МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Информационные технологии»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

ДЛЯ ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ РАБОТ ПО ДИСЦИПЛИНЕ «СОВРЕМЕННЫЕ ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ», ч.1

Ростов-на-Дону
ДГТУ
2022

УДК 372.8:004
Составители: М. В. Привалов

Методические указания для выполнения лабораторных работ по дисциплине «Современные технологии программирования», ч.1. - Ростов-на-Дону: Донской гос. техн. ун-т, 2022. - 34с.

Рассматриваются современные подходы и технологии разработки программного обеспечения для Web-ориентированных и распределённых информационных систем.

Предназначены для студентов направления 09.04.02 «Информационные технологии» всех форм обучения.

УДК 372.8:004

Печатается по решению редакционно-издательского совета
Донского государственного технического университета

Ответственный за выпуск зав. кафедрой «Информационные технологии»,
д-р техн. наук, профессор Б.В. Соболь

В печать ___.___. 20___г.
Формат 60×84/16. Объем___усл.п.л.
Тираж ___экз. Заказ №___.

Издательский центр ДГТУ
Адрес университета и полиграфического предприятия:
344000, г. Ростов-на-Дону, пл. Гагарина,1

Оглавление

Лабораторная работа №1. "Основы ООП в Java. Ввод/вывод, сериализация, наследование"

Цель работы: изучить структуру программы и объектную модель Java. Научиться корректно использовать ввод/вывод и сериализацию.

Порядок выполнения лабораторной работы №1

Разработать программу, которая получает на вход CSV файл с информацией об объектах согласно варианту. Программа считывает эти данные и помещает их в массив. Затем массив сериализуется в бинарный файл, после чего десериализуется из этого файла в новый.

При выполнении лабораторной работы требуется воспользоваться интерфейсами и абстрактными классами. Интерфейс должен содержать типовые операции, необходимые для обработки информации по алгоритму, заданному в задании. Второй интерфейс является базовым для сущностей индивидуального задания. При наличии общих операций рекомендуется создать дополнительный уровень иерархии с применением абстрактного класса.

Требования к результирующей программе.

- Входной файл задается в качестве параметра при запуске программы.
- Необходимо проанализировать вариант и использовать наследование, отражающее иерархию классов.
- Каждый класс должен содержать 3 - 5 свойств.
- Программа должна корректно обрабатывать ошибочные данные во входном файле!

- Программа должна вести отдельный текстовый файл журнала с информацией о создаваемых объектах, ошибках во входном файле, результатах сериализации и десериализации.

Варианты заданий

1) студент, преподаватель;
2) телевизор, монитор;
3) авторучка, карандаш;
4) катер, водный мотоцикл;
5) диван, кровать;
6) самолет, корабль;
7) штатный сотрудник, совместитель;
8) эллипс, многоугольник;
9) арифметическая и геометрическая прогрессия;
10) книга, журнал;
11) зачет, экзамен;
12) город, село;
13) млекопитающее, птица;
14) детская коляска, автокресло;
15) цифровая и обычная фоторамка;
16) кондиционер, обогреватель;
17) квадрат, прямоугольник;
18) стол, стул;
19) ноутбук, планшет;
20) HDD, SDD;
21) клавиатура, мышь;
22) беговая дорожка, степпер;

23) мобильный телефон, смартфон.

Лабораторная работа №2. "Основы работы с Java Graphics2D API, многопоточность"

Цель работы: получить навыки по созданию многопоточных приложений с графическим интерфейсом пользователя. Изучить возможности пакетов java.awt.geom, java.awt, классов java.awt.Graphics и java.awt.Graphics2D. Освоить основные графические примитивы и закрепить работу с потоками.

Порядок выполнения лабораторной работы №2

Разработать приложение, отображающее не менее двух «пассивных» движущихся объектов. Каждый объект должен управляется своим потоком, что должно быть визуально заметно (разная скорость движения). Количество этих объектов может быть произвольным. Кроме этих объектов должен быть один «активный» объект, управление которым осуществляется с помощью клавиатуры, или мыши (реализовать оба варианта). При столкновении объектов они должны менять направление движения. Вид фигуры определяется вариантом.

Варианты заданий:

1. Три концентрические окружности различных цветов и радиусов, движутся в различных направлениях и меняют свои цвета. Внутренняя окружность меняет заливку.

2. Два квадрата, расположенные один внутри другого, движутся по периметру экрана и меняют цвет линий при повороте. Внутренний квадрат меняет заливку.

3. Треугольник, вписанный в квадрат, движется по диагонали экрана и изменяет свои размеры и цвет при достижении края экрана.

4. Два пересекающихся эллипса, изменяют цвета заливки, положение одного относительно другого и общее местоположение.

5. Окружность, вписанная в квадрат, движется из стороны в сторону, изменяя свои размеры и цвета при достижении края экрана.

6. Пять, касающиеся друг друга окружностей, движутся по периметру экрана и меняют свои цвета при повороте, причем у каждой окружности изменяется свой цвет заливки.

7. Окружность, вписанная в квадрат, движется по окружности с центром в середине панели и изменяет свои цвет заливки и размеры при достижении верхней точки траектории.

8. Круг, вписанный в эллипс, движется по окружности с центром в середине панели и изменяет свой цвет заливки и размеры при достижении правой крайней точки траектории.

9. Окружности движутся по разным прямым траекториям и меняют свои цвета при достижении края экрана.

10. Ромб, вписанный в прямоугольник, попеременно изменяет свой цвет, местоположение и размеры.

11.Окружность, вписанная в ромб, изменяют свои цвета и размеры. Центр фигуры постоянно находится в центре панели.

12.Набор прямоугольников разных цветов, отбрасывающих тень, движутся по экрану вправо, а при столкновении с «активным» объектом – меняют направление на противоположное.

13.Набор пересекающихся квадратов разных цветов и размеров движутся по форме вниз и на каждом следующем шаге изменяют цвета и заливку. При

столкновении с «активным» объектом – меняют направление на противоположное.

14. Набор непересекающихся эллипсов разных цветов и размеров движутся по экрану вверх и на каждом следующем шаге изменяют цвета и заливку. При столкновении с «активным» объектом – меняют направление на противоположное.

15. Набор кругов разных цветов и размеров движутся по экрану вверх и на каждом следующем шаге изменяют цвета и заливку. При столкновении с «активным» объектом – меняют направление на противоположное.

16. Набор прямоугольников и треугольников различных цветов изменяют свои цвета и положение друг относительно друга.

17. В наборе кругов и прямоугольников различных цветов, круги движутся по своим траекториям, а прямоугольники стоят на месте и на каждом следующем шаге все изменяют цвета и заливку.

18. В наборе кругов и окружностей различных размеров и цветов, круги движутся и на каждом следующем шаге меняют цвета, а окружности остаются постоянными.

19. В наборе кругов и треугольников различных цветов круги движутся влево, а треугольники вправо и на каждом следующем шаге меняют цвета. При столкновении с «активным» объектом круги меняют направление на противоположное.

20. Окружности различных размеров и цветов движутся по экрану, постепенно увеличивая радиус, и изменяют цвет на каждом новом шаге.

21. Произвольная решетка из прямых линий разных цветов движется по панели по периметру, изменяя расцветку на каждом повороте.

22. Компактный набор окружностей одного цвета хаотически движется по экрану, изменяя размеры и общий цвет при достижении края экрана.

23. Компактный набор квадратов одного цвета движется по экрану по синусоидальной траектории, изменяя общий цвет при достижении края экрана.

24. Компактный набор кругов одного цвета и квадратов другого цвета произвольно движутся по экрану, изменяя размеры только окружностей и общие цвета.

25. Компактный набор концентрических окружностей различных цветов движется по экрану, изменяя постепенно размеры и изменяет цвет фона при достижении края экрана.

26. Компактный набор вписанных друг в друга квадратов различных цветов движется по экрану, изменяя размеры и общий цвет.

27. Компактный набор квадратов различных цветов движется по экрану, принимая то собственный цвет, то цвет поля, сохраняя собственным цвет периметра.

28. Набор вписанных друг в друга квадратов, имеющих различное положение диагоналей и различные цвета, движется по экрану, изменяя размеры и положения относительно друг друга.

Лабораторная работа №3. "Интерфейс работы с базами данных в Java "

Цель работы: получить навыки по работе с реляционными базами данных в Java с применением стандартного API JDBC. Ознакомиться со способами управления зависимостями и сборки Java-проектов.

Основные теоретические сведения.

Java Database Connectivity (JDBC) – это стандартный API для независимого соединения языка программирования Java с различными базами данных (далее – БД).

JDBC решает следующие задачи:

1. Создание соединения с БД.
2. Создание SQL выражений.
3. Выполнение SQL – запросов.
4. Просмотр и модификация полученных записей.

Если говорить о реализации, то JDBC – это библиотека, которая обеспечивает целый набор интерфейсов для доступа к различным БД.

Для доступа к каждой конкретной БД необходим специальный JDBC – драйвер, который является адаптером Java – приложения к БД.

Строение JDBC.

JDBC поддерживает как 2-звенную, так и 3-звенную модель работы с БД, но в общем виде, JDBC состоит из двух слоёв.

- JDBC API – обеспечивает соединение “приложение – JDBC Manager”.
- JDBC Driver API – обеспечивает соединение “JDBC Manager – драйвер”.

JDBC API использует менеджер драйверов и специальные драйверы БД для обеспечения подключения к различным базам данных.

JBDC Manager проверяет соответствие драйвера и конкретной БД. Он поддерживает возможность использования нескольких драйверов одновременно для одновременной работы с несколькими видами БД.

Схематично, JDBC можно представить в таком виде (рис. 3.1):

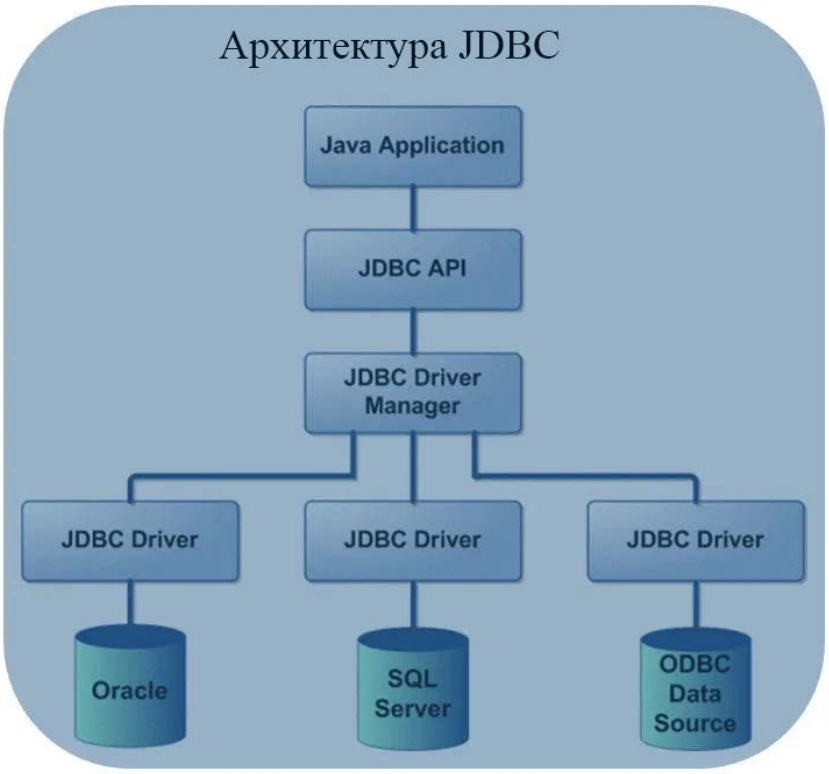

Рис. 3.1. – Архитектура JDBC

JDBC API состоит из следующих элементов:

Менеджер драйверов (Driver Manager). Этот элемент управляет списком драйверов БД. Каждой запрос на соединение требует соответствующего драйвера. Первое совпадение даёт нам соединение.

Драйвер (Driver). Этот элемент отвечает за связь с БД. Работать с ним приходится крайне редко. Вместо этого чаще используются объекты DriverManager, которые управляют объектами этого типа.

Соединение (Connection). Этот интерфейс обеспечивает нас методами для работы с БД. Все взаимодействия с БД происходят исключительно через Connection.

Выражение (Statement). Для подтверждения SQL-запросов используются объекты, созданные с использованием этого интерфейса.

Результат (ResultSet). Экземпляры этого элемента содержат данные, которые были получены в результате выполнения SQL – запроса. Он работает как итератор и “пробегает” по полученным данным.

Исключения (SQL Exception). Этот класс обрабатывает все ошибки, которые могут возникнуть при работе с БД.

Настройка, сборка проектов и управление зависимостями.

Все современные IDE позволяют управлять зависимостями проектов, собрать, запускать и даже развёртывать приложения. Однако, для универсальности, как правило, используют сторонние средства управления зависимостями и сборки проектов. Для Java приложений это Ant, Maven, Gradle.

Apache Maven — каркас для автоматизации управления зависимостями и сборки проектов, специфицированных на XML-языке POM(Project Object Model).

На примере бесплатно распространяемой популярной IDE Eclipse, создание проекта выполняется через меню File->New->Maven project. На первом экране настройки проекта требуется отметить Create simple project, чтобы не выполняль настройку архетипа проекта. После этого потребуется заполнить базовые поля Maven-проекта:

Group Id: наименование отдела или организации (правила аналогичны именованию пакетов)

Artifact Id: название проекта

Далее будет создан новый проект, в котором присутствует файл pom.xml, отвечающий за настройку разрешения зависимостей и сборку. Его содержимое

может быть примерно таким (в зависимости от заполнения указанных выше полей):

```
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.dstu</groupId>
  <artifactId>dbdemo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</project>
```

Структура проекта при этом будет иметь вид, показанный на рис. 8.2:

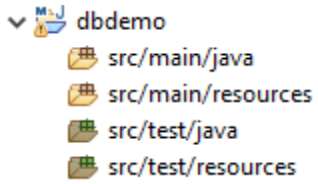

Рис. 3.2. – Структура нового Maven-проекта

Нас будет интересовать исключительно папка src/main/java, в которой будут находиться все исходные тексты нашего приложения.

Для подключения библиотек требуется создать элемент dependencies и каждую библиотеку вписать как отдельный элемент dependency.

Например, в случае Postgresql драйвер JDBC для доступа к данной СУБД может быть добавлен в конфигурацию следующим образом:

```
  <dependencies>
     <dependency>
          <groupId>org.postgresql</groupId>
          <artifactId>postgresql</artifactId>
          <version>9.4.1211</version>
     </dependency>
  </dependencies>
```

Для сборки проекта Maven используется команда mvn compile.

После сборки (из IDE или командной строки mvn compile) мы получим в случае успеха следующие сообщения:

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< org.dstu:dbdemo >-------------------------
[INFO] Building dbdemo 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ dbdemo ---
[WARNING] Using platform encoding (Cp1251 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ dbdemo ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding Cp1251, i.e. build is platform dependent!
[INFO] Compiling 1 source file to C:\Users\max\eclipse-workspace\dbdemo\target\classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.245 s
[INFO] Finished at: 2019-12-22T17:13:08+03:00
[INFO] ------------------------------------------------------------------------
```

Чтобы не получать предупреждений о платформенно-зависимой сборке, желательно явно указать кодировку исходных файлов и текстовых ресурсов:

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
```

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< org.dstu:dbdemo >-------------------------
[INFO] Building dbdemo 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ dbdemo ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ dbdemo ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\Users\max\eclipse-workspace\dbdemo\target\classes
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.290 s
[INFO] Finished at: 2019-12-22T17:14:44+03:00
[INFO] ------------------------------------------------------------------------
```

Порядок выполнения лабораторной работы №3

Разработать программу, которая получает на вход CSV файл с информацией об объектах согласно варианту (из работы №1). Программа считывает эти данные и помещает их в БД. Реализовать 2 запроса к БД.

К программе имеются следующие требования:

- Проект должен быть собран с применением Maven.
- Входной файл задается в качестве параметра при запуске программы.
- Каждый объект должен содержать 3 - 5 свойств.
- Необходимо использовать наследование.
- Реализовать запросы на добавление нескольких записей с использованием JDBC.
- Реализовать запрос на вывод всех объектов из БД и связанных с ними сущностей.
- Реализовать подготавливаемый запрос с условием, параметры которого заполняются в зависимости от пользовательского ввода.
- Реализовать изменение одного объекта и удаление другого внутри транзакции с уровнем изоляции Dirty Read.

Для выполнения работы необходимо:

1. Создать базу данных, в соответствии с описанием предметной области в задании.
2. Настроить проект, подключив к нему соответствующий драйвер, поставляемый в виде JAR-библиотеки (рекомендуется для этого и предыдущего случаев использовать СУБД MySQL или PostgreSQL).
3. Реализовать разбор командной строки и чтение исходного файла.
4. Реализовать подключение к БД и обработку ошибок.
5. Для выполнения требований реализовать создание всех требуемых SQL Statement, их вызов и обработку.
6. Обеспечить корректное освобождение всех задействованных ресурсов.
7. Продемонстрировать запуск проекта как с применением Maven, так и без.

Варианты заданий

1) студент, преподаватель;
2) телевизор, монитор;
3) авторучка, карандаш;
4) катер, водный мотоцикл;
5) диван, кровать;
6) самолет, корабль;
7) штатный сотрудник, совместитель;
8) эллипс, многоугольник;
9) арифметическая и геометрическая прогрессия;
10) книга, журнал;
11) зачет, экзамен;
12) город, село;
13) млекопитающее, птица;
14) детская коляска, автокресло;
15) цифровая и обычная фоторамка;
16) кондиционер, обогреватель;
17) квадрат, прямоугольник;
18) стол, стул;
19) ноутбук, планшет;
20) HDD, SDD;
21) клавиатура, мышь;
22) беговая дорожка, степпер;
23) мобильный телефон, смартфон.

Лабораторная работа №4. "Java Persistence API, ORM, Hibernate"

Цель работы: получить навыки по работе с реляционными базами данных в Java. Научиться использовать отображение объектов на реляционную модель с использованием современных ORM-каркасов.

Основные теоретические сведения.

Hibernate — это популярный ORM-каркас, цель которого – связать ООП и реляционную базу данных. Работа с Hibernate сокращает время разработки проекта в сравнении с обычным JDBC, так как создаёт связь между таблицами в базе данных (далее – БД) и Java-классами и наоборот. Схематично это можно отразить так (рис. 4.1):

Рис. 4.1. –Hibernate и реляционная СУБД в Java-приложении

Преимущества использования Hibernate:

- Обеспечивает простой API для записи и получения Java-объектов в/из БД.
- Минимизирует доступ к БД, используя стратегии fetching.
- Не требует сервера приложения.
- Позволяет нам не работать с типами данных языка SQL, а иметь дело с привычными нам типами данных Java (POJO – Plain Old Java Objects).
- Заботится о создании связей между Java-классами и таблицами БД с помощью XML-файлов или аннотаций, не внося изменения в программный

код. Если нам необходимо изменить БД, то достаточно лишь внести изменения в XML-файлы.

Hibernate поддерживает все популярные СУБД: MySQL, Oracle, PostgreSQL, Microsoft SQL Server Database, HSQL, DB2 и может работать в связке с такими технологиями, как Maven и J2EE, а также NoSQL СУБД.

Приложение, использующее Hibernate, очень упрощённо имеет следующую архитектуру (рис. 4.2.):

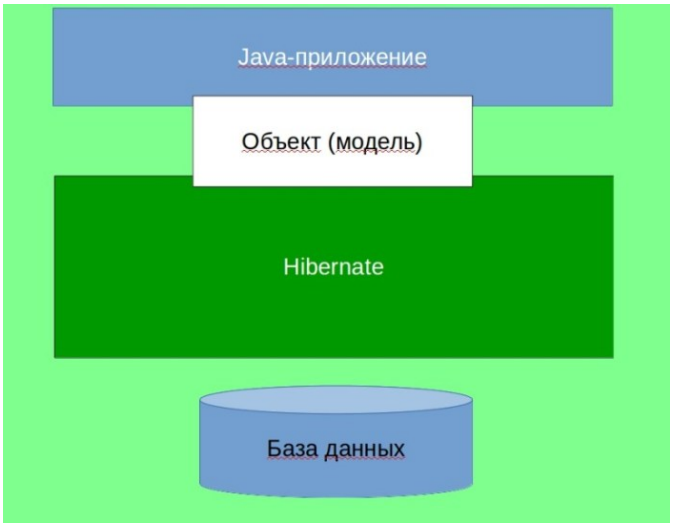

Рис. 4.2. – Упрощённая архитектура Java-приложения, использующего Hibernate

Если мы рассмотрим строение самого Hibernate более подробно, что этот же рисунок будет уже выглядеть следующим образом (рис. 4.3):

Рис. 4.3. – Детальная архитектура приложения с Hibernate

Hibernate поддерживает такие API, как JDBC, JNDI, JTA.
JDBC обеспечивает простейший уровень абстракции функциональности для реляционных БД. JTA и JNDI, в свою очередь, позволяют Hibernate использовать серверы приложений J2EE.

Transaction. Представляет собой рабочую единицу, изолирующую порцию работы с БД. В Hibernate транзакции обрабатываются менеджером транзакций.

SessionFactory. Самый важный и самый сложный и функционально насыщенный объект, который представляет собой фабрику. Обычно фабрика создаётся в единственном экземпляре, при запуске приложения. Как минимум одна SessionFactory требуется для каждой БД, при этом каждая из них конфигурируется отдельным конфигурационным файлом.

Session. Сессия используется для получения физического соединения с БД. Обычно сессия создаётся при необходимости, а после этого закрывается. Это возможно, потому что эти объекты крайне легковесны. В целом, можно сказать,

что создание, чтение, изменение и удаление объектов происходит через объект Session.

Query. Этот объект использует HQL или SQL для чтения/записи данных из/в БД. Экземпляр запроса используется для связывания параметров запроса, ограничения количества результатов, которые будут возвращены, и для выполнения запроса.

Configuration. Этот объект используется для создания объекта SessionFactory и конфигурирует сам Hibernate с помощью конифигурационного XML-файла, который объясняет, как обрабатывать объект Session.

Criteria. Используется для создания и выполнения объектно-ориентированных запросов для получения объектов. Критерии часто используются при реализации поиска информации в БД на основании пользовательских критериев.

Необходимые шаги по подключению Hibernate к проекту.

Для работы с Hibernate понадобится, как и в предыдущей работе, подключить необходимые библиотеки и выполнить некоторые настройки. Более детально:

1. Подключить в виде JAR-библиотеки драйвер соответствующей СУБД.
2. Подключить JAR-библиотеку с самим Hibernate (рекомендуется версия 5.x).
3. Выполнить конфигурацию ORM-каркаса в hibernate.cfg.xml.
4. Создать XML-конфигурации сущностей или аннотированные классы.

Следует отметить, что первые два пункта требуется выполнять, используя системы и технологии управления зависимостями приложения (Maven, Gradle и им подобные).

Порядок выполнения лабораторной работы №4

По аналогии с предыдущей работой создайте программу, которая получает на вход CSV файл с информацией об объектах согласно варианту. Программа считывает эти данные и помещает их в БД. При этом вся работа с БД должна вестись посредством объектно-реляционного отображения.

Требования к программе.

- Конфигурация зависимостей должна быть выполнена посредством Maven.
- Входной файл задается в качестве параметра при запуске программы.
- Каждый объект должен содержать 3 - 5 свойств.
- Отражение таблиц на объектную модель должно быть задано с применением аннотаций.
- Реализуйте DAO для доступа к соответствующим сущностям.
- Реализуйте и продемонстрируйте работу следующих запросов:
- запроса на вставку новых данных (внесите в БД данные из CSV файла);
- запроса на вывод всех объектов из БД;
- запроса на изменение одного и нескольких объектов в БД;
- запроса на выборку данных по условию (требуется задать условие, задействующее все таблицы, а сам запрос – в виде HQL).

Варианты заданий

1) студент, преподаватель, группа;
2) телевизор, монитор, категория товаров;
3) авторучка, карандаш, отдел канцтоваров;
4) катер, водный мотоцикл, причал;
5) диван, кровать, комната;
6) самолет, корабль, (аэро)порт;
7) штатный сотрудник, совместитель, кафедра;

8) эллипс, многоугольник, презентация;

9) арифметическая и геометрическая прогрессия;

10) книга, журнал, полка;

11) зачет, экзамен, предмет;

12) город, село, регион;

13) млекопитающее, птица, зоопарк;

14) детская коляска, автокресло, гараж;

15) цифровая и обычная фоторамка, фото;

16) кондиционер, обогреватель;

17) квадрат, прямоугольник, рисунок;

18) стол, стул, офис;

19) ноутбук, планшет, установленная операционная система;

20) HDD, SDD, сервер;

21) клавиатура, мышь, компьютер;

22) беговая дорожка, степпер, зал;

23) мобильный телефон, смартфон, приложение.

Лабораторная работа №5. "Конфигурирование Web-приложений на Java"

Цель работы: получить навыки по созданию и конфигурированию простого Web-приложения, использующего технологию JSP.

Основные теоретические сведения.

С точки зрения конфигурации проекта, Web-приложение будет отличаться от обычного тем, что имеет конфигурацию для развёртывания, имеет отличающуюся сборку, например, артефактом будет являться WAR-архив Web-приложения, и должно развёртываться с учётом инфраструктуры имеющегося

Web-сервера. В качестве Web-сервера, который может выступать в качестве контейнера Java-приложения может выступать Tomcat или Jetty, а также любой сервер приложений (Glassfish, Wildfly, JBoss и им подобные), так как они уже имеют в своём составе Web-контейнер.

Для создания приложения удобно сгенерировать его с использованием архетипов Maven, после чего импортировать проект и выполнять разработку уже в IDE.

Таким образом, создание и проверка базового Web-приложения с помощью Maven включает в себя следующие шаги: установку самостоятельной версии Maven и настройку окружения, генерацию и импорт проекта, модификацию, сборку, развёртывание и тестирование.

Ход работы.

1. Установите и настройте Maven.

После скачивания с https://maven.apache.org/ и установки Maven, чтобы начать работу со сборщиком приложения, необходимо провести настройку среды. Для этого нужно установить переменную окружения M2_HOME. В свойствах окружения необходимо добавить переменную «M2_HOME», значение которой – это путь к распакованному Maven.

Установите переменную окружения PATH. Для этого к путям PATH требуется добавить к списку директорий строку «%M2_HOME%\bin». Проверьте установку переменной окружения JAVA_HOME, которая должна указывать путь к установленному JDK (по аналогии с предыдущими пунктами).

Проверьте правильность сделанных настроек, запустив в командной строке *mvn --version*

2. Создайте Web-приложение.

Воспользуйтесь командной строкой и сгенерируйте приложение с помощью Maven. Для этого используйте команду:

mvn archetype:generate -DgroupId={project-packaging}
-DartifactId={project-name} –DarchetypeGroupId=org.apache.maven.archetypes
-DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false

В качестве groupId и artifactId задайте пакет и имя приложения, которые придумайте самостоятельно.

Ознакомьтесь с содержимым сгенерированного проекта: изучите структуру каталогов и содержимое сгенерированных файлов. Создайте каталог src/main/java, после чего выполните тестовую сборку проекта:

mvn clean install

Если всё сделано правильно, Maven скачает необходимые зависимости, выполнит очистку проекта и его сборку, а сам процесс будет завершён выводом строки «[INFO] BUILD SUCCESS», а также времени, затраченного на построение.

3. Изучите результаты построения проекта

Просмотрите содержимое директории target/ в проекте. Изучите созданные директории и файлы. Найдите Web-архив, подготовленный к развёртыванию и изучите его содержимое (файл с расширением .war).

Изучите содержимое локального репозитория Maven и найдите артефакты, созданные в результате построения проекта. Архив, предназначенный для развёртывания должен располагаться по следующему пути:

<домашняя директория>/.m2/repository/ <groupIdPath>/<artifactId>/<version>/ <artifactId>-<version>.war.

4. Подготовьте проект и откройте его в IDE

Проект Maven, сгенерированный данным образом, годится для разработки, автоматизированного построения и тестирования, но любую Java-разработку удобно делать в популярных IDE. В этом случае требуется подготовить проект к использованию в IDE, импортировать его и начинать разработку. В случае IDEA от JetBrains специальная подготовка не требуется и проект можно импортировать, используя дерево каталогов Maven: среда разработки «из коробки» способна корректно импортировать Maven-проекты. При использовании Eclipse требуется выполнить

mvn eclipse:clean eclipse:eclipse

Затем импортировать проект в рабочее пространство IDE как существующий сторонний проект.

5. Разработка и наполнение проекта

Наш проект готов к тому, чтобы менять его содержимое, тестировать, запускать и развёртывать внутри контейнера сервлетов. В качестве последнего подойдут следующие популярные и вполне легковесные контейнеры: Tomcat и Jetty. Для получения возможности запустить наше приложение с помощью Maven в контейнере необходимо включить в секцию <build> нашего проекта плагин, который позволит это сделать. Для Tomcat:

```
<plugin>
     <groupId>org.apache.tomcat.maven</groupId>
     <artifactId>tomcat7-maven-plugin</artifactId>
     <version>2.2</version>
     <configuration>
          <port>9090</port>
     </configuration>
</plugin>
```

После этого достаточно дать команду:

mvn tomcat7:run

Ваше приложение будет запущено в контейнере Tomcat и будет доступно по ссылке http://localhost:9090/имя

В случае успеха в браузере вы должны увидеть примерно следующее:

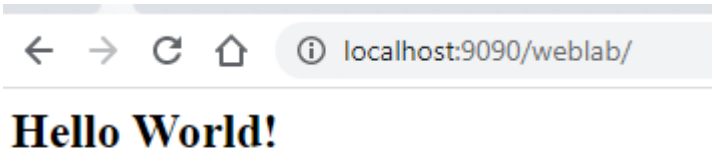

Рис. 5.1. Результат отображения страницы

Аналогичным образом можно использовать для просмотра страниц сервер Jetty.

Вопросы для самопроверки

1. Чем в Maven зависимости отличаются от плагинов?

2. Каковы основные этапы жизненного цикла приложения при использовании сборки Maven?

3. В чем отличие этапа clean от запуска плагина eclipse:clean?

4. Какие бывают репозитории Maven (maven repository)?

5. Для чего нужен плагин archetype:generate? Как его использовать?

6. Разобрать в настройке дополнительных параметров плагина maven Tomcat и плагина Jetty. Как применяются дополнительные параметры и на что они влияют?

Лабораторная работа №6. "Создание Web-приложения на Java, использующего технологию Java Server Pages"

Цель работы: получить навыки по разработке Web-приложения, использующего технологию Java EE – Java Server Pages, изучить жизненный цикл и способы применения сервлетов.

Основные теоретические сведения.

Для разработки используя технологий сервлетов и Java Serve Pages (JSP) нужно научиться понимать сценарии использования данных технологий.

При работе с Web-приложением через браузер сервер получает запросы от пользователя или клиента и передает их своим модулям на выполнение (возможно, другим серверам), производит обработку запросов через них путём обращения к данным, а также генерирует результаты, пригодные для отображения в браузере.

Сервлет – это диспетчер процесса, который находится на веб-сервере, и по поручению Web-контейнера обрабатывает различные входящие запросы и выдаёт исходящие ответы на них.

Сервлет можно использовать напрямую для записи в поток, который добавляет содержимое к веб-странице, но при этом, как правило, происходит смешение логики представления и бизнес-логики.

Большинство Java-сервлетов предназначены для ответов на различные HTTP-запросы в рамках текущего Web-приложения. Следовательно, каждому разработчику системы приходится работать с HTTP-классами из пакетов javax.servlet и javax.servlet.http. Для создания Java-сервлета создается подкласс

класса HttpServlet. В базовом классе имеются методы, вызываемые при обработке запросов разного типа. Чаще всего (как и запросы) используются методы:

- doGet – для обработки запросов GET;
- doPost – для обработки запросов POST.

Есть и другие (doPut, doDelete).

Пример простого сервлета, обрабатывающего запросы GET и выдающего в качестве ответа фрагмент HTML-кода:

```
package org.dstu.servlets;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloWorldServlet extends HttpServlet {
     private static final long serialVersionUID = 1L;

     private ServletConfig config;

     public void init(ServletConfig config) throws ServletException {
          this.config = config;
     }

     public void destroy() {
     }

     public ServletConfig getServletConfig() {
          return config;
     }

     public String getServletInfo() {
          return "A Simple Servlet";
     }

     @Override
     protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
          resp.setContentType("text/html");
          PrintWriter out = resp.getWriter();
          out.println("<html><head>");
          out.println("<title>A Sample Servlet!</title>");
```

```
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Hello, World!</h1>");
            out.println("</body></html>");
            out.close();
        }

        @Override
        protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
            // TODO Auto-generated method stub
            super.doPost(req, resp);
        }
}
```

Жизненный цикл сервлета.

1. Если сервлет отсутствует в контейнере, тогда класс сервлета загружается самим контейнером. Контейнер, используя класс сервлета, создает его экземпляр. Создание происходит вызовом метода init(). Этот метод инициализирует сам сервлет и всегда вызывается первым, до того, как сервлет начинает обслуживать запросы пользователей. За весь цикл жизни сервлета метод init() вызывается только один раз.

2. Далее происходит обслуживание клиентских запросов. Запросы обрабатываются в своих отдельных потоках. Сервер вызывает метод service() для каждого запроса из запросов.

3. В случае если серверу необходимо удалить сервлет, он вызывает метод destroy(), который снимает сервлет из эксплуатации.

Технология Java Server Pages

Данная технология интегрирует статическое содержимое, например, созданное с применением HTML и CSS, и динамическое содержимое, обеспечиваемое логикой, сосредоточенной в классах Java.

Чтобы встроить в страницы динамическое содержимое, применяются дополнительные элементы – JSP-теги, которые обеспечивают различные виды наполнения. Пример JSP-страницы приведен ниже.

```
<html>
    <head>
        <title>Очень простая страница JSP</title>
    </head>
    <body>
        <h1>Очень простая страница JSP</h1>
        <h3> Этот пример показывает, как работать с выражениями JSP</h3>
        Текущая дата: <%= new java.util.Date()%>
        <br>
        Значение переданного параметра "param": <%=
request.getParameter("param")%>
    </body>
</html>
```

Создания и компиляции сервлетов мало для обеспечения работы страницы. Помимо этого, требуется описать сервлеты в дескрипторе развёртывания Web-приложения. Этот файл находится в каталоге WEB-INF/ приложения и всегда называется web.xml. В нём описываются контекстные пути, классы, ресурсы, способы взаимодействия с кодом.

Ход работы

1. Создайте пакет для размещения исходного кода Java

2. Создайте и реализуйте сервлеты в соответствии с вариантом задания. Студенты, чей номер в списке нечётный, выбирают вариант 1, остальные – вариант 2.

Вариант 1.

Создайте сервлет, который:

- принимает 5 входящих параметров: имя, фамилию, отчество студента, название дисциплины и полученный балл;
- создает новый файл по фамилии, имени и отчеству студента в любом заданном каталоге на жестком диске и записывает все переданные параметры в каждую строку в файле.

В качестве результата необходимо выдать html-страницу с указанием созданного файла и его текущим содержимым. Если в файле с именем в качестве ФИО уже существует запись о добавляемом предмете и оценка отличается от неудовлетворительной, требуется отобразить html-страницу с ошибкой.

Вариант 2.

Создать сервлет, который:

- в качестве параметра принимает имя файла с изображением (считается, что все изображения находятся в одном каталоге, имя которого заранее известно);
- необходимо считать картинку с диска в память, после чего выдать html-страницу с ней и её именем в качестве ответа на запрос (сервлет должен выдать само изображение, а не обеспечивать прямой доступ к файлу по URL).

Если картинка с переданными именем файла не была найдена, требуется отобразить html-страницу с ошибкой.

Необходимо создать jsp-страницу, содержащую html-форму для отправки данных и параметров сервлетам в соответствии с заданным вариантом задания.

Вопросы для самопроверки

1. Что такое сервлет?
2. Что необходимо для запуска сервлетов?
3. Перечислите этапы жизненного цикла сервлетов.
4. Что требуется для того, чтобы сервлету можно было передать параметры формы?

Лабораторная работа №7. «Создание простого Web-приложения, использующего паттерн MVC на основе Spring Framework»

Цель работы: получить навыки по разработке Web-приложения с применением Spring Framework, ознакомиться с принципами работы Spring и шаблонизатора Thymeleaf.

Основные теоретические сведения.

Spring Framework – это каркас для разработки приложений, использующих распределённую Internet-архитектуру. Как известно, существует несколько вариантов реализации этой архитектуры, в нашем же случае мы будем опираться на применение шаблона проектирования Model-View-Controller (MVC).

Данный каркас интегрирует в себе большое количество технологий и стандартов платформы Java. Наиболее важные из них:

1. Библиотеки J2EE для разработки Web-приложений.
2. Поддержка стандарта Java Persistence API.
3. Поддержка REST.
4. Интегрированные Web-контейнеры.
5. Планировщик.

Чтобы упростить доступ к этим технологиям и сократить время конфигурирования можно использовать Spring Boot.

Для создания Web-приложения с применением шаблона проектирования MVC требуется шаблонизатор. Задачей шаблонизатора является интеграция HTML-шаблона и динамических данных (как правило, специальных элементов,

которые транслируются в вызовы сервлетов). Таким образом, шаблон и сервлеты обеспечивают работу интерактивного приложения.

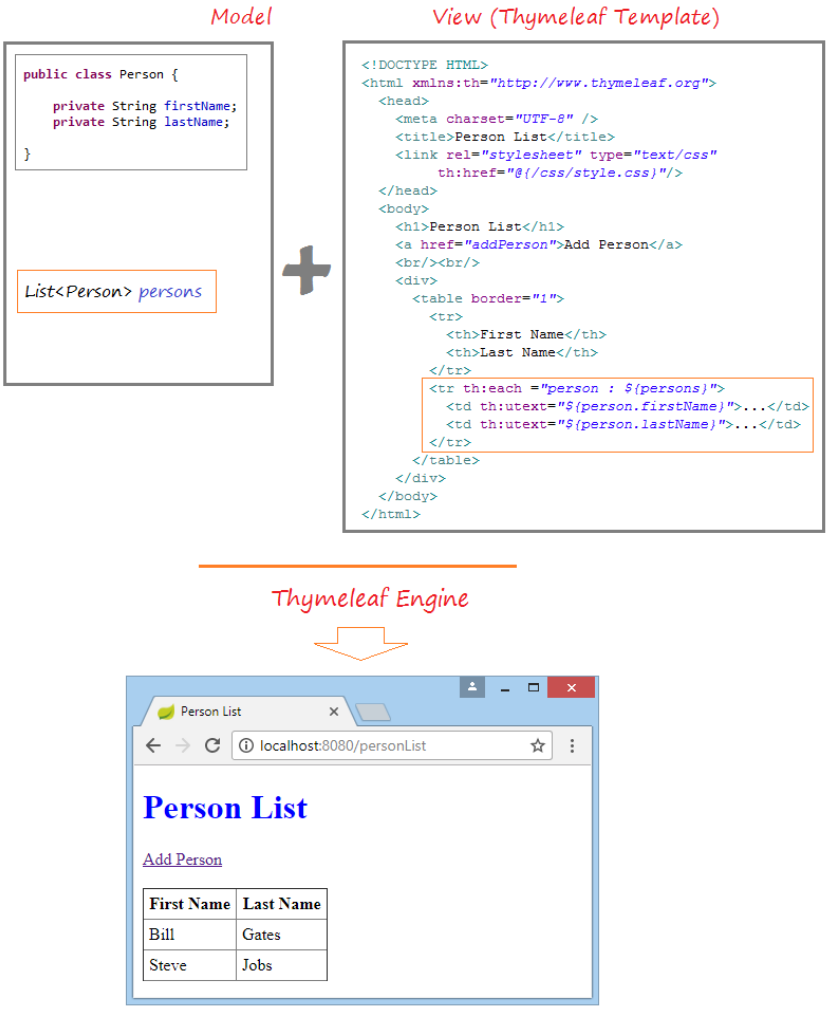

Рис. 7.1. Генерация страницы с помощью шаблонизатора Thymeleaf

Для того, чтобы создать новое приложение Spring Boot с установленными зависимостями и при этом сократить количество ручной работы, можно воспользоваться онлайн-инициализатором Spring. Он располагается по адресу https://start.spring.io/ Для настройки проекта надо выбрвать:

- систему сборки (Maven или Gradle);
- язык;

- версию Spring Boot;
- зависимости.

Ход работы

1. Создайте с помощью онлайн-конфигуратора приложение Spring Boot, использующее шаблонизатор Thymeleaf и сборщик Maven.

2. Импортируйте приложение в вашу IDE и настройте JPA для соединения с базой данных (требуется любая SQL РСУБД, рекомендуется MySQL или PostgreSQL).

3. Согласно варианту задания работы №9 создайте сущности, используя аннотации JPA.

4. Создайте шаблоны страниц с использованием Thymeleaf, которые будут давать возможность выполнять следующие действия:

- добавлять сущность;

- добавлять связанную сущность;

- выводить список сущностей;

- выводить список связанных сущностей (доступно по щелчку на главной сущности в списке);

5. Создайте Java-классы, представляющие формы из п.4

6. Создайте контроллер, обеспечивающий обработку форм и отображение списков. Рекомендуется создать два контроллера: один для главной сущности, второй – для связанной.

Вопросы для самопроверки

1. Для чего используется Spring Framework?

2. Для чего нужны шаблонизаторы?

3. Как шаблонизатор обеспечивает интерактивную работу приложения?

4. Какие существуют альтернативные варианты работы с шаблонизатором?