LÖSUNGEN
ABAS JAVA OBJECTS (AJO) - AUFBAU

2018r2n04

abas ACADEMY

RECHTLICHE HINWEISE

Die Namen und Daten in dieser Unterlage sind frei erfunden. Wir weisen darauf hin, dass die abas ERP-Version dieser Unterlage von Ihrer abas ERP-Version abweichen kann.

Der Inhalt wurde mit größter Sorgfalt erstellt. Sollten Sie dennoch einen Fehler finden, freuen wir uns über einen Hinweis. Bitte beachten Sie, dass die abas Software AG keine Gewähr für die Korrektheit der Inhalte übernimmt.

1. Ausgabe

1 LÖSUNGEN ZU DATENEXPORT NACH XML

LÖSUNG ZU ÜBUNG 1: ARTIKEL MIT STRUKTURSTÜCKLISTE EXPORTIEREN

Beachten Sie, dass die Selektion mit der Klasse **RowSelectionBuilder** Zeilen liefert. Mit der Methode **header** können Sie auf den Kopf zugreifen.

Führen Sie folgende Schritte aus:

1. Erzeugen Sie im Paket **de.abas.training.advanced.xml** eine neue Klasse **WriteProductsToXMLRowSelectionBuilder**, die im Server- und Client-Kontext ausgeführt werden kann.
2. Bauen Sie die gemeinsame Methode **run** ein.
3. Definieren Sie die Variable **fileOutputStream** vom Typ **FileOutputStream** und initialisieren Sie diese mit null.
4. Definieren Sie die Variable **xmlFile** vom Typ **String** und initialisieren Sie diese mit "xmlWrite/ProductsRowsRowSelection.xml".
5. Erzeugen Sie mit der Methode **RowSelectionBuilder.create(Product.class, Product.Row.class)** ein RowSelectionBuilder-Objekt und speichern Sie dieses in der Variablen **rowSelectionBuilder** vom Typ **RowSelectionBuilder**.
6. Definieren Sie mit der Methode **rowSelectionBuilder.addForHead(Conditions.between (Product.META.idno, "10053", "10060"))** die Selektionskriterien der gewünschten Artikeldatensätze.
7. Erzeugen Sie mit der Methode **dbContext.createQuery(rowSelectionBuilder.build())** das Query-Objekt und speichern Sie dieses in der Variablen **query** vom Typ **RowQuery<Product, Row>**.
8. ---
9. Definieren Sie mit der Methode **FieldSet.of("idno", "swd", "descrOperLang", "salesPrice", "DBNo", "grpNo")** das FieldSet mit den zu selektierenden Kopffeldern und speichern Sie dieses in der Variablen **fieldSetHead** vom Typ **FieldSet<FieldValueProvider>**.
10. Erzeugen Sie mit der Methode **fieldSetHead.getFields()** ein String-Array und speichern Sie es in der Variablen **headerFields**.
11. Weisen Sie mit der Methode **query.setFieldsForHead(fieldSetHead)** das Objekt **fieldSetHead** dem Objekt **query** zu.
12. ---
13. Definieren Sie mit der Methode **FieldSet.of("productListElem", "sparePartTab", "elemQty")** das FieldSet mit den zu selektierenden Tabellenfeldern und speichern Sie dieses in der Variablen **fieldSetRow** vom Typ **FieldSet<FieldValueProvider>**.
14. Erzeugen Sie mit der Methode **fieldSetRow.getFields()** ein String-Array und speichern Sie es in der Variablen **rowFields**.
15. Weisen Sie mit der Methode **query.setFields(fieldSetRow)** das Objekt **fieldSetRow** dem Objekt **query** zu.
16. ---
17. Erzeugen Sie mit der Methode **new Element("ABASData")** das RootElement und speichern Sie es in der Variablen **rootElement** vom Typ **Element**.
18. Erzeugen Sie mit der Methode **new Document(rootElement)** das Dokument und speichern Sie dieses in der Variablen **document** vom Typ **Document**.

19. Erzeugen Sie mit der Methode **new Element("RecordSet")** das RecordSet-Element und speichern Sie es in der Variablen **recordSet** vom Typ **Element**.
20. Definieren Sie mit der Methode **recordSet.setAttribute("action", "export")** das Attribut **action** mit dem Wert **export**.
21. Ermitteln Sie mit der Methode **Integer.toString(query.execute().get(0).header(). getDBNo().getCode())** vom ersten Selektionsobjekt die Datenbanknummer und speichern Sie diese in der Variablen **dbNo** vom Typ **String**.
22. Definieren Sie mit der Methode **recordSet.setAttribute("database", dbNo)** das Attribut **database** mit dem Inhalt der Variablen **dbNo**.
23. Ermitteln Sie mit der Methode **Integer.toString(query.execute().get(0).header(). getGrpNo())** vom ersten Selektionsobjekt die Gruppennummer und speichern Sie diese in der Variablen **grpNo** vom Typ **String**.
24. Definieren Sie mit der Methode **recordSet.setAttribute("group", grpNo)** das Attribut **group** mit dem Inhalt der Variablen **grpNo**.
25. Fügen Sie mit der Methode **rootElement.addContent(recordSet)** das Element **recordSet** zum Element **rootElement** hinzu.
26. Definieren Sie die Variable **actIdno** vom Typ **String** und initialisieren Sie diese mit "".
27. Definieren Sie die Variable **actRowNo** vom Typ **int** und initialisieren Sie diese mit 0.
28. Definieren Sie die Variable **record** vom Typ **Element** und initialisieren Sie diese mit null.

29. Erzeugen Sie eine for-each-Schleife über query. Das aktuelle Objekt steht über die Variable **row** vom Typ **Product.Row** zur Verfügung. Führen Sie je Schleifendurchlauf folgende Schritte aus:
    a. Prüfen Sie mit der Methode **if(!row.header().getIdno().equals(actIdno))**, ob ein neuer Artikel gelesen wurde. Falls ja, führen Sie folgende Schritte aus:
        i. Ermitteln Sie mit der Methode **row.header()** den zur Zeile gehörenden Artikel und speichern Sie diesen in der Variablen **product** vom Typ **Product**.
        ii. Geben Sie mit der Methode **dbContext.out().println("head schreiben - " + row.header().getIdno())** die Information aus.
        iii. Erzeugen Sie mit der Methode **new Element("Record")** ein neues Record-Element und speichern Sie dieses in der Variablen **record** vom Typ **Element**.
        iv. Fügen Sie mit der Methode **recordSet.addContent(record)** das Element **record** dem Element **recordSet** hinzu.
        v. Erzeugen Sie mit der Methode **new Element("Head")** ein neues Head-Element und speichern Sie dieses in der Variable **head** vom Typ **Element**.
        vi. Fügen Sie mit der Methode **record.addContent(head)** das Element **head** dem Element **record** hinzu.
        vii. Erzeugen Sie eine for-Schleife über die Elemente des String-Arrays **headerFields** und führen Sie je Schleifendurchlauf folgende Schritte aus:
            1. Rufen Sie die Methode **createHeaderFieldElement(product, headerFields[i])** auf. Speichern Sie das zurückgelieferte Objekt in der Variablen **field** vom Typ **Element**.
            2. Fügen Sie mit der Methode **head.addContent(field)** das Element **field** dem Element **head** hinzu.
        viii. Initialisieren Sie die Variable **actRowNo = 1**.
    b. Ermitteln Sie mit der Methode **row.header().getIdno()** die Identnummer des Artikels und speichern Sie diese in der Variablen **actIdno** vom Typ **String**.
    c. Ermitteln Sie mit der Methode **row.getProductListElem()** das Objekt im Zeilenfeld **Fertigungslistenelement** und speichern Sie dieses in der Variablen **selectableObject** vom Typ **SelectableObject**.
    d. Prüfen Sie mit der Methode **if (selectableObject instanceof Product)**, ob der Inhalt der Variablen **selectableObject** eine Instanz der Klasse **Product** ist. Falls ja, führen Sie folgende Schritte aus:
        i. Erzeugen Sie mit der Methode **new Element("Row")** ein neues Row-Element und speichern Sie dieses in der Variablen **rowElement** vom Typ **Element**.
        ii. Definieren Sie mit der Methode **rowElement.setAttribute("number", Integer.toString(actRowNo))** das Attribut **number** mit dem Inhalt der aktuellen Zeilennummer.
        iii. Erzeugen Sie eine for-Schleife über die Elemente des String-Arrays **rowFields** und führen Sie je Schleifendurchlauf folgende Schritte aus:
            1. Rufen Sie die Methode **createRowFieldElement(row, fields[i])** auf. Speichern Sie das zurückgelieferte Objekt in der Variablen **field** vom Typ **Element**.
            2. Fügen Sie mit der Methode **rowElement.addContent(field)** das Element **field** dem Element **rowElement** hinzu.
        iv. Fügen Sie mit der Methode **record.addContent(rowElement)** das Element **rowElement** dem Element **record** hinzu.
        v. Geben Sie mit der Methode **dbContext.out().println(actRowNo + ". row schreiben")** die Information aus.
        vi. Inkrementieren Sie den Inhalt der Variablen **actRowNo** mit actRowNo++.
30. Erzeugen Sie mit der Methode **new XMLOutputter(Format.getPrettyFormat())** ein neues Objekt vom Typ **XMLOutputter** und speichern Sie es in der Variablen **xmlOutputter** vom Typ **XmlOutputter**.

31. Erzeugen Sie mit der Methode **new FileOutputStream(xmlFile)** ein neues Objekt vom Typ **FileOutputStream** und speichern Sie es in der Variablen **fileOutputStream** vom Typ **FileOutputStream**.
32. Markieren Sie alle Zeilen im Gültigkeitsbereich der run-Methode. Öffnen Sie im markierten Bereich das Kontextmenü und wählen Sie den Eintrag **Surround With > Try/catch Block**. Um die markierten Zeilen wird ein try-catch-Block generiert.
33. Schreiben Sie mit der Methode **xmlOutputter.output(document, fileOutputStream)** die xml-Datei.
34. Fügen Sie den catch-Block der reklamierten IOException zum vorhandenen try-catch-Block hinzu.
35. Geben Sie mit der Methode **dbContext.out().println("xml file created")** die Information aus.
36. Geben Sie in den catch-Blöcken mit der Methode **dbContext.out().println(e.getMessage())** die passende Exception-Message aus.

DIE METHODE CREATEROWFIELDELEMENT(ROW ROW, STRING FIELDNAME)

1. Erzeugen Sie mit der Methode **new Element("Field")** ein Field-Element und speichern Sie es in der Variablen **field** vom Typ **Element**.
2. Definieren Sie mit der Methode **field.setAttribute("name", fieldname)** das Attribut **name** mit dem Inhalt der Variablen **fieldName**.
3. Definieren Sie mit der Methode **field.setAttribute("abasType", Product.Row.META.getField (fieldName).getErpType())** das Attribut **abasType** mit dem ermittelten ERPTyp.
4. Definieren Sie mit der Methode **field.setText(row.getString(fieldName))** den Wert des Field-Elements.
5. Geben Sie mit return field das Field-Element zurück.

DIE METHODE CREATEHEADERFIELDELEMENT(PRODUCT PRODUCT, STRING FIELDNAME)

1. Erzeugen Sie mit der Methode **new Element("Field")** ein Field-Element und speichern Sie es in der Variablen **field** vom Typ **Element**.
2. Definieren Sie mit der Methode **field.setAttribute("name", fieldname)** das Attribut **name** mit dem Inhalt der Variablen **fieldName**.
3. Definieren Sie mit der Methode **field.setAttribute("abasType", Product.META.getField (fieldName).getErpType())** das Attribut **abasType** mit dem ermittelten ERPTyp.
4. Definieren Sie mit der Methode **field.setText(product.getString(fieldName))** den Wert des Field-Elements.
5. Geben Sie mit return field das Field-Element zurück.

Resultat: Die Klasse **WriteProductsToXML** wurde erzeugt.

```
public class WriteProductsToXmlRowSelectionBuilder implements ContextRunnable {
    public static void main(String[] args) {
        WriteProductsToXmlRowSelectionBuilder writeProductsToXmlRowSelectionBuilder =
            new WriteProductsToXmlRowSelectionBuilder();
        DbContext dbContext = ContextHelper.createClientContext(host, port, mandant,
            password, appName));
        writeProductsToXmlRowSelectionBuilder.run(dbContext, args);
        dbContext.close();
    }
    @Override
    public int runFop(FOPSessionContext ctx, String[] args) throws FOPException {
        DbContext dbContext = ctx.getDbContext();
        run(dbContext, args);
        return 0;
    }
    private void run(DbContext dbContext, String[] args) {
        try {
            FileOutputStream fileOutputStream = null;
            String xmlFile = "xmlWrite/ProductsRowsRowSelection.xml";
            RowSelectionBuilder<Product, Row> rowSelectionBuilder =
                RowSelectionBuilder.create(Product.class, Product.Row.class);
            rowSelectionBuilder.addForHead(Conditions.between(Product.META.idno,
                "10053", "10060"));
            RowQuery<Product, Row> query = dbContext.createQuery(
                rowSelectionBuilder.build());
            FieldSet<FieldValueProvider> fieldSetHead = FieldSet.of("idno", "swd",
                "descrOperLang", "salesPrice", "DBNo", "grpNo");
            String[] headerFields = fieldSetHead.getFields();
            query.setFieldsForHead(fieldSetHead);
            FieldSet<FieldValueProvider> fieldSetRow =
                FieldSet.of("productListElem", "sparePartTab", "elemQty");
            //String[] rowFields = {"productListElem", "sparePartTab", "elemQty"};
            String[] rowFields = fieldSetRow.getFields();
            query.setFields(fieldSetRow);
            Element rootElement = new Element("ABASData");
            Document document = new Document(rootElement);
            Element recordSet = new Element("RecordSet");
            recordSet.setAttribute("action", "export");
            String dbNo = Integer.toString(query.execute().get(0).header().
                getDBNo().getCode());
            recordSet.setAttribute("database", dbNo);
            String grpNo = Integer.toString(query.execute().get(0).header().
                getGrpNo());
            recordSet.setAttribute("group", grpNo);
            rootElement.addContent(recordSet);
            String actIdno = "";
            int actRowNo = 0;
            Element record = null;
            for (Product.Row row : query) {
                if(!row.header().getIdno().equals(actIdno)){
                    Product product = row.header();
                    dbContext.out().println("head schreiben - " + row.header().
                        getIdno());
                    record = new Element("Record");
                    recordSet.addContent(record);
                    Element head = new Element("Head");
                    record.addContent(head);
                    // dbNo and grpNo will not be written
                    for (int i = 0; i < headerFields.length - 2; i++) {
                        Element field = createHeaderFieldElement(product,
                            headerFields[i]);
                        head.addContent(field);
```

```
                    }
                    actRowNo = 1;
                }
                actIdno = row.header().getIdno();
                SelectableObject selectableObject = row.getProductListElem();
                // Write products only
                if (selectableObject instanceof Product) {
                    Element rowElement = new Element("Row");
                    rowElement.setAttribute("number", Integer.toString(actRowNo));
                    for (int i = 0; i < rowFields.length; i++) {
                        Element field = createRowFieldElement(row, rowFields[i]);
                        rowElement.addContent(field);
                    }
                    record.addContent(rowElement);
                    dbContext.out().println(actRowNo + ". row schreiben");
                    actRowNo++;
                }
            }
            XMLOutputter xmlOutputter = new XMLOutputter(Format.getPrettyFormat());
            fileOutputStream = new FileOutputStream(xmlFile);
            xmlOutputter.output(document, fileOutputStream);
            dbContext.out().println("xml file created");
        } catch (FileNotFoundException e) {
            dbContext.out().println(e.getMessage());
        } catch (IOException e) {
            dbContext.out().println(e.getMessage());
        }
    }
    private Element createRowFieldElement(Row row, String fieldName) {
        Element field = new Element("Field");
        field.setAttribute("name", fieldName);
        field.setAttribute("abasType", Product.Row.META.getField(fieldName).
            getErpType());
        field.setText(row.getString(fieldName));
        return field;
    }
    private Element createHeaderFieldElement(Product product, String fieldName) {
        Element field = new Element("Field");
        field.setAttribute("name", fieldName);
        field.setAttribute("abasType", Product.META.getField(fieldName).getErpType());
        field.setText(product.getString(fieldName));
        return field;
    }
}
```

LÖSUNG ZU ÜBUNG 2: XML-DATEI MIT ALLEN KUNDENDATENSÄTZEN SCHREIBEN, DEREN FELD TELEFONNUMMER (PHONENO) LEER IST.

Führen Sie folgende Schritte aus:

1. Kopieren Sie das AJO-Programm **WriteCutomerToXML**.
2. Speichern Sie die Kopie unter **WriteCustomerWithoutPhoneNoToXML**.
3. Passen Sie die Selektionkriterien an,
   z.B. selectionBuilder.add(Conditions.empty(Customer.META.phoneNo));
4. Passen Sie das FieldSet an,
   z.B. fieldSet = FieldSet.of("idno", "swd", "phoneNo", "DBNo", "grpNo");

Resultat: Sie habe das AJO-Programm **WriteCustomerWithoutPhoneNoToXML** so geändert, dass eine xml-Datei mit Kunde ohne Telefonnummer geschrieben wird.

```
public class WriteCustomersWithoutPhoneNoToXml implements ContextRunnable {
    public static void main(String[] args) {
        WriteCustomersWithoutPhoneNoToXml writeCustomersWithoutPhoneNoToXml =
            new WriteCustomersWithoutPhoneNoToXml();
        DbContext dbContext = ContextHelper.createClientContext(host, port, mandant,
            password, appName);
        writeCustomersWithoutPhoneNoToXml.run(dbContext, args);
        dbContext.close();
    }
    @Override
    public int runFop(FOPSessionContext ctx, String[] args) throws FOPException {
        DbContext dbContext = ctx.getDbContext();
        run(dbContext, args);
        return 0;
    }
    private void run(DbContext dbContext, String[] args) {
        FileOutputStream fileOutputStream = null;
        String xmlFile ="xmlWrite/CustomersWithoutPhoneNo.xml";
        SelectionBuilder<Customer> selectionBuilder = SelectionBuilder.
            create(Customer.class);
        selectionBuilder.add(Conditions.empty(Customer.META.phoneNo));
        Query<Customer> query = dbContext.createQuery(selectionBuilder.build());
        FieldSet<FieldValueProvider> fieldSet = FieldSet.of("idno", "swd", "phoneNo",
            "DBNo", "grpNo");
        query.setFields(fieldSet);
        String[] fields = fieldSet.getFields();
        Element rootElement = new Element("ABASData");
        Document document = new Document(rootElement);
        Element recordSet = new Element("RecordSet");
        recordSet.setAttribute("action", "update");
        String dbNo = Integer.toString(query.execute().get(0).getDBNo().getCode());
        recordSet.setAttribute("database", dbNo);
        String grpNo = Integer.toString(query.execute().get(0).getGrpNo());
        recordSet.setAttribute("group", grpNo);
        rootElement.addContent(recordSet);
        try {
            for (Customer customer : query) {
                Element record = new Element("Record");
                record.setAttribute("idno", customer.getIdno());
                recordSet.addContent(record);
                Element head = new Element("Head");
                record.addContent(head);
                // Do not write DBNo and GrpNo into xml-File ->
                // see declaration FieldSet
                for (int i = 0; i < fields.length -2; i++) {
```

```
                    Element field = createFieldElement(customer, fields[i]);
                    head.addContent(field);
                }
            }
            XMLOutputter xmlOutputter = new XMLOutputter(Format.getPrettyFormat());
            fileOutputStream = new FileOutputStream(xmlFile);
            xmlOutputter.output(document, fileOutputStream);
            dbContext.out().println("xml file created");
        } catch (FileNotFoundException e) {
            dbContext.out().println(e.getMessage());
        } catch (IOException e) {
            dbContext.out().println(e.getMessage());
        } finally {
            if (fileOutputStream != null) {
                try {
                    fileOutputStream.close();
                } catch (IOException e) {
                    dbContext.out().println("Problem: " + e.getMessage());
                }
            }
        }
    }
    private Element createFieldElement(Customer customer, String fieldName) {
        Element field = new Element("Field");
        field.setAttribute("name", fieldName);
        field.setAttribute("abasType", Customer.META.getField(fieldName).
            getErpType());
        field.setText(customer.getString(fieldName));
        return field;
    }
}
```

NOTIZEN

2 LÖSUNGEN ZU DATENIMPORT AUS XML

LÖSUNG ZU ÜBUNG 1: KUNDENDATENSÄTZE AUS XML-DATEI LESEN UND IM MANDANT ANLEGEN

Führen Sie folgende Schritte aus:

1. Erzeugen Sie eine AJO-Anwendung **ReadXmlCreateCustomer**, die im Server- und im Client-Kontext ausgeführt werden kann.
2. Bauen Sie die gemeinsame run-Methode ein und führen Sie dort folgende Schritte aus:
3. Definieren Sie die Variable **xmlFile** vom Typ **String** und initialisieren Sie die Variable mit **"xmlRead/NewCustomers.xml"**.
4. Definieren Sie die Variable **customerEditor** vom Typ **CustomerEditor** und initialisieren Sie die Variable mit **null**.
5. Erzeugen Sie mit dem Konstruktor **new SAXBuilder()** ein neues SAXBuilder-Objekt und speichern Sie das Objekt in der Variablen **saxBuilder** vom Typ **SAXBuilder**.
6. Erzeugen Sie mit der Methode **saxBuilder.build(xmlFile)** ein neues Document-Objekt und speichern Sie das Objekt in der Variablen **document** vom Typ **Document**.
   Lassen Sie den try-catch-Block generieren und tragen Sie im try-Block folgende Schritte ein:
   a. Ermitteln Sie mithilfe der Methode **document.getRootElement()** das **RootElement** und speichern Sie es in der Variablen **rootElement** vom Typ **Element**.
   b. Ermitteln Sie mithilfe der Methode **rootElement.getName()** den Namen des rootElements und speichern Sie diesen in der Variablen **name** vom Typ **String**.
   c. Prüfen Sie, ob der Inhalt der Variablen **name** identisch mit "ABASData" ist.
   d. Falls nein, geben Sie die Meldung **"xml file: no abas data format"** aus und beenden Sie das AJO-Programm.
   e. Ansonsten ermitteln Sie mit der Methode **rootElement.getChild("RecordSet")** das RecordSet des rootElements und speichern Sie es in der Variablen **recordSet** vom Typ **Element**.
   f. Ermitteln Sie mit der Methode **recordSet.getAttributeValue("action")** den Wert des Attributes und speichern Sie diesen in der Variablen **action** vom Typ **String**.
   g. Geben Sie den Inhalt der Variablen **action** aus.
   h. Ermitteln Sie mit der Methode **recordSet.getChildren()** die Datensätze (records) und speichern Sie die Datensätze in der Variablen **records** vom Typ **List<Element>**.
   i. Erzeugen Sie eine **for-each**-Schleife über alle **records**. Das aktuelle Element steht über die Variable **record** zur Verfügung. Führen Sie folgende Schritte je Schleifendurchlauf aus:
      i. Erzeugen Sie mit der Methode **dbContext.newObjekt(CustomerEditor.class)** ein Editor-Objekt und speichern Sie das Objekt in der Variablen **customerEditor** vom Typ **CustomerEditor**.

iii. Ermitteln Sie mit der Methode **record.getChild("Head")** das Head-Element und speichern Sie das Element in der Variablen **head** vom Typ **Element** ab.
iv. Ermitteln Sie mit der Methode **head.getChildren()** die Felder (fields) und speichern Sie diese in der Variablen **fields** vom Typ **List<Element>**.
v. Erzeugen Sie eine **for-each**-Schleife über alle **fields**. Das aktuelle Element steht über die Variable **field** zur Verfügung. Führen Sie folgende Aktion je Schleifendurchlauf aus:
    1. Ermitteln Sie mit der Methode **field.getAttributeValue("name")** den Feldnamen und speichern Sie diesen in der Variablen **fieldName**.
    2. Ermitteln Sie mit der Methode **field.getValue()** den Feldwert und speichern Sie diesen in der Variablen **fieldValue**.
    3. Geben Sie Feldname und Feldwert aus.
    4. Füllen Sie mit der Methode **customerEditor.setString(fieldName, fieldValue.trim())** das Feld aus.
vi. Speichern Sie mit der Methode **customerEditor.commit()** den neuen Datensatz.
vii. Geben Sie die Identummer und das Suchwort des neu erzeugten Kundendatensatzes aus.

7. Geben Sie im jeweiligen **catch**-Block die zugehörige Message aus.
8. Bauen Sie den **finally**-Block ein.
9. Prüfen Sie, ob das Objekt **customerEditor** vorhanden und aktiv ist.
10. Falls ja, beenden Sie den Editor mit der Methode **customerEditor.abort()**.

ReadXmlCreateCustomers.java

```
public class ReadXmlCreateCustomers implements ContextRunnable {
    public static void main(String[] args) {
        ReadXmlCreateCustomers readXmlCreateCustomers = new ReadXmlCreateCustomers();
        DbContext dbContext = ContextHelper.createClientContext("oxford",    6550, "schulvajo",
"sy", readXmlCreateCustomers.getClass().getSimpleName());
        readXmlCreateCustomers.run(dbContext, args);
        dbContext.close();
    }
    @Override
    public int runFop(FOPSessionContext ctx, String[] args) throws FOPException {
        DbContext dbContext = ctx.getDbContext();
        run(dbContext, args);
        return 0;
    }
    private void run(DbContext dbContext, String[] args) {
        String xmlFile = "xmlRead/NewCustomers.xml";
        CustomerEditor customerEditor = null;
        try {
            SAXBuilder saxBuilder = new SAXBuilder();
            Document document = saxBuilder.build(xmlFile);
            Element rootElement = document.getRootElement();
            String name = rootElement.getName();
            if (!name.equals("ABASData")) {
                dbContext.out().println("xml file: no abas data format");
                return;
            }
            dbContext.out().println("RootElement: " + name);
            // Only one RecordSet element is existing in xml file
            Element recordSet = rootElement.getChild("RecordSet");
            String action = recordSet.getAttributeValue("action");
            dbContext.out().println("Action: " + action);
            List<Element> records = recordSet.getChildren();
            for (Element record : records) {
                // Only Head elements are existing in xml file
                customerEditor = dbContext.newObject(CustomerEditor.class);
```

```
                Element head = record.getChild("Head");
                List<Element> fields = head.getChildren();
                for (Element field : fields) {
                    String fieldName = field.getAttributeValue("name");
                    String fieldValue = field.getValue();
                    dbContext.out().println(fieldName + " -> " + fieldValue);
                    customerEditor.setString(fieldName, fieldValue.trim());
                }
                //customerEditor.abort();
                customerEditor.commit();
                Customer customer = customerEditor.objectId();
                dbContext.out().println(customer.getIdno() + " - " + customer.getSwd());
                dbContext.out().println(" ----------------> ");
            }
        } catch (JDOMException e) {
            dbContext.out().println(e.getMessage());
        } catch (IOException e) {
            dbContext.out().println(e.getMessage());
        } finally {
            if (customerEditor != null & customerEditor.active()) {
                customerEditor.abort();
            }
        }
    }
}
```

NOTIZEN

LÖSUNG ZU ÜBUNG 2: KUNDENDATENSÄTZE AUS XML-DATEI LESEN UND DIE FEHLENDE TELEFONNUMMER EINTRAGEN

Führen Sie folgende Schritte aus:

1. Erzeugen Sie eine AJO-Anwendung **ReadXmlUpdateCustomerPhoneNo**, die im Server- und im Client-Kontext ausgeführt werden kann.
2. Bauen Sie die gemeinsame run-Methode ein und führen Sie dort folgende Schritte aus:
3. Definieren Sie die Variable **xmlFile** vom Typ **String** und initialisieren Sie die Variable mit " **xmlRead/UpdateCustomerPhoneNo.xml**".
4. Definieren Sie die Variable **customerEditor** vom Typ **CustomerEditor** und initialsieren Sie die Variable mit **null**.
5. Erzeugen Sie mit dem Konstruktor **new SAXBuilder()** ein neues SAXBuilder-Objekt und speichern Sie das Objekt in der Variablen **saxBuilder** vom Typ **SAXBuilder**.
6. Erzeugen Sie mithilfe der Methode **saxBuilder.build(xmlFile)** ein neues Document-Objekt und speichern Sie das Objekt in der Variablen **document** vom Typ **Document**.
   Lassen Sie die **try-catch**-Block generieren und tragen Sie folgende Schritte ein:
   a. Ermitteln Sie mithilfe der Methode **document.getRootElement()** das RootElement und speichern Sie das Element in der Variablen **rootElement** vom Typ **Element**.
   b. Ermitteln Sie mithilfe der Methode **rootElement.getName()** den Namen des rootElements und speichern Sie den Namen in der Variablen **name** vom Typ **String**.
   c. Prüfen Sie, ob der Inhalt der Variablen **name** identisch ist mit "ABASData".
   d. Falls **nein**, geben Sie die Meldung **"xml file: no abas data format"** aus und beenden Sie das AJO-Programm.
   e. **Ansonsten** ermitteln Sie mit der Methode **rootElement.getChild("RecordSet")** das RecordSet des rootElements und speichern Sie es in der Variablen **recordSet** vom Typ **Element**.
   f. Ermitteln Sie mit der Methode **recordSet.getAttributeValue("action")** den Wert des Attributes und speichern Sie den Wert in der Variablen **action** vom Typ **String**.
   g. Prüfen Sie, ob der Inhalt der Variablen **action** identisch ist mit "**update**".
   h. Falls **nein**, geben Sie die Meldung **"appropriate for update only"** aus und beenden Sie das AJO-Programm.
   i. **Ansonsten** ermitteln Sie mit der Methode **recordSet.getChildren()** die zu ändernden Datensätze (records) und speichern Sie diese in der Variablen **records** vom Typ **List<Element>**.
   j. Erzeugen Sie eine **for-each**-Schleife über alle **records**. Das aktuelle Element steht über die Variable **record** zur Verfügung. Führen Sie folgende Schritte je Schleifendurchlauf (record) aus:
      i. Ermitteln Sie mit der Methode **record.getAttribute(idno")** das Attribut mit dem Bezeichner **idno** und speichern Sie das Attribut in der Variablen **attribute** vom Typ **Attribute**.
      ii. Ermitteln Sie mit der Methode **attribute.getValue()** den Wert des Attributs und speichern Sie den Wert in der Variablen **idno**.
      iii. Geben Sie die den Inhalt der Variablen **idno** aus.
      iv. Ermitteln Sie mit der Methode **getSelectedCusomer(dbContext, idno)** den zu ändernden Kundendatensatz und speichern Sie diesen in der Variablen **customer** vom Typ **Customer**.
      v. Prüfen Sie, ob ein Kundendatensatz verfügbar (!= null) ist. Falls **ja**, führen Sie folgende Schritte aus:
         1. Erzeugen Sie mit der Methode **customer.createEditor()** den zugehörigen Editor und speichern Sie diesen in der Variablen **customerEditor** vom Typ **CustomerEditor**.

2. Öffnen Sie mit der Methode **customerEditor.open(EditorAction.UPDATE)** den Editor zum Bearbeiten des Datensatzes.
3. Ermitteln Sie mit der Methode **record.getChild("Head")** das Head-Element und speichern Sie das Element in der Variablen **head** vom Typ **Element**.
4. Ermitteln Sie mit der Methode **head.getChildren()** die Felder des Head-Elements und speichern Sie diese in der Variablen **fields** vom Typ **List<Element>**.
5. Erzeugen Sie eine **for-each**-Schleife über **fields**. Das aktuelle Element steht über die Variable **field** zur Verfügung. Führen Sie folgende Schritte je Schleifendurchlauf aus:
   a. Ermitteln Sie mit der Methode **field.getAttribute("name")** das Attribut und speichern Sie es in der Variablen **attribute2** vom Typ **Attribute**.
   b. Ermitteln Sie mit der Methode **attribute2.getValue()** den Feldnamen und speichern Sie diesen in der Variablen **fieldName** vom Typ **String**.
   c. Prüfen Sie mit der Methode **fieldName.equals("phoneNo")**, ob der Inhalt der Variablen **fieldName** identisch ist mit **phoneNo**.
   d. Falls **ja**, führen Sie folgende Schritte aus:
      i. Ermitteln Sie mit der Methode **field.getTextTrim()** den Feldwert ohne führende /nachfolgende Leerzeichen und speichern Sie diesen in der Variablen **fieldValue** vom Typ **String**.
      ii. Schreiben Sie mit der Methode **customerEditor.setString(fieldName, fieldValue)** den Inhalt des Feldes **phoneNo**.
      iii. Geben Sie Feldname und Feldwert aus.
6. Speichern Sie den geänderten Datensatz mit der Methode **customerEditor.commit()**.

k. Geben Sie die Info **end of program** aus.

7. Geben Sie im jeweiligen **catch**-Block die zugehörige Message aus.
8. Bauen Sie den **finally**-Block ein.
9. Prüfen Sie ob das Objekt **customerEditor** vorhanden und aktiv ist.
10. Falls ja, beenden Sie den Editor mit der Methode **customerEditor.abort()**.

ReadXmlUpdateCustomers.java

```
public class ReadXmlUpdateCustomerPhoneNo implements ContextRunnable {
    public static void main(String[] args) {
        ReadXmlUpdateCustomerPhoneNo readXmlEditCustomerPhoneNo = new
ReadXmlUpdateCustomerPhoneNo();
        DbContext dbContext = ContextHelper.createClientContext("oxford",    6550, "schulvajo",
"sy", readXmlEditCustomerPhoneNo.getClass().getSimpleName());
        readXmlEditCustomerPhoneNo.run(dbContext, args);
        dbContext.close();
    }
    @Override
    public int runFop(FOPSessionContext ctx, String[] args) throws FOPException {
        DbContext dbContext = ctx.getDbContext();
        run(dbContext, args);
        return 0;
    }
    private void run(DbContext dbContext, String[] args){
        String xmlFile = "xmlRead/UpdateCustomerPhoneNo.xml";
        CustomerEditor customerEditor = null;
        try {
            SAXBuilder saxBuilder = new SAXBuilder();
            Document document = saxBuilder.build(xmlFile);
            Element rootElement = document.getRootElement();
            String name = rootElement.getName();
            if (!name.equals("ABASData")) {
```

```
                dbContext.out().println("xml file: no abas data format");
                return;
            }
            Element recordSet = rootElement.getChild("RecordSet");
            String action = recordSet.getAttributeValue("action");
            if (!action.equals("update")) {
                dbContext.out().println("appropriate for update only");
                return;
            }
            List<Element> records = recordSet.getChildren();
            for (Element record : records) {
                Attribute attribute = record.getAttribute("idno");
                String idno = attribute.getValue();
                dbContext.out().println("record: " + idno);
                // select customer record
                Customer customer = getSelectedCustomer(dbContext, idno);
                if (customer != null) {
                    customerEditor = customer.createEditor();
                    customerEditor.open(EditorAction.UPDATE);
                    Element head = record.getChild("Head");
                    List<Element> fields = head.getChildren();
                    for (Element field : fields) {
                        Attribute attribute2 = field.getAttribute("name");
                        String fieldName = attribute2.getValue();
                        if (fieldName.equals("phoneNo")) {
                            String fieldValue = field.getTextTrim();
                            dbContext.out().println("Field: " + fieldName + " Value: " +
fieldValue );
                            customerEditor.setString(fieldName,
fieldValue);
                        }
                    }
                    customerEditor.commit();
                    //customerEditor.abort();
                }
            }
            dbContext.out().println("end of program");
        } catch (JDOMException e) {
            dbContext.out().println(e.getMessage());
        } catch (IOException e) {
            dbContext.out().println(e.getMessage());
        } catch (CommandException e) {
            dbContext.out().println(e.getMessage());
        } finally {
            if (customerEditor != null & customerEditor.active()) {
                customerEditor.abort();
            }
        }
    }
    private Customer getSelectedCustomer(DbContext dbContext, String idno) {
        SelectionBuilder<Customer> selectionBuilder = SelectionBuilder.create(Customer.class);
        selectionBuilder.add(Conditions.eq(Customer.META.idno, idno));
        return QueryUtil.getFirst(dbContext, selectionBuilder.build());
    }
}
```

NOTIZEN

3 LÖSUNGEN ZU XML-DATEIEN LESEN

LÖSUNG ZU ÜBUNG 1: KUNDEN AUS EINER XML-DATEI IMPORTIEREN

NEUES PAKET ERSTELLEN

Führen Sie folgende Schritte aus:

1. Kicken Sie mit der rechten Maustaste auf Ihr Projekt.
   Ein Kontextmenü öffnet sich.
2. Wählen Sie den Menüpunkt **New > Package**.
   Ein neues Paket wird erstellt.
3. Benennen Sie das neue Paket **de.abas.training.record.objectsfromtxt**.

Resultat: Sie haben das neue Paket **de.abas.training.record.objectsfromtxt** erstellt.

NEUE KLASSE ERSTELLEN UND ANPASSEN

Führen Sie folgende Schritte aus:

1. Klicken Sie mit der rechten Maustaste auf das Paket **de.abas.documentation.advanced.record.objectsfromxml**.
   Ein Kontextmenü öffnet sich.
2. Wählen Sie den Menüpunkt **New > Class**.
   Das Fenster **New Java Class** öffnet sich.
3. Schreiben Sie in das Feld **Name** die Bezeichnung **CreateNewProductsFromTXT**.
4. Schreiben Sie in das Feld **Superclass** den vollqualifizierten Namen Ihrer Klasse **AbstractAjoAccess**. Alternativ können Sie auf den Button **Browse...** klicken, um die Klasse auszuwählen.
5. Markieren Sie das Kontrollfeld **public static void main(String[] args)**, um die Methodenvorlage für die main-Methode bereits in die Klasse zu generieren und erzeugen Sie dort eine Instanz ihrer Klasse, die die Methode **runClientProgram** aufruft.
6. Fügen Sie folgenden Code in die von der Klasse generierte run-Methode ein:

```
@SuppressWarnings("unchecked")
@Override
public void run() {
    ProductEditor productEditor = null;
    // path of XML file
    String xmlFile =
            "src/de/abas/documentation/advanced/record/"
                    + "objectsfromxml/New Products.xml";
    try {
        // instantiates SAXBuilder
        SAXBuilder saxBuilder = new SAXBuilder();
        // parses the xml file
        Document document = saxBuilder.build(xmlFile);
        // the <ABASData> tag
        Element rootElement = document.getRootElement();
        // the <RecordSet> tag
        List<Element> recordSet = rootElement.getChildren();
        // the <Record> tags
        List<Element> records = recordSet.get(0).getChildren();
        for (Element record : records) {
```

```
            productEditor = getDbContext().newObject(ProductEditor.class);
            // the <Head> tag
            Element head = record.getChild("Head");
            // the <Field> tags
            List<Element> fields = head.getChildren();
            for (Element field : fields) {
                // sets head fields for new product
                EditableFieldMeta fieldMeta =
                        productEditor.getFieldMeta(field
                                .getAttributeValue("name"));
                fieldMeta.setValue(field.getValue());
            }
            // the <Row> tags
            List<Element> rows = record.getChildren("Row");
            for (Element row : rows) {
                Row appendRow = productEditor.table().appendRow();
                // the <Field> tags of the rows
                List<Element> rowFields = row.getChildren();
                for (Element rowField : rowFields) {
                    // set row fields for new product
                    EditableFieldMeta fieldMeta =
                            appendRow.getFieldMeta(rowField
                                    .getAttributeValue("name"));
                    fieldMeta.setValue(rowField.getValue());
                }
            }
            // saves the new product
            productEditor.commit();
            // displays success message
            getDbContext().out().println(
                    "The following customer was successfully created: "
                            + productEditor.objectId());
        }
    }
    catch (JDOMException e) {
        getDbContext().out().println(e.getMessage());
    }
    catch (IOException e) {
        getDbContext().out().println(e.getMessage());
    }
    finally {
        if (productEditor != null) {
            if (productEditor.active()) {
                productEditor.abort();
            }
        }
    }
}
```

Resultat: Ihr Programm sieht nun folgendermaßen aus:

```
package de.abas.documentation.advanced.record.objectsfromxml;
import java.io.IOException;
import java.util.List;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;
import de.abas.erp.db.field.editable.EditableFieldMeta;
import de.abas.erp.db.schema.part.ProductEditor;
import de.abas.erp.db.schema.part.ProductEditor.Row;
import de.abas.training.common.AbstractAjoAccess;
public class CreateNewProductsFromXML extends AbstractAjoAccess {
    public static void main(String[] args) {
        CreateNewProductsFromXML createNewProductsFromXML =
                new CreateNewProductsFromXML();
        createNewProductsFromXML.runClientProgram();
    }
    @SuppressWarnings("unchecked")
    @Override
    public void run() {
        ProductEditor productEditor = null;
        // path of XML file
        String xmlFile =
                "src/de/abas/documentation/advanced/record/"
                        + "objectsfromxml/New Products.xml";
        try {
            // instantiates SAXBuilder
            SAXBuilder saxBuilder = new SAXBuilder();
            // parses the xml file
            Document document = saxBuilder.build(xmlFile);
            // the <ABASData> tag
            Element rootElement = document.getRootElement();
            // the <RecordSet> tag
            List<Element> recordSet = rootElement.getChildren();
            // the <Record> tags
            List<Element> records = recordSet.get(0).getChildren();
            for (Element record : records) {
                productEditor = getDbContext().newObject(ProductEditor.class);
                // the <Head> tag
                Element head = record.getChild("Head");
                // the <Field> tags
                List<Element> fields = head.getChildren();
                for (Element field : fields) {
                    // sets head fields for new product
                    EditableFieldMeta fieldMeta =
                            productEditor.getFieldMeta(field
                                    .getAttributeValue("name"));
                    fieldMeta.setValue(field.getValue());
                }
                // the <Row> tags
                List<Element> rows = record.getChildren("Row");
                for (Element row : rows) {
                    Row appendRow = productEditor.table().appendRow();
                    // the <Field> tags of the rows
                    List<Element> rowFields = row.getChildren();
                    for (Element rowField : rowFields) {
                        // set row fields for new product
```

```
                        EditableFieldMeta fieldMeta =
                                appendRow.getFieldMeta(rowField
                                        .getAttributeValue("name"));
                        fieldMeta.setValue(rowField.getValue());
                    }
                }
                // saves the new product
                productEditor.commit();
                // displays success message
                getDbContext().out().println(
                        "The following customer was successfully created: "
                                + productEditor.objectId());
            }
        }
        catch (JDOMException e) {
            getDbContext().out().println(e.getMessage());
        }
        catch (IOException e) {
            getDbContext().out().println(e.getMessage());
        }
        finally {
            if (productEditor != null) {
                if (productEditor.active()) {
                    productEditor.abort();
                }
            }
        }
    }
}
```

4 LÖSUNG ZU LOG4J

LÖSUNG: AJO-CLIENT-ANWENDUNG MIT LOG4J-LOGGING IMPLEMENTIEREN

Erzeugen Sie die Datei **log4j.properties** im src-Folder. Die Log-Messages sollen nur auf der Konsole ausgegeben werden. Verwenden Sie für das Logger-System dasselbe Package wie die Klasse, deren Messages geloggt werden sollen, z.B. **de.abas.training.advanced.log4j**

LOG4J.PROPERTIES ERZEUGEN

Führen Sie folgende Schritte aus:

1. Erzeugen Sie in [AJO-Project]**/**src die Properties-Datei **log4j.properties**.
2. Definieren Sie einen Logger für die Konsolenausgabe. Der Log-Level ist INFO.
   Hinweis: LogLevel INFO gibt INFO, WARN, ERROR und FATAL log-Messages aus.
   log4j.logger.de.abas.training.advanced.log4j=Info, de.abas.training.advanced.log4j.Console
3. Definieren Sie den Appender für die Konsolenausgabe:
   log4j.appender.de.abas.training.advanced.log4j.Console=org.apache.log4j.ConsoleAppender
4. Definieren Sie das Layout für die Konsolenausgabe:
   log4j.appender.de.abas.training.advanced.log4j.Console.layout=org.apache.log4j.PatternLayout
5. Definieren Sie das PatternLayout für die Konsolenausgabe mit:
   log4j.appender.de.abas.training.advanced.log4j.Console.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %C{7}:%L - %m%n

Resultat: Die Properties-Ddatei **log4j.properties** wurde erzeugt. Das Logging wurde für die Konsolenausgabe konfiguriert.

log4j.properties

```
define logger
log4j.logger.de.abas.training.advanced.log4j=Info, de.abas.training.advanced.log4j.Console
# console appender
log4j.appender.de.abas.training.advanced.log4j.Console=org.apache.log4j.ConsoleAppender
log4j.appender.de.abas.training.advanced.log4j.Console.layout=org.apache.log4j.PatternLayout
log4j.appender.de.abas.training.advanced.log4j.Console.layout.ConversionPattern=%d{yyyy-MM-dd
HH:mm:ss} %-5p %C{7}:%L - %m%n
```

LOG-MESSAGES IN CREATENEWPRODUCTSFROMXMLTRANSACTIONLOG4J IMPLEMENTIEREN

In diesem Teil wird in der AJO-Anwendung **CreateNewProductsFromXmlTransactionLog4j** die Standard-Konsolenausgabe durch das Logging mit log4j ersetzt.

Um log4j in Ihrer Klasse **CreateNewProductsFromXMLTransactionLog4j** nutzen zu können, müssen Sie eine Instanz der Klasse **Logger** im Paket **org.apache.log4j** erzeugen. Der Bezeichner des Loggersystems ist identisch mit dem Klassennamen.

Führen Sie folgende Schritte aus:

1. Erzeugen Sie ein Logger-Objekt und speichern Sie es in der Variablen **logger** der Klasse.
   private static final Logger logger = Logger.getLogger (CreateNewProductsFromXmlTransactionLog4j.class);
2. Ersetzen Sie alle Standard-Konsolenausgaben in der run-Methode durch log-Ausgaben. Verwenden Sie die Methode des passenden LogLevels.

Resultat: Die Standard-Konsolenausgabe wurde durch die log-Ausgabe ersetzt.

log4j.properties

```
public class CreateNewProductsFromXmlTransactionLog4j implements ContextRunnable {
    private String xmlFile;
    private boolean rollback;
    private static final Logger logger = Logger.getLogger
(CreateNewProductsFromXmlTransactionLog4j.class);
    private ProductEditor productEditor;
    public static void main(String[] args) {
        CreateNewProductsFromXmlTransactionLog4j createNewProductsFromXmlTransactionLog4j = new
CreateNewProductsFromXmlTransactionLog4j();
        DbContext dbContext = ContextHelper.createClientContext("oxford",    6550, "schulvajo",
"sy", createNewProductsFromXmlTransactionLog4j.getClass().getSimpleName());
        createNewProductsFromXmlTransactionLog4j.xmlFile = "xml/products.xml";
        logger.info("xml file path defined");
        createNewProductsFromXmlTransactionLog4j.run(dbContext, args);
        dbContext.close();
    }
    @Override
    public int runFop(FOPSessionContext ctx, String[] args) throws FOPException {
        DbContext dbContext = ctx.getDbContext();
        this.xmlFile = "java/tmp/products.xml";
        int status = run(dbContext, args);
        return status;
    }
    private int run(DbContext dbContext, String[] args) {
        if (!existsXmlFile(dbContext)) {
            logger.error("xml-file " + xmlFile + " does not exist");
            return 1;
        }
        try {
            Element rootElement = new SAXBuilder().build(this.xmlFile).getRootElement();
            if (isValidXml(rootElement)) {
                logger.info("import started");
                this.rollback = false;
                displayRootElementName(rootElement, dbContext);
                displayRecordSetAttributes(rootElement, dbContext);
                Transaction transaction = beginTransaction(dbContext);
                logger.info("begin transaction");
```

```
                createProductsIfNotExisting(rootElement, dbContext);
            }else {
                logger.error("is not valid abas xml formatting");
            }
            logger.info("end of program");
        } catch (JDOMException e) {
            logger.error(e.getMessage());
            return 1;
        } catch (IOException e) {
            logger.error(e.getMessage());
            return 1;
        } finally {
            closeProductEditor();
        }
        return 0;
    }
    private void createProductsIfNotExisting(Element rootElement, DbContext dbContext) {
        Element recordSet = rootElement.getChild("recordSet");
        List<Element> records = recordSet.getChildren();
        for (Element record : records) {
            String swdValue = record.getAttributeValue("swd");
            checkWetherProductExists(swdValue, dbContext);
            if (rollback) {
                break;
            }else {
                createProduct(record, dbContext);
                productEditor.abort();
            }
        }
    }
    private void createProduct(Element record, DbContext dbContext) {
        productEditor = dbContext.newObject(ProductEditor.class);
        List<Element> headerAndRows = record.getChildren();
        for (Element headerOrRow : headerAndRows) {
            if (headerOrRow.getName().equals("header")) {
                writeProductHeaderFields(headerOrRow, dbContext);
            }else if (headerOrRow.getName().equals("row")) {
                writeProductRowFileds(headerOrRow, dbContext);
            }
        }
    }
    private void writeProductRowFileds(Element row, DbContext dbContext) {
        logger.info("writing row");
        List<Element> fields = row.getChildren();
        Row appendRow = productEditor.table().appendRow();
        for (Element field : fields) {
            String name = field.getAttributeValue("name");
            String value = field.getValue();
            logger.info("row field: " + name + " -> " + value);
            appendRow.setString(name, value);
        }
    }
    private void writeProductHeaderFields(Element header, DbContext dbContext) {
        logger.info("writing header");
        List<Element> fields = header.getChildren();
        for (Element field : fields) {
            String name = field.getAttributeValue("name");
            String value = field.getValue();
            logger.info("header field: " + name + " -> " + value);
            productEditor.setString(name, value);
        }
    }
    private void checkWetherProductExists(String swd, DbContext dbContext) {
```

```
        SelectionBuilder<Product> selectionBuilder = SelectionBuilder.create(Product.class);
        selectionBuilder.add(Conditions.eq(Product.META.swd, swd));
        Product product = QueryUtil.getFirst(dbContext, selectionBuilder.build());
        if (product != null) {
            logger.error("Product: " + swd + " already exists");
            rollback = true;
        }
    }
    private Transaction beginTransaction(DbContext dbContext) {
        Transaction transaction = dbContext.getTransaction();
        transaction.begin();
        return transaction;
    }
    private boolean existsXmlFile(DbContext dbContext) {
        File file = new File(xmlFile);
        boolean status = false;
        if (file.exists()) {
            status = true;
        }
        return status;
    }
    private void displayRecordSetAttributes(Element rootElement, DbContext dbContext) {
        Element recordSet = rootElement.getChild("recordSet");
        List<Attribute> recordSetAttributes = recordSet.getAttributes();
        for (Attribute attribute : recordSetAttributes) {
            logger.info("attribute: " + attribute.getName() + " -> " + attribute.getValue());
        }
    }
    private void displayRootElementName(Element rootElement, DbContext dbContext) {
        logger.info("rootElement: " +  rootElement.getName());
    }
    private boolean isValidXml(Element rootElement) {
        return rootElement.getName().equals("abasData");
    }
    private void closeProductEditor() {
        if(productEditor != null){
            if (productEditor.active()) {
                productEditor.abort();
            }
        }
    }
}
```

CREATENEWPRODUCTSFROMXMLTRANSACTIONLOG4J AUSFÜHREN

Beim Ausführen von CreateNewProductsFromXmlTransactionLog4j werden die Log-Messages einheitlich formatiert in der Konsole ausgegeben.

Wenn Sie die Logging-Messsages sowohl in der Konsole als auch in eine Datei ausgeben möchten, dann müssen Sie die zugehörigen Logger in einer Zeile der Datei **log4j.properties** definieren, z.B. log4j.logger.de.abas.training.advanced.log4j=Info, de.abas.training.advanced.log4j.Console, **de.abas.training.advanced.log4j.RollingFile**

LOG4J.PROPERTIES FÜR KONSOLEN- UND DATEI-AUSGABE

In diesem Teil wird die Datei **log4j.properties** im src-Folder erweitert. Die Log-Messages sollen sowohl auf der Konsole als auch in eine LogDatei ausgegeben werden. Verwenden Sie für das Logger-System dasselbe Package wie die Klasse, deren Messages geloggt werden sollen, z.B. de.abas.training.advanced.log4j.

LOG4J.PROPERTIES ERZEUGEN

Führen Sie folgende Schritte aus:

1. Erweitern Sie in [AJO-Project]**/**src die Properties-Datei **log4j.properties**.
2. Definieren Sie einen zusätzlichen Logger für die Datei-Ausgabe. Der Log-Level ist INFO.
   Um die Log-Messages sowohl in der Konsole als auch in eine LogDatei auszugeben, müssen Sie diese in ein und derselben Zeile definieren. Sie erhalten dadurch einen Logger mit zwei unterschiedlichen Loggersystemen.
   Hinweis: LogLevel INFO gibt INFO, WARN, ERROR und FATAL log-Messages aus.
   log4j.logger.de.abas.training.advanced.log4j=Info, de.abas.training.advanced.log4j.Console**, de.abas.training.advanced.log4j.RollingFile**
3. Definieren Sie einen Appender (Loggersystem) für die LogDatei-Ausgabe:
   log4j.appender.de.abas.training.advanced.log4j.RollingFile=org.apache.log4j.RollingFileAppender
4. Definieren Sie den Speicherort der LogDatei**:**
   log4j.appender.de.abas.training.advanced.log4j.RollingFile.File=log/log4j.log
5. Definieren Sie die maximale Größe der LogDatei:
   log4j.appender.de.abas.training.advanced.log4j.RollingFile.MaxFileSize=5MB
6. Definieren Sie die maximale Anzahl der LogDateien:
   log4j.appender.de.abas.training.advanced.log4j.RollingFile.MaxBackupIndex=10
7. Definieren Sie das Layout für die LogDatei-Ausgabe:
   log4j.appender.de.abas.training.advanced.log4j.Console.layout=org.apache.log4j.PatternLayout
8. Definieren Sie das PatternLayout für das Loggingsystem für die LogDatei-Ausgabe:
   log4j.appender.de.abas.training.advanced.log4j.Console.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %C{7}:%L - %m%n

Resultat: Die Properties-Datei **log4j.properties** wurde erzeugt. Das Logging wurde für Konsolenausgabe konfiguriert.

log4j.properties

```
# define logger
#log4j.logger.de.abas.training.advanced.log4j=Info, de.abas.training.advanced.log4j.Console
log4j.logger.de.abas.training.advanced.log4j=Info, de.abas.training.advanced.log4j.Console, de.
abas.training.advanced.log4j.RollingFile
#, de.abas.training.advanced.log4j.RollingFile
# console appender
log4j.appender.de.abas.training.advanced.log4j.Console=org.apache.log4j.ConsoleAppender
log4j.appender.de.abas.training.advanced.log4j.Console.layout=org.apache.log4j.PatternLayout
log4j.appender.de.abas.training.advanced.log4j.Console.layout.ConversionPattern=%d{yyyy-MM-dd
HH:mm:ss} %-5p %C{7}:%L - %m%n
# file appender
log4j.appender.de.abas.training.advanced.log4j.RollingFile=org.apache.log4j.RollingFileAppender
log4j.appender.de.abas.training.advanced.log4j.RollingFile.File=log/log4j.log
log4j.appender.de.abas.training.advanced.log4j.RollingFile.MaxFileSize=5MB
log4j.appender.de.abas.training.advanced.log4j.RollingFile.MaxBackupIndex=10
log4j.appender.de.abas.training.advanced.log4j.RollingFile.layout=org.apache.log4j.PatternLayout
log4j.appender.de.abas.training.advanced.log4j.RollingFile.layout.ConversionPattern=%d{yyyy-MM-
dd HH:mm:ss} %-5p %C{7}:%L - %m%n
```

CREATENEWPRODUCTSFROMXMLTRANSACTIONLOG4J ALS AJO-CLIENT-ANWENDUNG AUSFÜHREN

Beim Ausführen von CreateNewProductsFromXmlTransactionLog4j als AJO-Standalone-Programm werden die Log-Messages einheitlich formatiert in der Konsole ausgegeben und zusätzlich in die LogDatei **[AJO-Projekt]/loglog4j.log** geschrieben. Die LogDatei wird erzeugt und ist nach einem Refresh (F5) sichtbar.

Das Logging mit log4j wird mithilfe der Datei **log4j.properties** gesteuert. Änderungen an der AJO-Anwendung sind nicht notwendig.

AJO-SERVER-ANWENDUNG MIT LOG4J-LOGGING

In diesem Teil soll die vorhandene AJO-Anwendung **CreateNewProductsFromXmlTransactionLog4j**, in der das Logging mit log4j bereits implementiert ist, als AJO-Server-Anwendung ausgeführt werden. Beachten Sie, dass für AJO-Server-Anwendungen das Logging mit log4j im Mandanten explizit konfiguriert werden muss.

Die Properties-Datei muss **logging.custom.properties** heißen und sich in $MANDANTDIR/java/log/config befinden.

Die erzeugten Log-Dateien werden in $MANDANTDIR/java/log gespeichert.

In diesem Teil wird die Datei **logging.custom.properties** in $MANDANTDIR/java/log/config erzeugt. In der Datei **logging.custom.properties** darf nur die LogDatei-Ausgabe definiert werden. Die LogDatei **log4j.log** wird in $MANDANTDIR/java/log gespeichert.

LOGGING.CUSTOM.PROPERTIES ERZEUGEN

Führen Sie folgende Schritte aus:

1. Erzeugen Sie in $MANDANTDIR/java/log/config die Properties-Datei **logging.custom.properties**.
2. Definieren Sie einen Logger für die LogDatei-Ausgabe. Der Log-Level ist INFO.
   Hinweis: LogLevel INFO gibt INFO, WARN, ERROR und FATAL log-Messages aus.
   log4j.logger.de.abas.training.advanced.log4j=INFO, de.abas.training.advanced.log4j.RollingFile
3. Definieren Sie den Speicherort der LogDatei**:**
   log4j.appender.de.abas.training.advanced.log4j.RollingFile.File=log/log4j.log
4. Definieren Sie die maximale Größe der LogDatei:
   log4j.appender.de.abas.training.advanced.log4j.RollingFile.MaxFileSize=5MB
5. Definieren Sie die maximale Anzahl der LogDateien:
   log4j.appender.de.abas.training.advanced.log4j.RollingFile.MaxBackupIndex=10
6. Definieren Sie das Layout für die Konsolenausgabe:
   log4j.appender.de.abas.training.advanced.log4j.Console.layout=org.apache.log4j.PatternLayout
7. Definieren Sie das PatternLayout für die LogDatei-Ausgabe:
   log4j.appender.de.abas.training.advanced.log4j.Console.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %C{7}:%L - %m%n

Resultat: Die Properties-Datei **logging.cutom.properties** wurde erzeugt.

log4j.properties

```
# logger
log4j.logger.de.abas.training.advanced.log4j=INFO, de.abas.training.advanced.log4j.RollingFile
# first appender
log4j.appender.de.abas.training.advanced.log4j.RollingFile=org.apache.log4j.RollingFileAppender
log4j.appender.de.abas.training.advanced.log4j.RollingFile.File=java/log/log4j.log
log4j.appender.de.abas.training.advanced.log4j.RollingFile.MaxFileSize=5MB
log4j.appender.de.abas.training.advanced.log4j.RollingFile.MaxBackupIndex=10
log4j.appender.de.abas.training.advanced.log4j.RollingFile.layout=org.apache.log4j.PatternLayout
log4j.appender.de.abas.training.advanced.log4j.RollingFile.layout.ConversionPattern=%d{yyyy-MM-
dd HH:mm:ss} %-5p %C{1}:%L - %m%n
```

CREATENEWPRODUCTSFROMXMLTRANSACTIONLOG4J ALS AJO-CLIENT-ANWENDUNG AUSFÜHREN

Die anzulegenden Artikel werden aus der XML-Datei **products.xml** gelesen. Mithilfe dieser Datei können Fehler simuliert werden, z.B. ein Product mit dem Suchwort MYPCXX führt zu einem Rollback.

Beim Ausführen von CreateNewProductsFromXmlTransactionLog4j werden die Log-Messages in die LogDatei **$MANDANTDIR/java/log/log4j.log** geschrieben.

Das Logging mit log4j wird mithilfe der Datei **$MANDANTDIR/java/log/config/logging.custom.properties** gesteuert. Änderungen an der AJO-Anwendung sind nicht notwendig.

logging.custom..properties erzeugen

abas ACADEMY