5

Programmierung mit XML

In diesem Kapitel erfahren Sie:

- welche Parsertypen es gibt,
- wie sie verwendet werden,
- welche XML-Tools für die iSeries verfügbar sind.

Nun kennen Sie die Struktur von XML-Dokumenten und können Dokumente erstellen, die Informationen für die iSeries enthalten. Aber wie werden die Daten von der iSeries in die XML-Dokumente geladen?

Es gibt eine große Zahl von Tools für die automatische Erzeugung von XML-Daten. In vielen Fällen bieten diese eine Lösung, die die meisten technischen Anforderungen erfüllt. Was tun Sie aber, wenn die Funktionalität des Tools nicht ausreicht? Wenn nicht genügend Funktionen bereitstehen, müssen Sie selbst eine Lösung finden.

Es gibt viele Tools und Komponenten, die Sie bei der Entwicklung von XML-fähigen Anwendungen unterstützen können. Diese können XML-Dokumente zur Interaktion mit der iSeries einlesen und erzeugen. Zu den verfügbaren Tools gehören:

- XML-Parser: DOM und SAX
- DB2 XML Extender
- XForms

Mit diesen Tools können XML-fähige Anwendungen auf der iSeries in verschiedenen Programmiersprachen erstellt werden. Dazu gehören Java, RPG, COBOL und ILE C. Verwenden Sie SAX und DOM als XML-Parser. Andere Tools, wie z.B. der DB2 XML Extender, können zur Erweiterung der Funktionen der DB2-Datenbank verwendet werden und bieten Schnittstellen zur Nutzung von XML als Importformat.

Parser

Zu den wichtigsten Komponenten der XML-Technik gehört der XML-Parser. Dabei handelt es sich um einen Prozessor, der die Struktur eines XML-Dokuments versteht. So wird die Manipulation und Validierung der Daten möglich.

Parser werden von einer großen Vielfalt von XML-Techniken für den Lese- und Schreibzugriff auf die Daten in einem XML-Dokument genutzt. Für den Entwickler sind sie außerdem die einzige Methode für den Abruf von Informationen aus dem XML-Dokument. Bei Anforderung gibt der Parser Daten aus dem XML-Dokument zurück.

Ein Parser kann auch Daten innerhalb eines XML-Dokuments validieren. Ein XML-Dokument kann eine Referenz auf eine DTD oder ein XML-Schema enthalten. Beides steuert die Struktur und den Typ der Daten innerhalb eines XML-Dokuments, wie in den Kapiteln 3 und 4 beschrieben. Beim Öffnen eines XML-Dokuments stellt der Parser sicher, dass die enthaltenen Daten den Regeln entsprechen, die innerhalb der DTD oder des Schemas festgelegt wurden. Wenn das nicht der Fall ist, wird das Dokument als ungültig betrachtet und die weitere Verarbeitung wird abgebrochen.

Parsertypen

Mit den Parsern DOM und SAX sind zwei Parsertypen verfügbar, die jeweils eine eigene Sicht des Dokuments und verschiedene Methoden für den Zugriff auf die Daten bieten.

Der Unterschied zwischen der Spezifikation und der spezifischen Parser-Implementierung besteht darin, dass die Spezifikation die gesamte Funktionalität beschreibt, die für den Parser erforderlich ist, während jede Implementierung die beschriebenen Funktionen aufweist, jedoch eigene Namenskonventionen anwendet. Wenn Sie also die Spezifikation verstehen, wissen Sie auch bald, wie Sie mit den verschiedenen Parsern arbeiten können.

DOM

Der Document Object Model (DOM) Parser erlaubt den dynamischen Zugriff sowie die Aktualisierung der Daten und Struktur eines XML-Dokuments durch die Anwendungen. Der DOM-Parser stellt das XML als Hierarchie von Knoten dar. Jeder Knoten kann dabei andere Knoten sowie Attribute enthalten.

Die DOM-Spezifikation beschreibt ausführlich die Objekte und Schnittstellen, die bei der Implementierung dieses Parsers verwendet werden müssen. Sie entstand in Gemeinschaftsarbeit durch IBM, Inso EPS, SoftQuad, Arbortext, Software AG und JavaSoft, die das Ergebnis ihrer Arbeit bei der W3 Group einreichten. Der aktuelle Stand der Spezifikation ist Level 3 Core Version 1.0, wobei die letzte Aktualisierung im November 2003 erfolgte. Unter anderem haben Microsoft und Apache verschiedene Implementierungen der DOM-Spezifikation entwickelt. Dieses Buch bezieht sich auf Level 3 Core Version 1.0.

Sie benötigen nur eine einzige Implementierung des DOM-Parsers für Ihre Arbeit. Diese ist sowohl für die Windows- als auch für die Java-Entwicklung verfügbar.

DOM-Schnittstellen

Die DOM-Spezifikation bietet eine Übersicht über die Objekte, die implementiert werden müssen. Jeder Objekttyp verfügt über eine eigene Schnittstelle, die in den folgenden Absätzen beschrieben wird.

Dokumentschnittstelle

Die Spezifikation beschreibt die zu implementierenden Objekte, wobei das Dokumentobjekt das wichtigste ist, da es sich um das Dokumentelement oder Root-Element des XML-Dokuments handelt. Da es keine Elternknoten haben kann, kann das Root-Element nur einmal vorhanden sein. Die Dokumentschnittstelle dient als „Produktionsstätte" für die Erstellung von Elementen, Text, Kommentaren und anderen XML-Objekten. Sobald diese Objekte vom Dokument erstellt wurden, werden sie mit dem Dokument verknüpft, von dem sie erstellt wurden.

Tabelle 5.1 zeigt eine Liste der Attribute, die erforderlich für die Implementierung der Dokumentenschnittstelle sind:

Attributnamen	Beschreibung
doctype	Der doctype gibt die Document Type Declaration für das Dokument zurück. Wenn der Wert nicht festgelegt wurde, wird eine Null zurückgegeben. Dieses Attribut verwendet den DocumentType-Knoten. Dies ist ein Read-Only-Attribut.
documentElement	Dieses Attribut bietet eine einfache Methode für den Zugriff auf das Dokumentelement des Dokuments. Es gibt eine Element-schnittstelle zurück und ist ein Read-Only-Attribut.
documentURI	Die documentURI gibt die Adresse des Dokuments zurück. Dies ist ein Read-Only-Attribut.
inputEncoding	inputEncoding gibt den Kodierungstyp des Dokuments an. Dies ist ein Read-Only-Attribut.
strictErrorChecking	Wenn dieses Attribut auf “true” gesetzt wird, erfolgt eine strenge Fehlerprüfung des Dokuments. Wenn das Attribut auf “false” gesetzt wird, wird nicht jeder Fehler gemeldet, der während der Ausführung gefunden wurde.
xmlEncoding	Dieses Read-Only-Attribut legt die Kodierung für das XML-Dokument fest. Es wird der Wert Null zurückgegeben, wenn kein Kodierungstyp angegeben wird.
xmlStandalone	Dieses Attribut informiert darüber, ob das XML ein separates Dokument ist oder nicht.
xmlVersion	xmlVersion gibt die Versionsnummer zurück, die im XML-Deklarationselement angegeben wird. Dies ist ein Read-Only-Attribut, das Null zurückgibt, wenn kein Wert angegeben wird.

Tabelle: 5.1: Attribute der DOM-Dokumentenschnittstelle

Tabelle 5.2 zeigt eine Liste der Methoden, die für die Dokumentschnittstelle implementiert werden müssen:

Methodenname	Beschreibung
adoptNode	Dies ist eine neue Methode, die in Level 3 DOM eingeführt wurde. Sie erlaubt das Hinzufügen eines Elements von einem anderen Dokument zum aktuellen Dokument. Damit wird der gesamte untergeordnete Baum in das Zieldokument verschoben.
createAttribute createAttributeNS	Diese Methode erstellt eine Attributschnittstelle. Der Name des Attributs wird bei Ausführung dieser Methode übergeben. Um das Attribut einem Element hinzuzufügen, muss setAttributeNode ausgeführt werden. createAttributeNS erstellt ein Attribut unter Verwendung einer Namensraum-URI.
createCDATASection	Erstellt einen CDATA-Abschnitt innerhalb des XML-Dokuments.
createComment	Die Methode createComment erstellt einen Kommentar innerhalb des XML-Dokuments.
createElement createElementNS createTextNode createDocumentFragment	Diese Methoden erstellen jeweils einen eindeutigen Typ der Elementschnittstelle. Der Name des Elements wird in der Methode übergeben, und wenn das Element als Teil eines Namensraums vorgesehen ist, wird die Namensraum-URI ebenfalls übergeben. Das neue Element wird dem Zielelement durch Verwendung der Methode appendChild hinzugefügt.
getElementById getElementsByTagName getElementsByTagNameNS	Diese Methoden werden das Dokument nach dem Element auf Grundlage der eingegebenen Kriterien durchsuchen. Die Methode getElementById wird dabei nur ein einziges Element zurückgeben, die anderen Methoden geben jedoch eine Gruppe von Elementen innerhalb einer NodeList-Schnittstelle zurück. Wenn mit der ID-Suche keine Elemente gefunden werden, wird ein Nullwert zurückgegeben. Wenn die weiteren Suchen keine Elemente finden, wird eine Knotenliste mit Null Knoten zurückgegeben.
importNode	Diese Methode importiert einen Knoten von einem anderen Dokument, ohne Werte zu ändern.
normalizeDocument	Diese Funktion simuliert das Speichern und erneute Laden des Dokuments.
renameNode	Dies ist eine neue Funktion für Level 3 Core. Die Methode renameNode benennt einen Knoten auf einen neuen Namen um.

Tabelle 5.2: Methoden der DOM-Dokumentenschnittstelle

Knotenschnittstelle

Die Knotenschnittstelle wird als Basisschnittstelle für viele andere Schnittstellen im DOM-Modell genutzt. Innerhalb eines XML-Dokuments kann mit Hilfe der Knotenschnittstelle auf beliebige individuelle Knoten verwiesen werden.

Tabelle 5.3 zeigt eine Liste der Schlüsselmethoden für den Knoten-Schnittstellentyp:

Methodenname	Beschreibung
appendChild insertBefore	Diese Methode ergänzt die Liste der Kinder mit dem Knoten. Die Methode insertBefore verwendet einen Referenzknoten als den Knoten, vor den der neue Knoten gesetzt wird.
cloneNode	Die Methode cloneNode erstellt eine Kopie der Knotenstruktur. Bei Verwendung für einen Knoten sind keine Daten oder Elterninformationen enthalten. Bei Verwendung mit einem Attribut werden die Daten und Kindknoten kopiert. Standardmäßig werden beim Klonen von Elementen die Daten, nicht aber die Kindknoten kopiert. Um Kindknoten zu kopieren, muss "tiefes Klonen" verwendet werden. Beim Klonen einer Entity-Referenzschnittstelle wird eine Kopie des gesamten untergeordneten Baums erstellt, gleichgültig, ob "tiefes Klonen" erfolgt oder nicht.
compareDocumentPosition	Vergleicht die Position des aktuellen Knotens mit der Position des übergebenen Knotens. Es wird ein Wert vom Typ "short" zurückgegeben, der die relative Position angibt.
getUserData	Ruft das Objekt ab, das mit dem Schlüssel verknüpft ist. Wenn ein Objekt, wie z.B. ein Knoten oder Attribut, mit dem aktuellen Knoten mit Hilfe von setUserData verknüpft wurde, erfolgt der Abruf.
hasAttributes	Gibt den Wert "true" (wahr) oder "false" (falsch) zurück, um anzugeben, ob der Knoten über Attribute verfügt oder nicht.
hasChildNodes	Dieses Attribut gibt den Wert "true" oder "false" zurück, um anzuzeigen, ob der Knoten über Kindknoten verfügt oder nicht.
isDefaultNamespace	isDefaultNamespace prüft, ob der angegebene Namensraum der Standardnamensraum für das Dokument ist.

Tabelle 5.3: Methoden für DOM-Knotenschnittstelle (Teil 1 von 2)

Methodenname	Beschreibung
isEqualNode	Diese Methode testet zwei Knoten und stellt fest, ob diese äquivalent sind oder nicht. Äquivalente Knoten müssen über denselben Typ, Namen, lokalen Namen, Namensraum, Präfix, Knotenwert exakt die gleichen Attribute und Kindknoten verfügen.
isSameNode	Die isSameNode Methode prüft, ob der aktuelle Knoten dem übergebenen Knoten entspricht. Derselbe Knoten kann in zwei verschiedenen Abfragen übergeben worden sein. So wird geprüft, ob der Knoten derselbe ist.
isSupported	Diese Methode prüft, ob dieser Knoten die übergebene Funktion unterstützt.
lookupNamespaceURI	Gibt die Namensraum-URI auf Grundlage des angegebenen Präfix zurück.
lookupPrefix	Gibt das angegebene Präfix für den aktuellen Knoten zurück.
normalize	Diese neue Funktion simuliert das Speichern und erneute Laden des Knotens.
removeChild	Entfernt den Knoten aus dem untergeordneten Baum.
replaceChild	Ersetzt einen Knoten mit einem neuen Knoten.
setUserData	Diese Methode verknüpft ein Objekt mit dem Schlüssel für den aktuellen Knoten. Das Objekt kann später durch Aufruf der Methode getUserData im selben Knoten abgerufen werden.

Tabelle 5.3: Methoden für DOM-Knotenschnittstelle (Teil 2 von 2)

Tabelle 5.4 liefert eine Liste der verfügbaren Attribute für die Knotenschnittstelle:

Attributnamen	Beschreibung
attributes	Dieses Attribut gibt die Attribute zurück, die für den Knoten spezifiziert wurden. Wenn keine Attribute vorhanden sind, lautet der Rückgabewert Null.
childNodes	Die Kindknotenattribute geben eine Collection (Sammlung) der Kinder des aktuellen Knotens zurück.
firstChild lastChild nextSibling previousSibling	Gibt einen Kindknoten zurück. Ein Nullwert wird zurückgegeben, wenn der angeforderte Knoten nicht existiert.
namespaceURI	Dieses Read-Only-Attribut gibt den Namensraum für den aktuellen Knoten zurück.
nodeName	Gibt den Namen des Knotens zurück.
nodeType	Gibt den Knotentyp zurück.
nodeValue	Gibt den Knotenwert zurück.
ownerDocument	Dieses Attribut gibt die Owner-Dokument-Schnittstelle zurück.
parentNode	Gibt den Elternknoten zurück.
prefix	Gibt das Namensraum-Präfix zurück.
textContent	Dieses Attribut gibt den Textinhalt für den aktuellen Knoten zurück. Je nach Knotentyp werden unterschiedliche Teile des Knotens zurückgegeben.

Tabelle 5.4: Attribute für DOM-Knotenschnittstelle

Es gibt mehrere Knotentypen, von denen jeder die Knotenschnittstelle implementiert. Beachten Sie dabei, dass die einzelne Implementierung nicht unbedingt alle Methoden und Attribute der Knotenschnittstelle übernehmen muss.

Attributschnittstelle

Die Attributschnittstelle, auch unter der Bezeichnung Attr-Schnittstelle bekannt, ist eine Schnittstellendarstellung eines Knotenattributs. Um den Wert des Attributs abzurufen, kann das Attribut nodeValue verwendet werden. In Abbildung 5.1 sehen Sie ein Beispiel, dass die ID ein Attribut des Knotens „kunde" ist.

```
<kunde ID="123456">
…
</kunde>
```

Abbildung 5.1: Attribut

Diese Implementierung übernimmt nicht alle verfügbaren Methoden und Attribute der Knotenschnittstelle. Attribute enthalten z.B. keine Kindknoten und können daher keine dieser Methoden und Attribute nutzen; Eltern- und Geschwisterknoten geben daher Nullwerte zurück.

Das angegebene Attribut informiert darüber, ob das Attribut explizit im XML-Dokument angegeben wurde. Dies betrifft Werte, die von einem XML-Schema bereitgestellt werden. Wenn für das Attribut kein Wert angegeben wurde und wenn das XML-Schema einen Standardwert aufweist, wird das angegebene Attribut auf „false" gesetzt.

Schnittstelle für den CDATA-Abschnitt

Der Abschnitt CDATA ist eine Schnittstelle, die einen Abschnitt von Text innerhalb eines XML-Dokuments enthält. Innerhalb der Schnittstelle ist fast jede Sequenz der Daten zulässig.

```
<script>
<![CDATA[
function add(a,b) {
  return a + b
}
]]>
</script>
```

Abbildung 5.2: Beispiel für Abschnitt CDATA

In Abbildung 5.2 sind die Daten von CDATA-Tags umgeben, wobei das End-Tag die Zeichenfolge „]]>“ ist. So können Zeichendaten, die sonst umgeschaltet werden müssten, an das XML-Dokument übergeben werden, ohne dass der Verlust von Funktionen befürchtet werden muss.

Kommentarschnittstelle

Diese Schnittstelle positioniert Kommentare innerhalb eines XML-Dokuments. So werden die Werte innerhalb eines Knotens ordnungsgemäß umgeschaltet, wie Sie in Abbildung 5.3 sehen können.

```
<parentNode>
<!– Dies ist ein Kommentarknoten innerhalb des Elternknotens –>
</parentNode>
```

Abbildung 5.3 Beispiel für eine XML-Kommentarschnittstelle

Dokumentfragment-Schnittstelle

Die Dokumentfragment-Schnittstelle hat vieles mit der Dokumentschnittstelle gemeinsam. Ihre Funktionen ermöglichen die Erstellung von Knotenschnittstellen und die Entwicklung eines untergeordneten Baums. Der große Unterschied ist jedoch, dass die Schnittstelle eine komprimierte oder eine „leichtgewichtige" Version der Dokumentschnittstelle darstellt. So kann mit Hilfe des Dokumentfragments ein untergeordneter Baum erstellt und anschließend in die Dokumentschnittstelle eingefügt werden. Diese Funktionalität wird zwar auch von der Dokumentschnittstelle bereitgestellt, je nach Anwendungsimplementierung kann deren Anwendung jedoch problematisch sein.

Dokumenttyp-Schnittstelle

Die Dokumenttyp-Schnittstelle ermöglicht die Bearbeitung der Entities innerhalb einer DTD. Bei Anforderung gibt die Schnittstelle eine Liste aller Entities innerhalb der DTD zurück und bietet eine Methode für die Aktualisierung der Werte.

Elementschnittstelle

Die Elementschnittstelle repräsentiert ein Element, das man innerhalb eines XML- oder HTML-Dokuments findet. Sie hat vieles mit der Knotenschnittstelle gemeinsam, verfügt jedoch über weitere attributbezogene Methoden und Attribute.

Tabelle 5.5 zeigt eine Aufzählung der Methoden, die innerhalb der Elementschnittstelle implementiert werden:

Methodenname	Beschreibung
getAttribute getAttributeNS getAttributeNode getAttributeNodeNS	Diese Methoden geben den Attributwert basierend auf einen Namen zurück: Die Methode getAttributeNS gibt den Attributwert basierend auf Namen und Namensraum zurück. getAttribute Node und getAttributeNodeNS geben einen Attributknoten basierend auf Namen und ggf. Namensraum zurück.
getElementsByTagName getElementsByTagNameNS	Diese Methoden geben eine Liste von Knoten basierend auf den angegebenen Namen und den Namensraum zurück.
hasAttribute hasAttributeNS	hasAttribute und hasAttributeNS bestimmen anhand des Namens und Namensraums, ob ein Attribut im Element vorhanden ist.
removeAttribute removeAttributeNS	Diese Methoden entfernen ein Attribut basierend auf Namen und Namensraum. Wenn das Attribut einen Standardwert aus der DTD oder aus einem Schema enthält, wird ein neues Attribut angezeigt.
removeAttributeNode	Diese Methode entfernt ein Attribut basierend auf der eingegebenen Attributschnittstelle. Ähnlich wie bei removeAttribute und removeAttributeNS wird ein neues Attribut angezeigt, wenn ein Standardwert angegeben wird.
setAttribute setAttributeNS setAttributeNode setAttributeNodeNS	Diese Methoden legen den Wert eines Attributs fest. Der Name, der Wert und ggf. der Namensraum werden für den Wert verwendet. Wenn das Attribut bereits existiert, wird mit diesen Methoden das Attribut erstellt. setAttributeNode und setAttributeNodeNS fügen ein neues Attribut auf Grundlage eines Objekts anstelle auf Grundlage von Werten hinzu.
setIdAttribute setIdAttributeNS setIdAttributeNode setIdAttributeNodeNS	Diese Methoden legen die ID für das Attribut basierend auf den angegebenen Namen und Namensraum fest. setIdAttributeNode und setIdAttributeNodeNS legen die ID des Attributs auf Grundlage des übergebenen Knotens anstelle auf Grundlage von Werten fest.

Tabelle 5.5: Methoden der DOM-Elementschnittstelle

In Tabelle 5.6 werden die Attribute aufgelistet, die in der Elementschnittstelle implementiert wurden:

Attributname	Beschreibung
schemaTypeInfo	Gibt aus dem Schema die Typinformationen zurück, die mit diesem Element verknüpft sind.
tagName	Das Attribut gibt den Namen des Elements zurück. Dieses Attribut verhält sich etwas anders als das Attribut nodeName, das von der Knotenschnittstelle geerbt wird. Bei Verwendung mit einem XML-Dokument wird der Name des Elements zurückgegeben, wobei Groß- und Kleinschreibung beibehalten werden. Bei Verwendung mit einem HTML-Dokument wird der Name des Tags in Großbuchstaben zurückgegeben, gleichgültig, wie die Schreibung in HTML erfolgt.

Tabelle 5.6: Attribute der DOM-Elementschnittstelle

Entity-Schnittstelle

Eine Entity-Schnittstelle repräsentiert eine Entity innerhalb einer DTD. Diese Implementierung ermöglicht die Bearbeitung der Werte für die Entity. Das Attribut nodeName gibt den Namen der Entity zurück.

Die Entity-Schnittstelle verfügt über weitere interessante Attribute, die in Tabelle 5.7 aufgelistet sind:

Attributname	Beschreibung
inputEncoding	Gibt die Kodierung für die Entity zurück.
notationName	Gibt den Namen der Notation für ungeparste Entities zurück. Bei geparsten Entities wird ein Nullwert zurückgegeben.
publicId	Das publicId-Attribut gibt den Public Identifier für die Entity zurück. Wenn kein Identifier vorhanden ist, wird Null zurückgegeben.
systemId	Dieses Attribut gibt den System Identifier für die Entity zurück. Wenn kein Identifier vorhanden ist, wird Null zurückgegeben.
xmlEncoding	Gibt die XML-Kodierung für die angegebene Entity zurück.
xmlVersion	Gibt die XML-Version für die Entity zurück.

Tabelle 5.7: Attribute der DOM-Entity-Schnittstelle außer Attribut nodeName

Entity-Referenzschnittstelle

Die Entity-Referenzschnittstelle erlaubt die Erstellung und Manipulation von Entity-Referenzen, die die Wertsubstitution während des Parsens ermöglichen. Diese Funktionalität ist in der HTML-Entwicklung sehr verbreitet. Wenn für das richtige Setzen ein leeres Zeichen benötigt wird, gibt man ** ** in den HTML-Code ein. ** ** wird so zu ** ** umgewandelt, d.h. in den hexadezimalen Wert für ein Leerzeichen.

```
<!ENTITY entit     yName “Entity-Wert“>
```

Abbildung 5.4: Entity-Referenzformat

Abbildung 5.4 zeigt die Struktur eines Entity-Referenzknotens. Eine Entity-Referenz enthält nur zwei Komponenten: den Namen und den Wert der Entity. Die DOM-Implementierung bietet eine Methode für die Erstellung derartiger Knoten, die keine normalen, wohlgeformten XML-Knoten sind.

```
<!ENTITY nbsp “ ” –>
```

Abbildung 5.5: Beispiel für eine Entity-Referenz für ein Leerzeichen

```
<kundenName>Joe&nbspSmith</kundenName>
```

Abbildung 5.6 Verwendung einer Entity-Referenz innerhalb eines XML-Dokuments

Die Abbildungen 5.5 und 5.6 zeigen, wie eine Entity-Referenz deklariert und verwendet wird. Abbildung 5.5 deklariert eine einfache Entity mit dem Namen **nbsp** und dem Wert ** .** In Abbildung 5.6 wird die Entity-Referenz im Knoten kundenName verwendet. Dem Namen wird das Zeichen **&** vorangestellt.

Notationsschnittstelle

Die Notationsschnittstelle definiert eine Notationsreferenz für die Verwendung innerhalb einer DTD. Ein Notationselement wird verwendet, um ein Programm für die Verarbeitung eines Typs basierend auf dem Datenformat oder den Typ Datenformat anzugeben. Der Notationsknoten ist in der DOM 3-Implementierung Read-Only und enthält keine Kindknoten.

Schnittstelle für Verarbeitungsanweisungen

Diese Schnittstelle enthält eine Verarbeitungsanweisung, die zur Aufnahme spezifischer Informationen benötigt wird. Sie bietet die Möglichkeit der Verknüpfung eines Cascading Style Sheets oder sogar eines XSL-Stylesheets mit XML. Während der Verarbeitung kann XSL abgerufen und anschließend auf das XML-Dokument angewendet werden.

Die Schnittstelle für Verarbeitungsanweisungen bietet einen Mechanismus für die Identifizierung des Dokuments sowie des Dokumenttyps zur Verknüpfung mit XML. Darüber hinaus bietet ihre Implementierung eine Möglichkeit, zusätzliche Informationen bezüglich der jeweiligen Verarbeitungsanweisungen hinzuzufügen.

Textschnittstelle

Die Textschnittstelle repräsentiert eine textuelle Version von beliebigem Text, der sich innerhalb eines Elements oder einer Attributschnittstelle befindet. Diese Schnittstelle wird zur Manipulation des Texts verwendet, der innerhalb eines Elements oder Attributs eingefügt wurde. Die Textschnittstelle bietet Zugriff auf den Text im aktuellen Element sowie in allen Kindelementen.

Tabelle 5.8 listet die zusätzlichen Methoden für die Textschnittstelle auf:

Methodenname	Beschreibung
replaceWholeText	Diese Methode ersetzt den Text im aktuellen Knoten sowie alle Werte in den benachbarten Elementen.
splitText	Diese Methode teilt den Text auf Grundlage des eingehenden Indexes in zwei Teile auf. Der erste Teil ist der Text bis zum Indexwert; die andere Hälfte besteht aus dem übrigen Text.

Tabelle 5.8: Methoden für DOM-Textschnittstelle

Die Textschnittstelle verfügt über die Attribute, die in Tabelle 5.9 aufgelistet sind:

Attributnamen	Beschreibung
isElementContentWhitespace	Dieses Attribut gibt "true" zurück, wenn der Text innerhalb des Elements Leerräume enthält.
wholeText	Das Attribut wholeText gibt den gesamten Text von allen benachbarten Elementen zurück.

Tabelle 5.9: Attribute der DOM-Textschnittstelle

Verwendung des DOM-Parsers

Um den DOM-Parser nutzen zu können, sind folgende einfachen Schritte notwendig:

1. Erstellen Sie den Document Builder mit Hilfe der Document Builder Factory.
2. Laden Sie die XML-Datei in ein Dokument mit Hilfe der Parse-Methode des Document Builders.
3. Führen Sie die erforderlichen Aktionen aus (z.B. Werte einlesen oder aktualisieren).
4. Speichern Sie das Dolument soweit erforderlich.

Um das Zusammenspiel dieser Schritte zu zeigen, verwenden wir das XML-Dokument aus Abbildung 5.7. Das XML-Dokument enthält einfache Kundendaten.

```
<?xml version="1.0" ?>
<kunde id="12345">
   <vorname>Joe</vorname>
   <initiale>E.</initiale>
   <nachname>Smith</nachname>
   <adressen>
      <adresse typ="zuhause">
         <strasse>123 Avenue</strasse>
         <stadt>New York City</stadt>
         <staat>New York</staat>
         <plz>12345</plz>
      </adresse>
      <adresse typ="post">
         <strasse>123 Strasse</strasse>
         <stadt>Atlanta</stadt>
         <staat>Georgia</staat>
         <plz>23456</plz>
      </adresse>
   </adressen>
</kunde>
```

Abbildung 5.7: Kundendaten

Der folgende Java-Code in Abbildung 5.8 parst das Dokument und sendet die Informationen zum Systemausgabe-Stream. Dieses Java-Beispiel verwendet die Apache-Implementierung des DOM-Modells namens Xerces Version 2.6.0. Weitere Informationen über die Apache-Implementierung erhalten Sie unter *http://xml.apache.org/*.

```
01 public class domRead {
02   public static void main(String args[]) throws
       Exception
03   {
04      DocumentBuilder db =
          DocumentBuilderFactory.newInstance().
          newDocumentBuilder();
05     Document xml = db.parse(new InputSource(new
          FileReader(new File("customerinformation.
          xml"))));
06     Element documentRoot = xml.getDocumentElement();
07     displayElementDetails(documentRoot);
08     NodeList firstNameList =
         documentRoot.getElementsByTagName("vorname");
09     if (firstNameList.getLength() > 0)
10     {
11        Element firstName = (Element) firstNameList.
            item(0);
12        displayElementDetails(vorname);
13     }
14     NodeList addressList =
         documentRoot.getElementsByTagName("adresse");
15     for (int i = 0; i < addressList.getLength(); i++)
16     {
17        Element addressElement = (Element) addressList.
             item(i);
18        displayElementDetails(addressElement);
19        for(int x = 0; x <
            addressElement.getChildNodes().getLength();
            x++)
20        {
21            if (addressElement.getChildNodes().item(x).
              getNodeType() == Node.ELEMENT_NODE)
22            {
23               Element childNode = (Element)
                   addressElement.getChildNodes().item(x);
```

5.8: Der Java-Code für das Parsen des Kundendaten-XML (Teil 1 von 2)

```
24              System.out.print(childNode.
                  getParentNode().getNodeName() + " - ");
25              System.out.print(childNode.getNodeName()
                  + " - ");
26              if (childNode.hasChildNodes() &&
27                   childNode.getFirstChild().
                       getNodeType() == Node.TEXT_NODE) {
28                   Text textNode = (Text) childNode.
                       getFirstChild();
29                   System.out.println(textNode.
                       getNodeValue());
30             }
31           }
32        }
33      }
34    }
35    private static void displayElementDetails(Element
        inElement) throws Exception
36    {
37      if (inElement.getParentNode() != null)
38      System.out.print(inElement.getParentNode().
          getNodeName() + " - ");
39      System.out.println(inElement.getNodeName());
40      for(int i = 0; i < inElement.getAttributes().
          getLength(); i++)
41      {
42         Attr attribute = (Attr) inElement.getAttributes
             ().item(i);
43         System.out.print(attribute.getName() + " - ");
44         System.out.println(attribute.getValue());
45      }
46    }
47 }
```

5.8: Der Java-Code für das Parsen des Kundendaten-XML (Teil 2 von 2)

Wie oben beschrieben ist der erste Schritt zur Verwendung des DOM-Modells die Erstellung des Document Builders mit Hilfe der Document Builder Factory. Dies erfolgt in Zeile 04. Im nächsten Schritt wird der Document Builder zur Vorbereitung des XML-Dokuments verwendet. In Zeile 05 öffnet der Document Builder die „customerinformation.xml“ (Abbildung 5.7) im Dokumentobjekt.

Als Erstes wird das Dokumentelement vom Dokument abgerufen. In Zeile 06 erfolgt dies mit Hilfe der Methode getDocumentElement. Die Methode displayElementDetails zeigt die Informationen über das Element an. In dieser Methode werden der Knotenname (Zeile 39), der Elternknotenname (Zeilen 37 und 38) und die gesamten Attributinformationen (Zeilen 40 bis 45) angezeigt.

In Zeile 08 wird das Element „vorname“ mit der Methode getElementsByTagName abgerufen. Das Resultat ist eine Knotenlistenklasse, die alle Knoten mit dem Namen „vorname“ enthält. Um sicherzustellen, dass die Abfrage Resultate zurückgibt, wird überprüft, ob der Wert, der mit der Methode getLength in der Knotenliste zurückgegeben wird (Zeile 9), größer als 0 ist.

Im nächsten Schritt werden alle Adressen angezeigt, die im XML enthalten sind. Zeile 14 führt eine Abfrage des Dokumentelements aus und ruft alle Adressknoten ab. Das Ergebnis der Abfrage sind die beiden Adressknoten. In den Zeilen 15 bis 33 werden die Informationen zu den Adressen gezeigt. In Zeile 21 wird der Kindknoten geprüft, um sicherzustellen, dass es sich um ein Element handelt. Wenn das Adresselement ein Attribut enthält, wird das Attribut ebenfalls als Kind zurückgegeben. In Zeile 26 wird außerdem vor Anzeige des Wertes des Adresskindele-

ments geprüft, ob das Element Kindknoten enthält und ob es sich um einen Textknoten handelt. Der Grund dafür ist, dass der Textwert des Elements in einem Kindknoten mit dem Typ Textknoten enthalten ist.

In Abbildung 5.7 sehen Sie zwei Adressknoten innerhalb des Elements „adressen". Wären im Dokument weitere Adresselemente vorhanden, die auch außerhalb dieses Elements vorkommen könnten, würde die Abfrage auch diese Elemente zurückgeben. Zur Eingrenzung der Suche kann getElementsByTagName mit dem Element „adressen" anstatt mit dem Dokumentelement ausgeführt werden.

```
#dokument – kunde
id – 12345
kunde – vorname
adressen – adresse
typ – zuhause
adresse – strasse – 123 Avenue
adresse – stadt – New York City
adresse – staat – New York
adresse – plz – 12345
adressen – adresse
typ – post
adresse – strasse – 123 Strasse
adresse – stadt – Atlanta
adresse – staat – Georgia
adresse – plz – 23456
```

Abbbildung 5.9: Ausgabe nach Ausführung

Das Ergebnis der Ausführung der domRead-Anwendung in dieser XML-Datei sehen Sie in Abbildung 5.9.

Mit DOM in ein XML-Dokument schreiben

Das Schreiben in ein XML-Dokument mit Hilfe des DOM-Parsers erfolgt ähnlich wie das Einlesen. Das Dokument muss geöffnet werden und die Elemente werden abgefragt. Der Unterschied zeigt sich beim Zugriff auf die Werte der Elemente und Attribute.

Der erste Schritt beim Schreiben von XML ist immer die Erstellung eines Dokuments. Hierzu wird mit dem Dokumentobjekt die Create-Elementmethode eingesetzt. Der folgende Code in Abbildung 5.10 erstellt z.B. ein Element „geburtsdatum“, mit dem anschließend das Dokumentelement ergänzt wird. Das Element kann dem Dokumentelement oder jedem beliebigen Element innerhalb des XML-Dokuments hinzugefügt werden.

```
Element birthDateElement = xml.createElement(“geburtsdatum”);
documentRoot.appendChild(birthDateElement);
```

Abbildung 5.10: Erstellen eines Elements und Hinzufügen zu einem Dokument

Der zweite Schritt ist das Hinzufügen eines Werts zum Element. Hierzu müssen Sie lediglich eine Textklasse erstellen und an das Element anhängen. In Abbildung 5.11 sehen Sie ein Code-Beispiel:

```
Element geburtsdatumElement = xml.createElement(“geburtsdatum”);
Text geburtsdatumText = xml.createTextNode(“12/15/1968“);
geburtsdatumElement.appendChild(geburtsdatumText);
documentRoot.appendChild(geburtsdatumElement);
```

Abbildung 5.11: Hinzufügen eines Werts zu einem Element

Die Änderung vorhandenen Texts kann mit einer von zwei Methoden erfolgen: der alte Textknoten kann ersetzt oder bearbeitet werden. Um den Textknoten des vorhandenen Elements „nachname" in Abbildung 5.7 zu ersetzen, gehen Sie wie in Abbildung 5.12 vor:

```
Text nachnameText = xml.createTextNode("Brown");
nachname.replaceChild(nachnameText,
  (Text)nachname.getChildNodes().item(0));
```

Abbildung 5.12: Ersetzen eines Textknotens

Mit diesem Code wird des möglich, den ersten Kindknoten des Elements „nachname" mit dem neuen Text zu ersetzen. Anschließend kann der Code in Abbildung 5.13 mit denselben Ergebnissen verwendet werden:

```
nachname.getChildNodes().item(0).setNodeValue("Brown");
```

Abbildung 5.13: Ersetzen von Text innerhalb eines Knotens

Um Attribute eines Elements hinzuzufügen oder zu ändern, kann die Methode setAttribute zum Einsatz kommen. Die Methode setAttribute erwartet Namen und Wert des Attributs. Wenn das Attribut bereits vorhanden ist, wird der Wert aktualisiert. Abbildung 5.14 zeigt ein Beispiel, wie das ID-Feld des Kundenelements geändert werden kann:

```
documentElement.setAttribute("id", "456789");
```

Abbildung 5.14: Setzen eines Attributs

SAX

Der zweite wichtige Parser ist der Simple API for XML (SAX) Parser. Der SAX-Parser war der erste Parser in der Java-Welt. Dabei handelt es sich, wie der Name bereits andeutet, um einfache APIs, die das Parsen und die Erstellung eines XML-Dokuments erlauben.

SAX funktioniert völlig anders als DOM. Wie bereits beschrieben, lädt der DOM-Parser das Dokument in einer Hierarchie in den Speicher. Der SAX-Parser hingegen basiert auf einem Event-basierten Modell. Bei spezifischen Schlüsselkomponenten werden Events mit Hilfe von Callbacks zurück zur Implementierung gesendet und der Code wird dementsprechend formuliert.

Dieser Parser ist das Ergebnis einer Zusammenarbeit der XML-DEV Mailing-Liste. Seine Spezifikation kann frei verwendet werden. Jede Organisation, von der der Parser eingesetzt wird, ist selbst für die Wartung des eigenen Codes zuständig. Die aktuelle Version der SAX-Spezifikation lautet 2.0.1.

Komponenten des SAX-Parsers

DefaultHandler

Der SAX-Parser weist nur wenige Gemeinsamkeiten mit anderen Parsern auf. Zu den wichtigsten Klassen im SAX-Modells gehört der DefaultHandler. Diese Klasse ist die Basisklasse, zu der alle Events gesendet werden.

Folgende Methoden in Tabelle 5.10 sind Teil des DefaultHandler:

Methodenname	Beschreibung
characters	Diese Methode erhält eine Nachricht, wenn Zeichen innerhalb eines Elements erkannt werden. Die Zeichen werden in die Methode innerhalb eines Zeichenfelds übergeben.
endDocument	Die Methode endDocument erhält eine Nachricht, wenn das Ende des Dokuments erkannt wird.
endElement	Diese Methode erhält eine Nachricht, wenn ein zuvor geöffnetes Element endet. Beispiele für übergebene Parameter sind URI, Elementname und Name.
endPrefixMapping	Diese Methode wird benachrichtigt, wenn eine Namensraum-Zuordnung endet. Der Name des beendeten Namensraums wird an die Methode als Parameter übergeben.
error	Diese Methode wird ausgeführt, wenn während des Parsens ein behebbarer Fehler auftritt. Als Parameter wird ein SAXParseException-Objekt übergeben.
fatalError	fatalError wird übergeben, wenn ein nicht behebbarer Fehler auftritt. Als Parameter wird ein SAXParseException-Objekt übergeben.
ignorableWhitespace	Diese Methode erhält eine Nachricht, wenn in einem Element ignorierbarer Leerraum erkannt wird. Die zurückgegebenen Werte werden in einem Zeichenfeld übergeben.
notationDecl	notationDecl erhält eine Nachricht, wenn eine Notationsdeklaration erkannt wurde.

Tabelle 5.10: Methoden des SAX DefaultHandler (Teil 1 von 2)

Methodenname	Beschreibung
processingInstruction	Erhält eine Nachricht, wenn eine Verarbeitungsanweisung im XML-Dokument erkannt wurde. Das Ziel und die Daten werden an die Methode übergeben.
resolveEntity	Diese Methode löst eine externe Entity auf. Die Public ID und die System ID werden als Parameter übergeben.
setDocumentLocator	setDocumentLocator gibt ein Locator-Objekt für die Dokument-Events an. Das Locator-Objekt wird später in diesem Kapitel ausführlicher beschrieben.
skippedEntity	Eine Nachricht bezüglich einer ausgelassenen Entity wird mit Hilfe dieser Methode gesendet. Entities können mit einem Filter ausgelassen werden.
startDocument	Diese Nachricht trifft ein, wenn der Anfang eines Dokuments erkannt wird.
startElement	startElement wird ausgeführt, wenn der Anfang eines Elements erkannt wird. URI, Name, Namensraum und alle Attribute werden als Parameter gesendet.
startPrefixMapping	Diese Methode wird aufgerufen, wenn der Anfang eines Namensraums erkannt wurde. Der Namensraum wird an diese Methode als Parameter übergeben.
unparsedEntityDecl	Diese Methode informiert darüber, dass eine ungeparste Entity erkannt wurde. Alle Informationen über die Entity werden an die Methode übergeben.
warning	Diese Methode wird aufgerufen, wenn eine Parse-Warnung erkannt wurde. An die Methode wird ein SAX-Fehler übergeben.

Tabelle 5.10: Methoden des SAX DefaultHandler (Teil 2 von 2)

Der DefaultHandler wird von den folgenden vier SAX-Basis-Handlern implementiert:

- ContentHandler
- DTDHandler
- EntityResolver
- ErrorHandler

Bei allen Handlern werden die zurückgegebenen Informationen von der Anwendung gespeichert. Der Parser meldet nur, wenn eine spezifische Komponente erkannt wird. Die Werte können innerhalb einer Hash-Tabelle oder in Variablen für die zukünftige Verwendung gespeichert werden.

ContentHandler

ContentHandler ist der am häufigsten verwendete Handler, der für alle Typen wohlgeformter Dokumente eingesetzt wird.

Die Reihenfolge, in der die Events aufgerufen werden, ist für die Struktur des XML-Dokuments sehr wichtig. Sehen wir uns z.B. Abbildung 5.15 an, die einfachen XML-Code mit Kundendaten zeigt.

```
<?xml version=”1.0” ?>
<kunde>
   <vorname>Joe</vorname>
   <initiale>E.</initiale>
   <nachname>Smith</nachname>
</kunde>
```

Abbildung 5.15 Einfaches Kunden-XML

Der SAX-Parser liest dieses Dokument ein, indem die folgenden Event-Informationen zur Implementierung zurückgesendet werden:

1. Beginn des XML-Dokument
2. Beginn des Elements Kunde
3. Beginn des Elements Vorname
4. Enthält die Zeichen: Joe
5. Ende des Elements Vorname
6. Beginn des Elements Initiale
7. Enthält die Zeichen: E.
8. Ende des Elements Initiale
9. Beginn des Elements Nachname
10. Enthält die Zeichen: Smith
11. Ende des Elements Nachname
12. Ende des Elements Kunde
13. Ende des XML-Dokuments

DTDHandler

Wenn eine Anwendung Informationen über Notationen und ungeparste Entities benötigt, kann der DTDHandler verwendet werden. Dieser gibt DTD-spezifische Ereignisse zurück, damit ein DTD-Dokument geparst und interpretiert werden kann. Der Handler kann Informationen in beliebiger Reihenfolge ungeachtet der Reihenfolge zurückgeben, in der er deklariert wurde.

Der DTDHandler verfügt über zwei zusätzliche Methoden zum Parsen eines DTD-Dokuments, die Sie in Tabelle 5.11 sehen können:

Methodenname	Beschreibung
notationDecl	Diese Methode erhält eine Nachricht, wenn eine Notation erkannt wurde. Die Methode erhält den Namen, Public ID und System ID als Parameter.
unparsedEntityDecl	Diese Methode erhält eine Nachricht, wenn eine Entity erkannt wurde. Der Name, Public ID, System ID und der Notationsname werden als Parameter an die Methode gesendet.

Tabelle 5.11: Methoden des SAX DTDHandler

ErrorHandler

Wenn während des Parsens Fehler auftreten, können diese in einem ErrorHandler abgefangen werden. Fehlt diese Schnittstelle, könnten Warnungen und nicht behebbare Fehler unbemerkt bleiben. Die Anwendung ist für die Meldung der Fehler und Bereitstellung der geeigneten Fehlerbehandlung zuständig.

Die ErrorHandler-Schnittstelle kann drei Fehlertypen abfangen:

- Warnungen
- Fehler
- Nicht behebbare Fehler

Warnungen werden abgefangen, wenn ein nicht kritischer Fehler erkannt wird, z.B. wenn ein zusätzliches Attribut für ein Element existiert. Dies muss nicht unbedingt zu Problemen in der Anwendung führen, von der die Ausführung auch fortgesetzt werden kann. Standardmäßig werden Warnungen ignoriert ohne die Verarbeitung abzubrechen.

Fehler-Events entstehen bei Regelverletzungen in den XML-Daten. Dieser Typ „Event“ wird aufgerufen, wenn die Daten für ein Attribut oder ein Element nicht geeignet sind. Die Anwendung kann die Verarbeitung auch abbrechen, da weitere Daten ungültig sein können.

Events vom Typ „nicht behebbarer Fehler“ entstehen, wenn das Format des XML-Dokuments problematisch ist. In diesem Fall ist das XML nicht wohlgeformt und die Verarbeitung kann nicht fortgesetzt werden.

Ein Fehler-Handler wird mit Hilfe von setErrorHandler der Klasse XMLReader festgelegt. Die Angabe muss vor dem Erhalt von Dokument-Events erfolgen. Abbildung 5.16 zeigt, wie ein Fehler-Handler für einen XML Reader festgelegt wird.

```
XMLReader saxReader = XMLReaderFactory.createXMLReader();
SaxHandler myHandler = new SaxHandler();
saxReader.setContentHandler(myHandler);
saxReader.setErrorHandler(myHandler);
```

Abbildung 5.16: Fehler-Handler für XML Reader

EntityResolver

Die Referenz auf einen EntityResolver erfolgt, wenn der Parser Daten identifizieren muss, die mit einer URI angegeben werden. Der Resolver gibt entweder die System ID oder ein InputSource-Objekt für das Einlesen zurück. So kann die Anwendung die Entity innerhalb des referenzierten Dokuments bestätigen. Ein EntityResolver wird für den XML Reader durch Aufruf des setErrorHandler festgelegt.

Attribute

Die Attribut-Schnittstelle enthält alle Attribute für ein Element. Wenn während der Verarbeitung ein Element mit Attributen erkannt wird, wird eine Attribut-Klasse als Parameter an die Methode übergeben. Zu diesem Zeitpunkt können die Werte aus den erforderlichen Attributen extrahiert werden.

Die Attribut-Schnittstelle implementiert die Methoden, die In Tabelle 5.12 aufgelistet sind:

Methodenname	Beschreibung
getIndex	Die Methode getIndex gibt den Indexwert für ein Attribut zurück. Der qualifizierte Name oder die URI und der lokale Name können verwendet werden, um den Index abzurufen.
getLength	Diese Methode gibt die Anzahl der Attribute zurück, die für das Element zurückgegeben werden.
getLocalName	getLocalName gibt den lokalen Namen des Attributs zurück. Der Index wird genutzt, um festzustellen, welches Attribut verwendet werden soll.
getQName	Die Methode getQName gibt den qualifizierten Namen des Attributs zurück. Der Index wird genutzt, um festzustellen, welches Attribut verwendet werden soll.
getType	Mit dieser Methode wird der Attributtyp abgerufen. Durch Angabe des Index, des qualifizierten Namens oder sowohl der URI als auch des lokalen Namens kann der Typ abgerufen werden. Das Attribut kann einen der folgenden Werte aufweisen: “CDATA”, “ID”, “IDREF”, “IDREFS”, “NMTOKEN”, “NMTOKENS”, “ENTITY”, “ENTITIES” oder “NOTATION”.
getURI	Die Methode getURI ruft den Attribut-Namensraum entsprechend dem angegebenen Index ab.
getValue	Die Methode getValue ruft den Wert des Attributs durch Angabe des Indexes, des qualifizierten Namens oder sowohl der URI als auch des lokalen Namens ab.

Tabelle 5.12: Methoden der SAX-Attributschnittstelle

Locator-Schnittstelle

Die Locator-Schnittstelle informiert die Anwendung darüber, wo sie sich im aktuellen XML-Dokument befindet. Sie stellt die genauen Daten zu Zeile und Spalte zur Verfügung, die vom Parser aktuell bearbeitet werden. Das kann beim Debuggen von Parser-Fehlern nützlich sein. Beim Abfangen einer Ausnahme erhalten Sie so z.B. ausführliche Informationen zum aufgetretenen Fehler.

Ein Locator wird im ContentHandler mit Hilfe von setDocumentLocator angegeben. Bevor er Events empfangen kann, muss er spezifiziert werden.

XMLFilters

XML-Filter ermöglichen die Änderung oder Trennung von zurückgegebenen Werten, bevor sie vom Handler verarbeitet werden. Dies ermöglicht die Änderung oder das Ignorieren von Werten je nach Bedarf.

Die XML-Filter-Schnittstelle erweitert die Klasse XMLReader. Der Code führt alle erforderlichen Schritte aus, bevor der Handler die Nachricht erhält. Anschließend wird der Filter für das Einlesen des Dokuments verwendet, wodurch das Dokument an den Handler übergeben wird. Der Handler sieht nur die Aktionen, die der Filter zulässt.

Parsen von XML mit SAX

Das Parsen eines XML-Dokuments mit dem SAX-Parser ist ebenfalls völlig anders als beim DOM-Modell. Hierfür sind die folgenden Schritte erforderlich:

1. Erstellen Sie eine Handler-Klasse für die Aufnahme der vom XML eintreffenden Events. Dieser Handler kann für das XML spezifisch sein.
2. Erstellen Sie innerhalb einer Anwendung eine Instanz eines XML Readers. Diese sendet die Ereignisse zum Handler.
3. Erstellen Sie eine Instanz des Handlers und verknüpfen Sie diese mit dem XML Reader.
4. Öffnen Sie das XML-Dokument.
5. Parsen Sie das Dokument.

Bei Verwendung des XML aus Abbildung 5.7 kann der folgende Java-Code in Abbildung 5.17 zum Parsen verwendet werden. Dieser Code ist für den Empfang der Warnmeldungen vom XML Reader zuständig.

```
01 import org.xml.sax.*;
02 import org.xml.sax.helpers.*;
03 public class MyHandler extends DefaultHandler
04 {
05       public MyHandler ()
06       {
07          super();
08       }
09       public void startDocument ()
10       {
11          System.out.println("Start document");
12       }
13       public void endDocument ()
14       {
15          System.out.println("End document");
16       }
17       public void startElement(String uri, String name,
           String qName, Attributes atts)
18       {
19          System.out.println("Start element: " + qName);
20       }
21       public void endElement (String uri, String name,
           String qName)
22       {
23          System.out.println("End element: " + qName);
24       }
25       public void characters (char ch[], int start, int
           length)
26       {
27          System.out.print("Characters: ");
28          for (int i = start; i < start + length; i++)
29          {
```

Abbildung 5.17: SAX Handler (Teil 1 von 2)

```
30          System.out.print(ch[i]);
31          }
32          System.out.println("");
33       }
34 }
```

Abbildung 5.17: SAX Handler (Teil 2 von 2)

Der Handler sieht einige spezifische Events für den Beginn und das Ende des XML-Dokuments vor. Die Zeilen 09 bis 12 verfügen über den Event-Code für den Beginn des XML-Dokuments, die Zeilen 13 bis 16 verfügen über den Event-Code für das Ende des Dokuments.

Die meisten Import-Events beziehen sich auf die Anfangs- und End-Events eines Elements. Diese beiden Events erledigen den Großteil der Verarbeitung des XML-Dokuments. Die Zeilen 17 bis 20 enthalten den Code für den Anfang eines Elements und die Zeilen 21 bis 24 enthalten den Code für das End-Event.

Die letzte Event-Methode erhält die Zeichendaten für den aktuellen Knoten. Die Methode empfängt einen beliebigen Bereich von Zeichen sowie die Länge und den Startwert dieses Bereichs.

```
01 import org.xml.sax.*;
02 import org.xml.sax.helpers.*;
03 import java.io.*;
04 public class SaxRead
05 {
05      public static void main(String args[]) throws
        Exception
06      {
07         XMLReader xmlReader = XMLReaderFactory.
           createXMLReader();
08         MyHandler handler = new MyHandler();
09         xmlReader.setContentHandler(handler);
10         xmlReader.setErrorHandler(handler);
11         FileReader fileReader = new FileReader
           (“customerinformation.xml“);
12         xmlReader.parse(new InputSource(fileReader));
13      }
14 }
```

Abbildung 5.18 Verwendung des Handlers

Abbildung 5.18 zeigt die Anwendung, die für die Kombination aller Klassen zuständig ist. Wie beschrieben wird durch Anforderung von der XML Reader Factory der XML Reader erstellt. Das sehen Sie in Zeile 07. Der Handler, der in Abbildung 5.17 erstellt wurde, wird in Zeile 08 kodiert und in den Zeilen 09 und 10 als ContentHandler und ErrorHandler festgelegt.

In Zeile 11 wird die Datei „customerinformation.xml“, die in Abbildung 5.7 definiert wurde, in eine File Reader-Klasse geladen. Anschließend wird sie in Zeile 12 mit der Klasse XML Reader in der Parse-Methode verwendet.

Die folgende Ausgabe in Abbildung 5.19 ist das Ergebnis des Parsens.

```
Start document
Start element: kunde
Characters:
Start element: vorname
Characters: Joe
End element: vorname
Characters:
Start element: initiale
Characters: E.
End element: initiale
Characters:
Start element: nachname
Characters: Smith
End element: nachname
Characters:
Start element: adressen
Characters:
Start element: adresse
Characters:
Start element: strasse
Characters: 123 Avenue
End element: strasse
Characters:
Start element: stadt
Characters: Boston
End element: stadt
Characters:
Start element: staat
Characters: Mass
End element: staat
```

Abbildung 5.19: Die Ausgabe der SAX-Parse-Daten (Teil 1 von 2)

```
Characters:
Start element: plz
Characters: 12345
End element: plz
Characters:
End element: adresse
Characters:
Start element: adresse
Characters:
Start element: strasse
Characters: 123 Strasse
End element: strasse
Characters:
Start element: stadt
Characters: Atlanta
End element: stadt
Characters:
Start element: staat
Characters: Georgia
End element: staat
Characters:
Start element: plz
Characters: 23456
End element: plz
Characters:
End element: adresse
Characters:
End element: adressen
Characters:
End element: kunde
End document
```

Abbildung 5.19: Die Ausgabe der SAX-Parse-Daten (Teil 2 von 2)