Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение
высшего образования «Национальный исследовательский университет
«Высшая школа экономики»

Кафедра «Компьютерная безопасность»

ОТЧЕТ
К ЛАБОРАТОРНОЙ РАБОТЕ №7-8

по дисциплине

«Языки программирования»

Работу выполнила
студентка группы
СКБ-232 _____ Д.В. Иванова
подпись, дата

Работу проверил _____ С.А. Булгаков
подпись, дата

Москва 2023

Содержание

Постановка задачи

Нужно было разработать набор классов позволяющий выполнять манипуляции над файлами формата XML. Основной сущностью является должен был являться класс ‘Element‘ описывающий тег, содержащий атрибуты, значение и/или другие дочерние теги. Класс должен был соответствовать принципу инкапсуляции. В классе надо было реализовать конструкторы умолчания и копирования, деструктор, оператор присваивания для объекта класса и ‘const char*‘. Прочий функционал вводить при необходимости. Класс мог быть полиморфным.

Функционал для Записи XML-файлов надо было реализовать в классе ‘XMLWriter‘ методами:

- void DocumentBegin(const char *version, const char *encoding)
- void DocumentEnd()
- void ElementBegin() - группа перегруженных методов для записи тега, в том числе с атрибутами
- void ElementEnd()
- void WriteAttribute() и void WriteAttributes() - группа перегруженных методов для записи аттрибутов
- void WriteElement(const Element)
- void WriteElements(const Element *)

Функционал для чтения XML-файлов реализовать в классе ‘XMLReader‘ на основе подхода SAX.

Предусмотреть возможность поступления некорректных данных.

Для обработки нештатных ситуаций задействовать механизм исключений.

1 Описания классов

Описание класса Element

Класс содержит поля std::string name -имя элемента, std::string val - значение элемента и AttributeList attributes - список аттрибутов. Имеет методы:

- Element(const std::string name) - конструктор
- const std::string getName() const - геттер имени элемента
- Element(const std::string name, const std::string val) - конструктор со значением
- const AttributeList getAttributes() const - геттер атрибутов
- const std::string getVal() const - геттер значение
- bool addAttribute(const Attribute attr) - добавляет атрибуты
- Element() - деструктор

Описание класса Attribute

Класс содержит поля std::string name - имя аттрибута, std::string value - его значение. Имеет методы:

- Attribute(); //конструктор умолчания
- Attribute(const Attribute other) - конструктор копирования
- Attribute(const std::string name) - конструктор с именем
- const std::string getName() const - геттер имени
- const std::string getValue() const - геттер значения
- void setValue(const std::string value) - сеттер значения
- Attribute() - деструктор

Описание класса AttributeList

Класс содержит поля Attribute *list - список атрибутов, size_t count - количество атрибутов, size_t maxsize - максимальное число атрибутов

- AttributeList(size_t size) - конструктор с размером
- const Attribute getAttribute(size_t i) const - геттер атрибута
- bool addAttribute(const Attribute attr) - добавляет атрибуты
- size_t Size() const - размер листа
- void printList(const AttributeList attrlst) - печатает список атрибутов

Описание класса XMLWriter

Класс содержит поля std::ostream out - поток вывода, std::string *elements -элементы, size_t numelements - кол-во элементов, bool inelem - индекс, показывающий, находимся ли мы в элементе

- XMLWriter(std::ostream out);
- void DocumentBegin(const char *version, const char *encoding) - печатает начало документа
- void DocumentEnd() - печатает конец документа
- void ElementBegin(const char *name) - печатает начало элемента
- void ElementBegin(const Element elem) - печатает начало элемента
- void ElementEnd(const Element elem) - печатает конец элемента
- void WriteAttribute(const char *name, const char *value) - печатает атрибуты
- void WriteAttribute(const Attribute attr) - печатает атрибуты
- void WriteAttributes(const AttributeList attrlst)- печатает атрибуты
- void WriteAttributes(const Attribute a1, const Attribute a2) - печатает атрибуты
- void WriteElement(const Element element) - печатает элемент
- void WriteElements(const Element *es, int count) - печатает элементы
- void WriteVal(const char *value) -печатает значение
- XMLWriter() -деструктор

Описание класса SAXHandler

У класса нет полей. Есть чисто виртуальные методы:

- virtual void OnElementBegin(const char *name)
- virtual void OnElementEnd (const char *name)
- virtual void OnAttrBegin (const char *name, const char *val)
- virtual void OnAttrEnd (const char *name, const char *val)
- virtual void justVal (const char *name)

Описание класса XMLReader

Класс содержит поля std::istream in - поток ввода и SAXHandler handler. Имеет методы:

- void OnElementBegin(const char *name) - делает необходимое на начале эл-та
- void OnElementEnd (const char *name) - делает необходимое на конце эл-та
- XMLReader(std::istream in, SAXHandler handler) - конструктор
- void ParseXML() - парсер

2 Тестирование

1 Тестирование записи в файл

```
std::ofstream out("fileout.xml");
        Element name("name", "Serj"), student("student"), group("group", "CoSec-2023");
        student.addAttribute(Attribute("class", "mage"));
        student.addAttribute(Attribute("element","Hydro"));
        student.addAttribute(Attribute());
        if( out.is_open() ){
        XMLWriter xml(out);
        xml.DocumentBegin("1.1", "UTF-8");
        xml.ElementBegin(student);
        xml.WriteElement(name);
        xml.WriteElement(group);
        xml.ElementEnd(student);
        xml.DocumentEnd();
        out.close();
    }
```

```
fileout.xml:
<?xml version="1.1" encoding="UTF-8" ?>
    <student>
        <name> Serj </name>
        <group> CoSec-2023 </group>
    </student>
```

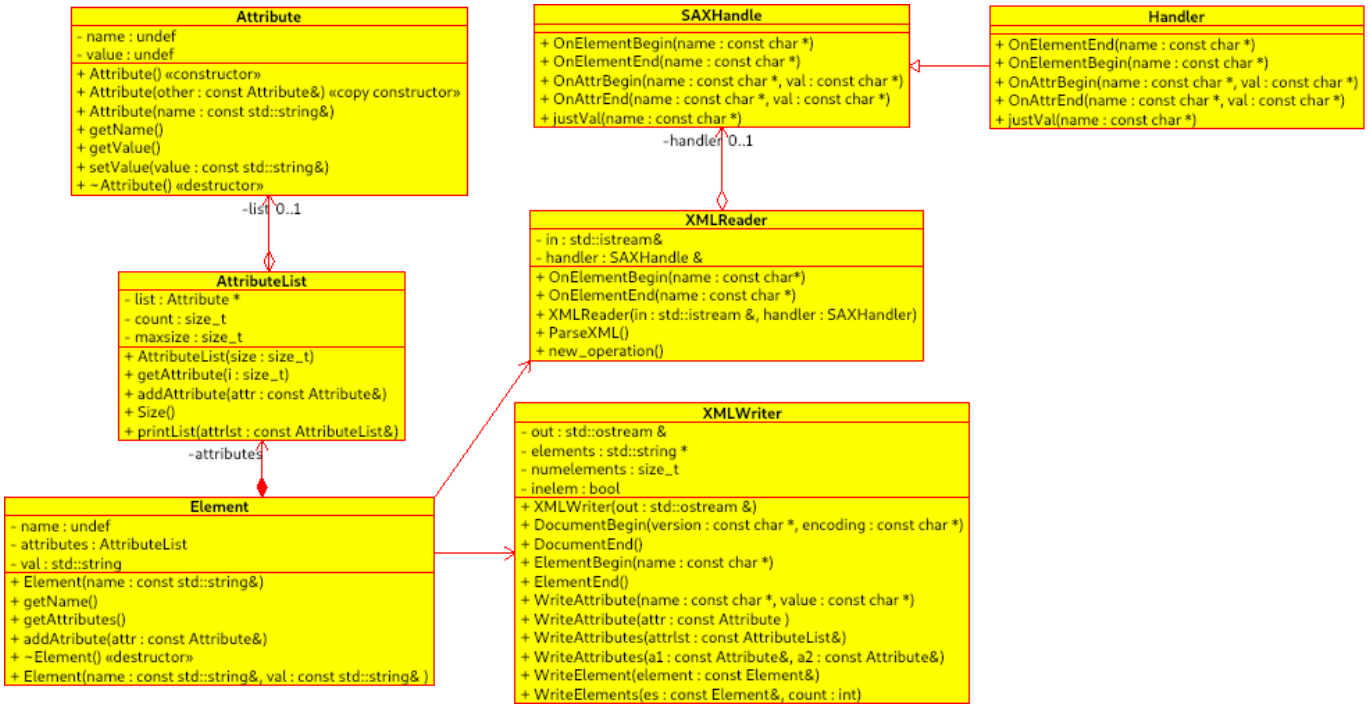
2 Тестирование чтения из файла

```
std::ifstream in("file.xml");
    if( in.is_open() ){
        Handler   h;
        XMLReader xml(in,h);
        xml.ParseXML();
    }
```

```
output:
Found element: student
Attribute of element: class="mage"
Attribute of element: level="cool"
Attribute of element: element="Hydro"
Found element: name
Value of element:Serj
Found element: group
```

Приложение А

UML 2.0-схема классов



Приложение Б

Код Element.h

```
#ifndef  Element_h
#define  Element_h

#include <string>
#include "AttributeList.h"

class Element{
    std::string   name;
    std::string   val;
    AttributeList attributes; //список аттрибутов

  public:
    Element(const std::string& name); //конструктор
    Element(const std::string& name, const std::string& val); //конструктор
    const std::string& getName() const; //геттер имени элемента, const потому что геттер
    const std::string& getVal() const; //геттер имени элемента, const потому что геттер
    const AttributeList& getAttributes() const; //геттер атрибутов, const потому что геттер
    bool  addAttribute(const Attribute& attr);
    ~Element(); //деструктор
};

#endif //Element_h
```

Код Element.cpp

```
#include "Element.h"

Element::Element(const std::string& name) //конструктор
    : name(name), val(""), attributes(0) {}

Element::Element(const std::string& name, const std::string& val) //конструктор
    : name(name), val(val), attributes(0) {}

const std::string& Element::getName() const{ //геттер имени элемента, const потому что геттер
    return name;
}

const AttributeList& Element::getAttributes() const{ //геттер атрибутов, const потому что геттер
    return attributes;
}

bool  Element::addAttribute(const Attribute& attr){
    return attributes.addAttribute(attr);
}

const std::string& Element::getVal() const{
    return val;
}

Element::~Element(){}
```

Код Attribute.h

```
#ifndef  Attribute_h
#define  Attribute_h

#include <string>

class Attribute {
    std::string name; //имя аттрибута
    std::string value; //его значение

  public:
    Attribute(); //конструктор умолчания
    Attribute(const Attribute& other); //конструктор копирования, конст потому что не хотим менять ничё

    Attribute(const std::string& name); // конструктор с именем
    Attribute(const std::string& name, const std::string& value); // конструктор с именем

    const std::string& getName()  const; //геттер имени
    const std::string& getValue() const; //геттер значения
    void  setValue(const std::string& value); //сеттер значения
```

```
    ~Attribute(); //деструктор
};

#endif //Attribute_h
```

Код Attribute.cpp

```
#include <iostream>
#include "Attribute.h"


Attribute::Attribute() //конструктор умолчания
    :name(""), value("") {}

Attribute::Attribute(const Attribute& other) //конструктор копирования, конст потому что не хотим менять ничё
    :name(other.name), value(other.value) {}

Attribute::Attribute(const std::string& name) // конструктор с именем
    :name(name) {}

Attribute::Attribute(const std::string& name, const std::string& value)
    :name(name), value(value) {}

const std::string& Attribute::getName()  const{ //геттер имени
    return name;
}

const std::string& Attribute::getValue() const{ //геттер значения
    return value;
}
void  Attribute::setValue(const std::string& value) {//сеттер значения
    this->value = value;
}

Attribute::~Attribute(){}
```

Код AttributeList.h

```
#ifndef  AttributeList_h
#define  AttributeList_h

#include "Attribute.h"

class AttributeList
{
    Attribute  *list ; // список атрибутов
    size_t      count; // количество атрибутов
    size_t      maxsize;  // максимальное число атрибутов

  public:
    AttributeList(size_t size); // конструктор с размером

    const Attribute& getAttribute(size_t i) const; //геттер(?) атрибута

    bool  addAttribute(const Attribute& attr);

    size_t Size() const; // размер листа

    void printList(const AttributeList& attrlst);
};

#endif //AttributeList_h
```

Код AttributeList.cpp

```
#include "Attribute.h"
#include "AttributeList.h"

AttributeList::AttributeList(size_t size) // конструктор с размером
    : list(NULL), count(0), maxsize(size)
{
    list = new Attribute[maxsize]; //!new?
}

const Attribute& AttributeList::getAttribute(size_t i) const{ //геттер(?) атрибута
    return list[i];
```

```
}

bool  AttributeList::addAttribute(const Attribute& attr) {
  if (count < maxsize){
    list[count++] = attr;
    return true;
  }
  //?тут можно вставить вывод ошибки
  return false;
}

size_t AttributeList::Size() const{ // размер листа
    return count;
}
```

Код SAXHandler.h

```
#ifndef  SAXHandler_h
#define  SAXHandler_h

class SAXHandler
{
  public:
    virtual void OnElementBegin(const char *name) = 0;
    virtual void OnElementEnd  (const char *name) = 0;
    virtual void OnAttrBegin   (const char *name, const char *val) = 0;
    virtual void OnAttrEnd     (const char *name, const char *val) = 0;
    virtual void justVal       (const char *name) = 0;

};

#endif //SAXHandler_h
```

Код XMLReader.h

```
#ifndef  XMLReader_h
#define  XMLReader_h

#include <istream>
#include "SAXHandler.h"
#include "Element.h"

class XMLReader
{
    std::istream & in;
    SAXHandler &handler;

  public:
    void OnElementBegin(const char *name);
    void OnElementEnd  (const char *name);
    XMLReader(std::istream &in, SAXHandler &handler); //! тут наследуется!!
    void ParseXML();
};



#endif //XMLReader_h
```

Код XMLReader.cpp

```
#include "XMLReader.h"

#include <string>
#include <iostream>

bool isVer(const std::string& str) {
    bool isVersion = true;

    for (size_t i =0; i<str.size(); i++){
        if ((str[i]<'0' || str[i]>'9') && (str[i]!='.' && str[i]!='"')){
            isVersion = false;
            std::cout << str[i] << "sec:" << (str[i]!='.') << " " << (str[i]!='"') <<  "\n";
        }
    }

    return isVersion;
```

```
}

XMLReader::XMLReader(std::istream &in, SAXHandler &handler)
  : in(in), handler(handler) {}
// g++ .\main.cpp .\attributelist.cpp .\attribute.cpp .\element.cpp .\XMLWriter.cpp .\XMLReader.cpp   -o prog.exe
void XMLReader::ParseXML() {
    if( in.get() == '<' ){

        if( in.get() == '?' ){
            std::string str;
            in >> str;
            if( str == "xml" ){
                in >> str;

                if( str.substr(0, 9) == "version=\"" && (isVer(str.substr(8, 11)) && str[12]=='"') ){
                    in >> str;


                    if( str == "encoding=\"UTF-8\"" ){
                        in >> str;

                        if( str == "?>" ){

                          if( in.peek() == '\n' ) in.get();
                          while( in >> str ) {
                              if(str[0] == '<'){
                                  size_t end = str.find(' ');
                                if( end == std::string::npos )
                                        end =  str.find('>');

                                    //20231203L
                                if( str[1] == '/' )
                                    handler.OnElementEnd(str.substr(2,end-1).c_str());
                                else{
                                    handler.OnElementBegin(str.substr(1,end-1).c_str());

                                    }
                                }
                                  else {
                                      if (str.find('=')!=std::string::npos){
                                          size_t beg_attr_nm = 0;
                                          // <student class="mage"  level="cool" element="Hydro">
                                          size_t end_attr_nm = str.find('=', beg_attr_nm);
                                          size_t beg_attr_val = end_attr_nm+1;
                                          size_t end_attr_val = str.size()-1;
                                          if (str[end_attr_val] == '>')
                                              end_attr_val -=1;
                                          handler.OnAttrBegin(str.substr(beg_attr_nm, end_attr_nm).c_str(), str.subst
                                      }
                                      else
                                          handler.justVal(str.c_str());
                                  }
                              }
                              /* ---- */
                            }
                              else
                                  throw std::runtime_error("Close tag error\n");
                        }
                        else
                            throw std::runtime_error("Encoding error\n");
                    }
                    else
                        throw std::runtime_error("Version error\n");
                }
            }
        }
}
```

Код XMLWriter.h

```
#ifndef  XMLWriter_h
#define  XMLWriter_h

#include "Element.h"
#include <ostream>
#include <string>

class XMLWriter {
    std::ostream & out;
```

```
    std::string   *elements;
    size_t      numelements;
    bool             inelem;

    public:
    XMLWriter(std::ostream & out);
    void DocumentBegin(const char *version, const char *encoding); //*
    void DocumentEnd(); //*
    void ElementBegin(const char *name); //группа перегруженных методов для записи тега, в том числе с аттрибутами
    void ElementBegin(const Element& elem); //группа перегруженных методов для записи тега, в том числе с аттрибута
    void ElementEnd(const Element& elem); //*
    void WriteAttribute(const char *name, const char *value);//*
    void WriteAttribute(const Attribute attr); //*
    void WriteAttributes(const AttributeList& attrlst); //группа перегруженных методов для записи аттрибутов
    void WriteAttributes(const Attribute &a1, const Attribute &a2);
    void WriteElement(const Element& element); //*
    void WriteElements(const Element *es, int count);
    void WriteVal(const char *value);
    ~XMLWriter();
};
#endif // XMLWriter_h
```

Код XMLWriter.cpp

```
#include "XMLWriter.h"


XMLWriter::XMLWriter(std::ostream & out)
    : out(out), elements(NULL), numelements(0), inelem(true) {}

void XMLWriter::DocumentBegin(const char *version, const char *encoding){
    out << "<?xml version=\"" << version << "\" encoding=\"" << encoding << "\" ?>" << std::endl;
}

void XMLWriter::DocumentEnd() {}

void XMLWriter::ElementBegin(const char *name)
{
    //std::string *tmp = new std::string[numelements+1];
    //for(size_t i =0; i < numelements; ++i)
    //    tmp[i] = elements[i];
    //tmp[numelements] = name;
    numelements++;
    //delete [] elements;
    //elements = tmp;

    for(size_t i = 0; i < numelements; ++i)
        out << "    ";
    out << '<' << elements[numelements-1] ;

    //inelem = false;
    //if( inelem == false )
    out << '>' << std::endl;

}

void XMLWriter::ElementBegin(const Element& element){
    for(size_t i = 0; i <= numelements; ++i)
        out << "    ";
    numelements++;
    out << '<'  << element.getName();
    if (element.getAttributes().Size()>0)
        out << " ";
    for (size_t i =0; i< element.getAttributes().Size(); i++){
        Attribute attr = element.getAttributes().getAttribute(i);
        out << attr.getName() << "= \"" << attr.getValue() << "\""; //!something may be wrong
    }
    out  << '>' << std::endl;
    if (element.getVal()!="")
        out << " " << element.getVal() << " ";
}

void XMLWriter::ElementEnd(const Element& elem){
    numelements--;
    for(size_t i = 0; i <= numelements; ++i)
        out << "    ";
    out << "</" << elem.getName() << '>' << std::endl;

}
```

```
void XMLWriter::WriteElement(const Element& element) {
    for(size_t i = 0; i <= numelements; ++i)
        out << "    ";
    out << '<'  << element.getName();
    if (element.getAttributes().Size()>0)
        out << " ";
    for (size_t i =0; i< element.getAttributes().Size(); i++){
        Attribute attr = element.getAttributes().getAttribute(i);
        out << attr.getName() << "= \"" << attr.getValue() << "\""; //!something may be wrong
    }
    out  << '>';
    out << " " << element.getVal() << " ";
    out << "</" << element.getName() << ">" << std::endl;
}

void XMLWriter::WriteElements(const Element *es, int count){
    for (size_t i=0; i<count; i++)
        WriteElement(es[i]);
}

void XMLWriter::WriteAttribute(const char *name, const char *value) {
  if( inelem == true ) // was false
    out << ' '  << name << "=\"" << value << '"' ; //!!!!
}

void XMLWriter::WriteAttribute(const Attribute attr) {
  if( inelem == true ) // was false
    out << ' '  << attr.getName() << "=\"" << attr.getValue() << '"' ; //!!!!
}

void XMLWriter::WriteAttributes(const AttributeList& attrlst) {
    if (inelem == true){
        for (size_t i = 0; i< attrlst.Size(); i++)
            out << ' '  << attrlst.getAttribute(i).getName() << "=\"" << attrlst.getAttribute(i).getValue() << '"'
    }
}

void XMLWriter::WriteAttributes(const Attribute& a1, const Attribute& a2){
    if (inelem == true){
        out << ' '  << a1.getName() << "=\"" << a1.getValue() << '"' ;
        out << ' '  << a2.getName() << "=\"" << a2.getValue() << '"' ;
    }
}

void XMLWriter::WriteVal(const char *value) {
    out << ' ' << value << ' ' ; //!!!!
}


XMLWriter::~XMLWriter(){}
```

Код main.cpp

```
#include "XMLWriter.h"
#include "XMLReader.h"

#include <fstream>
#include <iostream>

class Handler: public SAXHandler
{
  public:
    void OnElementBegin (const char *name) {
      std::cout << "Found element: " << name << std::endl;
    }
    void OnElementEnd   (const char *name) {}
    void OnAttrBegin    (const char *name, const char *val) {
        std::cout << "Attribute of element: " << name << "=" << val << std::endl;
    }
    void OnAttrEnd      (const char *name, const char *val){}
    void justVal(const char *name) {
        std::cout << "Value of element:" << name << std::endl;
    }
};

int main() {

    try {
    std::ifstream in("file.xml");
```

```
    if( in.is_open() ){
        Handler   h;
        XMLReader xml(in,h);
        xml.ParseXML();
    }
    std::ofstream out("fileout.xml");
        Element name("name", "Serj"), student("student"), group("group", "CoSec-2023");
        student.addAttribute(Attribute("class", "mage"));
        student.addAttribute(Attribute("element","Hydro"));
        student.addAttribute(Attribute());
        if( out.is_open() ){
        XMLWriter xml(out);
        xml.DocumentBegin("1.1", "UTF-8");
        xml.ElementBegin(student);
        xml.WriteElement(name);
        xml.WriteElement(group);
        xml.ElementEnd(student);
        xml.DocumentEnd();
        out.close();
    }
    }
    catch (std::exception &e) {
        std::cerr << "Cought exception: " << e.what() << std::endl;
    }
    catch (...) {
        std::cerr << "Cought something..." << std::endl;
    }
}
```