Laboratorium 6 - SOA.
Tematyka: JAXB

Praca z plikami XML – łączenie plików XML z obiektami w javie

Celem laboratorium jest zaznajomienie z technologią pracy z plikami XML z wykorzystaniem specyfikacji JAXB.

https://jaxb.java.net/tutorial/

https://docs.oracle.com/javase/tutorial/jaxb/intro/

Pod tym adresami znajdziecie Państwo bardzo dokładny opis zagadnień związanych z obsługą XML w JAVA EE. Zachęcam do zapoznania.

Część Teoretyczna

XML i SOAP

Pierwsza część niniejszej instrukcji zawiera podstawowe wiadomości teoretyczne dotyczące języka XML, schematów XML i protokołu SOAP oraz wykorzystania ich do zdalnego wywoływania obiektów i przesyłania danych. Poznanie tych wiadomości umożliwi prawidłowe zrealizowanie praktycznej części laboratorium.

Tematyka laboratorium

Tematyką laboratorium jest programowanie aplikacji klient-serwer w oparciu o protokół SOAP. SOAP (ang. *Simple Object Access Protocol*) jest protokołem warstwy aplikacji modelu ISO/OSI standaryzowanym przez organizację W3C.
Protokół SOAP został opracowany w celu sprawnej komunikacji między aplikacjami w sieci Internet. Jest prostym i rozszerzalnym protokołem komunikacyjnym, niezależnym od platformy i języka programowania. SOAP umożliwia zdalny dostęp do obiektów (porównaj: RPC, CORBA, DCOM, DCOP, IPC, D-Bus). Wykorzystuje XML do kodowania wywołań, natomiast do ich przenoszenia najczęściej wykorzystywany jest protokół HTTP (możliwe jest też wykorzystanie protokołu RPC, SMTP i innych).
Komunikacja za pomocą protokołu SOAP odbywa się wg schematu zgłoszenie – odpowiedź (podobnie jak pobieranie stron internetowych za pomocą protokołu HTTP). Zgłoszenie i odpowiedź nazywamy komunikatem. Komunikat to dokument XML zbudowany wg poniższego schematu:

- *koperta* – określa szkielet opisujący, co znajduje się w komunikacie i jak go przetwarzać:
  - *nagłówek* – zbiór reguł kodujących potrzebnych do rozszyfrowania typów i struktur danych zdefiniowanych wewnątrz aplikacji,
  - *treść* – reguły (obiekt, metoda, parametry) dotyczące wywołania zdalnych metod i odczytu odpowiedzi.

Cel laboratorium

Celem laboratorium jest poznanie podstawowych własności języka XML i protokołu SOAP oraz ich praktycznego jego zastosowania. Podczas tego laboratorium zostanie:
1. Zbudowana prosta aplikacja w języku Java zapisująca dane do pliku XML oraz wczytująca dane z wcześniejszą walidacją poprawności dokumentu na podstawie pliku schematu XML.
2. Zbudowana aplikacja, która pokaże jak zapisać stan całych obiektów klas Java do pliku XML oraz wczytania tego stanu. Zostanie też wygenerowany plik XML Schema opisujący strukturę dokumentu przechowującego stan obiektu tej klasy.

Opis laboratorium

Język XML służy do przechowywania danych w formacie tekstowym i strukturze drzewa (struktura hierarchiczna). Język XML jest niezależny od platformy i języka programowania, co umożliwia wymianę dokumentów między heterogenicznymi systemami. Przykładowy plik XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="books" type="books"/>
<xs:complexType name="books">
<xs:sequence>
<xs:element name="book" type="book" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="book">
<xs:sequence>
<xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
<xs:element name="title" type="title" maxOccurs="unbounded"/>
<xs:element name="isbn" type="xs:string"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="title">
<xs:simpleContent>
<xs:extension base="xs:string">
<xs:attribute name="lang" type="xs:string" use="required"/>
```

```
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:schema>
```

Poprawny plik XML składa się z nagłówka określającego wersję języka XML zastosowanego w pliku i kodowania znaków. Atrybut `standalone="yes"` jest nieobowiązkowy i informuje, iż plik ten nie zawiera powiązań z innymi plikami.
Struktura danych w pliku XML składa się z elementów (ang. *elements*), które można porównać do gałęzi i liści w strukturze drzewa.
Każdy element rozpoczyna się znacznikiem początku elementu (ang. *tag*) np. `<book>` i kończy się znacznikiem końca elementu, w naszym przykładzie odpowiednio `</book>`. Wyjątkiem jest element pusty np. `<books />`. Jest to odpowiednik wyrażenia `<books></books>`. Między znacznikami początku i końca elementu mogą znajdować się dane tekstowe lub inne elementy.
Elementy mogą być zagnieżdżanie w sobie i występować w dowolnej ilości. Wyjątkiem jest element główny (ang. *root*), który musi występować tylko w jednym egzemplarzu.
Elementy muszą być poprawnie zagnieżdżone, tj. niedozwolona jest np. taka konstrukcja: `<book><title>Słownik</book></title>`, ponieważ element „`title`" nie jest prawidłowo zagnieżdżony w elemencie „`book`".
Elementy mogą zawierać atrybuty (ang. *attributes*). Definiowane są one w znaczniku początku elementu. Np. znacznik `<title lang="pl">` zawiera jeden atrybut o nazwie „`lang`" i wartości „`pl`". Nazwy atrybutów nie mogą się powtarzać w obrębie jednego znacznika.
Dane tekstowe i wartość atrybutów nie mogą zawierać znaków początku i końca znaczników. Znak mniejszości (<) należy zastąpić wyrażeniem `<` a znak większości (>) wyrażeniem `>`.Jeżeli potrzebujemy zapisać znak ampersand (&), stosujemy wyrażenie `&`.
Nazwy znaczników i nazwy atrybutów mogą składać się z liter, cyfr oraz znaków „-" (minus), „_" (podkreślenie) i „." (kropka). Nazwy nie mogą rozpoczynać się od cyfry, kropki ani minusa. Nie mogą też zaczynać się od „`xml`" (wielkość znaków dowolna).
Dokument XML może zawierać komentarze. Umieszcza się je pomiędzy wyrażeniami `<!--` i `-->`.
Dokument może zawierać sekcję danych znakowych, która nie będzie przetwarzana przez analizator składni. Wyrażeniem rozpoczynającym taką sekcję jest `<![CDA-TA[`, a wyrażeniem kończącym: `]]>`. Sekcja może posłużyć do zagnieżdżenia tekstu zawierającego znaki mniejszości i większości bez zamiany ich odpowiednio na `<` i `>`.
O dokumencie XML mówimy, że jest poprawny składniowo (ang. *well-formed*), jeżeli jest zgodny z regułami składni XML, tj. wszystkie znaczniki i ich atrybuty są poprawnie domknięte i nie zawierają znaków specjalnych w nazwie.

O dokumencie XML mówimy, że jest poprawny strukturalnie (ang. *valid*) jeżeli jest zgodny określoną wcześniej tzw. definicją dokumentu, czemu będzie poświęcony kolejny podrozdział. Zauważmy, że element „`books`" zawiera atrybut „`xmlns:xsi`" i „`xsi:noNamespaceSchemaLocation`". Pierwszy z nich określa format schematu XML, a drugi ścieżkę dostępu do pliku schematu. Przeważnie plik schematu publikuje się w Internecie, wtedy ścieżka jest adresem URL. My jednak skorzystamy z pliku dostępnego lokalnie.

Schematy XML
Schemat XML opisuje strukturę dokumentu XML. Określa jakie znaczniki mogą zostać użyte w dokumencie i z jakimi parametrami. Określa też jakie elementy w jakich elementach mogą zostać zagnieżdżane i w jakiej ilości.
Istnieją różne formaty zapisu schematu XML. Dwa najpopularniejsze to DTD i XML Schema. Format XML Schema jest nowszy od DTD i posiada znacznie większe możliwości. Ponadto do zapisu schematu wykorzystuje język XML. Pliki XML Schema mają zazwyczaj rozszerzenie `xsd`. Przykładowy plik `books.xsd`:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="books" type="books"/>
<xs:complexType name="books">
<xs:sequence>
<xs:element name="book" type="book" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="book">
<xs:sequence>
<xs:element name="author" type="xs:string" maxOccurs="unbounded"/>
<xs:element name="title" type="title" maxOccurs="unbounded"/>
<xs:element name="isbn" type="xs:string"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="title">
<xs:simpleContent>
<xs:extension base="xs:string">
<xs:attribute name="lang" type="xs:string" use="required"/>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:schema>
```

Powyższy plik odnosi się do pliku XML z przykładu zamieszczonego w poprzednim podrozdziale. Określa on, że element główny ma nazwę „books", który jest elementem złożonym (może zawierać elementy zagnieżdżone) i składa się z dowolnej ilości (od zera do nieskończoności: `minOccurs="0"` `maxOccurs="unbounded"`) elementów o nazwie „book".
Element „book" również jest elementem złożonym i składa się z co najmniej jednego elementu prostego „author" (`minOccurs` i `maxOccurs` domyślnie przyjmują wartość `1`), dokładnie jednego elementu „isbn" i co najmniej jednego elementu złożonego „title".
Element „title" co prawda przechowuje wartość prostą, ale posiada atrybut „lang", którego użycie jest wymagane (`use="required"`).
Dzięki dokumentom schematu XML możliwe jest wykorzystanie gotowych narzędzi do wstępnego zweryfikowania poprawności formatu danych w dokumencie.

Protokół SOAP
SOAP jest protokołem służącym do komunikacji między aplikacjami sieciowymi przez sieć rozległą WAN (np. Internet). Jest on protokołem niezależnym od platformy i języka programowania. Do kodowania wiadomości i ich przesyłu wykorzystuje istniejące technologie – odpowiednio XML i HTTP.
Jednymi z najistotniejszych własności protokołu SOAP są:
 Prosty w implementacji protokół zaprojektowany z myślą o komunikacji w Internecie z zastosowaniem istniejących technologii do komunikacji i kodowania danych.
 Do kodowania wykorzystywany jest język XML, który jest niezależny od języka programowania i architektury procesora. Ponadto dzięki wysokiej popularności języka XML, dostępna jest pełna gama bibliotek i narzędzi do przetwarzania dokumentów XML.
 Do komunikacji wykorzystywany jest najczęściej protokół HTTP/HTTPS, który jest protokołem bardzo popularnym w sieci Internet. Aplikacja kliencka może łączyć się z odległym serwerem bez konieczności zmiany konfiguracji firewalla, jeżeli wcześniej jego ustawienia umożliwiały użytkownikowi na przeglądanie stron internetowych. Ponadto zastosowanie protokołu HTTPS zapewnia bezpieczeństwo przesyłanych danych.

 Protokoły HTTP i HTTPS nie są jedynymi protokołami, które mogą posłużyć jako medium komunikacyjne. Za medium może posłużyć praktycznie każda technologia pozwalająca na przesyłanie danych. Z praktycznych przykładów można wymienić: JMS, SMTP i RPC.
 Minusem jest jednak często duży narzut komunikacyjny na rozmiar danych przesyłanych między aplikacjami. W szczególności nie nadaje się do przesyłania dużej ilości danych binarnych – rozwiązaniem tego problemu jest zastosowanie protokołu SwA (*SOAP with Attachments*).

Przedstawiony poniżej kod dokumentu SOAP przedstawia zlecenie wykonania funkcji `GetStockPrice` z parametrem `StockName` o wartości „`IBM`”.

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
<?xml version="1.0"?>

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
<soap:Header>
</soap:Header>
<soap:Body>
<m:GetStockPrice xmlns:m="http://www.example.org/stock">
<m:StockName>IBM</m:StockName>
</m:GetStockPrice>
</soap:Body>
</soap:Envelope>
```

Język XML w Java (przypomnienie)

Język Java wraz ze środowiskiem uruchomieniowym dostarczonym przez Oracle (dawniej Sun) dostarcza biblioteki ułatwiające parsowanie, edycję i walidację dokumentów XML. Do bardziej zaawansowanych zastosowań dostępna jest biblioteka JAXB, która umożliwia zapis stanu całych obiektów odpowiednio wcześniej przygotowanych klas do pliku XML i odtworzenie ich stanu.
W standardowej bibliotece Java SE sposób reprezentacji dokumentów XML odbywa się za pomocą modelu obiektowego DOM (*Document Object Model*).
Aby stworzyć plik DOM potrzebujemy najpierw uzyskać obiekt klasy fabryki budowniczych dokumentów (klasa `DocumentBuilderFactory` dostępna w pakiecie `javax.xml.parsers`). Obiekt wspomnianej klasy uzyskujemy za pomocą instrukcji:

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
```

W obiekcie klasy `DocumentBuilderFactory` można dokonać dodatkowych ustawień takich jak:

`dbf.setNamespaceAware(true);` – przetwarzanie przestrzeni nazw,
`dbf.setValidating(true);` – walidacja w oparciu o plik schematu XML.

Jeżeli decydujemy się na skorzystanie z walidacji, a schemat XML został zapisany w formacie pliku XML Schema, musimy o tym odpowiednio powiadomić obiekt fabrykę:

```
dbf.setAttribute( "http://java.sun.com/xml/jaxp/properties/schemaLanguage",
"http://www.w3.org/2001/XMLSchema");
```

Następnie musimy utworzyć obiekt budowniczego dokumentów (klasa `DocumentBuilder` z pakietu `javax.xml.parsers`):

```
DocumentBuilder db = dbf.newDocumentBuilder();
```

Obiekt klasy budowniczego dokumentów umożliwia nam utworzenie nowego, pustego dokumentu (metoda `newDocument()`) lub wczytanie istniejącego (metoda `parse(File plikXml)`). Metoda `parse` występuje w kilku wariantach, m.in. z możliwością podania adresu w postaci URI lub obiektu klasy `InputStream`.

Wartością zwracaną obu metod jest obiekt klasy `Document` z pakietu `org.w3c.dom`.

Nowe elementy tworzy się za pomocą metody `createElement(String name)` wykonanej na obiekcie klasy `Document`, która tworzy obiekt klasy `Element` z pakietu `org.w3c.dom`. Stworzony element nie zostaje jednak automatycznie dodany do dokumentu. Klasy `Element` i `Document` dziedziczą po klasie `Node` z pakietu `org.w3c.dom`, która z kolei udostępnia takie metody jak:

- `appendChild(Node node)` – dodanie elementu,
- `removeChild(Node node)` – usunięcie elementu,
- `getChildNodes()` – pobranie listy elementów,
- `getTextContent()`– pobranie danych tekstowych zapisanych w elemencie
- `setTextContent(String text)` – wprowadzenie danych tekstowych do elementu.

Natomiast klasa `Element` implementuje dodatkowo metody:

- `getTagName()` - pobranie nazwy znacznika,
- `getAttribute(String attrName)` – pobranie wartości atrybutu o nazwie attrName,
- `setAttribute(String attrName, String value)` – ustawienie wartości atrybutu,
- `hasAttribute(String attrName)` – sprawdzenie czy wartość atrybutu została wprowadzona,
- `removeAttribute(String attrName)` – usunięcie atrybutu,
- `getElementsByTagName(String name)` – pobranie listy elementów o podanej nazwie.

Należy pamiętać, że wszystkie elementy muszą być gdzieś umieszczone, a najwyższy w hierarchii element należy umieścić w obiekcie klasy `Document` za pomocą metody `appendChild`.

Powyższe informacje są wystarczające do wczytania i walidacji pliku XML. Do wyeksportowania modelu DOM do pliku XML potrzebna jest jeszcze klasa `Transformer` z pakietu `javax.xml.transform` wraz z klasami pomocniczymi. Do uzyskania obiektu klasy `Transformer` potrzebujemy wcześniej uzyskać obiekt fabryki `TransformerFactory` w następujący sposób:

```
TransformerFactory tf = TransformerFactory.newInstance();
```

Dopiero z obiektu fabryki możemy uzyskać obiekt klasy Transformer:

```
Transformer t = tf.newTransformer();
```

W obiekcie klasy Transformer trzeba jeszcze ustawić format dokumentu wyjściowego, w naszym przypadku XML:

```
t.setOutputProperty(OutputKeys.METHOD, "xml");
```

Jeżeli chcemy aby wygenerowany plik XML nie był w jednej linii tylko w formacie bardziej czytelnym dla człowieka, możemy ustawić parametr:

```
t.setOutputProperty(OutputKeys.INDENT,"yes");
```

Potrzebny jest nam jeszcze obiekt klasy `Source` z pakietu `javax.xml.transform`, który otrzymamy tworząc obiekt klasy po niej dziedziczącej `DOMSource` z pakietu `javax.xml.transform.dom`:

```
Source source = new DOMSource(doc);
```

Gdzie `doc` to obiekt klasy `Document`. W zapisie wyniku pośredniczy obiekt klasy `Result` z pakietu `javax.xml.transform`, który również uzyskamy tworząc obiekt klasy po niej dziedziczącej np. `StreamResult` z pakietu `javax.xml.transform.stream`:

```
Result result = new StreamResult(System.out);
```

W tym przypadku wynik zostanie wypisany na standardowe wyjście. Jednak aby się to stało trzeba jeszcze wykonać instrukcję:

```
t.transform(source, result);
```

Idea obiektów POJO (*Plain Old Java Object*) narodziła się z konieczności maksymalnego uproszczenia sposobu zapisu stanu obiektów i odtwarzania ich w innym systemie. Obiekty POJO są obiektami zwykłych klas Java. Jedyne co wyróżnia klasy ich obiektów to odpowiednie adnotacje, które informują jakie dane za pomocą jakich getterów powinny zostać pobrane z obiektu oraz za pomocą jakich *setterów* powinny zostać wprowadzone do nowo utworzonego obiektu tej klasy w niekoniecznie tym samym systemie i niekoniecznie zgodnej binarnie.
W Java do importu i eksportu stanu obiektów POJO do plików XML zastosujemy bibliotekę JAXB.

Aby klasa była zgodna z POJO musi spełniać następujące założenia:

1. Musi posiadać bezargumentowy publiczny konstruktor.

2. Biblioteka JAXB automatycznie wyszukuje właściwości oraz właściwości listowych do składowania w XML na podstawie publicznych właściwości klasy lub publicznych *getterów* i *setterów* o nazwach zgodnych z koncepcją:

```
public Klasa getWłaściwość(); public void setWłaściwość(Klasa value);
```

Natomiast dla list musi zostać zachowana koncepcja:

```
public List<Klasa> właściwość;
```

lub w przypadku getterów i setterów:

```
public List<Klasa> getWłaściwość(); public void setWłaściwość(List<Klasa>
value);
```

Za pomocą odpowiednich adnotacji można włączyć do przechowywania właściwość prywatną lub wyłączyć właściwość publiczną. Adnotacje umieszcza się przed właściwością klasy (jeżeli dostęp do niej ma być bezpośredni) lub przed getterem (jeżeli dostęp do niej odbywa się za pośrednictwem *getterów* i *setterów*).

Biblioteka JAXB wyszukuje wszystkie właściwości spełniające ww. reguły. Jeżeli jakiejś właściwości nie chcemy przechowywać w XML musimy opatrzyć ją adnotacją:

```
@XmlTransient
```

4. Jeżeli jakaś właściwość nie jest publiczna, a chcemy ją przechowywać w XML, musimy opatrzyć ją adnotacją `@XmlValue`, `@XmlAttribute` albo `@XmlElement`.

5. Adnotacja `@XmlValue` jest bezparametrowa. Właściwość opatrzona tą adnotacją będzie przechowywana jako wartość elementu w pliku XML (czyli to, co jest między znacznikami `<klasa>wartość</klasa>`). Tylko jedna właściwość w klasie może zawierać adnotację `@XmlValue`. Jeżeli jakaś właściwość w klasie zawiera adnotację `@XmlValue`, to wszystkie inne muszą być opatrzone adnotacją `@XmlAttribute`.

6. Adnotacja `@XmlElement` jest użyta jako domyślna dla wszystkich publicznych właściwości. Właściwość może być typu prostego lub złożonego albo listą typów prostych lub złożonych. Przyjmuje parametry:
a. `String defaultValue` – wartość domyślna, jeżeli w pliku XML nie została wprowadzona.

b. `String name` – nazwa znacznika elementu.
c. `boolean required` – element jest wymagany, jeżeli `true` to w pliku XML Schema element będzie oznaczony atrybutem `minOccurs="1"` lub `minOccurs="0"` jeżeli wartość parametru wynosi `false`. Natomiast w obu przypadkach `maxOccurs="1"` dla właściwości pojedynczej, natomiast dla listy `maxOccurs="unbounded"`.

7. Adnotacja `@XmlAttribute` oznacza, że właściwość w pliku XML będzie reprezentowana jako atrybut elementu reprezentowanego przez klasę. Właściwość taka musi być typu prostego, nie może być też listą. Przyjmuje parametry:
a. `String name` – nazwa atrybutu w XML.
b. `boolean required` – atrybut jest wymagany, w XML Schema będzie opatrzony atrybutem `use="required"`.

8. Klasa której element będzie korzeniem pliku XML, musi zawierać adnotację:
`@XmlRootElement`
Adnotacja ta przyjmuje parametr `String name`, który określa nazwę znacznika, pod którym będzie reprezentowany obiekt tej klasy, np.:
`@XmlRootElement(name="books")`
Domyślnie parametr `name` przyjmuje wartość taką, jak nazwa klasy z przekonwertowaniem pierwszego ciągu wielkich liter na małe.
Nie może być dwóch elementów o takiej samej nazwie. W szczególności należy uważać na taką konstrukcję:

```
@XmlElement(name="inaczej")
private String costam;
public String getCostam() {
return costam;
}

public void setCostam(String costam) {
this.costam = costam;
}
```

Biblioteka JAXB użyje właściwości `costam` z bezpośrednim dostępem do niej, a potem jeszcze raz spróbuje użyć właściwości dostępnej za pośrednictwem *getterów* i *setterów* i użyje dla niego domyślnego znacznika „`costam`”. Wczytanie takiej klasy spowoduje nadpisanie jednej wartości drugą. W tym przypadku autor za pewne chce, by biblioteka JAXB wprowadzała dane do klasy za pomocą *getterów* i *setterów*, więc adnotacja `@XmlElement` powinna zostać przeniesiona przed metodę `getCostam()`. Natomiast jeżeli autor chce, by użytkownik używał wartości za pomocą *getterów* i *setterów*, a biblioteka JAXB miała dostęp bezpośredni, powinien przed `getCostam()` wstawić adnotację `@XmlTransient`.

Aby wygenerować plik schematu XML, musimy najpierw utworzyć kontekst JAXB:
`JAXBContext context = JAXBContext.newInstance(Books.class);`

Klasa `Books` jest klasą, której element w pliku XML ma być korzeniem. Metoda odpowiedzialna za wygenerowanie pliku XML Schema nazywa się `generateSchema` i przyjmuje jako parametr obiekt klasy `SchemaOutputResolver`. Klasa ta jest klasą abstrakcyjną, którą musimy przedefiniować. Jeżeli nasz plik `.xsd` nie będzie niczego importował z innych plików, możemy posłużyć się konstrukcją:

```
final File outFile = new File("books.xsd");
context.generateSchema(new SchemaOutputResolver() {
```

```
@Override
public Result createOutput(String string, String string1)
throws IOException {
StreamResult result = new StreamResult(outFile);
result.setSystemId(outFile.toURI().toURL().toString());
return result;
}
});
```

Do zapisu stanu klasy do pliku XML wykorzystuje się klasę `Marshaller`. Obiekt tej klasy tworzy się z obiektu kontekstu JAXB w następujący sposób:

```
Marshaller m = context.createMarshaller();
```

Ustawienie parametru `Marshaller.JAXB_FORMATTED_OUTPUT` na `true` spowoduje, że wygenerowany plik XML będzie sformatowany za pomocą wcięć co uczyni go bardziej czytelnym dla człowieka. Parametr ten ustawia się za pomocą instrukcji:

```
m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
```

Skoro wygenerowaliśmy już plik `books.xsd`, warto go wykorzystać dopisując go w parametrze `Marshaller.JAXB_NO_NAMESPACE_SCHEMA_LOCATION`.

Zapis stanu obiektu klasy dokonuje się w chwili wywołania instrukcji:

```
m.marshal(books, System.out);
```

Gdzie `books` jest obiektem klasy stanowiącej korzeń. Wynik zostanie wypisany na ekran. Metoda `marshal` posiada wiele wariantów drugiego parametru. Między innymi przyjmuje obiekt klasy `File`. Do tworzenia obiektu klasy ze stanu zapisanego w pliku XML służy klasa `Unmarshaller`. Obiekt tej klasy tworzy się z obiektu kontekstu JAXB w następujący sposób:

```
Unmarshaller u = context.createUnmarshaller();
```

Klasa `Unmarshaller` dokonuje walidacji na podstawie adnotacji. Możemy ją jednak zmusić do dodatkowej walidacji na podstawie podanego przez nas pliku XML Schema. Dla pliku `books.xsd` możemy wykonać taką instrukcję:

```
u.setSchema(SchemaFactory.newInstance( XMLConstants.W3C_XML_SCHEMA_NS_URI)
.newSchema(new File("books.xsd")));
```

Ma to jednak sens jedynie wtedy, kiedy dokonaliśmy ręcznych modyfikacji pliku `books.xsd` i muszą one zostać uwzględnione.
Za utworzenie nowego obiektu ze stanu zapisanego w pliku XML odpowiada metoda `unmarshal`. Ma ona wiele wariantów, w tym dla obiektów klasy `InputStream` i `File`. Jeżeli stan chcemy wczytać z pliku `books.xml`, musimy wywołać instrukcję:

```
Books books = (Books) u.unmarshal(xmlFile);
```

Część praktyczna

Druga część instrukcji zawiera zadania do praktycznej realizacji, które demonstrują zastosowanie technik z omawianego zagadnienia.

Tutoriale do przerobienia:
http://www.mkyong.com/java/jaxb-hello-world-example/
http://www.vogella.com/tutorials/JAXB/article.html

Zadanie 1. Zapis pliku XML w Java
Celem zadania jest zaimplementowanie programu, który stworzy plik `books.xml` i zapisze w nim informacje o wprowadzonych książkach. Wygenerowany plik `books.xml` musi być zgodny z dokumentem schematu XML zapisanym w pliku `books.xsd`
Należy pamiętać, że książka musi mieć co najmniej jednego autora i tytuł w co najmniej jednym języku.
Wszystkie informacje na temat książek program ma pobierać ze standardowego wejścia według następującego algorytmu:

1. Wypisz: „Czy wprowadzić nową książkę?".

2. Jeżeli wprowadzono „nie" zakończ algorytm i zapisz dane na dysku.

3. Wypisz: „Podaj autora książki" i pobierz odpowiedź.

4. Jeżeli wprowadzona odpowiedź nie jest linią pustą to przejdź do pkt. 5.

5. Wypisz: „Podaj język tytułu" i pobierz odpowiedź.

6. Wypisz: „Podaj tytuł w tym języku" i pobierz odpowiedź.

7. Jeżeli odpowiedzi z pkt. 5 i 6 nie są puste to przejdź do pkt. 8.

8. Wypisz: „Podaj numer ISBN książki" i pobierz odpowiedź.

9. Wypisz: „Czy dane są poprawne?" i pobierz odpowiedź.

10. Jeżeli wprowadzono „tak" - dodaj książkę do drzewa XML i przejdź do pkt. 1.

Program powinien być tak napisany, aby możliwe było podanie więcej niż jednego autora, oraz tytułu w kilku językach. Dane te powinny następnie zostać odzwierciedlone w pliku XML.

Zadanie 2. Odczyt pliku XML z walidacją w Java

Celem zadania jest zaimplementowanie programu, który wczyta stworzony w poprzednim zadaniu plik books.xml, dokona jego walidacji za pomocą pliku books.xsd, następnie wypisze informacje o zapisanych książkach w formacie:

Tytuł w języku {title[1].lang}: {title[1]}

Tytuł w języku {title[2].lang}: {title[2]}

…

Tytuł w języku {title[n].lang}: {title[n]}

Autorzy: {author[1]}; {author[2]}; …; {author[n]}

ISBN: {isbn}

/linia pusta i następna książka.../

Oznaczenia w klamerkach:

title[n] – n-ty wariant tytułu

title[n].lang – język n-tego wariantu tytułu

author[n] – n-ty autor

isbn – numer ISBN

Zadanie 3. Biblioteka JAXB i obiekty POJO

Napisać program wykonujący identyczne zadanie co program z zadania 1, ale z wykorzystaniem biblioteki JAXB i obiektów POJO. Program musi w odpowiedni sposób implementować klasy Books (korzeń pliku XML), Book i Title (tytuł książki z informacją o języku).

Należy pamiętać, że elementy book, author oraz title są wielokrotne. Ponadto elementy author oraz title są wymagane w ilości co najmniej 1.

Do wprowadzania danych ma być użyty identyczny algorytm jak w zadaniu 1. Program w chwili uruchomienia ma sprawdzić czy w lokalnym katalogu plik books.xml istnieje. Jeżeli tak, to powinien go wczytać. Nowe książki mają być dopisywane do tego pliku.

Zadanie 4. Stwórz dokument XML zawierający przykładowe dane w postaci odpowiednich powiązanych ze sobą elementów tworzących listę pracowników, oraz definicję XML Schema dla tego dokumentu. Rozwiązanie powinno ponadto posiadać następujące własności:

- Element pracownik powinien mieć podelement lub atrybut "okresZatrudnienia", mogący przybierać wartość typu odcinek czasu (duration)albo łańcuch tekstowy "bezterminowo". Odpowiedni typ powinien zostać zadeklarowany jako typ nazwany "okresZatrudnienia"

- Element pracownik powinien posiadać podelement "email" opisany typem, który zapewni przynajmniej zgrubną (obecność symbolu @ i co najmniej 2 segmenty nazwy za nią) walidację

- Element pracownik posiada podelement lub atrybut "identyfikator"

- Element pracownik posiada zero lub więcej podelementów "podwladny", zawierający wartość identyfikatora innego pracownika. Sprawdź, czy opisane w wykładzie środki dla deklarowania integralności referencyjnej pozwalają na zbudowanie definicji klucza obcego, który w tym wypadku

jest wielowartościowy (realizuje powiązanie "do-wielu"). Jeśli tak, dokonaj odpowiedniej deklaracji określającej zastosownie elementu "podwladny"

- Dokument wraz z jego definicją schematu powinien być pomyślnie sprawdzony pod kątem jego poprawności strukturalnej.