## Содержание

[Содержание 1](#_Toc1)

[Введение 2](#_Toc2)

[Список задач 3](#_Toc3)

[Описание решенных задач 5](#_Toc4)

[Заключение 11](#_Toc5)

[Библиографический список 12](#_Toc6)

[Приложения 13](#_Toc7)

## Введение

Учебная практика проходила в ФГБОУ ВО «Вятский государственный университет», на кафедре прикладной математики и информатики факультета компьютерных и физико-математических наук с ... по ..., количество недель: 12. Установочная конференция состоялась ...

Целью учебной практики является закрепление и углубление теоретической подготовки, приобретение практических навыков и компетенций в сфере профессиональной деятельности, в основном путём самостоятельного решения предусмотренных программой задач.

Форма проведения учебной практики: компьютерная (практикум по решению задач на ЭВМ).

В ходе прохождения практики необходимо выполнить задания. Используемый язык программирования: Java, среда программирования Intellij Idea.

Решенные задачи выложены в систему контроля версий Git.

## Список задач

Формулировка индивидуального задания: — список заданий, которые нужно выполнить.

**Задание 1. Регулярные выражения**

Составьте регулярные выражения для следующих задач:

**Задача 1.** Время имеет формат часы: минуты. И часы, и минуты

состоят из двух цифр, пример:

09:00. Напишите регулярное

выражение для поиска времени в

строке.

– пример правильных

выражений: Завтрак в 09:00.

– пример неправильных

выражений: 37:98, 24:00.

**Задача 2.** Написать регулярное выражение,

определяющее является ли

заданная строка правильным

MAC-адресом.

– пример правильных

выражений: aE:dC:cA:56:76:54.

– пример неправильных

выражений: 01:23:45:67:89:Az.

**Задача 3.** Написать регулярное выражение,

определяющее является ли

данная строчка валидным E-mail

адресом согласно RFC под

номером 2822.

– пример правильных

выражений: user@example.com,

root@localhost

– пример неправильных

выражений: bug@@@com.ru,

@val.ru, Just Text2.

**Задача 4.** Проверить, надежно ли составлен пароль. Пароль считается надежным, если

он состоит из 8 или более символов. Где символом может быть английская

буква, цифра и знак подчеркивания. Пароль должен содержать хотя бы одну

заглавную букву, одну маленькую букву и одну цифру.

– пример правильных выражений: C00l\_Pass, SupperPas1.

– пример неправильных выражений: Cool\_pass, C00l.

**Задание 2. Написать сервис преобразования документов (данных) из формата**

**XML в формат JSON и обратно.**

Задача 1. Определите предметную область. Создайте XML-документ, хранящий

данные об этой предметной области. Вложенность документа не меньше 4-х.

Задача 2. Выберите библиотеку(и) для работы с форматами XML и JSON.

Задача 3. Создайте проект (репозиторий) FileConverterService. Этот проект должен

решать 2 задачи: преобразование xml в json и обратное преобразование.

**Задание 3. Собрать данные**

Задача 1) Собрать отзывы об интернет магазинах <https://nanegative.ru/internet-magaziny>

Задача 2) Скачать картинки на какую-то тему. Например, карандаши цветные, цвет индиго, кенгуру. Запрос ввести в строку поиска

**Задание 4. Stream**

Задача 1. Возьмите список из классов, описанных в задании 2 (прототипом для класса

может служить как xml, так и json файл).

Задача 2. Напишите 5 различных методов, содержащих одну или несколько

следующих операций: map, filter, forEach, sorted, collect. Попробуйте не

только выполнить сортировку и выборку данных, но и придумать их

преобразование.

## Описание решенных задач

**Задание 1.**

Регулярное выражение для каждой задачи. Тестовые строки, для которых регулярное выражение работает и для которых не работает.

Задача 1

|  |  |
| --- | --- |
| Пример подходящей строки | Пример неподходящей строки |
| Завтрак в 09:00 | Завтрак в 8:60 |
| Обед в 12:00 | Обед в 24:00 |
| Ужин в 18:00 | Ужин в 25:04 |
| 01:59 | 23 44 |
| 23:55 | Обед в 10:61 |
| 19:30 | Ужин в десять |

Задача 2

|  |  |
| --- | --- |
| Пример подходящей строки | Пример неподходящей строки |
| aE:dC:cA:56:76:54 | ab:cd:ef:AB:CD:EG |
| aa:bb:cc:dd:ee:ff | 01:23:45:67:89:Az |
| ab:cd:ef:AB:CD:E1 | aE:dC:cP:56:76:54 |
| ba:c4:f3:88:55:1A | 01:23:45:67:89:Az |
| DD:EE:FF:33:44:F1 | 01:23:45:FJ:89:Az |
| Ab:Cd:Ef:Ca:A5:67 | ba:c4:f3:88:55:101 |

Задача 3

|  |  |
| --- | --- |
| Пример подходящей строки | Пример неподходящей строки |
| root@localhost | bug@@@com.ru |
| user@example.com | @val.ru |
| sfdfrdfdk@google.ru | Just Text2 |
| gabenlox@gmail.com | sd fe asqsq data |
| sddfx@google.com | incorrect@yandex. dsw |
| data@localhost | sdd.ru |

Задача 4

|  |  |
| --- | --- |
| Пример подходящей строки | Пример неподходящей строки |
| C00l\_Pass | Cool\_pass |
| SupperPas1 | C00l |
| Ruby2001ArtemkaPROG2077 | MaksimMaks |
| sdsfQWQ123 | \_\_\_\_\_Pass |
| lgffdfs\_23\_AS | 234510030453 |
| FGLK343smkkm\_\_12 | qwertyuiop123 |

**Ссылка на Github:** <https://github.com/rubysln/RegularJava>

**Задание 2.**

XML-файл, JSON-файл. Описание способов работы с файлами формата XML и JSON. Обоснование своего выбора способа работы с файлами. Обзор библиотек, обоснование выбора библиотеки.

Мною был создан XML-файл по тематике библиотек с книгами, который в ходе выполнения работы программы изменяет свою структуру в список авторов имеющих книги из изначальных библиотек. Для десериализации XML-файла на первом этапе я использовал StAX, для де/сериализации JSON использовал библиотеку от GOOGLE – GSON, для сериализации в XML-файл использовал JAXb.

**StAX**

Мною был выбран StAX для первого этапа исключительно из-за возможности прописать вручную handler для парсера, при помощи StAX, я добился не только десериализации XML-файла, но и изменил структуру работая с новыми объектами. В ходе алгоритма handler не только работает с XML-файлом пробегаясь по тегам, но еще и сохраняет полученные данные в нужные для изменения структуры объекты.

**GSON**

Документация: <http://www.studytrails.com/java/json/java-google-json-introduction/>

GSON как мне кажется самая удобная, самая простая и самая приятная библиотека для работы с JSON, для де/сериализации JSON, мне в методе пришлось всего лишь объявить объект GSON, и при помощи метода .toJSON был создан уже сериализованный файл с нужными мне строками, в отличии от конкурентов GSON потребляет меньше памяти в работе с объектами и при тестировании работы, GSON показал самый стабильный результат и полное отсутствие багов. C его помощью я выполнил де/сериализацию JSON.

**JAXB**

JAXB так же, как и GSON отличная библиотека для работы с XML. Но есть некоторые нюансы, при работе с JAXB я столкнулся с трудностями подключения и адекватной работы данной библиотеки, ведь её буквально искоренили из 11 Java, поэтому в pom.xml вы не найдете прямого подключения JAXB, вместо него там будет Jakarta и GlassFish.

Идя по порядку, у нас есть десериализированное POJO после работы GSON, Следовательно нам нужно изменить структуру данных обратно и сериализировать их в XML файл. Почему я использую именно JAXB. На первом шаге исключительно важно было сохранять данные из XML в POJO изменяя при этом структуру, следовательно нужно было вручную прописывать handler, в сериализации ситуация немного другая, у нас уже есть объекты, написан алгоритм изменения структуры и требуется всего лишь сериализовать их в файл, поэтому прописывать снова handler для создания правильного XML-файла не требуется.

JAXB отличается от GSON удобной работой с аннотациями, достаточно просто указать root object для XML, указать поля которые будут элементами в документе и JAXB наравне с GSON парсит объекты в файл формата XML, при работе с ним так же не встретил багов.

**Ссылка на Github:** <https://github.com/rubysln/FileConverterService>

**Задание 3.**

Описание данных, которые нужно собрать, запросы для получения данных. Особенности сбора данных с каждого ресурса.

Для первой задачи выбрал интернет магазины -

Для парсинга использовал библиотеку JSOUP(https://mvnrepository.com/artifact/org.jsoup/jsoup/1.15.3)

В своей реализации мне понадобилось парсить сначала главный URL, чтобы посмотреть количество страниц магазинов, и собрать все магазины в 1 список данных.

Далее по запросу пользователя парсить отзывы с конкретного сайта, выводя ответ в виде класса, если же отзывы имеются на сайте. Для нахождения нужных мне данных использовал xpath.

**Ссылка на Github:** <https://github.com/rubysln/WebScraping>

**Задание 4.**

Работа со Stream API. Сортировка, выборка и преобразование данных с операциями map, filter, forEach, sorted, collect.

В ходе десериализаций из заданий 3-6 недели мною уже были созданы методы получения данных с файлов, следовательно в классе Stream API, для получения данных используется конструктор в который необходимо передать данные класса Library для дальнейшей работы.

**Map**

В первом методе была сделана выборка данных по количеству страниц, метод возвращает коллекцию Integer переменных.

books.stream().map(Book::getPages).collect(Collectors.toList());

**Filter**

Второй метод реализует фильтрацию данных по входящим переменным и выводит коллекцию объектов класса Book которые выполняют требованиям, а именно больше какого-то входного количества страниц и новее какого-то входного года публикации.

books.stream().filter(Book -> Book.getPages() > underThanPages && Book.getPublicationYear() > underThanPublicationYear).collect(Collectors.toList());

**ForEach**

В третьем методе в каждую книгу добавляется какой-то жанр, какой именно решает сам пользователь, я в качестве примера добавил в каждую книгу жанр “Классика”.

books.stream().forEach(Book -> Book.setGenres(genreName));

**Sorted**

В четвертом методе Sorted метод сортирует данные по году публикации, от самой старой книги в коллекции до самой новой.

books.stream().sorted(Comparator.comparing(Book::getPublicationYear)).collect(Collectors.toList());

**Collect**

В заключительном - пятом Collect методе входные данные изменяются из List в Set, сортировка же идет по авторам, что позволяет убрать дубликаты.

books.stream().map(Book::getAuthor).collect(Collectors.toSet());

**Ссылка на Github:** <https://github.com/rubysln/FileConverterService>

## Заключение

Учебная практика способствовала закреплению и углублению теоретической подготовки, приобретению практических навыков и компетенций в сфере профессиональной деятельности путём самостоятельного решения задач.

В ходе практики были решены следующие задания на языке программирования... Листинги приложения приведен в приложении.

Наиболее интересным оказалось работать со Stream API, использование более продвинутых функций, нежели работа с обычными массивами, сортировка данных, выборка и другие функции библиотеки очень мощные и пригождаются в работе.

Наиболее сложным оказалось парсинг XML и JSON, потому что не во всех библиотеках есть документация, и приходилось читать информацию со сторонних источников.

## Библиографический список

1. Мархакшинов А. Л. Практикум по программированию на языке Java : практикум / Улан-Удэ : БГУ, 2017. - 70 с.
2. Блох, Д. Java Эффективное программирование / Д. Блох. - М.: Лори, 2016. - 440 c.
3. Васильев, А.Н. Java. Объектно-ориентированное программирование: Учебное пособие: для магистров и бакалавров. Базовый курс по объектно-ориентированному программированию / А.Н. Васильев. - СПб.: Питер, 2013. - 400 c.

## Приложения

**Задание 1.**

public class RegularJava {

public boolean timeCheck(String line){

String[] inputArray = line.split("\\s+");

for(var e: inputArray){

if(e.matches("\\d+:\\d+")){

String[] numbers = e.split(":");

return Integer.parseInt(numbers[0]) < 24 && Integer.parseInt(numbers[1]) < 60;

}

}

return false;

}

public boolean macAdressCheck(String line){

return line.matches("([A-Fa-f0-9]{2}:){5}[A-Fa-f0-9]{2}");

}

public boolean emailAdressCheck(String line){

return line.matches("([A-Za-z0-9]+@[^@].+\\.[a-z])|([A-Za-z0-9]+@[^@].+)");

}

public boolean passwordCheck(String line){

return line.matches("(?=.\*[a-z])(?=.\*[A-Z])(?=.\*[0-9])[A-Za-z0-9\_]{8,}");

}

}

**Задание 2.**

**Класс Parser**

public class Parser {

public Authors parseFromXML(String input) throws ParserConfigurationException, SAXException, IOException {

/\*

В данном методе мы десериализируем файл XML, изменяем его структуру и сохраняем в объект.

\*/

SAXParserFactory saxParserFactory = SAXParserFactory.newInstance(); // Создание фабрики для работы с парсером.

ParserHandler handler = new ParserHandler(); // Инициализируем наш хандлер для ручного парсинга файла.

SAXParser saxParser = saxParserFactory.newSAXParser(); // Создаем парсер.

File file = new File(input);

saxParser.parse(file, handler);

Authors authors = handler.getAuthors();

return endParse(authors);

}

private Authors endParse(Authors authors){

/\*

Данный метод необходим, потому что мы из библиотеки книг будем делать список авторов с их книгами, внутри метода

при помощи алгоритма дубликаты авторов в коллекции будут удалены, а книги дубликатов присвоены оригиналам.

\*/

for(int i = 0; i < authors.getAuthors().size(); i++){

Author author = authors.getAuthors().get(i);

String name = author.getName();

for(int j = 0; j < authors.getAuthors().size(); j++){

if(i == j) continue;

else{

Author secondAuthor = authors.getAuthors().get(j);

String secondName = secondAuthor.getName();

if(name.equals(secondName)){

author.getBooks().add(secondAuthor.getBook());

authors.getAuthors().remove(secondAuthor);

}

}

}

}

return authors;

}

public void parseToJson(Authors authors, String output) throws IOException {

/\*

Для сериализации была выбрана библиотека GSON, удобство, простота, отсутствие багов после тестирования в созданном файле.

\*/

Gson gson = new Gson();

Files.write(Path.of(output + ".json"), gson.toJson(authors).toString().getBytes());

}

public Library parseFromJson(File file) throws IOException {

/\*

Для десериализации была выбрана та же библиотека GSON по тем же причинам, идеальная работа с .json файлами!

\*/

Gson gson = new Gson();

FileReader fileReader = new FileReader(file);

Authors authors = gson.fromJson(fileReader, Authors.class);

List<Book> books = new ArrayList<>();

for(var e:authors.getAuthors()){

for(var j:e.getBooks()){

j.setAuthor(e.getName());

books.add(j);

}

}

return new Library(books);

}

public void parseToXML(Library library, String output) throws JAXBException {

/\*

Тут уже интереснее..

Для сериализации в файл XML был выбран JAXB, т.к в ручную прописывать хандлер не имеет смысла, мною это было сделано без использования лишних библиотек,

вручную.

А JAXB как по мне отличный аналог для де/сериализации XML-файлов, имеет похожий принцип построения кода как GSON, за исключением того,

что для работы сериализации в файл необходимо прописывать в объектах аннотации, но как по мне, это очень даже удобно.

\*/

JAXBContext context = JAXBContext.newInstance(Library.class);

Marshaller marshaller = context.createMarshaller();

marshaller.setProperty(Marshaller.JAXB\_FORMATTED\_OUTPUT, Boolean.TRUE);

marshaller.marshal(library, new File(output));

}

}

**Класс ParserHandler**

public class ParserHandler extends DefaultHandler {

/\*

В данном классе описан принцип парсинга из XML-файла в объектную модель, библиотека StAX была выбрана не случайна, при помощи неё

можно сразу избирать данные из тегов и помещать в конкретные классы, тем самым я добился не только десериализации XML-файла, но еще и изменил его структуру,

поместив необходимые данные сразу классы измененной структуры.

\*/

private static final String TAG\_LIBRARY = "library";

private static final String TAG\_BOOK = "book";

private static final String TAG\_AUTHOR = "author";

private static final String TAG\_TITLE = "title";

private static final String TAG\_DETAILS = "details";

private static final String TAG\_GENRES = "genres";

private static final String TAG\_PAGES = "pages";

private static final String TAG\_PUBLICATIONYEAR = "publicationYear";

private static final String TAG\_ABOUT = "about";

// Выше представленны тэги из XML-файла для быстрой идентификации внутри десериализации.

private boolean isBook = false;

private boolean isDetails = false;

/\*

Выше представлены логические переменные которые помогают алгоритму хандлера понимать где конкретно происходит

извлечение данных из файла.

\*/

// Ниже представлены искомые внутри файла объекты для десериализации.

private Authors authors = new Authors();

private List<Author> authorList = new ArrayList<>();

private Author author;

// Ниже представлены автономные объекты не требующие изменения структуры.

private List<Book> books = new ArrayList<>();

private Book book = new Book();

private Details details = new Details();

// Ниже представлен необходимый для получения итогового объекта геттер.

public Authors getAuthors() {

return authors;

}

private String currentTagName;

@Override

public void startDocument() throws SAXException {

System.out.println("Start parsing");

}

@Override

public void endDocument() throws SAXException {

System.out.println("End parsing");

authors.setAuthors(authorList); // Финальный этап работы алгоритма - присваивание значений коллекции авторов в объект Authors, который мы и должны получить.

}

@Override

public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {

System.out.println("Loading...");

currentTagName = qName; // Присваивание переменной название текущего тега.

if(currentTagName != null){

// Проверка на null и ниже проверка на нахождения тега внутри внешних тегов.

if(currentTagName.equals(TAG\_BOOK)) isBook = true;

else if(currentTagName.equals(TAG\_DETAILS)) isDetails = true;

}

}

@Override

public void endElement(String uri, String localName, String qName) throws SAXException {

System.out.println("Loading...");

currentTagName = qName; // Повторное присваивание значения необходимо, т.к когда идет закрытие нескольких тегов подряд, currentTagName не обновляется.

if(currentTagName == null) return; // Проверка на null

else{

if(currentTagName.equals(TAG\_BOOK)) {

isBook = false;

books.add(book);

author.setBooks(books);

authorList.add(author);

// Выше алгоритм добавляет все необходимые данные в искомые и автономные объекты, ниже - обнуляет для присваивания данных из новых тегов.

book = new Book();

books = new ArrayList<>();

}

else if(currentTagName.equals(TAG\_DETAILS)){

isDetails = false;

// Выше алгоритм добавляет все необходимые данные в искомые и автономные объекты, ниже - обнуляет для присваивания данных из новых тегов.

book.setDetails(details);

details = new Details();

};

currentTagName = null; // Обнуление переменной в которой хранится текущий тег.

}

}

@Override

public void characters(char[] ch, int start, int length) throws SAXException {

if(currentTagName == null){

return; // Проверка на null с целью исключения ошибок эксепшена.

}

/\*

Ниже представлен алгоритм создания и / или присваивания найденных значений внутри тега - созданным выше переменным.

\*/

if(isBook){

switch (currentTagName){

case (TAG\_AUTHOR):

author = new Author(new String(ch, start, length));

break;

case (TAG\_TITLE):

book.setTitle(new String(ch, start, length));

break;

}

if(isDetails){

switch (currentTagName){

case (TAG\_GENRES):

details.setGenres(new String(ch, start, length));

break;

case(TAG\_PAGES):

details.setPages(Integer.valueOf(new String(ch, start, length)));

break;

case(TAG\_PUBLICATIONYEAR):

details.setPublicationYear(Integer.valueOf(new String(ch, start, length)));

break;

case(TAG\_ABOUT):

details.setAbout(new String(ch, start, length));

break;

}

}

}

}

}

**Задание 3.**

**Задача 1.**

public class Scrapping {

public static void doScrapping() throws IOException, IndexOutOfBoundsException {

List<Integer> numberOfShop = new ArrayList<>();

Document docMain = Jsoup.connect("https://nanegative.ru/internet-magaziny").get();

HashMap<Integer, String> mapShops = new HashMap<>();

System.out.println(docMain.title());

Elements pagesCount = docMain.selectXpath("//a[starts-with(@href,'/internet-magaziny?page=')]");

for(int i = 1; i <= pagesCount.size(); i++){

Document doc = Jsoup.connect("https://nanegative.ru/internet-magaziny?page=" + i).get();

Elements postTitleElement = doc.getElementsByAttributeValue("class", "ss");

Elements postRateElement = doc.getElementsByAttributeValue("class", "sro");

Elements postRateCountElement = doc.getElementsByAttributeValue("class", "reviewers");

for(int j = 0; j < postTitleElement.size(); j++){

Element rateCountElement = postRateCountElement.get(j);

Element titleElement = postTitleElement.get(j);

Element rateElement = postRateElement.get(j);

String str = titleElement.text().replace("Отзывы о", "");

System.out.println((numberOfShop.size()) + ") " + str + " | " + rateElement.text() + " | " + rateCountElement.text());

numberOfShop.add(1);

mapShops.put(j, titleElement.attr("href"));

}

}

System.out.println("Введите номер интересующего магазина: ");

Scanner scanner = new Scanner(System.in);

int inputShop = scanner.nextInt();

docMain = Jsoup.connect("https://nanegative.ru" + mapShops.get(inputShop)).get();

var pages = docMain.selectXpath("//a[starts-with(@href,'" + mapShops.get(inputShop) + "?page=')]");

if(pages.size() == 0){

Elements reviews = docMain.getElementsByAttributeValue("itemprop", "description");

if(reviews.size() == 0){

System.out.println("Отзывы о данном магазине отсутствуют!");

}

else{

for(var e: reviews){

System.out.println(e.text());

}

}

}

else{

for(int i = 0; i < pages.size(); i++){

docMain = Jsoup.connect("https://nanegative.ru" + mapShops.get(inputShop) + "?page=" + i).get();

Elements reviews = docMain.getElementsByAttributeValue("itemprop", "description");

for(var e: reviews){

System.out.println(e.text());

}

}

}

}

}

**Задача 2.**

public class Picture {

static String imageFolder = "C:\\Java\\WebScraping\\images";

public static void downloadPicture(String request) throws IOException {

if(request.matches(".+\\s+.+")) request = request.replace(' ','+');

Document doc = Jsoup.connect("https://www.google.ru/images?q=" + request).get();

Elements pictures = doc.selectXpath("//img[contains(@class,'rg\_i')]");

for(Element e:pictures){

if(e.hasAttr("data-src")){

String pictureURL = e.attr("data-src");

downloadPic(pictureURL);

}

}

}

public static void downloadPic(String url) throws IOException {

Random random = new Random();

String pictureName = random.toString() + ".jpg";

System.out.println("Saving: " + pictureName + ", from: " + url);

URL urlImage = new URL(url);

InputStream inputStream = urlImage.openStream();

byte[] buffer = new byte[4096];

int n = -1;

OutputStream os =

new FileOutputStream( imageFolder + "\\" + pictureName );

while ( (n = inputStream.read(buffer)) != -1 ){

os.write(buffer, 0, n);

}

os.close();

System.out.println("Saved!");

}

}

**Задание 4.**

public class StreamAPI {

private Library library;

private List<Book> books;

public StreamAPI(Library library) {

this.library = library;

this.books = library.getBook();

}

public List<Integer> map(){

return books.stream().map(Book::getPages).collect(Collectors.toList());

}

public List<Book> filter(Integer underThanPages, Integer underThanPublicationYear){

return books.stream().filter(Book -> Book.getPages() > underThanPages && Book.getPublicationYear() > underThanPublicationYear).collect(Collectors.toList());

}

public List<Book> forEach(String genreName){

books.stream().forEach(Book -> Book.setGenres(genreName));

return books;

}

public Set<String> collect(){

return books.stream().map(Book::getAuthor).collect(Collectors.toSet());

}

public List<Book> sorted(){

return books.stream().sorted(Comparator.comparing(Book::getPublicationYear)).collect(Collectors.toList());

}

}