Poradnik jak zdać XML'a od debila dla debila

Autor: PuckMoment, dgradowski i takie tam inne moje osobowości

Treść kolosa (cała treść to tylko moja wyobraźnia, a zbieżność z jakąkolwiek wersją kolosa z poprzednich lat jest przypadkowa, panie dziekanie proszę mnie nie zamykać w więzieniu, okok?):

Jesteś pracownikiem firmy ... jakieś tam pierdolenie, opis struktury zrobić trza.
Firma otrzymuje dokument od klientów w następującej postaci:

```
<CarModels>
    <Brands>
        <Brand id=”b01”>Alfa Romeo</Brand>
        <Brand id=”b02”>BMW</Brand>
        <Brand id=”b03”>Audi</Brand>
    </Brands>
    <Car brand=”b01”>
        <price currency="EURO">165.00</price>
        <catalogNumber>AU.34</catalogNumber>
        <type>
            <name>Spider 3</name>
            <color>white</color>
            <size>1:18</size>
        </type>
    </Car>
    <Car brand="b03">
        <price currency="EURO">95.70</price>
        <catalogNumber>AU.123</catalogNumber>
        <type>
            <name>AB L</name>
            <year>2017</year>
        </type>
    </Car>
</CarModels>
```

A teraz zadania:

1. (1 pkt) Zapisz przy użyciu DTD definicje elementu elementu (tylko, bez atrubutu) *Car* (element *price* występuje minimum raz, element catalogNumber ma wystąpić dokładnie raz; element type może nie wystąpić wcale lub raz).

```
<!ELEMENT Car (price+, catalogNumber, type?)>
```

Każda definicja ELEMENTU wygląda bardzo podobnie:

a. Zaczynamy od <!ELEMENT
b. Następnie jest nazwa elementu, w tym przypadku Car
c. Później w nawiasie jest zapisana informacja o tym co zawiera dany element.
d. I na końcu zamykamy tag za pomocą „>”

Każdy taki element może zawierać w sobie inne elementy oraz tekst.

Jeśli element zawiera w sobie tylko tekst wygląda on następująco:

```
<!ELEMENT catalogNumber (#PCDATA)>
```

Dla każdej zawartości określonej dla elementu możemy określić ilość jego wystąpień, aby to zrobić korzystamy ze znaków:

- „?” – element jest opcjonalny i może nie występować wcale
- „+” – element występuje co najmniej raz ale może wystąpić wiele razy, (od 1 do ∞)
- „*” – jest opcjonalny ale może wystąpić wiele rsazy, (od 0 do ∞)

Jak po elemencie nie ma symbolu to oznacza, że musi wystąpić dokładnie raz.

W elementach może występować coś takiego jak sekwencja (a,b) bądź alternatywa (a|b). Mamy następujący przykład:

```
<!ELEMENT elemencik ((A, B?) | (B, B+) | (A, A*))>
```

Posiadamy tutaj alternatywę 3 sekwencji, możliwe alternatywy są następujące:

1) Element A i element B, który jest opcjonalny
2) Element B i element B, których jest 1 bądź więcej
3) Element A i element A, których jest 0 bądź więcej

Jeśli chcielibyśmy by elemencik miał w sobie od 2 do 5 elementów A, moglibyśmy określić to tak:

```
<!ELEMENT elemencik (A, A, A?, A?, A?)>
```

2. (2 pkt) Zapisz przy użyciu XML Schema definicję elementu price.

Szybkie przypomnienie jak wygląda element price

```
        <price currency="EURO">95.70</price>
```

W XML Schema najbardziej kluczowe jest ustalanie typów. Są dwa rodzaje typów

- simpleType – dla elementów, które nie maja w sobie zagnieżdżonych innych elementów
- complexType – dla elementów, które w sobie posiadają jeszcze inne elementy

Istnieją również 2 rodzaje kontentu w tagu:

- simpleContent – kiedy, zawartość jest taka jak w simpleType
- complexContet – kiedy, zawartość jest taka jak w complexType

Aby stworzyć znacznik price musimy określić typ zawartości wewnątrz znacznika jak i atrybutu currency. Można to zrobić albo tworząc jedną długą definicje znacznika albo podzielić sobie to na 2 typy

Aby ustalić typ dla waluty należy napisać coś takiego

```
    <xsd:simpleType name="currencyType">
        <xsd:restriction base="xsd:token">
            <xsd:enumeration value="PLN"/>
            <xsd:enumeration value="EURO"/>
            <xsd:enumeration value="USD"/>
        </xsd:restriction>
    </xsd:simpleType>
```

Taki oto typ daje nam ograniczenie (restriction), że są w danym typie mogą wystąpić tylko „PLN”, „EUR” i „USD”. Jako baze naszego typu ustalamy token, bo to taki lepszy string bez whitespace’ów oraz wszystkie wartości naszej enumeracji to tak naprawdę teksty. Początek „xsd:” oznacza, że odnosimy się do konkretnego namespace’a. W prawdziwym pliku to zależy od tego jak sobie zlinkujemy ten namespace w naszym kodzie na początku pliku, na kolosie polecam jednak używać xsd (XML Schema Definitions).

Aby ustalić typ dla samej ceny możemy to zrobić w takie sposób:

```
    <xsd:simpleType name="priceType">
        <xsd:restriction base="xsd:decimal">
            <xsd:fractionDigits value="2"/>
            <xsd:minExclusive value="0"/>
        </xsd:restriction>
    </xsd:simpleType>
```

<xsd:fractionDigits> określa ile jest liczb po przecinku
<xsd:minExclusive> określa minimalną wartość z wyłączeniem tej wartości, w tym przypadku oznacza to cena > 0

Teraz utworzone typy wystarczy tylko wykorzystać.

```
<xsd:element name="price">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="priceType">
                <xsd:attribute name="currency" type="currencyType"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
```

3. O patternach słów kilka

Było następujące zadanie:

Zapisz pyrz użyciu XML Schema definicję nazwanego typu prostego opisującego numer katalogowy, w formacie:

LiteraLiteraKropkaLiczbaLiczba

lub

LiteraLiteraKropkaLiczbaLiczbaLiczba

```
<xsd:simpleType name="numerKat">
    <xsd:restriction base="xsd:token">
        <xsd:pattern value="[A-Z]{2}[.][0-9]{2}"/>
        <xsd:pattern value="[A-Z]{2}[.][0-9]{3}"/>
    </xsd:restriction>
</xsd:simpleType>
```

Aby można było wpisać również małe litery, musimy napisać tak:

```
<xsd:simpleType name="numerKat">
    <xsd:restriction base="xsd:token">
        <xsd:pattern value="[A-Za-z]{2}[.][0-9]{2}"/>
        <xsd:pattern value="[A-Za-z]{2}[.][0-9]{3}"/>
    </xsd:restriction>
</xsd:simpleType>
```

Aby zapisać to w postaci tylko jednego patternu możemy zapisać tak

```
<xsd:simpleType name="numerKat">
    <xsd:restriction base="xsd:token">
        <xsd:pattern value="[A-Za-z]{2}[.][0-9]{2,3}"/>
    </xsd:restriction>
</xsd:simpleType>
```

Teraz by go użyć piszemy:

```
<xsd:element name="catalogNumber" type="numerKat"/>
```

4. (2 pkt) Zapisz przy użyciu DTD oraz XML Schema definicję atrubutu id dla elementu Brand który jest kluczem elementu Brand
   DTD:
   <!ATTLIST Brand id ID # REQUIRED>

   XML Schema:

```
    <xsd:element name="brand">
        <xsd:complexType>
            <xsd:simpleContent>
                <xsd:extension base="xsd:string">
                    <xsd:attribute name="id">
                        <xsd:simpleType>
                            <xsd:restriction base="xsd:token">
                                <xsd:pattern value="[b][0-9]{2}"/>
                            </xsd:restriction>
                        </xsd:simpleType>
                    </xsd:attribute>
                </xsd:extension>
            </xsd:simpleContent>
        </xsd:complexType>
    </xsd:element>

    <xsd:key name="brandKey">
            <xsd:selector xpath="Brands/Brand"/>
            <xsd:field xpath="@id"/>
    </xsd:key>
```

5. Zadanie z CSS

```
price[currency]::after {
   content: attr(currency);
   color: red;
}
```

Ten kod CSS spowoduje, po zawartości price zostanie wyświetlona wartość atrybutu currency, ale tylko gdy atrybut currency występuje.

Aby dla każdej wartości był inny kolor, możemy zapisać to w taki sposób

```
price[currency="PLN"]::after {
    content: attr(currency);
    color: red;
}

price[currency="USD"]::after {
    content: attr(currency);
    color: green;
}

price[currency="EURO"]::after {
    content: attr(currency);
    color: blue;
}
```

6. XSLT 1.0 vs XSLT 2.0
   W XSLT 1.0 będzie tylko pierwszy wynik a w XSLT 2.0 będą wszystkie znalezione wyniki zapisywane ze spacją odstępu
7. Napisz instrukcję XSLT, która wyświetli sumę wartości zaokrągloną do góry dla wszystkich elementów prize i wyświetli wyniki w PLN, należy przyjąć przelicznik dla kwot w EURO równym 4,33.

```
<xsl:template match="/">
    <xsl:variable name="sumPrice" select="sum(//price[@currency='EURO'])">
    <xsl:variable name="sumPricePLN" select="$sumPrice * 4.33">
    Wartosc w EURO: <xsl:value-of select="celling($sumPrice)"/>
    Wartosc w PLN: <xsl:value-of select="celling($sumPricePLN)"/>
</xsl:template>
```