Министерство образования Республики Беларусь

Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЕНИЯ

Кафедра интеллектуальных информационных технологий

Отчет

По дисциплине: Проектирование программного обеспечения в интеллектуальных системах
Лабораторная работа №2

Выполнил: Липский Р. В.,
гр. 121701

Проверил: Никифоров С. А.

Минск 2022

Цель: получить навыки проведения объектно-ориентированного анализа предметной области.

Задание: провести объектно-ориентированный анализ предметной области, выделить классы и связи между ними. На основании объектной модели реализовать программу на C++.

Индивидуальное задание: необходимо разработать объектную модель для представления XML-документа. Разработанная объектная модель должна обеспечивать представление XML-документа в виде дерева и поддерживать следующие понятия XML-разметки:
· документ
· инструкция обработки
· тэг
· комментарий
· CDATA
· атрибут
Разработанная объектная модель должна соответствовать следующим требованиям:
· включать классы представления XML-документа;
· возможность обрабатывать поисковые запросы. В качестве языка поисковых запросов будет использоваться подмножество языка XPath

Ход выполнения:

1. Реализация XmlLexer

Для упрощения обработки XML файлы решено разбить этот процесс на несколько этапов:

- Лексинг — разбиение простой строки на промежуточное представление, состоящее из «токенов» (открывающая скобка, закрывающая скобка, слово, строковый литерал и т.д.)
- Парсинг — генерация объектов из модели на основе промежуточного представления.

1.1. *Реализация XmlLexer::Token*

```
class Token {
private:
    Type type;
    std::string text;
    size_t row;
    size_t col;
```

XmlLexer::Token представляет из себя класс, содержащий информацию о типе токена (XmlLexer::Token::Type), его содержимом (для Type::STRING и Type::WORD) и местоположении в исходной строке.

1.2. *Реализация XmlLexer::OpLogEntry*

```
struct OpLogEntry {
    int row;
    int col;
    int ind;

    OpLogEntry(int row, int col, int ind) {
        this->row = row;
        this->col = col;
        this->ind = ind;
    }
};
```

Для возможности возврата на один (или несколько) токенов назад при лексинге, лексер хранит информацию о перемещениях при создании токенов.

1.3. Реализация XmlLexer::Lexer

Сам лексер включает в себя методы для обработки разных видов токенов:

- XmlLexer::Lexer::parseToken() - для токенов, состоящих из нескольких символов.

```
XmlLexer::Token XmlLexer::Lexer::parseToken() {
    switch (currentChar()) {
```

```
case XmlLexer::Symbol::OP_SHARD: {
    // Open comment
    if (relativeChar(1) == Symbol::BANG &&
        relativeChar(2) == Symbol::DASH &&
        relativeChar(3) == Symbol::DASH) {
        move(4); return MAKE_TOKEN(Token::Type::OP_COMMENT);
    }

    // Open close tag
    if (relativeChar(1) == Symbol::SLASH) {
        move(2); return MAKE_TOKEN(Token::Type::OP_CLOSE_TAG);
    }

    // Processing instruction
    if (relativeChar(1) == Symbol::QUESTION) {
        move(2); return MAKE_TOKEN(Token::Type::OP_INSTRUCTION);
    }
    break;
}

case Symbol::DASH: {
    // Close comment
    if (relativeChar(1) == Symbol::DASH &&
        relativeChar(2) == Symbol::CL_SHARD) {
        move(3); return MAKE_TOKEN(Token::Type::CL_COMMENT);
    }
    break;
}

case Symbol::QUESTION: {
    // Close instruction
    if (relativeChar(1) == Symbol::CL_SHARD) {
        move(2); return MAKE_TOKEN(Token::Type::CL_INSTRUCTION);
    }
}

case Symbol::SLASH: {
    // Close empty tag
    if (relativeChar(1) == Symbol::CL_SHARD) {
        move(2); return MAKE_TOKEN(Token::Type::CL_EMPTY_TAG);
    }
}

case Symbol::BAR: {
    // OR
    if (relativeChar(1) == Symbol::BAR) {
        move(2); return MAKE_TOKEN(Token::Type::DBL_BAR);
    }
}

case Symbol::AMPERSAND: {
    if (relativeChar(1) == Symbol::AMPERSAND) {
        move(2); return MAKE_TOKEN(Token::Type::DBL_AMPERSAND);
    }
}
```

```
    }
    return parseUnaryToken();
}
```

- parseUnaryToken() - для обработки токенов, состоящих из одного символа.

```
XmlLexer::Token XmlLexer::Lexer::parseUnaryToken() {
    move(1);
    switch (relativeChar(-1)) {
        case XmlLexer::Symbol::OP_SHARD:
            return MAKE_TOKEN(Token::Type::OP_SHARD_BRACKET);
        case XmlLexer::Symbol::CL_PAREN:
            return MAKE_TOKEN(Token::Type::CL_PAREN);
        case XmlLexer::Symbol::OP_PAREN:
            return MAKE_TOKEN(Token::Type::OP_PAREN);
        case XmlLexer::Symbol::CL_SHARD:
            return MAKE_TOKEN(Token::Type::CL_SHARD);
        case XmlLexer::Symbol::BANG:
            return MAKE_TOKEN(Token::Type::BANG);
        case XmlLexer::Symbol::EQUALS:
            return MAKE_TOKEN(Token::Type::EQUALS);
        case XmlLexer::Symbol::DOT:
            return MAKE_TOKEN(Token::Type::DOT);
        case XmlLexer::Symbol::SLASH:
            return MAKE_TOKEN(Token::Type::SLASH);
        case XmlLexer::Symbol::OP_SQUARE:
            return MAKE_TOKEN(Token::Type::OP_SQUARE);
        case XmlLexer::Symbol::CL_SQUARE:
            return MAKE_TOKEN(Token::Type::CL_SQUARE);
        case XmlLexer::Symbol::COMMA:
            return MAKE_TOKEN(Token::Type::COMMA);
    }
    throw UnreachableError();
}
```

- parseStringLiteral() - для обработки строковых литералов.

```
XmlLexer::Token XmlLexer::Lexer::parseStringLiteral() {
    std::string str;
    move(1);
    while (currentChar() != Symbol::DBL_QUOTE) {
        str += currentChar();
        move(1);
    }
    move(1);
    return MAKE_TEXT_TOKEN(Token::Type::STRING, str);
}
```

- parseWord() - для обработки ключевых слов и содержимого тэгов.

```
XmlLexer::Token XmlLexer::Lexer::parseWord() {
    std::string word;
    while (isNumberOrDigit(currentChar())) {
        word += currentChar();
        move(1);
    }
```

```
    return MAKE_TEXT_TOKEN(Token::Type::WORD, word);
}
```

2. *Реализация модели Xml*

2.1. *Реализация Xml::Document*

Xml::Document представляет из себя корневой узел документа, содержащий информацию об специальных инструкциях обработки и дочерних тэгах.

```
class Document {

protected:
    std::map<std::string, std::string> instructions;
    std::vector<std::shared_ptr<Tag>> children;
    std::string content;

public:
    std::string getContent();

    std::vector<std::shared_ptr<Tag>> getChildren();

    void addChild(const std::shared_ptr<Tag>& child);

    std::string getInstruction(std::string name);

    void setInstruction(std::map<std::string, std::string> &instructions);
};
```

Для хранения ссылок на дочерние тэги, как в Xml::Document, так и в Xml::Tag используется стандартный класс std::shared_ptr. Преимущество его использования заключается в отсутствии необходимости вручную управлять памятью — встроенный счётчик ссылок сам удалит объект из кучи, когда все ссылки на него выйдут из области видимости.

2.2. Реализация Xml::Tag

Xml::Tag — рядовой узел xml-документа, содержащий в себе информацию о своих аттрибутах, родительском узле, корневом узле документа, ссылки на дочерние узлы, своё содержимое, имя.

```
class Tag {

    std::shared_ptr<Document> root;

    std::shared_ptr<Tag> parent;

    std::vector<std::shared_ptr<Tag>> children;

    std::string content;

    std::string name;
```

```
    std::map<std::string, std::string> attributes;
```

3. Реализация XmlParser

Парсер берёт за основу промежуточное представление полученное в лексере и формирует объекты для представления заданного xml документа.

```
Xml::Document XmlParser::Parser::parse() {
    root = std::make_shared<Xml::Document>();
    tag = nullptr;

    while(lexer.hasNext()) {
        auto token = lexer.next();
        switch (token.getType()) {
            case XmlLexer::Token::Type::OP_INSTRUCTION: {
                parseInstruction();
            } break;
            case XmlLexer::Token::Type::OP_COMMENT: {
                skipComments();
            } break;
            case XmlLexer::Token::Type::OP_SHARD_BRACKET: {
                openTag();
            } break;
            case XmlLexer::Token::Type::OP_CLOSE_TAG: {
                closeTag();
            } break;
            case XmlLexer::Token::Type::END: break;
            default:
                parseContent();
        }
    }

    return *root;
}
```

Он хранит ссылки на тэг, с которым идёт работа в данный момент и корневой узел документа.

4. Реализация XmlPath

XmlPath для обработки запросов использует тот же XmlLexer, который использовался в XmlParser, сначала ищет все узлы, соответствующие указанному пути, затем фильтрует их в соответствии с заданными параметрами:

```
void XmlPath::Request::parsePath(XmlLexer::Lexer& lexer) {
    lexer.next();
    while (lexer.current().getType() != XmlLexer::Token::Type::OP_SQUARE &&
           lexer.current().getType() != XmlLexer::Token::Type::END) {
        lexer.expect({XmlLexer::Token::Type::SLASH});
        lexer.next();
        lexer.expect({XmlLexer::Token::Type::WORD});
        path.emplace_back(lexer.current().getText());
        lexer.next();
```

```
    }
}

void XmlPath::Request::parseFilters(XmlLexer::Lexer& lexer) {
    lexer.next();
    while (lexer.current().getType() != XmlLexer::Token::Type::CL_SQUARE) {
        Filter filter{};
        if (lexer.current().getType() == XmlLexer::Token::Type::BANG) {
            filter.negate = true;
            lexer.next();
        }
        lexer.expect({XmlLexer::Token::Type::WORD});
        auto fby = lexer.current().getText();
        if (fby == "attr") filter.by = Filter::By::ATTR;
        else if (fby == "text") filter.by = Filter::By::TEXT;

        lexer.next();
        lexer.expect({XmlLexer::Token::Type::OP_PAREN});
        lexer.next();
        if (filter.by == Filter::By::ATTR) {
            lexer.expect({XmlLexer::Token::Type::STRING});
            filter.attrKey = lexer.current().getText();
            lexer.next();
            lexer.expect({XmlLexer::Token::Type::COMMA});
            lexer.next();
            lexer.expect({XmlLexer::Token::Type::STRING});
            filter.value = lexer.current().getText();
        } else {
            lexer.expect({XmlLexer::Token::Type::STRING});
            filter.value = lexer.current().getText();
        }
        lexer.next();
        lexer.expect({XmlLexer::Token::Type::CL_PAREN});
        lexer.next();
        switch (lexer.current().getType()) {
            case XmlLexer::Token::Type::DBL_AMPERSAND:
                filter.oper = Filter::Operator::AND;
                lexer.next();
                break;
            case XmlLexer::Token::Type::DBL_BAR:
                filter.oper = Filter::Operator::OR;
                lexer.next();
                break;
            default:
                filter.oper = Filter::Operator::NO;
                break;
        }
        filters.emplace_back(filter);
    }
}
```

Сам алгоритм поиска и фильтрации:

```
std::vector<std::shared_ptr<Xml::Tag>> XmlPath::Holder::find(XmlPath::Request req) {
    std::vector<std::shared_ptr<Xml::Tag>> tags = findInVector(document.getChildren(),
req.path[0]);
    for (int i = 1; i < req.path.size(); i++) {
        tags = findInVector(tags[0]->getChildren(), req.path[i]);
```

```
    }
    if (!req.filters.empty()) {
        tags = filter(req, tags);
    }
    return tags;
}

std::vector<std::shared_ptr<Xml::Tag>> XmlPath::Holder::filter(XmlPath::Request req,
std::vector<std::shared_ptr<Xml::Tag>> tags) {
    int filterNumber = 0;
    std::vector<std::shared_ptr<Xml::Tag>> prev;
    bool interrupt = false;
    while (!interrupt) {
        if (req.filters[filterNumber].oper == Filter::Operator::NO) {
            interrupt = true;
        }
        auto filter = req.filters[filterNumber++];
        auto filtered = filter.filter(tags);
        if (filterNumber == 1) { prev = filtered; continue; }
        auto oper = req.filters[filterNumber - 1].oper;
        switch (oper) {
            case Filter::Operator::OR:
                for (auto& elem : filtered) {
                    if (std::find(prev.begin(), prev.end(), elem) == prev.end()) {
                        prev.emplace_back(elem);
                    }
                }
                break;
            case Filter::Operator::AND:
                for (auto& elem : prev) {
                    if (std::find(filtered.begin(), filtered.end(), elem) == prev.end()) {
                        filtered.erase(std::remove(filtered.begin(), filtered.end(), elem));
                    }
                }
                prev = filtered;
                break;
        }
    }
    return prev;
}

std::vector<std::shared_ptr<Xml::Tag>>
XmlPath::Holder::findInVector(std::vector<std::shared_ptr<Xml::Tag>> vect, std::string
name) {
    std::vector<std::shared_ptr<Xml::Tag>> tags;
    for (auto& tag : vect) {
        if (tag->getName() == name) {
            tags.emplace_back(tag);
        }
    }
    return tags;
}
```