Лабораторный практикум по дисциплине "Архитектура информационных систем"

Вместо предисловия

К сожалению, идея вести историю изменений пришла спустя несколько лет проведения данного практикума, который проходит в рамках данной дисциплины уже много лет. И только сейчас она появилась здесь. Тем не менее данный практикум постоянно совершенствуется, следуя передовым тенденциям в области Java.

История изменений

Версия	Изменения
V.4 30.08.2021	1. Добавлена лр — Настройка переменных окружения 2. Добавлена лр — Ведение истории изменения объектов

Лабораторный практикум по дисциплине "Архитектура информационных систем" представляет собой набор практических работ, выполняемых в текущем семестре, который должен стать основой законченного приложения с тестами и документацией и покрывать следующие этапы:

1. Проектирование. Результатом этого этапа является набор канонических UML диаграмм.
2. Разработка. Результат — java приложение.
3. Тестирование. Результат — наборы модульных и интеграционных тестов.

Полученная библиотека, будет являться основой для выполнения лабораторного практикума в следующем семестре, путём реализации основных CRUD методов с помощью ORM библиотеки Hibernate.

Основные технологии и инструменты необходимые для выполнения лабораторных работ:

1. ОС — Linux, Windows, в силу того что в основном современные корпоративные системы функционирую с среде Linux, данная система является предпочтительнее. Использование Linux это во первых надежность, простота обслуживания, снижение стоимости проекта.
2. Java v 11 и выше — как основная платформа разработки
3. Apache NetBeans v11 и выше — как инструментальное средство разработки (IDE) (https://netbeans.apache.org)
4. Библиотека для организации взаимодействия Java объектов с CSV — источниками

данных, например opencsv (http://opencsv.sourceforge.net/)

5. Библиотека для организации взаимодействия Java объектов с XML — источниками данных, например jaxb или simple-xml
6. log4j — библиотека для журнализации событий (http://logging.apache.org/log4j/2.x/)
7. Umberello UML modeller — инструмент для реализации канонических UML диаграмм. (https://umbrello.kde.org/)
8. Средство для автоматизации сборки проекта Maven.
9. DB — при выборе базы данных следует остановится на предпочтительном для себя варианте, тип платформы и версия не имеют значения, основным критерием выбора является наличие для данной платформы драйверов JDBC 4-го поколения. Я рекомендую остановиться на двух платформах HSQL или MySQL.
10. MongoDB v.5 и выше, No-SQL база дынных, для реализации возможности ведения истории объектов.
11. Библиотека Jackson v.2 для преобразования объектов в формат JSON и работы с YAML конфигами.
12. JDBC драйвер для взаимодействия в Mongo DB (https://www.cdata.com)

Внимание! Все перечисленные технологии и инструменты распространяются по свободной лицензии и соответственно являются бесплатными. При загрузке с ресурсов-производителей вам необходимо выбрать дистрибутив исходя из версии вашей операционной системы. Хотя, в большинстве своем, предлагаемые инструменты являются кросс-платформенными.

Результаты лабораторных работ, являются основой для волнения курсовой работы.

Курсовая работы или сводный отчет по лабораторным работам должны состоять из следующих частей:

1 Пояснительная записка, в которой представлены и описаны модели выбранной предметной области.

2 Библиотеки классов в виде jar пакетов, с реализованным CLI API.

3 Набор тестов, реализованных с помощью JunitTest технологии.

Методы для работы с библиотекой согласовываются с преподавателем!

Пояснительная записка должна состоять из следующих разделов:

1 Техническое задание.

Описываются основные функциональные возможности библиотеки, конкретно без лишних слов, и предметная область в рамках которой выполняется моделирование.

2. Диаграмма вариантов использования.

3. Диаграмма классов.

4. Диаграмма деятельности. Описывает основные элементы бизнес-логики.

5. Диаграмма компонент. Детализирует основные компоненты.

6. Тестирование. Данный раздел должен полностью соответствовать лабораторной работе «Создание Unit тестов» данной методичке.

Таким образом, общий объем ПЗ должен быть не менее 10 страниц!!!!

При сдаче курсовой работы, необходимо виртуозно ориентироваться в UML моделировании. Блестяще, разбираться в исходных кодах, уметь изменять структуру бинов и методов. Просто нереально круто писать JunitTest.

Структура и содержание пояснительной записки:

1 Титульный лист. Стандартная структура, с указанием исполнителя и темы работы. Тема работы согласовывается с преподавателем и должна четка отражать реализацию поставленной задачи. Например:

«Моделирование бизнес-логики для Web-ресуса по тестированию знаний языка SQL» или «Разработка основных функциональных возможностей приложения по оценке знаний студентов с учётом их отношения к профессору и нереальной любви к игре в Мафию»

2. Техническое задание.

3. UML моделирование

3.1. Диаграмма вариантов использования

3.2. Диаграмма классов

3.3. Диаграмма деятельности

3.4. Диаграмма компонент.

(Могут быть добавлены другие типы канонических UML диаграмм, если они уместны и планируется их использование при выполнении выпускной работы)

4. Сценарии тестирования и результаты тестов.

Лабораторная работа

«Проектирование модели предметной области»

Целью работы является проектирование бизнес-объектов и логики позволяющей реализовать основные функциональные возможности.

Результаты работы должны быть представлены в виде UML диаграмм (классов и вариантов использования и т.д.). В конечном итоге на основании диаграммы классов должны быть сгенерированы сущности в Java коде.

Порядок выполнения:

1. Выбор предметной области. Данный этап является довольно важным, так как от понимания сути предметной области зависит логика реализации основных операций с бизнес-объектами. Поэтому при выборе предметной области я рекомендую опираться на тему своей будущей выпускной работы. Необходимо выбрать несколько стержневых сущностей и кратко описать их назначение.

Например:

Предметная область работы связана с кадровой службой организации. Стержневые сущности: 1) Сотрудники - содержит информацию о каждом сотруднике включающую, идентификационные номер, ФИО, год рождения, пол; 2) Подразделения — содержит информацию о структурных подразделениях организации и сотрудниках, которые в них работают; 3) Проекты — содержит информацию о проектах, которые ведутся в данной организации, об отделе который ведёт проект и о сотруднике, который участвует в данном проекте. При разработке бизнес-логики необходимо учитывать, что сотрудник должен работать только в одном подразделении, одно подразделение может вести несколько проектов и сотрудник может участвовать в нескольких проектах.

После описания модели предметной области, необходимо описать основные операции которые нужно будет реализовать для работы с бизнес-объектами.

Например:

Основные функциональные возможности разрабатываемой информационной системы:

1. Реализовать базовые операции добавления, удаления, редактирования для каждой стержневой сущности.

2. Реализовать функции поиска объектов стержневых сущностей, Сотрудников по ФИО, проекту в котором участвует подразделения в котом работает.

Внимание!!!! При возникновении трудностей с выбором предметной области нужно незамедлительно обратиться к преподавателю!!!!!!

2. Проектирование.

С помощью Umberello UML modeller необходимо построить UML диаграммы вариантов использования и классов.

Для каждого атрибута стержневой сущности необходимо добавить соответствующие методы getXXX() и setXXX(). При разработке информационной системы необходимо опираться на концепцию «слоев». Архитектура информационной системы на базе концепции слоёв представлена на рис.1. В качестве конвенции имен стержневых сущностей и бизнес-логики, рекомендую вам придерживаться следующего подхода, при именовании пакетов:

ru.sfedu.<app_name>. <func_pkg>

1 2 3

где: 1 — общая часть характеризующая страну и организацию, где выполняется проект;

2 — сокращенное имя проекта;

3 — название пакета по функциональному назначению, например для java bean - ru.sfedu.mars.beans или ru.sfedu.mars.model

При именовании классов и файлов следует учитывать прикладной и функциональный аспект, например mars-config.xml, DbTools.java — класс для работы с базой данный, IdbTools.java — интерфейс для работы с базой. Обращаю внимание, что для именования конфигурационных файлов, я рекомендую использовать только буквы в нижнем регистре. Для простоты понимания функционального назначения файла, рекомендую использовать сокращения двойные и более, например DsToolCsv.java или ValidBeanField.java.

На следующем этапе проектирования необходимо создать абстрактные классы в которых реализованы методы для взаимодействия со стрежневыми сущностями.

Заключительным этапом проектирования, является генерация Java-кода с помощью Umberello UML modeller.

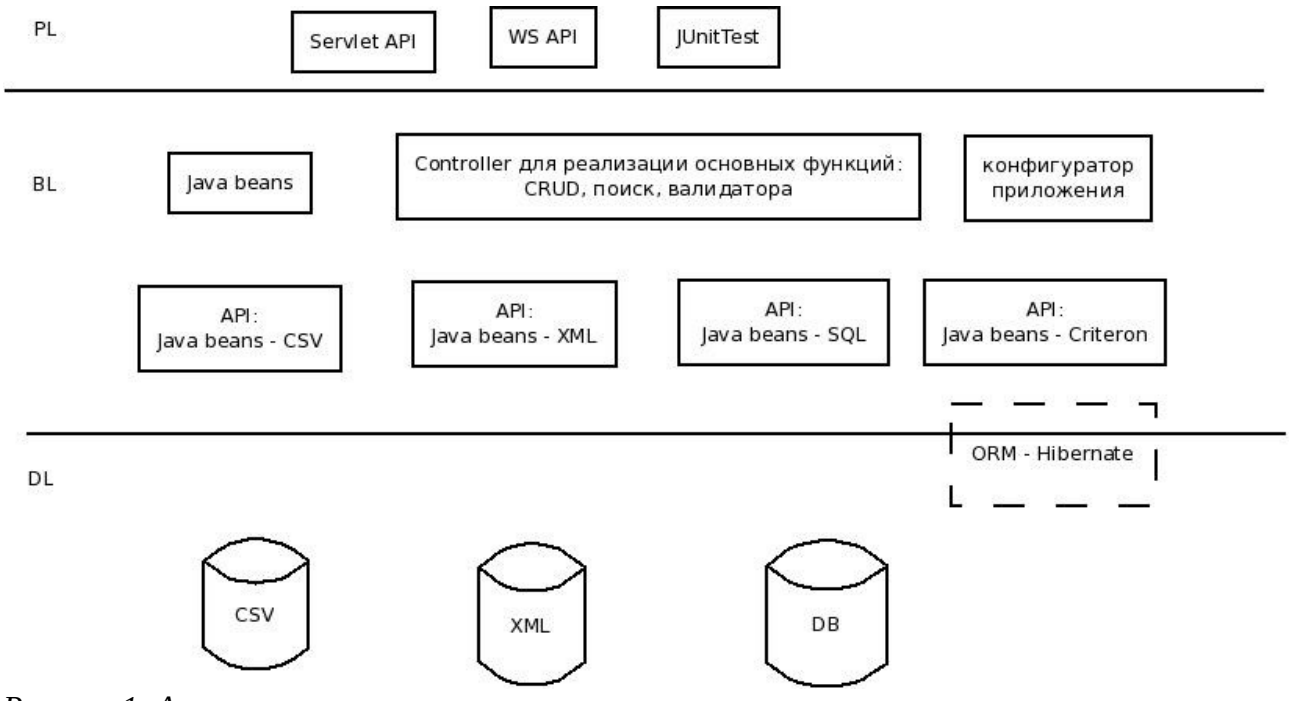

Рисунок 1: Архитектура системы

Лабораторная работа

Настройка журнализации событий

Цель работы: создание maven — проекта, подключение библиотеки log4j и создание консольных и файловых аппендеров, разработка Unit — тестов.

Результаты работы: maven проект, в виде Java библиотеки с реализацией методов и тестов для проверки логгирования.

Порядок выполннения работы:

1. ***Создание проекта.***

В основном меню NetBeans необходимо выбрать пункт File далее New Project далее в окне New Project выбрать ноду Maven и в правом поле Project выбрать Java Application. В результате во вкладке project можно увидеть структуру созданного проекта которая представляет собой ноды Source Packages, Test Packages, Other Source. Далее в ноде Source Packages необходимо создать пакет для своего приложения — ru.sfedu.<my_pk_name>.

2. ***Подключение библиотеки логирования log4j.***

В ноде Other Source (src/main/resources) необходимо создать конфигурационный файл log4j.properties на основании которого в приложении будут созданы аппендеры. Рекомендую создать два аппендера — консольный и файловый, в файловом аппендере будем ограничивать размер файла до 500 Кб. После чего в ноде Dependencies добавить ссылку на библиотеку log4j.

Листинг файла log4j.properties:

```
# Root logger option
log4j.rootLogger=DEBUG, stdout
 # Внимание!!! Указываете имя своего пакета, и устанавливаете уровень логгирования DEBUG
log4j.logger.ru.sfedu.<yuor packege name>=DEBUG,<Имя файлового аппендера>

# Direct log messages to stdout
#Определение косольного аппендера
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n
```

direct messages to file

log4j.appender.<**Имя файлового аппендера**> =org.apache.log4j.RollingFileAppender

log4j.appender.<**Имя файлового аппендера**>.File=<**Имя файла**>

log4j.appender.<**Имя файлового аппендера**>.append=true

log4j.appender.<**Имя файлового аппендера**>.MaxFileSize=500KB

log4j.appender.<**Имя файлового аппендера**>.MaxBackupIndex=2

log4j.appender.<**Имя файлового аппендера**>.immediateFlush=true

log4j.appender.<**Имя файлового аппендера**>.layout=org.apache.log4j.PatternLayout

log4j.appender.<**Имя файлового аппендера**>.layout.ConversionPattern=%d{MM.dd:HH.mm.ss} %5p\t%c:%L - %m%n

Зависимость последней версии логгера

```
<dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-slf4j-impl</artifactId>
        <version>2.9.0</version>
    </dependency>
```

Пример конфигурации последней версии логгера:

```
appenders = console

appender.console.type = Console
appender.console.name = STDOUT
appender.console.layout.type = PatternLayout
appender.console.layout.pattern = %d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n

rootLogger.level = debug
rootLogger.appenderRefs = stdout
rootLogger.appenderRef.stdout.ref = STDOUT
```

Далее вам необходимо в ноде Source Packages, в корневом пакете вашего проекта создать Java класс - <MyProject>Client.java. Этот класс, буде являться API для вызовов основных методов вашего приложения. Далее в разделе объявления переменных, необходимо объявить переменную логгера, в разделе импорт при этом должен добавиться импорт пакетов из библилотеки log4j:

```
    private static Logger log = Logger.getLogger(<Your class Name>.class);
```

После чего необходимо создать конструктор данного класса без параметров и в нем вывести сообщение в логгер:

log.debug("<Your constructor name>[0]: starting application.....")

Далее, создаем в нашем клиенте простой метод:

```
   private void logBasicSystemInfo() {
      log.info("Launching the application...");
      log.info(
            "Operating System: " + System.getProperty("os.name") + " "
            + System.getProperty("os.version")
      );
      log.info("JRE: " + System.getProperty("java.version"));
      log.info("Java Launched From: " + System.getProperty("java.home"));
      log.info("Class Path: " + System.getProperty("java.class.path"));
      log.info("Library Path: " + System.getProperty("java.library.path"));
      log.info("User Home Directory: " + System.getProperty("user.home"));
      log.info("User Working Directory: " + System.getProperty("user.dir"));
      log.info("Test INFO logging.");
}
```

3. ***Создание JUnitTest.***

Для проверки правильности конфигурирования логгера создадим тесты, в дальнейшем для каждого метода нужно будет создавать свой тест. Для этого необходимо вызвать контекстное меню на файле в ноде проекта и выбрать пункт Tools/Create/Update test. После чего в структуре проекта в ноде Test Packages будет создан тест-файл автоматически. Переместив курсор в нужный тестовый метод и вызвав контекстное меню, необходимо выбрать пункт Run focused test method. Перед запуском метода необходимо проверить наличие аннотации @Test перед методом. В результате выполнения метода в вкладке output будет выедена системная информация, а в файловой структуре должны создаться log — файлы с аналогичной информацией. В случае если метод выполнился с ошибками или отсутствует логгируемая информация, необходимо еще раз проверить все настройки и добиться успешного выполнения методов.

Внимание!!!! В дальнейшем для выполнения отладки вашего приложения категорически недопустимо использовать системный вывод в консоль в основном коде проекта с помощью команды System.out.println(). Это будет считаться ГРУБЕЙШЕЙ ОШИБКОЙ и приведет к значительному снижению оценки по дисциплине!!! Для

просмотра отладочной информации и записи всех событий приложения необходимо пользоваться только логгером!!!!

Использование System.out.println() допустипо только в модулях UnitTest.

4. Реализация основных функциональных методов и создание тестов для них.

Необходимо создать соответствующий пакет и скопировать в него java классы стержневых сущностей полученных в результате первой работы. Далее необходимо создать общие (jeneric) методы которые оперируют только Java коллекциями и объектами для реализации заявленного функционала. Далее необходимо написать тесты, в которых инициализировать с помощью java объектов соответствующие параметры методов и выводить результат их выполнения. В каждый метод необходимо добавлять как можно больше сообщений для логгирования, используя уровень DEBUG.

Лабораторная работа

Настройка переменных окружения библиотеки.

К переменным окружения относятся параметры которые необходимо передать приложению для обеспечения его функционирования. К ним относятся параметры соединения с базой данных, путь к рабочему каталогу и т.п. Таким образом, любое изменения внешних параметров не требует перекомпиляции и сборки проекта, и позволяет довольно легко выполнять отладку при изменении внешних параметров. Для хранения переменных окружения могут использоваться следующие форматы файлов -properties, yml и xml. Непосредственно сам путь к файлу переменных окружения передается через свойства JVM, а нужное свойство вызывается через системные методы JVM — System.getProperty(<имя свойства>). Передача пути переменных окружения приложения, осуществляется путем установки параметров -D при старте JVM. Однако при разработке приложения для удобства прописывается путь по умолчанию, что означает, если указанное свойство не будет найдено среди свойств JVM то будет использован путь по умолчанию.

Внимание!!! При разработке вашего приложения необходимо избегать указания в коде абсолютных путей к файлам, инициализации текстовых переменных с заданными значениями!!!!

Для взаимодействия с конфигурационными файлами в корневом пакете вашего проекта необходимо создать класс Constants.java. Данный файл должен содержать строковые константы для взаимодействия с конфигурационным файлом. Т.е. это должны быть имена параметров по которым будут получены значения. Например:

public final String PATH_TO_DB = "ru.myapp.dbhome";

Обращаю внимание на правило именования констант, имена должны содержать только заглавные буквы и нести смысловую нагрузку.

Рабочее задание.

1. Реализовать хранение переменных окружения в трех форматах properties, yml и xml, написать для каждого утилиту с соответствующими методами.
2. Реализовать в тестах возможность получения коллекций типа List <String> например, для набора планет Земля, Сатурн, Марс, Венера.
3. Реализовать в тестах возможность получения коллекций типа Map<Integer,String> например, для набора номер — месяц года.

Лабораторная работа

«Реализация возможности ведения истории изменения объектов»

Идея данной работы возникла на основании моего практического опыта разработки систем. Наверное, прочтя тему работы, у некоторых студентов, выбравших светлый путь добра, возникнет вопрос, зачем еще добавлять дополнительный механизм логгирования усложняя логику работы системы? Вот аргументы, которые должны развеять все сомненья:

1 Для корпоративных систем вопросы быстродействия не являются первоочередными. На первом месте стоят вопросы связанные с надежностью операций над объектами. Поэтому внесение дополнительной логики, которая может повлиять на быстродействие некритично.

2 Логгирование представляет собой процесс сбора информации о всех событиях происходящих во время работы системы, которая хранится в специальных файлах, размер которых как правило ограничен либо временным диапазоном либо параметрически. Цель логгирования — определение причины сбоя. Этот факт затрудняет поиск основных манипуляций над каким-то конкретным объектом во временной ретроспективе. Особенно остра эта проблема в производственных условиях, когда имеется большое количество таких файлов, например размером 100Мб.....

3 Уникальная возможность реально познакомится с No-Sql базой данных!

Если сомненья не развеяны, то у вас нет вариантов ;-)

Поэтому, учебная задача будет заключаться в следующем. Для каждого entity объекта необходимо фиксировать его состояние после операции в NO-Sql базе данных.

Для этого нужно создать entity-bean — HistoryContent, который будет храниться в MongoDB. Предлагаю следующую сигнатуру этого объекта.

1) Id — иденификтор (UIID)

2) className — имя класса объекта, которое можно получить используя метод getSimpleName().

3) createdDate — текущая дата создания

4) actor — имя пользователя выполнявшего операцию, по умолчанию в константах указать значение «system»

5) methodName — имя метода в котором объект участвовал

6) object <String, Object>- json представление состояние объекта после выполнения операции

7) status — статус выполнения операции, типа enum (SUCCESS, FAULT)

Для реализации возможности конвертации JSON объекта рекомендую использовать следующий метод:

```
protected <T> List<T> jsonArrayToObjectList(List<Map<String, Object>> map, Class<T> tClass) {
    try {
        ObjectMapper mapper = new ObjectMapper();
        CollectionType listType = mapper.getTypeFactory()
                .constructCollectionType(ArrayList.class, tClass);
        List<T> objects = mapper.convertValue(map, listType);
        return objects;
    } catch (Exception ex) {
        log.error(ex.getMessage());
        throw new ClassCastException(ex.getMessage());
    }
}
```

Рабочее задание

1 Установить драйвер MongoDB и подключить его к вашей IDE. На сайте разработчкиа Cdata, подробно описан процесс подключения драйвера к лучшей IDE - Apache NetBeans! Это позволит вам выполнять навигацию по вашей базе данных.

2 Создать Entity-bean и написать тест для его сохранения в MongoDB.

3 Добавить в интерфейса репозитория MongoDB специфинчые методы для работы с вашими объектами

4 Написать метод конвертации вашего объекта в JSON -объект, обратный методу приведённому выше.

Лабораторная работа

«Реализация API для работы с файловыми источниками данных»

Цель работы: создание API с реализацией основных CRUD функций (Create Read Update Delete - Создание чтение обновление удаление при работе с персистентными хранилищами данных) для *csv и *xml источников данных, разработка Unit — тестов.

Результаты работы: модуль JunitTest, позволяющий проверить правильность работы методов.

Порядок выполннения работы:

1 Ознакомится с документацией openscv и simple-xml библиотек.

2. Добавить зависимости для этих библиотек в pom.xml вашего проекта.

3. Разработать класс для каждой библиотеки, реализующий абстрактные методы интерфейса с CRUD методами. Например: IDataProvider — Абстрактный интерфейс.

Содержит методы : saveRecord(<Ваш класс>), deleteRecord(<Ваш класс>), getRecordById(long id), initDataSource();

Реализация для CSV источника данных: DataProviderCsv implements IDataProvider.

Реализация методов абстрактного интерфейса, для каждого источника данных, заключается фактически к приведению объекта модели вашей предметной области в виде Java-класса к формату записи данного источника. Например, для CSV — необходимо преобразовать объект в строку соответствующую вашей записи в файле.

4. Создать Unit тесты

Во избежания плагиата и нарушения авторских прав, более детальная информация по реализации методов будет обсуждаться лично с каждым студентом.

Лабораторная работа

«Обработка ошибок»

Цель работы: изучение команды try-catch-finally, научиться писать тесты принудительно инициирующие исключения, научится обрабатывать эти исключения и обеспечивать прохождение этих тестов.

Результаты работы: набор методов JunitTest, позволяющих проверить правильность пути к заданному файлу, контроль контекста файла, как для значения так и для количества записей. Для каждого случая необходимо реализовать два сценария тестов — положительный и отрицательный.

Порядок выполннения работы:

Создайте тестовый метод получения файла по имени:

/**

```
* Отрицательный сценарий
*/
@Test (expected = MyException.class )
public void checkInvalidFileName(){
<вызов собственного метода>
}
```

Далее необходимо в вашем API реализовать метод загрузки файла и затем его использовать в тестах:

```
public void uploadFile(String fileName){
try{
FileInputStream stream = new FileImputStream(fileName);
stream.close();
}catch (FileNOtFoundException e){
throw new MyException («something wrong!»,e);
}
}
```

Обращаю ваше внимание, что бездумное копирование кода из методички может приводить к нежелательным последствиям. В конечном итоге, метод API должен иметь два теста, первый — успешное выполнение, при корректных начальных параметрах. Второй — успешное выполнение при некорректных начальных параметрах. Использование в работе методов без обработки исключительных ситуаций считается **грубейшей ошибкой!**

Лабораторная работа

Реализация API для работы с базами данных

Цель работы: создание API с реализацией основных CRUD функций для платформ СУБД, разработка Unit — тестов.

Результаты работы: модуль JunitTest, позволяющий проверить правильность работы методов.

Порядок выполннения работы:

1 Ознакомится JDBC API .

2. Добавить зависимости для выбранных драйверов БД в pom.xml вашего проекта.

3. Разработать класс для работы с БД, реализующий абстрактные методы интерфейса с CRUD методами.

4. Реализвать возможность переключения платформы БД через конфигурационный файл.

5. Создать Unit тесты

Во избежания плагиата и нарушения авторских прав, более детальная информация по реализации методов будет обсуждаться лично с каждым студентом.

Лабораторная работа

Создание Unit тестов

Цель работы: Научится создавать модульные тесты, учитывающие разные сценарии выполнения методов бизнес-логики, разрабатываемой библиотеки.

Результаты работы: Набор тестов расположенных в папке проекта - Test Packages.

Порядок выполннения работы:

Для создания тестов необходимо использовать разработанные на этапе проектирования UML диаграммы вариантов использования, деятельности и классов. Диаграмма вариантов использования позволяет получить представление о функциональных возможностях метода, диаграмма деятельности — бизнес-логике, а диаграмма классов - о сущностях, которыми оперирует метод. Рассмотрим процесс написания тестов на примере некоторого абстрактного варианта использования «Создание заказа». Предметная область данного варианта может относиться, как к сфере услуг, например, создание заказа на выполнение каких либо работ, либо к сфере общепита, например, заказ на организацию банкета, либо к сфере торговли и т.п. Иными словами, это универсальный вариант, который может быть использован в различных областях. Определим входные данные, которые по сути являются сигнатурой метода. Очевидно, в данном случае это будет:

1. Название, косвенно указывающее на принадлежность заказа к чему либо. Например, «Организация свадебного банкета» или «Закупка СИЗ для ЮФУ»;
2. Предмет договора, список услуг или товаров;
3. Исполнитель или ответственное лицо за формирование заказа;
4. Заказчик;
5. Срок исполнения заказа.

Результатом исполнения метода, является Entity сущность отображённая в источнике данных. Хочу напомнить, что в учебных целях для методов внешнего API не допустимо использовать ссылки на сущности предметной области. Это значит, что список услуг или товаров, как и ответственное лицо, передаётся в метод в виде их идентификаторов в источнике данных. Следовательно, исходный вариант использования, должен включать в себя ряд методов, позволяющих получать по идентификатору сущности из внешних источников данных, типа

getXXXById (long userId). Поэтому написание тестов следует начинать с методов сохранения этого объекта saveMyObj(MyObj myObg) и затем, получения по идентификатору getXXXById (long userId).

Отдельно хочу, уделить особое внимание вопросам создания объектов. Очевидно, что экземпляра класса одного объекта, может использоваться в нескольких методах. В данном конкретном примере, нам нужно вначале его сохранить, а затем получить. Следовательно речь идет об идентичных экземплярах класса. Самое простое решение, в каждом методе создавать экземпляр класса с аналогичными атрибутами. Но во-первых, будет очень много повторяющегося кода, во-вторых, при необходимости изменения какого-либо значения атрибута придется в каждом методе это отслеживать, в третьих, в случае года речь идет о коллекциях объектов, размер повторяющегося кода будет сильно увеличен. А с учетом того, что тесты нужно будет написать для различных дата провайдеров, проблема компактности кода будет очень актуальной. Одно из решений, которое я предлагаю реализовать, заключается в следующем. В корневом пакете тестов, создать класс TestBase, в этом классе написать методы для инициализации необходимых объектов MyObj createMyObj() и наследовать этот класс в соответствующем классе тестов для дата провайдера. Выглядеть это должно следующим образом:

```
public class CsvDataProviderTest extend TestBase{
// Инициализация логгера
private static final Logger log = Logger.getLogger(CsvDataProviderTest .class.getName());
// Метод котрый будет выполнять перед каждым тестом
@Before
initTestData(){
saveMyObj(createMyObj());
}

//  сохранение объекта
saveMyObj(MyObj myObj){
try{
MyObj myObj = myObj.....
// реализация сохранения объектами

}catch(Exception ex){
throw e;
```

}

}

Выше, приведен пример реализации, которая будет зависеть от каждого конкретного случая. Что нужно для себя вынести из этого примера? :

1 - в тестах можно и нужно использовать наследование;

2 - в тестах нужно избегать многократной инициализации данных, и вообще наличие повторяющегося кода это моветон .

Таким образом, стратегия написания тестов должна быть следующей:

1 Необходимо написать сценарий тестирования. Каждый тест должен иметь позитивный и негативный сценарий исполнения; (данный пункт будет рассмотрен ниже)

2 Реализация логики метода в тестах и его отладка;

3 Перенос логики в класс API соответствующего дата провайдера;

4. Контролировать процесс выполнения теста только с помощью соответствующего метода assretXXXX класса Assert;

5. При необходимости использовать логгер.

Рассмотрим более подробнее пункт 1, на примере абстрактного метода createOrder({signature}). Предположим, данный метод включает исполнение двух методов calculatePrice({signature}) и calculateTime({signature}), тогда сценарий исполнения тестов, можно свести в следующую таблицу:

Тестовый метод	Тип сценария	Тестируемый метод	Что контролируется?
Создание заказа testCreateOrder	Позитивный	DataProvider.createOrder: 1) Инициализация объекта Order 2)Вычисление цены calculatePrice({signature}) 3)Вычисление времени calculateTime({signature}) 4) сохранение заказа	1) Рассчитанная цена assertEquals("Цена:", expectedPrice, order.getPrice()); 2) Рассчитанное время assertEquals("Время:", expectedTime, order.getTime()); 3)Успешное выполнение assertNull(createOrder({signature}))
Создание заказа testCreateOrderWrongPrice	Негативный	DataProvider.createOrder: 1) Инициализация объекта Order 2)Вычисление цены calculatePrice({Wrong data})	1) Null значение assertEquals(createOrder({signature}),NullPointerException)
Создание заказа testCreateOrderWrongTime	Негативный	DataProvider.createOrder: 1) Инициализация объекта Order 2)Вычисление цены calculatePrice({signature}) 3)Вычисление времени calculateTime({Wrong Data})	1) Null значение assertEquals(createOrder({signature}),NullPointerException)

Таким образом, в проекте не должно быть ни одного метода для которого отсутствуют

позитивные и негативные тесты!

Регламент приёмки курсовой работы

1 Проверка диаграммы вариантов использования.

1.1 Таблица детализации вариантов использования должна точно соответствовать диаграмме, таблица должна иметь два столбца — Вариант использования и Детализация;

1.2 При детализации варианта использования, должен быть описан список входных и выходных параметров при необходимости;

1.3 При наличии расширяющих вариантов использования в детализации родительского варианта должно быть описано условие их вызова и каким образом этот вариант получает параметры для своего исполнения;

1.4 При наличии включающих вариантов использования, необходимо указать каким образом данный вариант получает свои параметры, либо они вычисляются в процессе выполнения родительского варианта, либо они передаются в сигнатуре родительского метода;

1.5 В сигнатуре внешних методов библиотеки не должно быть ссылок на спецефичные объекты предметной области;

1.6 Диаграмма должна строго соответствовать графической нотации, т.е. наличие несоответствующих связей или графических примитивов недопустимо.

2. Проверка диаграммы классов.

2.1 В курсовой работе должно быть не менее 5 Entity сущностей.

2.2 Обязательно должны быть реализованны отношения обобщения, агрегации или композиции.

2.3 Имена классов должны быть понятны, и являться именем существительным на английском языке.

2.4 Не должно быть кольцевых связей, т.е. ситуации когда класс по отношениям замыкается сам на себя.

2.5 Не должно быть повторяющихся отношений между дочерними классами и другой сущностью.

2.6 Не допустимо наличие класса «обертки», не имеющего никакого семантического смысла в рамках предметной области.

2.7 Каждая Entity - сущность должна иметь свою реализацию CRUD операций для каждого источника данных, исключения составляют сущности находящиеся в отношении обобщения, для которых можно использовать общие методы.

2.8 Диаграмма должна строго соответствовать графической нотации, т.е. наличие несоответствующих связей или графических примитивов недопустимо.

3. Проверка диаграммы вариантов использования и кода;

3.1 Методы диаграммы использования должны точно соответствовать методам абстрактного интерфейса, т.е. не должно быть ситуации, когда метод присутствует на диаграмме, но его нет в API. Обратная ситуация допустима, например, наличие специфичных методов в API;

3.2 Все входные и выходные параметры должны быть описаны в каждом методе в аннотациях JavaDoc;

4 Проверка диаграммы деятельности по диаграммам вариантов использования и классов.

4.1 Допустимо, что бы диаграмма деятельности описывала только базовый вариант, а дочерние варианты были включены в нее без детализации.

4.2 Диаграмма деятельности должна точно соответствовать детализации варианта использования.

4.2 В случае если для реализации логики в нотации диаграммы нужен какой-то класс, то он должен точно соответствовать диаграмме классов.

5 Проверка кода

5.1 Недопустимо использование системного вывода.

5.2 Обязательно наличие логгеров в базовых методах уровня INFO и DEBUG, а также ERROR при отслеживании ошибок.

5.3 Обязательно отслеживание исключительных ситуаций с помощь try и catch.

5.4 Обязательно наличие аннотаций для JavaDoc и они должны соответствовать диаграмме вариантов использования.

5.5 Обязательно использование методов интерфейса Stream и лямда выражений при работе с коллекциями.

5.4 В случае если метод возвращает какой либо экземпляр класса предметной области, то он должен быть «упакован» в класс java.util.Optional.

5.5 Обязательно наличие тестов, которые должны соответствовать пункту Тестирование в ПЗ. Каждый метод API должен иметь соответствующий тест.

Правила сдачи курсовой работы.

Каждый студент в установленный срок должен прислать архив в формате zip, структура архива должна строго соответствовать описанному ниже. Все имена файлов

```
и самого архива должны содержать только латинские буквы. Имя архива должно
соответствовать фамилии в английской транскрипции.
Структура архива:
1 собранный  jar файл проекта .
2 environment.property - файл со свойствами проекта.
3. log4j.property - конфигурация логгера.
4. readme.txt файл с примерами команд для запуска приложений и с кратким
комментарием на русском языке
5. Если нужно, то можно размещать необходимые jar файлы для запуска вашего
приложения.
6. ПЗ курсовой работы
7. Папка проекта, с исходными кода и тестами

Сигнатура команды запуска:
java -jar -Dconfig.<название свойства и путь к конфигу свойств> -
Dlog4j.<название свойства и путь к конфигу логгера> <тип провайдера> <имя
операции> <данные>
При запуске программы должен быть активирован консольный и файловый аппендер для
логера. После запуска будет проанализирован выходной поток, который  не должен
содержать никаких ошибок.
```

На первом этапе выполняется проверка работы на основе регламента представленного выше.

Далее, будет выполнен запуск тестов, который должен пройти без ошибок.

Последний этап заключается в запуске команд представленных в ReadMe.txt и изменением уровня логирования.

Курсовая считается сданной при успешном прохождении всех этапов.

Желаю всем успехов!!!!

Б.Б.