МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Пятигорский институт (филиал) СКФУ

Колледж Пятигорского института (филиал) СКФУ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

МДК.03.02 Разработка кода информационных систем

Специальности СПО 09.02.11 Разработка и управление программным обеспечением

Пятигорск 2026

Методические указания для практических занятий по дисциплине МДК.03.02 Разработка кода информационных систем составлены в соответствии с требованиями ФГОС СПО к подготовке выпуска для получения квалификации. Предназначены для студентов, обучающихся по специальности 09.02.11 Разработка и управление программным обеспечением.

Практическая работа № 1

Тема : Разработка программы имитирующей деятельность библиотеки в соответствии с принципами объектно-ориентированного программирования

Цель: разработать полнофункциональное Java- приложение для автоматизации работы библиотеки, демонстрирующее глубокое понимание и практическое применение принципов ООП (инкапсуляция, наследование, полиморфизм, абстракция), а также навыков проектирования классов и взаимодействия между объектами.

Теоретическая часть:

Объектно-ориентированное программирование (ООП) представляет собой парадигму разработки программного обеспечения, основанную на концепции объектов, содержащих данные и методы для манипулирования этими данными. Основные принципы ООП включают:

1. Абстракция: процесс выделения существенных характеристик объекта и игнорирования несущественных деталей.Пример абстракции в библиотеке: книга характеризуется названием, автором и годом издания, тогда как подробности типографского процесса игнорируются.
2. Инкапсуляция: механизм сокрытия внутренней реализации класса и предоставления интерфейса для взаимодействия с объектом.Например, читатель не должен заботиться о внутренних механизмах хранения записей о выданных книгах, достаточно иметь доступ к методам выдачи и возврата книг.
3. Наследование: способность создавать новые классы путем расширения существующих, позволяя наследовать свойства и поведение родительских классов.К примеру, класс Book может стать родителем для подклассов вроде FictionBook, NonFictionBook.
4. Полиморфизм: свойство объектов реагировать по-разному на одни и те же методы в зависимости от типа объекта.Читатель может вернуть книгу, журнал или DVD-диск одним методом, но внутренняя логика обработки различна.

Задание:

Разработайте Java-приложение «Библиотека», реализующее следующие функ
ции:

Функциональные требования:

- добавление/удаление книг в каталог;
- регистрация/удаление читателей;
- выдача книг читателям;
- возврат книг;
- поиск книг по названию/автору;
- просмотр списка выданных книг;
- расчёт штрафов за просрочку.

Шаг 1. Создайте структуру классов

1. Класс Book:
    - поля: private String isbn, private String title, private String author, pri
      vate boolean isAvailable;
    - конструктор с параметрами isbn, title, author;
    - геттеры и сеттеры;
    - метод public String toString() для удобного вывода.
2. Абстрактный класс LibraryMember:
    - поля: private int id, private String name, private List<Loan> loans;
    - абстрактный метод public double calculateFine(int daysOverdue);
    - методы для управления списком loans.
3. Классы-наследники:
    - StudentMember - реализует calculateFine() (штраф 10 руб./день);
    - FacultyMember - реализует calculateFine() (штраф 5 руб./день).
4. Класс Loan:
    - поля: private Book book, private LibraryMember member, private Lo
      calDate dueDate;

- метод public boolean isOverdue() - проверяет просрочку.

5. Класс Library (Singleton):
    - поля: private static Library instance, private List<Book> books, priva te List<LibraryMember> members, private List<Loan> activeLoans;
    - приватный конструктор;
    - статический метод getInstance();
    - методы: addBook(), registerMember(), issueBook(), returnBook(), se archBooks(), listOverdueBooks().
6. Класс MemberFactory:
    - статический метод createMember(String type, int id, String name) - создаёт нужный тип читателя.

Шаг 2. Реализуйте обработку исключений

Создайте пользовательские исключения:

- BookNotAvailableException;
- MemberNotFoundException;
- OverdueException.

Используйте их в методах issueBook() и других при нарушении бизнес-логик и.

Шаг 3. Создайте консольный интерфейс

Класс LibraryApp с методом main():

- реализует простое меню:
    1. Добавить книгу.
    2. Зарегистрировать читателя.
    3. Выдать книгу.
    4. Вернуть книгу.
    5. Поиск книг.
    6. Список просроченных.

7. Выход.

- обрабатывает ввод пользователя через Scanner;
- вызывает соответствующие методы класса Library.

Шаг 4. Напишите тесты

Создайте класс LibraryTest с методами JUnit:

- тест добавления книги;
- тест выдачи книги;
- тест расчёта штрафа;
- тест поиска книг.

Требования к реализации:

- все поля классов - private;
- используйте LocalDate для работы с датами;
- реализуйте валидацию данных (не пустые названия, корректные ID);
- обеспечьте обработку исключений в пользовательском интерфейсе;
- код должен быть документирован (комментарии к классам и методам);
- приложение должно компилироваться и запускаться без ошибок.

Ход выполнения работы

Этап 1. Проектирование

1. Нарисуйте UML-диаграмму классов с полями, методами и связями.
2. Определите сигнатуры всех методов.
3. Продумайте сценарии использования (use cases).

Этап 2. Реализация базовых классов

1. Реализуйте классы Book, Loan, LibraryMember, StudentMember, Faculty Member.
2. Напишите конструкторы и геттеры/сеттеры.

3. Реализуйте абстрактный метод в наследниках.

Этап 3. Реализация главного класса

1. Реализуйте класс Library как Singleton.
2. Добавьте коллекции для хранения данных.
3. Напишите методы addBook(), registerMember().
4. Реализуйте issueBook() с проверкой доступности и наличием просрочек .
5. Реализуйте returnBook() с расчётом штрафа.

Этап 4. Создание интерфейса и фабрики

1. Напишите класс MemberFactory.
2. Реализуйте консольное меню в LibraryApp.
3. Добавьте обработку ввода через Scanner.

Этап 5. Обработка исключений

1. Создайте классы исключений.
2. Добавьте try-catch блоки в методах Library.
3. Обработайте исключения в main().

Этап 6. Тестирование

1. Напишите JUnit-тесты для ключевых сценариев.
2. Проверьте граничные случаи (попытка взять недоступную книгу и т. д.).
3. Запустите приложение и протестируйте все пункты меню.

Практическая работа № 2

Тема : 2. Расширение функционала проекта программы библиотеки добавлением коллекций и интерфейсов

Цель: Расширить ранее разработанное приложение библиотеки, реализовав поддержку коллекций и интерфейсов в Java для улучшения организации данных и повышения гибкости системы. Включить

дополнительные возможности по работе с книгами, читателями и операциями над ними, обеспечивая соблюдение принципов объектно-ориентированного проектирования.

Теоретическая часть:

Коллекции -
это структуры данных для хранения и управления наборами объектов.

Пакет java.util.Collections предоставляет иерархию интерфейсов и классов.

Основные интерфейсы коллекций:

- List -
  упорядоченная коллекция с дубликатами. Реализуется классами ArrayL ist,

  LinkedList. В библиотеке используется для хранения списков книг, читате лей, выдач.

- Set - неупорядоченная коллекция без дубликатов. Реализуется HashSet

  , TreeSet. Подходит для уникальных идентификаторов (ISBN), тегов кн иг.

- Map -
  коллекция пар «ключ-значение». Реализуется HashMap, TreeMap.

  Используется для быстрого поиска по ключу: книги по ISBN, читатели по ID.

Обобщения (Generics)

Позволяют создавать типобезопасные коллекции. Синтаксис: List<Book>, Ma p<String, Book>. Преимущества:

- проверка типов на этапе компиляции;
- отсутствие необходимости явного приведения типов;
- повышение читаемости кода.

3. Интерфейсы и их роль

Интерфейс - контракт, определяющий набор методов без реализации.

Применение в проекте:

- абстрагирование поведения (например, Searchable, Borrowable);
- полиморфизм: разные классы реализуют один интерфейс;
- разделение ответственности: логика поиска отделена от хранения данн ых.

Итерация по коллекциям

- классический цикл for;
- улучшенный цикл for-each;
- итератор (Iterator<E>);
- Stream API (Java 8+) для фильтрации и преобразования.

Паттерны проектирования

- Iterator -
  предоставляет унифицированный доступ к элементам коллекции.
- Strategy -
  инкапсулирует алгоритмы поиска (по названию, автору, году).
- Observer -
  уведомляет подписчиков о событиях (книга возвращена, штраф начисл ен).

Принципы SOLID

- Single Responsibility -
  каждый класс выполняет одну задачу (хранение книг, поиск, выдача).
- Open/Closed -
  расширение функционала через новые классы, а не изменение

  существующих.
- Liskov Substitution -
  подклассы должны корректно заменять родительские классы.

Задание:

Расширьте функционал ранее созданного приложения «Библиотека», добавив коллекции и интерфейсы.

Функциональные требования:

- хранение книг и читателей в коллекциях разных типов;
- реализация интерфейса Searchable для поиска книг по различным крите риям;
- использование Map для быстрого доступа к книгам по ISBN и читателя м по ID;
- добавление функционала тегов (жанров) для книг через Set;
- поддержка нескольких стратегий поиска через интерфейс SearchStrateg у;
- вывод статистики библиотеки (количество книг, читателей, выданных э кземпляров).

Шаг 1. Создайте интерфейсы

1. Интерфейс Searchable<T>:
    - метод List<T> search(String query) - общий поиск;
    - метод List<T> advancedSearch(Map<String, String> criteria) - расширенный поиск.
2. Интерфейс Borrowable:
    - методы boolean isAvailable(), void borrow(), void returnItem().
3. Интерфейс SearchStrategy:
    - метод List<Book> find(List<Book> books, String query) - алгоритм поиска.

Шаг 2. Расширьте классы коллекций

В классе Library:

- замените List<Book> на Map<String, Book> для хранения книг по ISBN (ключ);
- добавьте Map<Integer, LibraryMember> для быстрого доступа к читател ям по ID;
- добавьте поле Set<String> genres для хранения уникальных жанров биб лиотеки;

- в классе Book добавьте поле Set<String> tags для тегов книги (жанры, к лючевые слова).

Шаг 3. Реализуйте стратегии поиска

Создайте классы-стратегии:

- TitleSearchStrategy - ищет книги по названию;
- AuthorSearchStrategy - ищет по автору;
- GenreSearchStrategy - ищет по тегу (жанру).

Каждый класс реализует SearchStrategy и содержит логику фильтрации.

Шаг 4. Интегрируйте поиск в Library

Добавьте в Library методы:

- void setSearchStrategy(SearchStrategy strategy) - установка стратегии;
- List<Book> searchBooks(String query) -
  выполнение поиска по текущей стратегии;
- Map<String, List<Book>> groupByGenre() -
  группировка книг по жанрам (возвращает Map<жанр, список книг>).

Шаг 5. Добавьте статистику

Реализуйте в Library метод LibraryStats getStats(), возвращающий объект со ст атистикой:

- общее количество книг;
- количество доступных книг;
- количество читателей;
- количество выданных книг;
- топ-3 самых популярных жанров.

Класс LibraryStats должен содержать соответствующие поля и метод toString( ).

Шаг 6. Обновите консольный интерфейс

В LibraryApp добавьте пункты меню:

- поиск книг (с выбором стратегии: по названию, автору, жанру);
- просмотр статистики библиотеки;
- добавление тегов к книге;
- группировка книг по жанрам.

Практическая работа № 3

Тема : привязка тестовой базы данных на основе MySQL к java приложению

Цель: Изучить основы работы с базой данных MySQL из Java-приложения, освоив основные этапы подключения, выполнения запросов и обработки результатов выборки. Научиться разрабатывать простейшие CRUD-операции (Create, Read, Update, Delete) с использованием драйверов JDBC и стандартной библиотеки Java для работы с SQL-запросами.

Теоретическая часть:

При взаимодействии Java-приложений с реляционными базами данных используется технология JDBC (Java Database Connectivity). Она обеспечивает стандартный API для связи с различными СУБД, такими как MySQL, PostgreSQL, Oracle и др., стандартизируя подход к выполнению SQL-запросов и обработке полученных данных.

JDBC - это фреймворк, включенный в стандартную поставку Java SE. Его задача заключается в предоставлении универсального набора инструментов для взаимодействия с любыми реляционными базами данных независимо от производителя БД. Чтобы задействовать JDBC, необходимы два компонента:

1. Драйвер базы данных. Драйвер - это программа-посредник, позволяющая установить соединение с сервером базы данных и передавать запросы.
2. API-интерфейс, позволяющий отправлять SQL-команды и получать результаты.

Этап 1: Установка драйвера MySQL Connector/J

Перед началом работы необходимо скачать и подключить соответствующий драйвер для MySQL. Обычно он доступен на официальном сайте MySQL.

После загрузки подключаете jar-файл к своему проекту либо используете менеджер зависимостей Maven/Gradle для автоматической установки:

Для Gradle:

```
dependencies {
    implementation 'mysql:mysql-connector-java:8.0.x'
}
```

Для Maven:

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.x</version>
</dependency>
```

Где 8.0.x - версия используемого драйвера.

Этап 2: Получение соединения с базой данных

Чтобы начать взаимодействие с базой данных, сначала необходимо создать подключение. Рассмотрим пример:

```
// Импортируем необходимые пакеты
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnectionExample {
    public static Connection connect(String url, String user, String password) throws SQLException {
```

```
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
        return DriverManager.getConnection(url, user, password);
    }

    public static void main(String[] args) {
        String dbUrl = "jdbc:mysql://localhost:3306/your_database";
        String username = "your_username";
        String password = "your_password";

        try(Connection conn = connect(dbUrl, username, password)) {
            System.out.println("Подключение установлено!");
        } catch(SQLException ex) {
            System.err.println(ex.getMessage());
        }
    }
}
```

Здесь важно обратить внимание на строку подключения (URL): "jdbc:mysql://hostname:port/database_name". Порт обычно установлен на значение 3306, если иное не указано в настройках сервера.

Этап 3: Выполнение SQL-запросов

Запросы SELECT

Запрос SELECT предназначен для извлечения данных из таблицы. Простой пример:

```
String sqlQuery = "SELECT * FROM users WHERE age > ?";
try (PreparedStatement stmt = connection.prepareStatement(sqlQuery)) {
    stmt.setInt(1, 18); // Устанавливаем аргумент в позицию ?
    ResultSet resultSet = stmt.executeQuery(); // Выполняем запрос
    while(resultSet.next()) {
        System.out.println(resultSet.getString("username"));
    }
}
```

Запрос INSERT добавляет записи в таблицу:

```
String insertSql = "INSERT INTO users(username, email) VALUES(?,?)";
try (PreparedStatement pstmt = connection.prepareStatement(insertSql)) {
    pstmt.setString(1, "JohnDoe"); // Первый аргумент (?)
    pstmt.setString(2, "john@example.com"); // Второй аргумент (?)
    pstmt.executeUpdate(); // Выполнить вставку
}
```

UPDATE-запросы обновляют данные в таблице:

```
String updateSql = "UPDATE users SET email=? WHERE username=?";
try (PreparedStatement pstmt = connection.prepareStatement(updateSql)) {
    pstmt.setString(1, "new_email@example.com");
    pstmt.setString(2, "JohnDoe");
    pstmt.executeUpdate();
```

```
}
```

DELETE-запросы удаляют строки из таблицы:

```
String deleteSql = "DELETE FROM users WHERE username=?";
try (PreparedStatement pstmt = connection.prepareStatement(deleteSql)) {
    pstmt.setString(1, "JohnDoe");
    pstmt.executeUpdate();
}
```

Задание:

Вашей задачей станет разработка простого Java-приложения, выполняющего CRUD-операции с базой данных MySQL. Приложение должно позволить вам:

1. Подключиться к заранее подготовленной базе данных.
2. Извлекать и выводить данные о пользователях.
3. Добавлять новых пользователей.
4. Редактировать данные о существующих пользователях.
5. Удалять ненужных пользователей.

Структура таблиц должна выглядеть примерно так:

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50),
    email VARCHAR(100),
    age INT
);
```

Практическая работа № 4

Тема : Разработка приложения на основе технологий Java и JDBC

Цель: Освоить технологии Java и JDBC для построения клиент-серверного приложения, обеспечивающего хранение и обработку данных с использованием реляционных баз данных. Основной упор сделать на изучение механизмов работы с базами данных средствами JDBC и разработку простых бизнес-процессов на стороне клиента.

Теоретическая часть:

Приложения, построенные на платформе Java, часто используют базы данных для хранения и обработки данных. Одним из стандартных способов взаимодействия с реляционными базами данных является использование Java Database Connectivity (JDBC) - стандартного API, предоставляемого платформой Java.

Типичное клиент-серверное приложение состоит из двух частей:

- Клиентская сторона - пользовательский интерфейс, отправляющий запросы серверу.
- Серверная сторона - база данных и слой бизнес-логики, обрабатывающий запросы и возвращающий результат клиенту.

В данном приложении мы рассмотрим простую архитектуру, где клиент обращается непосредственно к базе данных через JDBC-драйвер, минуя промежуточный сервер приложений.

JDBC - это низкоуровневый API, предназначенный для взаимодействия Java-кода с реляционными базами данных. Он предоставляет общие интерфейсы для отправки SQL-запросов и обработки результатов. Основными компонентами JDBC являются:

Driver Manager - система, управляющая соединениями с базой данных.
Connection - объект, представляющий физическое соединение с базой данных.
Statement - объект для выполнения SQL-запросов.
ResultSet - объект, содержащий результат выполнения запроса.
Подготовка базы данных
Предположим, у вас есть таблица employees в базе данных MySQL:

```
CREATE DATABASE employee_db;
USE employee  db;

CREATE TABLE employees(
   emp_id INT AUTO_INCREMENT PRIMARY KEY,
   first_name VARCHAR(50),
   last_name VARCHAR(50),
   salary DECIMAL(10,2)
);
```

Задача состоит в том, чтобы создать простое Java-приложение, осуществляющее CRUD-операции (Create, Read, Update, Delete) с этой таблицей.

Задание:

Разработайте Java-приложение, которое сможет выполнять следующие операции с базой данных:

1. Добавление нового сотрудника.
2. Просмотр списка сотрудников.
3. Изменение зарплаты конкретного сотрудника.
4. Удаление сотрудника по его ID.

Подробное описание этапов выполнения задания:

1. Настройка рабочего пространства

Установите необходимую версию Java Development Kit (JDK) и интегрированную среду разработки (IDE), такую как IntelliJ IDEA или Eclipse.

Также скачайте и установите драйвер MySQL Connector/J для работы с MySQL через JDBC. Если используете Maven, укажите зависимость:

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.x</version>
</dependency>
```

2. Соединение с базой данных

Реализуйте метод для открытия соединения с базой данных:

```
import java.sql.*;

public class DBConnector {
    private static final String URL = "jdbc:mysql://localhost:3306/employee_db";
```

```
    private static final String USERNAME = "root";
    private static final String PASSWORD = "";

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL, USERNAME, PASSWORD);
    }
}
```

3. Реализация операций CRUD

a) Добавление нового сотрудника

Напишите метод, создающий запись в таблице employees:

```
public static void createEmployee(int empId, String firstName, String lastName, double salary) throws SQLException {
    try (Connection con = DBConnector.getConnection();
         PreparedStatement ps = con.prepareStatement("INSERT INTO employees(emp_id, first_name, last_name, salary) VALUES(?,?,?,?)")) {
        ps.setInt(1, empId);
        ps.setString(2, firstName);
        ps.setString(3, lastName);
        ps.setDouble(4, salary);
        ps.executeUpdate();
    }
}
```

b) Просмотр списка сотрудников

Получите список всех сотрудников из базы данных:

```
public static void readEmployees() throws SQLException {
    try (Connection con = DBConnector.getConnection();
         Statement st = con.createStatement();
         ResultSet rs = st.executeQuery("SELECT * FROM employees")) {
        while(rs.next()) {
            System.out.printf("%d %s %s %.2f\n",
                    rs.getInt("emp_id"),
                    rs.getString("first_name"),
                    rs.getString("last_name"),
                    rs.getDouble("salary"));
        }
    }
}
```

c) Изменение зарплаты сотрудника

Измените зарплату сотрудника по его ID:

```
public static void updateSalary(int empId, double newSalary) throws
SQLException {
    try (Connection con = DBConnector.getConnection();
         PreparedStatement ps = con.prepareStatement("UPDATE employees SET
salary=? WHERE emp_id=?")) {
        ps.setDouble(1, newSalary);
        ps.setInt(2, empId);
        ps.executeUpdate();
    }
}
```

d) Удаление сотрудника

Удалите сотрудника по его ID:

```
public static void deleteEmployee(int empId) throws SQLException {
    try (Connection con = DBConnector.getConnection();
         PreparedStatement ps = con.prepareStatement("DELETE FROM employees WHERE emp_id=?")) {
        ps.setInt(1, empId);
        ps.executeUpdate();
    }
}
```

4. Сборка основного класса приложения

Соберите весь функционал вместе, написав основной класс приложения:

```
import java.sql.SQLException;

public class MainApp {
    public static void main(String[] args) {
        try {
            // Создать нового сотрудника
            createEmployee(1, "Игорь", "Смирнов", 50000.00);
            createEmployee(2, "Анна", "Петрова", 60000.00);

            // Показать всех сотрудников
            System.out.println("\nСотрудники:");
            readEmployees();
```

```
            // Обновить зарплату сотруднику
            updateSalary(1, 55000.00);
            // Удалить сотрудника
            deleteEmployee(2);
            // Повторно показать сотрудников
            System.out.println("\nСотрудники после изменений:");
            readEmployees();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Основная литература:

1. Грекул В.И., Коровкина Н.Л., Куприянов Ю.В. Проектирование информационных систем. М.: Национальный открытый университет «ИНТУИТ», 2026, Режим доступа: http://www.iprbookshop.ru/67376.html

2. Вичугова А.А. Инструментальные средства разработки компьютерных систем и комплексов [Электронный ресурс] : учебное пособие для СПО / А.А. Вичугова. - Электрон. текстовые данные. - Саратов: Профобразование, 2026. - 135 c. - 978-5-4488-0015-3. - Режим доступа: http://www.iprbookshop.ru/66387.html

3. Долженко А.И. Технологии командной разработки программного обеспечения информационных систем [Электронный ресурс]/ Долженко А.И.- Электрон. текстовые данные.- М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2026.- 300 c.- Режим доступа: http://www.iprbookshop.ru/39569.html.- ЭБС «IPRbooks»

Дополнительная литература:

1. Рудаков А. Технология разработки программных продуктов: учебник. Изд.Academia. Среднее профессиональное образование. 2023 г. 208 стр.