Architektura zorientowana na usługi											
3	Temat:	***XML, XSD, XSLT***	Zadania:								Data:
	Autor:	**Sylwia Kaleta**	1	2	3	4	5	6	7	8	**17 X 2018**
	Autor:	**Kamil Wanat**	H	E	E	E	H	M	-	-	**18:00-19:30**

Zadanie 1. Tworzenie XML z klasy

W tym zadaniu należało w dowolnym języku zaimplementować klasy, wedle specyfikacji podanej w treści zadania podane przez prowadzącego, w naszym wypadku był to język Python:
Zaimplementowane klasy:
W następnym kroku należało w miarę możliwości użyć mechanizmu do serializacji danych w celu stworzenia pliku XML z klasy, niestety jedynym mechanizmem do serializacj danych, jaki udało nam się znaleźć dla języka Python był mechanizm umożliwiający zapis słownika, a nie całego obiektu, dlatego też sami zaimplementowaliśmy potrzebne metody:
Zapis klasy Osoba jako XML:

```
def toXml(self):
    if self.typOsoby == TypOsoby.NAUCZYCIEL:
        typ = "NAUCZYCIEL"
    else:
        typ = "UCZEN"
    xml = "\n".join(["\t\t\t<typOsoby>"+typ+"</typOsoby>",
                     "\t\t\t<imie>" + self.imie + "</imie>",
                     "\t\t\t<nazwisko>" + self.nazwisko + "</nazwisko>",
                     "\t\t\t<rokUrodzenia>" + str(self.rokUrodzenia) +
"</rokUrodzenia>"])
    return xml
```

Zapis klasy Klasa jako XML:

```
def toXml(self):
    xml = "\n".join(["<klasa>",
                     "\t<numer>"+str(self.numer)+"</numer>",
                     "\t<litera>" + str(self.litera) + "</litera>",
                     "\t<numer>" + str(self.numer) + "</numer>",
                     "\t<wychowawca>",
                     self.wychowawca.toXml(),
                     "\t</wychowawca>",
                     "\t<uczniowie>"])
    for ucz in self.uczniowie:
        xml = "\n".join([xml,
                         "\t\t<uczen>",
                         ucz.toXml(),
                         "\t\t</uczen>"])
    xml = "\n".join([xml, "\t</uczniowie>"])
    return xml
```

Odczyt klasy z pliku XML wykonany został z użyciem biblioteki xml.etree.ElementTree. Wyszukiwane były kolejne wartości z pliku XML, a następnie zapisywany do obiektu typu Klasa.

```
klasa = Klasa()
klasa.numer = tree[0].text
klasa.litera = tree[1].text

wychowawca = Osoba()
wychowawca.typOsoby = TypOsoby.NAUCZYCIEL
wychowawca.imie = tree[2][1].text
wychowawca.nazwisko = tree[2][2].text
wychowawca.rokUrodzenia = tree[2][3].text
klasa.wychowawca = wychowawca
```

```
for student in tree.findall('uczniowie/uczen'):
    uczen = Osoba()
    uczen.typOsoby = TypOsoby.UCZEN
    uczen.imie = student.find('imie').text
    uczen.nazwisko = student.find('nazwisko').text
    uczen.rokUrodzenia = student.find('rokUrodzenia').text
    klasa.uczniowie.append(uczen)
```

Zadanie 2. Tworzenie XSD z klasy

W tym zadaniu użyliśmy zewnętrznego narzędzia aby stworzyć plik XSD na podstawie klasy, stworzony plik XSD znajduje się w archiwum z kodem źródłowym pod nazwą klasa.xsd

Zadanie 3. Walidacja XML

Celem zadania była walidacja pliku XML na podstawie pliku XSD, w tym celu wykorzystaliśmy bibliotekę xmlschema, która automatycznie przeprowadza walidację pliku:

```
mySchema = xmlschema.XMLSchema(xsdFilePath)
print("sprawdzanie poprawnosci:")
print(mySchema.is_valid(xmlFilePath))
```

xsdFilePath – ścieżka do pliku XSD
xmlDilePath – ścieżka do pliku XML

Zadanie 4. Odczyt i zapis XML

Zadanie polegało na dodaniu metody pozwalającej na odczyt pliku XML, wypisanie danych dotyczących klasy oraz sprawdzenie, czy dane osób są zgodne z obowiązującymi regułami. Zadanie wykonane zostało przy użyciu biblioteki xml.etree.ElementTree. Listing przedstawia część odpowiedzialną za wypisanie informacji dotyczących klasy.

```
xml = ElementTree.parse(xmlPath)
tree=xml.getroot()
print("Klasa: "+tree[0].text+tree[1].text)
print("WYCHOWAWCA: "+tree[2][1].text+" "+tree[2][2].text)
print("UCZNIOWIE:")
for student in tree.findall('uczniowie/uczen'):
    print(student.find('imie').text+" "+student.find('nazwisko').text)
```

Zadanie 5. XPath

Do wykonania tego zadania użyta została biblioteka lxml pozwalająca na łatwe użycie XPath. W programie zaimplementowano dwie metody. Pierwsza pozwala na wyszukiwanie uczniów poprzez datę urodzenia, druga poprzez imię i nazwisko. Poniżej kod metody wyszukującej uczniów przez imię i nazwisko:

```
def findByName():
    print("Podaj sciezke do pliku xml:")
    xmlPath = input()
    print("Podaj imie ucznia:")
    imie = input()
    print("Podaj nazwisko ucznia:")
    nazwisko = input()
    path = '/klasa/uczniowie/uczen[imie="' + imie + '"][nazwisko="' +
nazwisko + '"]'
    tree = lxml.etree.parse(xmlPath)
    students = tree.xpath(path)
    for e in students:
        print("Imie: " + e.find("imie").text)
        print("Nazwisko: " + e.find("nazwisko").text)
        print("Rok urodzenia: " + e.find("rokUrodzenia").text)
```

Zadanie 6. XSLT

Plik XSLT wymagany w tym zadaniu został przygotowany w oparciu o szablon ze strony w3schools.com. Następnie przy pomocy biblioteki lxml zostały wczytane pliki xml oraz xslt. Kolejnym krokiem była translacja pliku XML na HTML przy pomocy pliku XSLT. Kod metody zamieszczony został poniżej.

```
def makeHtmlFile():
    print("Podaj sciezke do pliku xml:")
    xmlPath = input()
    print("Podaj sciezke do pliku xslt:")
    xsltPath = input()
    xml=lxml.etree.parse(xmlPath)
    xslt=lxml.etree.parse(xsltPath)
    transform = lxml.etree.XSLT(xslt)
    data=transform(xml)
    file=open('klasa.html','w')
    data=str(data)
    file.write(data)
    file.close()
```

Podsumowanie

Wykonanie zadań przy pomocy języka Python pozwoliło nam na szybkie wykonanie zadań. Było to możliwe dzięki mnogości bibliotek utworzonych dla tego języka które są łatwo dostępne w sieci internet. Oczywiście pojawiły się pewne trudności takie jak: brak mechanizmu serializacji/deserializacji (który jednak mógł być łatwo zastąpiony innymi metodami z bibliotek zewnętrznych). Brak skonkretyzowanych informacji oraz przykładów dotyczących mechanizmu XPath również wydłużył czas wykonywania zadania.