

Министерство науки и высшего образования РФ  
ФГАОУ ВПО  
Национальный исследовательский технологический университет  
«МИСИС»

---

Институт Информационных технологий и компьютерных наук (ИТКН)  
Кафедра Автоматизированных систем управления (АСУ)

**Отчет по курсовой работе**  
по дисциплине «Технологии программирования»  
на тему «Разработка клиент-серверного приложения»

Выполнила:  
Студентка: Жиделёва София

Группа: БИВТ-20-4

Проверил:  
Козлов М. Е.

Москва, 2021

## Оглавление

1. Введение	3
2. Постановка задачи	4
2.3 Детальные требования	4
3. Теоретическая часть	5
3.1 Описание формата CSV	5
3.2. Обоснование выбора представленных в работе технологий	6
3.3 Выбор СУБД.	6
4. Практическая часть	9
4.1 Описание работы программы.	9
4.2 Реализация программы	9
4.3 Тестирование приложения	18
5. Заключение	21
6. Список литературы	22

## 1. Введение

В наше время, чем удобнее и разнообразнее функционал приложения, тем более удобно с ним обращаться. Поэтому клиент-серверные приложения набирают популярность по использованию в различных областях и используются повсеместно. Причем по такому принципу совмещения интернета и баз данных устроено большинство современных программных продуктов. Из-за растущей популярности клиент-серверных приложений. Растёт потребность в современных языках, на которых будет удобно работать как с базой данных, так и с вебом, не прибегая к погружению в изучение их языков.

На практике это проявляется в том, что при работе определенных баз данных с некоторыми приложениями возникают ошибки, которые могут быть как явными, так и скрытыми, причем второй вариант более опасен, поскольку такие несоответствия со временем будут накапливаться, и может оказаться, что к моменту обнаружения проблемы база данных будет серьезно повреждена.

Если учитывать, что такие программные продукты часто ориентированы на оптимизацию бизнес-процессов, возникновение ошибок может повлечь за собой серьезные финансовые потери.

Одним из таких приложений, позволяющих корректно осуществлять связь базы и веба, станет наше, написанное на языке Java, клиент-серверное приложение.

## **2. Постановка задачи**

Задача заключается в разработке клиент-серверного приложения со следующим функционалом:

- Работа с данными в csv файлах;
- Автоматическое создание таблиц в базе данных;
- Веб-страница для загрузки данных в таблицы базы данных из csv файлов;
- Вывод результатов в csv файлы;
- Вывод ошибок в консоль;
- Веб-страница с формой для регистр независимого текстового поиска по описанию транзакций с выводом на ту же страницу в виде таблицы;

### **2.3 Детальные требования**

Приложение должно осуществлять чтение данных из CSV файла, заносить полученную информацию в связанную с приложением базу данных PostgreSQL и предоставлять возможность записи новых данных в базу через форму на веб-странице, поиск по базе по введённой части слова без учёта регистра.

CSV файл представляет собой построчный набор данных записанный соответственно (id,firstName,secondName,profession).

### 3. Теоретическая часть

#### 3.1 Описание формата CSV

CSV (comma-separated value) — это формат представления табличных данных (например, это могут быть данные из таблицы или данные из БД).

В этом формате каждая строка файла — это строка таблицы.

Однако разделителем может быть не только запятая, несмотря на название формата. Хотя форматы с другими разделителями может быть и собственное название, например, TSV (tab separated values), всё-таки, под форматом CSV понимают любые разделители.

Правила содержимого для строк и столбцов: столбцы первой строки.

Тип артефакта – обязательное поле. Каждая строка в файле CSV должна содержать в этом столбце запись, соответствующую типу артефакта в системе.

Имя – обязательное поле, если пропущен основной текст. По умолчанию каждый артефакт в системе имеет поле Имя. Данные в этом поле отображаются при просмотре структур папок или просмотре результатов поиска и фильтрации.

Основной текст – обязательное поле, если пропущено имя. Каждый текстовый артефакт в системе содержит поле Основной текст, соответствующий основному материалу, связанному с артефактом. Например, если артефакт является требованием, то его основной текст представляет фактическое требование, такое как "Механизм должен иметь два колеса".

Описание – рекомендуется. По умолчанию каждый артефакт в системе имеет поле Описание. Описание отображается во всплывающей подсказке при наведении курсора на заголовок.

Если ячейка содержит материалы для столбца, не соответствующие типу артефакта, заданному в значении Тип, ячейка

будет проигнорирована. Даже если импорт CSV завершится, определенные данные могут быть не включены, так как соответствующие столбцы не существуют для типа артефакта.

### **3.2. Обоснование выбора представленных в работе технологий**

Java — сильно типизированный объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, с помощью виртуальной Java-машины.

Программы на Java транслируются в байт-код Java, выполняемый виртуальной машиной Java (JVM) — программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор.

Достоинством подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина.

Ещё одной важной особенностью технологии Java является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером), вызывают немедленное прерывание.

### **3.3 Выбор СУБД.**

В качестве используемой системы управления базами данных была выбрана PostgreSQL.

PostgreSQL — это свободно распространяемая объектно-реляционная система управления базами данных, наиболее развитая из открытых СУБД в мире и являющаяся реальной альтернативой коммерческим базам данных. PostgreSQL, как видно из названия, базируется на языке SQL и поддерживает многие из возможностей стандартов SQL.

У PostgreSQL выделяют множество сильных сторон, например, она выполняет высокопроизводительные и надёжные механизмы транзакций, постоянно расширяет систему встроенных языков программирования, поддерживает наследование, имеет встроенную поддержку слабоструктурированных данных в формате JSON с возможностью индексации и т. д.

База данных отлично подходит под поставленные задачи, быстро и безопасно обрабатывает направляемые в базу данных запросы, возвращает данные, записывает и выгружает их из файлов.

В комбинации с Java данная система (PostgreSQL) почти мгновенно направляет запрос и возвращает полученный результат. Интегрировать базу данных в среду разработки не составляет особого труда с представленными самой средой функциями, позволяющими не устанавливать дополнительный софт.

### 3.4 Spring

Spring для Java — обширная платформа для создания веб-проектов на Java, состоящая из множества независимых модулей (проектов) для разных задач: от простых веб-приложений до Big Data.

Spring возник в виде облегчённого аналога платформы для корпоративных приложений Java Enterprise, позиционируется как простая в использовании платформа для веб-приложений. Spring поддерживает несколько языков JVM: Java, Kotlin и Groovy.

Spring Boot — комплексный фреймворк для создания и запуска приложений с минимальными усилиями и настройками. Этот модуль делится на два стека: основанный на API сервлетов Spring MVC и реактивный Spring WebFlux.

Spring WebFlux — веб-платформа, созданная, чтобы по максимуму использовать преимущества современных многоядерных процессоров и обрабатывать огромное количество одновременных подключений.

Основной механизм, реализуемый в Spring Data — репозиторий. Это набор интерфейсов, использующих JPA Entity для взаимодействия с данными. Spring Data используется везде, где нужен доступ к данным, и легко интегрируется с другими модулями Spring.



## 4. Практическая часть

### 4.1 Описание работы программы.

Программа должна уметь считывать данные из файла, лежащему в одной папке с проектом и записывать полученные данные в базу данных. У пользователя имеется три веб-страницы, через которые он может посмотреть базу данных, записать новую информацию и осуществить поиск по полю фамилия.

### 4.2 Реализация программы

```
package com.example.course_work;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class CourseWorkApplication {

    public static void main(String[] args) {
        SpringApplication.run(CourseWorkApplication.class, args);
    }

}
```

Рисунок 1. Класс CourseWorkApplication и метод main

#### Класс User

Данный класс создан для описания объекта Пользователь по четырём полям: ID, Имя, Фамилия и Профессия.

Класс будет удобно использовать при добавлении значений в таблицу и оперированием данными полями по всей программе.

```

package com.example.course_work.Model;

import lombok.*;

import javax.persistence.*;

@Entity
@Table(name = "account")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id; //ссылочный, чтоб мог быть NULL

    private String firstName;
    private String lastName;
    private String profession;
}

```

Рисунок 2. Класс User

### Класс UserController

Класс был создан для непосредственной обработки запросов от клиента и возвращения результата. В данном случае идёт демонстрация веб страницы **users\_page**, в которой выводится в таблицу все данные, которые хранятся в нашей базе данных.

```

package com.example.course_work.Controller;

import com.example.course_work.Model.User;
import com.example.course_work.repo.UsersRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

import java.util.List;

@Controller
public class UserController {

    @Autowired
    private UsersRepo usersRepo;

    @GetMapping("/users")
    public String getUsersPage(Model model){
        List<User> users = usersRepo.findAll();
        model.addAttribute("users", users);
        return "users_page";
    }
}

```

Рисунок 3. Класс UserController

### Интерфейс UsersRepo

```

package com.example.course_work.repo;

import com.example.course_work.Model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

public interface UsersRepo extends JpaRepository<User, Long> {
    // Ignore Case гарантирует что запрос не зависит от регистра
    @Query(value = "select * from where lastName.account like :lastName ", nativeQuery = true)
    User findByLastNameIgnoreCase(String lastName);
}

```

Рис. 4, интерфейс UsersRepo

Интерфейс был создан для его последующей имплементации.

### Веб-страница users\_page

На данной странице в таблицу выводятся столбцы с полями из базы данных.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
    content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
<div>
  <table>
    <tr>
      <th>ID</th>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Profession</th>
    </tr>
    <#list users as user>
      <tr>
        <td>${user.id}</td>
        <td>${user.firstName}</td>
        <td>${user.lastName}</td>
        <td>${user.profession}</td>
      </tr>
    </#list>
  </table>
</div>
</body>
```

Рисунок 5. Код страницы users\_page

### Класс AddUsersController

Класс для считываний данных на открытой веб-странице **addUsers\_page**.

```

package com.example.course_work.Controller;

import com.example.course_work.Model.User;
import com.example.course_work.WorkWithFile.AddToFile;
import com.example.course_work.repo.UsersRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;

@Controller
public class AddUsersController {

    @Autowired
    private UsersRepo usersRepo;

    @GetMapping("/addUsers")
    public String getAddUsersPage() { return "addUsers_page"; }

    @PostMapping("/addUsers")
    public String addUser_(User user){
        usersRepo.save(user);
        AddToFile.WriteToFile( data: user.getId()+
            user.getFirstName()+
            user.getLastName()+
            user.getProfession());
        return "redirect:/addUsers"; //перенаправляется снова к странице add_users
    }
}

```

Рисунок 6. Класс AddUserController

### Веб-страница addUsers\_page

Эта страница является мини-формой для добавления информации в базу данных.

```

<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
    content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Document</title>
</head>
<body>
  <div>
    <form action="/addUsers" method="post">
      <input name="firstName" placeholder="Имя">
      <input name="lastName" placeholder="Фамилия">
      <input name="profession" placeholder="Профессия">
      <input type="submit" value="AddUser">
    </form>
  </div>
</body>
</html>

```

Рисунок 7. Код страницы addUsers\_page

### Класс SignInController

Класс обрабатывает действия пользователя на веб-странице **signIn\_page**

```
package com.example.course_work.Controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class SignInController {
    @GetMapping("/signIn")
    public String getSignInPage(){
        return "signIn_page";
    }
}
```

Рисунок 8. Класс SignInController

### Веб-страница signIn\_page

На данной странице осуществляется поиск пользователя по фамилии.

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
        content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
<div>
    <form action="/signIn" method="post">
        <input name="lastName" placeholder="Last Name">
        <input type="submit" value="SignIn">
    </form>
</div>
</body>
</html>
```

Рисунок 9. Код страницы signIn\_page

### Класс UserDetailsServiceImpl

В классе осуществляется поиск пользователя по фамилии или выбрасывается исключение, что такого пользователя в базе данных не существует.

```

package com.example.course_work.security.details;

import ...

@Component(value = "customUserDetailsService")
public class UserDetailsServiceImpl implements UserDetailsService {

    @Autowired
    private UsersRepo usersRepo;

    @Override
    public UserDetails loadUserByUsername(String lastName){
        lastName = "%" + lastName + "%";
        User user = usersRepo.findByLastNameIgnoreCase(lastName);
        if (user != null){
            return new UserDetailsImpl(user);
        }
        else{
            throw new UsernameNotFoundException("User not found");
        }
    }
}

```

Рисунок 10. Класс UserDetailsServiceImpl

### Метод WriteToFile

В методе WriteToFile происходит запись внесённых изменений в файл. Ему на вход подаётся строка, составленная из (ID + firstName + lastName + profession), которая с переносом строки записывается в файл made.txt.

```

public static void WriteToFile(String data) {
    File file = new File( pathname: "C:\\Users\\Sofia\\IdeaProjects\\Course_Work\\made.txt");

    FileWriter file_writer = null;
    BufferedWriter buffered_writer = null;

    try{
        int noOfLines=1;
        file_writer = new FileWriter(file, append: true);
        buffered_writer = new BufferedWriter(file_writer);
        buffered_writer.append("ID, FirstName, LastName, Profession");

        for(int i = noOfLines; i > 0; i--)
        {
            buffered_writer.newLine(); //перенос строки
            buffered_writer.write(data); //запись строки в файл
        }
    }
    catch (IOException e) {
        e.printStackTrace();
    }
    finally{
        try {
            buffered_writer.close();
            file_writer.close();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
}

```

Рисунок 11. Метод WriteToFile

### Класс ReadFromFile

В классе реализуется метод UseScanner, который считывает данные из файла и записывает их в объект.



```

// открываем файл
String file_path = "C:\\Users\\Sofia\\IdeaProjects\\CourseWork\\test.txt";
BufferedReader reader = new BufferedReader(new FileReader(file_path));

// считываем построчно
String line = null;
Scanner scanner = null;
int index = 0;
List<User> empList = new ArrayList<>();

while ((line = reader.readLine()) != null) {
    User user = new User();

    scanner = new Scanner(line);
    scanner.useDelimiter(","); //указываем чем разделяются данные в файле

    while (scanner.hasNext()) {
        String data = scanner.next();
        switch (index){
            case 0:
                user.setFirstName(data);
                break;
            case 1:
                user.setLastName(data);
                break;
            case 2:
                user.setProfession(data);
                break;
            default:
                System.out.println("Некорректные данные::" + data);
                break;
        }
        index++;
    }
    index = 0;
    empList.add(user);
}

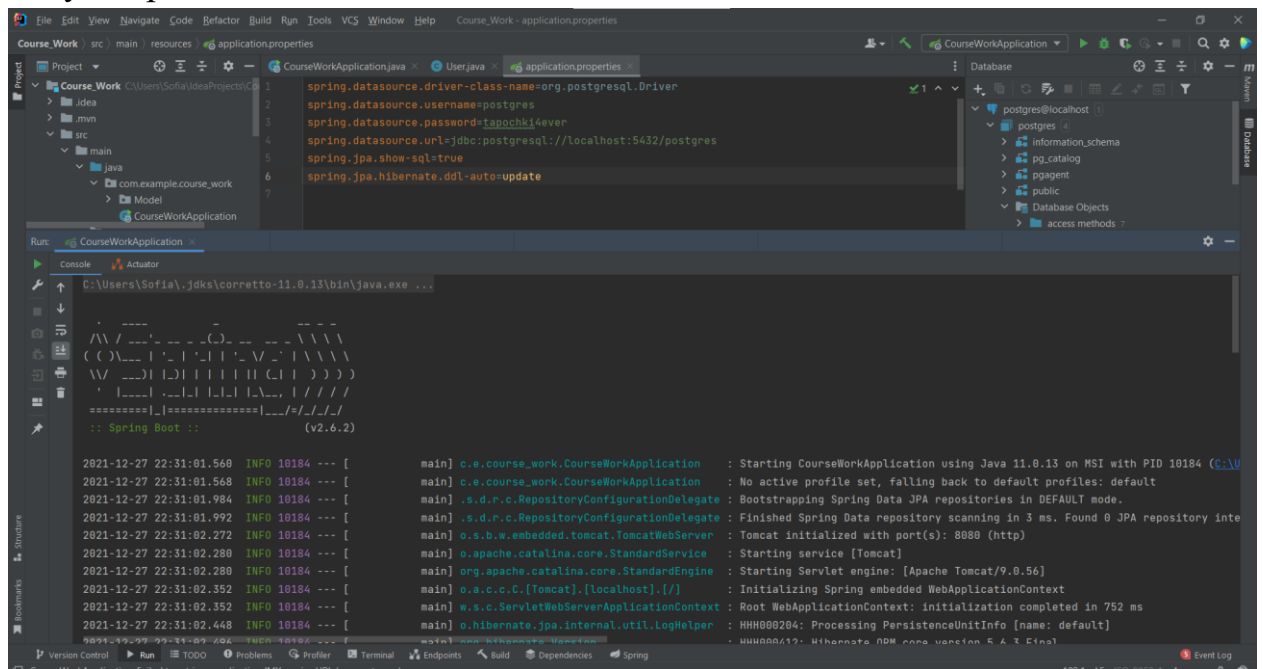
//закрываем ридер
reader.close();

```

Рисунок 12. Метод UseScanner

## 4.3 Тестирование приложения

Запуск приложения:

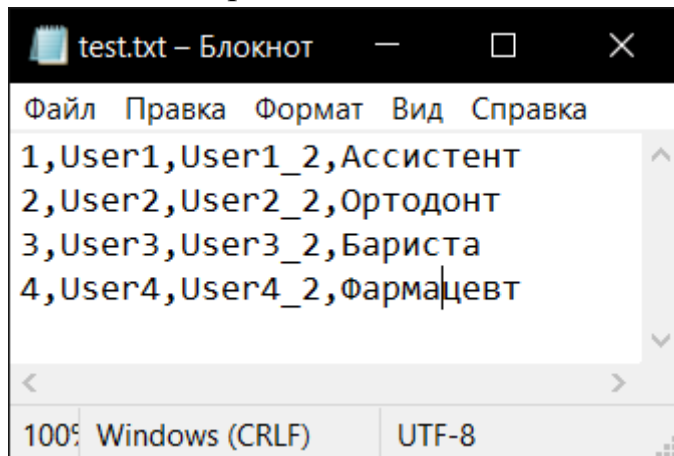


The screenshot shows an IDE with the following components:

- Project Explorer:** Shows the project structure with folders like `src`, `main`, `java`, and `com.example.course_work`.
- Code Editor:** Displays `application.properties` with the following configuration:

```
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.username=postgres
spring.datasource.password=postgres14ever
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```
- Database Explorer:** Shows a connection to `postgres@localhost` with a list of databases including `information_schema`, `pg_catalog`, `pgagent`, and `public`.
- Console:** Displays the startup logs of the application, including the Spring Boot banner and various initialization messages from Tomcat and Hibernate.

Файл, из которого идёт чтение данных:



The screenshot shows a Notepad window titled `test.txt – Блокнот`. The text content is as follows:

```
Файл  Правка  Формат  Вид  Справка
1,User1,User1_2,Ассистент
2,User2,User2_2,Ортодонт
3,User3,User3_2,Бариста
4,User4,User4_2,Фармацевт
```

At the bottom, the encoding is set to `UTF-8`.

Просмотр страницы с таблицей пользователей:

ID	First Name	Last Name	Profession
1	User1	User1_2	Ассистент
2	User2	User2_2	Ортодонт
3	User3	User3_2	Бариста
4	User4	User4_2	Фармацевт

Открываем форму для добавления информации в базу данных и заполняем:

Имя	Фамилия	Профессия	AddUser
-----	---------	-----------	---------

Видим, что выполнен ввод данных с формы на веб-странице:

```

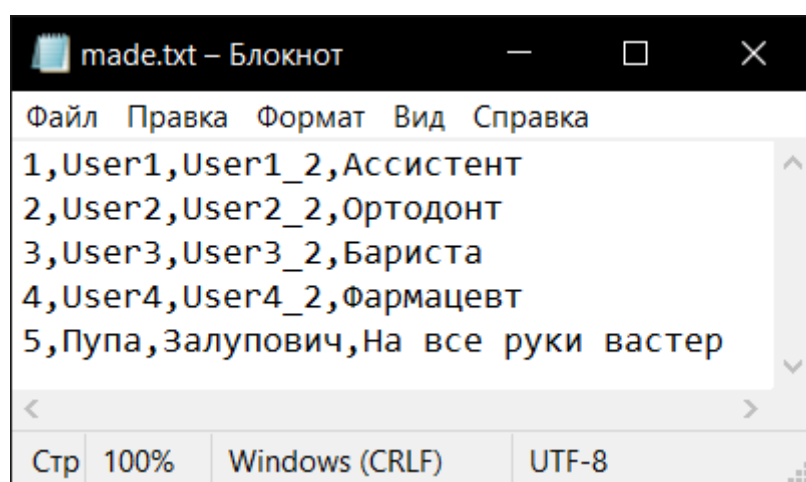
2021-12-28 00:42:33.803 INFO 7880 --- [nio-8090-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]
2021-12-28 00:42:33.803 INFO 7880 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet
2021-12-28 00:42:33.803 INFO 7880 --- [nio-8090-exec-1] o.s.web.servlet.DispatcherServlet
Hibernate: insert into account (first_name, last_name, profession) values (?, ?, ?)

```

Проверяем, что данные действительно занесли в базу данных (id = 5):

id	first_name	last_name	profession
1	Купер	Яколёв	Хороший мальчик
2	User2	User2_2	Ортодонт
3	User3	User3_2	Бариста
4	User4	User4_2	Фармацевт
5	Пуша	Залупович	На все руки мастер

Содержимое файла, в который записалась добавленная строка с информацией о новом пользователе:



## **5. Заключение**

В ходе выполнения курсовой работы были реализованы поставленные требования при помощи использования PostgreSQL и Java в среде разработки IntelliJ IDEA, что поспособствовало удобной разработки приложения для автономной работы с базой данных и клиентом через веб сервис. Разработанное приложение выполняет поставленные требования и результаты.

## **6. Список литературы**

1. Шилдт Г. Java. Руководство для начинающих, 2002
2. Шилдт Г. Java: полное руководство, 10 издание, 2018
3. Крейг Уоллс. Spring в действии, 2013
4. Джон Лонг, Кеннет Бастани. Java в облаке, 2019
5. Гэри Марк, Джош Лонг, Даниэль Рубио. Spring Recipes: Подход к решению проблем, 2010