**Министерство образования и науки Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**   
**высшего образования**   
**«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»**   
**(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ \_\_Робототехника и комплексная автоматизация\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_Компьютерные системы автоматизации производства\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОМУ ПРОЕКТУ***

***НА ТЕМУ:***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

Студент \_\_\_\_РК9-72\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Минеев М.А.\_\_\_\_\_\_\_\_\_

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Урусов А. В.**\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Урусов А. В.\_\_\_\_\_\_\_\_\_

(Подпись, дата) (И.О.Фамилия)

*2017 г.*

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc501648173)

[ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 4](#_Toc501648174)

[ВВЕДЕНИЕ 5](#_Toc501648175)

[1 Предпроектное обследование 5](#_Toc501648176)

[1.1 Информация о предприятии 5](#_Toc501648177)

[1.2 Процесс обработки заказа 5](#_Toc501648178)

[1.3 Хранение данных о заказах 6](#_Toc501648179)

[1.4 Постановка задачи 8](#_Toc501648180)

[2 Концептуальное проектирование 9](#_Toc501648181)

[2.1 Выбор средства проектирования 9](#_Toc501648182)

[3 Разработка технического задания 11](#_Toc501648183)

[3.1 Основные сведения 11](#_Toc501648184)

[3.2 Назначение разработки 11](#_Toc501648185)

[3.3 Требования к программе или программному изделию 11](#_Toc501648186)

[3.4 Стадии и этапы разработки 13](#_Toc501648187)

[3.5 Порядок контроля и приемки 13](#_Toc501648188)

[4 Структурное проектирование 15](#_Toc501648189)

[4.1 Диаграмма классов для имитационной модели 15](#_Toc501648190)

[4.2 Изменения в RAO-X 16](#_Toc501648191)

[5 Рабочее проектирование 26](#_Toc501648192)

[5.1 Изменения в RAO-X 26](#_Toc501648193)

[5.2 Проектирование модели 31](#_Toc501648194)

[ЗАКЛЮЧЕНИЕ 35](#_Toc501648195)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 36](#_Toc501648196)

[ПРИЛОЖЕНИЯ 37](#_Toc501648197)

[Приложение 1 38](#_Toc501648198)

[Приложение 2 40](#_Toc501648199)

# ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Заказ – запрос на установку, ремонт, обслуживание котельного оборудования с перечнем необходимых деталей.

Коннектор – java библиотека, реализующая стандарт JDBC для подключения к СУБД.

СУБД – система управления базами данных

Hibernate – библиотека для языка программирования Java, предназначенная для решения задач объектно-реляционного отображения.

JDBC (Java DataBase Connectivity) – платформенно-независимый промышленный стандарт взаимодействия Java-приложений с различными СУБД.

JPA (Java Persistence API) – технология, обеспечивающая объектно-реляционное отображение (ORM) простых java объектов и предоставляющая API для сохранения, получения и управления такими объектами.

JPQL – Java persistence query language, язык расширяющий SQL, и позволяющий писать запросы, указывая классы сущностей.

MySQL – свободная реляционная система управления базами данных. Распространяется под GNU General Public License.

ORM – Object-Relational Mapping, концепция объектно-реляционного отображения.

Query класс – класс для составления типозащищенного запроса, сгенерированный Queryqsl по классу сущности. Данный класс имеет имя сущности с приставленным вначале символом «Q».

Querydsl – java библиотека, использующая язык java для составления типозащищенных запросов к СУБД.

RAO-X – плагин для интегрированной среды разработки Eclipse, позволяющий вести разработку имитационных моделей на языке РДО.

# ВВЕДЕНИЕ

## Предпроектное обследование

### Информация о предприятии

ООО "Тепломеханика" – фирма, занимающаяся установкой, оперативным ремонтом и обслуживанием бытового и промышленного котельного оборудования большинства европейских производителей.

Основная деятельность:

* Срочный ремонт газовых котлов различных марок и моделей
* Пуско-наладочные работы, а также настройка котельного оборудования
* Проведение технического обслуживания котлов, заключаем договора
* Промывка отопительных систем а также магистралей водоснабжения
* Монтаж котельных с нуля, начиная с проектирования, подбора котла и т.п.
* Оформление документов, ТУ (технических условий) на подключение газа.

Данная фирма сотрудничает с такими европейскими производителями как: Viessmann, Buderus, DeDietrish, Wolf, Giersch, Immergas, Ferroli, Lamborghini, CipUnigas, Oilon [4].

### Процесс обработки заказа

Фирма принимает заказы на ремонт, обслуживание, установку котельного оборудования. При поступлении заказа формируется список деталей, необходимых для проведения данных операций с оборудованием.

На рисунке 1.1 изображен процесс обработки заказа, построенный на основе описания от заказчика.

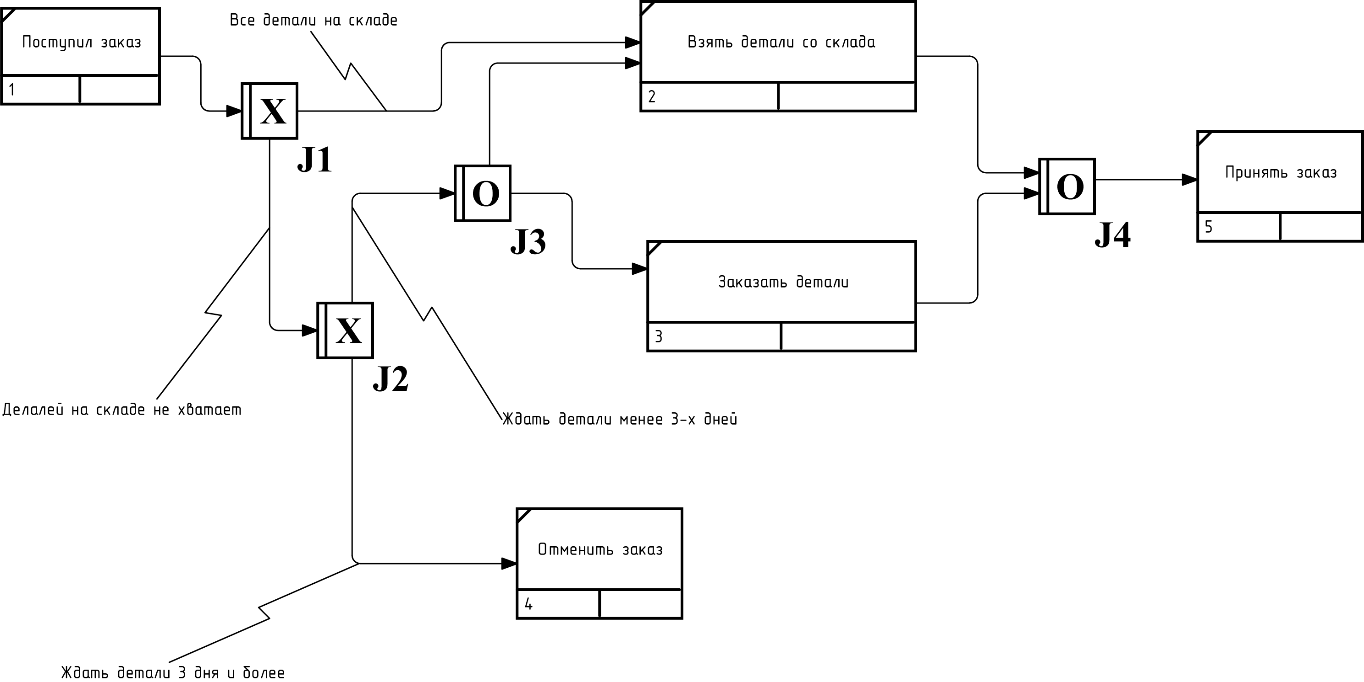


Рисунок 1.1 – IDEF3 диаграмма обработки заказа

При поступлении заказа проверяется наличие деталей на складе, если их хватает, то заказ принимается, в данном случае он считается успешным. В другом случае детали нужно заказать. Если доставку ждать менее 3-х дней, тогда заказ считается успешным, иначе он отменяется и считается неуспешным.

Также производится расчет выручки с каждого заказа. Стоимость продажи детали равна 1.2 от закупочной стоимости детали.

### Хранение данных о заказах

Данные о заказах на предприятии хранятся в СУБД MySQL. Для их исследования был выдан доступ к трем таблицам. По данным таблицам разработана IDEF1x диаграмма (рисунок 1.2).



Рисунок 1.2 – IDEF1x модель базы данных

Таблица *catalogue* хранит информацию о деталях. Приведена информация об аттрибутах, что необходимы для дальнейшего моделирования.

* *manufacturer* – производитель детали
* *name* – наименование детали
* *stocked* – количество на складе
* *catalogueprice* – цена детали

Таблица *sparepartsrequests* содержит информацию о заказах

* *DateOfCreation* – дата создания заказа
* *DateOfProcessing* – дата последнего изменения заказа
* *DateOfRealization* – дата реализации заказа

Таблица *requestedpartslist* содержит информацию о том, какая деталь относится к какому заказу.

* *DateOfDelivery* – дата доставки детали
* *Count* – количество запрашиваемых деталей

Остальные аттрибуты необходимыми для других процессов, и информация по ним не предоставлена.

### Постановка задачи

Проектирование системы начинается с выявления проблемы, для которой она создается. Под проблемой понимается несовпадение характеристик состояния систем, существующей и желаемой.

В результате предпроектного обследования выявлена необходимость определения наиболее выгодной стратегии пополнения склада. В связи с этим необходимо проводить моделирование данного процесса, используя реальную историю заказов, но с разными стратегиями пополнения. И в дальнейшем выбирать наиболее выгодную стратегию.

Для реализации данной задачи целесообразно использование системы имитационного моделирования RAO-X, поскольку она успешно применяется на данном предприятии. Однако данная система имеет существенное ограничение, в RAO-X не реализован механизм взаимодействия с СУБД.

Задачей данного семестра стала реализация данного механизма и построение модели обработки заказов as-is, принимающей данные из СУБД.

## Концептуальное проектирование

### Выбор средства проектирования

В RAO-X нет механизма чтения данных из СУБД. Однако данная система написана на языке java. В данном языке существует стандарт JDBC, благодаря которому можно осуществлять запросы к СУБД на языке SQL в коде java. Сложность возникает в отображении данных сущностей и классов. Обычно SQL запросы пишутся как строки, а данные возвращаются в виде массивов, где строки являются кортежами данных, а столбцы атрибутами. Также необходимо приведение типов. В итоге увеличивается время написания программного кода, и он становится менее читаемым.

Однако в java существует механизм JPA. В основе его лежит объектно-реляционное отображение (ORM). Данное отображение позволяет работать с сущностями из СУБД, как с объектами. Улучшается читаемость кода, пропадает необходимость осуществлять приведение типов. Есть много реализаций данного механизма, в виде библиотек java, однако наиболее распространенной и гибкой является Hibernate.

Для описания источника данных JPA требует наличия файла META-INF/persistence.xml. Данное требование недопустимо для RAO (подробно причины изложены в пункте 4.2.4), но библиотека Hibernate позволяет обойти его.

В JPA существует три способа осуществлять запросы к СУБД:

1. SQL запросы. JPA старается сопоставить возвращаемые данные из СУБД с объектами, которые должны вернуться
2. JPQL запросы. SQL-подобные запросы, где сущностями и атрибутами являются классы и их поля соответственно (Например «select part from part»)
3. Criteria Query запросы. Позволяют составлять типозащищенные запросы на языке java, однако являются трудночитаемыми.

В связи с этим необходимо решение для составления легкочитаемых типозащищенных запросов. В процессе поиска найдена библиотека Querydsl. Она позволяет генерировать специальные классы для составления легкочитаемых типозащищенных запросов.

В итоге данные решения позволят осуществлять запросы разной сложности к реляционным СУБД. Диаграмма пакетов проекта RAO-X после добавления перечисленных выше библиотек представлена на рисунке 2.1

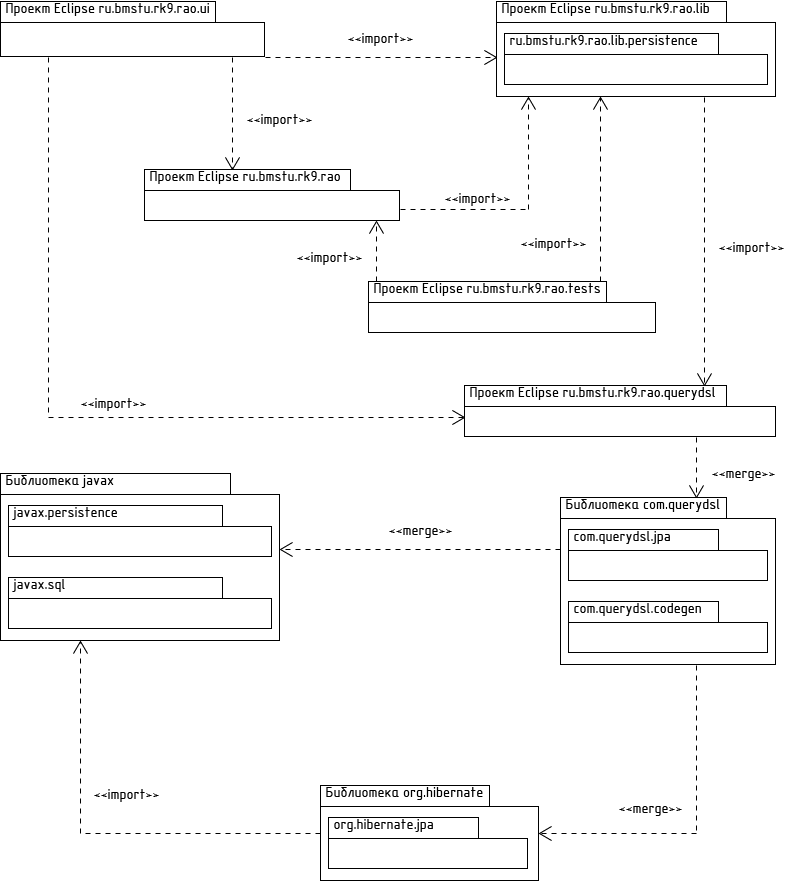


Рисунок 2.1 – Диаграмма пакетов системы RAO-X

## Разработка технического задания

### Основные сведения

Основание для разработки: задание на курсовой проект.

Заказчик: ООО "Тепломеханика"

Разработчик: студент кафедры «Компьютерные системы автоматизации производства» Минеев М. А.

Наименование темы разработки: «Разработка модели управления складом в системе имитационного моделирования РДО»

### Назначение разработки

Разработать модель пополнения складских запасов, использующую статистику заказов

### Требования к программе или программному изделию

#### Требования к функциональным характеристикам

Система должна реализовывать следующие возможности:

* Моделирование процесса обработки заказа as-is
* Чтение данных из СУБД в процессе моделирования
* Использование java классов для описания модели данных
* Использование типозащищенных запросов к СУБД

#### Требования к надежности

Система должна удовлетворять следующим требованиям к надежности:

* Поддержание в исправном и работоспособном состоянии системы RAO-X.
* Поддержание в исправном и работоспособном состоянии модели обработки заказов на языке РДО.

#### Условия эксплуатации

* Эксплуатация должна производиться на оборудовании, отвечающем требованиями к составу и параметрам технических средств, и с применением программных средств, отвечающим требованиям к программной совместимости
* Аппаратные средства должны эксплуатироваться в помещениях с выделенной розеточной электросетью 220В ±10%, 50 Гц с защитным заземлением

#### Требования к составу и параметрам технических средств

Программный продукт должен работать на компьютерах со следующими характеристиками:

* объем ОЗУ не менее 1024 Мб
* микропроцессор с тактовой частотой не менее 1600 МГц
* требуемое свободное место на жестком диске – 4 Гб

#### Требования к информационной и программной совместимости

* операционная система Windows Server 2003 и старше или Ubuntu

15.10 и старше

* наличие в операционной системе ПО Eclipse DSL Tools Mars 2 и

новее

#### Требования к маркировке и упаковке

Не предъявляются.

#### Требования к транспортированию и хранению

Не предъявляются.

### Стадии и этапы разработки

Разработка должна быть проведена в три стадии:

* техническое задание
* технический и рабочий проекты

На стадии «Техническое задание» должен быть выполнен этап разработки и согласования настоящего технического задания.

На стадии «Технический и рабочий проект» должна быть выполнена разработка системы

### Порядок контроля и приемки

Контроль и приемка работоспособности системы автоматизированной сборки, тестирования и развертывания должны осуществляться в процессе проверки функциональности (апробирования) системы в целом, а также в процессе проверки функциональности (апробирования) полученной в результате его работы системы имитационного моделирования RAO-X путем многократных тестов в соответствии с требованиями к функциональным характеристикам системы.

## Структурное проектирование

### Диаграмма классов для имитационной модели

JPA позволяет осуществлять работу с СУБД, используя концепцию ORM (Object-Relational Mapping). Разработчику достаточно описать сущности как классы на языке java, помечая классы и поля аннотациями для осуществления отображения (Подробнее в разделе 5). Для описания связей между классами лучше всего подходит UML диаграмма классов. В соответствии с IDEF1x моделью базы данных (Раздел 1) разработана соответствующая ей UML диаграмма (рисунок 4.1).



Рисунок 4.1 –Диаграмма классов деталей и заказов

### Изменения в RAO-X

#### Добавление java классов в проект

Для использования возможностей механизма JPA для описания модели данных (далее сущностей), необходимо создавать классы, размеченные аннотациями. Осуществить это можно двумя путями:

1. Расширить грамматику языка РДО для описания сущностей, данный подход крайне трудоемкий и потребует вносить изменения в случае изменений в спецификации JPA. Также потребуется составлять документацию.
2. Добавить возможность создавать java классы и использовать их в коде модели, данный метод менее трудоёмкий (не RAO-X генерирует java код, а он пишется самостоятельно), а также освобождает от необходимости составлять документацию, поскольку она уже существует.

Второй вариант однозначно эффективнее, однако необходимо реализовать загрузку java кода в симуляции, для этого были реализованы статические методы, которые сканируют наличие java классов и загружают их при симуляции

Чтобы загружать классы нужно знать их полные имена. Диаграмма метода, получающего полные имена всех классов приведена ниже на рисунке 4.2.

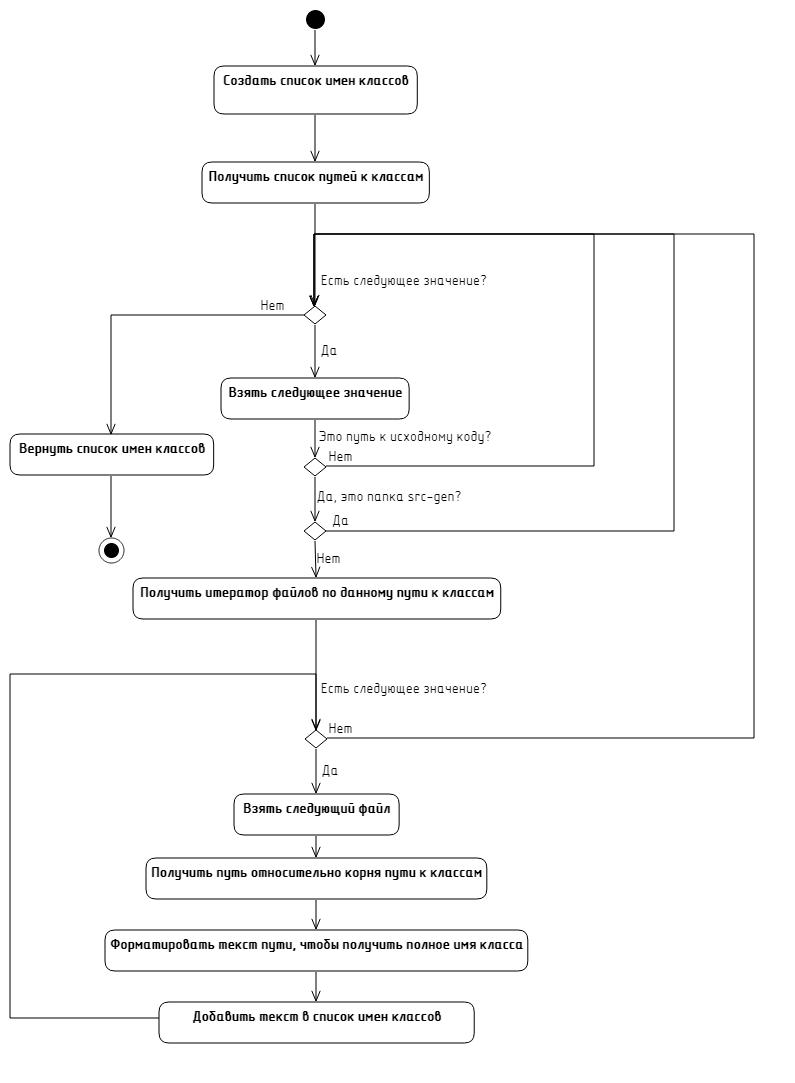


Рисунок 4.2 – Диаграмма активности получения списка классов

Далее полученные в списке классы необходимо загрузить, диаграмма загрузки представлена на рисунке 4.3.

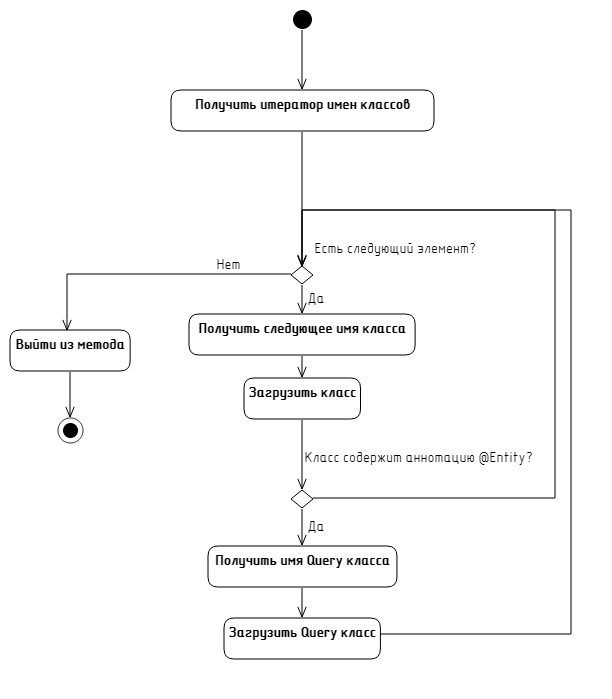


Рисунок 4.3 – Диаграмма активности загрузки классов

#### Добавление библиотек в проекты

За подключение к целевой СУБД в java отвечают коннекторы.   
Коннекторы – java библиотеки, реализующие интерфейс JDBC (Java database connection), и они поставляются разработчиками СУБД. Без них невозможно читать данные из СУБД. Возможны два способа добавления коннекторов в RAO-X:

1. Добавить коннекторы в RAO-X. Данный вариант имеет некоторые сложности. Коннекторов большое количество, для каждой СУБД существует свой коннектор, и в случае его отсутствия в RAO-X понадобится выпускать новую версию с коннектором. Различие версии СУБД и коннектора может привести к невозможности осуществить подключение. Не все коннекторы являются свободно распространяемыми и могут конфликтовать с лицензией RAO-X.
2. Добавлять коннектор как библиотеку в модель, данный подход освобождает от необходимости поддерживать актуальную версию коннектора, эта обязанность ложится на пользователя, если есть необходимость подключить коннектор, не распространяемый свободно, пользователь может получить его своим способом и использовать в модели, не нарушая лицензию RAO-X.

Таким образом, второй вариант является более предпочтительным. Для загрузки библиотек нужно добавлять их в *ClassLoader* модели*.* Пользователю достаточно добавить библиотеку в свойствах проекта. Алгоритм, по которому должен создаваться *ClassLoader* представлен на рисунке 4.4

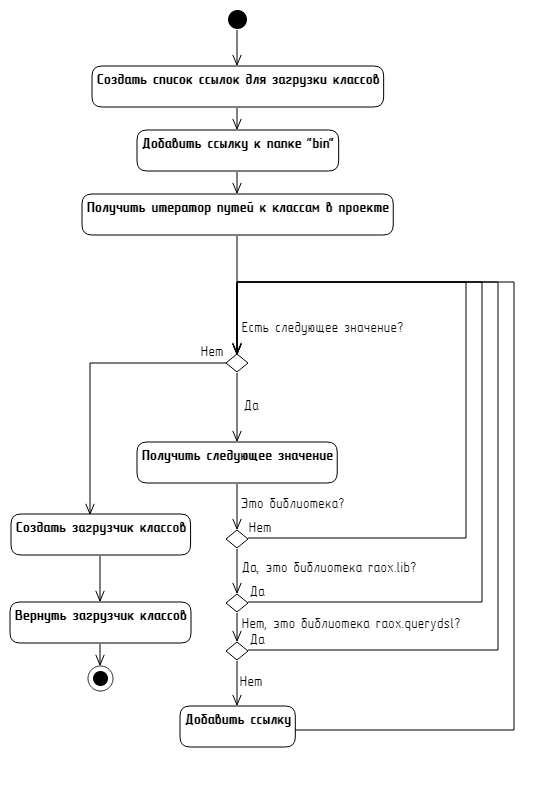


Рисунок 4.4 – Диаграмма активности создания загрузчика классов

#### Генерация классов Querydsl

Для генерации Query классов из классов сущностей требуется настроить генератор кода Querydsl *GenericExporter*. Для этого разработан класс QueryGenerator, осуществляющий данную настройку (рисунок 4.5).

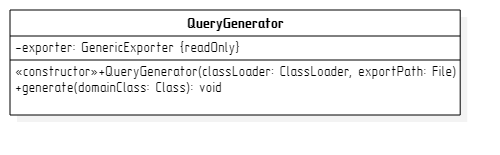


Рисунок 4.5 – Класс генератора Query классов

Библиотека Querydsl должна получать загруженные классы для генерации своих классов. Также нужно отслеживать, был ли сгенерирован хоть один класс, поскольку процесс сборки проекта довольно длительный и нет необходимости собирать проект дважды, если не были сгенерированы классы. Процесс генерации Query классов приведен на рисунке 4.6.

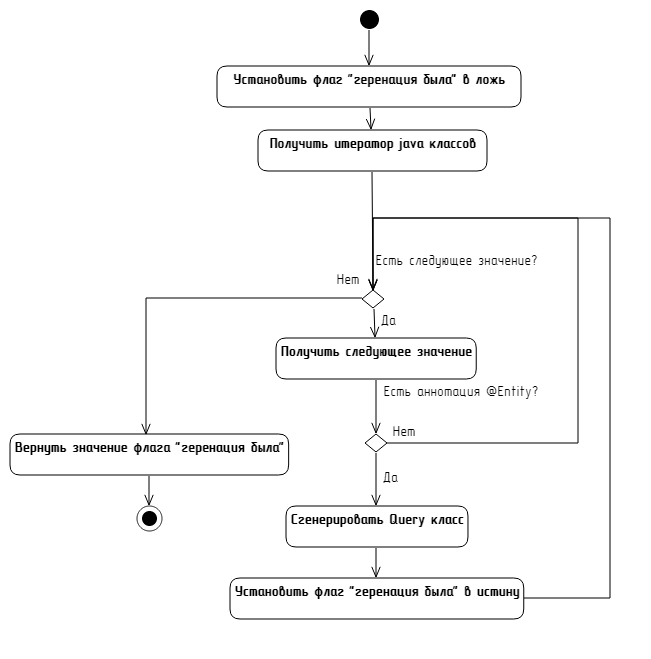


Рисунок 4.6 – Диаграмма активности генерации Query классов

Querydsl для генерации своих классов в формате .java требует передать загруженные .class скомпилированные классы. В связи с этим требуется двухэтапная сборка проекта. На первом этапе компилируются классы сущностей, далее по ним генерируются Query классы и затем они компилируются. Если классы сущностей не были найдены, то сборка проекта проводится только один раз. Процесс двухэтапной сборки проекта представлен на рисунке 4.7.

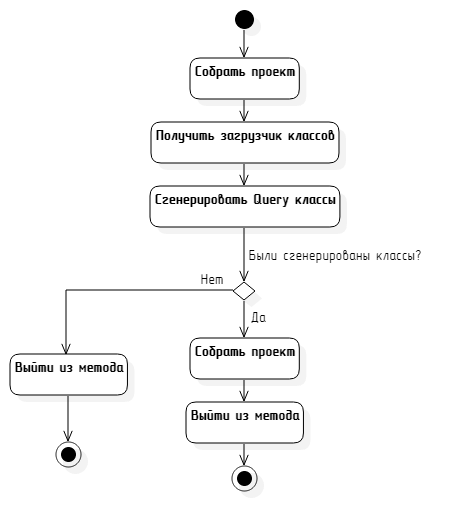


Рисунок 4.7 – Сборка проекта

#### Создание источника данных

Спецификация JPA требует наличия в приложении файла META-INF/persistence.xml, в котором описываются данные для подключения к СУБД. В RAO-X данный подход недопустим ввиду:

1. Наличия трудночитаемой xml структуры;
2. Отсутствия гибкости (указан строгий путь к данному файлу);
3. Необходимости создания отдельного файла (Нет возможности описать данные для подключения в коде модели).

Однако Hibernate позволяет обойти данное требование. В JPA данный файл должен обрабатываться классом, реализующим интерфейс *PersistenceUnitInfo*. В Hibernate присутствует конструктор, которому можно передать реализацию данного интерфейса напрямую, для чего был создан класс *PersistenceUnitInfoImpl* (рисунок 4.8)*.*

Для работы с СУБД в коде модели нужно создать класс, который будет принимать данные для подключения и возвращать объект, реализующие интерфейс *EntityManager,* и класс *JPAQuery.* Диаграмма данного класса представлена на рисунке 4.9.

*EntityManager –* интерфейс, реализации которого позволяют управлять транзакциями, создавать SQL, JPQL запросы (JPQL – Java persistence query language, язык расширяющий SQL, и позволяющий писать запросы, указывая классы сущностей), вызывать хранимые процедуры*.* Для его создания, а также возможности создавать запросы Querydsl нужно реализовать класс, который будет создавать подключение к СУБД.

*JPAQuery –* основной класс в библиотеке Querydsl, с помощью которого составляются типозащищенные запросы.



Рисунок 4.8 – Диаграмма классов реализации интерфейса PersistenceUnitInfo

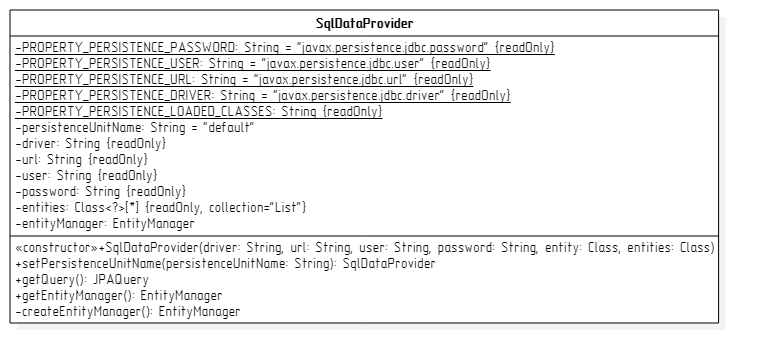


Рисунок 4.9 – Диаграмма класса подключения к СУБД

## Рабочее проектирование

### Изменения в RAO-X

#### Добавление java классов в проект

В разделе 4 была представлена диаграмма получения списка классов, которые нужно загрузить. Реализация данного алгоритма на языке java в виде метода *getJavaClassNames* представлена ниже. Здесь класс проекта оборачивается в класс java проекта, чтобы затем получить список папок с исходным кодом (далее папки). Папка src-gen игнорируется, поскольку классы, расположенные в ней должны загружаться иным образом. Для каждой папки выдается список файлов находящихся рекурсивно внутри. У каждого файла берется путь относительно его папки и преобразуется в полное имя класса. Преобразование осуществляется легко, поскольку полное имя класса совпадает с относительным путем.

**private** **static** List<String> getJavaClassNames(IProject project) **throws** JavaModelException {

IJavaProject javaProject = JavaCore.*create*(project);

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

**for** (IClasspathEntry entry : javaProject.getRawClasspath()) {

IPath path = entry.getPath();

String stringPath = path.toString();

**if** (entry.getEntryKind() != IClasspathEntry.***CPE\_SOURCE***)

**continue**;

**if** (stringPath.endsWith("src-gen"))

**continue**;

IWorkspaceRoot root = ResourcesPlugin.*getWorkspace*().getRoot();

IResource res = root.findMember(path);

List<IResource> resources = **new** ArrayList<>();

IPath sources = res.getLocation();

*recursiveFindFiles*(resources, sources, ResourcesPlugin.*getWorkspace*().getRoot(), "java");

**for** (IResource resource : resources) {

String relative = resource.getLocation().makeRelativeTo(sources).toString();

String className = relative.replace(".java", "").replace('/', '.');

classNames.add(className);

}

}

**return** classNames;

}

Метод *loadJavaAndQueryDslClasses* загружает java классы, на вход он получает проект для передачи методу *getJavaClassNames* и загрузчик классов, с помощью которого загружаются классы. Все классы загружаются вызовом метода *Class.forName.* Однако требуется загрузить Query классы. Они расположены в папке *src-gen* и метод *getJavaClassNames* не вернет их, однако известно, что эти классы генерируются из помеченных аннотацией *@Entity*, с добавлением символа «Q» в имя класса. Далее в коде проверяется наличие данной аннотации и при необходимости загружается нужный Query класс.

**static** **void** loadJavaAndQueryDslClasses(IProject project, ClassLoader classLoader)

**throws** URISyntaxException, CoreException, IOException, ClassNotFoundException {

**for** (String className : *getJavaClassNames*(project)) {

Class<?> entityClass = Class.*forName*(className, **true**, classLoader);

@SuppressWarnings("unchecked")

Class<Entity> entityAnnotationClass = (Class<Entity>) Class.*forName*(Entity.**class**.getCanonicalName(), **true**,

classLoader);

**if** (!entityClass.isAnnotationPresent(entityAnnotationClass))

**continue**;

String queryClassName = className.replaceAll("\\.(?!.\*\\.)", ".Q");

Class.*forName*(queryClassName, **true**, classLoader);

}

}

#### Добавление библиотек в проекты

Исходный загрузчик классов извлечен из класса *ModelInternalsParser* в статический метод *createClassLoader*, сделано это из-за необходимости использовать данный загрузчик не только в процедуре прогона модели, а также при генерации Query классов (в пункте 3)*.* Затем его логика была расширена для добавления возможности загружать внешние библиотеки. Сначала добавляется папка bin, она содержит скомпилированный код проекта. Далее берется список подключенных к проекту библиотек. Не нужно загружать библиотеки *ru.bmstu.rk9.rao.lib* и *ru.bmstu.rk9.rao.querydsl*. Они загружены в самой RAO-X. Остальные добавляются в загрузчик классов.

**static** URLClassLoader createClassLoader(IProject project) **throws** CoreException, MalformedURLException {

String location = *getProjectLocation*(project);

IJavaProject javaProject = JavaCore.*create*(project);

List<URL> urls = **new** ArrayList<>();

URL modelUrl = **new** URL(location + "/bin/");

urls.add(modelUrl);

**for** (IClasspathEntry entry : javaProject.getRawClasspath()) {

IPath path = entry.getPath();

String stringPath = path.toString();

**if** (entry.getEntryKind() != IClasspathEntry.***CPE\_LIBRARY***)

**continue**;

**if** (stringPath.contains(BundleType.***RAOX\_LIB***.name))

**continue**;

**if** (stringPath.contains(BundleType.***QUERYDSL\_LIB***.name))

**continue**;

IWorkspaceRoot root = ResourcesPlugin.*getWorkspace*().getRoot();

IResource res = root.findMember(path);

**if** (res != **null**) {

stringPath = res.getLocation().toString();

}

urls.add(**new** File(stringPath).toURI().toURL());

}

**return** **new** URLClassLoader(urls.toArray(**new** URL[urls.size()]), CurrentSimulator.**class**.getClassLoader());

}

#### Генерация классов Querydsl

Для генерации Query классов необходимо загрузить скомпилированные классы сущностей. При первом этапе сборки проекта классы сущностей компилируются. После создается загрузчик классов, который передается в метод *generateQueryDslCode.* Теперь в папке src-gen появляются Query классы. Второй этап сборки компилирует их.

*recentProject*.build(IncrementalProjectBuilder.***FULL\_BUILD***, monitor);

URLClassLoader classLoader = BuildUtil.*createClassLoader*(*recentProject*);

**boolean** compilationNeeded = BuildUtil.*generateQueryDslCode*(*recentProject*, classLoader);

classLoader.close();

**if** (compilationNeeded)

*recentProject*.build(IncrementalProjectBuilder.***FULL\_BUILD***, monitor);

Метод *generateQueryDslCode* создает генератор Query классов, осуществляет поиск классов сущностей и передает их генератору. Если хоть один класс был найдет, метод вернет *true*. Данное действие необходимо, чтобы не проводить лишний раз сборку проекта, если классы сущностей в проекте не присутствуют.

**static** **boolean** generateQueryDslCode(IProject project, ClassLoader classLoader)

**throws** URISyntaxException, CoreException, IOException, ClassNotFoundException {

IPath target = project.getFolder("src-gen").getLocation();

QueryGenerator queryGenerator = **new** QueryGenerator(classLoader, target.toFile());

**boolean** generatedAny = **false**;

**for** (String className : *getJavaClassNames*(project)) {

Class<?> entityClass = Class.*forName*(className, **true**, classLoader);

@SuppressWarnings("unchecked")

Class<Entity> entityAnnotationClass = (Class<Entity>) Class.*forName*(Entity.**class**.getCanonicalName(), **true**,

classLoader);

**if** (!entityClass.isAnnotationPresent(entityAnnotationClass))

**continue**;

queryGenerator.generate(entityClass);

generatedAny = **true**;

}

**return** generatedAny;

}

Исходный класс генератора Query классов *GenericExporter* не настроен для работы с JPA. Класс *QueryGenerator* осуществляет данную настройку, сообщая генератору классы аннотаций JPA.

**public** **final** **class** QueryGenerator {

**private** **final** GenericExporter exporter;

**public** QueryGenerator(ClassLoader classLoader, File exportPath) {

exporter = **new** GenericExporter(classLoader);

exporter.setKeywords(Keywords.***JPA***);

exporter.setEntityAnnotation(Entity.**class**);

exporter.setEmbeddableAnnotation(Embeddable.**class**);

exporter.setEmbeddedAnnotation(Embedded.**class**);

exporter.setSupertypeAnnotation(MappedSuperclass.**class**);

exporter.setSkipAnnotation(Transient.**class**);

exporter.setTargetFolder(exportPath);

}

**public** **void** generate(Class<?> domainClass) {

exporter.export(domainClass);

}

}

#### Создание источника данных

Как говорилось в разделе 4, спецификация JPA требует наличия в приложении файла META-INF/persistence.xml, и реализация Hibernate позволяет обойти данное требование, используя реализацию интерфейса *PersistenceUnitInfo*. Ниже приведен фрагмент кода данной реализации, содержащий поля класса и его инициализатор. Полный код класса приведен в приложении 1.

**public** **class** PersistenceUnitInfoImpl **implements** PersistenceUnitInfo {

**private** **static** **final** String ***PERSISTENCE\_PROVIDER*** = "org.hibernate.jpa.HibernatePersistenceProvider";

**private** **final** String persistenceUnitName;

**private** **final** ClassLoader classLoader;

**public** PersistenceUnitInfoImpl(String persistenceUnitName, ClassLoader classLoader) {

**this**.persistenceUnitName = persistenceUnitName;

**this**.classLoader = classLoader;

}

...

}

Основной интерфейс для работы с сущностями в JPA – *EntityManager*. Необходимо, чтобы существовал один его экземпляр за все время выполнения программы. Новый создается следующим образом в классе *SqlDataProvider:*

**private** EntityManager createEntityManager() {

**try** {

ClassLoader modelClassLoader = entities.get(0).getClassLoader();

Map<String, Object> properties = **new** HashMap<>();

properties.put(***PROPERTY\_PERSISTENCE\_DRIVER***, driver);

properties.put(***PROPERTY\_PERSISTENCE\_LOADED\_CLASSES***, entities);

properties.put(***PROPERTY\_PERSISTENCE\_URL***, url);

properties.put(***PROPERTY\_PERSISTENCE\_USER***, user);

properties.put(***PROPERTY\_PERSISTENCE\_PASSWORD***, password);

PersistenceUnitInfo persistenceUnitInfo = **new** PersistenceUnitInfoImpl(persistenceUnitName,

modelClassLoader);

EntityManagerFactory emf = **new** HibernatePersistenceProvider()

.createContainerEntityManagerFactory(persistenceUnitInfo, properties);

**return** emf.createEntityManager();

} **catch** (org.hibernate.exception.JDBCConnectionException e) {

e.printStackTrace();

**throw** **new** RuntimeException(e);

}

}

Конструктор данного класса имеет следующий код:

**public** SqlDataProvider(String driver, String url, String user, String password, Class<?> entity,

Class<?>... entities) {

**this**.driver = driver;

**this**.url = url;

**this**.user = user;

**this**.password = password;

**this**.entities = **new** ArrayList<>(Arrays.*asList*(entities));

**this**.entities.add(entity);

}

*Class<?> entity, Class<?>... entities,* такая реализация вынуждает добавить хоть один класс сущности в проект. Вызвано это необходимостью передать Hibernate тот загрузчик классов, в котором есть данные сущности.

### Проектирование модели

#### Описание модели данных классами java

Чтобы использовать возможности JPA для получения данных из СУБД в виде объектов и составления типозащищенных запросов в *Querydsl* нужно описать классы сущностей в коде *java*

@Entity

@Table(name = "requestedpartslist")

@IdClass(value = PartRequest.PartRequestId.**class**)

**public** **class** PartRequest {

@Id

@ManyToOne

@JoinColumn(name = "RequestID")

**public** Order order;

@Id

@ManyToOne

@JoinColumn(name = "SparePartID")

**public** Part part;

@Column(name = "Price")

**public** **int** price;

@Temporal(TemporalType.***DATE***)

@Column(name = "DateOfDelivery")

**public** Calendar dateOfDelivery;

@Column(name = "Count")

**public** **int** count;

**public** LocalDate getDateOfDelivery() {

**return** toLocalDate(dateOfDelivery);

}

@SuppressWarnings("serial")

**public** **static** **class** PartRequestId **implements** Serializable {

**public** Order order;

**public** Part part;

}

}

*@Entity* – аннотация объявляющая, что данный класс является сущностью.

*@Table(name = "requestedpartslist")* указывает, какую таблицу представляет данная сущность.

*@IdClass(value = PartRequest.PartRequestId.class)* указывает, что ключ данной таблицы является составным и обозначает класс составного ключа.

*@Id* указывает, что данное поле является ключевым атрибутом в базе данных.

*@ManyToOne* указывает, что данный класс имеет связь с классом *Order* «многие-ко-одному».

*@JoinColumn(name = "RequestID")* указывает, по какому атрибуту в таблице *requestedpartslist* осуществлять связывание.

*@Column(name = "Price")* обозначает название атрибута в таблице, необходим, если название поля не совпадает с названием атрибута, иначе берется название поля.

*@Temporal(TemporalType.DATE)* показывает, как обрабатывать атрибут даты в базе данных. Может быть DATE, TIME или TIMESTAMP

Класс *PartRequestId* необходим, чтобы показать, что в данной таблице ключ составной

Также далее приведен фрагмент класса «Заказ» для описания некоторых аннотаций

@Entity

@Table(name = "sparepartsrequests")

**public** **class** Order {

@Id

@Column(name = "ID")

**public** **int** id;

...

@Temporal(TemporalType.***TIMESTAMP***)

@Column(name = "DateOfCreation")

**public** Calendar dateOfCreation;

...

@OneToMany(mappedBy = "order", fetch = FetchType.***LAZY***)

**public** List<PartRequest> requestedParts;

}

*@OneToMany(mappedBy = "order", fetch = FetchType.LAZY)* показывает, что данный класс имеет связь с классом *Order* «один-ко-многим». Поле *mappedBy* – это имя поля данного класса в классе *PartRequest. FetchType.LAZY* указывает, что запрашивать содержимое данного списка из базы нужно только в тот момент, когда произойдет обращение к содержимому списка в коде, данная настройка существенно снижает нагрузку на память.

#### Подключение к СУБД

**constant** url = "jdbc:mysql://mikhailmineev.ru:3306/corpterminal?zeroDateTimeBehavior=convertToNull"

**constant** username = "jpademo"

**constant** password = "5xYB2e6T5Jo7ajA"

**dataprovider** data = **new** SqlDataProvider(*driver*, *url*, *username*, *password*, Part, Order, PartRequest)

*driver* – главный класс коннектора для соответствующей СУДБ, определяется типом СУБД и подключенным коннектором

*url* – строка подключения к СУБД

*username* – имя пользователя СУБД

*password* – пароль пользователя СУБД

Чтобы не нагружать СУБД заказчика во время отладки модели, данные были экспортированы, и был поднят свой MySQL сервер, к которому в дальнейшем и осуществлялось подключение.

#### Запрос к СУБД

Запрос к СУБД для получения заказов осуществляется с помощью библиотеки ***Querydsl***. Пример кода, осуществляющего запрос к СУБД приведен ниже:

**val** query = *data*.<Order>getQuery

**val** qOrder = QOrder.*order*

**val** orderList = query.from(qOrder).fetch

В результате возвращается список заказов

*query* – класс для осуществления запроса к СУБД

*qOrder* – класс, сгенерированный ***Querydsl*** из класса сущности для создания запроса

*orderList* – результат выполнения запроса к СУБД, возвращающий список сущностей

Полный код модели представлен в приложении 2.

# ЗАКЛЮЧЕНИЕ

В рамках данного курсового проекта были получены следующие результаты:

1. Проведено предпроектное обследование процесса обработки заказов и структуры БД, в которой хранится история заказов. Построены IDEF3 и IDEF1x диаграммы
2. На этапе концептуального проектирование были выявлены ограничения возможностей RAO-X для построения модели, определены средства их устранения. С помощью UML нотации пакетов определены новые зависимости. Сформулировано техническое задание
3. На этапе структурного проектирования разработаны диаграммы активностей и классов для совершенствования функционала RAO-X. Разработана диаграмма классов соответствующая IDEF1x диаграмме БД
4. На рабочем этапе проектирования реализован программный код по разработанным диаграммам активностей и классов.
5. Разработан код модели, осуществляющий симуляцию процесса обработки заказа на основе данных о заказах из БД.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация по РДО // Документация по языку РДО в открытом доступе URL: <http://raox.ru/docs/reference/base_types_and_functions.html> (дата обращения: 07.12.2017)
2. Документация по JPA // Документация по спецификации JPA в открытом доступе URL: <https://docs.oracle.com/javaee/5/tutorial/doc/bnbpz.html> (дата обращения: 07.12.2017)
3. Документация по Querydsl // Документация по библиотеке Querydsl в открытом доступе URL: <https://www.querydsl.com/static/querydsl/4.1.3/reference/html_single/> (дата обращения: 07.12.2017)
4. Тепломеханика // Официальный сайт фирмы «Тепломеханика» URL: <http://tm-sc.ru/> (дата обращения: 21.12.2017)
5. Hibernate API // Документация по коду библиотеки Hibernate в открытом доступе URL: <http://docs.jboss.org/hibernate/orm/5.2/javadocs/> (дата обращения: 07.12.2017)
6. Querydsl API // Документация по коду библиотеки Querydsl в открытом доступе URL: [http://www.querydsl.com/static/querydsl/4.1.3/apidocs/](http://www.querydsl.com/static/querydsl/4.1.3/apidocs/%22﷟HYPERLINK%20%22https://www.querydsl.com/static/querydsl/4.1.3/reference/html_single/)﷟HYPERLINK "https://www.querydsl.com/static/querydsl/4.1.3/reference/html\_single/" (дата обращения: 07.12.2017)

# ПРИЛОЖЕНИЯ

# Приложение 1

Программный код реализации интерфейса PersistenceUnitInfo

**package** ru.bmstu.rk9.rao.lib.persistence;

**import** java.net.URL;

**import** java.util.Collections;

**import** java.util.List;

**import** java.util.Properties;

**import** javax.persistence.SharedCacheMode;

**import** javax.persistence.ValidationMode;

**import** javax.persistence.spi.ClassTransformer;

**import** javax.persistence.spi.PersistenceUnitInfo;

**import** javax.persistence.spi.PersistenceUnitTransactionType;

**import** javax.sql.DataSource;

**public** **class** PersistenceUnitInfoImpl **implements** PersistenceUnitInfo {

**private** **static** **final** String ***PERSISTENCE\_PROVIDER*** = "org.hibernate.jpa.HibernatePersistenceProvider";

**private** **final** String persistenceUnitName;

**private** **final** ClassLoader classLoader;

**public** PersistenceUnitInfoImpl(String persistenceUnitName, ClassLoader classLoader) {

**this**.persistenceUnitName = persistenceUnitName;

**this**.classLoader = classLoader;

}

@Override

**public** String getPersistenceUnitName() {

**return** persistenceUnitName;

}

@Override

**public** String getPersistenceProviderClassName() {

**return** ***PERSISTENCE\_PROVIDER***;

}

@Override

**public** PersistenceUnitTransactionType getTransactionType() {

**return** PersistenceUnitTransactionType.***RESOURCE\_LOCAL***;

}

@Override

**public** DataSource getJtaDataSource() {

**return** **null**;

}

@Override

**public** DataSource getNonJtaDataSource() {

**return** **null**;

}

@Override

**public** List<String> getMappingFileNames() {

**return** Collections.*emptyList*();

}

@Override

**public** List<URL> getJarFileUrls() {

**return** Collections.*emptyList*();

}

@Override

**public** URL getPersistenceUnitRootUrl() {

**return** **null**;

}

@Override

**public** List<String> getManagedClassNames() {

**return** Collections.*emptyList*();

}

@Override

**public** **boolean** excludeUnlistedClasses() {

**return** **true**;

}

@Override

**public** SharedCacheMode getSharedCacheMode() {

**return** SharedCacheMode.***UNSPECIFIED***;

}

@Override

**public** ValidationMode getValidationMode() {

**return** ValidationMode.***AUTO***;

}

@Override

**public** Properties getProperties() {

**return** **new** Properties();

}

@Override

**public** String getPersistenceXMLSchemaVersion() {

**throw** **new** UnsupportedOperationException();

}

@Override

**public** ClassLoader getClassLoader() {

**return** classLoader;

}

@Override

**public** **void** addTransformer(ClassTransformer transformer) {

**throw** **new** UnsupportedOperationException();

}

@Override

**public** ClassLoader getNewTempClassLoader() {

**throw** **new** UnsupportedOperationException();

}

}

# Приложение 2

Программый код модели обработки заказов

**import** domain.Order

**import** domain.Part

**import** domain.PartRequest

**import** domain.QOrder

**import** ru.bmstu.rk9.rao.lib.persistence.SqlDataProvider

**import** java.util.ArrayList

**import** java.util.List

**import** java.util.Map

**import** java.util.HashMap

**import** java.time.temporal.ChronoUnit

**constant** driver = "com.mysql.jdbc.Driver"

**constant** url = "jdbc:mysql://mikhailmineev.ru:3306/corpterminal?zeroDateTimeBehavior=convertToNull"

**constant** username = "jpademo"

**constant** password = "5xYB2e6T5Jo7ajA"

**dataprovider** data = **new** SqlDataProvider(*driver*, *url*, *username*, *password*, Part, Order, PartRequest)

**type** OrderStats {

Map<String, Part> localPartData

**long** fails

**long** successes

List<Double> processDuration

List<Double> failPrices

List<Double> successfulPrices

}

**resource** orderStats = OrderStats.*create*(**new** HashMap, 0, 0, **new** ArrayList, **new** ArrayList, **new** ArrayList)

**long** getModificationInterval(Order order) {

**val** start = order.getDateOfCreation();

**val** end = order.getDateOfProcessing();

**val** interval = start.until(end, ChronoUnit.*DAYS*);

**if** (interval < 0)

**throw** **new** IllegalStateException("getModificationInterval " + interval);

**return** interval;

}

**long** getRealizationInterval(Order order) {

**val** start = order.getDateOfCreation();

**val** end = order.getDateOfRealization();

**val** interval = start.until(end, ChronoUnit.*DAYS*);

**if** (interval < 0)

**throw** **new** IllegalStateException("getRealizationInterval " + interval);

**return** interval;

}

**long** getDeliveryInterval(PartRequest partRequest) {

**val** start = partRequest.order.getDateOfCreation();

**val** end = partRequest.getDateOfDelivery();

**val** interval = start.until(end, ChronoUnit.*DAYS*);

**return** interval;

}

Part tryFetchLocal(Part part){

**if** (!*orderStats*.localPartData.containsKey(part.id))

*orderStats*.localPartData.put(part.id, part)

**return** *orderStats*.localPartData.get(part.id)

}

Double calculatePrice(Order order){

**var** sum = 0.0

**for** (partRequest : order.requestedParts){

**val** part = partRequest.part

sum += part.purchaseprice \* *priceMultiplier*

}

**return** sum

}

**boolean** takeParts(PartRequest request) {

**val** part = *tryFetchLocal*(request.part)

**if** (part.stocked >= request.count) {

part.stocked -= request.count;

**return** **true**;

} **else** {

part.stocked = 0;

**return** **false**;

}

}

**constant** allowedDeliveryWaitPeriodDays = 3

**constant** priceMultiplier = 1.2

**enum** PartState {REQUESTED, IN\_TRANSIT, ARRIVED}

**enum** OrderResult {OK, FAIL, LATE, EMPTY}

**enum** OrderState {REQUESTED, PROCESSING, FINISHED}

**type** OrderType {

Order original

OrderState state

OrderResult resulted

**double** creationTime

**double** emptyProcessDuration

List<PartRequestType> requests // One to many

}

**type** PartRequestType {

PartRequest original

PartState state

**double** creationTime

OrderType order // Many to one

}

**int** waitPeriod() {

**return** delivery\_to\_order.order\_request\_model.*allowedDeliveryWaitPeriodDays*

}

**event** OrderReceived(Order order) {

**if** (order.requestedParts.isEmpty && order.dateOfRealization === **null** && order.dateOfProcessing === **null**) {

**return**

}

OrderType.*create*(order, OrderState.*REQUESTED*, OrderResult.*OK*, *currentTime*, 0, **new** ArrayList())

*log*("Created order " + order.id + "(time:" + *currentTime* + ")")

}

**rule** OrderProcessing() {

**relevant** order = OrderType.*accessible*.*filter*[state == OrderState.*REQUESTED*].*any*

**def** execute() {

**val** parts = order.original.requestedParts

**if** (parts.isEmpty) {

order.state = OrderState.*FINISHED*

order.resulted = OrderResult.*EMPTY*

*log*("Processed order (no parts) " + order.getNumber)

**return**

}

**for** (partRequest : parts) {

**val** partRequestType = PartRequestType.*create*(partRequest, PartState.*REQUESTED*, *currentTime*, order)

order.requests.add(partRequestType)

}

order.state = OrderState.*PROCESSING*

*log*("Processed order (created " + parts.size() + " parts) " + order.getNumber)

}

}

**rule** UtilizeOrder() {

**relevant** order = OrderType.*accessible*.*filter*[state == OrderState.*FINISHED*].*any*

**def** execute() {

order.erase()

**if** (*currentTime* - order.creationTime > *waitPeriod*)

order.resulted = OrderResult.*LATE*

**var** duration = *currentTime* - order.creationTime

**switch** (order.resulted) {

**case** *LATE*: {

*orderStats*.fails = *orderStats*.fails + 1

}

**case** *FAIL*: {

*orderStats*.fails = *orderStats*.fails + 1

}

**case** *OK*: {

*orderStats*.successes = *orderStats*.successes + 1

}

**case** *EMPTY*: {

*orderStats*.fails = *orderStats*.fails + 1

**if** (order.original.dateOfRealization !== **null**) {

duration = *getRealizationInterval*(order.original)

}

duration = *getModificationInterval*(order.original)

}

}

*orderStats*.processDuration.add(duration)

**if**(order.resulted == OrderResult.*OK*)

*orderStats*.successfulPrices.add(*calculatePrice*(order.original))

**else**

*orderStats*.failPrices.add(*calculatePrice*(order.original))

*log*("Utilized order " + order.original.id + "\tResult:" + order.resulted + "\tDuration:" + duration + "\t")

}

}

**operation** PartProcessing() {

**relevant** partRequest = PartRequestType.*accessible*.*filter*[state == PartState.*REQUESTED*].*any*

**def** begin() {

partRequest.state = PartState.*IN\_TRANSIT*

}

**def** duration() {

**val** order = partRequest.original.order

**val** part = partRequest.original.part

**if** (*takeParts*(partRequest.original)) {

*log*("Деталь " + part.name + " взята со склада");

**return** 0

} **else** **if** (partRequest.original.dateOfDelivery !== **null** && *getDeliveryInterval*(partRequest.original) > 0) {

*log*("Деталь " + part.name + " будет доставлена через " + *getDeliveryInterval*(partRequest.original) +

" дней");

**return** *getDeliveryInterval*(partRequest.original)

} **else** **if** (partRequest.original.dateOfDelivery !== **null**) {

*log*("Деталь " + part.name + " имеет <0 длительность, считаем, что на складе");

**return** 0

} **else** **if** (order.dateOfRealization !== **null**) {

*log*("Деталь " + part.name + " будет доставлена через " + *getRealizationInterval*(order) +

" дней (по дате реализации заказа)");

**return** *getRealizationInterval*(order)

}

*log*("Деталь " + part.name + " будет доставлена через " + *getModificationInterval*(order) +

" дней (по дате изменения заказа)");

**return** *getModificationInterval*(order)

}

**def** end() {

partRequest.state = PartState.*ARRIVED*

}

}

**rule** UtilizePartRequest() {

**relevant** partRequest = PartRequestType.*accessible*.*filter*[state == PartState.*ARRIVED*].*any*

**def** execute() {

**val** orderType = partRequest.order

orderType.requests.remove(partRequest)

partRequest.erase()

**if** (orderType.requests.isEmpty())

orderType.state = OrderState.*FINISHED*

}

}

**logic** Model {

**activity** orderProcessing = **new** Activity(OrderProcessing.*create*())

**activity** utilizeOrder = **new** Activity(UtilizeOrder.*create*())

**activity** partProcessing = **new** Activity(PartProcessing.*create*())

**activity** utilizePartRequest = **new** Activity(UtilizePartRequest.*create*())

}

**def** init() {

**val** query = *data*.<Order>getQuery

**val** qOrder = QOrder.*order*

**val** orderList = query.from(qOrder).fetch

**for** (order : orderList) {

OrderReceived.*plan*(order.getEpochDayOfCreation(), order)

}

*log*("Orders total " + orderList.size)

}

**def** finish() {

**val** percentage = *orderStats*.successes \* 100 / (*orderStats*.successes + *orderStats*.fails)

*log*("Result:\tOrders " + *orderStats*.successes + ":" + *orderStats*.fails + " Successful percentage: " + percentage + "%")

**val** durationResults = *orderStats*.processDuration.stream.mapToDouble([f | f]).summaryStatistics

**val** avg = durationResults.getAverage

**val** max = durationResults.getMax

**val** min = durationResults.getMin

*log*("Duration:\tAverage " + avg + "\tMax:" + max + "\tMin:" + min)

**val** failPriceResults = *orderStats*.failPrices.stream.mapToDouble([f | f]).summaryStatistics

**val** avgFailPrice = failPriceResults.getAverage

**val** successPriceResults = *orderStats*.successfulPrices.stream.mapToDouble([f | f]).summaryStatistics

**val** avgSuccessPrice = successPriceResults.getAverage

*log*("Average cost:\tsuccessful:" + avgSuccessPrice + "\tfail:" + avgFailPrice)

}

**result** fails = Result.*create*([*orderStats*.fails])

**result** successes = Result.*create*([*orderStats*.successes])