Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Кафедра інтелектуальних програмних систем

Лабораторна робота №1

“ Представлення даних у форматі XML”

Виконав студент 3-го курсу

Групи ІПС-32

Грушевий Єгор Вікторович

2022

**Постановка задачі**

Розробити програму, що забезпечує введення і редагування інформації про об'єкти відповідно до заданої предметної області. Інформація про об'єкти повинна зберігатися в окремому файлі в форматі XML.

**Варіант 3**

Предметна область Відділ кадрів

Об'єкти Підрозділи, Співробітники

Примітка Існує множина підрозділів підприємства.

У кожному підрозділі працює множина

співробітників

Перевірка DTD

структури

документа XML

**Вихідний код програми**

from lxml import etree as ET

import colorama

from colorama import init, Fore

from enum import Enum

import os.path

import sys

from io import StringIO

init(autoreset=True)

helpDict = {

"help": "print help info",

"new [FILE]": "create xml file",

"load [FILE]": "load xml file",

"save": ("save file", {"-f[FILE]": "save to specified file"}),

"add unit": ("add new unit", {"-n[NAME]": "unit name"}),

"add employee": ("add new employee", {"-n[NAME]": "employee name", "-u[ID]": "unit id", "-r[RANK]": "employee's rank", "-e[EXPERIENCE]": "employee's experience"}),

"update unit [UNIT ID]": ("update unit info", {"-n[NAME]": "update name"}),

"update employee [EMPLOYEE ID]": ("update employee's info", {"-n[NAME]": "update name", "-u[ID]": "update unit id", "-r[RANK]": "update rank", "-e[EXPERIENCE]": "update experience"}),

"delete unit [UNIT\_ID]": "delete unit",

"delete employee [EMPLOYEE\_ID]": "delete employee",

"info unit [UNIT\_ID]": "print unit info",

"info employee [EMPLOYEE\_ID]": "print employee info",

"all units": "print all valid unit ids",

"all employees": "print all valid employee ids",

"stat": ("print statistics", {"-u": "print units order by number of employees", "-r": "print employees order by rank", "-e": "print employees order by experience"}),

"q": "exit"

}

def printHelpCommand(key):

if type(helpDict[key]) == str:

print("\t", key, " - ", helpDict[key])

else:

print("\t", key, " - ", helpDict[key][0])

print("\t\tOptions:")

nestedDict = helpDict[key][1]

for nestedKey in nestedDict:

print("\t\t", nestedKey, " - ", nestedDict[nestedKey])

def printHelpInfo():

for key in helpDict:

printHelpCommand(key)

def INFO(message):

print(Fore.GREEN + message)

def ERROR(message):

print(Fore.RED + message)

def WARNING(message):

print(Fore.CYAN + message)

def DEBUG(message):

if isDebug:

print(Fore.YELLOW + message)

def yesNoDialog():

while True:

answer = input()

if answer == "y": return True

elif answer == "n": return False

else: print("Please, answer y or n")

def parseOptions(args):

options = dict()

for arg in args:

if arg.startswith('-'):

options[arg[1]] = arg[2:]

return options

class State(Enum):

EMPTY = 0,

UNSAVED\_CHANGES = 1,

SAVED\_CHANGES = 2

class Employee:

def \_\_init\_\_(self, unitId, name, rank, experience):

self.mUnitId = unitId

self.mName = name

self.mRank = rank

self.mExperience = experience

class Unit:

def \_\_init\_\_(self, name):

self.mEmployees = set()

self.mName = name

class HumanResourcesManager:

def \_\_init\_\_(self):

self.mUnits = dict()

self.mEmployees = dict()

self.mState = State.EMPTY

self.mCurrentFile = ""

def createEmployee(self, employee):

if len(self.mEmployees) == 0:

key = 0;

else:

key = max(self.mEmployees.keys()) + 1

self.mEmployees[key] = employee

self.mUnits[employee.mUnitId].mEmployees.add(key)

return key

def createUnit(self, unit):

if len(self.mUnits) == 0:

key = 0;

else:

key = max(self.mUnits.keys()) + 1

self.mUnits[key] = unit

return key

def statU(self):

items = list(self.mUnits.items())

items.sort(key=lambda x: len(x[1].mEmployees), reverse=True)

for key, value in items:

print("Unit id: ", key, ", name: ", value.mName, ", number of employees: ", len(value.mEmployees))

def statR(self):

items = list(self.mEmployees.items())

items.sort(key=lambda x: x[1].mRank, reverse=True)

for key, value in items:

print("Employee id: ", key, ", name: ", value.mName, ", rank: ", value.mRank)

def statE(self):

items = list(self.mEmployees.items())

items.sort(key=lambda x: x[1].mExperience, reverse=True)

for key, value in items:

print("Employee id: ", key, ", name: ", value.mName, ", experience: ", value.mExperience)

def checkEmpty(self):

if self.mState == State.EMPTY:

ERROR("document is not loaded")

return False;

return True

def suggestSave(self):

if self.mState == State.UNSAVED\_CHANGES:

WARNING("You have unsaved changes, save it? [y/n]")

answer = yesNoDialog()

if answer:

self.save(self.mCurrentFile)

def clean(self):

self.mUnits = dict()

self.mEmployees = dict()

self.mState = State.EMPTY

self.mCurrentFile = ""

def save(self, file):

if not self.checkEmpty(): return

root = ET.Element('company')

tree = ET.ElementTree(root)

units = ET.SubElement(root, "units")

for key, value in self.mUnits.items():

unit = ET.SubElement(units, "unit")

unit.set("id", str(key))

name = ET.SubElement(unit, "unitName")

name.text = value.mName

unitEmployees = ET.SubElement(unit, "unitEmployees")

for employee in value.mEmployees:

unitEmployee = ET.SubElement(unitEmployees, "unitEmployee")

unitEmployee.set("id", str(employee))

employees = ET.SubElement(root, "employees")

for key, value in self.mEmployees.items():

employee = ET.SubElement(employees, "employee")

employee.set("id", str(key))

name = ET.SubElement(employee, "name")

name.text = value.mName

unitId = ET.SubElement(employee, "unitId")

unitId.text = str(value.mUnitId)

rank = ET.SubElement(employee, "rank")

rank.text = str(value.mRank)

experience = ET.SubElement(employee, "experience")

experience.text = str(value.mExperience)

with open(file, 'wb') as f:

tree.write(f, pretty\_print=True, encoding='utf-8')

if self.mCurrentFile == file:

self.mState = State.SAVED\_CHANGES

INFO(file + " saved")

def load(self, file):

self.suggestSave()

self.clean()

if not os.path.isfile(file):

ERROR("incorrect path")

return;

f = open(file, "r")

try:

xmlDoc = ET.fromstring(f.read())

dtd = ET.DTD(open(dtdPath))

if not dtd.validate(xmlDoc):

ERROR("Dtd validation not passed")

return

except:

ERROR("Error while parsing. Invalid document")

return

for xmlUnit in xmlDoc.findall("units/unit"):

id = int(xmlUnit.get('id'))

name = xmlUnit.find('unitName').text

unit = Unit(name)

self.mUnits[id] = unit

DEBUG("Parse unit. Id: " + str(id) + ", name: " + name)

for xmlUnitEmployee in xmlUnit.findall("unitEmployees/unitEmployee"):

employeeId = int(xmlUnitEmployee.get('id'))

DEBUG("Parse employee in unit. Id: " + str(employeeId))

self.mUnits[id].mEmployees.add(employeeId)

for xmlEmployee in xmlDoc.findall("employees/employee"):

id = int(xmlEmployee.get('id'))

unitId = int(xmlEmployee.find('unitId').text)

name = xmlEmployee.find('name').text

rank = int(xmlEmployee.find('rank').text)

experience = int(xmlEmployee.find('experience').text)

DEBUG("Parse employee. Id: " + str(id) + ", unitId: " + str(unitId) + ", name: " + name + ", rank: " + str(rank) + ", experience: " + str(experience))

employee = Employee(unitId, name, rank, experience)

self.mEmployees[id] = employee

loadedEmployees = [];

for key, value in self.mUnits.items():

for employee in value.mEmployees:

if not employee in self.mEmployees:

ERROR("Unexisting employee in unit " + value.mName)

self.clean()

continue

if self.mEmployees[employee].mUnitId != key:

ERROR("Unexpected unitId of employee " + employee)

self.clean()

continue

loadedEmployees.append(employee)

for key in self.mEmployees:

if not key in loadedEmployees:

ERROR("Unexpected employee " + key)

self.clean()

continue

INFO(file + " loaded. " + str(len(self.mUnits)) + " units, " + str(len(self.mEmployees)) + " employees.")

self.mState = State.SAVED\_CHANGES

self.mCurrentFile = file

def new(self, file):

if os.path.isfile(file):

ERROR("File already exists")

else:

root = ET.Element('company')

tree = ET.ElementTree(root)

units = ET.SubElement(root, "units")

employees = ET.SubElement(root, "employees")

with open(file, 'wb') as f:

tree.write(f, pretty\_print=True, encoding='utf-8')

self.load(file)

opts = parseOptions(sys.argv)

if "d" in opts:

isDebug = True;

else:

isDebug = False;

dtdPath = opts.get("s", "schema.dtd")

DEBUG("Dtd path: " + dtdPath)

manager = HumanResourcesManager()

while True:

if(manager.mCurrentFile != ""):

hint = "[" + manager.mCurrentFile + "]: "

else:

hint = ""

command = input(hint)

words = command.split()

if command == "q":

manager.suggestSave()

exit()

elif command == "help":

printHelpInfo()

elif words[0] == "load":

if len(words) != 2:

ERROR("Incorrect use of load. Help:")

printHelpCommand("load [FILE]")

else:

manager.load(words[1])

elif words[0] == "new":

words = command.split()

if len(words) != 2:

ERROR("Incorrect use of new. Help:")

printHelpCommand("new [FILE]")

else:

manager.new(words[1])

elif words[0] == "save":

words = command.split()

options = parseOptions(words)

file = options.get("f", manager.mCurrentFile)

DEBUG("save to file " + file)

manager.save(file)

elif len(words) >= 2 and words[0] == "add" and words[1] == "unit":

if not manager.checkEmpty(): continue

options = parseOptions(command.split())

try:

id = manager.createUnit(Unit(options["n"]))

INFO("Create unit. Id: " + str(id))

manager.mState = State.UNSAVED\_CHANGES

except:

ERROR("Failed to add unit. Help:")

printHelpCommand("add unit")

elif len(words) >= 2 and words[0] == "add" and words[1] == "employee":

if not manager.checkEmpty(): continue

options = parseOptions(command.split())

try:

unitId = int(options["u"])

if not unitId in manager.mUnits:

ERROR("Incorrect unit id")

continue

id = manager.createEmployee(Employee(unitId, options["n"], int(options["r"]), int(options["e"])))

INFO("Create employee. Id: " + str(id))

manager.mState = State.UNSAVED\_CHANGES

except:

ERROR("Failed to add employee. Help:")

printHelpCommand("add employee")

elif len(words) >= 2 and words[0] == "update" and words[1] == "unit":

if not manager.checkEmpty(): continue

words = command.split()

options = parseOptions(words)

if len(words) < 3:

ERROR("Incorrect use of update unit. Help:")

printHelpCommand("update unit [UNIT ID]")

continue;

try:

id = int(words[2])

if not id in manager.mUnits:

ERROR("Incorrect id")

continue;

if "n" in options:

manager.mUnits[id].mName = options["n"]

INFO("Update unit " + str(id))

manager.mState = State.UNSAVED\_CHANGES

except:

ERROR("Failed to update unit. Help:")

printHelpCommand("update unit [UNIT ID]")

elif len(words) >= 2 and words[0] == "update" and words[1] == "employee":

if not manager.checkEmpty(): continue

words = command.split()

options = parseOptions(words)

if len(words) < 3:

ERROR("Incorrect use of update employee. Help:")

printHelpCommand("update employee [EMPLOYEE ID]")

continue;

try:

id = int(words[2])

if not id in manager.mEmployees:

ERROR("Incorrect id")

continue;

if "n" in options:

manager.mEmployees[id].mName = options["n"]

if "e" in options:

manager.mEmployees[id].mExperience = int(options["e"])

if "r" in options:

manager.mEmployees[id].mRank = int(options["r"])

if "u" in options:

oldUnitId = manager.mEmployees[id].mUnitId

newUnitId = int(options["u"])

if not newUnitId in manager.mUnits:

ERROR("Incorrect unit id specified")

else:

manager.mUnits[oldUnitId].mEmployees.remove(employeeId)

manager.mUnits[newUnitId].mEmployees.add(employeeId)

manager.mEmployees[id].mUnitId = newUnitId

INFO("Update employee " + str(id))

manager.mState = State.UNSAVED\_CHANGES

except:

ERROR("Failed to update employee. Help:")

printHelpCommand("update employee [EMPLOYEE ID]")

elif len(words) >= 2 and words[0] == "delete" and words[1] == "unit":

if not manager.checkEmpty(): continue

if len(words) != 3:

ERROR("Incorrect use of delete unit. Help:")

printHelpCommand("delete unit [UNIT\_ID]")

try:

id = int(words[2])

if not id in manager.mUnits:

ERROR("Incorect id")

continue

for employeeId in manager.mUnits[id].mEmployees:

del manager.mEmployees[employeeId]

del manager.mUnits[id]

INFO("Delete unit " + str(id))

manager.mState = State.UNSAVED\_CHANGES

except:

ERROR("Failed to delete unit. Help:")

printHelpCommand("delete unit [UNIT\_ID]")

elif len(words) >= 2 and words[0] == "delete" and words[1] == "employee":

if not manager.checkEmpty(): continue

if len(words) != 3:

ERROR("Incorrect use of delete employee. Help:")

printHelpCommand("delete employee [EMPLOYEE\_ID]")

try:

id = int(words[2])

if not id in manager.mEmployees:

ERROR("Incorect id")

continue

manager.mUnits[manager.mEmployees[id].mUnitId].mEmployees.remove(id)

del manager.mEmployees[id]

INFO("Delete employee " + str(id))

manager.mState = State.UNSAVED\_CHANGES

except:

ERROR("Failed to delete employee. Help:")

printHelpCommand("delete employee [EMPLOYEE\_ID]")

elif len(words) >= 2 and words[0] == "info" and words[1] == "unit":

if not manager.checkEmpty(): continue

if len(words) != 3:

ERROR("Incorrect use of info unit. Help:")

printHelpCommand("info unit [UNIT\_ID]")

try:

id = int(words[2])

if not id in manager.mUnits:

ERROR("Incorect id")

continue

unit = manager.mUnits[id]

print("Name: ", unit.mName)

if len(unit.mEmployees) == 0:

employeesStr = "No employees"

else:

employeesStr = "Employees:"

for employeeId in unit.mEmployees:

employeesStr = employeesStr + " " + str(employeeId)

print(employeesStr)

except:

ERROR("Failed to print info about unit. Help:")

printHelpCommand("info unit [UNIT\_ID]")

elif len(words) >= 2 and words[0] == "info" and words[1] == "employee":

if not manager.checkEmpty(): continue

if len(words) != 3:

ERROR("Incorrect use of info employee. Help:")

printHelpCommand("info employee [EMPLOYEE\_ID]")

try:

id = int(words[2])

if not id in manager.mEmployees:

ERROR("Incorect id")

continue

employee = manager.mEmployees[id]

print("Name: ", employee.mName)

print("Unit: ", employee.mUnitId)

print("Rank: ", employee.mRank)

print("Experience: ", employee.mExperience)

except:

ERROR("Failed to print info about employee. Help:")

printHelpCommand("info employee [EMPLOYEE\_ID]")

elif words[0] == "stat":

try:

if not manager.checkEmpty(): continue

if "-u" in words:

manager.statU()

if "-r" in words:

manager.statR()

if "-e" in words:

manager.statE()

except:

ERROR("Failed to print statistics. Help:")

printHelpCommand("stat")

elif len(words) == 2 and words[0] == "all" and words[1] == "units":

if not manager.checkEmpty(): continue

try:

output = ""

for key in manager.mUnits.keys():

output = output + str(key) + " "

print(output)

except:

ERROR("Failed to print all units. Help:")

printHelpCommand("all units")

elif len(words) == 2 and words[0] == "all" and words[1] == "employees":

if not manager.checkEmpty(): continue

try:

output = ""

for key in manager.mEmployees.keys():

output = output + str(key) + " "

print(output)

except:

ERROR("Failed to print all employees. Help:")

printHelpCommand("all employees")

else:

ERROR("Unknown command")

**опис програми (опис класів, методів, полів)**

Enum State представляє три стани програми: відсутність завантаженого документа (EMPTY), наявність незбережених змін (UNSAVED\_CHANGES) і збережені зміни (SAVED\_CHANGES).

Клас Employee – представленя робітника (ім’я, id підрозділу, до якого він відноситься, ранг в компанії і досвід)

Клас Unit – представлення підрозділу (ім’я і множина id робітників)

Клас HumanResourcesManager відповідає за основну логіку програми. Його полями є mState – стан програми, mCurrentFile – шлях до завантаженого файлу, mEmployees – словник з ключем id працівника, значенням об’єктом Employee і mUnits – словник з ключем id підрозділу, значенням об’єктом Unit.

Методи createEmployee і createUnit створюють нового співробітника чи підрозділ і повертають його унікальний id.

Методи stat\* виводять статистичну інформацію: statU – підрозділи у порядку спадання кількості співробітників, statR – робітників за рангом, state – робітників за досвідом.

Метод save зберігає нинішній документ, load завантажує інший файл, а new створює новий.

printHelpCommand, printHelpInfo, INFO, ERROR, WARNING, DEBUG, yesNoDialog, parseOptions – допоміжні функції

Логіка редагування, видалення, виводу інформації реалізована в основному скрипті.

**Приклад xml-документа, що містить інформацію по заданій предметній області**

<company>

<units>

<unit id="0">

<unitName>Unit1</unitName>

<unitEmployees>

<unitEmployee id="0"/>

<unitEmployee id="1"/>

</unitEmployees>

</unit>

<unit id="1">

<unitName>KKK</unitName>

<unitEmployees>

<unitEmployee id="2"/>

<unitEmployee id="3"/>

<unitEmployee id="4"/>

</unitEmployees>

</unit>

<unit id="2">

<unitName>UNIT500</unitName>

<unitEmployees>

<unitEmployee id="5"/>

</unitEmployees>

</unit>

</units>

<employees>

<employee id="0">

<name>Petro\_Petrovych</name>

<unitId>0</unitId>

<rank>1</rank>

<experience>2</experience>

</employee>

<employee id="1">

<name>AAA</name>

<unitId>0</unitId>

<rank>5</rank>

<experience>10</experience>

</employee>

<employee id="2">

<name>BBB</name>

<unitId>1</unitId>

<rank>6</rank>

<experience>4</experience>

</employee>

<employee id="3">

<name>CCC</name>

<unitId>1</unitId>

<rank>100</rank>

<experience>0</experience>

</employee>

<employee id="4">

<name>DDD</name>

<unitId>1</unitId>

<rank>3</rank>

<experience>3</experience>

</employee>

<employee id="5">

<name>FFF</name>

<unitId>2</unitId>

<rank>2</rank>

<experience>1000</experience>

</employee>

</employees>

</company>

**Опис структури xml-документа в форматі DTD**

<!ELEMENT company (units, employees)>

<!ELEMENT units (unit\*)>

<!ELEMENT employees (employee\*)>

<!ELEMENT unit (unitName, unitEmployees)>

<!ELEMENT employee (name, unitId, rank, experience)>

<!ELEMENT unitName (#PCDATA)>

<!ELEMENT unitEmployees (unitEmployee\*)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT rank (#PCDATA)>

<!ELEMENT experience (#PCDATA)>

<!ELEMENT unitId (#PCDATA)>

<!ELEMENT unitEmployee (#PCDATA)>

<!ATTLIST unit id CDATA #REQUIRED>

<!ATTLIST unitEmployee id CDATA #REQUIRED>

<!ATTLIST employee id CDATA #REQUIRED>

**Опис структури xml-документа в форматі XMLSchema**

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="company">

<xs:complexType>

<xs:sequence>

<xs:element name="units">

<xs:complexType>

<xs:sequence>

<xs:element name="unit" maxOccurs="unbounded" minOccurs="0">

<xs:complexType>

<xs:sequence>

<xs:element type="xs:string" name="unitName"/>

<xs:element name="unitEmployees">

<xs:complexType>

<xs:sequence>

<xs:element name="unitEmployee" maxOccurs="unbounded" minOccurs="0">

<xs:complexType>

<xs:simpleContent>

<xs:extension base="xs:string">

<xs:attribute type="xs:integer" name="id" use="required"/>

</xs:extension>

</xs:simpleContent>

</xs:complexType>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:sequence>

<xs:attribute type="xs:integer" name="id" use="optional"/>

</xs:complexType>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name="employees">

<xs:complexType>

<xs:sequence>

<xs:element name="employee" maxOccurs="unbounded" minOccurs="0">

<xs:complexType>

<xs:sequence>

<xs:element type="xs:string" name="name"/>

<xs:element type="xs:integer" name="unitId"/>

<xs:element type="xs:integer" name="rank"/>

<xs:element type="xs:integer" name="experience"/>

</xs:sequence>

<xs:attribute type="xs:integer" name="id" use="required"/>

</xs:complexType>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>