Aufgabe "Absenzenmeldung"

# Gegeben ist folgendes Formular

|  |  |  |
| --- | --- | --- |
| **Absenzen Meldung** | | |
| Anrede | Herr | Frau | Nichtzutreffendes streichen |
| Personal Nr. |  | Beispiel: A-123-9 |
| Vorname |  |  |
| Nachname |  |  |
| Abteilung |  | Beispiel: Entwicklung1 |
| Datum |  | Beispiel: 22.11.2014 |
| Zeitraum | Ganzer Tag | Vormittag | Nachmittag | Nichtzutreffendes streichen |
| Bemerkung |  |  |

Dieses Formular soll durch eine Softwarelösung abgelöst werden.

Die Felder unterliegen folgenden Konventionen:

**Anrede**: Auswahl *Herr* oder *Frau*

**Personal Nr.**: Format [1 Zeichen][1 Bindestrich][3 Zahlen][1 Bindestrich][1 Zahl], Beispiel A-123-8

**Vorname**: min. 2, max. 100 Zeichen

**Nachname**: min. 2, max. 100 Zeichen

**Abteilung**: min. 4, max. 20 Zeichen

**Datum**: Kein zwingendes Format

**Zeitraum**: Auswahl *Ganzer Tag*, *Vormittag*, *Nachmittag*

Das Feld "**Bemerkung**" ist das einzige Feld welches optional ist, d.h. es muss nicht zwingend vorhanden sein. Wenn es vorhanden ist darf das Feld maximal die Länge von 200 Zeichen haben.

# Aufgabe

1. Erstelle ein XML-Schema welches die obigen Felder abbildet und den entsprechenden Konventionen folgt.
2. Erstelle ein XML-File welches gegenüber dem Schema erfolgreich validiert werden kann.
3. Erstelle mittels Java einen XML-Reader der das XML-File einlesen kann. Die dabei entstehenden Daten sollen in ein Daten-Objekt abgelegt werden, welches das untenstehende Interface implementiert.  
   Technologie: **SAX (Version 2.0)**
4. Erstelle in Java einen XML-Reader der das XML-File einlesen kann. Die dabei entstehenden Daten sollen in ein Daten-Objekt abgelegt werden (dasselbe wie bei Punkt 3), welches das untenstehende Interface implementiert.  
   Technologie: **DOM**
5. Für beide Parser sollen mittels **JUnit** entsprechende Test-Klassen entstehen welche das korrekte Verhalten mittels **unterschiedlichen Szenarien** überprüft.

## Eingesetzte Technologien

* Java JDK 1.8 oder höher
* JUnit 4.11 oder höher
* Eclipse Entwicklungsumgebung (Versionen: Luna, Mars)
* Code-Konvention
  + Sun/Oracle Code Conventions
  + <http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html>
  + Ausnahme Zeilenlänge max. 200 Zeichen

## Abgabe

Abgegeben soll ein ZIP-File in dem alle notwendigen XML-Dateien, XML-Schema, Klassen und Libraries (z.B. junit.jar) enthalten sind. Es kann das Eclipse-Projekt in das ZIP gepackt werden.

# Java-Klassen



# Informationen

## Das Document Object Model (DOM)

DOM ist eine Entwicklung des W3C und wird von vielen Programmiersprachen unterstützt. Das Standard-DOM ist so konzipiert, dass es unabhängig von einer Programmiersprache ist und eine strikte Hierarchie erzeugt. DOM definiert eine Reihe von Schnittstellen, die durch konkrete Programmiersprachen implementiert werden.

* Offizielle Dokumentation von Oracle zum Thema DOM für Java: <http://docs.oracle.com/javase/tutorial/jaxp/dom/index.html>
* Wikipedia zu DOM: <http://de.wikipedia.org/wiki/Document_Object_Model>

## Simple API for XML Parsing (SAX)

SAX ist zum schnellen Verarbeiten der Daten von David Megginson als Public Domain entworfen worden. SAX ist im Gegensatz zu DOM nicht so speicherhungrig, weil das XML-Dokument nicht vollständig im Speicher abgelegt ist, und daher auch für sehr große Dokumente geeignet. Da SAX auf einem Ereignismodell basiert, wird die XML-Datei wie ein Datenstrom gelesen, und für erkannte Elemente wird ein Ereignis ausgelöst. Dies ist aber mit dem Nachteil verbunden, dass wahlfreier Zugriff auf ein einzelnes Element nicht ohne Zwischenspeicherung möglich ist.

* Offizielle Dokumentation von Oracle zum Thema SAX für Java: <http://docs.oracle.com/javase/tutorial/jaxp/sax/index.html>
* Wikipedia zu SAX: <http://de.wikipedia.org/wiki/Simple_API_for_XML>

Zu **beiden Technologien** finden sich hier Tutorials in Deutsch:

* <http://www.torsten-horn.de/techdocs/java-xml.htm>
* <http://www.uzi-web.de/parser/parser_toc.htm>