS Topic unterstützt das Publish-Subscribe-Kommunikationsmodell.

- Mehrere Empfänger können sich bei einem Topic registrieren, um Nachrichten zu erhalten.
- Wenn eine Nachricht an ein Topic gesendet wird, erhalten alle aktiven Subscribers diese Nachricht. Dies ermöglicht eine effiziente Verteilung von Nachrichten an viele Empfänger.

**6. Was versteht man unter einem lose gekoppelten verteilten System? Nennen Sie ein Beispiel dazu. Warum spricht man hier von lose? `**

- Ein lose gekoppeltes verteiltes System ermöglicht die Interaktion zwischen Komponenten, die wenig voneinander wissen.
- **Beispiel:** Ein Web-Service System, in dem verschiedene Dienste über das Internet kommunizieren.
- **Lose Kopplung** bedeutet hier, dass die Dienste unabhängig voneinander entwickelt, bereitgestellt und skaliert werden können, ohne dass sie genau über die Implementierungsdetails der anderen Dienste Bescheid wissen müssen.