**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики» Рефакторинг баз данных и приложений**

Лабораторная работа №3

Выполнил: Студент группы

БПИ2301

Быков Максим   
Эдуардович

Москва

2025

# Цель:

1. Ознакомиться с основными принципами и задачами рефакторинга.
2. Научиться выявлять проблемные участки кода (code smells) и устранять их.
3. Применить техники рефакторинга для улучшения читаемости, структуры и производительности кода.
4. Развить навыки анализа и улучшения существующего кода.

# Задачи:

1. Ознакомьтесь с теоретическими основами рефакторинга, включая его цели, преимущества и отличия от оптимизации. Изучите основные техники рефакторинга, такие как: разделение больших функций на более мелкие, устранение дублирующегося кода, улучшение именования переменных, функций и классов, введение уровней абстракции.
2. Проведите обратное проектирование выбранного кода. Для этого: постройте диаграмму классов или модулей, чтобы визуализировать структуру программы, определите основные зависимости между компонентами, найдите участки кода, которые сложно читать, поддерживать или тестировать.
3. Примените следующие техники рефакторинга: разделение больших функций на более мелкие, устранение дублирующегося кода, улучшение именования переменных, функций и классов, введение уровней абстракции для упрощения сложных участков кода, убедитесь, что после каждого изменения программа сохраняет своё поведение.
4. Опишите все изменения, которые вы внесли в код, и обоснуйте их необходимость. Подготовьте отчет, включающий исходное состояние кода (с примерами проблемных участков), внесенные изменения (с пояснениями), итоговое состояние кода.
5. Проверьте, что после рефакторинга программа работает корректно.

# Ход работы

# Изучение и анализ исходного проекта

# Сначала был проанализирован исходный проект, содержащий несколько Java-классов без чёткой структуры и с отсутствующими или ошибочными методами. Основные проблемы заключались в:

# несоответствии сигнатур конструкторов и используемых параметров;

# отсутствии необходимых методов (getBooks(), searchBooksByTitle() и др.);

# попытках тестировать классы, которые были не готовы к тестированию;

# отсутствии поддержки Maven и JUnit в проекте;

# нарушении соглашений Java (например, классы в неправильных файлах, пакет org.example.library не соответствовал структуре каталогов и т. д.).

# 2. Исправление структуры проекта

# Были выполнены следующие структурные изменения:

# Созданы отдельные классы Book, LibraryManager, LibraryApp в соответствии с их назначением.

# Удалены или перенесены неправильно именованные классы и тестовые файлы.

# Приведены в порядок пакеты и пути файлов: src/main/java/org/example/library/ для основного кода и src/test/java/org/example/library/ для тестов.

# 3. Реализация недостающего функционала

# Были реализованы отсутствующие методы:

# В классе Book — конструктор и геттеры.

# В LibraryManager — методы addBook(), getBooks(), searchBooksByTitle().

# В LibraryApp — добавлены конструкторы и метод listBooks().

# 4. Настройка Maven

# Добавлен pom.xml с конфигурацией для:

# JDK 17;

# зависимостей JUnit Jupiter (версии 5.7.0);

# плагина maven-surefire-plugin для запуска тестов.

# 5. Разработка тестов

# Созданы модульные тесты:

# BookTest — проверка корректности создания объекта Book.

# LibraryManagerTest — тесты добавления и поиска книг.

# LibraryAppTest — тестирование вывода списка книг через LibraryApp.

# Анализ исходного кода

Исходный код представлял собой неструктурированное приложение для управления библиотекой с несколькими основными классами:

* Book — класс, представляющий книгу с несколькими атрибутами.
* LibraryManager — класс, отвечающий за управление коллекцией книг, включая добавление и поиск по книгам.
* LibraryApp — класс, который выполняет роль интерфейса для взаимодействия с пользователем через командную строку.

**Проблемы исходного кода:**

* Конструктор Book: Конструктор класса Book требовал большего количества параметров, чем было нужно, и его использование в тестах было неправильным.
* LibraryApp: Этот класс пытался использовать зависимость от LibraryManager, но конструктор не был правильно настроен, что делало приложение некорректным.
* Отсутствие тестов: В коде не было тестов, или они не соответствовали текущей логике приложения.

# Выполненные изменения

### 1. Класс Book

**Проблема**: Конструктор Book требовал дополнительных параметров, которых не было в тестах.

*Book book = new Book("Title", "Author", 2020);*

Исправлен конструктор Book для корректного приема только необходимых данных — названия книги, автора и года издания.

*public class Book {*

*private String title;*

*private String author;*

*private int year;*

*public Book(String title, String author, int year) {*

*this.title = title;*

*this.author = author;*

*this.year = year;*

*}*

*public String getTitle() {*

*return title;*

*}*

*public String getAuthor() {*

*return author;*

*}*

*public int getYear() {*

*return year;*

*}*

*}*

2. Класс LibraryApp

**Проблема**: Класс LibraryApp не имел конструктора по умолчанию, а также не использовал корректно LibraryManager.

*LibraryApp app = new LibraryApp(new LibraryManager());*

Добавлен конструктор по умолчанию и конструктор, принимающий LibraryManager. Также добавлен метод listBooks(), который теперь возвращает список книг.

*public class LibraryApp {*

*private LibraryManager libraryManager;*

*public LibraryApp(LibraryManager libraryManager) {*

*this.libraryManager = libraryManager;*

*}*

*public LibraryApp() {*

*this.libraryManager = new LibraryManager();*

*}*

*public List<String> listBooks() {*

*List<String> bookList = new ArrayList<>();*

*for (Book book : libraryManager.getBooks()) {*

*bookList.add(book.getTitle());*

*}*

*return bookList;*

*}*

*}*

**3. Класс LibraryManager**

Проблема: Некоторые методы, такие как getBooks(), отсутствовали, а также не было методов для поиска книг.

*List<Book> books = libraryManager.getBooks();*

Добавлены методы для добавления книг, получения списка книг и поиска книг по названию.

*public class LibraryManager {*

*private List<Book> books;*

*public LibraryManager() {*

*books = new ArrayList<>();*

*}*

*public void addBook(Book book) {*

*books.add(book);*

*}*

*public List<Book> getBooks() {*

*return books;*

*}*

*public List<Book> searchBooksByTitle(String title) {*

*List<Book> foundBooks = new ArrayList<>();*

*for (Book book : books) {*

*if (book.getTitle().contains(title)) {*

*foundBooks.add(book);*

*}*

*}*

*return foundBooks;*

*}*

*}*

**4. Тесты**

Проблема: Отсутствие тестов для основного кода.

Добавлены тесты для классов Book, LibraryManager и LibraryApp. Для тестирования использовался JUnit 5.

**Пример теста для класса Book**:

*import org.junit.jupiter.api.Test;*

*import static org.junit.jupiter.api.Assertions.\*;*

*public class BookTest {*

*@Test*

*public void testBookCreation() {*

*Book book = new Book("1984", "George Orwell", 1949);*

*assertEquals("1984", book.getTitle());*

*assertEquals("George Orwell", book.getAuthor());*

*assertEquals(1949, book.getYear());*

*}*

*}*

**Пример теста для класса LibraryManager**:

*import org.junit.jupiter.api.Test;*

*import static org.junit.jupiter.api.Assertions.\*;*

*public class LibraryManagerTest {*

*@Test*

*public void testAddBook() {*

*LibraryManager libraryManager = new LibraryManager();*

*Book book = new Book("1984", "George Orwell", 1949);*

*libraryManager.addBook(book);*

*assertTrue(libraryManager.getBooks().contains(book));*

*}*

*@Test*

*public void testSearchBooksByTitle() {*

*LibraryManager libraryManager = new LibraryManager();*

*Book book1 = new Book("1984", "George Orwell", 1949);*

*Book book2 = new Book("Brave New World", "Aldous Huxley", 1932);*

*libraryManager.addBook(book1);*

*libraryManager.addBook(book2);*

*assertTrue(libraryManager.searchBooksByTitle("1984").contains(book1));*

*assertTrue(libraryManager.searchBooksByTitle("Brave New World").contains(book2));*

*}*

*}*

**Пример теста для класса LibraryApp**:

*import org.junit.jupiter.api.Test;*

*import static org.junit.jupiter.api.Assertions.\*;*

*public class LibraryAppTest {*

*@Test*

*public void testListBooks() {*

*LibraryManager libraryManager = new LibraryManager();*

*LibraryApp app = new LibraryApp(libraryManager);*

*libraryManager.addBook(new Book("1984", "George Orwell", 1949));*

*libraryManager.addBook(new Book("Brave New World", "Aldous Huxley", 1932));*

*assertTrue(app.listBooks().contains("1984"));*

*assertTrue(app.listBooks().contains("Brave New World"));*

*}*

}

**5. Maven-конфигурация**

Конфигурация pom.xml была настроена для использования JUnit и для запуска тестов через Maven.

*xml*

*<project xmlns="http://maven.apache.org/POM/4.0.0"*

*xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"*

*xsi:schemaLocation="http://maven.apache.org/POM/4.0.0*

*http://maven.apache.org/xsd/maven-4.0.0.xsd">*

*<modelVersion>4.0.0</modelVersion>*

*<groupId>org.example</groupId>*

*<artifactId>librarymanager3000</artifactId>*

*<version>1.0-SNAPSHOT</version>*

*<name>LibraryManager3000</name>*

*<properties>*

*<maven.compiler.source>17</maven.compiler.source>*

*<maven.compiler.target>17</maven.compiler.target>*

*</properties>*

*<dependencies>*

*<dependency>*

*<groupId>org.junit.jupiter</groupId>*

*<artifactId>junit-jupiter-api</artifactId>*

*<version>5.7.0</version>*

*<scope>test</scope>*

*</dependency>*

*<dependency>*

*<groupId>org.junit.jupiter</groupId>*

*<artifactId>junit-jupiter-engine</artifactId>*

*<version>5.7.0</version>*

*<scope>test</scope>*

*</dependency>*

*</dependencies>*

*<build>*

*<plugins>*

*<plugin>*

*<groupId>org.apache.maven.plugins</groupId>*

*<artifactId>maven-surefire-plugin</artifactId>*

*<version>2.22.2</version>*

*</plugin>*

*</plugins>*

*</build>*

*</project>*

# ССЫЛКА НА РЕПОЗИТОРИЙ

https://github.com/A1e0nd/refactoring3

# ВЫВОД

В результате рефакторинга:

1. Исходный код был структурирован, улучшена читаемость и понимание программы.
2. Добавлены необходимые конструкторы и методы, что сделало код более гибким и функциональным.
3. Созданы тесты, которые теперь могут использоваться для автоматического тестирования функциональности приложения.
4. Настроена конфигурация Maven для сборки и запуска тестов.

В целом, проект стал более организованным, структурированным и готовым к дальнейшему развитию.